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 [6]:

```
%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 [7]:

```
# 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 1000
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 (100000, 10)
```

Out[7]:

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

In [8]:

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

In [9]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[9]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [10]:

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

Out[10]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	

In [11]:

```
display['COUNT(*)'].sum()
```

Out[11]:

```
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 [12]:

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

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessD
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	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 [13]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
e, kind='quicksort', na_position='last')
```

In [14]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
'first', inplace=False)
final.shape
```

Out[14]:

```
(87775, 10)
```

In [15]:

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

Out[15]:

```
87.775
```

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 [16]:

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

display.head()
```

Out[16]:

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

In [17]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [18]:

```
#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()
```

```
(87773, 10)
```

Out[18]:

```
1    73592
0    14181
Name: Score, dtype: int64
```

In [19]:

```
final=final.sort_values('Time')
```

In [20]:

```
61441*0.7
```

Out[20]:

```
43008.7
```

In [21]:

```
x=final.drop('Score',axis=1)
y=final.filter(['Score'],axis=1)
```

In [22]:

```
x_tr=x[0:43008]
y_tr=y[0:43008]

x_cv=x[43008:61441]
y_cv=y[43008:61441]

x_test=x[61441:]
y_test=y[61441:]
```

[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 [23]:

```
# 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 bought a few of these after my apartment was infested with fruit flies.
After only a few hours, the trap had &quot;attracted&quot; many flies and
within a few days they were practically gone. This may not be a long term
solution, but if flies are driving you crazy, consider buying this. One  c
aution- the surface is very sticky, so try to avoid touching it.
==================================================
I have made these brownies for family and for a den of cub scouts and no o
ne would have known they were gluten free and everyone asked for seconds!
These brownies have a fudgy texture and have bits of chocolate chips in th
em which are delicious.  I would say the mix is very thick and a little di
fficult to work with.  The cooked brownies are slightly difficult to cut i
nto very neat edges as the edges tend to crumble a little and I would also
say that they make a slightly thinner layer of brownies than most of the s
tore brand gluten containing but they taste just as good, if not better.
Highly recommended!<br /><br />(For those wondering, this mix requires 2 e
ggs OR 4 egg whites and 7 tbs melted butter to prepare.  They do have sugg
estions for lactose free and low fat preparations)
==================================================
This gum is my absolute favorite. By purchasing on amazon I can get the sa
vings of large quanities at a very good price. I highly recommend to all g
um chewers. Plus as you enjoy the peppermint flavor and freshing of breath
you are whitening your teeth all at the same time.
==================================================
This is an excellent product, both tastey and priced right. It's difficult
to find this product in regular local grocery stores, so I was thrilled to
find it.
==================================================
```

In [24]:

```
# 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 bought a few of these after my apartment was infested with fruit flies.
After only a few hours, the trap had &quot;attracted&quot; many flies and
within a few days they were practically gone. This may not be a long term
solution, but if flies are driving you crazy, consider buying this. One  c
aution- the surface is very sticky, so try to avoid touching it.
```

In [25]:

```
# 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 bought a few of these after my apartment was infested with fruit flies.
After only a few hours, the trap had "attracted" many flies and within a f
ew days they were practically gone. This may not be a long term  solution,
but if flies are driving you crazy, consider buying this. One  caution- th
e surface is very sticky, so try to avoid touching it.
==================================================
I have made these brownies for family and for a den of cub scouts and no o
ne would have known they were gluten free and everyone asked for seconds!
These brownies have a fudgy texture and have bits of chocolate chips in th
em which are delicious.  I would say the mix is very thick and a little di
fficult to work with.  The cooked brownies are slightly difficult to cut i
nto very neat edges as the edges tend to crumble a little and I would also
say that they make a slightly thinner layer of brownies than most of the s
tore brand gluten containing but they taste just as good, if not better.
Highly recommended!(For those wondering, this mix requires 2 eggs OR 4 egg
whites and 7 tbs melted butter to prepare.  They do have suggestions for l
actose free and low fat preparations)
==================================================
This gum is my absolute favorite. By purchasing on amazon I can get the sa
vings of large quanities at a very good price. I highly recommend to all g
um chewers. Plus as you enjoy the peppermint flavor and freshing of breath
you are whitening your teeth all at the same time.
==================================================
This is an excellent product, both tastey and priced right. It's difficult
to find this product in regular local grocery stores, so I was thrilled to
find it.
```

In [26]:

```
# 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 [27]:

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

```
This gum is my absolute favorite. By purchasing on amazon I can get the sa
vings of large quanities at a very good price. I highly recommend to all g
um chewers. Plus as you enjoy the peppermint flavor and freshing of breath
you are whitening your teeth all at the same time.
==================================================
```

In [28]:

```
#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 bought a few of these after my apartment was infested with fruit flies.
After only a few hours, the trap had &amp;quot;attracted&amp;quot; many flies and
within a few days they were practically gone. This may not be a long term
solution, but if flies are driving you crazy, consider buying this. One  c
aution- the surface is very sticky, so try to avoid touching it.
```

In [29]:

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

```
This gum is my absolute favorite By purchasing on amazon I can get the sav
ings of large quanities at a very good price I highly recommend to all gum
chewers Plus as you enjoy the peppermint flavor and freshing of breath you
are whitening your teeth all at the same time
```

In [30]:

```
# 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', 'ourselve
s', '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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', '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', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [31]:

```
# Combining all the above stundents
def prepro(x):
    # Combining all the above stundents
    from tqdm import tqdm
    preprocessed_reviews = []
    # tqdm is for printing the status bar
    for sentance in tqdm(x['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 sto
pwords)
        preprocessed_reviews.append(sentance.strip())
    return preprocessed_reviews
```

In [32]:

```
prepro_tr=prepro(x_tr)
```

```
100%|██████████| 43008/43008 [00:14<00:00, 2884.06it/s]
```

In [33]:

```
prepro_cv=prepro(x_cv)
```

```
100%|██████████| 18433/18433 [00:06<00:00, 2932.43it/s]
```

In [34]:

```
prepro_test=prepro(x_test)
```

```
100%|██████████| 26332/26332 [00:08<00:00, 2950.09it/s]
```

[4] Featurization

[4.1] BAG OF WORDS

FOR TRAIN DATA

In [35]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(prepro_tr)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

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

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaaaaaaaaaaa', 'aaaaaaarrrrr
ggghhh', 'aaaaah', 'aaaah', 'aaaand', 'aachen', 'aadp']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (43008, 38756)
the number of unique words  38756
```

FOR CV DATA

In [36]:

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (18433, 38756)
the number of unique words  38756
```

FOR TEST DATA

In [37]:

```
#BoW

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (26332, 38756)
the number of unique words  38756
```

In [38]:

```
len(count_vect.get_feature_names())
```

Out[38]:

```
38756
```

[4.2] Bi-Grams and n-Grams.

In [39]:

```
#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/modul
es/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)
final_bigram_counts = count_vect.fit_transform(prepro_tr)
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_c
ounts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (43008, 25454)
the number of unique words including both unigrams and bigrams  25454
```

[4.3] TF-IDF

In [40]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)

final_tf_idf = tf_idf_vect.fit_transform(prepro_tr)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.g
et_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (43008, 25454)
the number of unique words including both unigrams and bigrams  25454
```

In [41]:

```
final_tf_idf_cv = tf_idf_vect.transform(prepro_cv)
print("the type of count vectorizer ",type(final_tf_idf_cv))
print("the shape of out text TFIDF vectorizer ",final_tf_idf_cv.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf_c
v.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (18433, 25454)
the number of unique words including both unigrams and bigrams  25454
```

In [42]:

```
final_tf_idf_test = tf_idf_vect.transform(prepro_test)
print("the type of count vectorizer ",type(final_tf_idf_test))
print("the shape of out text TFIDF vectorizer ",final_tf_idf_test.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf_t
est.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (26332, 25454)
the number of unique words including both unigrams and bigrams  25454
```

[4.4] Word2Vec

w2v for TRAIN DATA

In [43]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in prepro_tr:
    list_of_sentance.append(sentance.split())
```

In [44]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    print('hiiii')
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=-1)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bi
n', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to
train your own w2v ")
```

```
hiiii
[('disappointment', 0.5205342173576355), ('translates', 0.483817487955093
4), ('pan', 0.4747382402420044), ('years', 0.4738738536834717), ('plug',
0.4585842490196228), ('oiled', 0.4414167106151581), ('horrific', 0.4395951
03263855), ('englishman', 0.4221752882003784), ('degree', 0.42215302586555
48), ('devours', 0.42191076278686523)]
==================================================
[('accidently', 0.5178179740905762), ('alba', 0.5032861232757568), ('da',
0.485734760761261), ('factory', 0.4648679196834564), ('initial', 0.4629889
130592346), ('attitude', 0.4545481503009796), ('horrid', 0.446190774440765
4), ('milled', 0.44090691208839417), ('nitrates', 0.4291522204875946), ('o
verfeed', 0.42856329679489136)]
```

In [45]:

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

```
number of words that occured minimum 5 times  12483
sample words  ['bought', 'apartment', 'infested', 'fruit', 'flies', 'hour
s', 'trap', 'attracted', 'many', 'within', 'days', 'practically', 'gone',
'may', 'not', 'long', 'term', 'solution', 'driving', 'crazy', 'consider',
'buying', 'one', 'caution', 'surface', 'sticky', 'try', 'avoid', 'touchin
g', 'really', 'good', 'idea', 'final', 'product', 'outstanding', 'use', 'c
ar', 'window', 'everybody', 'asks', 'made', 'two', 'thumbs', 'received',
'shipment', 'could', 'hardly', 'wait', 'love', 'call']
```

w2v FOR CV DATA

In [46]:

```
i=0
list_of_sentance_cv=[]
for sentance in prepro_cv:
    list_of_sentance_cv.append(sentance.split())
```

In [47]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model_cv=Word2Vec(list_of_sentance_cv,min_count=5,size=50, workers=-1)
    print(w2v_model_cv.wv.most_similar('great'))
    print('='*50)
    print(w2v_model_cv.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bi
n', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to
train your own w2v ")
```

```
[('disappointment', 0.5205342173576355), ('pan', 0.4747382402420044), ('ye
ars', 0.4738738536834717), ('plug', 0.4585842490196228), ('breadcrumbs',
0.45417919754981995), ('degree', 0.4221530258655548), ('devours', 0.421910
76278686523), ('usable', 0.42063838243484497), ('explode', 0.4159730970859
5276), ('souchong', 0.40671810507774353)]
==================================================
[('alba', 0.5032861232757568), ('da', 0.485734760761261), ('factory', 0.46
48679196834564), ('initial', 0.4629889130592346), ('attitude', 0.454548150
3009796), ('horrid', 0.4461907744407654), ('detox', 0.42774492502212524),
('stroganoff', 0.4259563684463501), ('departure', 0.4248652160167694), ('u
nits', 0.4221905767917633)]
```

In [48]:

```
w2v_words_cv = list(w2v_model_cv.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_cv))
print("sample words ", w2v_words_cv[0:50])
```

```
number of words that occured minimum 5 times  8193
sample words  ['ordered', 'tea', 'last', 'year', 'liked', 'much', 'like',
'try', 'friends', 'see', 'taste', 'not', 'normal', 'expected', 'drink', 's
outh', 'say', 'smooth', 'tasting', 'mine', 'unsweetened', 'sweetened', 'st
ill', 'coming', 'back', 'asking', 'kind', 'works', 'least', 'ca', 'beat',
'price', 'either', 'large', 'shepherd', 'beagle', 'spent', 'long', 'time',
'finishing', 'treat', 'big', 'dog', 'handle', 'anything', 'little', 'soli
d', 'gas', 'recommend', 'better']
```

w2v FOR TEST DATA

In [49]:

```
i=0
list_of_sentance_test=[]
for sentance in prepro_test:
    list_of_sentance_test.append(sentance.split())
```

In [50]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model_test=Word2Vec(list_of_sentance_test,min_count=5,size=50, workers=-1)
    print(w2v_model_test.wv.most_similar('great'))
    print('='*50)
    print(w2v_model_test.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bi
n', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to
train your own w2v ")
```

```
[('disappointment', 0.5205342173576355), ('saucebut', 0.4914477169513702
4), ('translates', 0.4838174879550934), ('pan', 0.4747382402420044), ('yea
rs', 0.4738738536834717), ('plug', 0.4585842490196228), ('horrific', 0.439
595103263855), ('degree', 0.4221530258655548), ('devours', 0.4219107627868
6523), ('usable', 0.42063838243484497)]
==================================================
[('da', 0.485734760761261), ('factory', 0.4648679196834564), ('initial',
0.4629889130592346), ('attitude', 0.4545481503009796), ('horrid', 0.446190
7744407654), ('milled', 0.44090691208839417), ('marrow', 0.433804392814636
23), ('nitrates', 0.4291522204875946), ('detox', 0.42774492502212524), ('s
troganoff', 0.4259563684463501)]
```

In [51]:

```
w2v_words_test = list(w2v_model_test.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_test))
print("sample words ", w2v_words_test[0:50])
```

```
number of words that occured minimum 5 times  9855
sample words  ['revolution', 'teas', 'superb', 'smooth', 'delicate', 'on
e', 'pleasant', 'comforting', 'pure', 'delight', 'tried', 'pomegranate',
'well', 'ginger', 'peach', 'dragon', 'eye', 'oolong', 'white', 'pear', 'wa
nt', 'drink', 'like', 'asian', 'grandma', 'go', 'ahead', 'follow', 'direct
ion', 'packet', 'however', 'non', 'alcoholic', 'cocktails', 'put', 'two',
'cup', 'add', 'hot', 'water', 'stir', 'pour', 'shaker', 'tablespoons', 'si
mple', 'syrup', 'fill', 'rest', 'ice', 'cubes']
```

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

[4.4.1.1] Avg W2v

In [52]:

```
def avgw2v(list_of_sentance,w2v_words,w2v_model):
    # average Word2Vec
    # compute average word2vec for each review.
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(list_of_sentance): # 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.append(sent_vec)
    print(len(sent_vectors))
    print(len(sent_vectors[0]))
    return sent_vectors
```

In [288]:

```
sent_vectors_tr=avgw2v(list_of_sentance,w2v_words,w2v_model)
```

```
100%|
| 43008/43008 [01:23<00:00, 513.81it/s]

43008
50
```

In [289]:

```
sent_vectors_cv=avgw2v(list_of_sentance_cv,w2v_words_cv,w2v_model_cv)
```

```
100%|██████████| 18433/18433 [00:19<00:00, 934.70it/s]

18433
50
```

In [290]:

```
sent_vectors_test=avgw2v(list_of_sentance_test,w2v_words_test,w2v_model_test)
```

```
100%|██████████| 26332/26332 [00:30<00:00, 860.01it/s]

26332
50
```

[4.4.1.2] TFIDF weighted W2v

In [56]:

```
def tfidfwei_w2v(preprocessed_reviews,list_of_sentance,w2v_words,w2v_model):
    # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
    model = TfidfVectorizer()
    tf_idf_matrix = model.fit_transform(preprocessed_reviews)
    # we are converting a dictionary with word as a key, and the idf as a value
    dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
    # TF-IDF weighted Word2Vec
    tfidf_feat = model.get_feature_names() # tfidf words/col-names
    # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfi
df



    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this
list
    row=0;
    for sent in tqdm(list_of_sentance): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words and word in tfidf_feat:
                vec = w2v_model.wv[word]
                #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                # to reduce the computation we are
                # dictionary[word] = idf value of word in whole courpus
                # sent.count(word) = tf valeus of word in this review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors.append(sent_vec)
        row += 1
    return tfidf_sent_vectors,model
```

In [57]:

```
def tfidfwei_w2v1(model,preprocessed_reviews,list_of_sentance,w2v_words,w2v_model):
    # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
    tf_idf_matrix = model.transform(preprocessed_reviews)
    # we are converting a dictionary with word as a key, and the idf as a value
    dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
    # TF-IDF weighted Word2Vec
    tfidf_feat = model.get_feature_names() # tfidf words/col-names
    # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfi
df



    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this
list
    row=0;
    for sent in tqdm(list_of_sentance): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words and word in tfidf_feat:
                vec = w2v_model.wv[word]
                #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                # to reduce the computation we are
                # dictionary[word] = idf value of word in whole courpus
                # sent.count(word) = tf valeus of word in this review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors.append(sent_vec)
        row += 1
    return tfidf_sent_vectors
```

In [58]:

```
tfidf_sent_vectors,model=tfidfwei_w2v(prepro_tr,list_of_sentance,w2v_words,w2v_model)
```

```
100%|
       | 43008/43008 [10:57<00:00, 65.40it/s]
```

In [59]:

```
tfidf_sent_vectors_cv=tfidfwei_w2v1(model,prepro_cv,list_of_sentance_cv,w2v_words_cv,w2
v_model_cv)
```

```
100%|
       | 18433/18433 [04:59<00:00, 50.31it/s]
```

In [60]:

```
tfidf_sent_vectors_test=tfidfwei_w2v1(model,prepro_test,list_of_sentance_test,w2v_words
_test,w2v_model_test)
```

```
100%|
       | 26332/26332 [07:57<00:00, 55.18it/s]
```

[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**

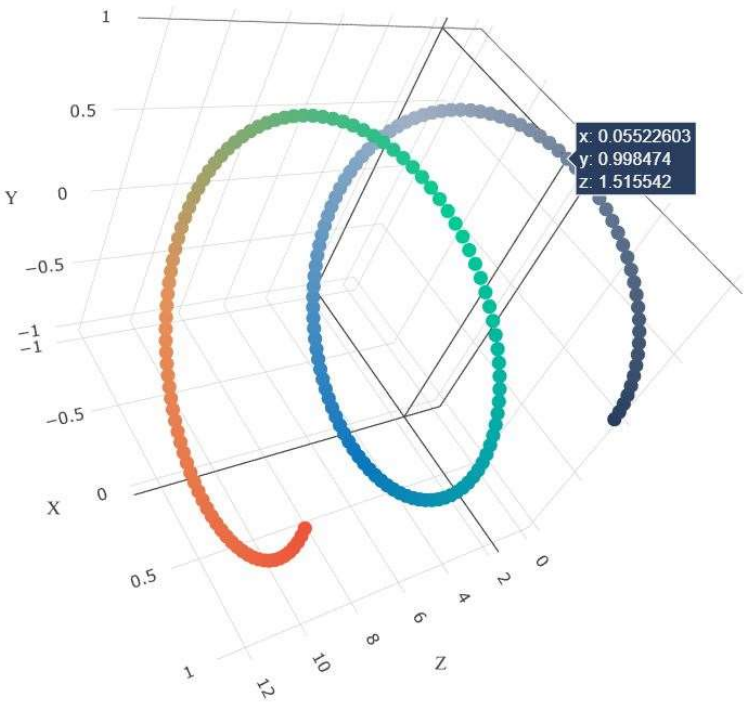
   - Get top 20 important features and represent them in a word cloud. Do this for BOW & TFIDF.

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

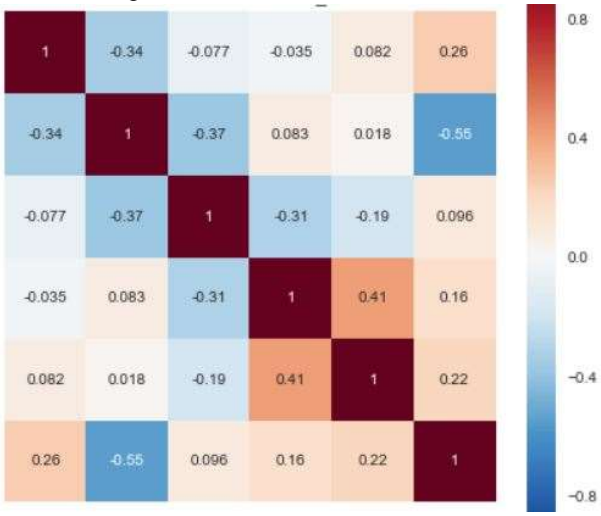


   with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive

3d_scatter_plot.ipynb

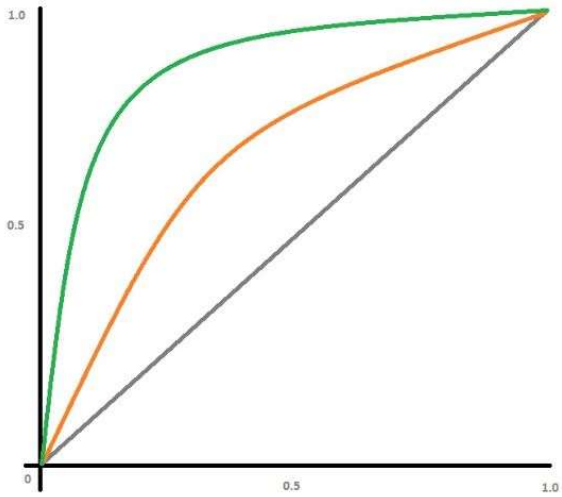
(or)

- 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.

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

(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/)

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

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

[5.1.1] Applying Random Forests on BOW, SET 1

In [87]:

```
from sklearn.ensemble import RandomForestClassifier as rfc
import seaborn as sns
```

In [68]:

```
auc_cv=[]
tpr_cv=[]
fpr_cv=[]
tpr_tr=[]
fpr_tr=[]
auc_tr=[]
data_rec=[]
data_rec=[]
hypermeter=[]
n_est = [5, 10, 50, 100, 200, 500, 1000]
max_dep = [2, 3, 5, 7,10, 12, 15, 20]
for i in n_est:
    for j in max_dep:
        clf=rfc(n_estimators=i,max_depth=j)
        clf.fit(final_counts,y_tr)
        pred = clf.predict_proba(final_counts_cv)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, pred)
        auc=metrics.auc(fpr, tpr)
        auc_cv.append(auc)
        tpr_cv.append(tpr)
        fpr_cv.append(fpr)

        # predict the response on the train
        pred = clf.predict_proba(final_counts)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
        auc=metrics.auc(fpr, tpr)
        auc_tr.append(auc)
        tpr_tr.append(tpr)
        fpr_tr.append(fpr)
        hypermeter.append([i,j])
        print(i,j)
```

```
5 2
5 3
5 5
5 7
5 10
5 12
5 15
5 20
10 2
10 3
10 5
10 7
10 10
10 12
10 15
10 20
50 2
50 3
50 5
50 7
50 10
50 12
50 15
50 20
100 2
100 3
100 5
100 7
100 10
100 12
100 15
100 20
200 2
200 3
200 5
200 7
200 10
200 12
200 15
200 20
500 2
500 3
500 5
500 7
500 10
500 12
500 15
500 20
1000 2
1000 3
1000 5
1000 7
1000 10
1000 12
1000 15
1000 20
```

In [69]:

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

In [83]:

```
dep=[]
n_est=[]
for i in range(len(hypermeter)):
    n_est.append(hypermeter[i][0])
    dep.append(hypermeter[i][1])
```

In [84]:

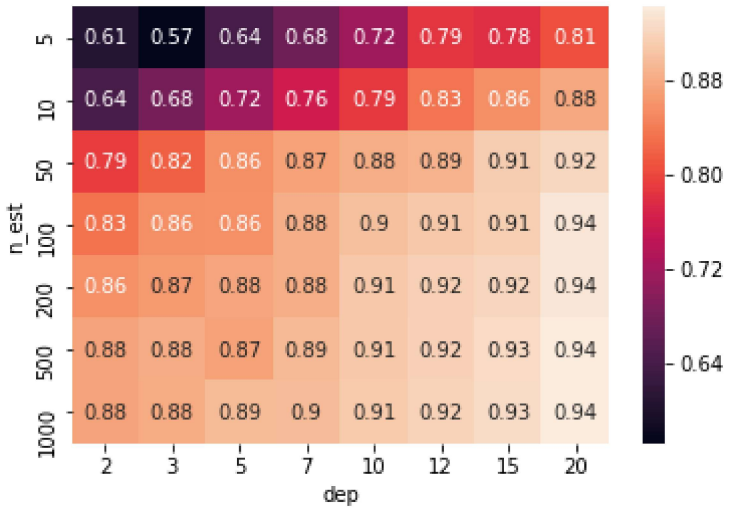
```
data1={'n_est':n_est,'dep':dep,'auc_cv':auc_cv[:],'auc_tr':auc_tr[:]}
data_f=pd.DataFrame(data1)
```

N_ESTIMATORS VS MAX_DEPTH

FOR TRAIN

In [97]:

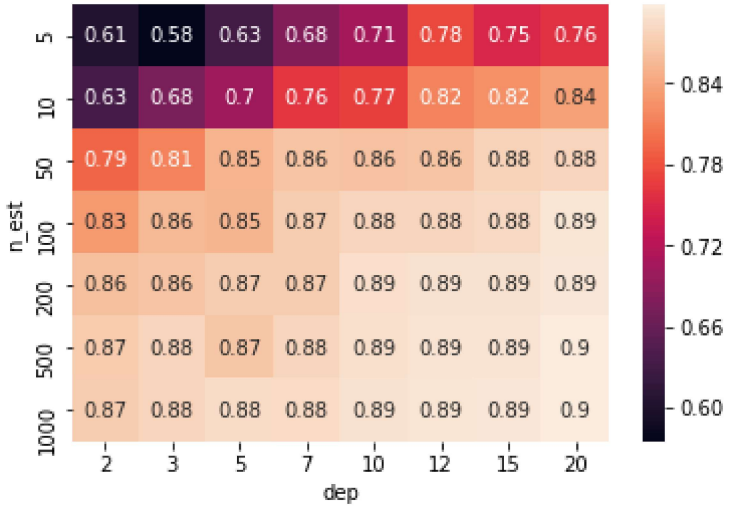
```
plo = data_f.pivot("n_est", "dep", "auc_tr");ax = sns.heatmap(plo, annot=True)
```



FOR CV

In [96]:

```
plo = data_f.pivot("n_est", "dep", "auc_cv");ax = sns.heatmap(plo, annot=True)
```



In [160]:

```
# TESTING THE MODEL ON TEST DATA

clf_final = rfc(n_estimators=10,max_depth=20)
clf_final.fit(final_counts, y_tr)
pred_final = clf_final.predict_proba(final_counts_test)[:,1]
fpr_f, tpr_f, thresholds_f = metrics.roc_curve(y_test, pred_final)
auc_final = metrics.auc(fpr_f, tpr_f)
auc_final
```

Out[160]:

```
0.832116079349039
```

In [161]:

```
# TESTING THE MODEL ON TRAIN DATA

pred_train=clf_final.predict_proba(final_counts)[:,1]
fpr_ft, tpr_ft, thresholds_ft=metrics.roc_curve(y_tr, pred_train)
auc_finalt=metrics.auc(fpr_ft, tpr_ft)
auc_finalt
```

Out[161]:

```
0.8772478852063753
```

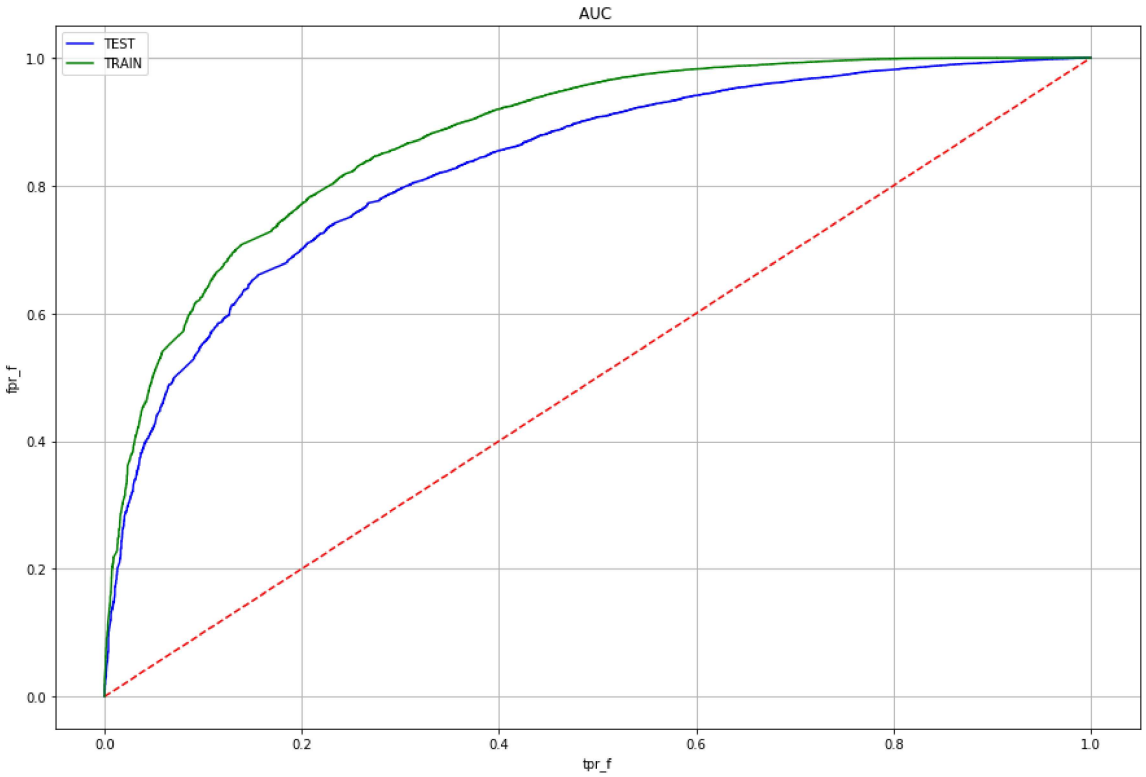
ROC CURVE

In [162]:

```
y_a=[0,0.5,1]
x_a=y_a
```

In [163]:

```
plt.figure(figsize=(15,10))
plt.plot(fpr_f,tpr_f,'b',label='TEST')
plt.plot(x_a,y_a,'--r')
plt.plot(fpr_ft,tpr_ft,'g',label='TRAIN')
plt.xlabel('tpr_f')
plt.ylabel('fpr_f')
plt.title('AUC ')
plt.grid()
plt.legend()
plt.show()
```



CONFUSION MATRIX

In [164]:

```
pred_final
pred_train
pred_final_classte=[]
pred_final_classtr=[]
for i in range(len(pred_final)):
    if pred_final[i]>0.5:
        pred_final_classte.append(1)
    elif pred_final[i]<=0.5:
            pred_final_classte.append(0)
for i in range(len(pred_train)):
    if pred_train[i]>0.5:
        pred_final_classtr.append(1)
    elif pred_train[i]<=0.5:
            pred_final_classtr.append(0)
```

TEST

In [165]:

```
cm = confusion_matrix(y_test, pred_final_classte)
cm
```

Out[165]:

```
array([[   23,  4534],
       [    2, 21773]], dtype=int64)
```

In [166]:

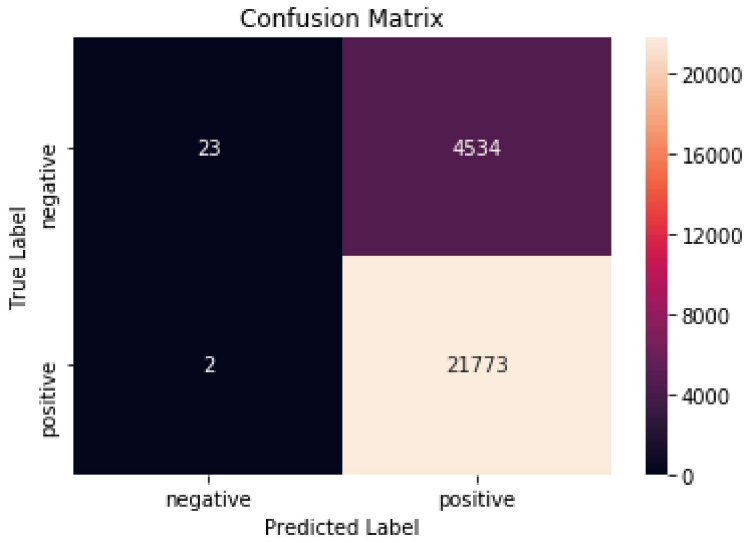
```
y_test["Score"].value_counts()
```

Out[166]:

```
1    21775
0     4557
Name: Score, dtype: int64
```

In [167]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



TRAIN

In [168]:

```
cm = confusion_matrix(y_tr, pred_final_classtr)
cm
```

Out[168]:

```
array([[  145,  6214],
       [    1, 36648]], dtype=int64)
```

In [169]:

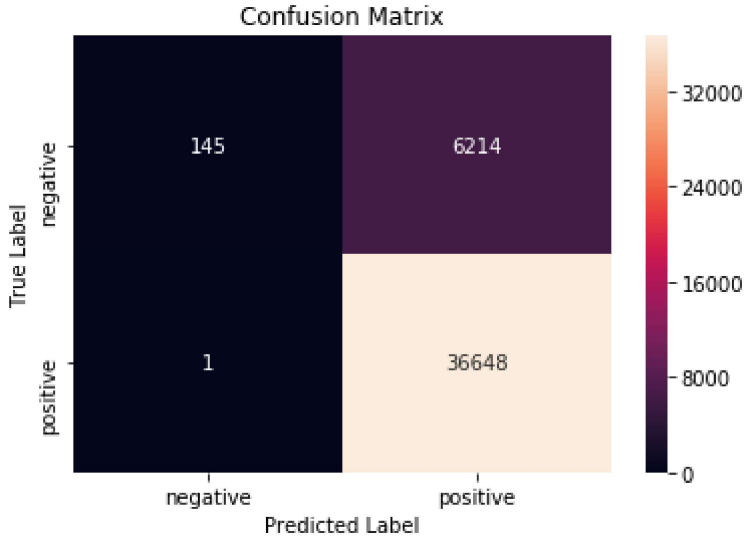
```
y_tr["Score"].value_counts()
```

Out[169]:

```
1    36649
0     6359
Name: Score, dtype: int64
```

In [170]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



CONCLUSION

In [171]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_final_classte))
```

```
              precision    recall  f1-score   support

           0       0.92      0.01      0.01      4557
           1       0.83      1.00      0.91     21775

   micro avg       0.83      0.83      0.83     26332
   macro avg       0.87      0.50      0.46     26332
weighted avg       0.84      0.83      0.75     26332
```

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

TOP 20 FEATURES

In [188]:

```
imp=clf_final.feature_importances_
```

In [189]:

```
imp1=np.argsort(imp)
```

In [211]:

```
features_name=count_vect.get_feature_names()[:]
s=''
for i in imp1[-21:]:
    s=s+' '+features_name[i]
    print("{0}   {1}".format(features_name[i],imp[i]))
```

```
beware   0.0073059421293608625
guess   0.008186884002930578
waste   0.00830330484503956
stale   0.008703821300106192
easy   0.009015481300385517
trash   0.009791441280290808
delicious   0.009889016546116389
disgusting   0.009908485160003477
maybe   0.010508089567417282
horrible   0.010588572339199097
poor   0.011955597831906885
bad   0.012914187071559566
return   0.012953396604233891
would   0.013915547358218908
awful   0.014125081082876748
threw   0.015241512975657881
thought   0.019850187918536506
not   0.021223911702391785
money   0.026190885768896544
worst   0.031490735973931316
great   0.03344145739973202
```

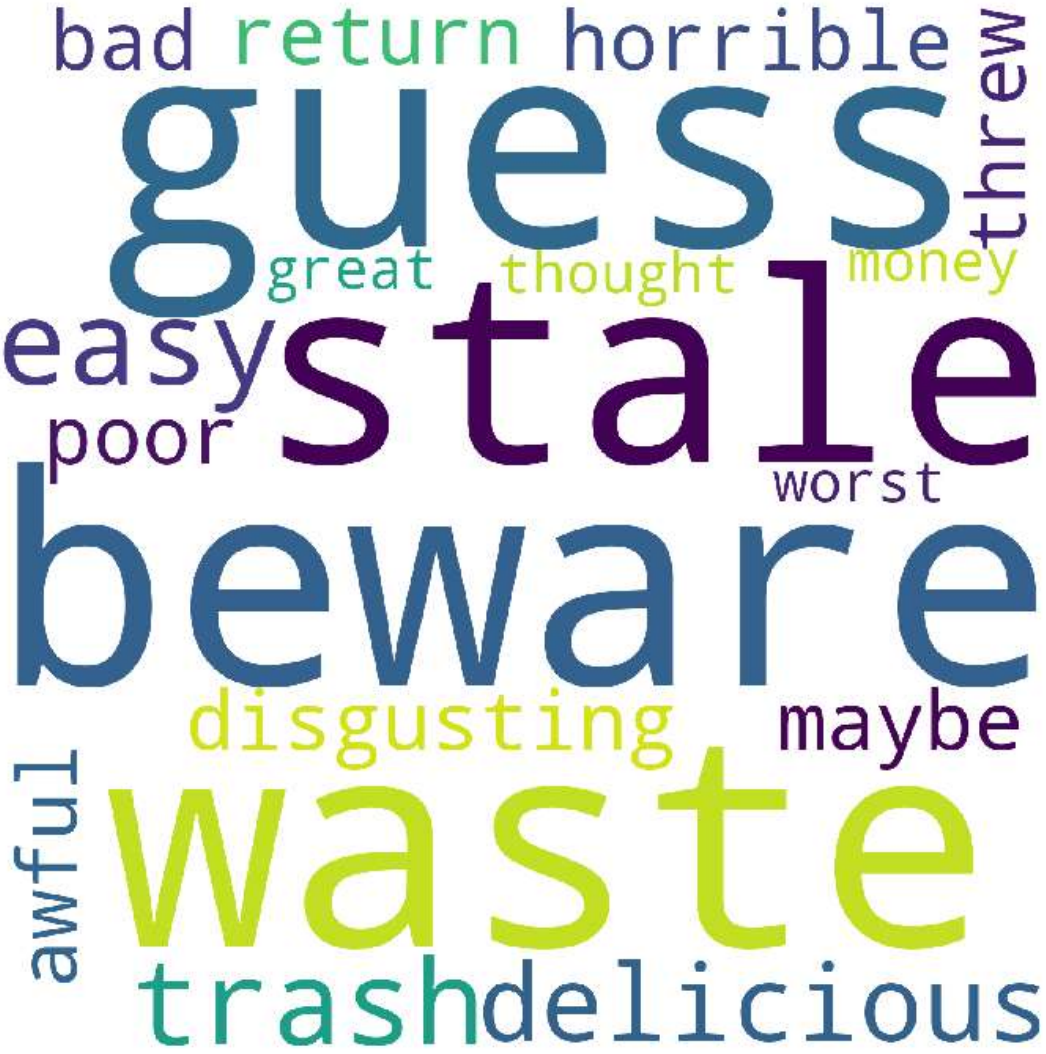
WORDCLOUD FOR TOP FEATURES

In [224]:

```
from wordcloud import WordCloud
wordcloud = WordCloud(width = 1600, height = 1600, stopwords=None,\
                      background_color ='white', min_font_size = 5).generate(s)

# plot the WordCloud image
plt.figure(figsize = (10,12), facecolor = 'white')
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



[5.1.3] Applying Random Forests on TFIDF, SET 2

In [182]:

```
from sklearn.ensemble import RandomForestClassifier as rfc
import seaborn as sns
```

In [183]:

```
auc_cv=[]
tpr_cv=[]
fpr_cv=[]
tpr_tr=[]
fpr_tr=[]
auc_tr=[]
data_rec=[]
data_rec=[]
hypermeter=[]
n_est = [5, 10, 50, 100, 200, 500, 1000]
max_dep = [2, 3, 5, 7,10, 12, 15, 20]
for i in n_est:
    for j in max_dep:
        clf=rfc(n_estimators=i,max_depth=j)
        clf.fit(final_tf_idf,y_tr)
        pred = clf.predict_proba(final_tf_idf_cv)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, pred)
        auc=metrics.auc(fpr, tpr)
        auc_cv.append(auc)
        tpr_cv.append(tpr)
        fpr_cv.append(fpr)

        # predict the response on the train
        pred = clf.predict_proba(final_tf_idf)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
        auc=metrics.auc(fpr, tpr)
        auc_tr.append(auc)
        tpr_tr.append(tpr)
        fpr_tr.append(fpr)
        hypermeter.append([i,j])
        print(i,j)
```

```
5 2
5 3
5 5
5 7
5 10
5 12
5 15
5 20
10 2
10 3
10 5
10 7
10 10
10 12
10 15
10 20
50 2
50 3
50 5
50 7
50 10
50 12
50 15
50 20
100 2
100 3
100 5
100 7
100 10
100 12
100 15
100 20
200 2
200 3
200 5
200 7
200 10
200 12
200 15
200 20
500 2
500 3
500 5
500 7
500 10
500 12
500 15
500 20
1000 2
1000 3
1000 5
1000 7
1000 10
1000 12
1000 15
1000 20
```

In [184]:

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

In [185]:

```
dep=[]
n_est=[]
for i in range(len(hypermeter)):
    n_est.append(hypermeter[i][0])
    dep.append(hypermeter[i][1])
```

In [186]:

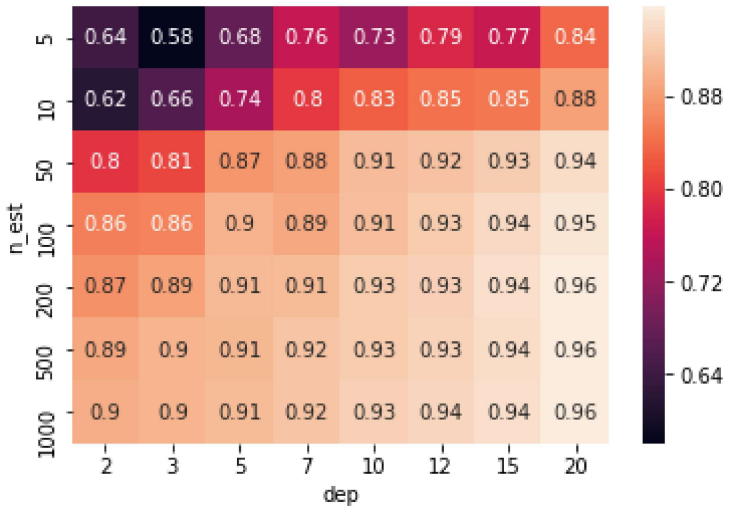
```
data1={'n_est':n_est,'dep':dep,'auc_cv':auc_cv[:],'auc_tr':auc_tr[:]}
data_f=pd.DataFrame(data1)
```

N_ESTIMATORS VS MAX_DEPTH

FOR TRAIN

In [187]:

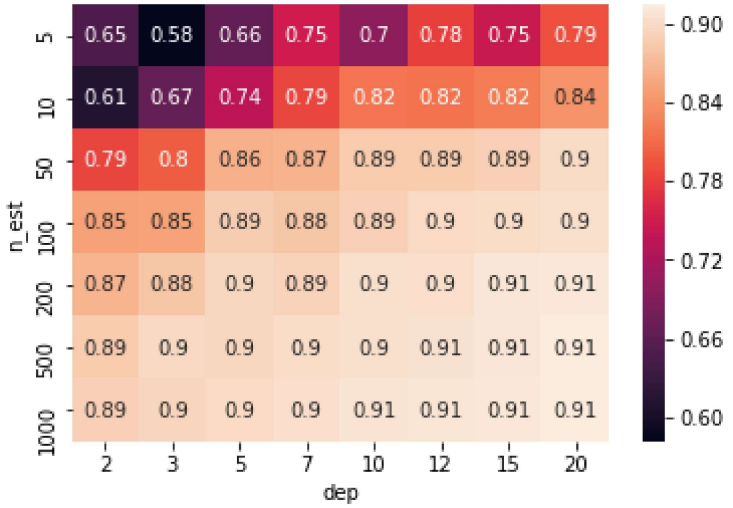
```
plo = data_f.pivot("n_est", "dep", "auc_tr");ax = sns.heatmap(plo, annot=True)
```



FOR CV

In [188]:

```
plo = data_f.pivot("n_est", "dep", "auc_cv");ax = sns.heatmap(plo, annot=True)
```



In [189]:

```
# TESTING THE MODEL ON TEST DATA

clf_final = rfc(n_estimators=10,max_depth=20)
clf_final.fit(final_tf_idf, y_tr)
pred_final = clf_final.predict_proba(final_tf_idf_test)[:,1]
fpr_f, tpr_f, thresholds_f = metrics.roc_curve(y_test, pred_final)
auc_final = metrics.auc(fpr_f, tpr_f)
auc_final
```

Out[189]:

```
0.8442510594845694
```

In [190]:

```
# TESTING THE MODEL ON TRAIN DATA

pred_train=clf_final.predict_proba(final_tf_idf)[:,1]
fpr_ft, tpr_ft, thresholds_ft=metrics.roc_curve(y_tr, pred_train)
auc_finalt=metrics.auc(fpr_ft, tpr_ft)
auc_finalt
```

Out[190]:

```
0.8908657504914879
```

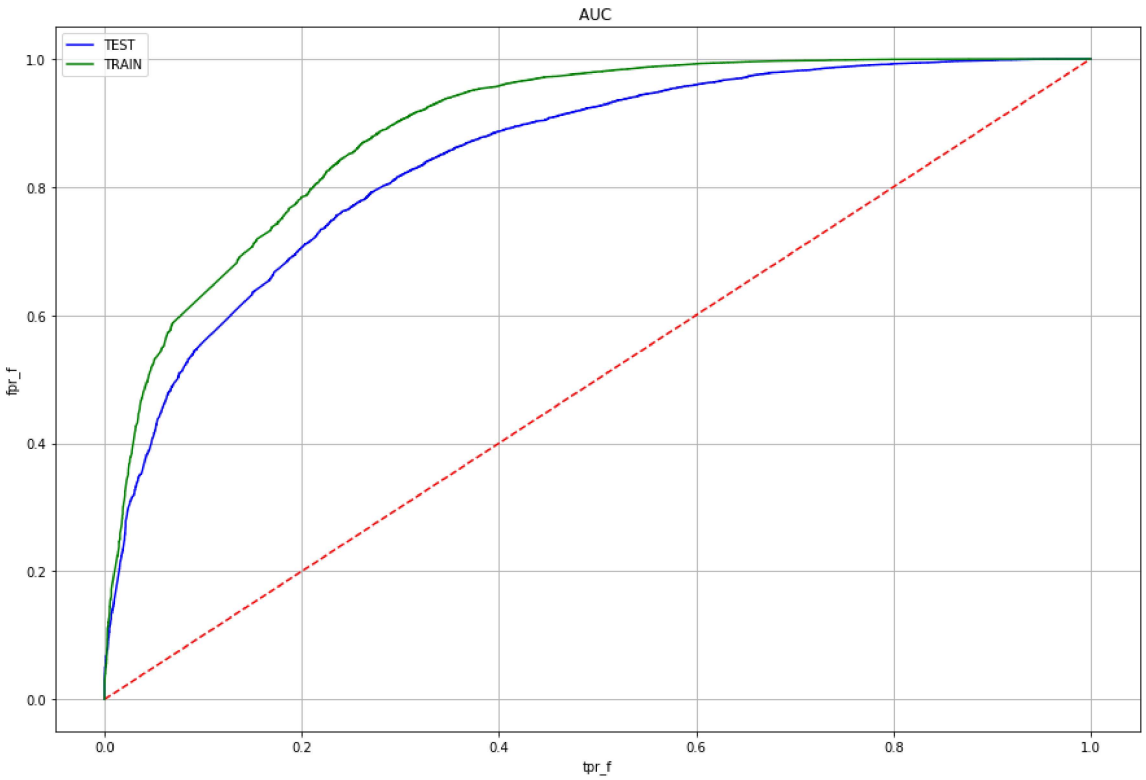
ROC CURVE

In [191]:

```
y_a=[0,0.5,1]
x_a=y_a
```

In [192]:

```
plt.figure(figsize=(15,10))
plt.plot(fpr_f,tpr_f,'b',label='TEST')
plt.plot(x_a,y_a,'--r')
plt.plot(fpr_ft,tpr_ft,'g',label='TRAIN')
plt.xlabel('tpr_f')
plt.ylabel('fpr_f')
plt.title('AUC ')
plt.grid()
plt.legend()
plt.show()
```



CONFUSION MATRIX

In [193]:

```
pred_final
pred_train
pred_final_classte=[]
pred_final_classtr=[]
for i in range(len(pred_final)):
    if pred_final[i]>0.5:
        pred_final_classte.append(1)
    elif pred_final[i]<=0.5:
            pred_final_classte.append(0)
for i in range(len(pred_train)):
    if pred_train[i]>0.5:
        pred_final_classtr.append(1)
    elif pred_train[i]<=0.5:
            pred_final_classtr.append(0)
```

TEST

In [194]:

```
cm = confusion_matrix(y_test, pred_final_classte)
cm
```

Out[194]:

```
array([[  144,  4413],
       [    3, 21772]], dtype=int64)
```

In [195]:

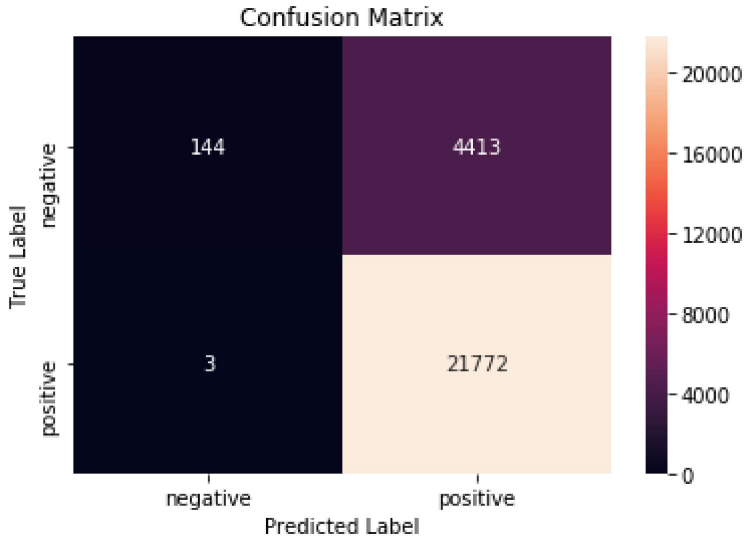
```
y_test["Score"].value_counts()
```

Out[195]:

```
1    21775
0     4557
Name: Score, dtype: int64
```

In [196]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



TRAIN

In [197]:

```
cm = confusion_matrix(y_tr, pred_final_classtr)
cm
```

Out[197]:

```
array([[  339,  6020],
       [    2, 36647]], dtype=int64)
```

In [198]:

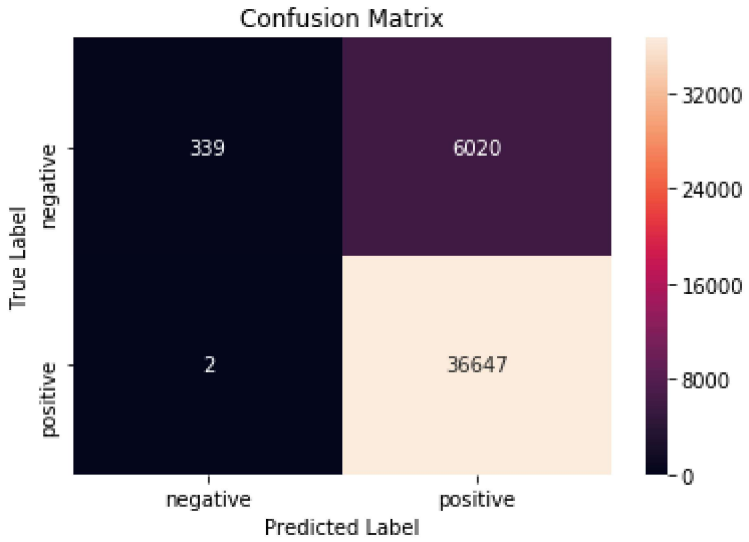
```
y_tr["Score"].value_counts()
```

Out[198]:

```
1    36649
0     6359
Name: Score, dtype: int64
```

In [199]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



CONCLUSION

In [200]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_final_classte))
```

```
              precision    recall  f1-score   support

           0       0.98      0.03      0.06      4557
           1       0.83      1.00      0.91     21775

   micro avg       0.83      0.83      0.83     26332
   macro avg       0.91      0.52      0.48     26332
weighted avg       0.86      0.83      0.76     26332
```

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

TOP 20 FEATURES

In [201]:

```
imp=clf_final.feature_importances_
```

In [202]:

```
imp1=np.argsort(imp)
```

In [203]:

```
features_name=count_vect.get_feature_names()[:]
s=''
for i in imp1[-21:]:
    s=s+' '+features_name[i]
    print("{0}   {1}".format(features_name[i],imp[i]))
```

```
maybe   0.00804294404341477
threw away   0.00860093043530056
would not   0.008614309119662645
return   0.008699425938863883
poor   0.009770253801220121
best   0.00992020490426555
not recommend   0.011982384084566877
horrible   0.012126391352590037
away   0.012780158078219295
waste money   0.01283471838835377
waste   0.013789053408477863
disappointment   0.014027181306267145
would   0.014862746727032536
threw   0.01500515687404958
bad   0.015857784542059362
disgusting   0.01801219496086897
disappointed   0.019509337516628273
great   0.022341615895747628
awful   0.022773635103315617
worst   0.02294455727021955
terrible   0.03606555834775608
```

WORDCLOUD FOR TOP FEATURES

In [204]:

```
from wordcloud import WordCloud
wordcloud = WordCloud(width = 1600, height = 1600, stopwords=None,\
                      background_color ='white', min_font_size = 5).generate(s)

# plot the WordCloud image
plt.figure(figsize = (10,12), facecolor = 'white')
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

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

In [205]:

```
from sklearn.ensemble import RandomForestClassifier as rfc
import seaborn as sns
```

In [206]:

```
auc_cv=[]
tpr_cv=[]
fpr_cv=[]
tpr_tr=[]
fpr_tr=[]
auc_tr=[]
data_rec=[]
data_rec=[]
hypermeter=[]
n_est = [5, 10, 50, 100, 200, 500, 1000]
max_dep = [2, 3, 5, 7,10, 12, 15, 20]
for i in n_est:
    for j in max_dep:
        clf=rfc(n_estimators=i,max_depth=j)
        clf.fit(sent_vectors_tr,y_tr)
        pred = clf.predict_proba(sent_vectors_cv)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, pred)
        auc=metrics.auc(fpr, tpr)
        auc_cv.append(auc)
        tpr_cv.append(tpr)
        fpr_cv.append(fpr)

        # predict the response on the train
        pred = clf.predict_proba(sent_vectors_tr)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
        auc=metrics.auc(fpr, tpr)
        auc_tr.append(auc)
        tpr_tr.append(tpr)
        fpr_tr.append(fpr)
        hypermeter.append([i,j])
        print(i,j)
```

```
5 2
5 3
5 5
5 7
5 10
5 12
5 15
5 20
10 2
10 3
10 5
10 7
10 10
10 12
10 15
10 20
50 2
50 3
50 5
50 7
50 10
50 12
50 15
50 20
100 2
100 3
100 5
100 7
100 10
100 12
100 15
100 20
200 2
200 3
200 5
200 7
200 10
200 12
200 15
200 20
500 2
500 3
500 5
500 7
500 10
500 12
500 15
500 20
1000 2
1000 3
1000 5
1000 7
1000 10
1000 12
1000 15
1000 20
```

In [207]:

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

In [208]:

```
dep=[]
n_est=[]
for i in range(len(hypermeter)):
    n_est.append(hypermeter[i][0])
    dep.append(hypermeter[i][1])
```

In [209]:

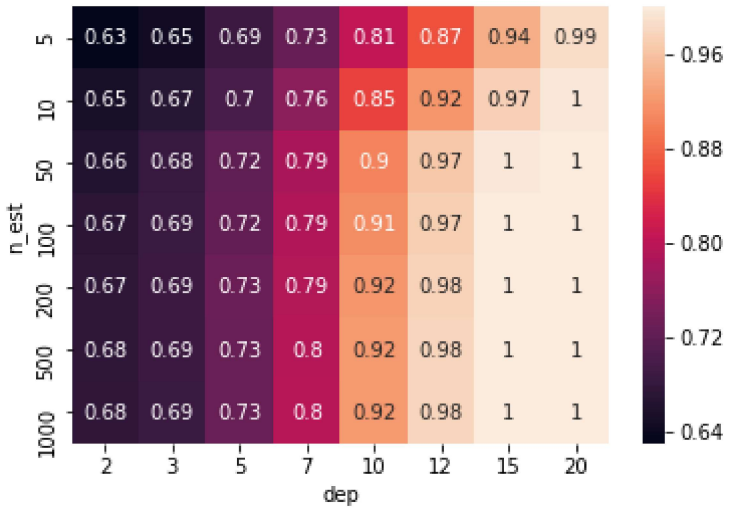
```
data1={'n_est':n_est,'dep':dep,'auc_cv':auc_cv[:],'auc_tr':auc_tr[:]}
data_f=pd.DataFrame(data1)
```

N_ESTIMATORS VS MAX_DEPTH

FOR TRAIN

In [210]:

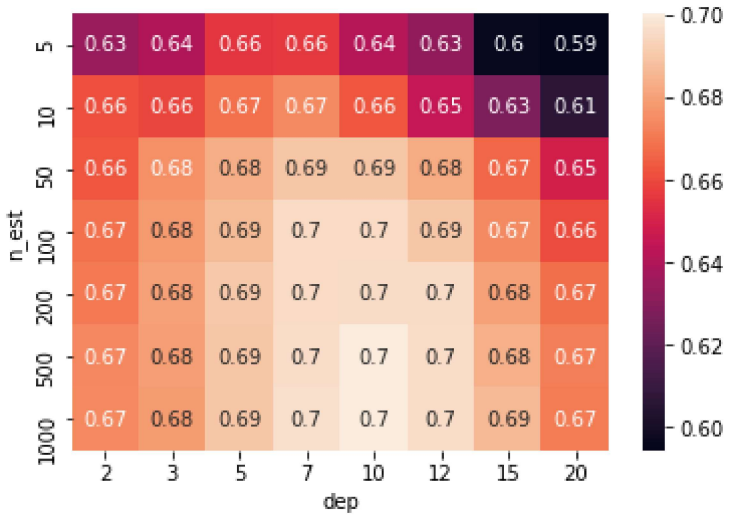
```
plo = data_f.pivot("n_est", "dep", "auc_tr");ax = sns.heatmap(plo, annot=True)
```



FOR CV

In [211]:

```
plo = data_f.pivot("n_est", "dep", "auc_cv");ax = sns.heatmap(plo, annot=True)
```



In [212]:

```
# TESTING THE MODEL ON TEST DATA

clf_final = rfc(n_estimators=1000,max_depth=15)
clf_final.fit(sent_vectors_tr, y_tr)
pred_final = clf_final.predict_proba(sent_vectors_test)[:,1]
fpr_f, tpr_f, thresholds_f = metrics.roc_curve(y_test, pred_final)
auc_final = metrics.auc(fpr_f, tpr_f)
auc_final
```

Out[212]:

```
0.693254152592484
```

In [213]:

```
# TESTING THE MODEL ON TRAIN DATA

pred_train=clf_final.predict_proba(sent_vectors_tr)[:,1]
fpr_ft, tpr_ft, thresholds_ft=metrics.roc_curve(y_tr, pred_train)
auc_finalt=metrics.auc(fpr_ft, tpr_ft)
auc_finalt
```

Out[213]:

```
0.9987116896662328
```

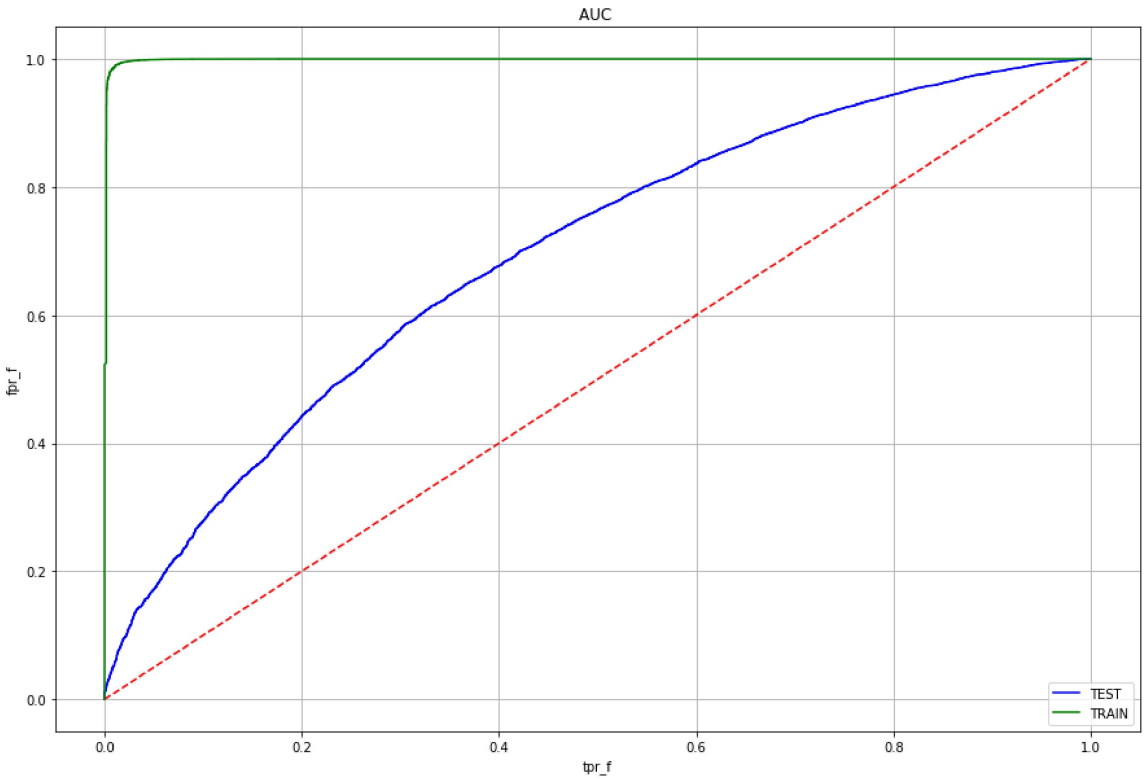
ROC CURVE

In [214]:

```
y_a=[0,0.5,1]
x_a=y_a
```

In [215]:

```
plt.figure(figsize=(15,10))
plt.plot(fpr_f,tpr_f,'b',label='TEST')
plt.plot(x_a,y_a,'--r')
plt.plot(fpr_ft,tpr_ft,'g',label='TRAIN')
plt.xlabel('tpr_f')
plt.ylabel('fpr_f')
plt.title('AUC ')
plt.grid()
plt.legend()
plt.show()
```



CONFUSION MATRIX

In [216]:

```
pred_final
pred_train
pred_final_classte=[]
pred_final_classtr=[]
for i in range(len(pred_final)):
    if pred_final[i]>0.5:
        pred_final_classte.append(1)
    elif pred_final[i]<=0.5:
            pred_final_classte.append(0)
for i in range(len(pred_train)):
    if pred_train[i]>0.5:
        pred_final_classtr.append(1)
    elif pred_train[i]<=0.5:
            pred_final_classtr.append(0)
```

TEST

In [217]:

```
cm = confusion_matrix(y_test, pred_final_classte)
cm
```

Out[217]:

```
array([[   17,  4540],
       [    4, 21771]], dtype=int64)
```

In [218]:

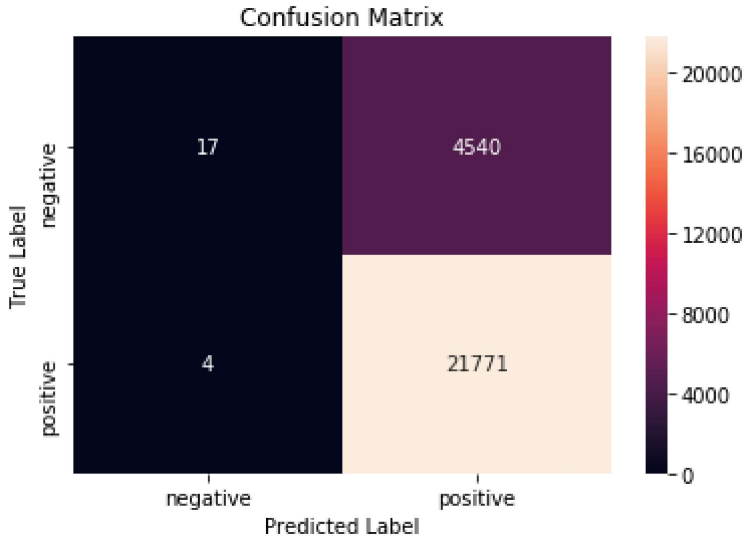
```
y_test["Score"].value_counts()
```

Out[218]:

```
1    21775
0     4557
Name: Score, dtype: int64
```

In [219]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



TRAIN

In [220]:

```
cm = confusion_matrix(y_tr, pred_final_classtr)
cm
```

Out[220]:

```
array([[ 1951,  4408],
       [    0, 36649]], dtype=int64)
```

In [221]:

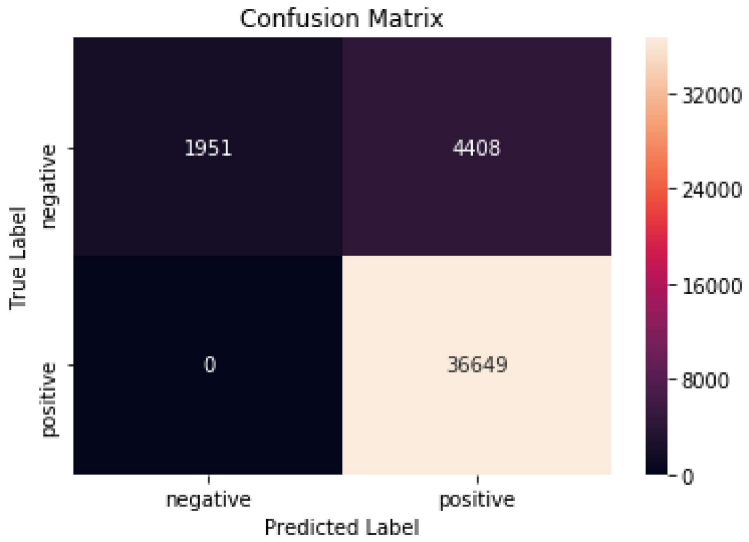
```
y_tr["Score"].value_counts()
```

Out[221]:

```
1    36649
0     6359
Name: Score, dtype: int64
```

In [222]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



CONCLUSION

In [223]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_final_classte))
```

```
              precision    recall  f1-score   support

           0       0.81      0.00      0.01      4557
           1       0.83      1.00      0.91     21775

   micro avg       0.83      0.83      0.83     26332
   macro avg       0.82      0.50      0.46     26332
weighted avg       0.82      0.83      0.75     26332
```

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

In [224]:

```
from sklearn.ensemble import RandomForestClassifier as rfc
import seaborn as sns
```

In [225]:

```
auc_cv=[]
tpr_cv=[]
fpr_cv=[]
tpr_tr=[]
fpr_tr=[]
auc_tr=[]
data_rec=[]
data_rec=[]
hypermeter=[]
n_est = [5, 10, 50, 100, 200, 500, 1000]
max_dep = [2, 3, 5, 7,10, 12, 15, 20]
for i in n_est:
    for j in max_dep:
        clf=rfc(n_estimators=i,max_depth=j)
        clf.fit(tfidf_sent_vectors,y_tr)
        pred = clf.predict_proba(tfidf_sent_vectors_cv)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, pred)
        auc=metrics.auc(fpr, tpr)
        auc_cv.append(auc)
        tpr_cv.append(tpr)
        fpr_cv.append(fpr)

        # predict the response on the train
        pred = clf.predict_proba(tfidf_sent_vectors)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
        auc=metrics.auc(fpr, tpr)
        auc_tr.append(auc)
        tpr_tr.append(tpr)
        fpr_tr.append(fpr)
        hypermeter.append([i,j])
        print(i,j)
```

```
5 2
5 3
5 5
5 7
5 10
5 12
5 15
5 20
10 2
10 3
10 5
10 7
10 10
10 12
10 15
10 20
50 2
50 3
50 5
50 7
50 10
50 12
50 15
50 20
100 2
100 3
100 5
100 7
100 10
100 12
100 15
100 20
200 2
200 3
200 5
200 7
200 10
200 12
200 15
200 20
500 2
500 3
500 5
500 7
500 10
500 12
500 15
500 20
1000 2
1000 3
1000 5
1000 7
1000 10
1000 12
1000 15
1000 20
```

In [226]:

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

In [227]:

```
dep=[]
n_est=[]
for i in range(len(hypermeter)):
    n_est.append(hypermeter[i][0])
    dep.append(hypermeter[i][1])
```

In [228]:

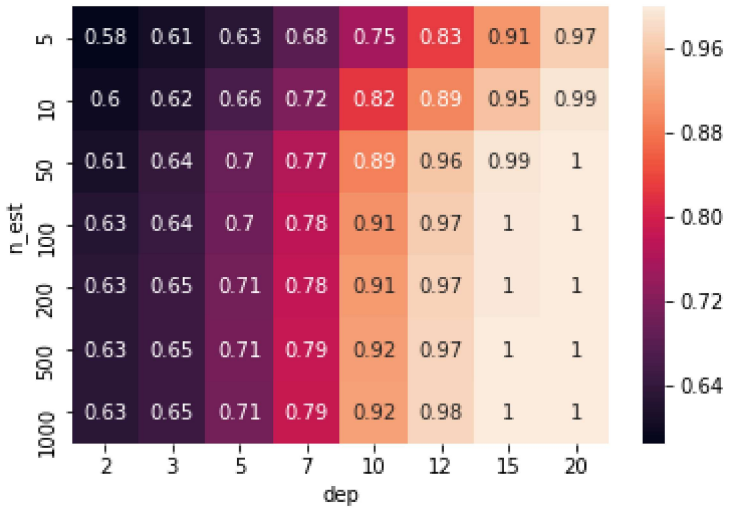
```
data1={'n_est':n_est,'dep':dep,'auc_cv':auc_cv[:],'auc_tr':auc_tr[:]}
data_f=pd.DataFrame(data1)
```

N_ESTIMATORS VS MAX_DEPTH

FOR TRAIN

In [229]:

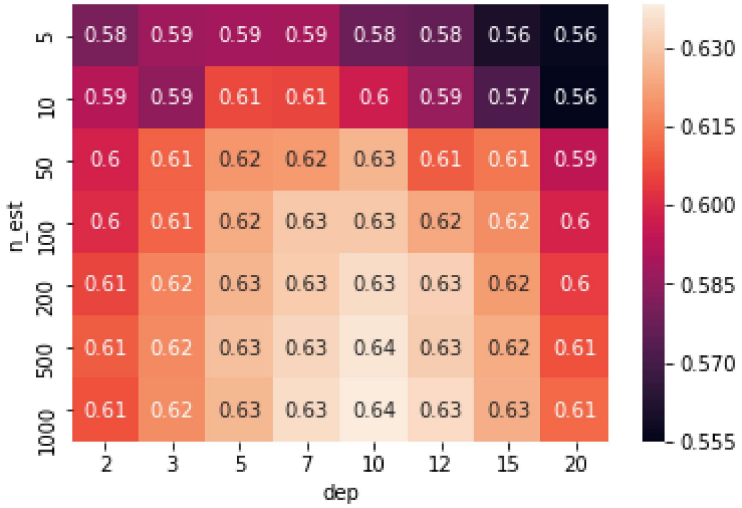
```
plo = data_f.pivot("n_est", "dep", "auc_tr");ax = sns.heatmap(plo, annot=True)
```



FOR CV

In [230]:

```
plo = data_f.pivot("n_est", "dep", "auc_cv");ax = sns.heatmap(plo, annot=True)
```



In [245]:

```
# TESTING THE MODEL ON TEST DATA

clf_final = rfc(n_estimators=500,max_depth=10)
clf_final.fit(tfidf_sent_vectors, y_tr)
pred_final = clf_final.predict_proba(tfidf_sent_vectors_test)[:,1]
fpr_f, tpr_f, thresholds_f = metrics.roc_curve(y_test, pred_final)
auc_final = metrics.auc(fpr_f, tpr_f)
auc_final
```

Out[245]:

```
0.6466805588203208
```

In [246]:

```
# TESTING THE MODEL ON TRAIN DATA

pred_train=clf_final.predict_proba(tfidf_sent_vectors)[:,1]
fpr_ft, tpr_ft, thresholds_ft=metrics.roc_curve(y_tr, pred_train)
auc_finalt=metrics.auc(fpr_ft, tpr_ft)
auc_finalt
```

Out[246]:

```
0.917311263439339
```

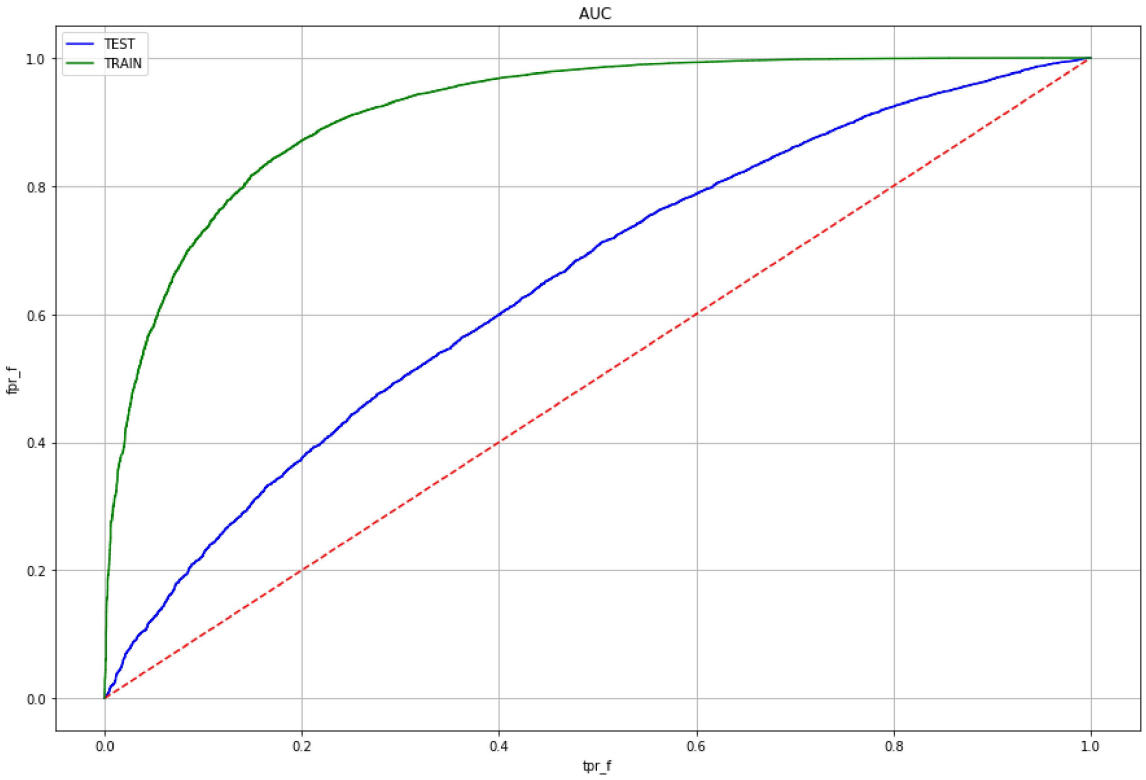
ROC CURVE

In [247]:

```
y_a=[0,0.5,1]
x_a=y_a
```

In [248]:

```
plt.figure(figsize=(15,10))
plt.plot(fpr_f,tpr_f,'b',label='TEST')
plt.plot(x_a,y_a,'--r')
plt.plot(fpr_ft,tpr_ft,'g',label='TRAIN')
plt.xlabel('tpr_f')
plt.ylabel('fpr_f')
plt.title('AUC ')
plt.grid()
plt.legend()
plt.show()
```



CONFUSION MATRIX

In [249]:

```
pred_final
pred_train
pred_final_classte=[]
pred_final_classtr=[]
for i in range(len(pred_final)):
    if pred_final[i]>0.5:
        pred_final_classte.append(1)
    elif pred_final[i]<=0.5:
            pred_final_classte.append(0)
for i in range(len(pred_train)):
    if pred_train[i]>0.5:
        pred_final_classtr.append(1)
    elif pred_train[i]<=0.5:
            pred_final_classtr.append(0)
```

TEST

In [250]:

```
cm = confusion_matrix(y_test, pred_final_classte)
cm
```

Out[250]:

```
array([[    3,  4554],
       [    1, 21774]], dtype=int64)
```

In [251]:

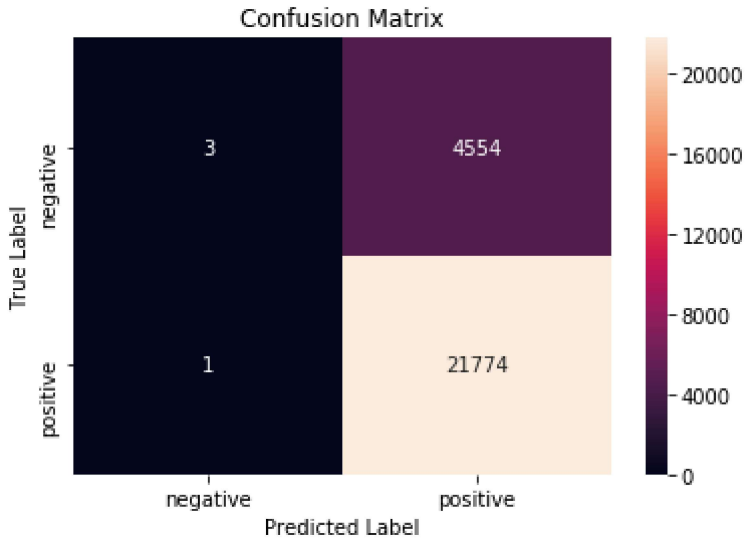
```
y_test["Score"].value_counts()
```

Out[251]:

```
1    21775
0     4557
Name: Score, dtype: int64
```

In [252]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



TRAIN

In [253]:

```
cm = confusion_matrix(y_tr, pred_final_classtr)
cm
```

Out[253]:

```
array([[   83,  6276],
       [    0, 36649]], dtype=int64)
```

In [254]:

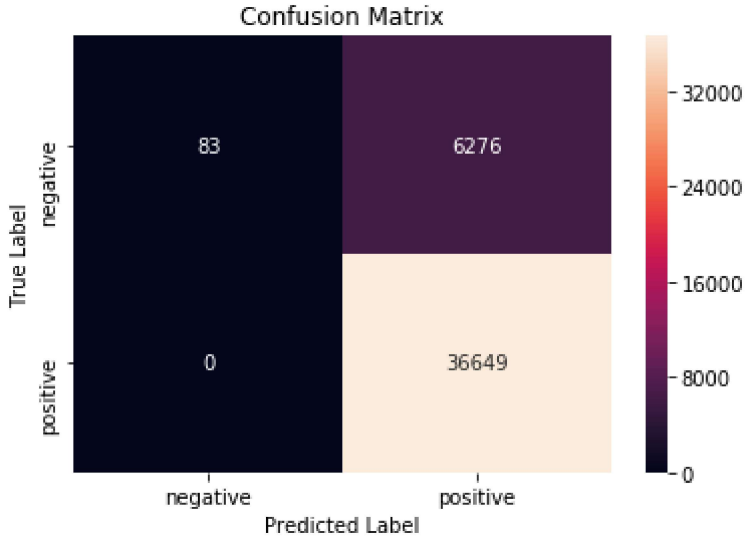
```
y_tr["Score"].value_counts()
```

Out[254]:

```
1    36649
0     6359
Name: Score, dtype: int64
```

In [255]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



CONCLUSION

In [256]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_final_classte))
```

```
              precision    recall  f1-score   support

           0       0.75      0.00      0.00      4557
           1       0.83      1.00      0.91     21775

   micro avg       0.83      0.83      0.83     26332
   macro avg       0.79      0.50      0.45     26332
weighted avg       0.81      0.83      0.75     26332
```

[5.2] Applying GBDT using XGBOOST

[5.2.1] Applying XGBOOST on BOW, SET 1

In [286]:

```
# from sklearn.ensemble import GradientBoostingClassifier as gbc
import xgboost as gbc
import seaborn as sns
```

In [264]:

```
auc_cv=[]
tpr_cv=[]
fpr_cv=[]
tpr_tr=[]
fpr_tr=[]
auc_tr=[]
data_rec=[]
data_rec=[]
hypermeter=[]
n_est = [5, 10, 50, 100, 200, 500, 1000]
max_dep = [2, 3, 5, 7,10, 12, 15, 20]
for i in n_est:
    for j in max_dep:
        clf=gbc.XGBClassifier(n_estimators=i,max_depth=j,n_jobs=-1)
        clf.fit(final_counts,y_tr)
        pred = clf.predict_proba(final_counts_cv)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, pred)
        auc=metrics.auc(fpr, tpr)
        auc_cv.append(auc)
        tpr_cv.append(tpr)
        fpr_cv.append(fpr)

        # predict the response on the train
        pred = clf.predict_proba(final_counts)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
        auc=metrics.auc(fpr, tpr)
        auc_tr.append(auc)
        tpr_tr.append(tpr)
        fpr_tr.append(fpr)
        hypermeter.append([i,j])
        print(i,j)
```

```
5 2
5 3
5 5
5 7
5 10
5 12
5 15
5 20
10 2
10 3
10 5
10 7
10 10
10 12
10 15
10 20
50 2
50 3
50 5
50 7
50 10
50 12
50 15
50 20
100 2
100 3
100 5
100 7
100 10
100 12
100 15
100 20
200 2
200 3
200 5
200 7
200 10
200 12
200 15
200 20
500 2
500 3
500 5
500 7
500 10
500 12
500 15
500 20
1000 2
1000 3
1000 5
1000 7
1000 10
1000 12
1000 15
1000 20
```

In [265]:

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

In [266]:

```
dep=[]
n_est=[]
for i in range(len(hypermeter)):
    n_est.append(hypermeter[i][0])
    dep.append(hypermeter[i][1])
```

In [267]:

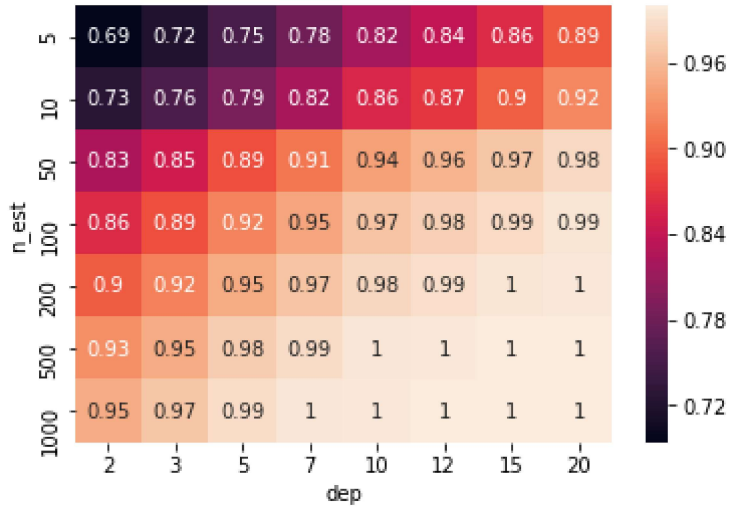
```
data1={'n_est':n_est,'dep':dep,'auc_cv':auc_cv[:],'auc_tr':auc_tr[:]}
data_f=pd.DataFrame(data1)
```

N_ESTIMATORS VS MAX_DEPTH

FOR TRAIN

In [268]:

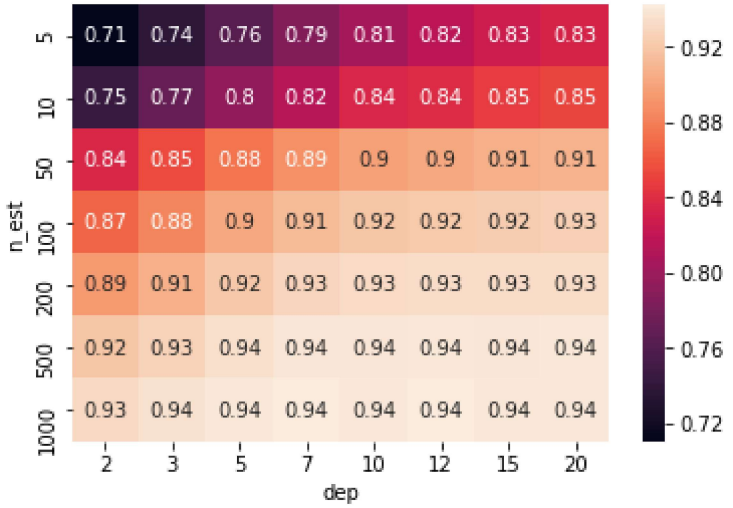
```
plo = data_f.pivot("n_est", "dep", "auc_tr");ax = sns.heatmap(plo, annot=True)
```



FOR CV

In [269]:

```
plo = data_f.pivot("n_est", "dep", "auc_cv");ax = sns.heatmap(plo, annot=True)
```



In [308]:

```
# TESTING THE MODEL ON TEST DATA

clf_final = gbc.XGBClassifier(n_estimators=500,max_depth=5,n_jobs=-1)
clf_final.fit(final_counts, y_tr)
pred_final = clf_final.predict_proba(final_counts_test)[:,1]
fpr_f, tpr_f, thresholds_f = metrics.roc_curve(y_test, pred_final)
auc_final = metrics.auc(fpr_f, tpr_f)
auc_final
```

Out[308]:

```
0.9368994647968443
```

In [309]:

```
# TESTING THE MODEL ON TRAIN DATA

pred_train=clf_final.predict_proba(final_counts)[:,1]
fpr_ft, tpr_ft, thresholds_ft=metrics.roc_curve(y_tr, pred_train)
auc_finalt=metrics.auc(fpr_ft, tpr_ft)
auc_finalt
```

Out[309]:

```
0.9756077072420601
```

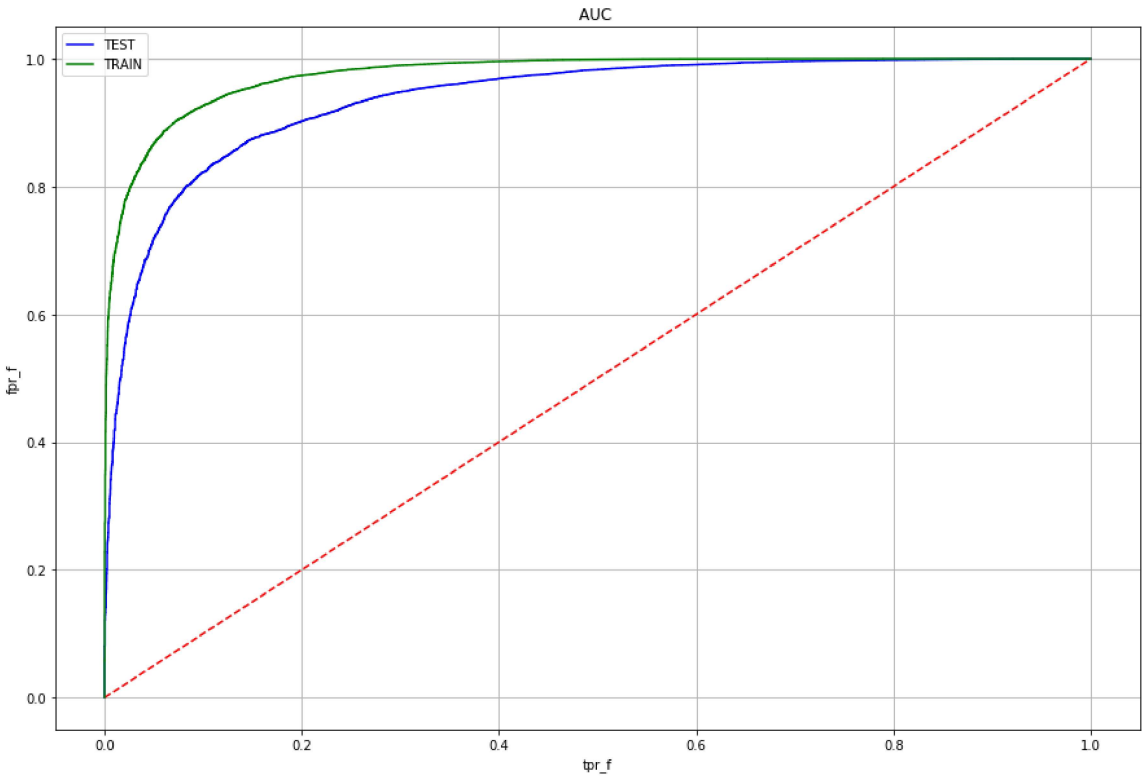
ROC CURVE

In [310]:

```
y_a=[0,0.5,1]
x_a=y_a
```

In [311]:

```
plt.figure(figsize=(15,10))
plt.plot(fpr_f,tpr_f,'b',label='TEST')
plt.plot(x_a,y_a,'--r')
plt.plot(fpr_ft,tpr_ft,'g',label='TRAIN')
plt.xlabel('tpr_f')
plt.ylabel('fpr_f')
plt.title('AUC ')
plt.grid()
plt.legend()
plt.show()
```



CONFUSION MATRIX

In [312]:

```
pred_final
pred_train
pred_final_classte=[]
pred_final_classtr=[]
for i in range(len(pred_final)):
    if pred_final[i]>0.5:
        pred_final_classte.append(1)
    elif pred_final[i]<=0.5:
            pred_final_classte.append(0)
for i in range(len(pred_train)):
    if pred_train[i]>0.5:
        pred_final_classtr.append(1)
    elif pred_train[i]<=0.5:
            pred_final_classtr.append(0)
```

TEST

In [313]:

```
cm = confusion_matrix(y_test, pred_final_classte)
cm
```

Out[313]:

```
array([[ 2281,  2276],
       [  369, 21406]], dtype=int64)
```

In [314]:

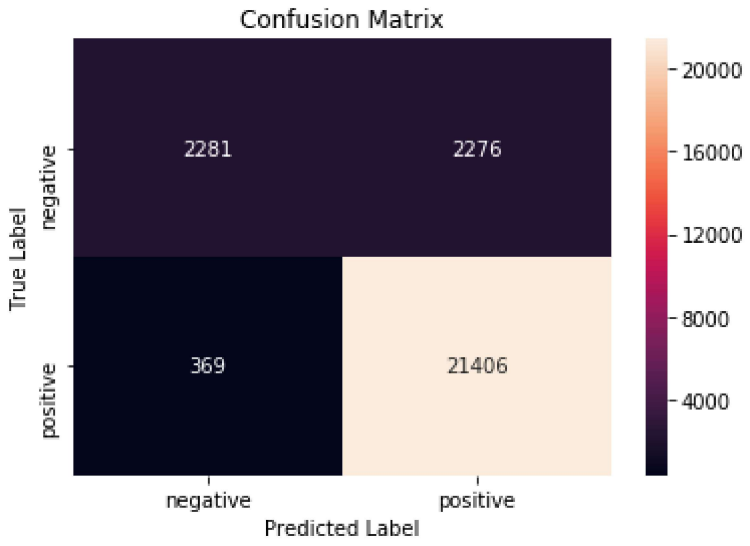
```
y_test["Score"].value_counts()
```

Out[314]:

```
1    21775
0     4557
Name: Score, dtype: int64
```

In [315]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



TRAIN

In [316]:

```
cm = confusion_matrix(y_tr, pred_final_classtr)
cm
```

Out[316]:

```
array([[ 3994,  2365],
       [  205, 36444]], dtype=int64)
```

In [317]:

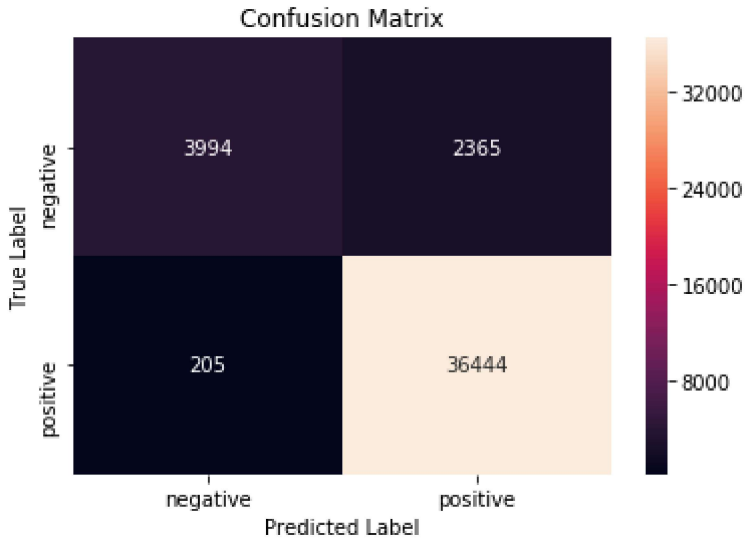
```
y_tr["Score"].value_counts()
```

Out[317]:

```
1    36649
0     6359
Name: Score, dtype: int64
```

In [318]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



CONCLUSION

In [319]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_final_classte))
```

```
              precision    recall  f1-score   support

           0       0.86      0.50      0.63      4557
           1       0.90      0.98      0.94     21775

   micro avg       0.90      0.90      0.90     26332
   macro avg       0.88      0.74      0.79     26332
weighted avg       0.90      0.90      0.89     26332
```

[5.2.2] Applying XGBOOST on TFIDF, SET 2

In [272]:

```
auc_cv=[]
tpr_cv=[]
fpr_cv=[]
tpr_tr=[]
fpr_tr=[]
auc_tr=[]
data_rec=[]
data_rec=[]
hypermeter=[]
n_est = [5, 10, 50, 100, 200, 500, 1000]
max_dep = [2, 3, 5, 7,10, 12, 15, 20]
for i in n_est:
    for j in max_dep:
        clf=gbc.XGBClassifier(n_estimators=i,max_depth=j,n_jobs=-1)
        clf.fit(final_tf_idf,y_tr)
        pred = clf.predict_proba(final_tf_idf_cv)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, pred)
        auc=metrics.auc(fpr, tpr)
        auc_cv.append(auc)
        tpr_cv.append(tpr)
        fpr_cv.append(fpr)

        # predict the response on the train
        pred = clf.predict_proba(final_tf_idf)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
        auc=metrics.auc(fpr, tpr)
        auc_tr.append(auc)
        tpr_tr.append(tpr)
        fpr_tr.append(fpr)
        hypermeter.append([i,j])
        print(i,j)
```

```
5 2
5 3
5 5
5 7
5 10
5 12
5 15
5 20
10 2
10 3
10 5
10 7
10 10
10 12
10 15
10 20
50 2
50 3
50 5
50 7
50 10
50 12
50 15
50 20
100 2
100 3
100 5
100 7
100 10
100 12
100 15
100 20
200 2
200 3
200 5
200 7
200 10
200 12
200 15
200 20
500 2
500 3
500 5
500 7
500 10
500 12
500 15
500 20
1000 2
1000 3
1000 5
1000 7
1000 10
1000 12
1000 15
1000 20
```

In [273]:

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

In [274]:

```
dep=[]
n_est=[]
for i in range(len(hypermeter)):
    n_est.append(hypermeter[i][0])
    dep.append(hypermeter[i][1])
```

In [275]:

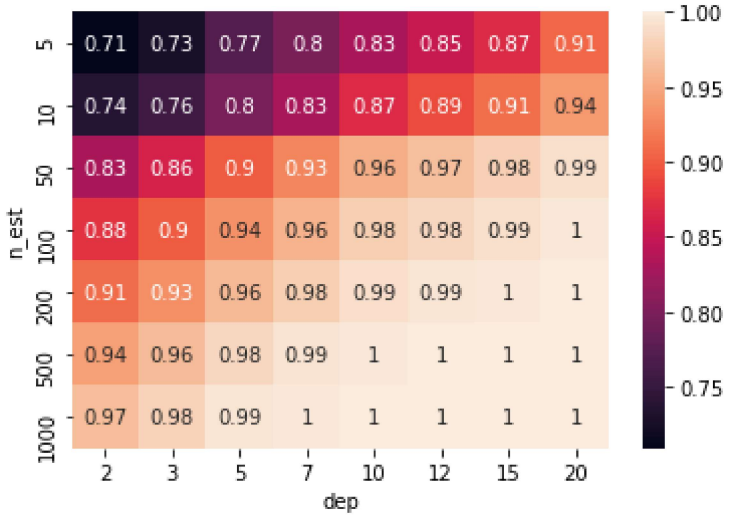
```
data1={'n_est':n_est,'dep':dep,'auc_cv':auc_cv[:],'auc_tr':auc_tr[:]}
data_f=pd.DataFrame(data1)
```

N_ESTIMATORS VS MAX_DEPTH

FOR TRAIN

In [276]:

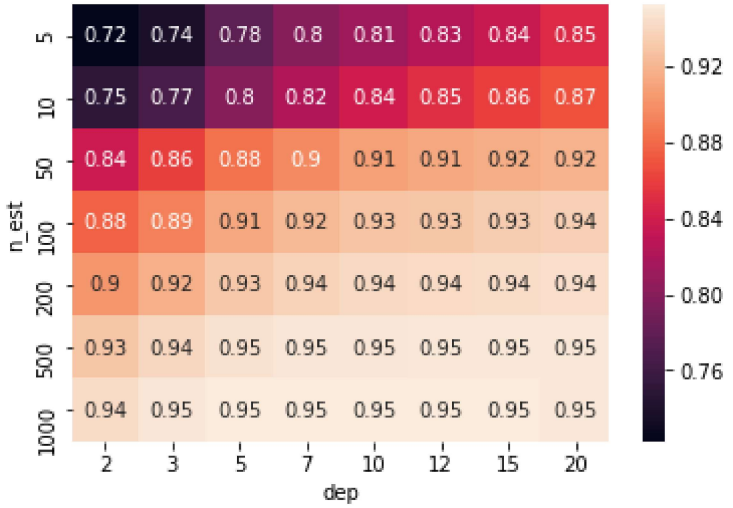
```
plo = data_f.pivot("n_est", "dep", "auc_tr");ax = sns.heatmap(plo, annot=True)
```



FOR CV

In [277]:

```
plo = data_f.pivot("n_est", "dep", "auc_cv");ax = sns.heatmap(plo, annot=True)
```



In [320]:

```
# TESTING THE MODEL ON TEST DATA

clf_final = gbc.XGBClassifier(n_estimators=500,max_depth=5,n_job=-1)
clf_final.fit(final_tf_idf, y_tr)
pred_final = clf_final.predict_proba(final_tf_idf_test)[:,1]
fpr_f, tpr_f, thresholds_f = metrics.roc_curve(y_test, pred_final)
auc_final = metrics.auc(fpr_f, tpr_f)
auc_final
```

Out[320]:

```
0.9492201019513764
```

In [321]:

```
# TESTING THE MODEL ON TRAIN DATA

pred_train=clf_final.predict_proba(final_tf_idf)[:,1]
fpr_ft, tpr_ft, thresholds_ft=metrics.roc_curve(y_tr, pred_train)
auc_finalt=metrics.auc(fpr_ft, tpr_ft)
auc_finalt
```

Out[321]:

```
0.9848145936440149
```

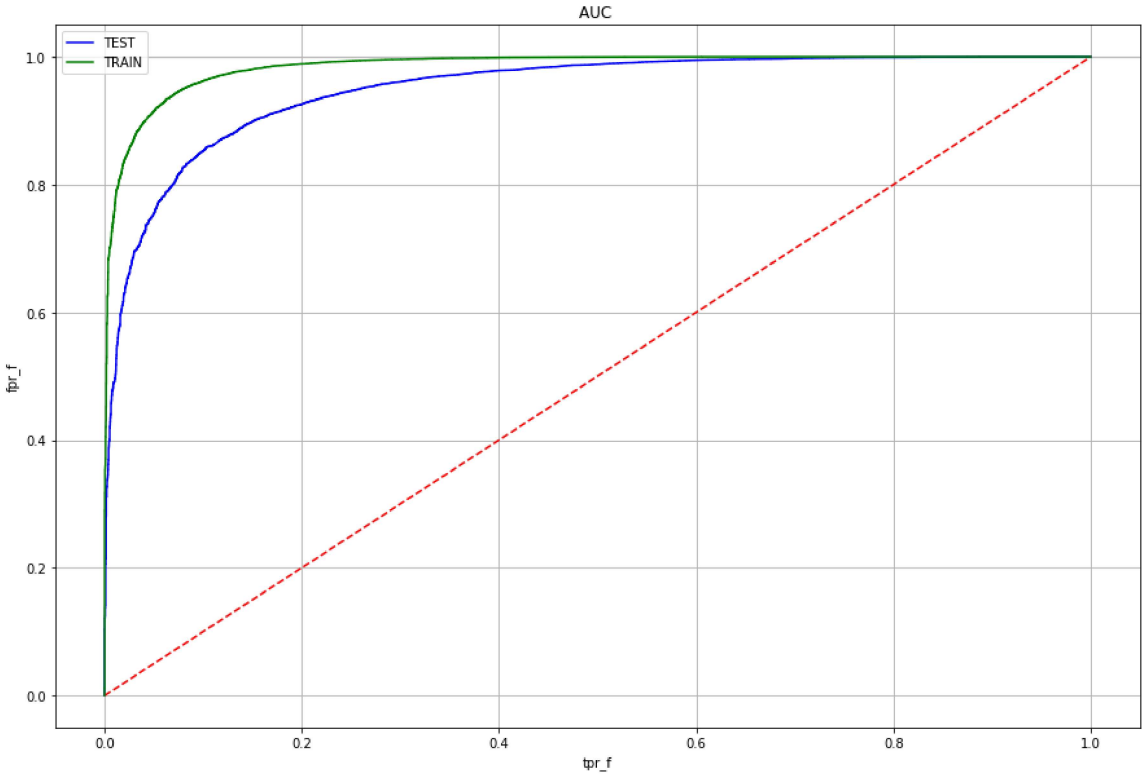
ROC CURVE

In [322]:

```
y_a=[0,0.5,1]
x_a=y_a
```

In [323]:

```
plt.figure(figsize=(15,10))
plt.plot(fpr_f,tpr_f,'b',label='TEST')
plt.plot(x_a,y_a,'--r')
plt.plot(fpr_ft,tpr_ft,'g',label='TRAIN')
plt.xlabel('tpr_f')
plt.ylabel('fpr_f')
plt.title('AUC ')
plt.grid()
plt.legend()
plt.show()
```



CONFUSION MATRIX

In [324]:

```
pred_final
pred_train
pred_final_classte=[]
pred_final_classtr=[]
for i in range(len(pred_final)):
    if pred_final[i]>0.5:
        pred_final_classte.append(1)
    elif pred_final[i]<=0.5:
            pred_final_classte.append(0)
for i in range(len(pred_train)):
    if pred_train[i]>0.5:
        pred_final_classtr.append(1)
    elif pred_train[i]<=0.5:
            pred_final_classtr.append(0)
```

TEST

In [325]:

```
cm = confusion_matrix(y_test, pred_final_classte)
cm
```

Out[325]:

```
array([[ 2463,  2094],
       [  334, 21441]], dtype=int64)
```

In [326]:

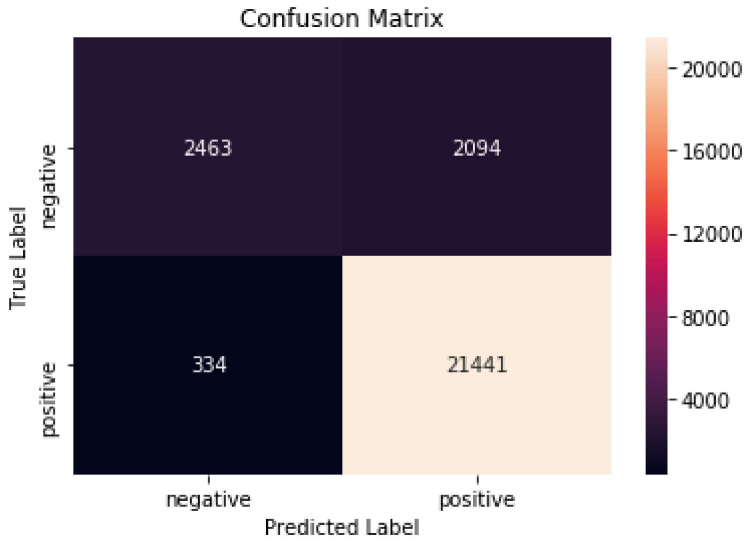
```
y_test["Score"].value_counts()
```

Out[326]:

```
1    21775
0     4557
Name: Score, dtype: int64
```

In [327]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



TRAIN

In [328]:

```
cm = confusion_matrix(y_tr, pred_final_classtr)
cm
```

Out[328]:

```
array([[ 4376,  1983],
       [  126, 36523]], dtype=int64)
```

In [329]:

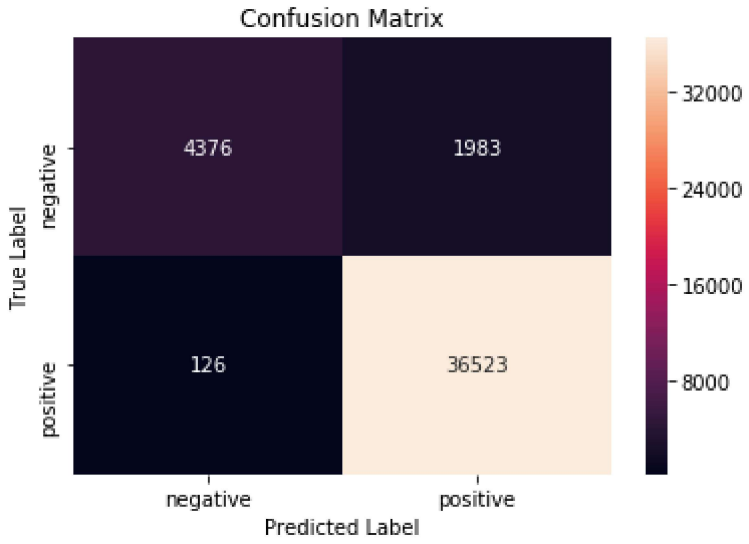
```
y_tr["Score"].value_counts()
```

Out[329]:

```
1    36649
0     6359
Name: Score, dtype: int64
```

In [330]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



CONCLUSION

In [331]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_final_classte))
```

```
              precision    recall  f1-score   support

           0       0.88      0.54      0.67      4557
           1       0.91      0.98      0.95     21775

   micro avg       0.91      0.91      0.91     26332
   macro avg       0.90      0.76      0.81     26332
weighted avg       0.91      0.91      0.90     26332
```

[5.2.3] Applying XGBOOST on AVG W2V, SET 3

In [301]:

```
auc_cv=[]
tpr_cv=[]
fpr_cv=[]
tpr_tr=[]
fpr_tr=[]
auc_tr=[]
data_rec=[]
data_rec=[]
hypermeter=[]
n_est = [5, 10, 50, 100, 200, 500, 1000]
max_dep = [2, 3, 5, 7,10, 12, 15, 20]
for i in n_est:
    for j in max_dep:
        clf=gbc.XGBClassifier(n_estimators=i,max_depth=j,n_jobs=-1)
        clf.fit(np.array(sent_vectors_tr),y_tr)
        pred = clf.predict_proba(sent_vectors_cv)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, pred)
        auc=metrics.auc(fpr, tpr)
        auc_cv.append(auc)
        tpr_cv.append(tpr)
        fpr_cv.append(fpr)

        # predict the response on the train
        pred = clf.predict_proba(sent_vectors_tr)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
        auc=metrics.auc(fpr, tpr)
        auc_tr.append(auc)
        tpr_tr.append(tpr)
        fpr_tr.append(fpr)
        hypermeter.append([i,j])
        print(i,j)
```

```
5 2
5 3
5 5
5 7
5 10
5 12
5 15
5 20
10 2
10 3
10 5
10 7
10 10
10 12
10 15
10 20
50 2
50 3
50 5
50 7
50 10
50 12
50 15
50 20
100 2
100 3
100 5
100 7
100 10
100 12
100 15
100 20
200 2
200 3
200 5
200 7
200 10
200 12
200 15
200 20
500 2
500 3
500 5
500 7
500 10
500 12
500 15
500 20
1000 2
1000 3
1000 5
1000 7
1000 10
1000 12
1000 15
1000 20
```

In [302]:

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

In [303]:

```
dep=[]
n_est=[]
for i in range(len(hypermeter)):
    n_est.append(hypermeter[i][0])
    dep.append(hypermeter[i][1])
```

In [304]:

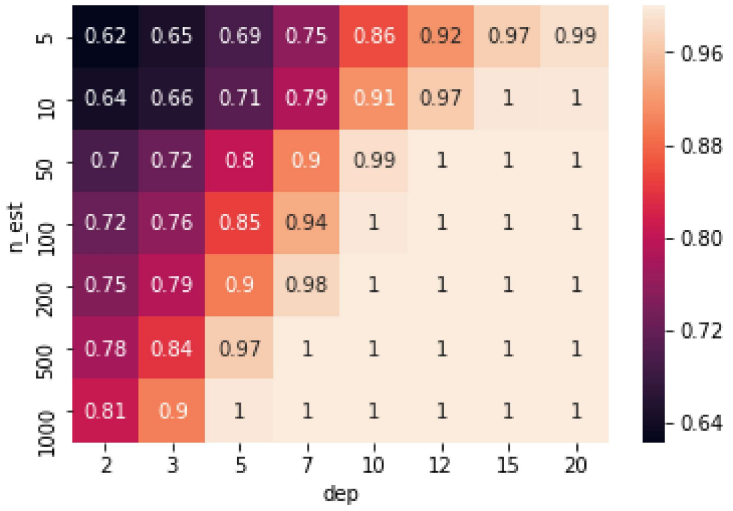
```
data1={'n_est':n_est,'dep':dep,'auc_cv':auc_cv[:],'auc_tr':auc_tr[:]}
data_f=pd.DataFrame(data1)
```

N_ESTIMATORS VS MAX_DEPTH

FOR TRAIN

In [305]:

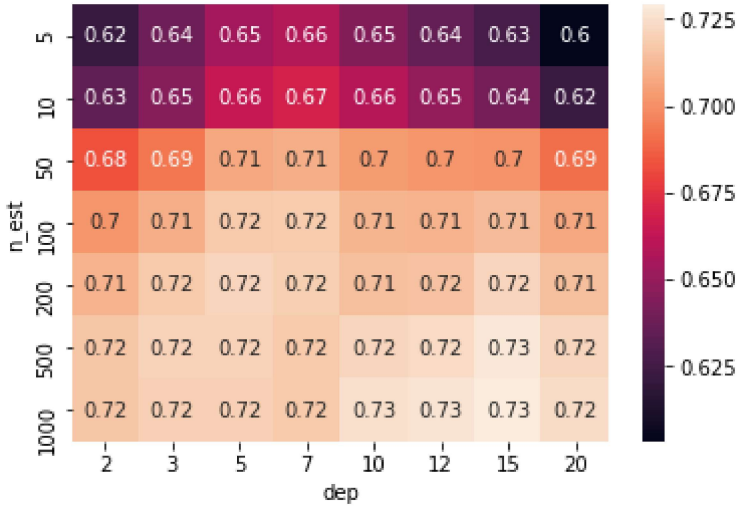
```
plo = data_f.pivot("n_est", "dep", "auc_tr");ax = sns.heatmap(plo, annot=True)
```



FOR CV

In [306]:

```
plo = data_f.pivot("n_est", "dep", "auc_cv");ax = sns.heatmap(plo, annot=True)
```



In [369]:

```
# TESTING THE MODEL ON TEST DATA

clf_final = gbc.XGBClassifier(n_estimators=100,max_depth=5,n_job=-1)
clf_final.fit(np.array(sent_vectors_tr), y_tr)
pred_final = clf_final.predict_proba(sent_vectors_test)[:,1]
fpr_f, tpr_f, thresholds_f = metrics.roc_curve(y_test, pred_final)
auc_final = metrics.auc(fpr_f, tpr_f)
auc_final
```

Out[369]:

```
0.7149328256171918
```

In [370]:

```
# TESTING THE MODEL ON TRAIN DATA

pred_train=clf_final.predict_proba(sent_vectors_tr)[:,1]
fpr_ft, tpr_ft, thresholds_ft=metrics.roc_curve(y_tr, pred_train)
auc_finalt=metrics.auc(fpr_ft, tpr_ft)
auc_finalt
```

Out[370]:

```
0.8491943786671132
```

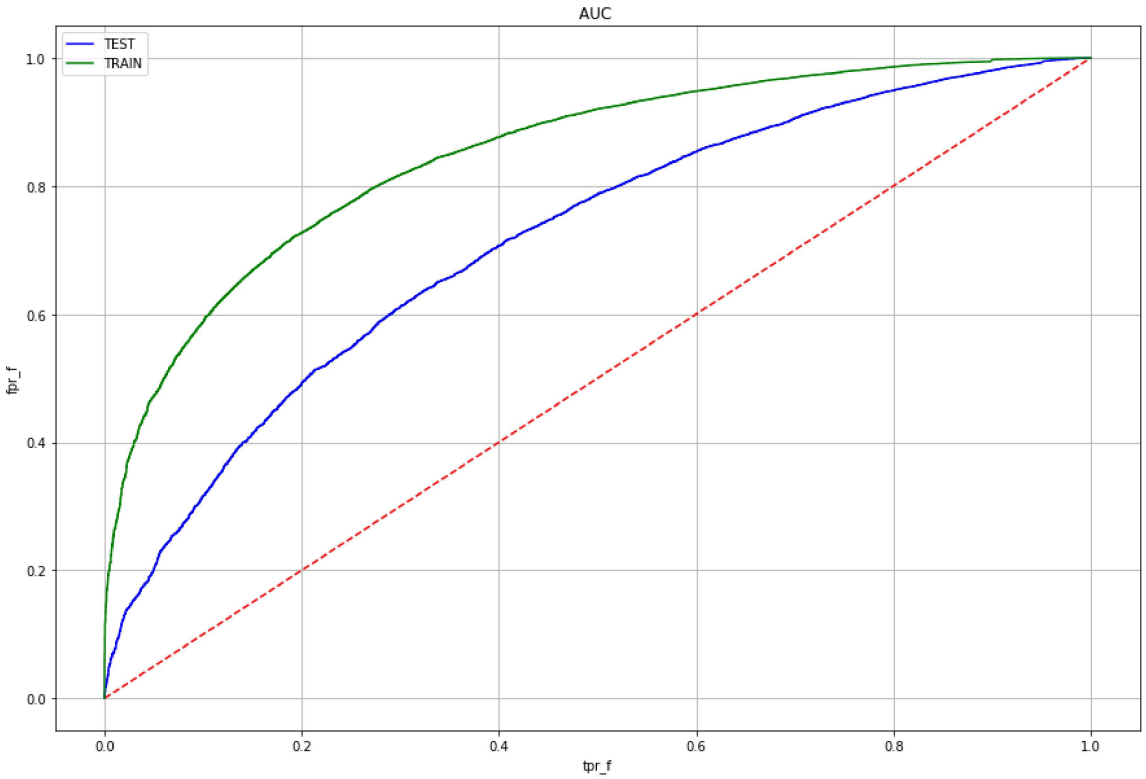
ROC CURVE

In [371]:

```
y_a=[0,0.5,1]
x_a=y_a
```

In [372]:

```
plt.figure(figsize=(15,10))
plt.plot(fpr_f,tpr_f,'b',label='TEST')
plt.plot(x_a,y_a,'--r')
plt.plot(fpr_ft,tpr_ft,'g',label='TRAIN')
plt.xlabel('tpr_f')
plt.ylabel('fpr_f')
plt.title('AUC ')
plt.grid()
plt.legend()
plt.show()
```



CONFUSION MATRIX

In [373]:

```
pred_final
pred_train
pred_final_classte=[]
pred_final_classtr=[]
for i in range(len(pred_final)):
    if pred_final[i]>0.5:
        pred_final_classte.append(1)
    elif pred_final[i]<=0.5:
            pred_final_classte.append(0)
for i in range(len(pred_train)):
    if pred_train[i]>0.5:
        pred_final_classtr.append(1)
    elif pred_train[i]<=0.5:
            pred_final_classtr.append(0)
```

TEST

In [374]:

```
cm = confusion_matrix(y_test, pred_final_classte)
cm
```

Out[374]:

```
array([[  113,  4444],
       [   48, 21727]], dtype=int64)
```

In [375]:

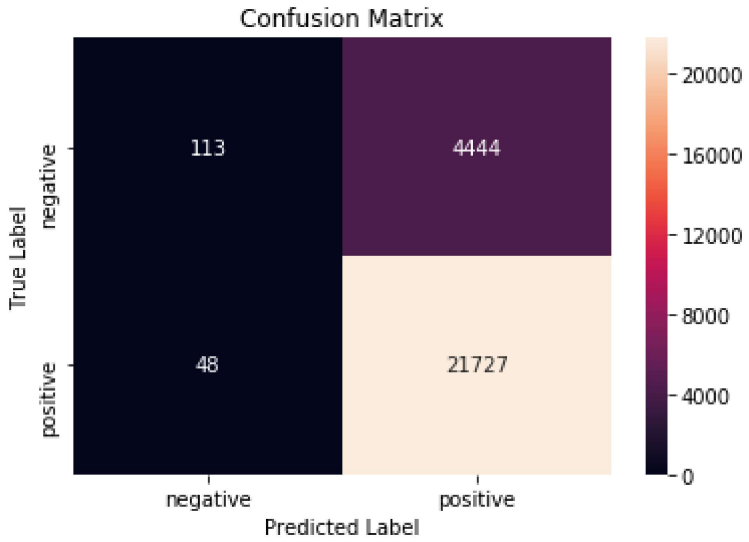
```
y_test["Score"].value_counts()
```

Out[375]:

```
1    21775
0     4557
Name: Score, dtype: int64
```

In [376]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



TRAIN

In [377]:

```
cm = confusion_matrix(y_tr, pred_final_classtr)
cm
```

Out[377]:

```
array([[  362,  5997],
       [   39, 36610]], dtype=int64)
```

In [378]:

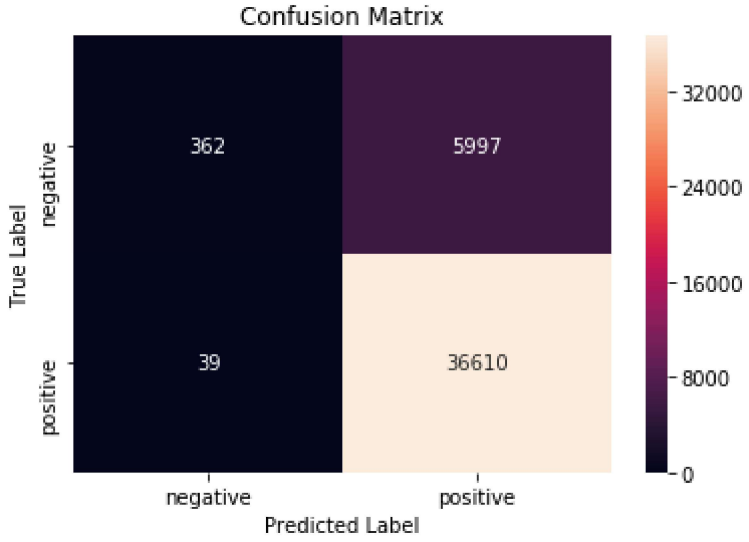
```
y_tr["Score"].value_counts()
```

Out[378]:

```
1    36649
0     6359
Name: Score, dtype: int64
```

In [379]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



CONCLUSION

In [380]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_final_classte))
```

```
              precision    recall  f1-score   support

           0       0.70      0.02      0.05      4557
           1       0.83      1.00      0.91     21775

   micro avg       0.83      0.83      0.83     26332
   macro avg       0.77      0.51      0.48     26332
weighted avg       0.81      0.83      0.76     26332
```

[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

In [50]:

```
auc_cv=[]
tpr_cv=[]
fpr_cv=[]
tpr_tr=[]
fpr_tr=[]
auc_tr=[]
data_rec=[]
data_rec=[]
hypermeter=[]
n_est = [5, 10, 50, 100, 200, 500, 1000]
max_dep = [2, 3, 5, 7,10, 12, 15, 20]
for i in n_est:
    for j in max_dep:
        clf=gbc.XGBClassifier(n_estimators=i,max_depth=j,n_jobs=-1)
        clf.fit(np.array(tfidf_sent_vectors),y_tr)
        pred = clf.predict_proba(tfidf_sent_vectors_cv)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, pred)
        auc=metrics.auc(fpr, tpr)
        auc_cv.append(auc)
        tpr_cv.append(tpr)
        fpr_cv.append(fpr)

        # predict the response on the train
        pred = clf.predict_proba(tfidf_sent_vectors)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
        auc=metrics.auc(fpr, tpr)
        auc_tr.append(auc)
        tpr_tr.append(tpr)
        fpr_tr.append(fpr)
        hypermeter.append([i,j])
        print(i,j)
```

```
5 2
5 3
5 5
5 7
5 10
5 12
5 15
5 20
10 2
10 3
10 5
10 7
10 10
10 12
10 15
10 20
50 2
50 3
50 5
50 7
50 10
50 12
50 15
50 20
100 2
100 3
100 5
100 7
100 10
100 12
100 15
100 20
200 2
200 3
200 5
200 7
200 10
200 12
200 15
200 20
500 2
500 3
500 5
500 7
500 10
500 12
500 15
500 20
1000 2
1000 3
1000 5
1000 7
1000 10
1000 12
1000 15
1000 20
```

In [51]:

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

In [52]:

```
dep=[]
n_est=[]
for i in range(len(hypermeter)):
    n_est.append(hypermeter[i][0])
    dep.append(hypermeter[i][1])
```

In [53]:

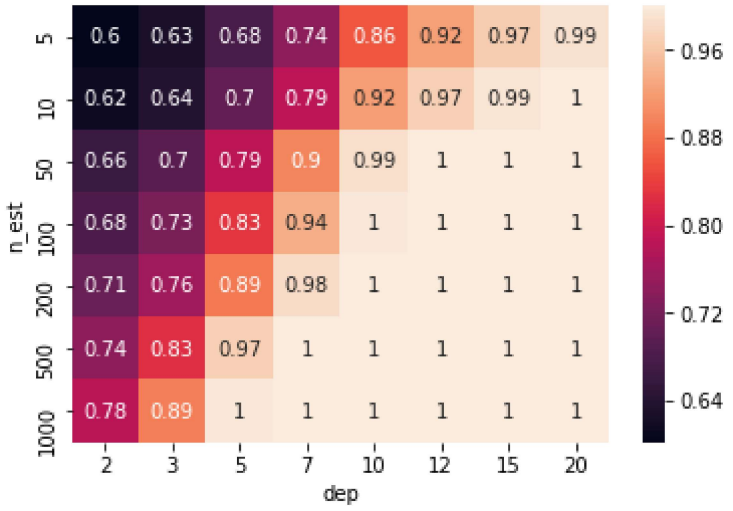
```
data1={'n_est':n_est,'dep':dep,'auc_cv':auc_cv[:],'auc_tr':auc_tr[:]}
data_f=pd.DataFrame(data1)
```

N_ESTIMATORS VS MAX_DEPTH

FOR TRAIN

In [54]:

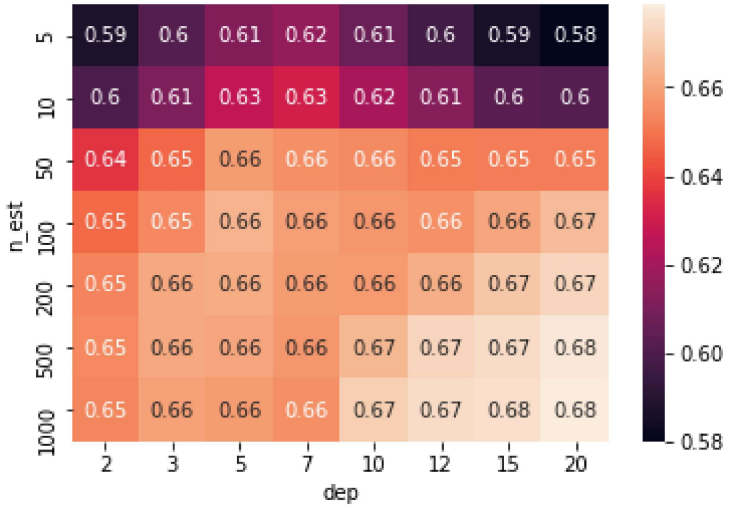
```
plo = data_f.pivot("n_est", "dep", "auc_tr");ax = sns.heatmap(plo, annot=True)
```



FOR CV

In [55]:

```
plo = data_f.pivot("n_est", "dep", "auc_cv");ax = sns.heatmap(plo, annot=True)
```



In [356]:

```
# TESTING THE MODEL ON TEST DATA

clf_final = gbc.XGBClassifier(n_estimators=100,max_depth=5,n_jobs=-1)
clf_final.fit(np.array(tfidf_sent_vectors), y_tr)
pred_final = clf_final.predict_proba(tfidf_sent_vectors_test)[:,1]
fpr_f, tpr_f, thresholds_f = metrics.roc_curve(y_test, pred_final)
auc_final = metrics.auc(fpr_f, tpr_f)
auc_final
```

Out[356]:

```
0.6532442008320679
```

In [357]:

```
# TESTING THE MODEL ON TRAIN DATA

pred_train=clf_final.predict_proba(tfidf_sent_vectors)[:,1]
fpr_ft, tpr_ft, thresholds_ft=metrics.roc_curve(y_tr, pred_train)
auc_finalt=metrics.auc(fpr_ft, tpr_ft)
auc_finalt
```

Out[357]:

```
0.8248312941093652
```

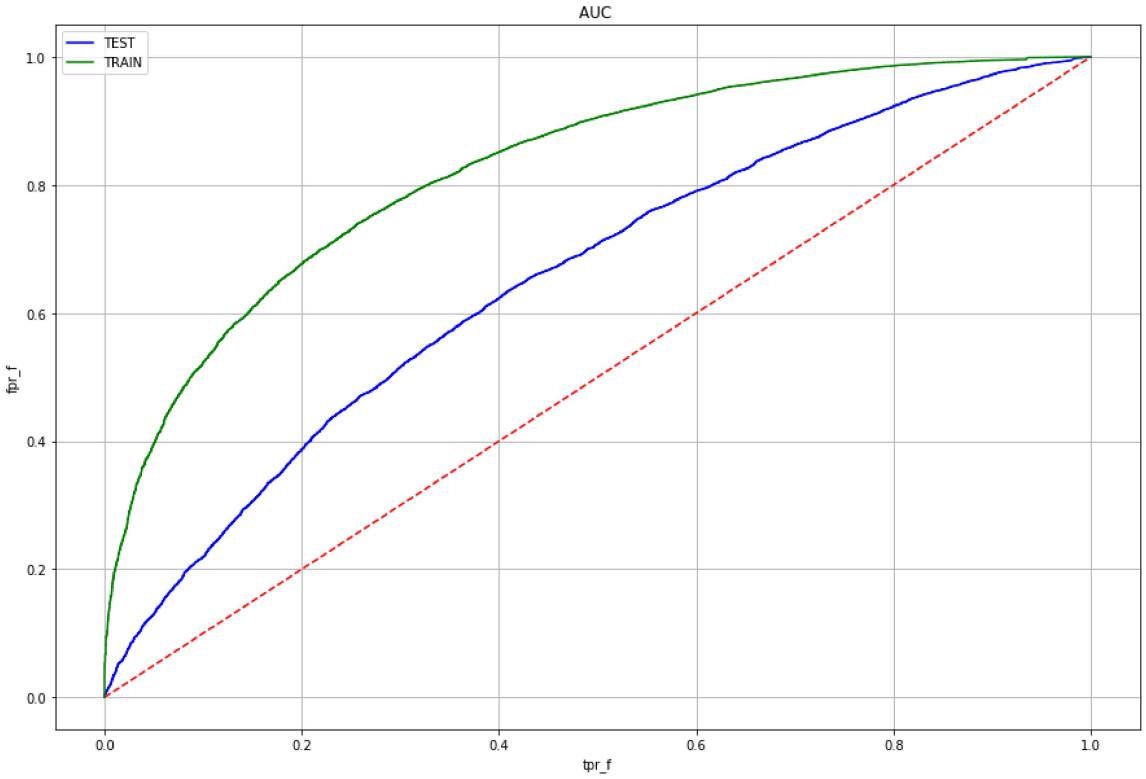
ROC CURVE

In [358]:

```
y_a=[0,0.5,1]
x_a=y_a
```

In [359]:

```
plt.figure(figsize=(15,10))
plt.plot(fpr_f,tpr_f,'b',label='TEST')
plt.plot(x_a,y_a,'--r')
plt.plot(fpr_ft,tpr_ft,'g',label='TRAIN')
plt.xlabel('tpr_f')
plt.ylabel('fpr_f')
plt.title('AUC ')
plt.grid()
plt.legend()
plt.show()
```



CONFUSION MATRIX

In [360]:

```
pred_final
pred_train
pred_final_classte=[]
pred_final_classtr=[]
for i in range(len(pred_final)):
    if pred_final[i]>0.5:
        pred_final_classte.append(1)
    elif pred_final[i]<=0.5:
            pred_final_classte.append(0)
for i in range(len(pred_train)):
    if pred_train[i]>0.5:
        pred_final_classtr.append(1)
    elif pred_train[i]<=0.5:
            pred_final_classtr.append(0)
```

TEST

In [361]:

```
cm = confusion_matrix(y_test, pred_final_classte)
cm
```

Out[361]:

```
array([[   38,  4519],
       [   20, 21755]], dtype=int64)
```

In [362]:

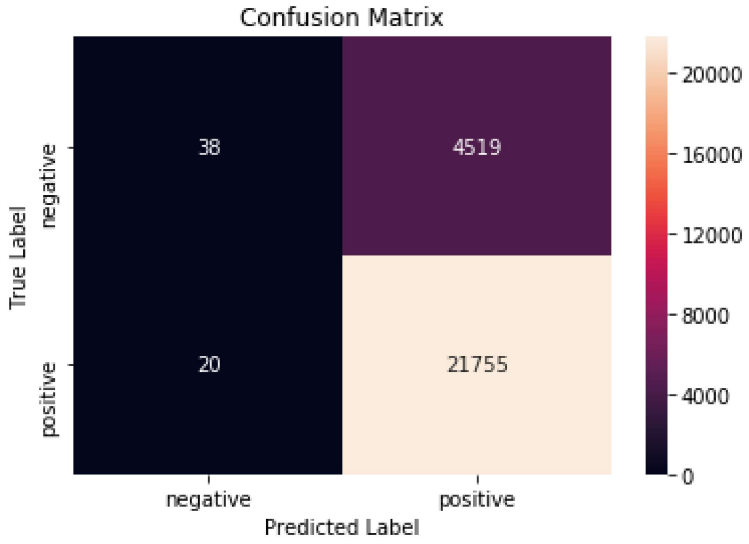
```
y_test["Score"].value_counts()
```

Out[362]:

```
1    21775
0     4557
Name: Score, dtype: int64
```

In [363]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



TRAIN

In [364]:

```
cm = confusion_matrix(y_tr, pred_final_classtr)
cm
```

Out[364]:

```
array([[  192,  6167],
       [   15, 36634]], dtype=int64)
```

In [365]:

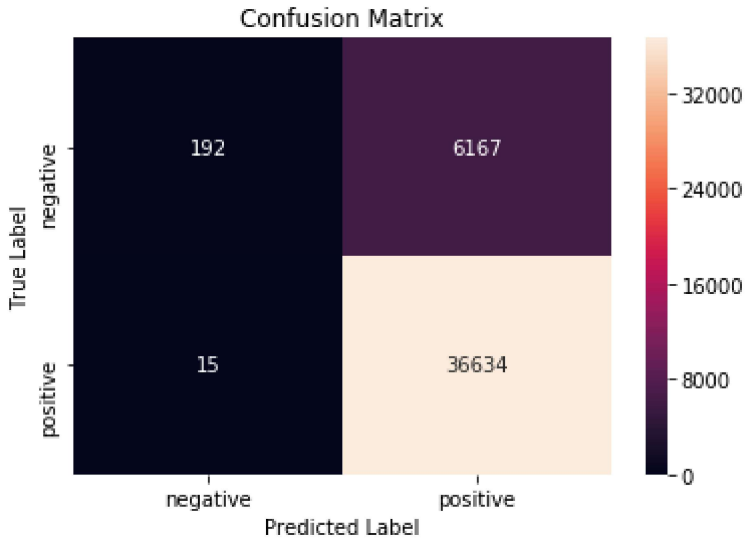
```
y_tr["Score"].value_counts()
```

Out[365]:

```
1    36649
0     6359
Name: Score, dtype: int64
```

In [366]:

```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



CONCLUSION

In [367]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_final_classte))
```

```
              precision    recall  f1-score   support

           0       0.66      0.01      0.02      4557
           1       0.83      1.00      0.91     21775

   micro avg       0.83      0.83      0.83     26332
   macro avg       0.74      0.50      0.46     26332
weighted avg       0.80      0.83      0.75     26332
```

[6] Conclusions

In [381]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["VECTORIZER",'ALGO','N_ESTIMATORS','MAX_DEPTH', "AUC"]
x.add_row(["BOW",'RANDOM FOREST',10,20,0.8321])
x.add_row(["TFIDF",'RANDOM FOREST',10,20,0.844251])
x.add_row(["AVG W2V",'RANDOM FOREST',1000,15,0.69325])
x.add_row(["TFIDF W2V",'RANDOM FOREST',500,10,0.6466])
x.add_row(["BOW",'GBDT',500,5,0.936899])
x.add_row(["TFIDF",'GBDT',500,5,0.94922])
x.add_row(["AVG W2V",'GBDT',100,5,0.714932])
x.add_row(["TFIDF W2V",'GBDT',100,5,0.653244])
```

In [382]:

```
print(x)
```

```
+------------+---------------+--------------+-----------+----------+
| VECTORIZER |      ALGO     | N_ESTIMATORS | MAX_DEPTH |   AUC    |
+------------+---------------+--------------+-----------+----------+
|    BOW     | RANDOM FOREST |      10      |     20    |  0.8321  |
|   TFIDF    | RANDOM FOREST |      10      |     20    | 0.844251 |
|  AVG W2V   | RANDOM FOREST |     1000     |     15    | 0.69325  |
| TFIDF W2V  | RANDOM FOREST |     500      |     10    |  0.6466  |
|    BOW     |      GBDT     |     500      |     5     | 0.936899 |
|   TFIDF    |      GBDT     |     500      |     5     | 0.94922  |
|  AVG W2V   |      GBDT     |     100      |     5     | 0.714932 |
| TFIDF W2V  |      GBDT     |     100      |     5     | 0.653244 |
+------------+---------------+--------------+-----------+----------+
```