Amazon Fine Food Reviews Analysis

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

EDA: 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
```

```python
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, roc_auc_score
from nltk.stem.porter import PorterStemmer

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

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

from tqdm import tqdm
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_validate
```

In [7]:

```python
# using SQLite Table to read data.
con = sqlite3.connect('/Users/puravshah/Downloads/amazon-fine-food-revi
ews/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 50
0000 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 Sco
re != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 LIMIT 50000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<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 (50000, 10)
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	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	COU

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

In [10]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[10]:

	UserId	ProductId	ProfileName	Time	Score	Text

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

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	Helpfuln

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

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

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

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

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

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

In [13]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

In [14]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[14]: (46072, 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]: 92.144

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	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

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()
```

```
(46071, 10)
```

Out[18]:
```
1    38479
0     7592
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [19]:
```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp
or cobbler. home made is better, but a heck of a lot more work. this is
great to have on hand for last minute dessert needs where you really wa
nt to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==================================================
For those of you wanting a high-quality, yet affordable green tea, you
should definitely give this one a try. Let me first start by saying tha
t everyone is looking for something different for their ideal tea, and
I will attempt to briefly highlight what makes this tea attractive to a
wide range of tea drinkers (whether you are a beginner or long-time tea
enthusiast).  I have gone through over 12 boxes of this tea myself, and
highly recommend it for the following reasons:<br /><br />-Quality:  Fi
rst, this tea offers a smooth quality without any harsh or bitter after
tones, which often turns people off from many green teas.  I've found m
y ideal brewing time to be between 3-5 minutes, giving you a light but
flavorful cup of tea.  However, if you get distracted or forget about y
```

```
our tea and leave it brewing for 20+ minutes like I sometimes do, the q
uality of this tea is such that you still get a smooth but deeper flavo
r without the bad after taste.  The leaves themselves are whole leaves
(not powdered stems, branches, etc commonly found in other brands), and
the high-quality nylon bags also include chunks of tropical fruit and o
ther discernible ingredients.  This isn't your standard cheap paper bag
with a mix of unknown ingredients that have been ground down to a fine
powder, leaving you to wonder what it is you are actually drinking.<br
/><br />-Taste:  This tea offers notes of real pineapple and other hint
s of tropical fruits, yet isn't sweet or artificially flavored.  You ha
ve the foundation of a high-quality young hyson green tea for those tru
e "tea flavor" lovers, yet the subtle hints of fruit make this a truly
unique tea that I believe most will enjoy.  If you want it sweet, you c
an add sugar, splenda, etc but this really is not necessary as this tea
offers an inherent warmth of flavor through it's ingredients.<br /><br
/>-Price:  This tea offers an excellent product at an exceptional price
(especially when purchased at the prices Amazon offers).  Compared to o
ther brands which I believe to be of similar quality (Mighty Leaf, Rish
i, Two Leaves, etc.), Revolution offers a superior product at an outsta
nding price.  I have been purchasing this through Amazon for less per b
ox than I would be paying at my local grocery store for Lipton, etc.<br
/><br />Overall, this is a wonderful tea that is comparable, and even b
etter than, other teas that are priced much higher.  It offers a well-b
alanced cup of green tea that I believe many will enjoy.  In terms of t
aste, quality, and price, I would argue you won't find a better combina
tion that that offered by Revolution's Tropical Green Tea.
==================================================
```

In [20]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
```

```
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [21]:

```python
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp
or cobbler. home made is better, but a heck of a lot more work. this is
great to have on hand for last minute dessert needs where you really wa
nt to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories. this one is a
```

great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==
For those of you wanting a high-quality, yet affordable green tea, you
should definitely give this one a try. Let me first start by saying tha
t everyone is looking for something different for their ideal tea, and
I will attempt to briefly highlight what makes this tea attractive to a
wide range of tea drinkers (whether you are a beginner or long-time tea
enthusiast). I have gone through over 12 boxes of this tea myself, and
highly recommend it for the following reasons:-Quality: First, this te
a offers a smooth quality without any harsh or bitter after tones, whic
h often turns people off from many green teas. I've found my ideal bre
wing time to be between 3-5 minutes, giving you a light but flavorful c
up of tea. However, if you get distracted or forget about your tea and
leave it brewing for 20+ minutes like I sometimes do, the quality of th
is tea is such that you still get a smooth but deeper flavor without th
e bad after taste. The leaves themselves are whole leaves (not powdere
d stems, branches, etc commonly found in other brands), and the high-qu
ality nylon bags also include chunks of tropical fruit and other discer
nible ingredients. This isn't your standard cheap paper bag with a mix
of unknown ingredients that have been ground down to a fine powder, lea
ving you to wonder what it is you are actually drinking.-Taste: This t
ea offers notes of real pineapple and other hints of tropical fruits, y
et isn't sweet or artificially flavored. You have the foundation of a
high-quality young hyson green tea for those true "tea flavor" lovers,
yet the subtle hints of fruit make this a truly unique tea that I belie
ve most will enjoy. If you want it sweet, you can add sugar, splenda,
etc but this really is not necessary as this tea offers an inherent war
mth of flavor through it's ingredients.-Price: This tea offers an exce
llent product at an exceptional price (especially when purchased at the
prices Amazon offers). Compared to other brands which I believe to be
of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), Revolution o
ffers a superior product at an outstanding price. I have been purchasi
ng this through Amazon for less per box than I would be paying at my lo
cal grocery store for Lipton, etc.Overall, this is a wonderful tea that
is comparable, and even better than, other teas that are priced much hi
gher. It offers a well-balanced cup of green tea that I believe many w
ill enjoy. In terms of taste, quality, and price, I would argue you wo

n't find a better combination that that offered by Revolution's Tropica l Green Tea.

In [22]:
```
# 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 [23]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==================================================
```

In [24]:
```
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
```

```
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [25]:

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

```
Great flavor low in calories high in nutrients high in protein Usually
protein powders are high priced and high in calories this one is a grea
t bargain and tastes great I highly recommend for the lady gym rats pro
bably not macho enough for guys since it is soy based
```

In [26]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <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', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
```

```
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [27]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 46071/46071 [00:13<00:00, 3344.73it/s]
```

In [28]:
```
preprocessed_reviews[1500]
```

Out[28]:
```
'great flavor low calories high nutrients high protein usually protein
powders high priced high calories one great bargain tastes great highly
recommend lady gym rats probably not macho enough guys since soy based'
```

[3.2] Preprocessing Review Summary

In [29]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [48]:
```
#BoW
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler(with_mean=False)
X_1, X_test, y_1, y_test = train_test_split(preprocessed_reviews, final
['Score'], test_size=0.3, random_state=0)
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3)
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(X_tr)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(X_tr)
final_counts=scaler.fit_transform(final_counts)
X_cv_bow=count_vect.transform(X_cv)
X_cv_bow=scaler.transform(X_cv_bow)
X_test_bow=count_vect.transform(X_test)
X_test_bow=scaler.fit_transform(X_test_bow)
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])
print(y_test.shape)
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaaaahhhhhh', 'aaaaaawwwww
wwwww', 'aadp', 'aafco', 'aah', 'aahhhs', 'aahing']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (22574, 28428)
the number of unique words  28428
(13822,)
```

[4.2] Bi-Grams and n-Grams.

In [49]:
```
#bi-gram, tri-gram and n-gram

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

count_vect_gram = CountVectorizer(ngram_range=(1,2), min_df=10, max_fea
tures=5000)
final_bigram_counts = count_vect_gram.fit_transform(X_tr)
final_bigram_counts=scaler.fit_transform(final_bigram_counts)
X_cv_ngram = count_vect_gram.transform(X_cv)
X_cv_ngram=scaler.fit_transform(X_cv_ngram)
X_test_ngram=count_vect_gram.transform(X_test)
X_test_ngram=scaler.fit_transform(X_test_ngram)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_s
hape())
print("the number of unique words including both unigrams and bigrams "
, final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (22574, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [50]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_tr)

print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
```

```python
print('='*50)

final_tf_idf = tf_idf_vect.transform(X_tr)
final_tf_idf=scaler.fit_transform(final_tf_idf)
X_cv_tfidf=tf_idf_vect.transform(X_cv)
X_cv_tfidf=scaler.fit_transform(X_cv_tfidf)
X_test_tfidf=tf_idf_vect.transform(X_test)
X_test_tfidf=scaler.fit_transform(X_test_tfidf)
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.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'a
ble buy', 'able drink', 'able eat', 'able enjoy', 'able find', 'able ge
t', 'able give', 'able make']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (22574, 13545)
the number of unique words including both unigrams and bigrams  13545
```

[4.4] Word2Vec

In [51]:
```python
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_train=[]
for sentance in X_tr:
    list_of_sentance_train.append(sentance.split())
```

In [52]:
```python
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
```

In [53]:
```python
i=0
list_of_sentance_test=[]
```

```
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
```

In [54]:
```
# 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 val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# 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_train=Word2Vec(list_of_sentance_train,min_count=5,size=50
, workers=4)
    print(w2v_model_train.wv.most_similar('great'))
    print('='*50)
    print(w2v_model_train.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.bin', 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_trai
n_w2v = True, to train your own w2v ")
```

```
[('awesome', 0.82865309715271), ('good', 0.7961898446083069), ('fantast
ic', 0.7938987016677856), ('excellent', 0.7852526903152466), ('wonderfu
l', 0.7739599347114563), ('amazing', 0.7680932879447937), ('perfect',
0.7604414224624634), ('decent', 0.70396888256073), ('satisfied', 0.6891
260147094727), ('outstanding', 0.6524614691734314)]
==================================================
[('ever', 0.7886314988136292), ('ive', 0.7581040263175964), ('awful',
0.7485695481300354), ('nastiest', 0.7350848913192749), ('closest', 0.73
43916893005371), ('compares', 0.7272354960441589), ('horrible', 0.72253
3106803894), ('tastiest', 0.7164811491966248), ('okay', 0.7122682929039
001), ('tasted', 0.7083225846290588)]
```

In [ ]:

```
''''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, work
ers=4)
    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.bin', 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_trai
n_w2v = True, to train your own w2v ")'''
```

In [ ]:
```
'''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=4)
    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.bin', 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_trai
n_w2v = True, to train your own w2v ")'''
```

In [55]:
```
w2v_words_train = list(w2v_model_train.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_tra
in))
print("sample words ", w2v_words_train[0:50])
```

```
number of words that occured minimum 5 times  9134
sample words  ['not', 'much', 'say', 'fast', 'flavored', 'oatmeal', 'ta
stes', 'good', 'really', 'easy', 'fix', 'couple', 'minutes', 'reasonabl
y', 'like', 'stuff', 'always', 'big', 'tea', 'fan', 'never', 'ginger',
'honey', 'tried', 'one', 'glad', 'great', 'kind', 'sip', 'reading', 'bo
ok', 'overpowering', 'teas', 'sometimes', 'sweet', 'either', 'give', 's
hot', 'tasty', 'find', 'interesting', 'see', 'wide', 'range', 'comment
s', 'product', 'particularly', 'food', 'drink', 'taste']
```

In [ ]:
```
'''w2v_words_cv = list(w2v_model_cv.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_c
```

```
v))
print("sample words ", w2v_words_cv[0:50])'''
```

In [ ]:

```
''''w2v_words_test = list(w2v_model_test.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_tes
t))
print("sample words ", w2v_words_test[0:50])'''
```

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

[4.4.1.1] Avg W2v

In [ ]:

```
# 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, yo
u 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/re
view
    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]))'''
```

In [56]:

```
sent_vectors_train = []; # the avg-w2v for each sentence/review is stor
ed in this list
```

```
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
print(len(sent_vectors_train))
print(len(sent_vectors_train[0]))
sent_vectors_train=scaler.fit_transform(sent_vectors_train)
```

```
100%|██████████| 22574/22574 [00:20<00:00, 1078.90it/s]
```

```
22574
50
```

In [57]:

```
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_cv): # 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_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_cv.append(sent_vec)
print(len(sent_vectors_cv))
```

```
print(len(sent_vectors_cv[0]))
sent_vectors_cv=scaler.fit_transform(sent_vectors_cv)
```

```
100%|██████████| 9675/9675 [00:08<00:00, 1117.01it/s]
```

```
9675
50
```

In [58]:

```
sent_vectors_test = []; # the avg-w2v for each sentence/review is store
d in this list
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u 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/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
sent_vectors_test=scaler.fit_transform(sent_vectors_test)
```

```
100%|██████████| 13822/13822 [00:12<00:00, 1111.76it/s]
```

```
13822
50
```

[4.4.1.2] TFIDF weighted W2v

In [59]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model_train= TfidfVectorizer()
tf_idf_matrix_train = model_train.fit_transform(X_tr)
```

```
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary_train = dict(zip(model_train.get_feature_names(), list(model
_train.idf_)))
```

In [60]:
```
'''# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model_cv= TfidfVectorizer()
tf_idf_matrix_cv = model_cv.fit_transform(X_cv)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary_cv= dict(zip(model_cv.get_feature_names(), list(model_cv.idf
_)))'''
```

Out[60]:
```
'\'# S = ["abc def pqr", "def def def abc", "pqr pqr def"]\nmodel_cv= T
fidfVectorizer()\ntf_idf_matrix_cv = model_cv.fit_transform(X_cv)\n# we
are converting a dictionary with word as a key, and the idf as a value
\ndictionary_cv= dict(zip(model_cv.get_feature_names(), list(model_cv.i
df_)))'
```

In [ ]:
```
'''# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model_test= TfidfVectorizer()
tf_idf_matrix_test = model_test.fit_transform(X_test)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary_test= dict(zip(model_test.get_feature_names(), list(model_te
st.idf_)))'''
```

In [61]:
```
# TF-IDF weighted Word2Vec
tfidf_feat_train = model_train.get_feature_names() # tfidf words/col-na
mes
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec_train = np.zeros(50) # as word vectors are of zero length
```

```
    weight_sum_train =0; # num of words with a valid vector in the sent
ence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train and word in tfidf_feat_train:
            vec_train = w2v_model_train.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_train[word]*(sent.count(word)/len(sent))
            sent_vec_train += (vec * tf_idf)
            weight_sum_train += tf_idf
    if weight_sum_train != 0:
        sent_vec_train /= weight_sum_train
    tfidf_sent_vectors_train.append(sent_vec_train)
    row += 1
tfidf_sent_vectors_train =scaler.fit_transform(tfidf_sent_vectors_train
)
```

```
100%|██████████| 22574/22574 [02:53<00:00, 130.23it/s]
```

In [63]:

```
tfidf_feat_cv = model_train.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is
 stored in this list
row=0;
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec_cv = np.zeros(50) # as word vectors are of zero length
    weight_sum_cv =0; # num of words with a valid vector in the sentenc
e/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train and word in tfidf_feat_cv:
            vec_cv = w2v_model_train.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_train[word]*(sent.count(word)/len(sent
))
            sent_vec_cv += (vec * tf_idf)
            weight_sum_cv += tf_idf
    if weight_sum_cv != 0:
        sent_vec_cv /= weight_sum_cv
    tfidf_sent_vectors_cv.append(sent_vec_cv)
    row += 1
tfidf_sent_vectors_cv =scaler.fit_transform(tfidf_sent_vectors_cv)
```

```
100%|██████████| 9675/9675 [01:11<00:00, 134.76it/s]
```

In [64]:

```
tfidf_feat_test = model_train.get_feature_names() # tfidf words/col-nam
es
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec_test = np.zeros(50) # as word vectors are of zero length
    weight_sum_test =0; # num of words with a valid vector in the sente
nce/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train and word in tfidf_feat_test:
            vec_test = w2v_model_train.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_train[word]*(sent.count(word)/len(sent
))
            sent_vec_test += (vec * tf_idf)
            weight_sum_test += tf_idf
    if weight_sum_test != 0:
        sent_vec_test /= weight_sum_test
    tfidf_sent_vectors_test.append(sent_vec_test)
```

```
    row += 1
tfidf_sent_vectors_test =scaler.fit_transform(tfidf_sent_vectors_test)
```

```
100%|██████████| 13822/13822 [01:42<00:00, 134.83it/s]
```

[5] Assignment 3: KNN

1. **Apply Knn(brute force version) 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. **Apply Knn(kd tree version) on these feature sets**
    NOTE: sklearn implementation of kd-tree accepts only dense matrices, you need to convert the sparse matrices of CountVectorizer/TfidfVectorizer into dense matices. You can convert sparse matrices to dense using .toarray() attribute. For more information please visit this [link]

    - SET 5:Review text, preprocessed one converted into vectors using (BOW) but with restriction on maximum features generated.

        ```
                    count_vect = CountVectorizer(min_df=10, 
        max_features=500)
                    count_vect.fit(preprocessed_reviews)
        ```

    - SET 6:Review text, preprocessed one converted into vectors using (TFIDF) but with restriction on maximum features generated.

        ```
                        tf_idf_vect = TfidfVectorizer(min_d
        f=10, max_features=500)
        ```

```
tf_idf_vect.fit(preprocessed_review
s)
```

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

3. **The hyper paramter tuning(find best K)**
   - Find the best hyper parameter which will give the maximum AUC 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

4. **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
   - 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 with predicted and original labels of test data points

5. **Conclusion**
   - 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 link

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.

[5.1] Applying KNN brute force

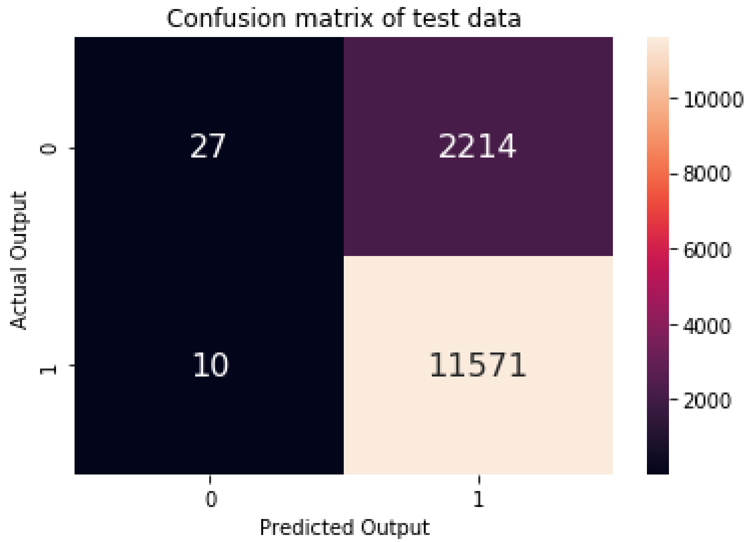
[5.1.1] Applying KNN brute force on BOW, SET 1

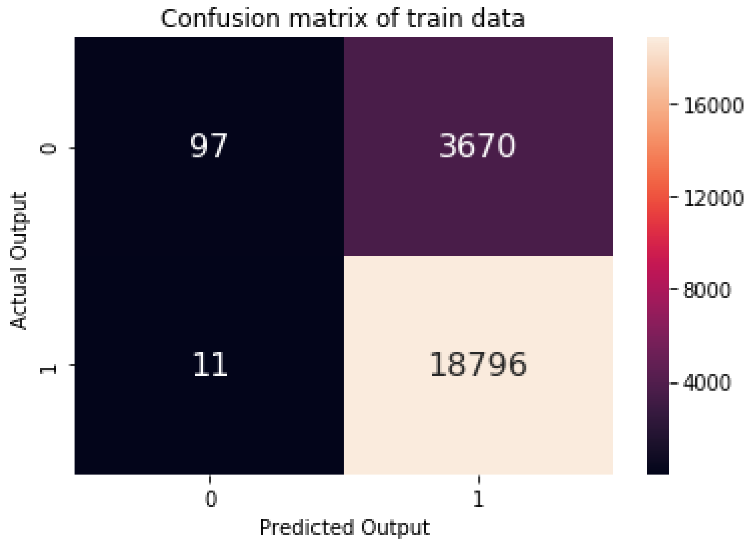
In [69]:
```
# Please write all the code with proper documentation
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
neighbors=list(range(1,30,2))
tuned_parameters=[{'n_neighbors':neighbors}]
model=GridSearchCV(KNeighborsClassifier(),tuned_parameters,scoring='roc
_auc',cv=10)
model.fit(final_counts,y_tr)
print(model.best_estimator_)
optimal_neighbors=model.best_estimator_.n_neighbors

print(model.score(X_cv_bow,y_cv))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                     metric_params=None, n_jobs=None, n_neighbors=9, p=
2,
                     weights='uniform')
0.6422105197944094
```

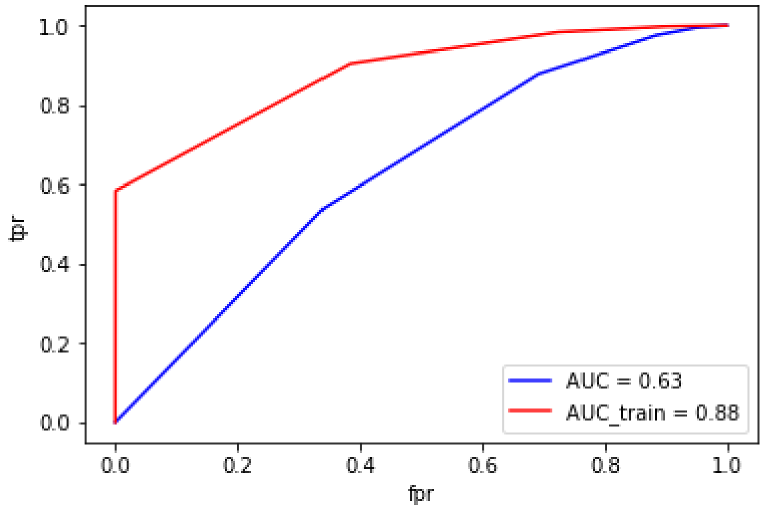
In [74]:
```
model_new=KNeighborsClassifier(n_neighbors=optimal_neighbors,algorithm=
'brute')
model_new.fit(final_counts,y_tr)
pred=model_new.predict(X_test_bow)
```

```
probab=model_new.predict_proba(X_test_bow)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
pred2=model_new.predict(final_counts)
probab2=model_new.predict_proba(final_counts)[:,1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of test data')
plt.show()

df_cm2 = pd.DataFrame(confusion_matrix(y_tr, pred2), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm2, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of train data')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```






```
Out[74]: Text(0,0.5,'tpr')
```



In [ ]:
```
'''# ============================== KNN with k = optimal_k ===========
====================================
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='bru
te')

# fitting the model
knn_optimal.fit(final_counts, y_tr)

# predict the response
pred = knn_optimal.predict(X_test_bow)
probabilities=knn_optimal.predict_proba(X_test_bow)
print(probabilities.shape)
probabilities=probabilities[:,1]
pred2=knn_optimal.predict(final_counts)
probab=knn_optimal.predict_proba(final_counts)
probab=probab[:,1]
```

```
# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k, acc))'''
```

In [ ]:

```
''''from sklearn.metrics import confusion_matrix as cm
from sklearn.metrics import roc_auc_score
print(y_tr.shape)
print(final_counts.shape)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
fpr, tpr, thresholds = metrics.roc_curve(y_test, probabilities)

plt.plot([0,1],[0,1],'k--')

auc = roc_auc_score(y_test, probabilities)
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab)
auc2=roc_auc_score(y_tr,probab)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train, tpr_train, 'r', label = 'AUC = %0.2f' %auc)
plt.legend(loc = 'lower right')
plt.title('Roc curve with optimal number of neighbors and also the AUC
 value for optimal number of neighbors displayed')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
print('AUC: %.2f' % auc)

#print(neighbors.shape)
#plt.plot(neighbors,auc1)'''
```

[5.1.2] Applying KNN brute force on TFIDF, SET 2

In [75]:

```
neighbors=list(range(1,30,2))
tuned_parameters=[{'n_neighbors':neighbors}]
```

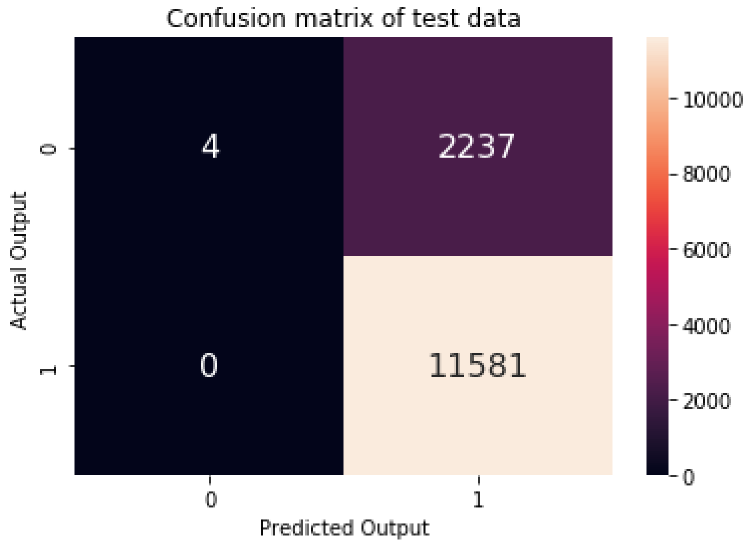
```
model=GridSearchCV(KNeighborsClassifier(),tuned_parameters,scoring='roc
_auc',cv=10)
model.fit(final_tf_idf,y_tr)
print(model.best_estimator_)
optimal_neighbors=model.best_estimator_.n_neighbors

print(model.score(X_cv_tfidf,y_cv))
```

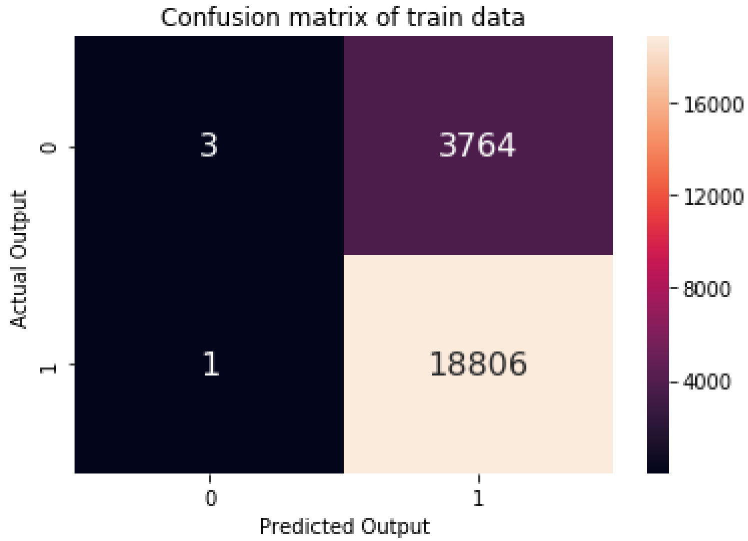
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=
2,
                     weights='uniform')
0.5106738813171886
```

In [76]:

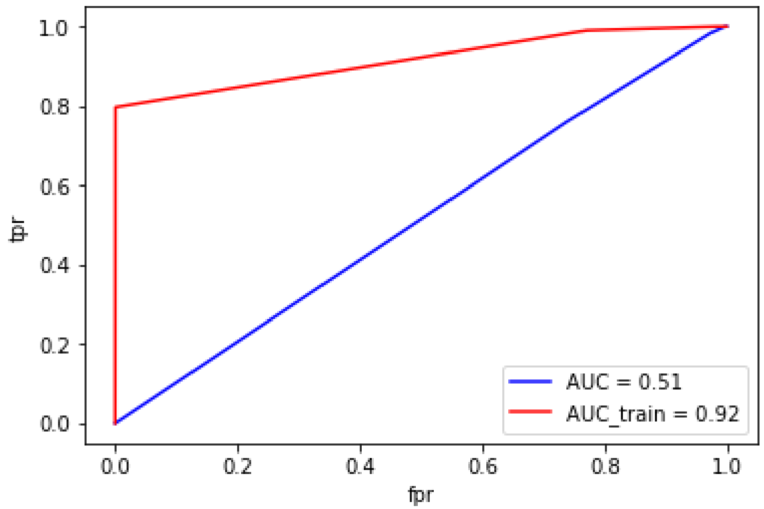
```
model_new=KNeighborsClassifier(n_neighbors=optimal_neighbors,algorithm=
'brute')
model_new.fit(final_tf_idf,y_tr)
pred=model_new.predict(X_test_tfidf)
probab=model_new.predict_proba(X_test_tfidf)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
pred2=model_new.predict(final_tf_idf)
probab2=model_new.predict_proba(final_tf_idf)[:,1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of test data')
plt.show()

df_cm2 = pd.DataFrame(confusion_matrix(y_tr, pred2), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm2, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
```

```
plt.ylabel('Actual Output')
plt.title('Confusion matrix of train data')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```





Out[76]: Text(0,0.5,'tpr')



In [ ]:
```
''''# ============================== KNN with k = optimal_k ===========
====================================
```

```
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='bru
te')

# fitting the model
knn_optimal.fit(final_tf_idf, y_tr)

# predict the response
pred = knn_optimal.predict(X_test_tfidf)
probabs=knn_optimal.predict_proba(X_test_tfidf)
probabs=probabs[:,1]
pred2=knn_optimal.predict(final_tf_idf)
probab=knn_optimal.predict_proba(final_tf_idf)
probab=probab[:,1]
# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k, acc))'''
```

In [ ]:

```
''''#from sklearn.metrics import confusion_matrix as cm
#from sklearn.metrics import roc_auc_score
#print(y_tr.shape)
#print(final_counts.shape)
df_cm_tfidf = pd.DataFrame(confusion_matrix(y_test, pred), range(2),ran
ge(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm_tfidf, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
fpr, tpr, thresholds = metrics.roc_curve(y_test, probabs)
#fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
plt.plot([0,1],[0,1],'k--')

auc_tfidf = roc_auc_score(y_test, probabs)
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab)
auc2=roc_auc_score(y_tr,probab)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %auc_tfidf)
plt.plot(fpr_train, tpr_train, 'r', label = 'AUC = %0.2f' %auc2)
plt.legend(loc = 'lower right')
plt.title('Roc curve with optimal number of neighbors and also the AUC
```

```
 value for optimal number of neighbors displayed with TFIDF')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
print('AUC: %.2f' % auc_tfidf)'''
```

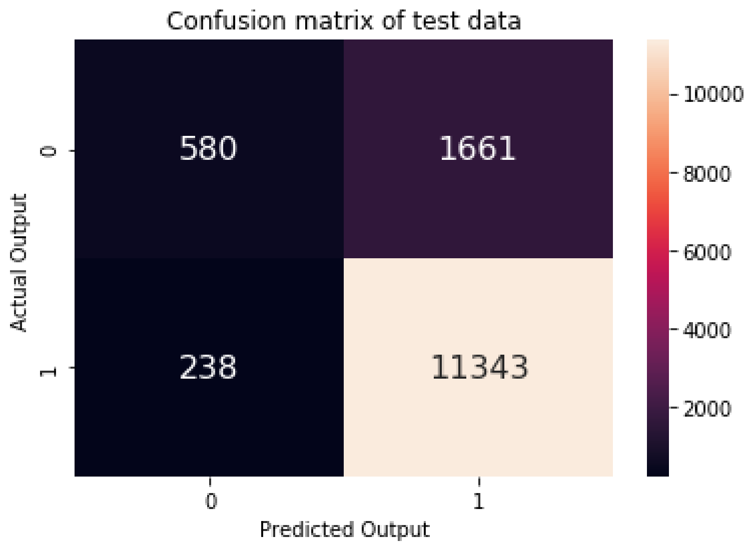
[5.1.3] Applying KNN brute force on AVG W2V, SET 3

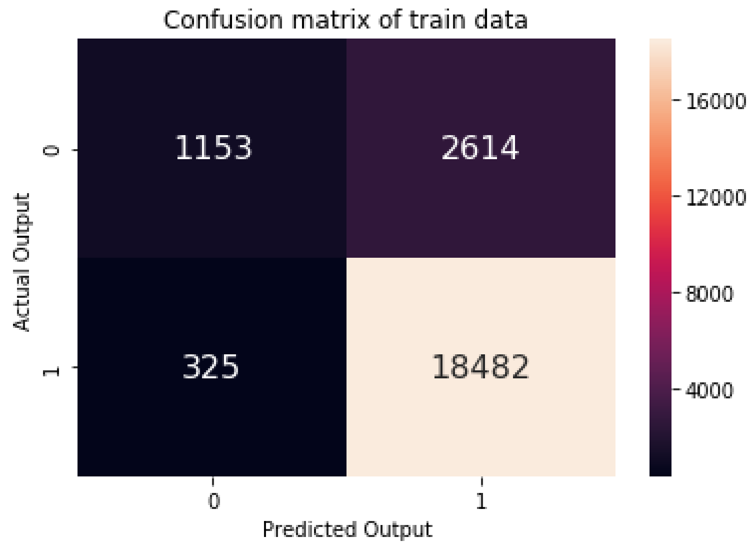
In [77]:
```
neighbors=list(range(1,30,2))
tuned_parameters=[{'n_neighbors':neighbors}]
model=GridSearchCV(KNeighborsClassifier(),tuned_parameters,scoring='roc
_auc',cv=10)
model.fit(sent_vectors_train,y_tr)
print(model.best_estimator_)
optimal_neighbors=model.best_estimator_.n_neighbors

print(model.score(sent_vectors_cv,y_cv))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                     metric_params=None, n_jobs=None, n_neighbors=29, p
=2,
                     weights='uniform')
0.8637954598512625
```

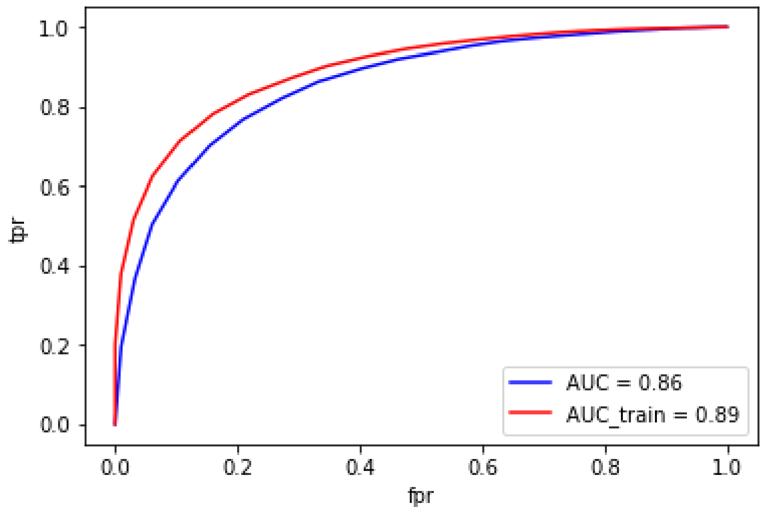
In [78]:
```
model_new=KNeighborsClassifier(n_neighbors=optimal_neighbors,algorithm=
'brute')
model_new.fit(sent_vectors_train,y_tr)
pred=model_new.predict(sent_vectors_test)
probab=model_new.predict_proba(sent_vectors_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
pred2=model_new.predict(sent_vectors_train)
probab2=model_new.predict_proba(sent_vectors_train)[:,1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
```

```
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of test data')
plt.show()

df_cm2 = pd.DataFrame(confusion_matrix(y_tr, pred2), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm2, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of train data')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```






```
Out[78]: Text(0,0.5,'tpr')
```



In [ ]:
```
'''model_optimal=KNeighborsClassifier(n_neighbors=optimalk,algorithm
='brute')
model_optimal.fit(sent_vectors_train,y_tr)
pred=model_optimal.predict(sent_vectors_test)
probab_w2v=model_optimal.predict_proba(sent_vectors_test)
probab_w2v=probab_w2v[:,1]
pred2=model_optimal.predict(sent_vectors_train)
probab=model_optimal.predict_proba(sent_vectors_train)
probab=probab[:,1]
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
print('The accuracy of the model with optimal k %d is=%f'%(optimalk,ac
c))'''
```

In [ ]:
```
'''df_cm_w2v = pd.DataFrame(confusion_matrix(y_test, pred), range(2),r
ange(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm_w2v, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
fpr, tpr, thresholds = metrics.roc_curve(y_test, probab_w2v)
#fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
plt.plot([0,1],[0,1],'k--')

auc_w2v = roc_auc_score(y_test, probab_w2v)
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab)
auc2=roc_auc_score(y_tr,probab)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %auc_w2v)
plt.plot(fpr_train, tpr_train, 'r', label = 'AUC = %0.2f' %auc2)
plt.legend(loc = 'lower right')
plt.title('Roc curve with optimal number of neighbors and also the AUC
 value for optimal number of neighbors displayed with average w2v')
```

```
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
print('AUC: %.2f' % auc_w2v)'''
```

[5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

In [79]:

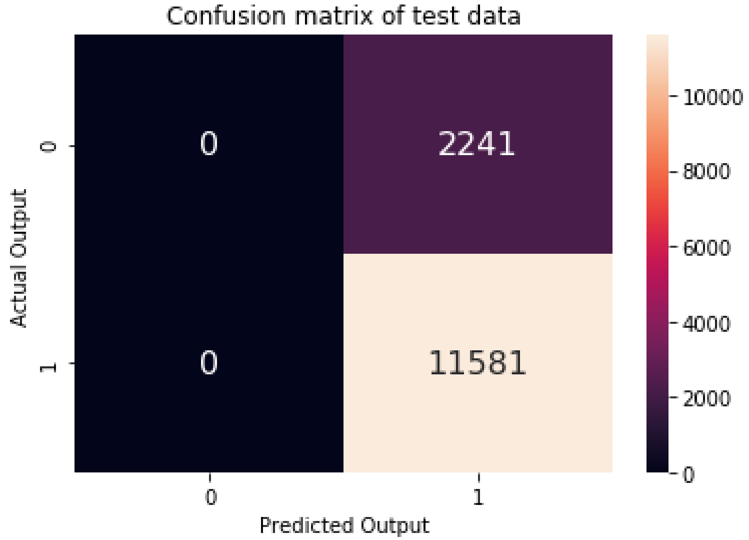
```
neighbors=list(range(1,30,2))
tuned_parameters=[{'n_neighbors':neighbors}]
model=GridSearchCV(KNeighborsClassifier(),tuned_parameters,scoring='roc
_auc',cv=10)
model.fit(tfidf_sent_vectors_train,y_tr)
print(model.best_estimator_)
optimal_neighbors=model.best_estimator_.n_neighbors

print(model.score(tfidf_sent_vectors_cv,y_cv))
```

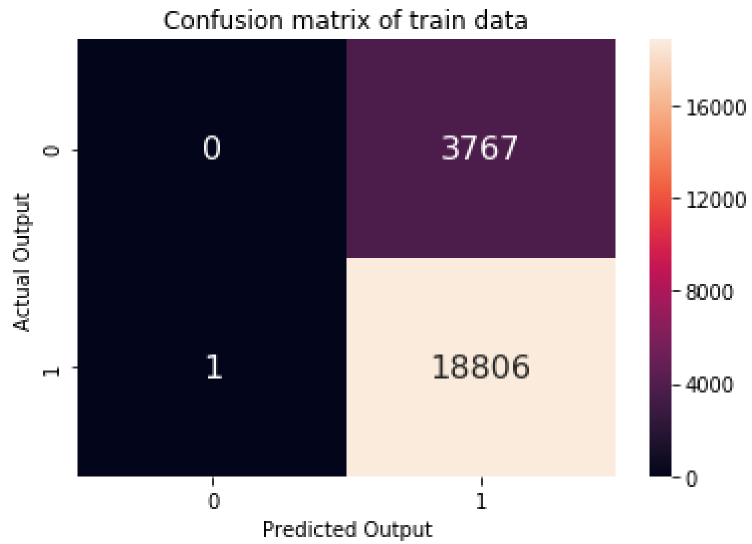
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                     metric_params=None, n_jobs=None, n_neighbors=29, p
=2,
                     weights='uniform')
0.5007900192132673
```

In [80]:

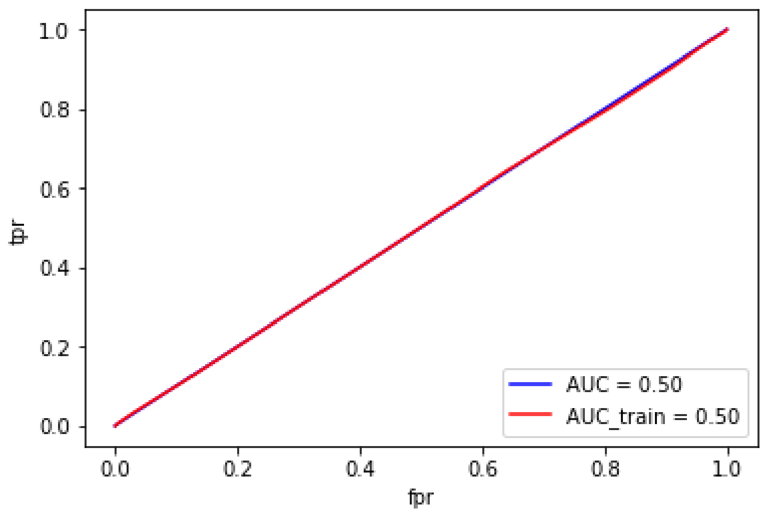
```
model_new=KNeighborsClassifier(n_neighbors=optimal_neighbors,algorithm=
'brute')
model_new.fit(tfidf_sent_vectors_train,y_tr)
pred=model_new.predict(tfidf_sent_vectors_test)
probab=model_new.predict_proba(tfidf_sent_vectors_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
pred2=model_new.predict(tfidf_sent_vectors_train)
probab2=model_new.predict_proba(tfidf_sent_vectors_train)[:,1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
```

```
    #heatman for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of test data')
plt.show()

df_cm2 = pd.DataFrame(confusion_matrix(y_tr, pred2), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm2, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of train data')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```





Out[80]: Text(0,0.5,'tpr')



In [ ]:
```
'''''model_optimal=KNeighborsClassifier(n_neighbors=optimalk,algorithm
='brute')
model_optimal.fit(tfidf_sent_vectors_train,y_tr)
```

```
pred=model_optimal.predict(tfidf_sent_vectors_test)
#print(pred.shape)
probab_w2v=model_optimal.predict_proba(tfidf_sent_vectors_test)
probab_w2v=probab_w2v[:,1]
pred2=model_optimal.predict(tfidf_sent_vectors_train)
probab=model_optimal.predict_proba(tfidf_sent_vectors_train)
probab=probab[:,1]
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
print('The accuracy of the model with optimal k %d is=%f'%(optimalk,ac
c))'''
```

In [ ]:

```
'''''df_cm_w2v = pd.DataFrame(confusion_matrix(y_test, pred), range(2),
range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm_w2v, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
fpr, tpr, thresholds = metrics.roc_curve(y_test, probab_w2v)
#fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
plt.plot([0,1],[0,1],'k--')

auc_w2v = roc_auc_score(y_test, probab_w2v)
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab)
auc2=roc_auc_score(y_tr,probab)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %auc_w2v)
plt.plot(fpr_train, tpr_train, 'r', label = 'AUC = %0.2f' %auc2)
plt.legend(loc = 'lower right')
plt.title('Roc curve with optimal number of neighbors and also the AUC
 value for optimal number of neighbors displayed with weighted TFIDF')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
print('AUC: %.2f' % auc_w2v)'''
```

[5.2] Applying KNN kd-tree

[5.2.1] Applying KNN kd-tree on BOW, SET 5

In [81]:

```
# Please write all the code with proper documentation
count_vect = CountVectorizer(min_df=10, max_features=50)
count_vect.fit(X_tr)

print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(X_tr)
final_counts=scaler.fit_transform(final_counts)
X_cv_bow=count_vect.transform(X_cv)
X_cv_bow=scaler.fit_transform(X_cv_bow)
X_test_bow=count_vect.transform(X_test)
X_test_bow=scaler.fit_transform(X_test_bow)
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])
print(y_test.shape)
```

```
some feature names  ['also', 'amazon', 'bag', 'best', 'better', 'bough
t', 'buy', 'chocolate', 'coffee', 'could']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (22574, 50)
the number of unique words  50
(13822,)
```

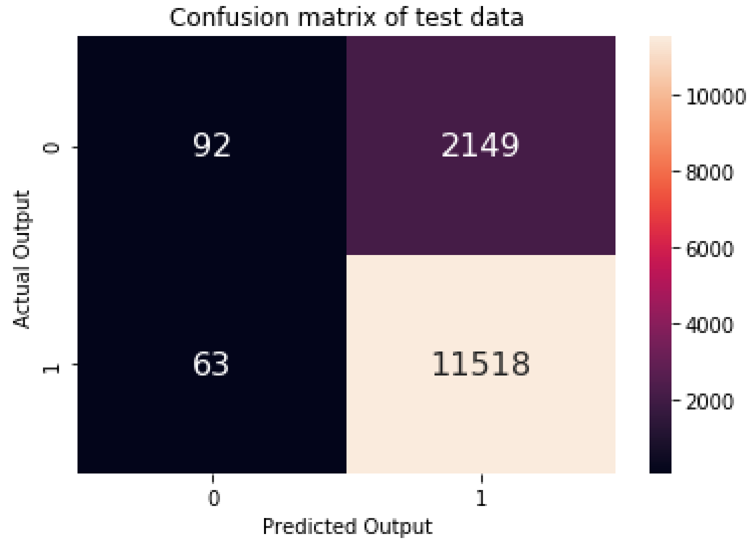
In [82]:

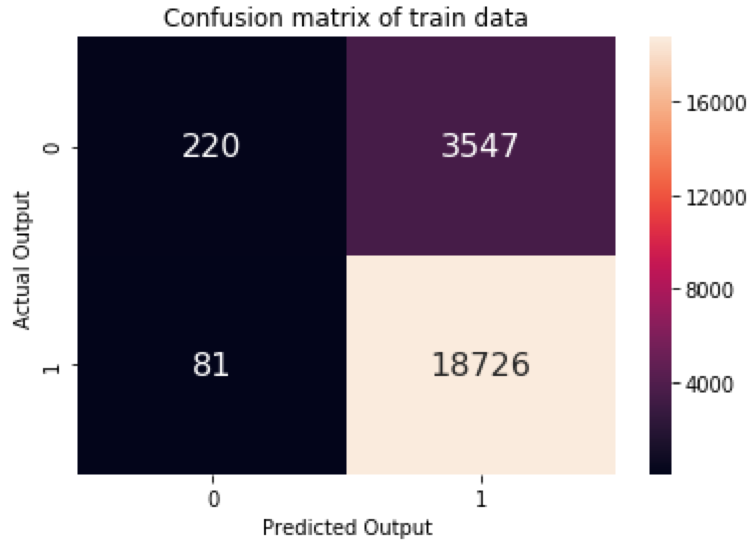
```
neighbors=list(range(1,30,2))
tuned_parameters=[{'n_neighbors':neighbors}]
model=GridSearchCV(KNeighborsClassifier(),tuned_parameters,scoring='roc
_auc',cv=10)
model.fit(final_counts,y_tr)
print(model.best_estimator_)
optimal_neighbors=model.best_estimator_.n_neighbors

print(model.score(X_cv_bow,y_cv))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                     metric_params=None, n_jobs=None, n_neighbors=29, p
```

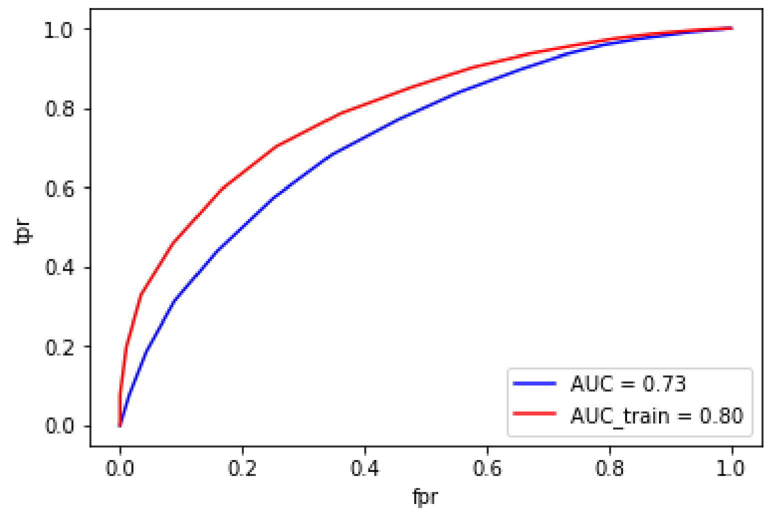
```
=2,
                     weights='uniform')
0.7404043291024196
```

In [83]:
```
model_new=KNeighborsClassifier(n_neighbors=optimal_neighbors,algorithm=
'kd_tree')
model_new.fit(final_counts,y_tr)
pred=model_new.predict(X_test_bow)
probab=model_new.predict_proba(X_test_bow)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
pred2=model_new.predict(final_counts)
probab2=model_new.predict_proba(final_counts)[:,1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of test data')
plt.show()

df_cm2 = pd.DataFrame(confusion_matrix(y_tr, pred2), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm2, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of train data')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```






```
Out[83]: Text(0,0.5,'tpr')
```



In [100]:
```
'''# ============================== KNN with k = optimal_k ============
===================================
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_
tree')

# fitting the model
knn_optimal.fit(final_counts, y_tr)

# predict the response
pred = knn_optimal.predict(X_test_bow)
probabilities=knn_optimal.predict_proba(X_test_bow)
print(probabilities.shape)
probabilities=probabilities[:,1]
# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k, acc))'''
```

Out[100]:
```
"# ============================== KNN with k = optimal_k ==============
=================================\n# instantiate learning model k = opt
```

```
imal_k\nknn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algori

thm='kd_tree')\n\n# fitting the model\nknn_optimal.fit(final_counts, y_
tr)\n\n# predict the response\npred = knn_optimal.predict(X_test_bow)\n
probabilities=knn_optimal.predict_proba(X_test_bow)\nprint(probabilitie
s.shape)\nprobabilities=probabilities[:,1]\n# evaluate accuracy\nacc =
accuracy_score(y_test, pred) * 100\nprint('\nThe accuracy of the knn cl
assifier for k = %d is %f%%' % (optimal_k, acc))"
```

In [101]:

```
'''from sklearn.metrics import confusion_matrix as cm
from sklearn.metrics import roc_auc_score
print(y_tr.shape)
print(final_counts.shape)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
fpr, tpr, thresholds = metrics.roc_curve(y_test, probabilities)
#fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
plt.plot([0,1],[0,1],'k--')

auc = roc_auc_score(y_test, probabilities)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %auc)
plt.legend(loc = 'lower right')
plt.title('Roc curve with optimal number of neighbors and also the AUC
 value for optimal number of neighbors displayed')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
print('AUC: %.2f' % auc)

#print(neighbors.shape)
#plt.plot(neighbors,auc1)'''
```

Out[101]:

```
'from sklearn.metrics import confusion_matrix as cm\nfrom sklearn.metri
cs import roc_auc_score\nprint(y_tr.shape)\nprint(final_counts.shape)\n
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
\n    #heatman for visualization of matrix\nsns.heatmap(df_cm, annot=Tr
ue,annot_kws={"size": 16}, fmt=\'g\')\nplt.show()\nfpr, tpr, thresholds
```

```
= metrics.roc_curve(y_test, probabilities)\n#fpr, tpr, thresholds = met
rics.roc_curve(y_tr, pred)\nplt.plot([0,1],[0,1],\'k--\')\n\nauc = roc_
auc_score(y_test, probabilities)  \nplt.plot(fpr, tpr, \'b\', label =
\'AUC = %0.2f\' %auc)\nplt.legend(loc = \'lower right\')\nplt.title(\'R
oc curve with optimal number of neighbors and also the AUC value for op
timal number of neighbors displayed\')\nplt.xlabel(\'FPR\')\nplt.ylabel
(\'TPR\')\nplt.show()\nprint(\'AUC: %.2f\' % auc)\n\n#print(neighbors.s
hape)\n#plt.plot(neighbors,auc1)'
```

[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

In [92]:
```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
count_vect_gram = CountVectorizer(ngram_range=(1,2), min_df=10, max_fea
tures=50)
count_vect_gram.fit(X_tr)
final_bigram_counts = count_vect_gram.transform(X_tr)
final_bigram_counts=scaler.fit_transform(final_bigram_counts)
X_cv_ngram = count_vect_gram.transform(X_cv)
X_cv_ngram=scaler.fit_transform(X_cv_ngram)
X_test_ngram=count_vect_gram.transform(X_test)
X_test_ngram=scaler.fit_transform(X_test_ngram)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_s
hape())
print("the number of unique words including both unigrams and bigrams "
, final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (22574, 50)
the number of unique words including both unigrams and bigrams  50
```

In [93]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features
=500)
tf_idf_vect.fit(X_tr)

print("some sample features(unique words in the corpus)",tf_idf_vect.ge
```

```
t_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(X_tr)
final_tf_idf=scaler.fit_transform(final_tf_idf)
X_cv_tfidf=tf_idf_vect.transform(X_cv)
X_cv_tfidf=scaler.fit_transform(X_cv_tfidf)
X_test_tfidf=tf_idf_vect.transform(X_test)
X_test_tfidf=scaler.fit_transform(X_test_tfidf)
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.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['able', 'absolutely',
'acid', 'actually', 'add', 'added', 'aftertaste', 'ago', 'almonds', 'al
most']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (22574, 500)
the number of unique words including both unigrams and bigrams  500
```

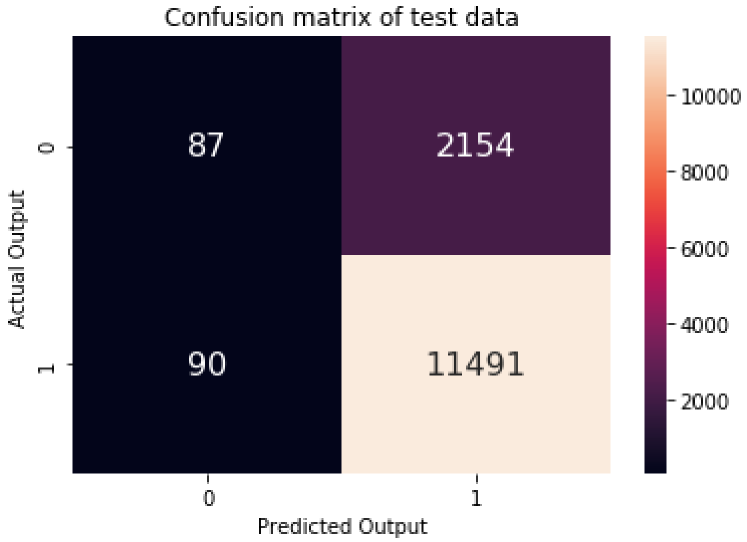
In [94]:

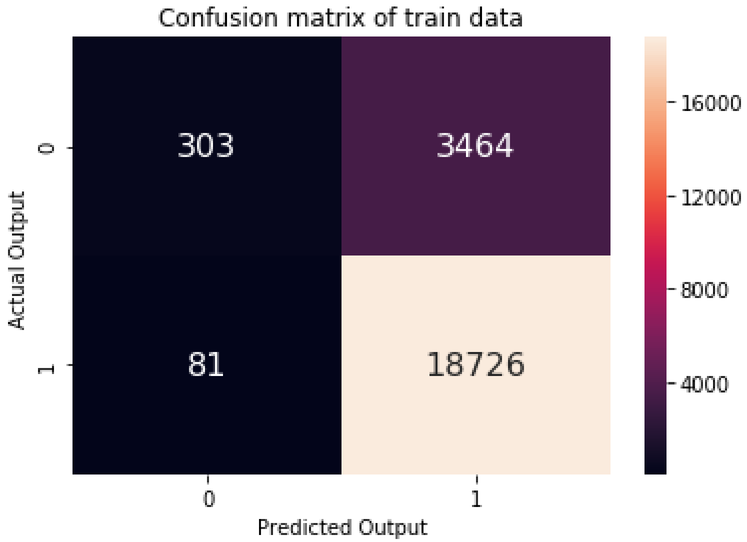
```
neighbors=list(range(1,30,2))
tuned_parameters=[{'n_neighbors':neighbors}]
model=GridSearchCV(KNeighborsClassifier(),tuned_parameters,scoring='roc
_auc',cv=10)
model.fit(final_tf_idf,y_tr)
print(model.best_estimator_)
optimal_neighbors=model.best_estimator_.n_neighbors

print(model.score(X_cv_tfidf,y_cv))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=
2,
                     weights='uniform')
0.5981790622826959
```

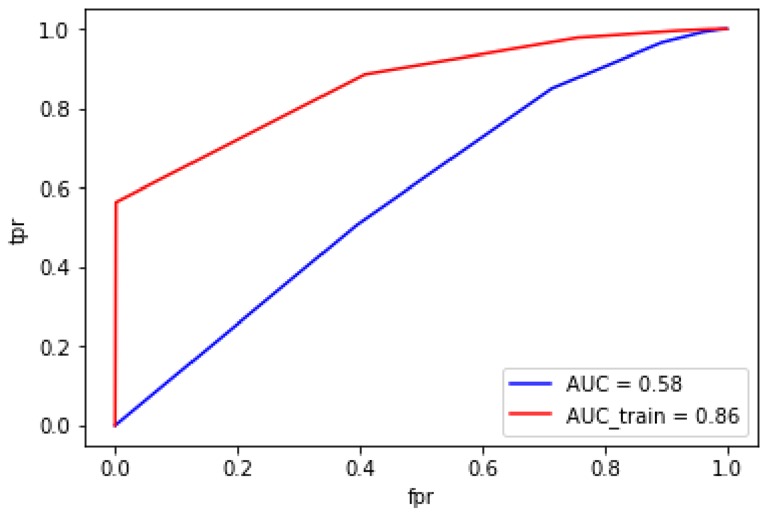
In [95]:

```
model_new=KNeighborsClassifier(n_neighbors=optimal_neighbors,algorithm=
'brute')
model_new.fit(final_tf_idf,y_tr)
pred=model_new.predict(X_test_tfidf)
probab=model_new.predict_proba(X_test_tfidf)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
pred2=model_new.predict(final_tf_idf)
probab2=model_new.predict_proba(final_tf_idf)[:,1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of test data')
plt.show()

df_cm2 = pd.DataFrame(confusion_matrix(y_tr, pred2), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm2, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of train data')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```






```
Out[95]: Text(0,0.5,'tpr')
```



In [102]:

```
'''# ============================== KNN with k = optimal_k ===========
===================================
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_
tree')

# fitting the model
knn_optimal.fit(final_tf_idf, y_tr)

# predict the response
pred = knn_optimal.predict(X_test_tfidf)
probabs=knn_optimal.predict_proba(X_test_tfidf)
probabs=probabs[:,1]
pred2=knn_optimal.predict(final_tf_idf)
probab=knn_optimal.predict_proba(final_tf_idf)
probab=probab[:,1]
# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k, acc))'''
```

Out[102]:

```
"# ============================== KNN with k = optimal_k =============
===============================\n# instantiate learning model k = opt
```

```
imal_k\nknn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algori
thm='kd_tree')\n\n# fitting the model\nknn_optimal.fit(final_tf_idf, y_
tr)\n\n# predict the response\npred = knn_optimal.predict(X_test_tfidf)
\nprobabs=knn_optimal.predict_proba(X_test_tfidf)\nprobabs=probabs[:,1]
\npred2=knn_optimal.predict(final_tf_idf)\nprobab=knn_optimal.predict_p
roba(final_tf_idf)\nprobab=probab[:,1]\n# evaluate accuracy\nacc = accu
racy_score(y_test, pred) * 100\nprint('\nThe accuracy of the knn classi
fier for k = %d is %f%%' % (optimal_k, acc))"
```

In [103]:

```
'''df_cm_tfidf = pd.DataFrame(confusion_matrix(y_test, pred), range(2),
range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm_tfidf, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
fpr, tpr, thresholds = metrics.roc_curve(y_test, probabs)
plt.plot([0,1],[0,1],'k--')

auc_tfidf = roc_auc_score(y_test, probabs)
fpr_train, tpr_train, thresholds = metrics.roc_curve(y_tr, probab)
auc2_tfidf = roc_auc_score(y_tr, probab)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %auc_tfidf)
plt.plot(fpr_train, tpr_train, 'r', label = 'AUC = %0.2f' %auc2_tfidf)
plt.legend(loc = 'lower right')
plt.title('Roc curve with optimal number of neighbors and also the AUC
 value for optimal number of neighbors displayed with TFIDF')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
print('AUC: %.2f' % auc_tfidf)'''
```

Out[103]:

```
'df_cm_tfidf = pd.DataFrame(confusion_matrix(y_test, pred), range(2),ra
nge(2))\n    #heatman for visualization of matrix\nsns.heatmap(df_cm_tf
idf, annot=True,annot_kws={"size": 16}, fmt=\'g\')\nplt.show()\nfpr, tp
r, thresholds = metrics.roc_curve(y_test, probabs)\nplt.plot([0,1],[0,
1],\'k--\')\n\nauc_tfidf = roc_auc_score(y_test, probabs)\nfpr_train, t
pr_train, thresholds = metrics.roc_curve(y_tr, probab)\nauc2_tfidf = ro
c_auc_score(y_tr, probab)\nplt.plot(fpr, tpr, \'b\', label = \'AUC = %
0.2f\' %auc_tfidf)\nplt.plot(fpr_train, tpr_train, \'r\', label = \'AUC
= %0.2f\' %auc2_tfidf)\nplt.legend(loc = \'lower right\')\nplt.title
```

```
(\'Roc curve with optimal number of neighbors and also the AUC value fo
r optimal number of neighbors displayed with TFIDF\')\nplt.xlabel(\'FPR
\')\nplt.ylabel(\'TPR\')\nplt.show()\nprint(\'AUC: %.2f\' % auc_tfidf)'
```

[5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

In [96]:

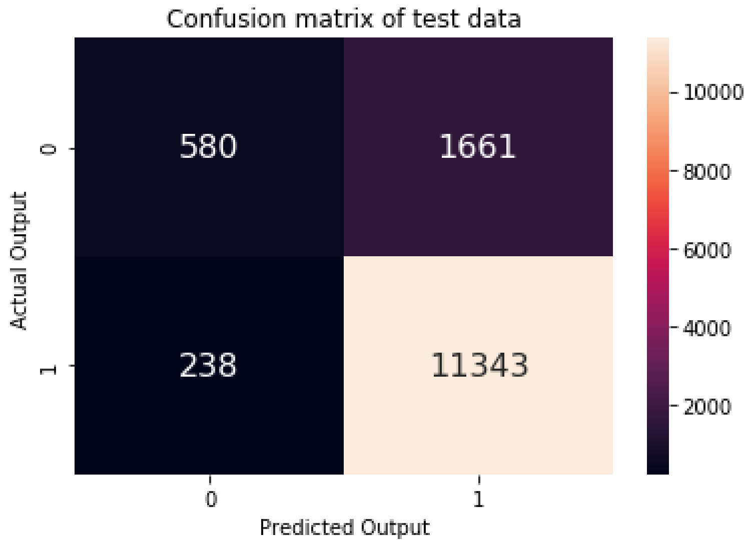
```
neighbors=list(range(1,30,2))
tuned_parameters=[{'n_neighbors':neighbors}]
model=GridSearchCV(KNeighborsClassifier(),tuned_parameters,scoring='roc
_auc',cv=10)
model.fit(sent_vectors_train,y_tr)
print(model.best_estimator_)
optimal_neighbors=model.best_estimator_.n_neighbors

print(model.score(sent_vectors_cv,y_cv))
```

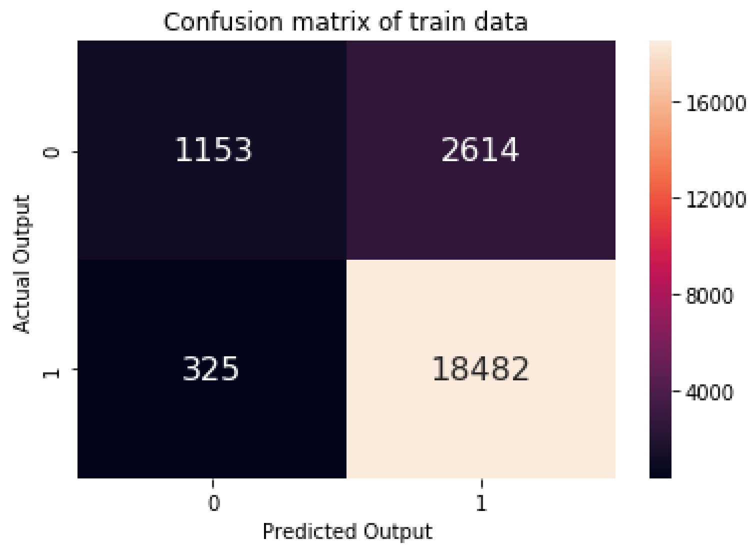
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                     metric_params=None, n_jobs=None, n_neighbors=29, p
=2,
                     weights='uniform')
0.8637954598512625
```

In [97]:

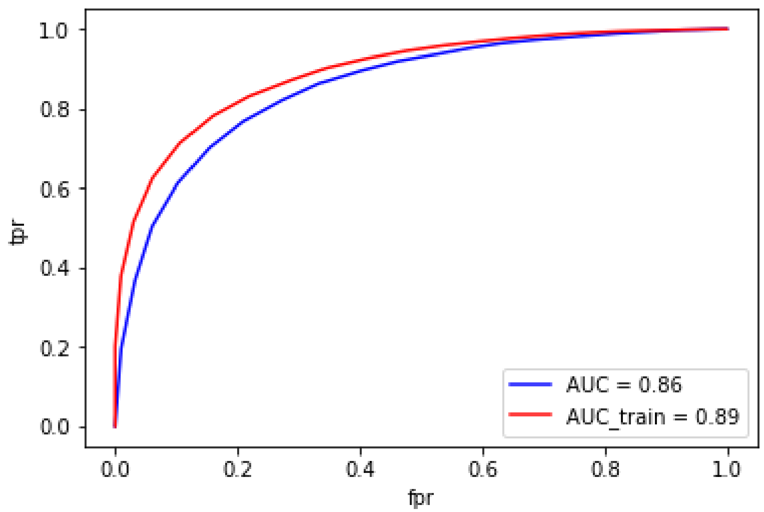
```
model_new=KNeighborsClassifier(n_neighbors=optimal_neighbors,algorithm=
'kd_tree')
model_new.fit(sent_vectors_train,y_tr)
pred=model_new.predict(sent_vectors_test)
probab=model_new.predict_proba(sent_vectors_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
pred2=model_new.predict(sent_vectors_train)
probab2=model_new.predict_proba(sent_vectors_train)[:,1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
```

```
    #heatman for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of test data')
plt.show()

df_cm2 = pd.DataFrame(confusion_matrix(y_tr, pred2), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm2, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of train data')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```





Out[97]: Text(0,0.5,'tpr')



In [104]:
```
'''model_optimal=KNeighborsClassifier(n_neighbors=optimalk,algorithm='k
d_tree')
model_optimal.fit(sent_vectors_train,y_tr)
```

```
pred=model_optimal.predict(sent_vectors_test)
probab_w2v=model_optimal.predict_proba(sent_vectors_test)
probab_w2v=probab_w2v[:,1]
pred2=model_optimal.predict(sent_vectors_train)
probab=model_optimal.predict_proba(sent_vectors_train)
probab=probab[:,1]
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
print('The accuracy of the model with optimal k %d is=%f'%(optimalk,ac
c))'''
```

Out[104]:

```
"model_optimal=KNeighborsClassifier(n_neighbors=optimalk,algorithm='kd_
tree')\nmodel_optimal.fit(sent_vectors_train,y_tr)\npred=model_optimal.
predict(sent_vectors_test)\nprobab_w2v=model_optimal.predict_proba(sent
_vectors_test)\nprobab_w2v=probab_w2v[:,1]\npred2=model_optimal.predict
(sent_vectors_train)\nprobab=model_optimal.predict_proba(sent_vectors_t
rain)\nprobab=probab[:,1]\nacc=accuracy_score(y_test,pred,normalize=Tru
e)*float(100)\nprint('The accuracy of the model with optimal k %d is=%
f'%(optimalk,acc))"
```

In [105]:

```
'''df_cm_w2v = pd.DataFrame(confusion_matrix(y_test, pred), range(2),ra
nge(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm_w2v, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
fpr, tpr, thresholds = metrics.roc_curve(y_test, probab_w2v)
#fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
plt.plot([0,1],[0,1],'k--')

auc_w2v = roc_auc_score(y_test, probab_w2v)
fpr_train, tpr_train, thresholds = metrics.roc_curve(y_tr, probab)
auc2_w2v = roc_auc_score(y_tr, probab)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %auc_w2v)
plt.plot(fpr_train, tpr_train, 'r', label = 'AUC = %0.2f' %auc2_w2v)
plt.legend(loc = 'lower right')
plt.title('Roc curve with optimal number of neighbors and also the AUC
 value for optimal number of neighbors displayed with average w2v')
plt.xlabel('FPR')
plt.ylabel('TPR')
```

```
plt.show()
print('AUC: %.2f' % auc_w2v)'''
```

Out[105]:

```
'df_cm_w2v = pd.DataFrame(confusion_matrix(y_test, pred), range(2),rang
e(2))\n    #heatman for visualization of matrix\nsns.heatmap(df_cm_w2v,
annot=True,annot_kws={"size": 16}, fmt=\'g\')\nplt.show()\nfpr, tpr, th
resholds = metrics.roc_curve(y_test, probab_w2v)\n#fpr, tpr, thresholds
= metrics.roc_curve(y_tr, pred)\nplt.plot([0,1],[0,1],\'k--\')\n\nauc_w
2v = roc_auc_score(y_test, probab_w2v)\nfpr_train, tpr_train, threshold
s = metrics.roc_curve(y_tr, probab)\nauc2_w2v = roc_auc_score(y_tr, pro
bab)\nplt.plot(fpr, tpr, \'b\', label = \'AUC = %0.2f\' %auc_w2v)\nplt.
plot(fpr_train, tpr_train, \'r\', label = \'AUC = %0.2f\' %auc2_w2v)\np
lt.legend(loc = \'lower right\')\nplt.title(\'Roc curve with optimal nu
mber of neighbors and also the AUC value for optimal number of neighbor
s displayed with average w2v\')\nplt.xlabel(\'FPR\')\nplt.ylabel(\'TPR
\')\nplt.show()\nprint(\'AUC: %.2f\' % auc_w2v)'
```

[5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

In [98]:

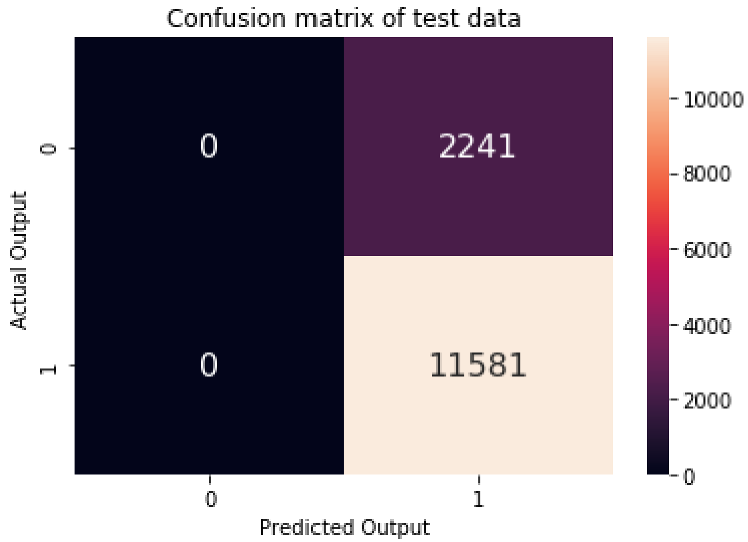
```
neighbors=list(range(1,30,2))
tuned_parameters=[{'n_neighbors':neighbors}]
model=GridSearchCV(KNeighborsClassifier(),tuned_parameters,scoring='roc
_auc',cv=10)
model.fit(tfidf_sent_vectors_train,y_tr)
print(model.best_estimator_)
optimal_neighbors=model.best_estimator_.n_neighbors

print(model.score(tfidf_sent_vectors_cv,y_cv))
```

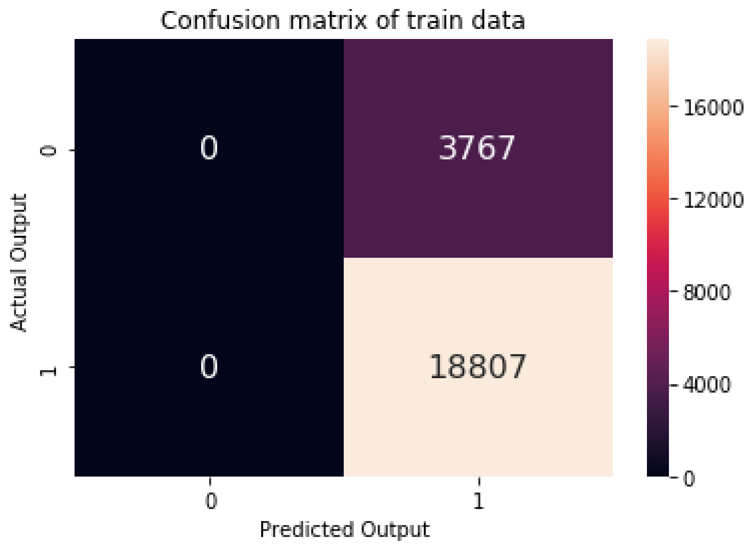
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                     metric_params=None, n_jobs=None, n_neighbors=29, p
=2,
                     weights='uniform')
0.5007900192132673
```

In [99]:

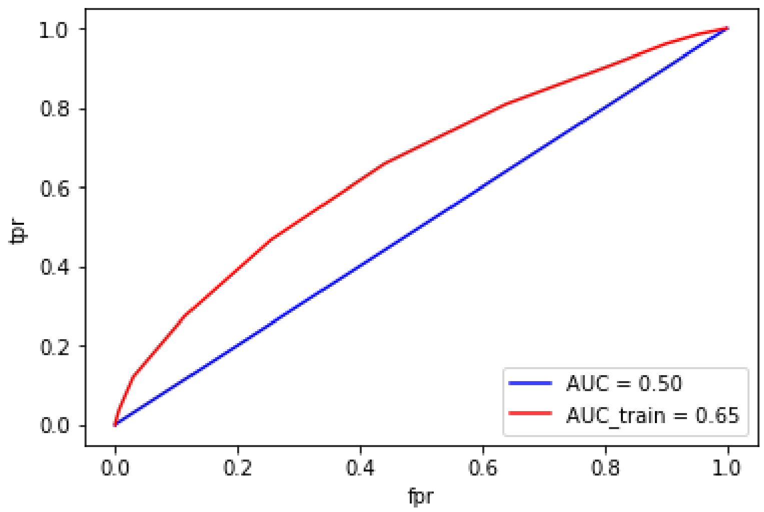
```
model_new=KNeighborsClassifier(n_neighbors=optimal_neighbors,algorithm=
'kd_tree')
```

```
model_new.fit(tfidf_sent_vectors_train,y_tr)
pred=model_new.predict(tfidf_sent_vectors_test)
probab=model_new.predict_proba(tfidf_sent_vectors_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,probab)
auc=roc_auc_score(y_test,probab)
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
pred2=model_new.predict(tfidf_sent_vectors_train)
probab2=model_new.predict_proba(tfidf_sent_vectors_train)[:,1]
fpr_train,tpr_train,thresholds=roc_curve(y_tr,probab2)
auc2=roc_auc_score(y_tr,probab2)
df_cm = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of test data')
plt.show()

df_cm2 = pd.DataFrame(confusion_matrix(y_tr, pred2), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm2, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Predicted Output')
plt.ylabel('Actual Output')
plt.title('Confusion matrix of train data')
plt.show()
plt.plot(fpr,tpr,'b', label = 'AUC = %0.2f' %auc)
plt.plot(fpr_train,tpr_train,'r', label = 'AUC_train = %0.2f' %auc2)
plt.legend(loc='lower right')
plt.xlabel('fpr')
plt.ylabel('tpr')
```






```
Out[99]: Text(0,0.5,'tpr')
```



In [106]:
```
'''model_optimal=KNeighborsClassifier(n_neighbors=optimalk,algorithm='k
d_tree')
model_optimal.fit(tfidf_sent_vectors_train,y_tr)
pred=model_optimal.predict(tfidf_sent_vectors_test)
probab_w2v=model_optimal.predict_proba(tfidf_sent_vectors_test)
probab_w2v=probab_w2v[:,1]
pred2=model_optimal.predict(tfidf_sent_vectors_train)
probab=model_optimal.predict_proba(tfidf_sent_vectors_train)
probab=probab[:,1]
acc=accuracy_score(y_test,pred,normalize=True)*float(100)
print('The accuracy of the model with optimal k %d is=%f'%(optimalk,ac
c))'''
```

Out[106]:
```
"model_optimal=KNeighborsClassifier(n_neighbors=optimalk,algorithm='kd_
tree')\nmodel_optimal.fit(tfidf_sent_vectors_train,y_tr)\npred=model_op
timal.predict(tfidf_sent_vectors_test)\nprobab_w2v=model_optimal.predic
t_proba(tfidf_sent_vectors_test)\nprobab_w2v=probab_w2v[:,1]\npred2=mod
el_optimal.predict(tfidf_sent_vectors_train)\nprobab=model_optimal.pred
ict_proba(tfidf_sent_vectors_train)\nprobab=probab[:,1]\nacc=accuracy_s
core(y_test,pred,normalize=True)*float(100)\nprint('The accuracy of the
```

```
model with optimal k %d is=%f'%(optimalk,acc))"
```

In [107]:

```
'''df_cm_w2v = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))
    #heatman for visualization of matrix
sns.heatmap(df_cm_w2v, annot=True,annot_kws={"size": 16}, fmt='g')
plt.show()
fpr, tpr, thresholds = metrics.roc_curve(y_test, probab_w2v)
#fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)
plt.plot([0,1],[0,1],'k--')

auc_w2v = roc_auc_score(y_test, probab_w2v)
fpr_train, tpr_train, thresholds = metrics.roc_curve(y_tr, probab)
auc2_w2v = roc_auc_score(y_tr, probab)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %auc_w2v)
plt.plot(fpr_train, tpr_train, 'r', label = 'AUC = %0.2f' %auc2_w2v)
plt.legend(loc = 'lower right')
plt.title('Roc curve with optimal number of neighbors and also the AUC
 value for optimal number of neighbors displayed with weighted TFIDF')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
print('AUC: %.2f' % auc_w2v)'''
```

Out[107]:

```
'df_cm_w2v = pd.DataFrame(confusion_matrix(y_test, pred), range(2),range(2))\n    #heatman for visualization of matrix\nsns.heatmap(df_cm_w2v, annot=True,annot_kws={"size": 16}, fmt=\'g\')\nplt.show()\nfpr, tpr, thresholds = metrics.roc_curve(y_test, probab_w2v)\n#fpr, tpr, thresholds = metrics.roc_curve(y_tr, pred)\nplt.plot([0,1],[0,1],\'k--\')\n\nauc_w2v = roc_auc_score(y_test, probab_w2v)\nfpr_train, tpr_train, thresholds = metrics.roc_curve(y_tr, probab)\nauc2_w2v = roc_auc_score(y_tr, probab)\nplt.plot(fpr, tpr, \'b\', label = \'AUC = %0.2f\' %auc_w2v)\nplt.plot(fpr_train, tpr_train, \'r\', label = \'AUC = %0.2f\' %auc2_w2v)\nplt.legend(loc = \'lower right\')\nplt.title(\'Roc curve with optimal number of neighbors and also the AUC value for optimal number of neighbors displayed with weighted TFIDF\')\nplt.xlabel(\'FPR\')\nplt.ylabel(\'TPR\')\nplt.show()\nprint(\'AUC: %.2f\' % auc_w2v)'
```

[6] Conclusions

In [108]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=['vectorizer','Model','Hyperparameter','AUC']
x.add_row(['Bag Of words','Brute',9,0.63])
x.add_row(['TFIDF','Brute',7,0.51])
x.add_row(['AVG W2V','Brute',29,0.86])
x.add_row(['TFIDF weighted W2V','Brute',29,0.50])
x.add_row(['BOW','kd-tree',29,0.73])
x.add_row(['TFIDF','kd-tree',7,0.58])
x.add_row(['AVG W2V','kd-tree',29,0.86])
x.add_row(['TFIDF weighted W2V','kd-tree',29,0.50])
print(x)
```

```
+--------------------+---------+----------------+------+
|     vectorizer     |  Model  | Hyperparameter | AUC  |
+--------------------+---------+----------------+------+
|    Bag Of words    |  Brute  |       9        | 0.63 |
|       TFIDF        |  Brute  |       7        | 0.51 |
|      AVG W2V       |  Brute  |       29       | 0.86 |
| TFIDF weighted W2V |  Brute  |       29       | 0.5  |
|        BOW         | kd-tree |       29       | 0.73 |
|       TFIDF        | kd-tree |       7        | 0.58 |
|      AVG W2V       | kd-tree |       29       | 0.86 |
| TFIDF weighted W2V | kd-tree |       29       | 0.5  |
+--------------------+---------+----------------+------+
```