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 [2]:

```
%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, 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 [3]:

```python
# 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 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 200000""", 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(5)
```

```
Number of data points in our data (200000, 10)
```

Out[3]:

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomin
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	

In [4]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [5]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
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 [6]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[6]:

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

In [7]:
```
display['COUNT(*)'].sum()
```

Out[7]:
```
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 [8]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
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	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
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 [9]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=Tr
ue, inplace=False, kind='quicksort', na_position='last')
```

In [10]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time"
```

```
,"Text"}, keep='first', inplace=False)
final.shape
```

Out[10]: (160178, 10)

In [11]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[11]: 80.089

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 [12]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[12]:

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

In [13]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [14]:
```
#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()
```

```
(160176, 10)
```

Out[14]:
```
1    134799
0     25377
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 [15]:
```
# 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 remembered this book from my childhood and got it for my kids.  It's
just as good as I remembered and my kids love it too.  My older daughte
r now reads it to her sister.  Good rhymes and nice pictures.
==================================================
The qualitys not as good as the lamb and rice but it didn't seem to bot
her his stomach, you get 10 more pounds and it is cheaper wich is a plu
s for me. You can always ad your own rice and veggies. Its fresher that
way and better for him in my opinion. Plus if you you can get it delive
rd to your house for free its even better.  Gotta love pitbulls
==================================================
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loa
n-word, and&quot;ko-&quot;  is &quot;child of&quot; or of &quot;derived
from&quot;.) Panko are used for katsudon, tonkatsu or cutlets served on
rice or in soups. The cutlets, pounded chicken or pork, are coated with
these light and crispy crumbs and fried. They are not gritty and dense
like regular crumbs. They are very nice on deep fried shrimps and decor
ative for a more gourmet touch.
==================================================
What can I say... If Douwe Egberts was good enough for my dutch grandmo
ther, it's perfect for me.  I like this flavor best with my Senseo... I
t has a nice dark full body flavor without the burt bean taste I tend s
```

```
ense with starbucks.  It's a shame most americans haven't bought into s
ingle serve coffe makers as our Dutch counter parts have.  Every cup is
fresh brewed and doesn't sit long enough on my desk to get that old tas
te either.
==================================================
```

In [16]:

```
# 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)
```

```
I remembered this book from my childhood and got it for my kids.  It's
just as good as I remembered and my kids love it too.  My older daughte
r now reads it to her sister.  Good rhymes and nice pictures.
```

In [17]:

```
# 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 remembered this book from my childhood and got it for my kids.  It's
just as good as I remembered and my kids love it too.  My older daughte
r now reads it to her sister.  Good rhymes and nice pictures.
==================================================
The qualitys not as good as the lamb and rice but it didn't seem to bot
her his stomach, you get 10 more pounds and it is cheaper wich is a plu
s for me. You can always ad your own rice and veggies. Its fresher that
way and better for him in my opinion. Plus if you you can get it delive
rd to your house for free its even better.  Gotta love pitbulls
==================================================
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loa
n-word, and"ko-"  is "child of" or of "derived from".) Panko are used f
or katsudon, tonkatsu or cutlets served on rice or in soups. The cutlet
s, pounded chicken or pork, are coated with these light and crispy crum
bs and fried. They are not gritty and dense like regular crumbs. They a
re very nice on deep fried shrimps and decorative for a more gourmet to
uch.
==================================================
What can I say... If Douwe Egberts was good enough for my dutch grandmo
ther, it's perfect for me.  I like this flavor best with my Senseo... I
t has a nice dark full body flavor without the burt bean taste I tend s
ense with starbucks.  It's a shame most americans haven't bought into s
ingle serve coffe makers as our Dutch counter parts have.  Every cup is
fresh brewed and doesn't sit long enough on my desk to get that old tas
te either.
```

In [18]:

```
# 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 [19]:

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

```
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loa
n-word, and&quot;ko-&quot;  is &quot;child of&quot; or of &quot;derived
from&quot;.) Panko are used for katsudon, tonkatsu or cutlets served on
rice or in soups. The cutlets, pounded chicken or pork, are coated with
these light and crispy crumbs and fried. They are not gritty and dense
like regular crumbs. They are very nice on deep fried shrimps and decor
ative for a more gourmet touch.
==================================================
```

In [20]:

```
#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)
```

```
I remembered this book from my childhood and got it for my kids.  It's
just as good as I remembered and my kids love it too.  My older daughte
r now reads it to her sister.  Good rhymes and nice pictures.
```

In [21]:

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

```
This is the Japanese version of breadcrumb pan bread a Portuguese loan
word and quot ko quot is quot child of quot or of quot derived from quo
t Panko are used for katsudon tonkatsu or cutlets served on rice or in
soups The cutlets pounded chicken or pork are coated with these light a
```

nd crispy crumbs and fried They are not gritty and dense like regular c
rumbs They are very nice on deep fried shrimps and decorative for a mor
e gourmet touch

In [22]:

```python
# 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 [23]:
```
# 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%|██████████████████████████████████████████████████████████████████
██████| 160176/160176 [01:18<00:00, 2033.98it/s]
```

In [24]:
```
preprocessed_reviews[1500]
```

Out[24]:
```
'japanese version breadcrumb pan bread portuguese loan word ko child de
rived panko used katsudon tonkatsu cutlets served rice soups cutlets po
unded chicken pork coated light crispy crumbs fried not gritty dense li
ke regular crumbs nice deep fried shrimps decorative gourmet touch'
```

In [27]:
```
final['Cleaned_Text']=preprocessed_reviews
```

[3.2] Preprocessing Review Summary

In [26]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [0]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
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', 'aahhhs', 'aback', 'abandon', 'abates', 'abb
ott', 'abby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [0]:
```
#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

# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features
=5000)
```

```
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_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  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [0]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
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(preprocessed_reviews)
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 find', 'able get', 'absolute', 'absolutely', 'absolutely deliciou
s', 'absolutely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [83]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [0]:
```
# 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=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    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.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 ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wond
erful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('espec
ially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted',
0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.99
36816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('p
opcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.99
92451071739197), ('melitta', 0.999218761920929), ('choice', 0.999210238
4567261), ('american', 0.9991837739944458), ('beef', 0.999178051948547
4), ('finish', 0.9991567134857178)]
```

In [0]:

```
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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'st
inky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'receiv
ed', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'ins
tead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use',
'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fu
n', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea',
'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'mad
e']
```

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

[4.4.1.1] Avg W2v

In [84]:

```python
# 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(sent_vectors.shape)
print(len(sent_vectors[0]))
```

```
100%|██████████| 160176/160176 [04:27<00:00, 599.80it/s]
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-84-22f593de542d> in <module>
     13         sent_vec /= cnt_words
     14     sent_vectors.append(sent_vec)
---> 15 print(sent_vectors.shape)
     16 print(len(sent_vectors[0]))

AttributeError: 'list' object has no attribute 'shape'
```

[4.4.1.2] TFIDF weighted W2v

```
In [0]: # 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 v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # 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 ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored 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/r
eview
    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
```

```
100%|██████████████████████████████████████████████████
████████| 4986/4986 [00:20<00:00, 245.63it/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_fe
    atures=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_df=10, m
    ax_features=500)
                 tf_idf_vect.fit(preprocessed_reviews)
    ```

    - 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

In [28]:
```
re_po = final[final["Score"] == 1].sample(n=15000)
re_ne = final[final["Score"] == 0].sample(n=15000)
tot_re = pd.concat([re_po,re_ne])
tot_re.shape
```

Out[28]:
```
(30000, 11)
```

In [29]:
```
x=tot_re['Cleaned_Text'].values
y=tot_re['Score'].values
print(type(x),type(y))
print(x.shape,y.shape)
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
(30000,) (30000,)
```

In [30]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [31]:
```
x_tr,x_te,y_tr,y_te = train_test_split(x,y,test_size=0.2,random_state=1
4)
x_tr,x_cv,y_tr,y_cv = train_test_split(x,y,test_size=0.2,random_state=1
4)
print('='*50)
print(x_tr.shape,y_tr.shape)
print(x_te.shape,y_te.shape)
print(x_cv.shape,y_cv.shape)
print('='*50)
```

```
==================================================
(24000,) (24000,)
(6000,) (6000,)
```

```
(6000,) (6000,)

(6000,) (6000,)
==================================================
```

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

In [32]:

```
vectorizer = CountVectorizer()
vectorizer.fit(x_tr)
x_tr_bow = vectorizer.transform(x_tr)
x_cv_bow = vectorizer.transform(x_cv)
x_te_bow = vectorizer.transform(x_te)
print('='*50)
print(x_tr_bow.shape,y_tr.shape)
print(x_cv_bow.shape,y_cv.shape)
print(x_te_bow.shape,y_te.shape)
print('='*50)
```

```
==================================================
(24000, 31385) (24000,)
(6000, 31385) (6000,)
(6000, 31385) (6000,)
==================================================
```

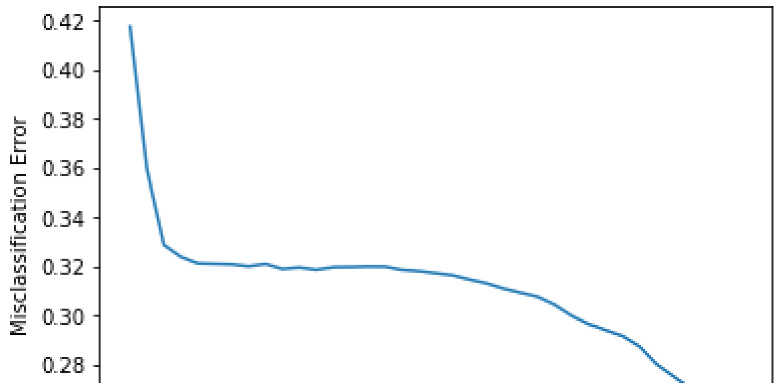
In [36]:

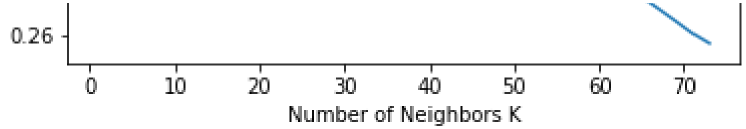
```
my_list = list(range(1,75))
neighbors = list(filter(lambda x : x%2!=0,my_list))
cv_scores = []
for k in tqdm(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    scores = cross_val_score(knn, x_tr_bow, y_tr, cv=10, scoring='roc_a
uc')
    cv_scores.append(scores.mean())
# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
```

```
plt.plot(neighbors, MSE)

# for xy in zip(neighbors, np.round(MSE,3)):
#     plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

print('='*50)
print('\nThe optimal number of neighbors is %d.' % optimal_k)
print("the misclassification error for each k value is : ", np.round(MS
E,3))
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
print('='*50)
```

```
100%|██████████████████████████████████████| 37/37 [18:27<00:00, 30.13s/it]
```

```
==================================================

The optimal number of neighbors is 73.
the misclassification error for each k value is :  [0.417 0.359 0.329
0.324 0.321 0.321 0.321 0.32  0.321 0.319 0.32  0.319
 0.32  0.32  0.32  0.32  0.319 0.318 0.317 0.316 0.315 0.313 0.311 0.
309
 0.308 0.305 0.3   0.297 0.294 0.292 0.287 0.28  0.275 0.271 0.266 0.
261
 0.256]
```






```
==================================================
```

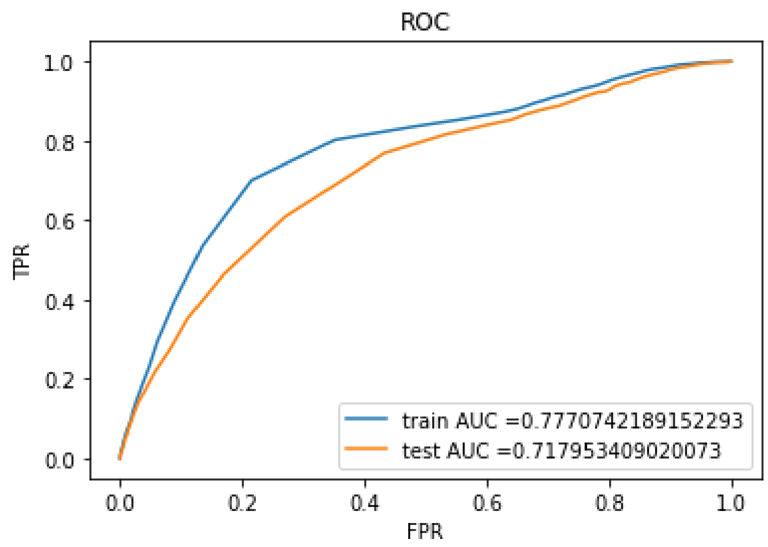
In [37]:

```python
# instantiate learning model k = optimal_k
knn_optimal =  KNeighborsClassifier(n_neighbors=optimal_k,algorithm='br
ute')
knn_optimal.fit(x_tr_bow,y_tr)
pred = knn_optimal.predict(x_te_bow)
# evaluate accuracy
acc = accuracy_score(y_te, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k, acc))
```

```
The accuracy of the knn classifier for k = 73 is 61.983333%
```

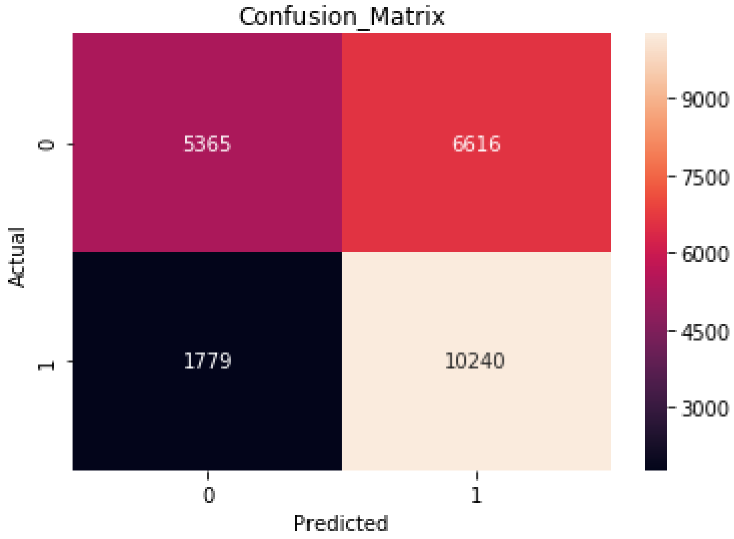
In [38]:

```python
#plotting Auc
tr_fpr,tr_tpr,threshold = roc_curve(y_tr, knn_optimal.predict_proba(x_t
r_bow)[:,1])
te_fpr,te_tpr,threshold =  roc_curve(y_te, knn_optimal.predict_proba(x_
te_bow)[:,1])
AUC1=str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr,tr_tpr,label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr,te_tpr,label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

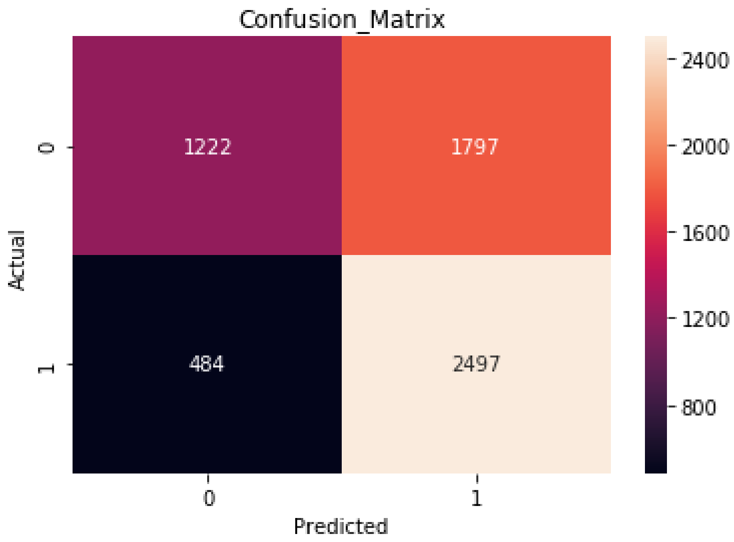


In [39]:

```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(y_tr, knn_optimal.predict(x_tr_bow))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [40]:
```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(y_te, knn_optimal.predict(x_te_bow))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [41]:
```
#Classifiaction Report
from sklearn.metrics import classification_report
print('='*50)
print(classification_report(y_te,pred))
print('='*50)
```

```
==================================================
              precision    recall  f1-score   support

           0       0.72      0.40      0.52      3019
           1       0.58      0.84      0.69      2981

   micro avg       0.62      0.62      0.62      6000
   macro avg       0.65      0.62      0.60      6000
weighted avg       0.65      0.62      0.60      6000

==================================================
```

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

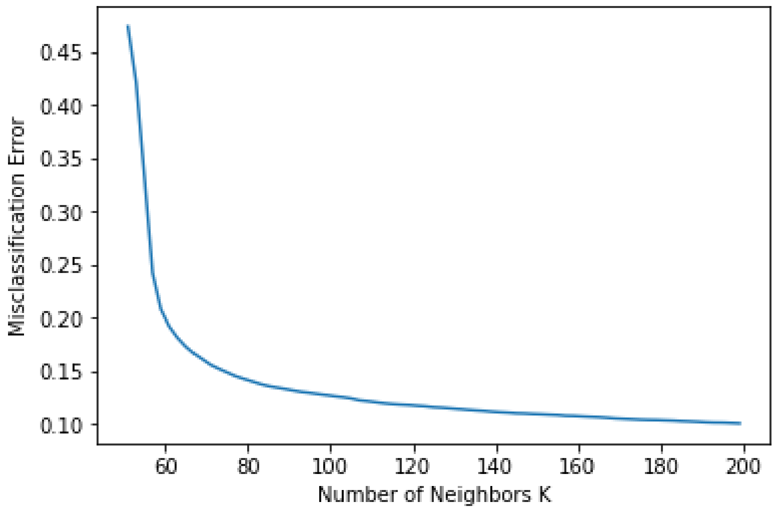
In [42]:
```python
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(x_tr)
x_tr_tfidf = tf_idf_vect.transform(x_tr)
x_cv_tfidf = tf_idf_vect.transform(x_cv)
x_te_tfidf = tf_idf_vect.transform(x_te)
print(x_tr_tfidf.shape,y_tr.shape)
```

```
(24000, 14757) (24000,)
```

In [43]:
```python
my_list = list(range(50,200))
neighbors = list(filter(lambda x : x%2!=0,my_list))
cv_scores = []
for k in tqdm(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    scores = cross_val_score(knn, x_tr_tfidf, y_tr, cv=10, scoring='roc_auc')
    cv_scores.append(scores.mean())
# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k1 = neighbors[MSE.index(min(MSE))]

plt.plot(neighbors, MSE)

# for xy in zip(neighbors, np.round(MSE,3)):
#     plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

print('='*50)
print('\nThe optimal number of neighbors is %d.' % optimal_k1)
print("the misclassification error for each k value is : ", np.round(MSE,3))
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
print('='*50)
```

```
100%|██████████████████████████████████████████████████████████████████| 75/75 [38:39<00:00, 32.04s/it]
```

```
=====================================================

The optimal number of neighbors is 199.
the misclassification error for each k value is :  [0.474 0.422 0.333
0.242 0.208 0.192 0.182 0.173 0.167 0.161 0.156 0.152
 0.149 0.145 0.143 0.14  0.138 0.136 0.134 0.133 0.132 0.13  0.129 0.12
8
 0.127 0.126 0.125 0.124 0.122 0.122 0.121 0.12  0.119 0.118 0.118 0.11
7
 0.117 0.116 0.115 0.115 0.114 0.113 0.113 0.112 0.112 0.111 0.111 0.11
 0.11  0.109 0.109 0.109 0.109 0.108 0.108 0.107 0.107 0.106 0.106 0.10
5
 0.105 0.105 0.104 0.104 0.104 0.103 0.103 0.103 0.102 0.102 0.102 0.10
1
 0.101 0.101 0.101]
```




```
=====================================================
```

In [44]:

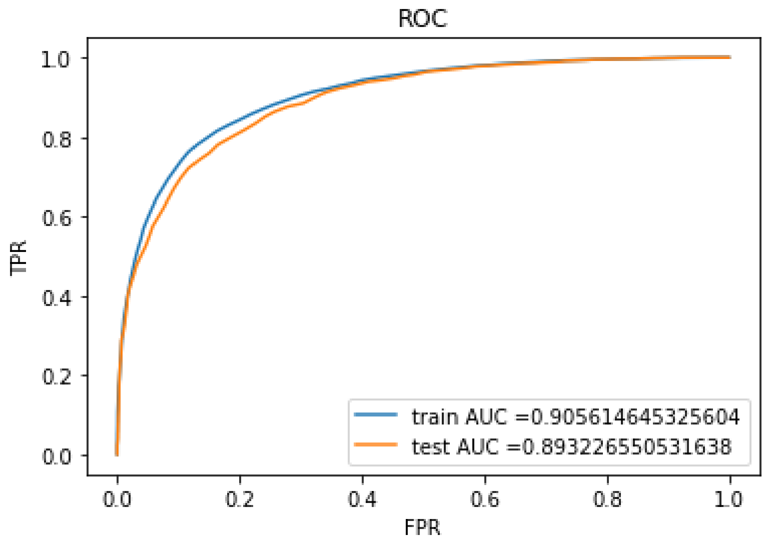
```
# instantiate learning model k = optimal_k
knn_optimal1 =  KNeighborsClassifier(n_neighbors=optimal_k1,algorithm=
'brute')
knn_optimal1.fit(x_tr_tfidf,y_tr)
pred1 = knn_optimal1.predict(x_te_tfidf)
```

```
# evaluate accuracy
acc = accuracy_score(y_te, pred1) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k1, acc))
# auc = roc_auc_score(y_te, pred1)
# print('AUC: %.3f' % auc)
```

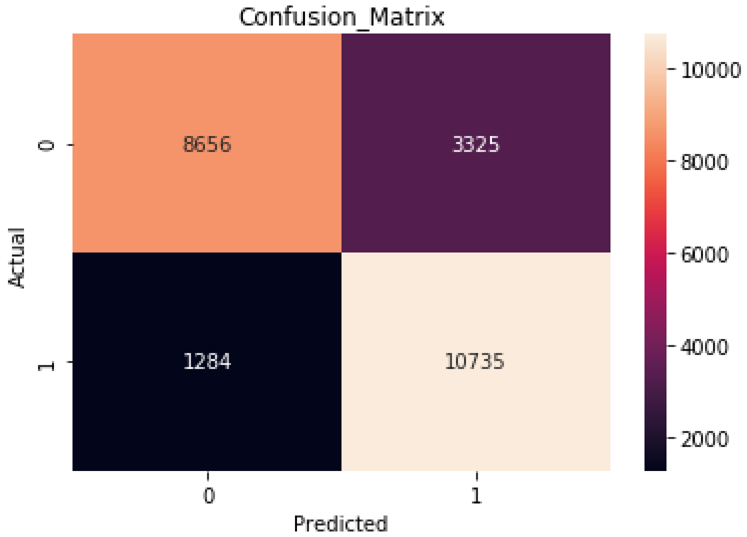
```
The accuracy of the knn classifier for k = 199 is 78.983333%
```

In [45]:

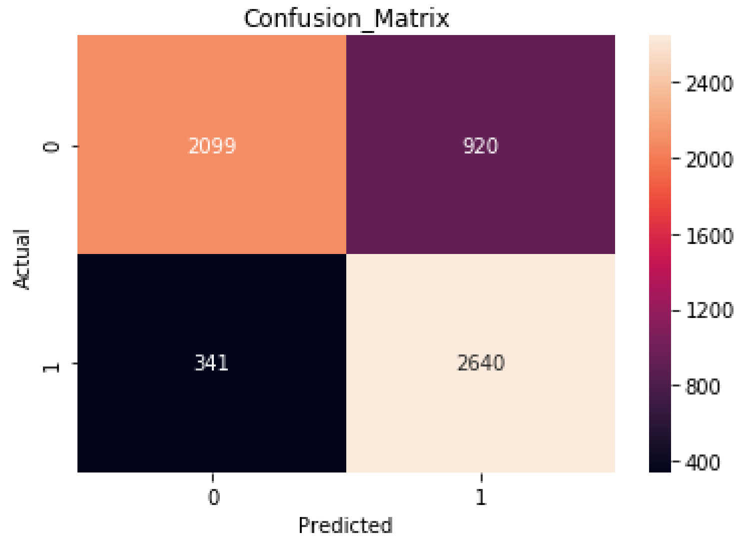
```
#plotting Auc
tr_fpr,tr_tpr,threshold = roc_curve(y_tr, knn_optimal1.predict_proba(x_tr_tfidf)[:,1])
te_fpr,te_tpr,threshold =  roc_curve(y_te, knn_optimal1.predict_proba(x_te_tfidf)[:,1])
AUC2=str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr,tr_tpr,label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr,te_tpr,label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



In [46]:
```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(y_tr, knn_optimal1.predict(x_tr_tfidf))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [47]:
```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(y_te, knn_optimal1.predict(x_te_tfidf))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [48]:
```
print('='*50)
print(classification_report(y_te,pred1))
print('='*50)
```

```
==================================================
              precision    recall  f1-score   support

           0       0.86      0.70      0.77      3019
           1       0.74      0.89      0.81      2981

   micro avg       0.79      0.79      0.79      6000
   macro avg       0.80      0.79      0.79      6000
weighted avg       0.80      0.79      0.79      6000

==================================================
```

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

In [49]:
```
i=0
list_of_sentance_tr=[]
```

```
for sentance in x_tr:
    list_of_sentance_tr.append(sentance.split())

# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_tr,min_count=5,size=50, workers=4)

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

```
D:\anaconda\lib\site-packages\gensim\models\base_any2vec.py:743: UserWa
rning: C extension not loaded, training will be slow. Install a C compi
ler and reinstall gensim for fast training.
  "C extension not loaded, training will be slow. "
```

```
number of words that occured minimum 5 times  10053
sample words  ['buy', 'syrups', 'make', 'soda', 'plain', 'seltzer', 'wa
ter', 'one', 'far', 'absolute', 'delicious', 'found', 'particular', 'fl
avor', 'incredibly', 'true', 'real', 'red', 'grapefruit', 'yet', 'swee
t', 'powerful', 'great', 'works', 'whether', 'want', 'little', 'lot',
'depending', 'given', 'day', 'comparable', 'squirt', 'fresca', 'honestl
y', 'think', 'actual', 'much', 'richer', 'satisfying', 'like', 'taste',
'even', 'bit', 'not', 'disappointed', 'first', 'thing', 'smaller', 'gia
nt']
```

In [50]:

```
##Train
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_tr = []; # the avg-w2v for each sentence/review is stored
 in this list
for sent in tqdm(list_of_sentance_tr): # 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_tr.append(sent_vec)
print(len(sent_vectors_tr))
print(sent_vectors_tr[0])
```

```
100%|██████████████████████████████████████████████████████████████████
█████████| 24000/24000 [00:42<00:00, 563.90it/s]
```

```
24000
[ 0.06484337  0.3176012  -0.64374802 -0.97687852 -0.24947005  0.0734587
3
 -0.08209863 -0.63508992 -0.33671707  0.70337829 -0.24334942 -0.2548012
2
  0.56096828  0.58465865  0.29430533  0.2930852   0.63449709  0.5831524
7
  0.43466243 -0.40909491  0.11850198  0.35758455  0.96805831  0.2325512
5
  0.39342728 -0.35691857 -0.0176755  -0.09203253  0.12825657  0.115413
 -0.28869493  0.13938682 -0.16692821 -0.33286151  0.29107467 -0.8116217
8
  0.48809975  0.01351592 -0.20518347 -0.08621672  0.26715764  0.2872682
9
  0.14193768 -0.29759811  0.20994722  0.34728008 -0.27289211  0.1223708
1
 -1.20830411 -0.18457564]
```

In [51]:

```
##CV
i=0
list_of_sentance_cv = []
for sentance in x_cv:
    list_of_sentance_cv.append(sentance.split())
sent_vectors_cv=[];# the avg-w2v for each sentence/review in CV is stor
ed 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, 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_cv.append(sent_vec)
print(len(sent_vectors_cv))
print(sent_vectors_cv[0])
```

```
100%|████████████████████████████████████████
███████████| 6000/6000 [00:10<00:00, 560.80it/s]
```

```
6000
[-0.02259439  0.47140595  0.0709593  -0.52594688  0.27537727  0.5997683
1
 -0.41313781 -0.62573318 -0.71371215  0.09766519 -0.20574257 -0.3914626
5
  0.43784188  0.75323528  0.4256558   0.57345164  0.48285142  0.3540677
5
  0.74258666 -0.25456232 -0.13211502 -0.24187992  1.15970008  0.2751539
3
  0.57660884 -0.48791247  0.23368932 -0.30854924 -0.36303412  0.3316030
3
  0.10838795 -0.26599739  0.2227034  -0.62396284  0.60153385 -1.0621096
6
  0.63478984 -0.1279924   0.03424346 -0.17528882  1.17674778  0.0525909
1
 -0.10348151  0.40585992 -0.50582543  0.09054748  0.06046833 -0.3021328
 -1.44126385 -0.23500798]
```

In [52]:

```
##Test
i=0
list_of_sentance_te = []
for sentance in x_te:
    list_of_sentance_te.append(sentance.split())
sent_vectors_te = []; # the avg-w2v for each sentence/review is stored
```

```
 in this list
for sent in tqdm(list_of_sentance_te): # 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_te.append(sent_vec)
print(len(sent_vectors_te))
print(sent_vectors_te[0])
```

```
100%|██████████████████████████████████████████████████████████████████
███████████| 6000/6000 [00:10<00:00, 559.86it/s]
```

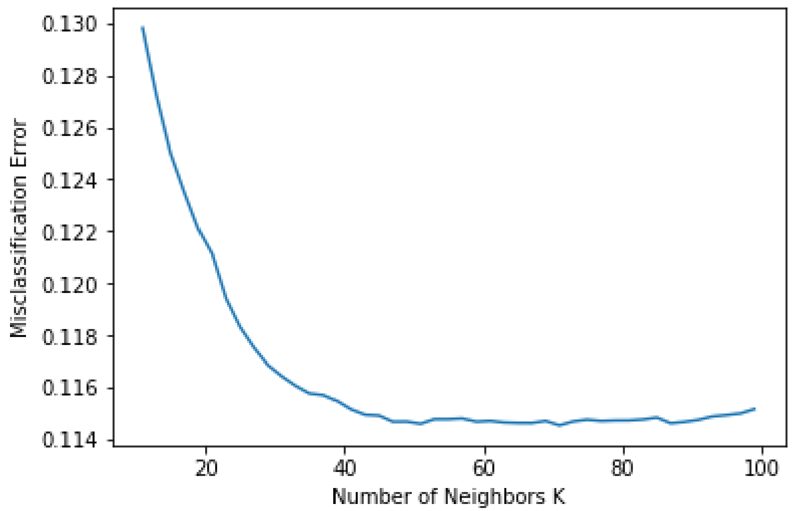
```
6000
[-0.02259439  0.47140595  0.0709593  -0.52594688  0.27537727  0.5997683
1
 -0.41313781 -0.62573318 -0.71371215  0.09766519 -0.20574257 -0.3914626
5
  0.43784188  0.75323528  0.4256558   0.57345164  0.48285142  0.3540677
5
  0.74258666 -0.25456232 -0.13211502 -0.24187992  1.15970008  0.2751539
3
  0.57660884 -0.48791247  0.23368932 -0.30854924 -0.36303412  0.3316030
3
  0.10838795 -0.26599739  0.2227034  -0.62396284  0.60153385 -1.0621096
6
  0.63478984 -0.1279924   0.03424346 -0.17528882  1.17674778  0.0525909
1
 -0.10348151  0.40585992 -0.50582543  0.09054748  0.06046833 -0.3021328
 -1.44126385 -0.23500798]
```

In [53]:
```
my_list = list(range(10,100))
```

```
neighbors = list(filter(lambda x : x%2!=0,my_list))
cv_scores = []
for k in tqdm(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    scores = cross_val_score(knn, sent_vectors_tr , y_tr, cv=10, scorin
g='roc_auc')
    cv_scores.append(scores.mean())
# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k2 = neighbors[MSE.index(min(MSE))]

plt.plot(neighbors, MSE)

# for xy in zip(neighbors, np.round(MSE,3)):
#     plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

print('='*50)
print('\nThe optimal number of neighbors is %d.' % optimal_k2)
print("the misclassification error for each k value is : ", np.round(MS
E,3))
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
print('='*50)
```

```
100%|██████████████████████████████████████████████████████████████████
████████████████| 45/45 [11:04<00:00, 15.20s/it]
```

```
==================================================

The optimal number of neighbors is 71.
the misclassification error for each k value is :  [0.13  0.127 0.125
0.124 0.122 0.121 0.119 0.118 0.118 0.117 0.116 0.116
 0.116 0.116 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.11
5
 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.11
5
 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115]
```



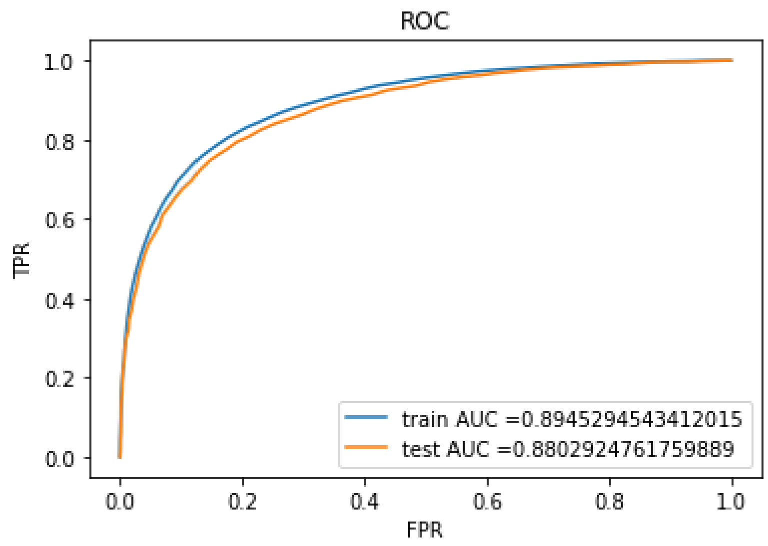
==

In [54]:
```
# instantiate learning model k = optimal_k
knn_optimal2 =  KNeighborsClassifier(n_neighbors=optimal_k2,algorithm=
'brute')
knn_optimal2.fit(sent_vectors_tr,y_tr)
pred2 = knn_optimal2.predict(sent_vectors_te)
# evaluate accuracy
acc = accuracy_score(y_te, pred2) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k2, acc))
# auc = roc_auc_score(y_te, pred2)
# print('AUC: %.3f' % auc)
```

```
The accuracy of the knn classifier for k = 71 is 80.050000%
```

In [55]:
```
#plotting Auc
tr_fpr,tr_tpr,threshold = roc_curve(y_tr, knn_optimal2.predict_proba(se
```

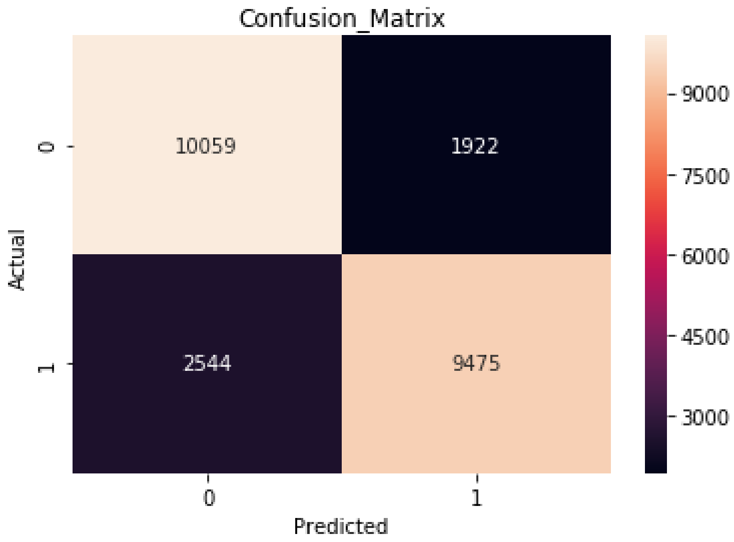
```
nt_vectors_tr)[:,1])
te_fpr,te_tpr,threshold =  roc_curve(y_te, knn_optimal2.predict_proba(s
ent_vectors_te)[:,1])
AUC3=str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr,tr_tpr,label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr,te_tpr,label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



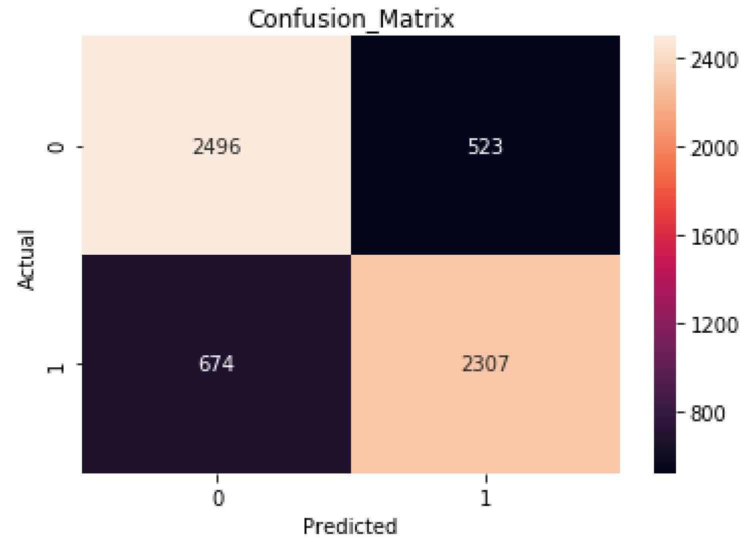
In [56]:

```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(y_tr, knn_optimal2.predict(sent_vectors_tr
))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
```

```
plt.ylabel("Actual")
plt.show()
```



In [57]:
```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(y_te, knn_optimal2.predict(sent_vectors_te
))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [58]:
```
print('='*50)
print(classification_report(y_te,pred2))
print('='*50)
```

```
==================================================
              precision    recall  f1-score   support

           0       0.79      0.83      0.81      3019
           1       0.82      0.77      0.79      2981

   micro avg       0.80      0.80      0.80      6000
   macro avg       0.80      0.80      0.80      6000
weighted avg       0.80      0.80      0.80      6000

==================================================
```

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

In [59]:
```
model = TfidfVectorizer()
```

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

In [60]:

```
i=0
list_of_sentance_tr=[]
for sentance in x_tr:
    list_of_sentance_tr.append(sentance.split())

# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.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_tr = []; # the tfidf-w2v for each sentence/review is
 stored in this list
row=0;
for sent in tqdm(list_of_sentance_tr): # 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/r
eview
    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_tr.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████████████
█████████| 24000/24000 [04:00<00:00, 99.76it/s]
```

In [61]:

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

# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.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 = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    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_cv.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████████████████████
████████████| 6000/6000 [01:00<00:00, 91.32it/s]
```

In [62]:

```
i=0
list_of_sentance_te=[]
for sentance in x_te:
```

```
    list_of_sentance_te.append(sentance.split())

# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.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_te = []; # the tfidf-w2v for each sentence/review is
 stored in this list
row=0;
for sent in tqdm(list_of_sentance_te): # 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/r
eview
    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_te.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████████████████████████
███████████| 6000/6000 [01:05<00:00, 91.84it/s]
```

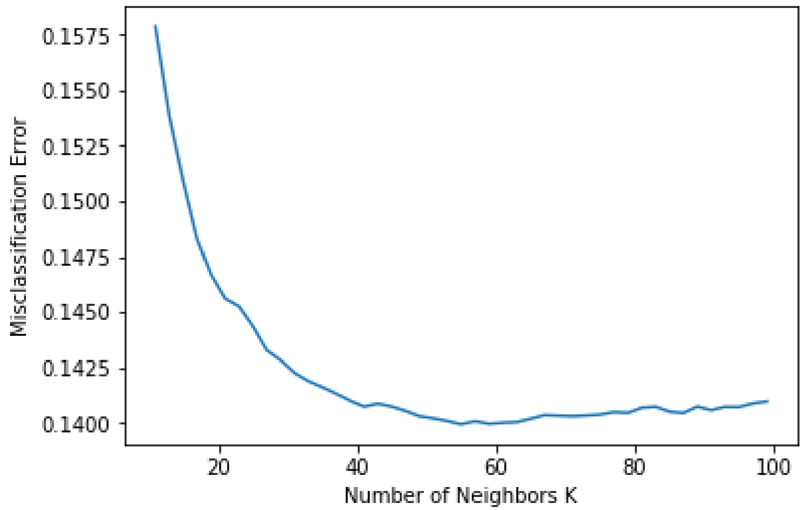
In [63]:

```
my_list = list(range(10,100))
neighbors = list(filter(lambda x : x%2!=0,my_list))
cv_scores = []
for k in tqdm(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    scores = cross_val_score(knn, tfidf_sent_vectors_tr , y_tr, cv=10,
scoring='roc_auc')
```

```
    cv_scores.append(scores.mean())
# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k3 = neighbors[MSE.index(min(MSE))]

plt.plot(neighbors, MSE)

# for xy in zip(neighbors, np.round(MSE,3)):
#     plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

print('='*50)
print('\nThe optimal number of neighbors is %d.' % optimal_k3)
print("the misclassification error for each k value is : ", np.round(MS
E,3))
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
print('='*50)
```

```
100%|██████████████████████████████████████████████████████████████████
████████████████| 45/45 [11:25<00:00, 15.42s/it]
```

```
==================================================

The optimal number of neighbors is 55.
the misclassification error for each k value is :  [0.158 0.154 0.151
0.148 0.147 0.146 0.145 0.144 0.143 0.143 0.142 0.142
 0.142 0.141 0.141 0.141 0.141 0.141 0.141 0.14  0.14  0.14  0.14  0.14
 0.14  0.14  0.14  0.14  0.14  0.14  0.14  0.14  0.14  0.14  0.14  0.14
1
 0.141 0.141 0.14  0.141 0.141 0.141 0.141 0.141 0.141]
```



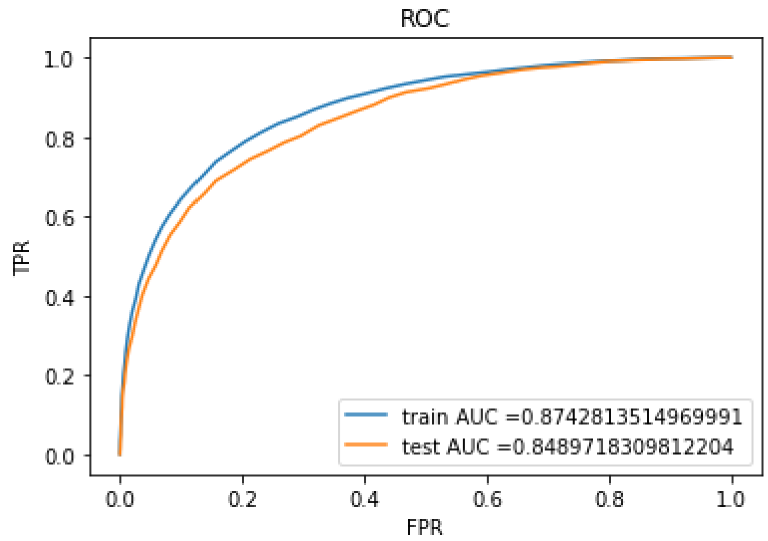

```
==================================================
```

In [64]:
```
# instantiate learning model k = optimal_k
knn_optimal3 =  KNeighborsClassifier(n_neighbors=optimal_k3,algorithm=
'brute')
knn_optimal3.fit(tfidf_sent_vectors_tr,y_tr)
pred3 = knn_optimal3.predict(tfidf_sent_vectors_te)
# evaluate accuracy
acc = accuracy_score(y_te, pred3) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k3, acc))
# auc = roc_auc_score(y_te, pred3)
# print('AUC: %.3f' % auc)
```

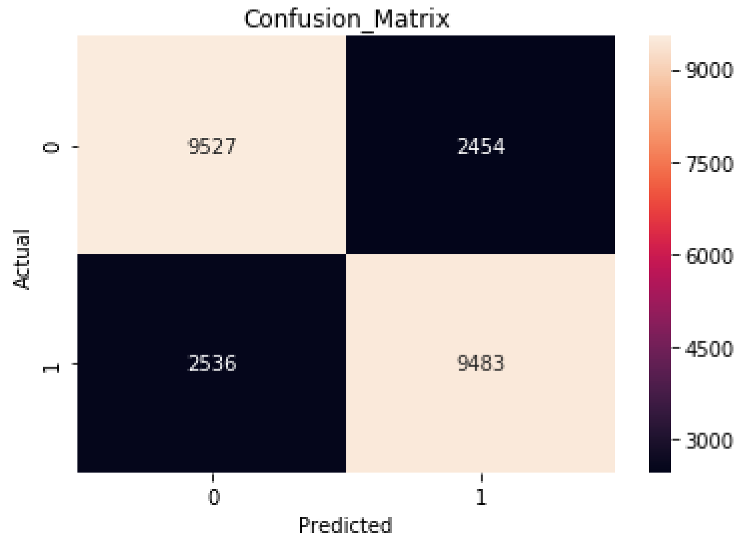
```
The accuracy of the knn classifier for k = 55 is 76.133333%
```

In [65]:
```
#plotting Auc
tr_fpr,tr_tpr,threshold = roc_curve(y_tr, knn_optimal3.predict_proba(tf
idf_sent_vectors_tr)[:,1])
```

```
te_fpr,te_tpr,threshold =  roc_curve(y_te, knn_optimal3.predict_proba(t
fidf_sent_vectors_te)[:,1])
AUC4=str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr,tr_tpr,label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr,te_tpr,label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

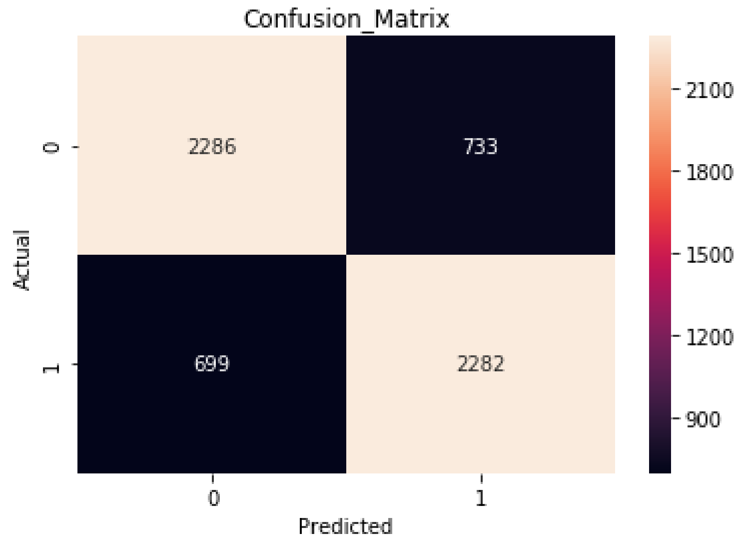


In [66]:
```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(y_tr, knn_optimal3.predict(tfidf_sent_vecto
rs_tr))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [67]:

```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(y_te, knn_optimal3.predict(tfidf_sent_vecto
rs_te))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [68]:
```
print('='*50)
print(classification_report(y_te,pred3))
print('='*50)
```

```
==================================================
              precision    recall  f1-score   support

           0       0.77      0.76      0.76      3019
           1       0.76      0.77      0.76      2981

   micro avg       0.76      0.76      0.76      6000
   macro avg       0.76      0.76      0.76      6000
weighted avg       0.76      0.76      0.76      6000

==================================================
```

[5.2] Applying KNN kd-tree

In [69]:
```
d_po = final[final['Score'] == 1].sample(n=5000)
```

```
d_ne = final[final['Score'] == 0].sample(n=5000)
final_kd = pd.concat([d_po,d_ne])
final_kd.shape
```

Out[69]: (10000, 11)

In [70]:
```
p = final_kd['Cleaned_Text'].values
q = final_kd['Score'].values
print(type(p),type(q))
print(p.shape,q.shape)
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
(10000,) (10000,)
```

In [71]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [72]:
```
p_tr,p_te,q_tr,q_te = train_test_split(p,q,test_size=0.2,random_state=1
2)
p_tr,p_cv,q_tr,q_cv = train_test_split(p,q,test_size=0.2,random_state=1
2)
print('='*50)
print(p_tr.shape,q_tr.shape)
print(p_te.shape,q_te.shape)
print(p_cv.shape,q_cv.shape)
print('='*50)
```

```
==================================================
(8000,) (8000,)
(2000,) (2000,)
(2000,) (2000,)
==================================================
```

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

In [73]:
```
vectorizer = CountVectorizer(min_df = 10,max_features = 500)
vectorizer.fit(p_tr)
p_tr_bow = vectorizer.transform(p_tr)
p_cv_bow = vectorizer.transform(p_cv)
p_te_bow = vectorizer.transform(p_te)
print('='*50)
print(p_tr_bow.shape,q_tr.shape)
print(p_te_bow.shape,q_te.shape)
print(p_cv_bow.shape,q_cv.shape)
print('='*50)
```

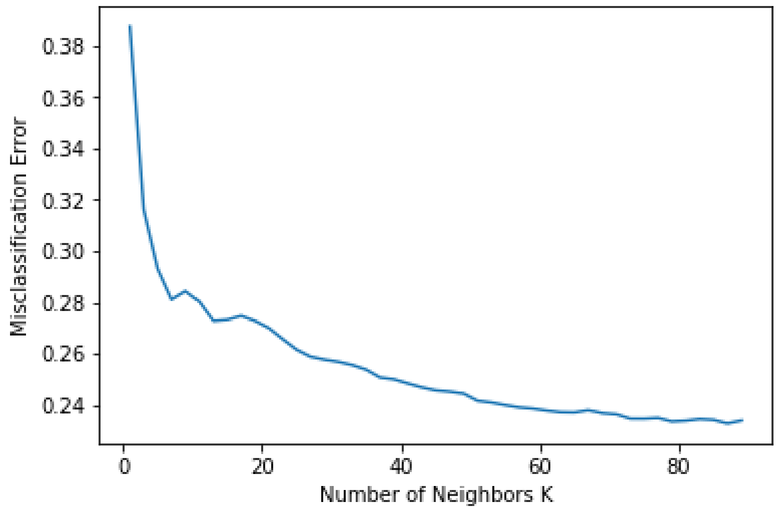
```
==================================================
(8000, 500) (8000,)
(2000, 500) (2000,)
(2000, 500) (2000,)
==================================================
```

In [74]:
```
my_list = list(range(1,90))
neighbors = list(filter(lambda x : x%2!=0,my_list))
cv_scores = []
for k in tqdm(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    scores = cross_val_score(knn, p_tr_bow.todense(), q_tr, cv=10, scor
ing='roc_auc')
    cv_scores.append(scores.mean())
# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k4 = neighbors[MSE.index(min(MSE))]

plt.plot(neighbors, MSE)

# for xy in zip(neighbors, np.round(MSE,3)):
#     plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
```

```
print('='*50)
print('\nThe optimal number of neighbors is %d.' % optimal_k4)
print("the misclassification error for each k value is : ", np.round(MS
E,3))
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
print('='*50)
```

```
100%|██████████████████████████████████████████████████
████████████| 45/45 [39:57<00:00, 53.10s/it]
```

```
==================================================

The optimal number of neighbors is 87.
the misclassification error for each k value is :  [0.387 0.316 0.293
0.281 0.284 0.28  0.273 0.273 0.275 0.273 0.27  0.266
 0.262 0.259 0.258 0.257 0.256 0.254 0.251 0.25  0.248 0.247 0.246 0.24
5
 0.244 0.242 0.241 0.24  0.239 0.239 0.238 0.237 0.237 0.238 0.237 0.23
6
 0.235 0.235 0.235 0.234 0.234 0.234 0.234 0.233 0.234]
```

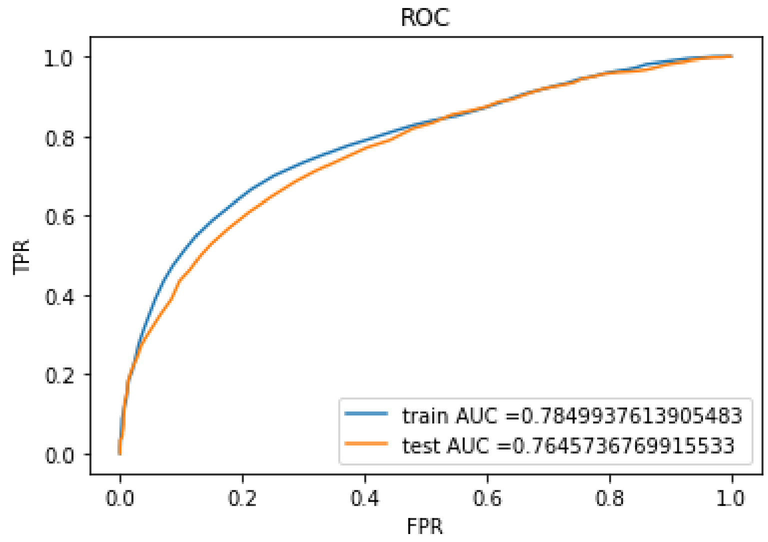


==

In [75]:
```
knn_optimal4 = KNeighborsClassifier(n_neighbors = optimal_k4 , algorith
m = 'kd_tree')
knn_optimal4.fit(p_tr_bow.todense(),q_tr)
pred4 = knn_optimal4.predict(p_te_bow.todense())
# evaluate accuracy
acc = accuracy_score(q_te, pred4) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k4, acc))
# auc = roc_auc_score(q_te, pred4)
# print('AUC: %.3f' % auc)
```

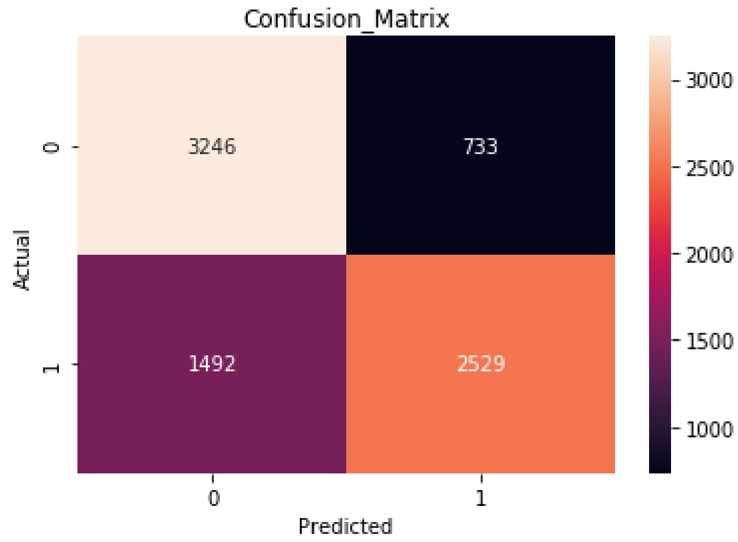
```
The accuracy of the knn classifier for k = 87 is 70.050000%
```

In [76]:
```
tr_fpr,tr_tpr,threshold = roc_curve(q_tr,knn_optimal4.predict_proba(p_t
r_bow.todense())[:,1])
te_fpr,te_tpr,threshold = roc_curve(q_te,knn_optimal4.predict_proba(p_t
e_bow.todense())[:,1])
AUC5=str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr, tr_tpr, label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr, te_tpr, label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

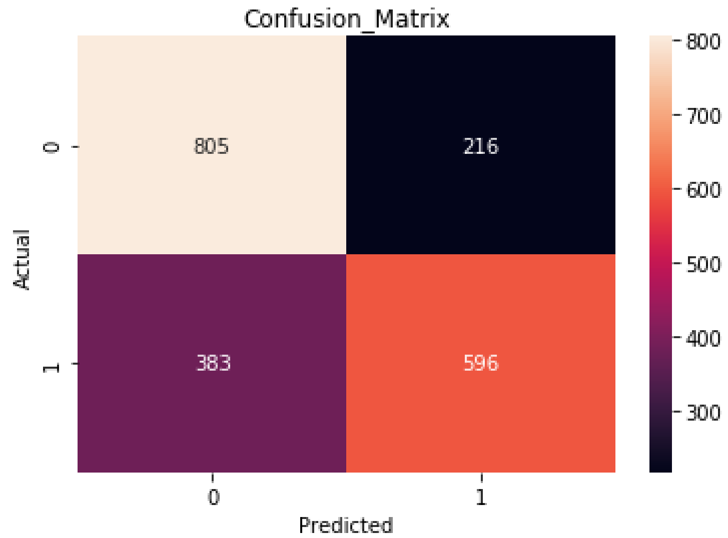


In [77]:

```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(q_tr, knn_optimal4.predict(p_tr_bow.todense
()))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [78]:
```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(q_te, knn_optimal4.predict(p_te_bow.todense
()))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [79]:
```
print('='*50)
print(classification_report(q_te,pred4))
print('='*50)
```

```
==================================================
              precision    recall  f1-score   support

           0       0.68      0.79      0.73      1021
           1       0.73      0.61      0.67       979

   micro avg       0.70      0.70      0.70      2000
   macro avg       0.71      0.70      0.70      2000
weighted avg       0.71      0.70      0.70      2000

==================================================
```

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

In [80]:
```
tfidfvect = TfidfVectorizer(min_df=10,max_features = 500)
```

```
tfidfvect.fit(p_tr)
p_tr_tfidf = tfidfvect.transform(p_tr)
p_cv_tfidf = tfidfvect.transform(p_cv)
p_te_tfidf = tfidfvect.transform(p_te)
print('='*50)
print(p_tr_tfidf.shape,q_tr.shape)
print(p_cv_tfidf.shape,q_cv.shape)
print(p_te_tfidf.shape,q_te.shape)
print('='*50)
```

```
==================================================
(8000, 500) (8000,)
(2000, 500) (2000,)
(2000, 500) (2000,)
==================================================
```

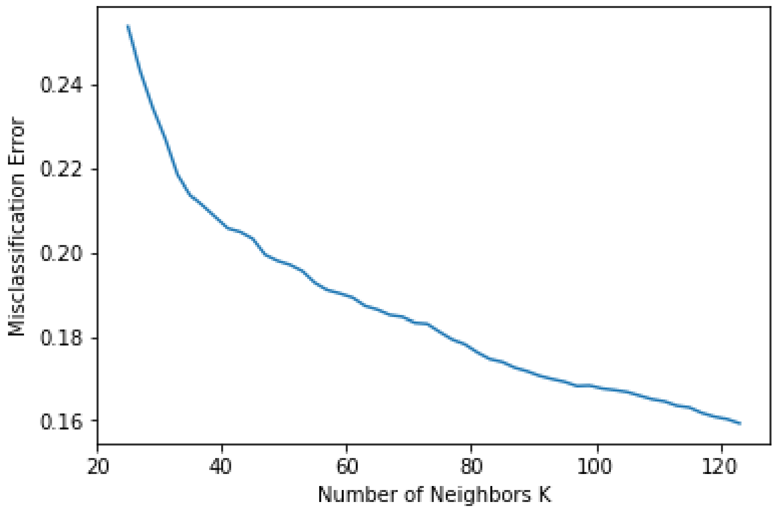
In [81]:

```
my_list = list(range(25,125))
neighbors = list(filter(lambda x : x%2!=0,my_list))
cv_scores = []
for k in tqdm(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    scores = cross_val_score(knn, p_tr_tfidf.todense(), q_tr, cv=10, sc
oring='roc_auc')
    cv_scores.append(scores.mean())
# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k5 = neighbors[MSE.index(min(MSE))]

plt.plot(neighbors, MSE)

# for xy in zip(neighbors, np.round(MSE,3)):
#     plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

print('='*50)
print('\nThe optimal number of neighbors is %d.' % optimal_k5)
print("the misclassification error for each k value is : ", np.round(MS
E,3))
```

```python
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
print('='*50)
```

```
100%|████████████████████████████████████████| 50/50 [44:50<00:00, 53.27s/it]
```

```
==================================================

The optimal number of neighbors is 123.
the misclassification error for each k value is :  [0.254 0.243 0.234
0.227 0.218 0.214 0.211 0.208 0.206 0.205 0.203 0.199
 0.198 0.197 0.196 0.193 0.191 0.19  0.189 0.187 0.186 0.185 0.185 0.18
3
 0.183 0.181 0.179 0.178 0.176 0.175 0.174 0.173 0.172 0.171 0.17  0.16
9
 0.168 0.168 0.168 0.167 0.167 0.166 0.165 0.165 0.164 0.163 0.162 0.16
1
 0.16  0.159]
```




```
==================================================
```

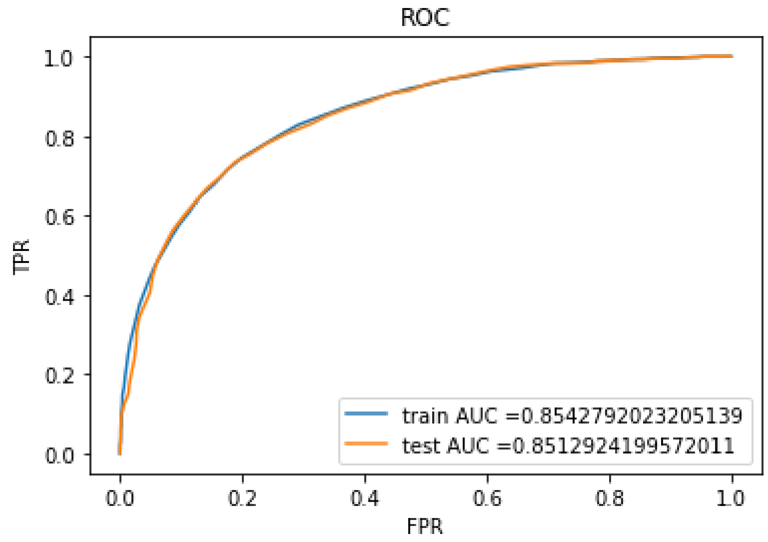
In [82]:

```
knn_optimal5 = KNeighborsClassifier(n_neighbors = optimal_k5 , algorith
m = 'kd_tree')
knn_optimal5.fit(p_tr_tfidf.todense(),q_tr)
pred5 = knn_optimal5.predict(p_te_tfidf.todense())
# evaluate accuracy
acc = accuracy_score(q_te, pred5) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k5, acc))
# auc = roc_auc_score(q_te, pred5)
# print('AUC: %.3f' % auc)
```

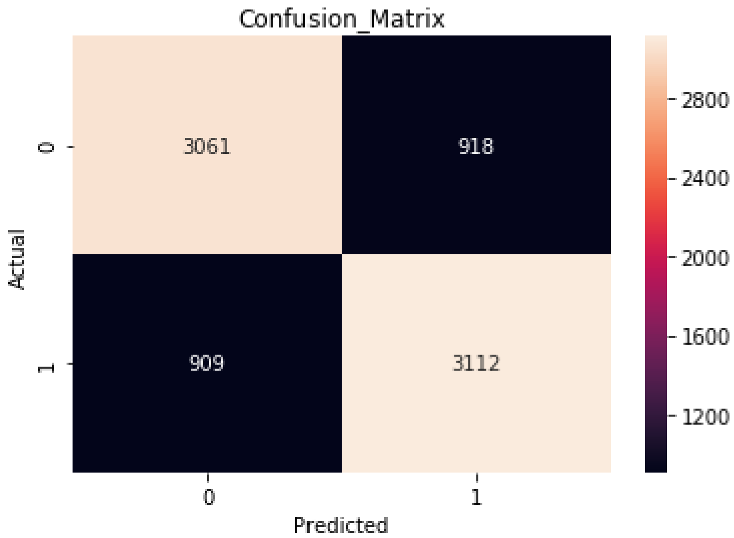
```
The accuracy of the knn classifier for k = 123 is 77.000000%
```

In [83]:

```
tr_fpr,tr_tpr,threshold = roc_curve(q_tr,knn_optimal5.predict_proba(p_t
r_tfidf.todense())[:,1])
te_fpr,te_tpr,threshold = roc_curve(q_te,knn_optimal5.predict_proba(p_t
e_tfidf.todense())[:,1])
AUC6=str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr, tr_tpr, label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr, te_tpr, label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

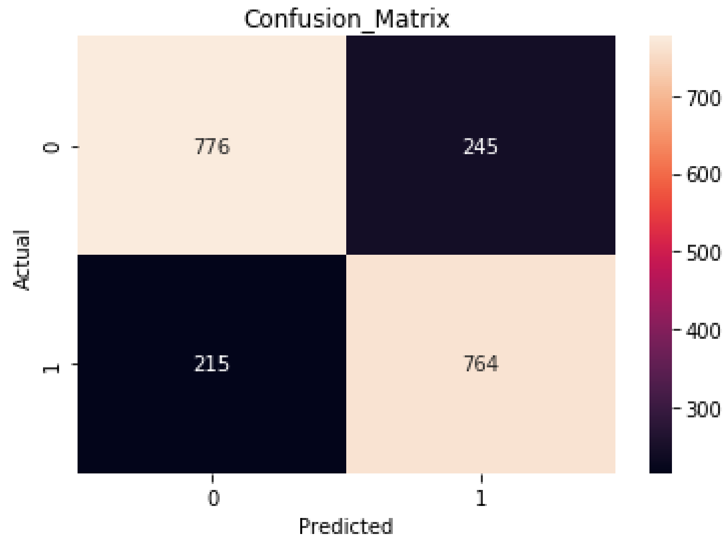


In [84]:
```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(q_tr, knn_optimal5.predict(p_tr_tfidf.toden
se()))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [85]:

```
#Confusion Matrix
import seaborn as sb
con_matr = confusion_matrix(q_te, knn_optimal5.predict(p_te_tfidf.toden
se()))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [86]:
```
print('='*50)
print(classification_report(q_te,pred5))
print('='*50)
```

```
==================================================
              precision    recall  f1-score   support

           0       0.78      0.76      0.77      1021
           1       0.76      0.78      0.77       979

   micro avg       0.77      0.77      0.77      2000
   macro avg       0.77      0.77      0.77      2000
weighted avg       0.77      0.77      0.77      2000

==================================================
```

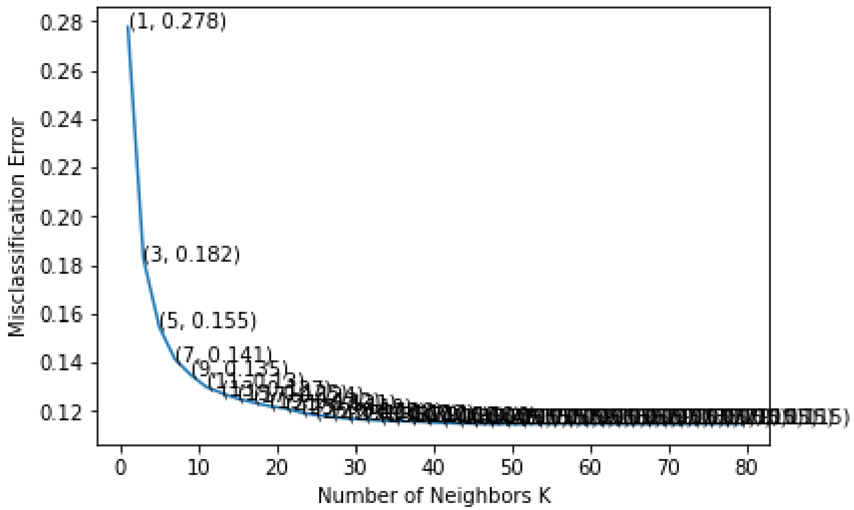
[5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

In [87]:
```
my_list = list(range(1,80))
```

```
neighbors = list(filter(lambda x : x%2!=0,my_list))
cv_scores = []
for k in tqdm(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    scores = cross_val_score(knn, sent_vectors_tr , y_tr, cv=10, scorin
g='roc_auc')
    cv_scores.append(scores.mean())
# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k6 = neighbors[MSE.index(min(MSE))]

plt.plot(neighbors, MSE)

for xy in zip(neighbors, np.round(MSE,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

print('='*50)
print('\nThe optimal number of neighbors is %d.' % optimal_k6)
print("the misclassification error for each k value is : ", np.round(MS
E,3))
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
print('='*50)
```

```
100%|██████████████████████████████████████████████████████████████████
████████████████| 40/40 [41:19<00:00, 64.99s/it]
```

```
==================================================

The optimal number of neighbors is 71.
the misclassification error for each k value is :  [0.278 0.182 0.155
0.141 0.135 0.13  0.127 0.125 0.124 0.122 0.121 0.119
 0.118 0.118 0.117 0.117 0.116 0.116 0.116 0.116 0.115 0.115 0.115 0.11
5
 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.11
5
 0.115 0.115 0.115 0.115]
```



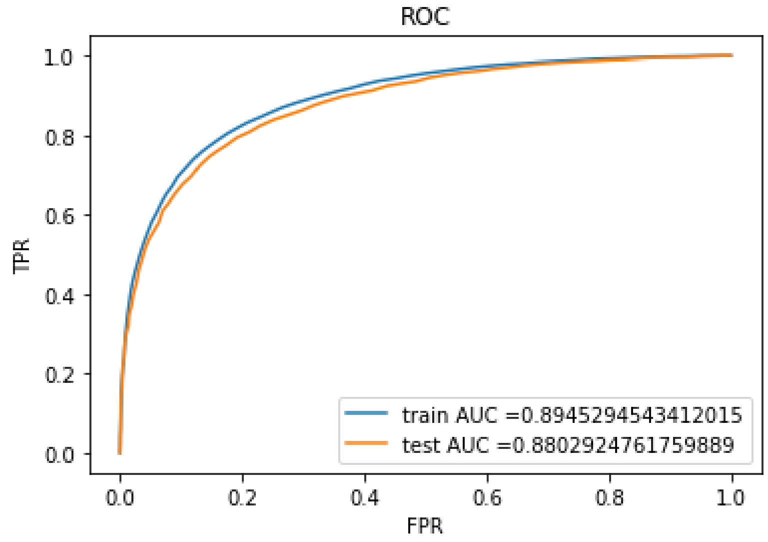
==

In [88]:
```
knn_optimal6 = KNeighborsClassifier(n_neighbors = optimal_k6 , algorith
m = 'kd_tree')
knn_optimal6.fit(sent_vectors_tr,y_tr)
pred6 = knn_optimal6.predict(sent_vectors_te)
# evaluate accuracy
acc = accuracy_score(y_te, pred6) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k6, acc))
# auc = roc_auc_score(y_te, pred6)
# print('AUC: %.3f' % auc)
```

```
The accuracy of the knn classifier for k = 71 is 80.050000%
```

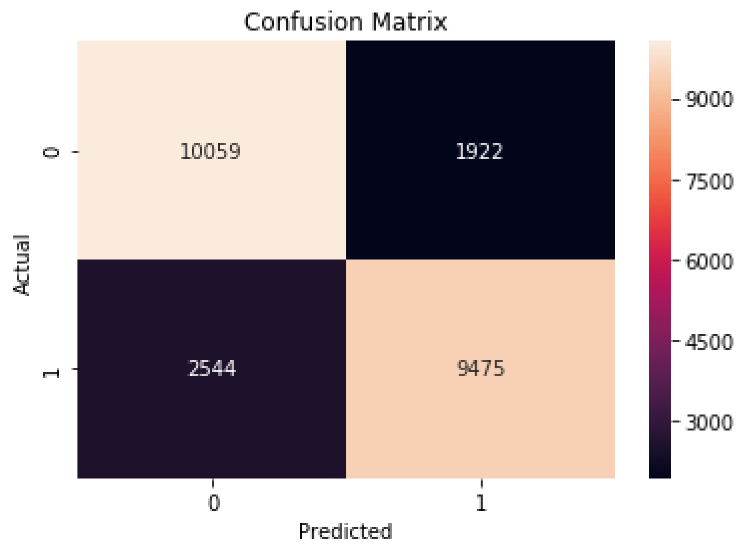
In [89]:
```
tr_fpr, tr_tpr, threshold = roc_curve(y_tr, knn_optimal6.predict_proba(
sent_vectors_tr)[:,1])
te_fpr, te_tpr, threshold = roc_curve(y_te, knn_optimal6.predict_proba(
```

```
sent_vectors_te)[:,1])
AUC7=str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr, tr_tpr, label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr, te_tpr, label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



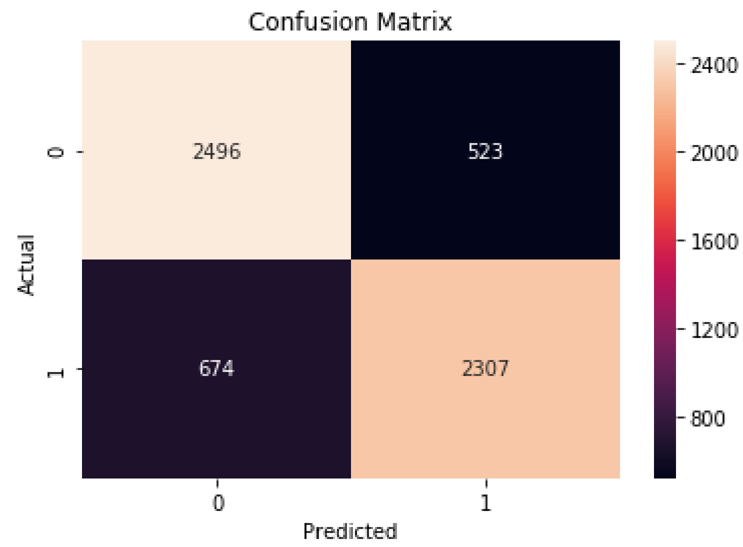
In [90]:

```
#Confusion Matrix
import seaborn as sb
conf_matrix = confusion_matrix(y_tr, knn_optimal6.predict(sent_vectors_
tr))
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(conf_matrix, index=class_label, columns=c
lass_label)
sb.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [91]:

```
#Confusion Matrix
import seaborn as sb
conf_matrix = confusion_matrix(y_te, knn_optimal6.predict(sent_vectors_
te))
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(conf_matrix, index=class_label, columns=c
lass_label)
sb.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [92]:

```
print('='*50)
print(classification_report(y_te, pred6))
print('='*50)
```

```
==================================================
              precision    recall  f1-score   support

           0       0.79      0.83      0.81      3019
           1       0.82      0.77      0.79      2981

   micro avg       0.80      0.80      0.80      6000
   macro avg       0.80      0.80      0.80      6000
weighted avg       0.80      0.80      0.80      6000

==================================================
```

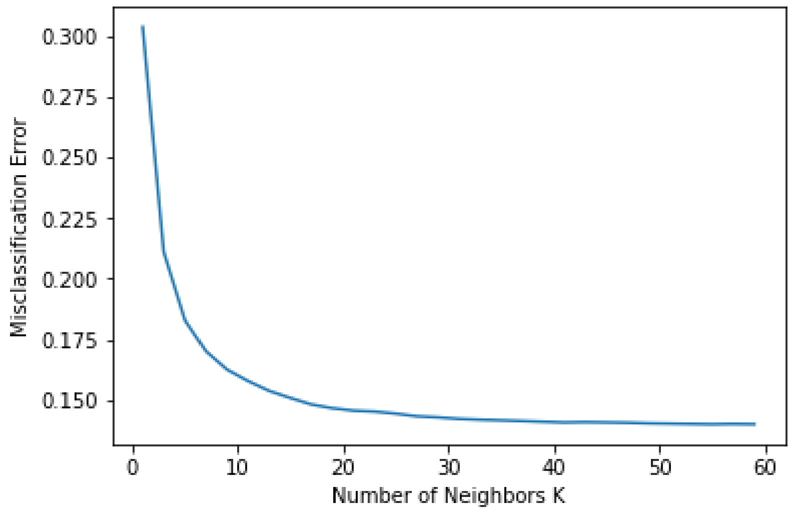
[5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

In [93]:

```python
my_list = list(range(1,60))
neighbors = list(filter(lambda x : x%2!=0,my_list))
cv_scores = []
for k in tqdm(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    scores = cross_val_score(knn, tfidf_sent_vectors_tr , y_tr, cv=10, 
scoring='roc_auc')
    cv_scores.append(scores.mean())
# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k7 = neighbors[MSE.index(min(MSE))]

plt.plot(neighbors, MSE)

# for xy in zip(neighbors, np.round(MSE,3)):
#     plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

print('='*50)
print('\nThe optimal number of neighbors is %d.' % optimal_k7)
print("the misclassification error for each k value is : ", np.round(MS
E,3))
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
print('='*50)
```

```
100%|██████████████████████████████████████████████████████████████████
████████████████| 30/30 [26:51<00:00, 58.39s/it]
```

```
==================================================

The optimal number of neighbors is 55.
the misclassification error for each k value is :  [0.304 0.211 0.183 
0.17  0.163 0.158 0.154 0.151 0.148 0.147 0.146 0.145
 0.144 0.143 0.143 0.142 0.142 0.142 0.141 0.141 0.141 0.141 0.141 0.14
1
 0.14  0.14  0.14  0.14  0.14  0.14 ]
```



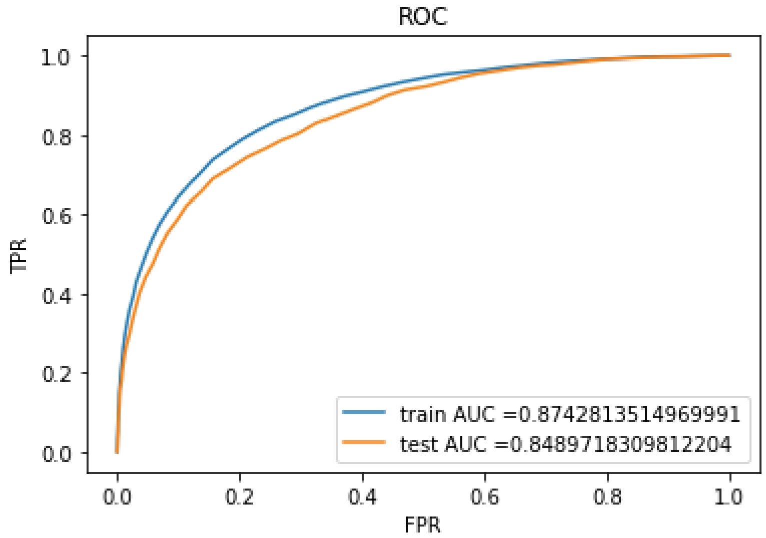

```
==================================================
```

In [94]:
```
knn_optimal7 = KNeighborsClassifier(n_neighbors = optimal_k7 , algorith
m = 'kd_tree')
knn_optimal7.fit(tfidf_sent_vectors_tr,y_tr)
pred7 = knn_optimal7.predict(tfidf_sent_vectors_te)
# evaluate accuracy
acc = accuracy_score(y_te, pred7) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k7, acc))
# auc = roc_auc_score(y_te, pred7)
# print('AUC: %.3f' % auc)
```

```
The accuracy of the knn classifier for k = 55 is 76.133333%
```

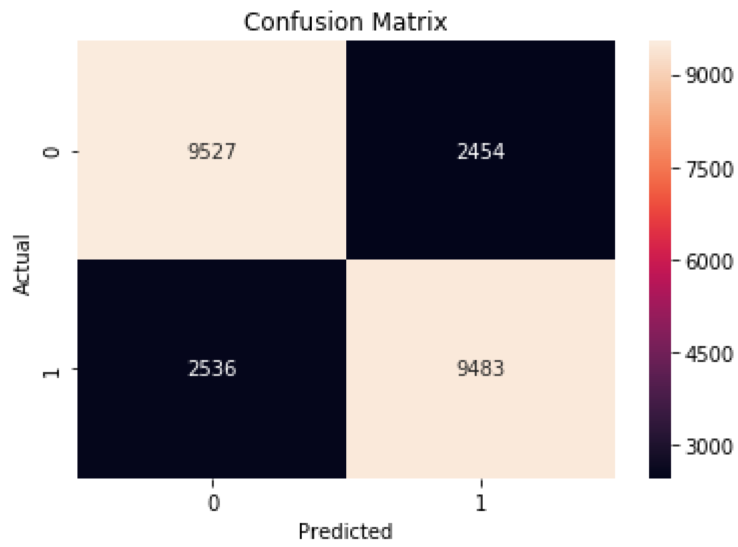
In [95]:
```
tr_fpr, tr_tpr, threshold = roc_curve(y_tr, knn_optimal7.predict_proba(
tfidf_sent_vectors_tr)[:,1])
te_fpr, te_tpr, threshold = roc_curve(y_te, knn_optimal7.predict_proba(
tfidf_sent_vectors_te)[:,1])
```

```
AUC8=str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr, tr_tpr, label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr, te_tpr, label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

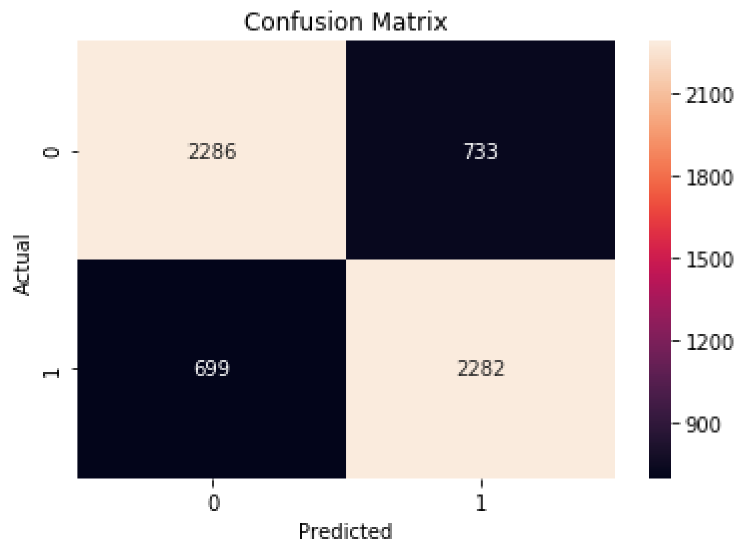


In [96]:

```
#Confusion Matrix
import seaborn as sb
conf_matrix = confusion_matrix(y_tr, knn_optimal7.predict(tfidf_sent_ve
ctors_tr))
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(conf_matrix, index=class_label, columns=c
lass_label)
sb.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [97]:
```
#Confusion Matrix
import seaborn as sb
conf_matrix = confusion_matrix(y_te, knn_optimal7.predict(tfidf_sent_ve
ctors_te))
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(conf_matrix, index=class_label, columns=c
lass_label)
sb.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [98]:
```
print('='*50)
print(classification_report(y_te, pred7))
print('='*50)
```

```
==================================================
              precision    recall  f1-score   support

           0       0.77      0.76      0.76      3019
           1       0.76      0.77      0.76      2981

   micro avg       0.76      0.76      0.76      6000
   macro avg       0.76      0.76      0.76      6000
weighted avg       0.76      0.76      0.76      6000

==================================================
```

[6] Conclusions

In [99]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
comparison = PrettyTable()
comparison.field_names = ["Vectorizer", "Model", "Hyperparameter", "AU
C"]
comparison.add_row(["BOW", 'brute', optimal_k, np.round(float(AUC1),3
)])
comparison.add_row(["TFIDF", 'brute', optimal_k1, np.round(float(AUC2),
3)])
comparison.add_row(["AVG W2V", 'brute', optimal_k2, np.round(float(AUC3
),3)])
comparison.add_row(["Weighted W2V", 'brute', optimal_k3,np.round(float(
AUC4),3)])
comparison.add_row(["BOW", 'kd_tree', optimal_k4, np.round(float(AUC5),
3)])
comparison.add_row(["TFIDF", 'kd_tree', optimal_k5, np.round(float(AUC6
),3)])
comparison.add_row(["AVG W2V", 'kd_tree', optimal_k6, np.round(float(AU
C7),3)])
comparison.add_row(["Weighted W2V", 'kd_tree', optimal_k7, np.round(flo
at(AUC8),3)])
print(comparison)
```

```
+--------------+---------+----------------+-------+
|  Vectorizer  |  Model  | Hyperparameter |  AUC  |
+--------------+---------+----------------+-------+
|     BOW      |  brute  |       73       | 0.718 |
|    TFIDF     |  brute  |      199       | 0.893 |
|   AVG W2V    |  brute  |       71       |  0.88 |
| Weighted W2V |  brute  |       55       | 0.849 |
|     BOW      | kd_tree |       87       | 0.765 |
|    TFIDF     | kd_tree |      123       | 0.851 |
|   AVG W2V    | kd_tree |       71       |  0.88 |
| Weighted W2V | kd_tree |       55       | 0.849 |
+--------------+---------+----------------+-------+
```

In [ ]: