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 [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

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

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

from tqdm import tqdm
import os
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", con)

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

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

```
Number of data points in our data (500000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

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

In [5]:

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

Out[5]:

	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 [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOAC QUADRA VANI WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOAC QUADRA VANI WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOAC QUADRA VANI WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOAC QUADRA VANI WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOAC QUADRA VANI WAFE

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

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

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

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

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

In [8]:

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

In [9]:

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

Out[9]:

```
(348262, 10)
```

In [10]:

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

Out[10]:

```
69.6524
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [11]:

```
display= pd.read_sql_query("""
```

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bough This fo My Son a College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

In [12]:

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

In [13]:

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

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

```
(348260, 10)
```

Out[13]:

```
1    293516
0     54744
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [14]:

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

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

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

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

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and hugged it
when told it was his to keep and he did not have to return it to the library.
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Arabica.  My vote is definitely wit
h the 100% Arabica.  The flavor has more bite and flavor (much more like European coffee than American)
.
==================================================
This is a great product. It is very healthy for all of our dogs, and it is the first food that they all
love to eat. It helped my older dog lose weight and my 10 year old lab gain the weight he needed to be
healthy.
==================================================
I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a tennis pa
rty, perfect! They were the size of malted milk balls. Unfortunately, they arrived 3 days after the par
ty. The caveat here is, not everything from Amazon may arrive at an impressive 2 or 3 days. This shipme
nt took 8 days from the Candy/Cosmetic Depot back east to southern California.
==================================================
```

In [15]:

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

print(sent_0)
```

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and hugged it
when told it was his to keep and he did not have to return it to the library.
```

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-elem
ent
from bs4 import BeautifulSoup

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

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

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

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

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and hugged it
when told it was his to keep and he did not have to return it to the library.
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Arabica.  My vote is definitely wit
h the 100% Arabica.  The flavor has more bite and flavor (much more like European coffee than American)
.
==================================================
This is a great product. It is very healthy for all of our dogs, and it is the first food that they all
love to eat. It helped my older dog lose weight and my 10 year old lab gain the weight he needed to be
healthy.
==================================================
```

```
I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a tennis pa
rty, perfect! They were the size of malted milk balls. Unfortunately, they arrived 3 days after the par
ty. The caveat here is, not everything from Amazon may arrive at an impressive 2 or 3 days. This shipme
nt took 8 days from the Candy/Cosmetic Depot back east to southern California.
```

In [17]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [18]:

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

```
This is a great product. It is very healthy for all of our dogs, and it is the first food that they all
love to eat. It helped my older dog lose weight and my 10 year old lab gain the weight he needed to be
healthy.
==================================================
```

In [19]:

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

```
This book was purchased as a birthday gift for a  year old boy. He squealed with delight and hugged it
when told it was his to keep and he did not have to return it to the library.
```

In [20]:

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

```
This is a great product It is very healthy for all of our dogs and it is the first food that they all l
ove to eat It helped my older dog lose weight and my 10 year old lab gain the weight he needed to be he
althy
```

In [21]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself'
, \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 't
heir',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'whil
e', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'bef
```

```
ore', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'a
gain', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each
', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', '
m', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn
't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't",
'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't",
'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

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

```
100%|██████████████████████████████████████████████████| 348260/348260 [02:40<00:
00, 2166.23it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'great product healthy dogs first food love eat helped older dog lose weight year old lab gain weight n
eeded healthy'
```

In [24]:

```
final["Cleaned_Text"] = preprocessed_reviews
```

[3.2] Preprocessing Review Summary

In [ ]:

```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [ ]:

```
#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])
```

[4.2] Bi-Grams and n-Grams

[4.2] Bi-Grams and n-Grams.

In [ ]:

```
#bi-gram, tri-gram and n-gram

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

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

[4.3] TF-IDF

In [ ]:

```
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.get_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])
```

[4.4] Word2Vec

In [ ]:

```
# 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 [ ]:

```
# Using Google News Word2Vectors

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=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_train_w2v = True, to train your own w2v ")
```

In [ ]:

```
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])
```

[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, you might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

[4.4.1.2] TFIDF weighted W2v

In [ ]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [ ]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

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

```
row += 1
```

[5] Assignment 7: SVM

1. **Apply SVM 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. **Procedure**
    - You need to work with 2 versions of SVM
        - Linear kernel
        - RBF kernel
    - When you are working with linear kernel, use SGDClassifier' with hinge loss because it is computationally less expensive.
    - When you are working with 'SGDClassifier' with hinge loss and trying to find the AUC score, you would have to use [CalibratedClassifierCV]()
    - Similarly, like kdtree of knn, when you are working with RBF kernel it's better to reduce the number of dimensions. You can put min_df = 10, max_features = 500 and consider a sample size of 40k points.

3. **Hyper paramter tuning (find best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**
    - 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. **Feature importance**
    - When you are working on the linear kernel with BOW or TFIDF please print the top 10 best features for each of the positive and negative classes.

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

6. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
    - 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. Please visualize your confusion matrices using [seaborn heatmaps.]()

7. **[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.]()

Applying SVM

[5.1] Linear SVM

In [25]:

```
final["Time"] = pd.to_datetime(final["Time"],unit="s")
final.sort_values(by="Time")
```

Out[25]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	1999-10-08
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	1999-10-25
417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	1	1999-12-02
346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	1	1999-12-06
417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	1	2000-01-03
346116	374422	B00004CI84	A1048CYU0OV4O8	Judy L. Eans	2	2	1	2000-01-09
346041	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19	23	0	2000-01-19
70688	76882	B00002N8SM	A32DW342WBJ6BX	Buttersugar	0	0	1	2000-01-24
346141	374450	B00004CI84	ACJR7EQF9S6FP	Jeremy Robertson	2	3	1	2000-02-26
417883	451903	B00004CXX9	A2DEE7F9XKP3ZR	jerome	0	1	1	2000-06-03
346094	374400	B00004CI84	A2DEE7F9XKP3ZR	jerome	0	3	1	2000-06-03
1146	1245	B00002Z754	A29Z5PI9BW2PU3	Robbie	7	7	1	2000-06-23
1145	1244	B00002Z754	A3B8RCEI0FXFI6	B G Chase	10	10	1	2000-06-29
121041	131217	B00004RAMX	A5NQLNC6QPGSI	Kim Nason	7	8	1	2000-07-31

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
138017	149789	B00004S1C6	A1KXONFPU2XQ5K	Stephanie Manley	26	28	1	2000-08-09
138001	149770	B00004S1C5	A1KXONFPU2XQ5K	Stephanie Manley	8	8	1	2000-08-09
346115	374421	B00004CI84	A1FJOY14X3MUHE	Justin Howard	2	2	1	2000-08-15
346102	374408	B00004CI84	A1GB1Q193DNFGR	Bruce Lee Pullen	5	5	1	2000-10-03
138000	149768	B00004S1C5	A7P76IGRZZBFJ	E. Thompson "Soooooper Genius"	18	18	1	2000-12-05
346078	374383	B00004CI84	A34NBH479RB0E	"dmab6395"	0	1	1	2000-12-19
346054	374358	B00004CI84	A1HWMNSQF14MP8	will@socialaw.com	1	2	1	2000-12-30
138018	149790	B00004S1C6	A1IU7S4HCK1XK0	Joanna Daneman	25	27	1	2001-02-22
417901	451923	B00004CXX9	ANIMV3SPDD8SH	Guy De Federicis	1	12	0	2001-06-11
346037	374339	B00004CI84	AZRJH4JFB59VC	Lynwood E. Hines	21	23	0	2001-08-08
346140	374449	B00004CI84	A3K3YJWV0N54ZO	Joey	2	3	1	2001-09-24
138020	149792	B00004S1C6	A3B5QJVM1TLYJG	Dan Crevier	11	12	1	2001-10-23
346033	374335	B00004CI84	A3L5V40F14R2GP	AARON	0	0	1	2001-10-26
138682	150500	0006641040	A1IJKK6Q1GTEAY	A Customer	2	2	1	2001-12-26
333930	361317	B00005IX96	A3ODTU118FKC5J	Rosemarie E Smith	5	7	1	2002-01-06
346032	374334	B00004CI84	A2HIZRVOKXKZ52	KAY N. FOWLER	0	0	1	2002-02-04
...	...	...	...	...	...	...	...	...
281574	305063	B004CQYODW	AM2ADIDTH4SI4	DC Kids	0	0	1	2012-

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
51204	55627	B004CYLW7A	A2CMS7BYL8BKP1	luv2fly	0	0	0	2012-10-26
156284	169487	B000NM1BHQ	A2JDXKFZ0PFHKU	James W. Shondel	0	0	0	2012-10-26
264610	286810	B001EPPXRK	A1L1T27A33DDP6	Quietwolf	0	0	1	2012-10-26
6548	7178	B004OQLIHK	AKHQMSUORSA91	Pen Name	0	0	1	2012-10-26
78921	85817	B000NCXHFA	ATMTE0BSP7W6S	Lor "Lor"	0	0	1	2012-10-26
89213	97089	B004O8KBK8	A1JPKFGGF128X1	MTNick	0	0	1	2012-10-26
198657	215299	B004O86R2O	ABSMK7ETXKES0	ak	0	0	1	2012-10-26
420088	454287	B004NZ0JIQ	A2K082DNME4G2P	Julie Bartholoma	0	0	1	2012-10-26
222056	240775	B000NBQUNW	A10F34HEJG1QW1	WALLACE CHOY	0	0	1	2012-10-26
273030	295923	B0002DGL26	A3SZ0N66423CE5	Cynthia Z. Wrinn	0	0	0	2012-10-26
347730	376143	B004NB7A1O	A2JDXKFZ0PFHKU	James W. Shondel	0	0	1	2012-10-26
119196	129256	B004MMNNDS	A248RO4GSIWDII	Robert Kawalec	0	0	1	2012-10-26
416667	450607	B004MBJPV8	AG7EPW4BU9SG4	2 Toddlers' Mom "2 Toddlers' Mom"	0	0	0	2012-10-26
198769	215423	B0001UK0CM	A2V8WXAFG1TEOC	ronald (NY)	0	0	1	2012-10-26
43703	47562	B004M0Y8T8	A2QJS6MHTIFSRI	Georgie	0	0	1	2012-10-26
265436	287728	B004J402A6	A2OGEXIK9IG4WU	Beth	0	0	1	2012-10-26
236904	256994	B004ITWDKO	A3934PPUIWRZVX	Chellie H.	0	0	1	2012-10-26

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
135216	146763	B004IN6GVM	A5PFLD64SYXYK	Bruce F	0	0	1	2012-10-26
397181	429454	B004IMYBIS	A33V118ZVFLGVK	Wings42 "David"	0	0	1	2012-10-26
206654	223963	B004H6MM28	AFF6F08FRSYWG	Kentucky Woman "Emily"	0	0	1	2012-10-26
469195	507351	B004H4P6VI	A1V7KJHE8SKJ4G	Christine	0	0	1	2012-10-26
407675	440836	B004GN8NP6	A1GENHMJBIS42V	TechLover70	0	0	1	2012-10-26
481144	520251	B001EQ5IAG	A225UWT247BBBH	J. Blair	0	0	1	2012-10-26
8731	9564	B001EQ5IPQ	AA2104NO2VE8H	Lakshminarayan Iyer	0	0	0	2012-10-26
1005	1089	B004FD13RW	A1BPLP0BKERV	Paul	0	0	1	2012-10-26
55158	59851	B001EQ5KTK	A3GS4GWPIBV0NT	R. Chester "ricki1966"	0	0	1	2012-10-26
433652	468954	B004DMGQKE	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	0	0	1	2012-10-26
214905	232914	B001EQ4OY2	A19N1RDH99TTJZ	Lahn	0	0	1	2012-10-26
250496	271605	B004S0332A	A1P232KPXTRR3C	XtraKargo	0	0	1	2012-10-26

348260 rows × 11 columns

In [26]:

```
tot_svm = final.sample(n=50000)
tot_svm.shape
```

Out[26]:

```
(50000, 11)
```

In [27]:

```
x = tot_svm["Cleaned_Text"].values
y = tot_svm["Score"].values
print(x.shape , y.shape)
```

```
(50000,) (50000,)
```

In [28]:

```
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
```

```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.calibration import CalibratedClassifierCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier

import matplotlib.pyplot as plt
```

In [29]:

```python
x_tr,x_te,y_tr,y_te = train_test_split(x,y,test_size=0.3,random_state=42)
x_tr,x_cv,y_tr,y_cv = train_test_split(x,y,test_size=0.3,random_state=42)
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)
```

```
==================================================
(35000,) (35000,)
(15000,) (15000,)
(15000,) (15000,)
==================================================
```

[5.1.1] Applying Linear SVM on BOW, SET 1

In [30]:

```python
vect = CountVectorizer()
vect.fit(x_tr)
x_tr_bow = vect.transform(x_tr)
x_te_bow = vect.transform(x_te)
x_cv_bow = vect.transform(x_cv)
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)
```

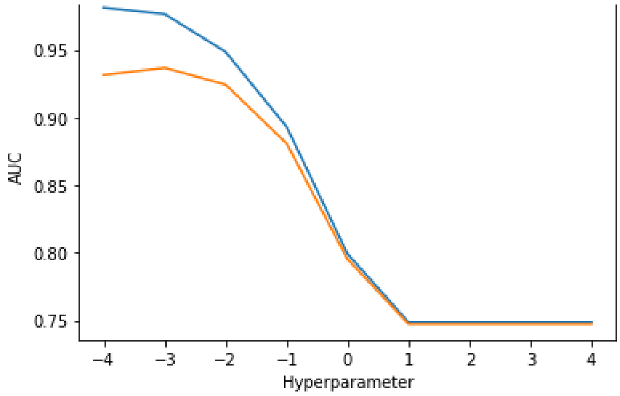
```
==================================================
(35000, 36655) (35000,)
(15000, 36655) (15000,)
(15000, 36655) (15000,)
==================================================
```

In [31]:

```python
C = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]
tr_auc = []
cv_auc = []
for c in tqdm(C):
    m = SGDClassifier(alpha=c,class_weight="balanced") #Default loss is hinge
    svm = CalibratedClassifierCV(m, cv=3)
    svm.fit(x_tr_bow,y_tr)
    probcv = svm.predict_proba(x_cv_bow)[:,1]
    cv_auc.append(roc_auc_score(y_cv,probcv))
    probtr = svm.predict_proba(x_tr_bow)[:,1]
    tr_auc.append(roc_auc_score(y_tr,probtr))
optimal_c = C[cv_auc.index(max(cv_auc))]
C=[np.log10(x) for x in C]
plt.plot(C,tr_auc,label="tr auc")
plt.plot(C,cv_auc,label="cv auc")
plt.title("AUC vs Hyperparameter")
plt.xlabel("Hyperparameter")
plt.ylabel("AUC")
print("optimal lambda : ",optimal_c)
```

```
100%|██████████████████████████████████████████████████| 9/9 [00:05<0
0:00,  1.82it/s]
```

```
optimal lambda :  0.001
```



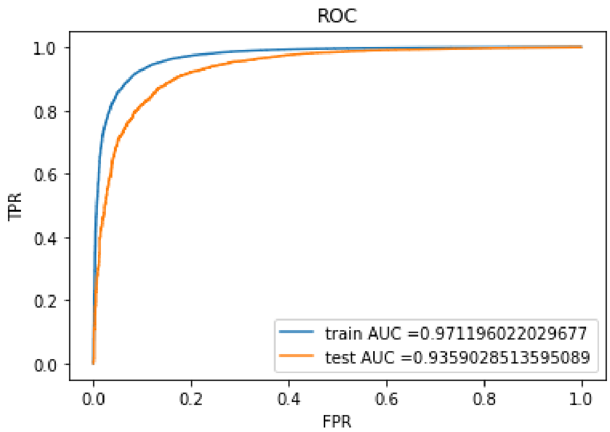
In [32]:

```
m = SGDClassifier(alpha=optimal_c)
svm = CalibratedClassifierCV(m, cv=3)
svm.fit(x_tr_bow,y_tr)
pred=svm.predict(x_te_bow)
# evaluate accuracy
acc = accuracy_score(y_te, pred) * 100
print('\nThe accuracy of the SVM Classifier C = %f is %f%%' % (optimal_c, acc))
```

```
The accuracy of the SVM Classifier C = 0.001000 is 91.293333%
```

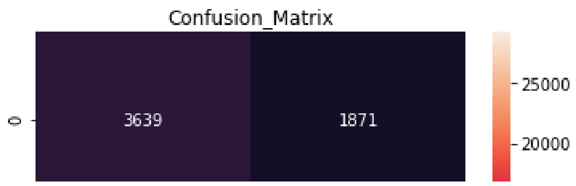
In [33]:

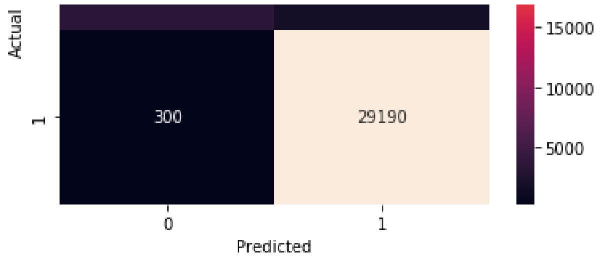
```
tr_fpr,tr_tpr,threshold1 = roc_curve(y_tr,svm.predict_proba(x_tr_bow)[:,1])
te_fpr,te_tpr,threshold2 = roc_curve(y_te,svm.predict_proba(x_te_bow)[:,1])
AUC = str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr,tr_tpr,label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr,te_tpr,label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



In [34]:

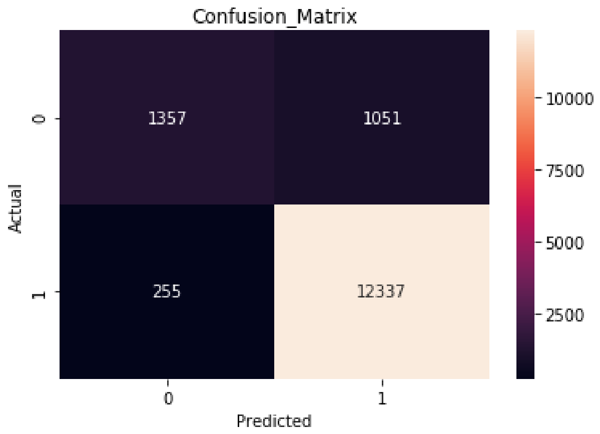
```
#Confusion Matrix for train
import seaborn as sb
con_matr = confusion_matrix(y_tr, svm.predict(x_tr_bow))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")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





In [35]:

```
#Confusion Matrix for test
import seaborn as sb
con_matr = confusion_matrix(y_te, svm.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 [36]:

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

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

           0       0.84      0.56      0.68      2408
           1       0.92      0.98      0.95     12592

   micro avg       0.91      0.91      0.91     15000
   macro avg       0.88      0.77      0.81     15000
weighted avg       0.91      0.91      0.91     15000

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

In [37]:

```
#Top 10 Positive features
a_f = vect.get_feature_names()
m = SGDClassifier(alpha=optimal_c)
m.fit(x_tr_bow,y_tr)
we = m.coef_
p_index = np.argsort(we)[:,::-1]
print("Top 10 Positive features :")
for s in list(p_index[0][0:10]):
    print(a_f[s])
```

```
Top 10 Positive features :
perfect
delicious
great
loves
amazing
smooth
```

```
best
excellent
thank
complaint
```

In [38]:

```python
#Top 10 Negative features
n_index = np.argsort(we)[:,::1]
print("Top 10 negative features :")
for s in list(n_index[0][0:10]):
    print(a_f[s])
```

```
Top 10 negative features :
disappointing
worst
awful
terrible
unfortunately
disappointment
disappointed
return
threw
horrible
```

[5.1.2] Applying Linear SVM on TFIDF, SET 2

In [39]:

```python
tfidfvect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tfidfvect.fit(x_tr)
x_tr_tfidf = tfidfvect.transform(x_tr)
x_te_tfidf = tfidfvect.transform(x_te)
x_cv_tfidf = tfidfvect.transform(x_cv)
print(x_tr_tfidf.shape,y_tr.shape)
```

```
(35000, 20328) (35000,)
```

In [40]:

```python
std = StandardScaler(with_mean=False)
std.fit(x_tr_tfidf)
x_tr_tfidf = std.transform(x_tr_tfidf)
x_te_tfidf = std.transform(x_te_tfidf)
x_cv_tfidf = std.transform(x_cv_tfidf)
print(x_tr_tfidf.shape,y_tr.shape)
```

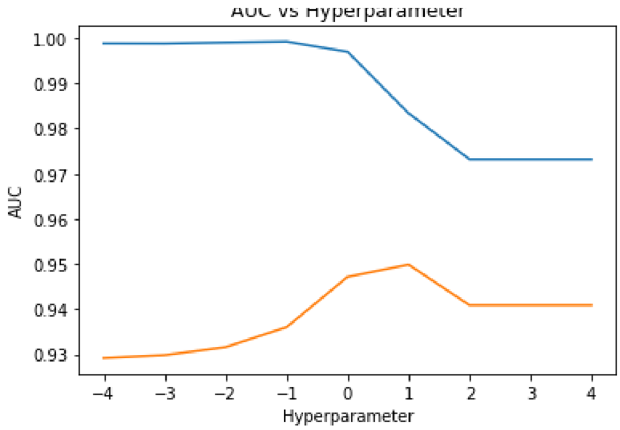
```
(35000, 20328) (35000,)
```

In [41]:

```python
C = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]
tr_auc = []
cv_auc = []
for c in tqdm(C):
    m = SGDClassifier(alpha=c,class_weight="balanced") #Default loss is hinge
    svm = CalibratedClassifierCV(m, cv=3)
    svm.fit(x_tr_tfidf,y_tr)
    probcv = svm.predict_proba(x_cv_tfidf)[:,1]
    cv_auc.append(roc_auc_score(y_cv,probcv))
    probtr = svm.predict_proba(x_tr_tfidf)[:,1]
    tr_auc.append(roc_auc_score(y_tr,probtr))
optimal_c1 = C[cv_auc.index(max(cv_auc))]
C=[np.log10(x) for x in C]
plt.plot(C,tr_auc,label="tr auc")
plt.plot(C,cv_auc,label="cv auc")
plt.title("AUC vs Hyperparameter")
plt.xlabel("Hyperparameter")
plt.ylabel("AUC")
print("optimal lambda : ",optimal_c1)
```

```
100%|██████████████████████████████████████████| 9/9 [00:04<00:00,  2.16it/s]
```

```
optimal lambda :  10
```



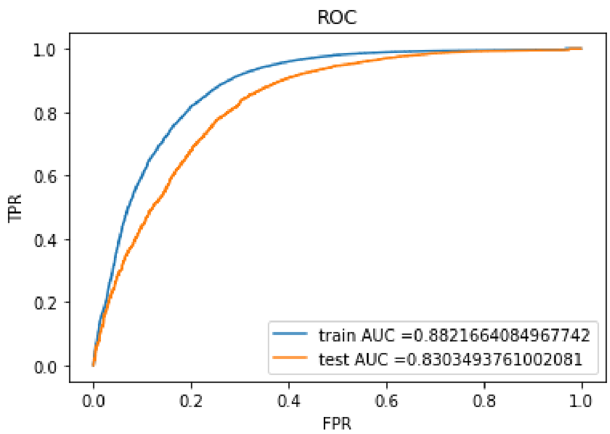
In [42]:

```
m = SGDClassifier(alpha=optimal_c1)
svm = CalibratedClassifierCV(m,cv=3)
svm.fit(x_tr_tfidf,y_tr)
pred1 = svm.predict(x_te_tfidf)
# evaluate accuracy
acc = accuracy_score(y_te, pred1) * 100
print('\nThe accuracy of the Logistic Regression C = %d is %f%%' % (optimal_c1, acc))
```

```
The accuracy of the Logistic Regression C = 10 is 87.593333%
```

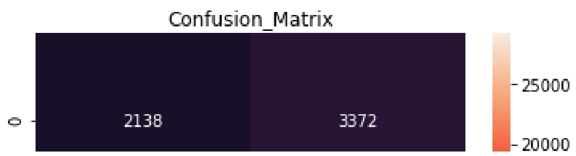
In [43]:

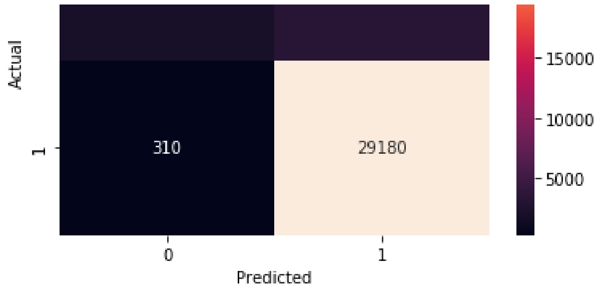
```
tr_fpr,tr_tpr,threshold1 = roc_curve(y_tr,svm.predict_proba(x_tr_tfidf)[:,1])
te_fpr,te_tpr,threshold2 = roc_curve(y_te,svm.predict_proba(x_te_tfidf)[:,1])
AUC1 = str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr,tr_tpr,label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr,te_tpr,label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



In [44]:

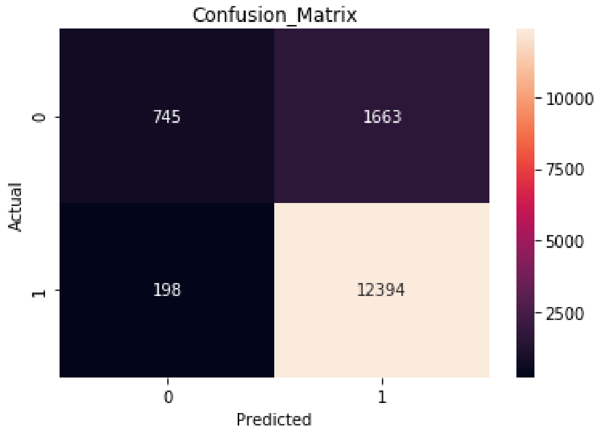
```
#Confusion Matrix for train
import seaborn as sb
con_matr = confusion_matrix(y_tr, svm.predict(x_tr_tfidf))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")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





In [45]:

```
#Confusion Matrix for test
import seaborn as sb
con_matr = confusion_matrix(y_te, svm.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 [46]:

```
print('='*50)
print(classification_report(y_te, pred1))
print('='*50)
```

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

           0       0.79      0.31      0.44      2408
           1       0.88      0.98      0.93     12592

   micro avg       0.88      0.88      0.88     15000
   macro avg       0.84      0.65      0.69     15000
weighted avg       0.87      0.88      0.85     15000

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

In [47]:

```
#Top 10 Positive features
a_f = tfidfvect.get_feature_names()
m = SGDClassifier(alpha=optimal_c1)
m.fit(x_tr_tfidf,y_tr)
we = m.coef_
p_index = np.argsort(we)[:,::-1]
print("Top 10 Positive features :")
for s in list(p_index[0][0:10]):
    print(a_f[s])
```

```
Top 10 Positive features :
great
love
good
best
delicious
```

```
delicious
loves
like
price
flavor
favorite
```

In [48]:

```
n_index = np.argsort(we)[:,::1]
print("Top 10 Negative features :")
for s in list(n_index[0][0:10]):
    print(a_f[s])
```

```
Top 10 Negative features :
return
worst
not worth
awful
waste money
terrible
not buy
horrible
threw
not recommend
```

[5.1.3] Applying Linear SVM 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: UserWarning: C extension not loaded, t
raining will be slow. Install a C compiler and reinstall gensim for fast training.
  "C extension not loaded, training will be slow. "
```

```
number of words that occured minimum 5 times  11613
sample words  ['dogs', 'love', 'chewies', 'little', 'smaller', 'others', 'type', 'overall', 'good', 'va
lue', 'well', 'liked', 'croutons', 'even', 'though', 'wife', 'ordering', 'mail', 'combination', 'differ
ent', 'breads', 'rye', 'pumpernickel', 'flavoring', 'not', 'overpowering', 'pretty', 'large', 'soak', '
lot', 'dressing', 'mmmm', 'grossly', 'overpriced', 'product', 'price', 'organic', 'weak', 'flavor', 'ap
pears', 'highly', 'diluted', 'never', 'buy', 'products', 'jerky', 'tastiest', 'stuff', 'ever', 'could']
```

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, you might need to change this to 3
00 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_tr.append(sent_vec)
print(len(sent_vectors_tr))
#print(sent_vectors_tr[0])

##CV
i=0
list_of_sentance_cv = []
```

```python
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 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 3
00 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_cv.append(sent_vec)
print(len(sent_vectors_cv))
#print(sent_vectors_cv[0])

##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, you might need to change this to 3
00 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_te.append(sent_vec)
print(len(sent_vectors_te))
#print(sent_vectors_te[0])
```

```
100%|██████████████████████████████████████████████████| 35000/35000 [01:03<00
:00, 552.02it/s]
```

```
35000
```

```
100%|██████████████████████████████████████████████████| 15000/15000 [00:27<00
:00, 546.09it/s]
```

```
15000
```

```
100%|██████████████████████████████████████████████████| 15000/15000 [00:27<00
:00, 547.35it/s]
```

```
15000
```

In [51]:

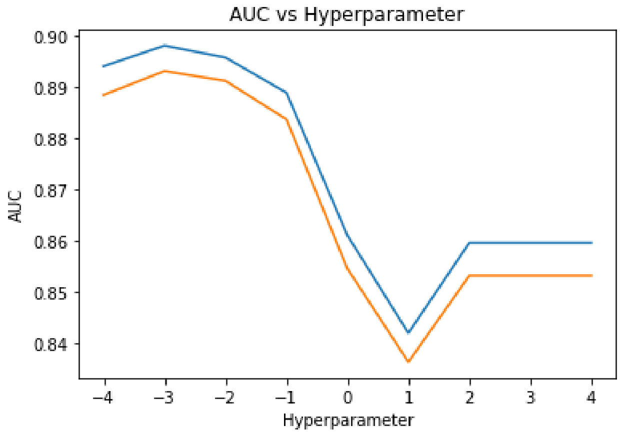
```python
x_tr_w2v = sent_vectors_tr
x_cv_w2v = sent_vectors_cv
x_te_w2v = sent_vectors_te

C = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]
tr_auc = []
cv_auc = []
for c in tqdm(C):
    m = SGDClassifier(alpha=c,class_weight="balanced") #Default loss is hinge
    svm = CalibratedClassifierCV(m, cv=4)
    svm.fit(x_tr_w2v,y_tr)
    probcv = svm.predict_proba(x_cv_w2v)[:,1]
    cv_auc.append(roc_auc_score(y_cv,probcv))
    probtr = svm.predict_proba(x_tr_w2v)[:,1]
    tr_auc.append(roc_auc_score(y_tr,probtr))
optimal_c2 = C[cv_auc.index(max(cv_auc))]
C=[np.log10(x) for x in C]
plt.plot(C,tr_auc,label="tr auc")
plt.plot(C,cv_auc,label="cv auc")
plt.title("AUC vs Hyperparameter")
plt.xlabel("Hyperparameter")
plt.ylabel("AUC")
```

```
plt.ylabel("AUC")
print("optimal lambda : ",optimal_c2)
```

```
100%|██████████████████████████████████████████████████| 9/9 [00:04<0
0:00,  2.17it/s]
```

```
optimal lambda :  0.001
```



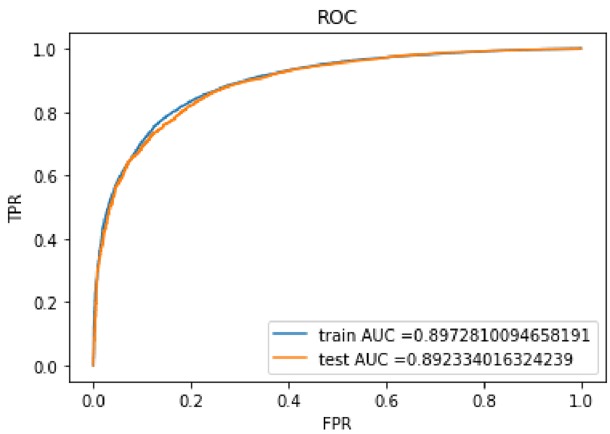
In [52]:

```
m = SGDClassifier(alpha=optimal_c2)
svm = CalibratedClassifierCV(m, cv=3)
svm.fit(x_tr_w2v,y_tr)
pred2=svm.predict(x_te_w2v)
# evaluate accuracy
acc = accuracy_score(y_te, pred2) * 100
print('\nThe accuracy of the Logistic Regression C = %f is %f%%' % (optimal_c2, acc))
```

```
The accuracy of the Logistic Regression C = 0.001000 is 87.973333%
```

In [53]:

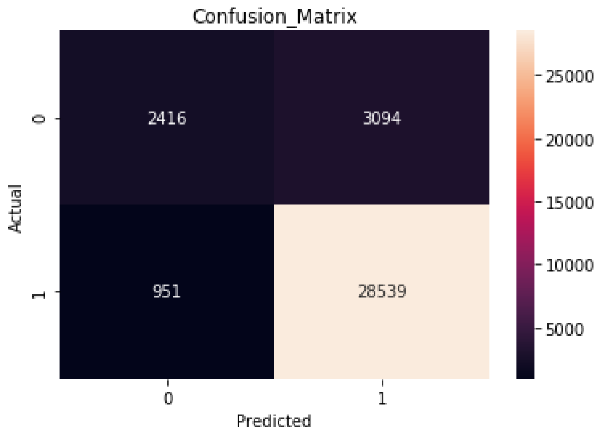
```
tr_fpr,tr_tpr,threshold1 = roc_curve(y_tr,svm.predict_proba(x_tr_w2v)[:,1])
te_fpr,te_tpr,threshold2 = roc_curve(y_te,svm.predict_proba(x_te_w2v)[:,1])
AUC2 = str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr,tr_tpr,label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr,te_tpr,label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



In [54]:

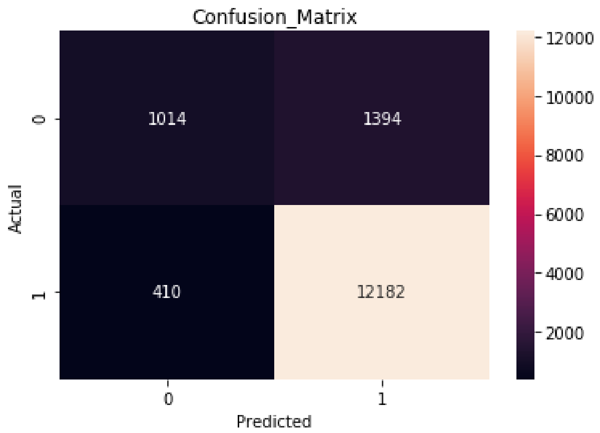
```
#Confusion Matrix for train
import seaborn as sb
con_matr = confusion_matrix(y_tr, svm.predict(x_tr_w2v))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
plt.show()
```



In [55]:

```
#Confusion Matrix for test
import seaborn as sb
con_matr = confusion_matrix(y_te, svm.predict(x_te_w2v))
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 [56]:

```
print('='*50)
print(classification_report(y_te, pred2))
print('='*50)
```

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

           0       0.71      0.42      0.53      2408
           1       0.90      0.97      0.93     12592

   micro avg       0.88      0.88      0.88     15000
   macro avg       0.80      0.69      0.73     15000
weighted avg       0.87      0.88      0.87     15000

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

[5.1.4] Applying Linear SVM on TFIDF W2V, SET 4

In [57]:

```
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 value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [58]:

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

# TF-IDF weighted Word2Vec
tfidf_feat = tfidfvect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_tr.append(sent_vec)
    row += 1

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

# TF-IDF weighted Word2Vec
tfidf_feat = tfidfvect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_cv.append(sent_vec)
    row += 1

##test
i=0
list_of_sentance_te=[]
for sentance in x_te:
    list_of_sentance_te.append(sentance.split())

# TF-IDF weighted Word2Vec
tfidf_feat = tfidfvect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_te.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████| 35000/35000 [07:56<00:00, 73.45it/s]
100%|████████████████████████████████| 15000/15000 [03:24<00:00, 62.02it/s]
100%|████████████████████████████████| 15000/15000 [03:18<00:00, 75.38it/s]
```

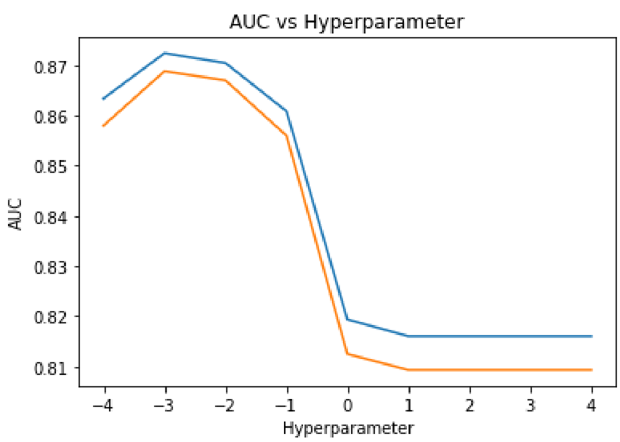
In [59]:

```
x_tr_tfw2v = tfidf_sent_vectors_tr
x_cv_tfw2v = tfidf_sent_vectors_cv
x_te_tfw2v = tfidf_sent_vectors_te

C = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]
tr_auc = []
cv_auc = []
for c in tqdm(C):
    m = SGDClassifier(alpha=c,class_weight="balanced") #Default loss is hinge
    svm = CalibratedClassifierCV(m, cv=3)
    svm.fit(x_tr_tfw2v,y_tr)
    probcv = svm.predict_proba(x_cv_tfw2v)[:,1]
    cv_auc.append(roc_auc_score(y_cv,probcv))
    probtr = svm.predict_proba(x_tr_tfw2v)[:,1]
    tr_auc.append(roc_auc_score(y_tr,probtr))
optimal_c3 = C[cv_auc.index(max(cv_auc))]
C=[np.log10(x) for x in C]
plt.plot(C,tr_auc,label="tr auc")
plt.plot(C,cv_auc,label="cv auc")
plt.title("AUC vs Hyperparameter")
plt.xlabel("Hyperparameter")
plt.ylabel("AUC")
print("optimal lambda : ",optimal_c3)
```

```
100%|████████████████████████████████| 9/9 [00:04<00:00,  2.30it/s]
```

```
optimal lambda :  0.001
```



In [60]:

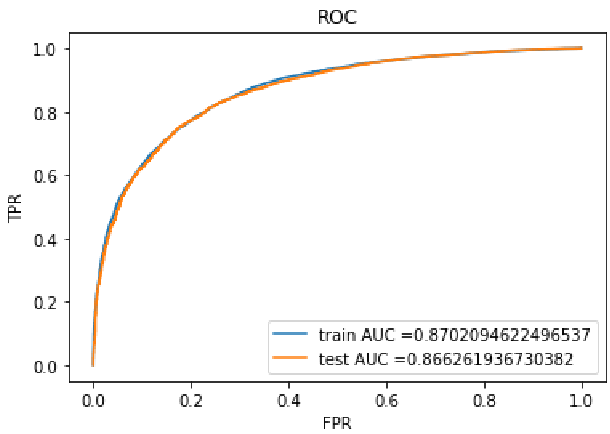
```
m = SGDClassifier(alpha=optimal_c3)
svm = CalibratedClassifierCV(m, cv=3)
svm.fit(x_tr_tfw2v,y_tr)
pred3=svm.predict(x_te_tfw2v)
```

```
# evaluate accuracy
acc = accuracy_score(y_te, pred3) * 100
print('\nThe accuracy of the Logistic Regression C = %f is %f%%' % (optimal_c3, acc))
```

The accuracy of the Logistic Regression C = 0.001000 is 87.000000%

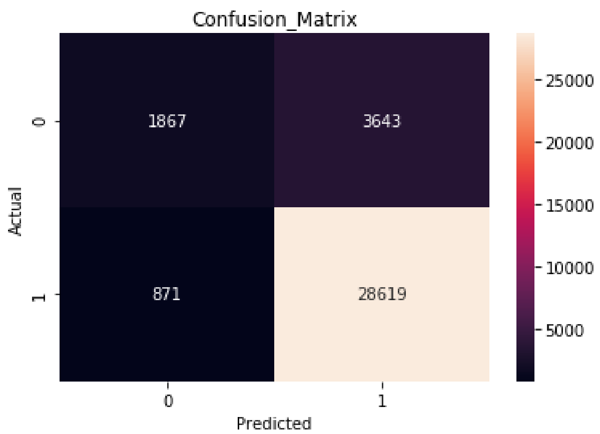
In [61]:

```
tr_fpr,tr_tpr,threshold1 = roc_curve(y_tr,svm.predict_proba(x_tr_tfw2v)[:,1])
te_fpr,te_tpr,threshold2 = roc_curve(y_te,svm.predict_proba(x_te_tfw2v)[:,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 [62]:

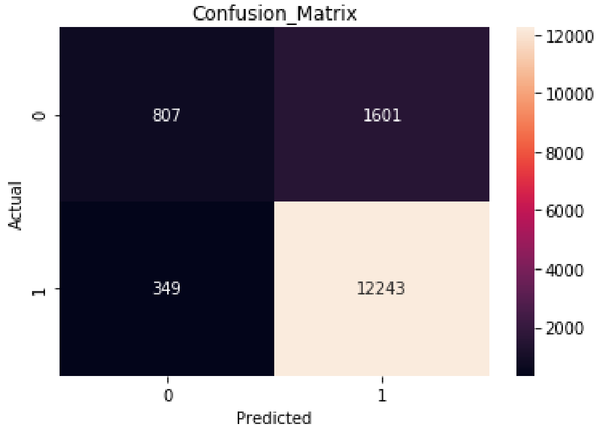
```
#Confusion Matrix for train
import seaborn as sb
con_matr = confusion_matrix(y_tr, svm.predict(x_tr_tfw2v))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [63]:

```
#Confusion Matrix for test
import seaborn as sb
con_matr = confusion_matrix(y_te, svm.predict(x_te_tfw2v))
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 [64]:

```
print('='*50)
print(classification_report(y_te, pred3))
print('='*50)
```

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

           0       0.70      0.34      0.45      2408
           1       0.88      0.97      0.93     12592

   micro avg       0.87      0.87      0.87     15000
   macro avg       0.79      0.65      0.69     15000
weighted avg       0.85      0.87      0.85     15000

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

[5.2] RBF SVM

[5.2.1] Applying RBF SVM on BOW, SET 1

In [65]:

```
vect1 = CountVectorizer(min_df=10, max_features=500)
vect1.fit(x_tr)
x_tr_bow1 = vect1.transform(x_tr)
x_te_bow1 = vect1.transform(x_te)
x_cv_bow1 = vect1.transform(x_cv)
print('='*50)
print(x_tr_bow1.shape,y_tr.shape)
print(x_cv_bow1.shape,y_cv.shape)
print(x_te_bow1.shape,y_te.shape)
print('='*50)
```

```
==================================================
(35000, 500) (35000,)
(15000, 500) (15000,)
(15000, 500) (15000,)
==================================================
```

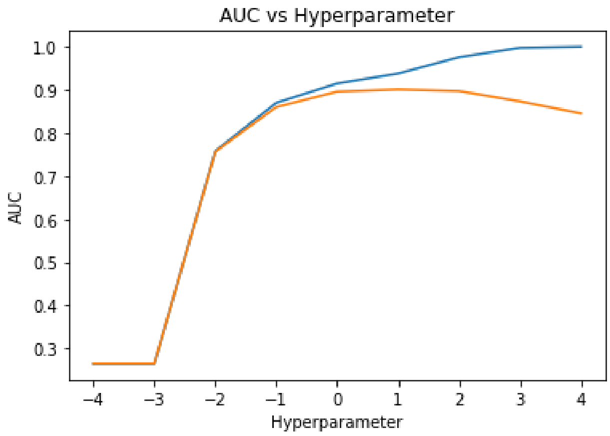
In [66]:

```
C = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]
tr_auc = []
cv_auc = []
for c in tqdm(C):
    svm = SVC(C=c,probability=True,class_weight="balanced")
    svm.fit(x_tr_bow1,y_tr)
    probcv = svm.predict_proba(x_cv_bow1)[:,1]
    cv_auc.append(roc_auc_score(y_cv,probcv))
    probtr = svm.predict_proba(x_tr_bow1)[:,1]
    tr_auc.append(roc_auc_score(y_tr,probtr))
optimal_c4 = C[cv_auc.index(max(cv_auc))]
C=[np.log10(x) for x in C]
plt.plot(C,tr_auc,label="tr_auc")
```

```
plt.plot(C,tr_auc,label="tr auc")
plt.plot(C,cv_auc,label="cv auc")
plt.title("AUC vs Hyperparameter")
plt.xlabel("Hyperparameter")
plt.ylabel("AUC")
print("optimal lambda : ",optimal_c4)
```

```
100%|████████████████████████████████████████| 9/9 [8:21:39<00:00, 5436.04s/it]
```

```
optimal lambda :  10
```



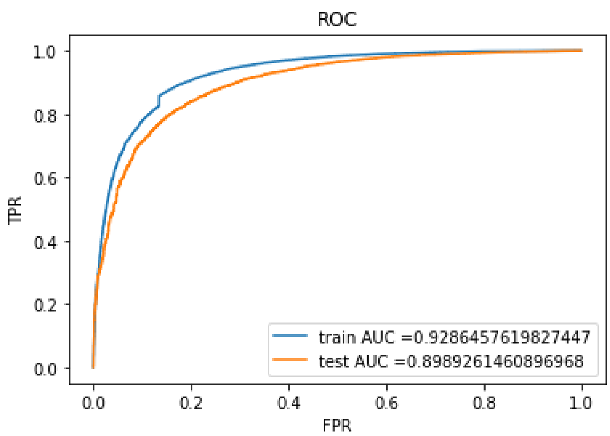
In [67]:

```
svm = SVC(C=optimal_c4,probability=True)
svm.fit(x_tr_bow1,y_tr)
pred4=svm.predict(x_te_bow1)
# evaluate accuracy
acc = accuracy_score(y_te, pred4) * 100
print('\nThe accuracy of the RBF SVM for C = %f is %f%%' % (optimal_c4, acc))
```

```
The accuracy of the RBF SVM for C = 10.000000 is 88.800000%
```

In [68]:

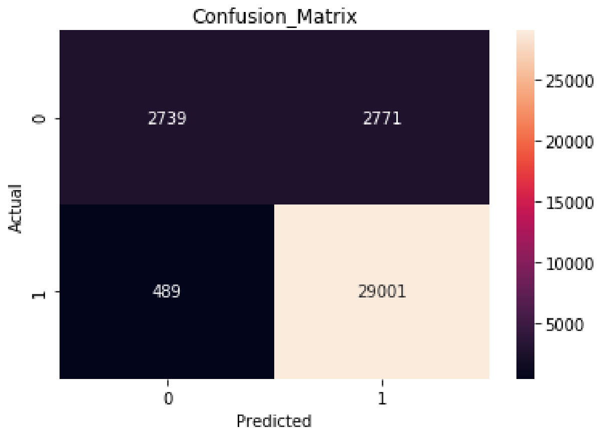
```
tr_fpr,tr_tpr,threshold1 = roc_curve(y_tr,svm.predict_proba(x_tr_bow1)[:,1])
te_fpr,te_tpr,threshold2 = roc_curve(y_te,svm.predict_proba(x_te_bow1)[:,1])
AUC4 = str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr,tr_tpr,label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr,te_tpr,label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



In [69]:

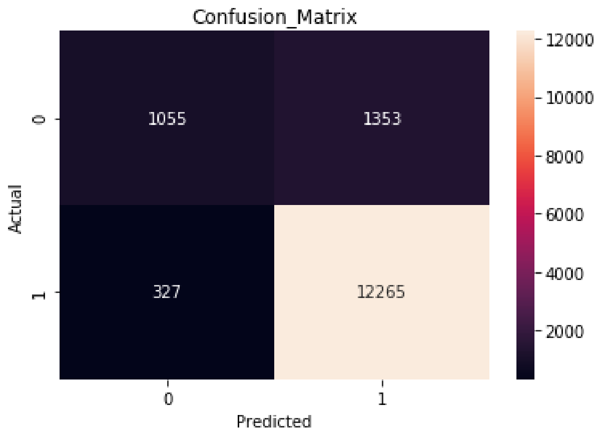
```
#Confusion Matrix for train
import seaborn as sb
con_matr = confusion_matrix(y_tr, svm.predict(x_tr_bow1))
c_l = [0, 1] #Class Label
df_con_matr = pd.DataFrame(con_matr, index=c_l, columns=c_l)
sb.heatmap(df_con_matr, annot=True, fmt='d')
```

```
plt.title("Confusion_Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [70]:

```
#Confusion Matrix for test
import seaborn as sb
con_matr = confusion_matrix(y_te, svm.predict(x_te_bow1))
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 [71]:

```
print('='*50)
print(classification_report(y_te, pred4))
print('='*50)
```

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

           0       0.76      0.44      0.56      2408
           1       0.90      0.97      0.94     12592

   micro avg       0.89      0.89      0.89     15000
   macro avg       0.83      0.71      0.75     15000
weighted avg       0.88      0.89      0.88     15000

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

[5.2.2] Applying RBF SVM on TFIDF, SET 2

In [72]:

```
tfidfvect1 = TfidfVectorizer(ngram_range=(1,2),min_df=10,max_features=500)
tfidfvect1.fit(x_tr)
```

```
x_tr_tfidf1 = tfidfvect1.transform(x_tr)
x_te_tfidf1 = tfidfvect1.transform(x_te)
x_cv_tfidf1 = tfidfvect1.transform(x_cv)
print(x_tr_tfidf1.shape,x_te_tfidf1.shape)
```

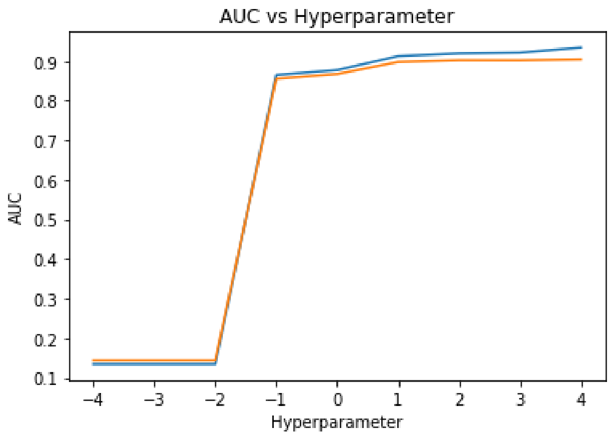
```
(35000, 500) (15000, 500)
```

In [73]:

```
C = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]
tr_auc = []
cv_auc = []
for c in tqdm(C):
    svm = SVC(C=c,probability=True,class_weight="balanced")
    svm.fit(x_tr_tfidf1,y_tr)
    probcv = svm.predict_proba(x_cv_tfidf1)[:,1]
    cv_auc.append(roc_auc_score(y_cv,probcv))
    probtr = svm.predict_proba(x_tr_tfidf1)[:,1]
    tr_auc.append(roc_auc_score(y_tr,probtr))
optimal_c5 = C[cv_auc.index(max(cv_auc))]
C=[np.log10(x) for x in C]
plt.plot(C,tr_auc,label="tr auc")
plt.plot(C,cv_auc,label="cv auc")
plt.title("AUC vs Hyperparameter")
plt.xlabel("Hyperparameter")
plt.ylabel("AUC")
print("optimal lambda : ",optimal_c5)
```

```
100%|██████████████████████████████| 9/9 [6:38:52<00:00, 3314.61s/it]
```

```
optimal lambda :  10000
```



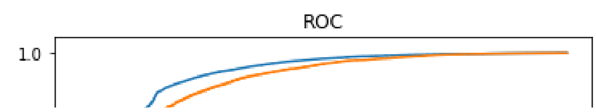
In [74]:

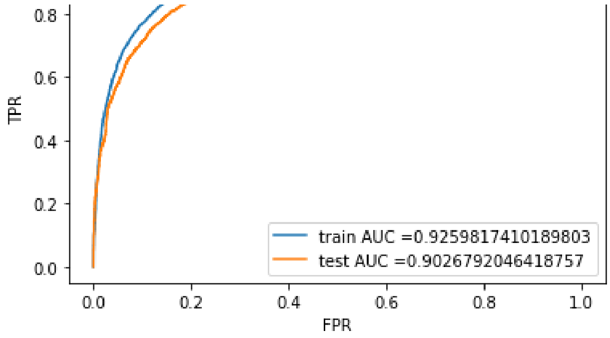
```
svm = SVC(C=optimal_c5,probability=True)
svm.fit(x_tr_tfidf1,y_tr)
pred5 = svm.predict(x_te_tfidf1)
# evaluate accuracy
acc = accuracy_score(y_te, pred5) * 100
print('\nThe accuracy of the RBF SVM for C = %f is %f%%' % (optimal_c5, acc))
```

```
The accuracy of the RBF SVM for C = 10000.000000 is 89.006667%
```

In [75]:

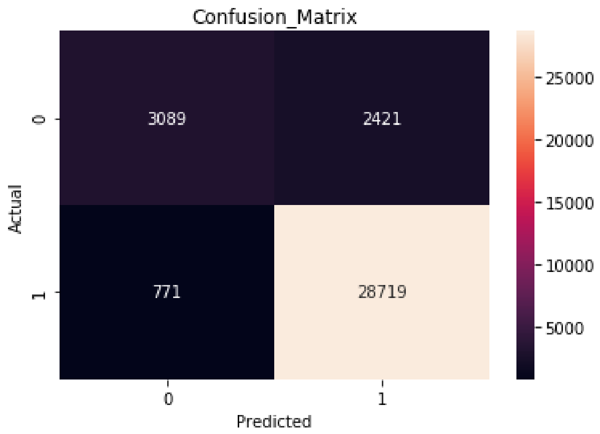
```
tr_fpr,tr_tpr,threshold1 = roc_curve(y_tr,svm.predict_proba(x_tr_tfidf1)[:,1])
te_fpr,te_tpr,threshold2 = roc_curve(y_te,svm.predict_proba(x_te_tfidf1)[:,1])
AUC5 = str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr,tr_tpr,label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr,te_tpr,label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```





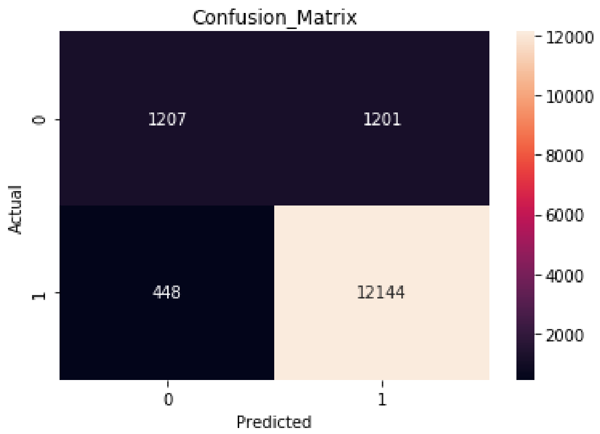
In [76]:

```
#Confusion Matrix for train
import seaborn as sb
con_matr = confusion_matrix(y_tr, svm.predict(x_tr_tfidf1))
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 [77]:

```
#Confusion Matrix for test
import seaborn as sb
con_matr = confusion_matrix(y_te, svm.predict(x_te_tfidf1))
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]:

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

```
print(classification_report(y_te, pred5))
print('='*50)
```

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

           0       0.73      0.50      0.59      2408
           1       0.91      0.96      0.94     12592

   micro avg       0.89      0.89      0.89     15000
   macro avg       0.82      0.73      0.77     15000
weighted avg       0.88      0.89      0.88     15000

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

[5.2.3] Applying RBF SVM on AVG W2V, SET 3

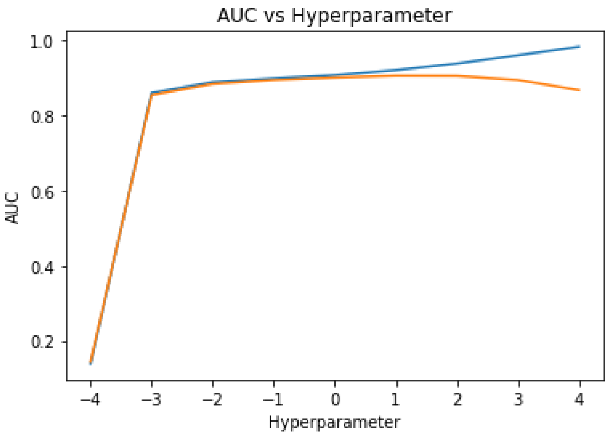
In [79]:

```
x_tr_w2v1 = sent_vectors_tr
x_cv_w2v1 = sent_vectors_cv
x_te_w2v1 = sent_vectors_te

C = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]
tr_auc = []
cv_auc = []
for c in tqdm(C):
    svm = SVC(C=c,probability=True,class_weight="balanced")
    svm.fit(x_tr_w2v1,y_tr)
    probcv = svm.predict_proba(x_cv_w2v1)[:,1]
    cv_auc.append(roc_auc_score(y_cv,probcv))
    probtr = svm.predict_proba(x_tr_w2v1)[:,1]
    tr_auc.append(roc_auc_score(y_tr,probtr))
optimal_c6 = C[cv_auc.index(max(cv_auc))]
C=[np.log10(x) for x in C]
plt.plot(C,tr_auc,label="tr auc")
plt.plot(C,cv_auc,label="cv auc")
plt.title("AUC vs Hyperparameter")
plt.xlabel("Hyperparameter")
plt.ylabel("AUC")
print("optimal lambda : ",optimal_c6)
```

```
100%|██████████████████████████████████████████| 9/9 [9:34:37<00:00, 8611.53s/it]
```

```
optimal lambda :  10
```



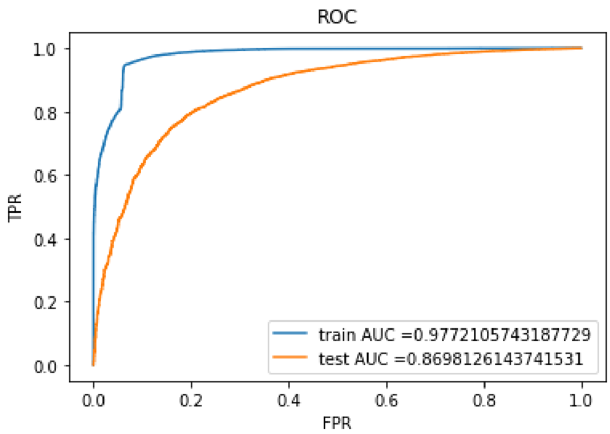
In [80]:

```
svm = SVC(C=c,probability=True)
svm.fit(x_tr_w2v1,y_tr)
pred6 = svm.predict(x_te_w2v1)
# evaluate accuracy
acc = accuracy_score(y_te, pred6) * 100
print('\nThe accuracy of the RBF SVM for C = %f is %f%%' % (optimal_c6, acc))
```

```
The accuracy of the RBF SVM for C = 10.000000 is 87.106667%
```

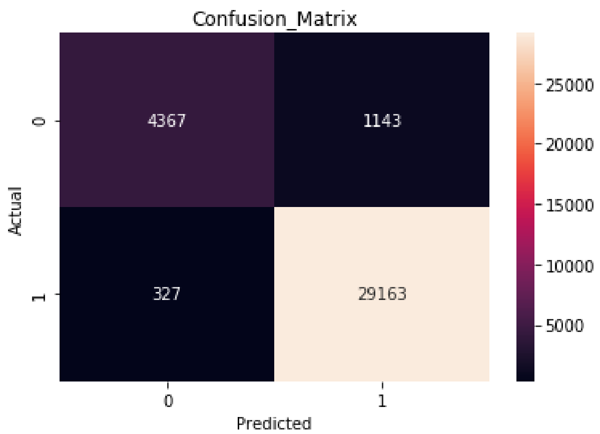
In [81]:

```
tr_fpr,tr_tpr,threshold1 = roc_curve(y_tr,svm.predict_proba(x_tr_w2v1)[:,1])
te_fpr,te_tpr,threshold2 = roc_curve(y_te,svm.predict_proba(x_te_w2v1)[:,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()
```



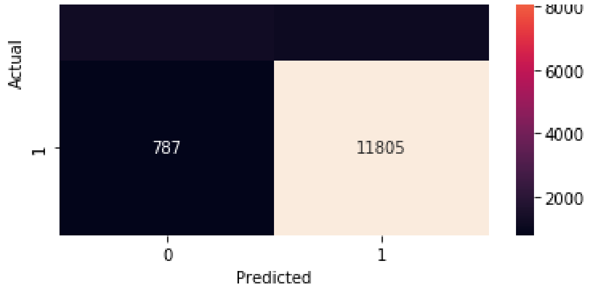
In [82]:

```
#Confusion Matrix for train
import seaborn as sb
con_matr = confusion_matrix(y_tr, svm.predict(x_tr_w2v1))
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 [83]:

```
#Confusion Matrix for test
import seaborn as sb
con_matr = confusion_matrix(y_te, svm.predict(x_te_w2v1))
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 [84]:

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

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

           0       0.62      0.52      0.57      2408
           1       0.91      0.94      0.92     12592

   micro avg       0.87      0.87      0.87     15000
   macro avg       0.76      0.73      0.75     15000
weighted avg       0.86      0.87      0.87     15000

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

[5.2.4] Applying RBF SVM on TFIDF W2V, SET 4

In [85]:

```
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 value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [86]:

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

# TF-IDF weighted Word2Vec
tfidf_feat = tfidfvect1.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_tr1 = []; # 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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_tr1.append(sent_vec)
    row += 1

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

```
    list_of_sentance_cv.append(sentance.split())

# TF-IDF weighted Word2Vec
tfidf_feat = tfidfvect1.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_cv1 = []; # 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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_cv1.append(sent_vec)
    row += 1

##test
i=0
list_of_sentance_te=[]
for sentance in x_te:
    list_of_sentance_te.append(sentance.split())

# TF-IDF weighted Word2Vec
tfidf_feat = tfidfvect1.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_te1 = []; # 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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_te1.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████| 35000/35000 [01:16<00:00, 459.81it/s]
100%|██████████████████████████████| 15000/15000 [00:32<00:00, 457.06it/s]
100%|██████████████████████████████| 15000/15000 [00:32<00:00, 456.63it/s]
```

In [87]:

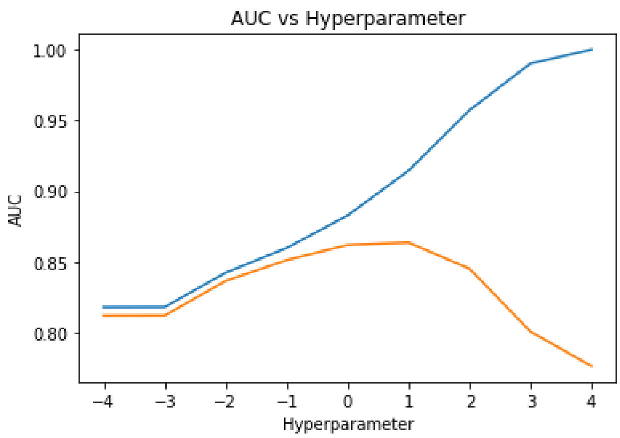
```
x_tr_tfw2v1 = tfidf_sent_vectors_tr1
x_cv_tfw2v1 = tfidf_sent_vectors_cv1
x_te_tfw2v1 = tfidf_sent_vectors_te1

C = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]
tr_auc = []
cv_auc = []
for c in tqdm(C):
    svm = SVC(C=c,probability=True,class_weight="balanced") #Default loss is hinge
    svm.fit(x_tr_tfw2v1,y_tr)
    probcv = svm.predict_proba(x_cv_tfw2v1)[:,1]
    cv_auc.append(roc_auc_score(y_cv,probcv))
```

```
    probtr = svm.predict_proba(x_tr_tfw2v1)[:,1]
    tr_auc.append(roc_auc_score(y_tr,probtr))
optimal_c7 = C[cv_auc.index(max(cv_auc))]
C=[np.log10(x) for x in C]
plt.plot(C,tr_auc,label="tr auc")
plt.plot(C,cv_auc,label="cv auc")
plt.title("AUC vs Hyperparameter")
plt.xlabel("Hyperparameter")
plt.ylabel("AUC")
print("optimal lambda : ",optimal_c7)
```

```
100%|| 9/9 [12:23:05<00:0
0, 10944.72s/it]
```

```
optimal lambda :  10
```



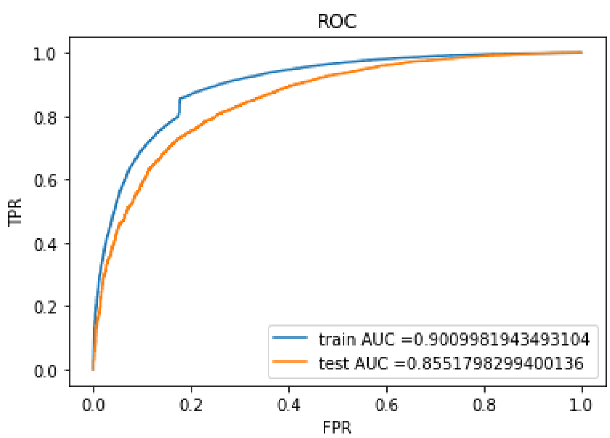
In [88]:

```
svm = SVC(C=optimal_c7,probability=True) #Default loss is hinge
svm.fit(x_tr_tfw2v1,y_tr)
pred7=svm.predict(x_te_tfw2v1)
# evaluate accuracy
acc = accuracy_score(y_te, pred7) * 100
print('\nThe accuracy of the RBF SVM for C = %f is %f%%' % (optimal_c7, acc))
```

```
The accuracy of the RBF SVM for C = 10.000000 is 86.826667%
```

In [89]:

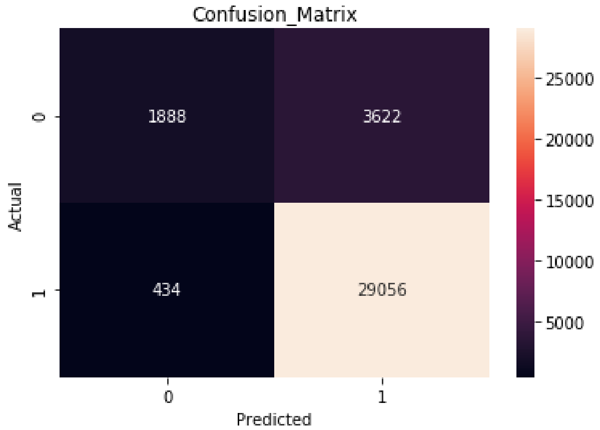
```
tr_fpr,tr_tpr,threshold1 = roc_curve(y_tr,svm.predict_proba(x_tr_tfw2v1)[:,1])
te_fpr,te_tpr,threshold2 = roc_curve(y_te,svm.predict_proba(x_te_tfw2v1)[:,1])
AUC7 = str(auc(te_fpr, te_tpr))
plt.plot(tr_fpr,tr_tpr,label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr,te_tpr,label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.title("ROC")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



In [90]:

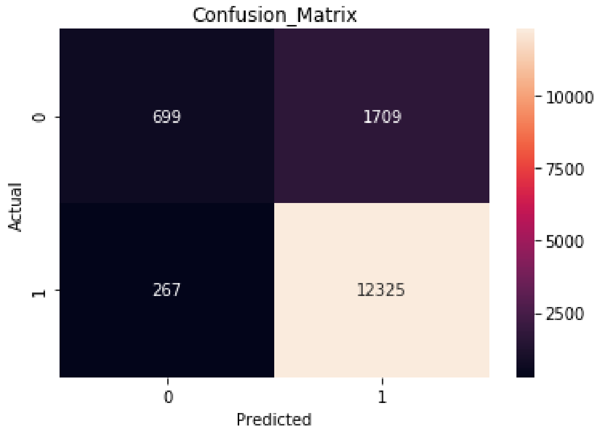
```
#Confusion Matrix for train
import seaborn as sb
```

```
con_matr = confusion_matrix(y_tr, svm.predict(x_tr_tfw2v1))
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 [91]:

```
#Confusion Matrix for test
import seaborn as sb
con_matr = confusion_matrix(y_te, svm.predict(x_te_tfw2v1))
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 [92]:

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

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

           0       0.62      0.52      0.57      2408
           1       0.91      0.94      0.92     12592

   micro avg       0.87      0.87      0.87     15000
   macro avg       0.76      0.73      0.75     15000
weighted avg       0.86      0.87      0.87     15000

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

[6] Conclusions

In [93]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
comparison = PrettyTable()
comparison.field_names = ["Vectorizer", "Linear/RBF", "Hyperparameter", "AUC"]
comparison.add_row(["BOW", 'Linear', optimal_c, np.round(float(AUC),3)])
comparison.add_row(["TFIDF", 'Linear', optimal_c1, np.round(float(AUC1),3)])
comparison.add_row(["AVG W2V", 'Linear', optimal_c2, np.round(float(AUC2),3)])
comparison.add_row(["Weighted W2V", 'Linear', optimal_c3,np.round(float(AUC3),3)])
comparison.add_row(["BOW", 'RBF', optimal_c4, np.round(float(AUC4),3)])
comparison.add_row(["TFIDF", 'RBF', optimal_c5, np.round(float(AUC5),3)])
comparison.add_row(["AVG W2V", 'RBF', optimal_c6, np.round(float(AUC6),3)])
comparison.add_row(["Weighted W2V", 'RBF', optimal_c7, np.round(float(AUC7),3)])
print(comparison)
```

```
+--------------+------------+----------------+-------+
|  Vectorizer  | Linear/RBF | Hyperparameter |  AUC  |
+--------------+------------+----------------+-------+
|     BOW      |   Linear   |     0.001      | 0.936 |
|    TFIDF     |   Linear   |       10       |  0.83 |
|   AVG W2V    |   Linear   |     0.001      | 0.892 |
| Weighted W2V |   Linear   |     0.001      | 0.866 |
|     BOW      |    RBF     |       10       | 0.899 |
|    TFIDF     |    RBF     |     10000      | 0.903 |
|   AVG W2V    |    RBF     |       10       |  0.87 |
| Weighted W2V |    RBF     |       10       | 0.855 |
+--------------+------------+----------------+-------+
```