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 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 100000""", con)

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

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

```
Number of data points in our data (100000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

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	COU
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2

	UserId	ProductId	ProfileName	Time	Score	Text	COU
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	COU

	UserId	ProductId	ProfileName	Time	Score	Text	COU
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	Helpfuln

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

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

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

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

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

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

In [8]:

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

In [9]:

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

Out[9]:

```
(87775, 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]:

```
87.775
```

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

In [11]:

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

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()
```

```
(87773, 10)
```

Out[13]:

```
1    73592
0    14181
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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the
candy has little taste to it.  Very little of the 2 lbs that I bought w
ere eaten and I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil sme
ll. So if you are afraid of the fishy smell, don't get it. But I think
my dog likes it because of the smell. These treats are really small in
size. They are great for training. You can give your dog several of the
se without worrying about him over eating. Amazon's price was much more
reasonable than any other retailer. You can buy a 1 pound bag on Amazon
for almost the same price as a 6 ounce bag at other retailers. It's def
initely worth it to buy a big bag if your dog eats them a lot.
==================================================
```

In [15]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
```

```
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [16]:

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

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the
candy has little taste to it.  Very little of the 2 lbs that I bought w
ere eaten and I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil sme
ll. So if you are afraid of the fishy smell, don't get it. But I think
my dog likes it because of the smell. These treats are really small in
size. They are great for training. You can give your dog several of the
se without worrying about him over eating. Amazon's price was much more
reasonable than any other retailer. You can buy a 1 pound bag on Amazon
for almost the same price as a 6 ounce bag at other retailers. It's def
initely worth it to buy a big bag if your dog eats them a lot.
```

In [17]:

```python
# 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]:

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

```
was way to hot for my blood, took a bite and did a jig  lol
```

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

In [19]:

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

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [20]:

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

```
was way to hot for my blood took a bite and did a jig lol
```

In [21]:

```
# 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 [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%|██████████| 87773/87773 [00:26<00:00, 3312.25it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

In [24]:

```
## Similartly you can do preprocessing for review summary also.## Similartly you can do preprocessing for review summary also.
# Combining all the above stundents
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].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_summary.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:16<00:00, 5479.65it/s]
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

```
#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', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaa
aaaaaaaaaaaa', 'aaaaaaahhhhhh', 'aaaaaaarrrrrggghhh', 'aaaaaawwwwwwwwww
w', 'aaaaah']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 54904)
the number of unique words  54904
```

[4.2] Bi-Grams and n-Grams.

In [26]:

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

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

# you can choose these 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])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [27]:

```
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) ['aa', 'aafco', 'abac
k', 'abandon', 'abandoned', 'abdominal', 'ability', 'able', 'able add',
'able brew']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (87773, 51709)
the number of unique words including both unigrams and bigrams  51709
```

[4.4] Word2Vec

In [28]:

```
# 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 [29]:

```
# 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 ")
```

```
[('fantastic', 0.8411622643470764), ('awesome', 0.8402246236801147),
('good', 0.822651207447052), ('excellent', 0.8104960918426514), ('perfe
ct', 0.7841705679893494), ('terrific', 0.7831875681877136), ('wonderfu
l', 0.7752300500869751), ('nice', 0.7223321199417114), ('amazing', 0.71
71570658683777), ('decent', 0.6966366767883301)]
==================================================
[('greatest', 0.7975115776062012), ('tastiest', 0.7416020631790161),
('best', 0.7161286473274231), ('nastiest', 0.6732054948806763), ('disgu
sting', 0.614565372467041), ('smoothest', 0.6093403100967407), ('vile',
0.5996800065040588), ('horrible', 0.588977575302124), ('terrible', 0.58
74980688095093), ('awful', 0.5867300033569336)]
```

In [30]:

```
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  17386
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont',
'buying', 'anymore', 'hard', 'find', 'products', 'made', 'usa', 'one',
'isnt', 'bad', 'good', 'take', 'chances', 'till', 'know', 'going', 'imp
orts', 'love', 'saw', 'pet', 'store', 'tag', 'attached', 'regarding',
'satisfied', 'safe', 'infestation', 'literally', 'everywhere', 'flyin
g', 'around', 'kitchen', 'bought', 'hoping', 'least', 'get', 'rid', 'we
eks', 'fly', 'stuck', 'squishing', 'buggers', 'success', 'rate']
```

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

[4.4.1.1] Avg W2v

In [31]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 87773/87773 [02:23<00:00, 612.41it/s]
```

```
87773
50
```

[4.4.1.2] TFIDF weighted W2v

In [32]:

```
# 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 [33]:

```
# 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%|██████████| 87773/87773 [28:15<00:00, 34.31it/s]
```

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression 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. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**
   - 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

3. **Pertubation Test**
   - Get the weights W after fit your model with the data X i.e Train data.

- Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
- Fit the model again on data X' and get the weights W'
- Add a small eps value(to eliminate the divisible by zero error) to W and W’ i.e W=W+10^-6 and W’ = W’+10^-6
- Now find the % change between W and W' (| (W-W') / (W) |)*100)
- Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector
- Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
- Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4. **Sparsity**

   - Calculate sparsity on weight vector obtained after using L1 regularization

   NOTE: Do sparsity and multicollinearity for any one of the vectorizers. Bow or tf-idf is recommended.

5. **Feature importance**

   - Get top 10 important features for both positive and negative classes separately.

6. **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.

7. **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.

8. **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 Logistic Regression

[5.1] Logistic Regression on BOW, SET 1

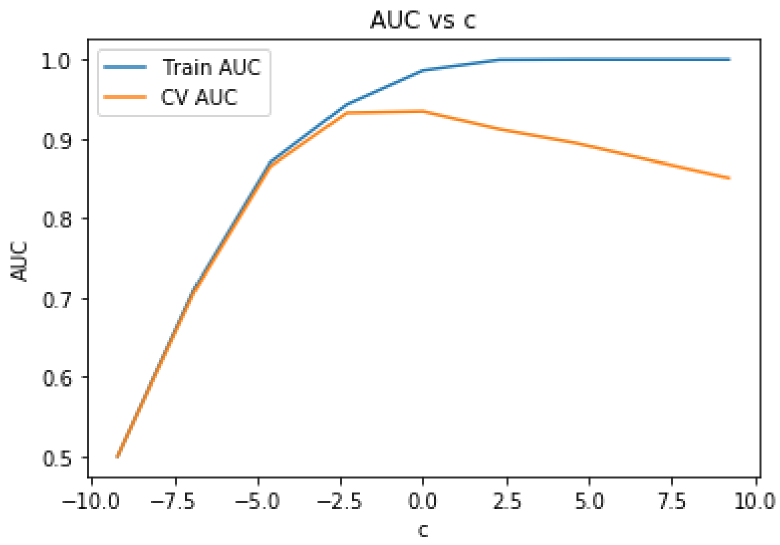
[5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

In [34]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
a=preprocessed_reviews
b=np.array(final['Score'])
bow_vectorizer=CountVectorizer()
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-an
d-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_select
ion.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3,random_state
=0)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)

#https://datascience.stackexchange.com/questions/12321/difference-betwe
en-fit-and-fit-transform-in-scikit-learn-models
#the above link is been used to clarify whether to use .fit() or .fit_t
ransform().I am using fit_transform() on train data and transform() on
 cv and test data
from sklearn.preprocessing import StandardScaler
xTrain2=bow_vectorizer.fit_transform(xTrain)
x_cv2=bow_vectorizer.transform(x_cv)
xTest2=bow_vectorizer.transform(xTest)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
```

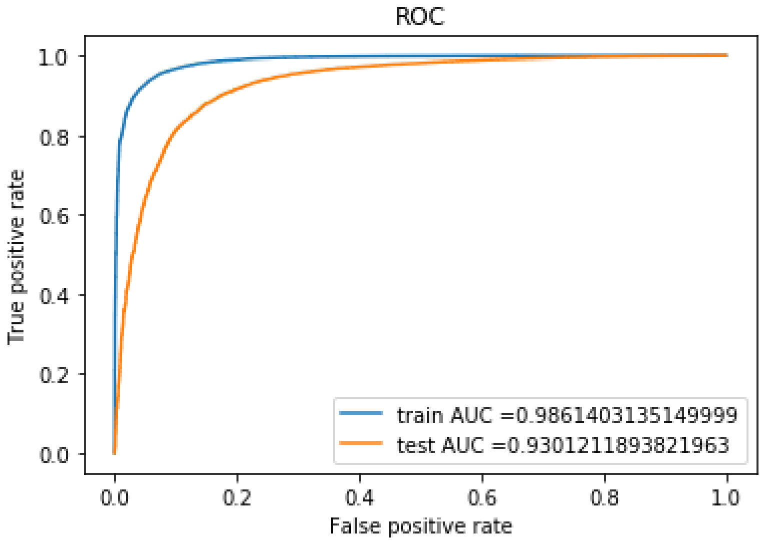
```
for i in c:
    mdl = LogisticRegression(penalty='l1',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

```
best lambda is 1
```



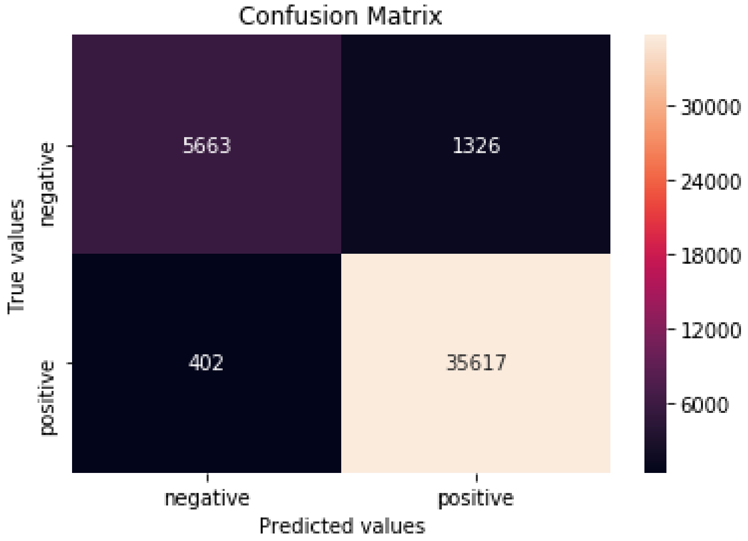
In [35]:

```
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()
```



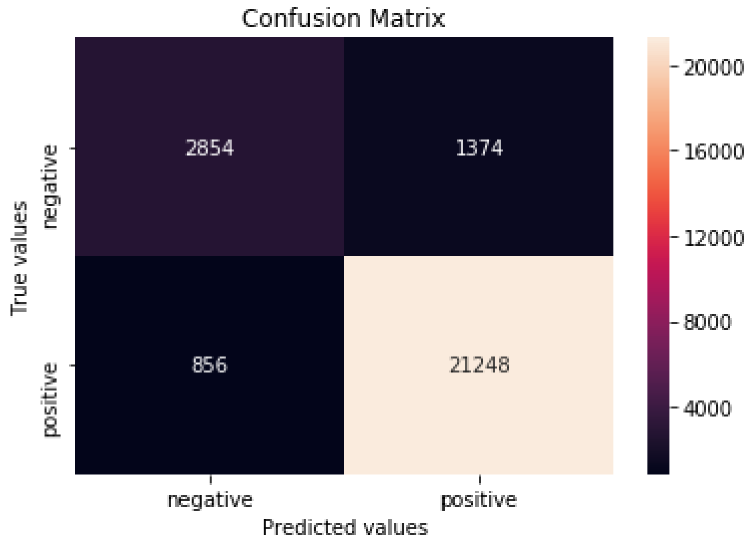
In [36]:

```
#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



In [37]:

```
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



[5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1

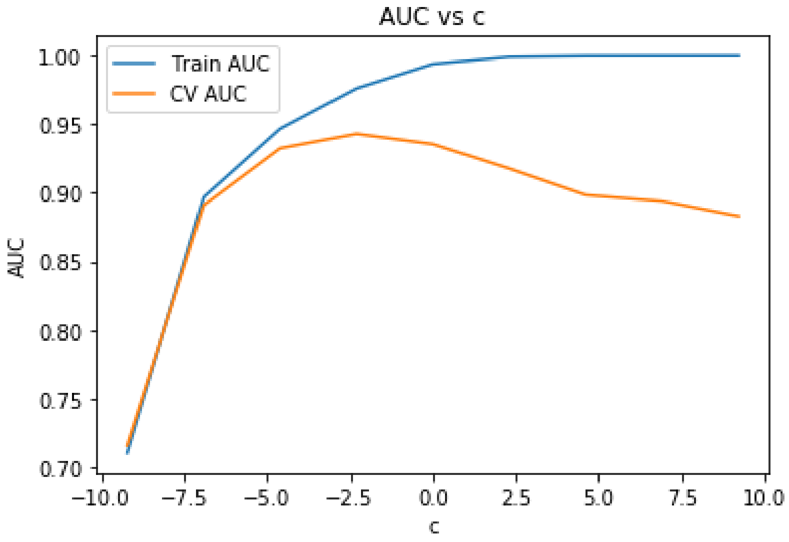
In [38]:
```
# Please write all the code with proper documentation
mdl = LogisticRegression(C=0.01, penalty='l1');
mdl.fit(xTrain2,yTrain);
w = mdl.coef_
print("Numbe of non-zero elemnts in weight vector is",np.count_nonzero(
w))
```

```
Numbe of non-zero elemnts in weight vector is 89
```

[5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

In [39]:
```
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
a=preprocessed_reviews
b=np.array(final['Score'])
tfidf_vectorizer=CountVectorizer()
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3,random_state=0)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)

#https://datascience.stackexchange.com/questions/12321/difference-between-fit-and-fit-transform-in-scikit-learn-models
#the above link is been used to clarify whether to use .fit() or .fit_transform().I am using fit_transform() on train data and transform() on cv and test data
from sklearn.preprocessing import StandardScaler
xTrain2=tfidf_vectorizer.fit_transform(xTrain)
x_cv2=tfidf_vectorizer.transform(x_cv)
xTest2=tfidf_vectorizer.transform(xTest)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l2',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

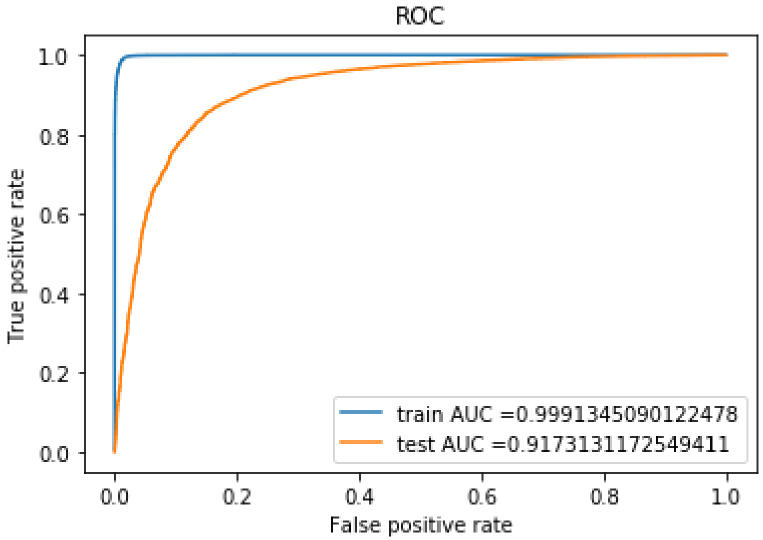
```
best lambda is 9.0
```



In [40]:

```
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
```

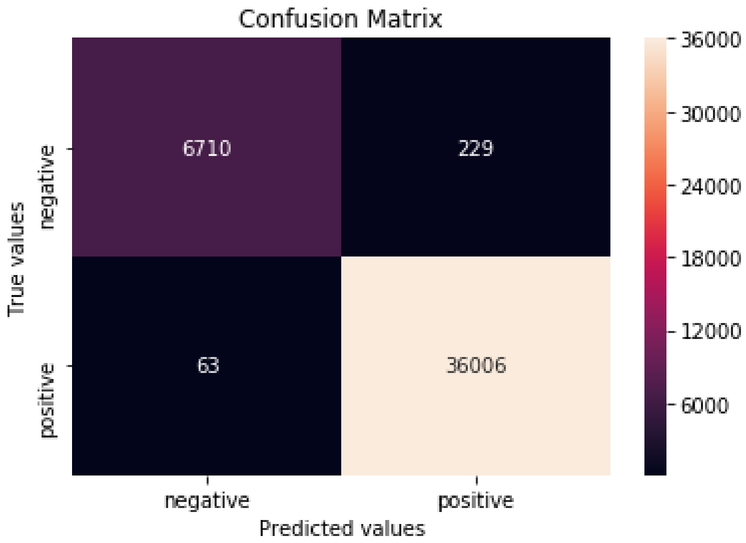
```
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()
```



In [41]:

```
#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
```

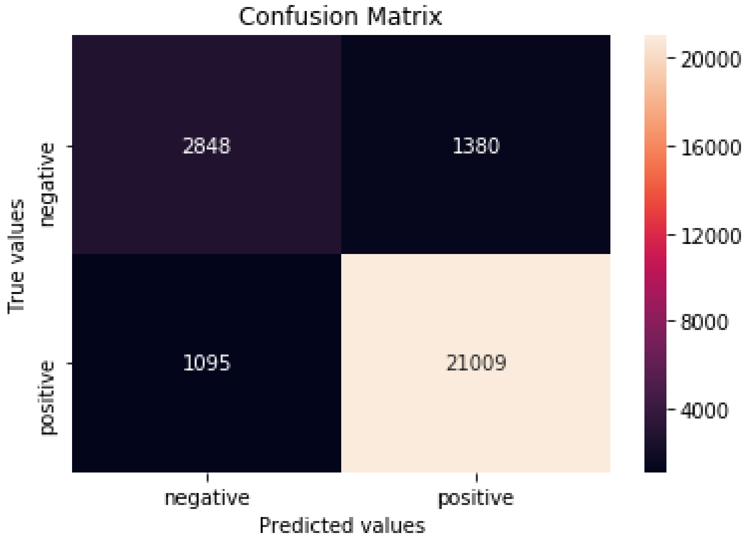
```
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



In [42]:

```
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
```

```
n-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



[5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1

In [43]:
```
# Please write all the code with proper documentation
weight1=mdl.coef_
xTrain3=xTrain2
xTrain3.data=np.random.uniform(low=-0.00001, high=0.00001,size=xTrain2.
data.shape)
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain3,yTrain)
weight2=mdl.coef_
```

```
weight1=weight1+10**-6
weight2=weight2+10**-6
for i in range(len(weight1)):
    p=np.abs((weight2[i]-weight1[i])/weight1[i])
q=p*100
```

In [44]:
```
for i in range(11):
    print(str(i*10)+'th percentile is    '+str(np.percentile(q,i*10)))
```

```
0th percentile is    0.009896793401745789
10th percentile is    92.13439264395211
20th percentile is    99.7837743232163
30th percentile is    99.99029840001475
40th percentile is    99.99868260998487
50th percentile is    99.99959550995153
60th percentile is    99.99984622632233
70th percentile is    100.00003475677873
80th percentile is    100.00016966015303
90th percentile is    100.00086171498015
100th percentile is    9254.687475431878
```

In [45]:
```
for i in range(90,101):
    print(str(i)+'th percentile is      '+str(np.percentile(q,i)))
```

```
90th percentile is      100.00086171498015
91th percentile is      100.00112627076935
92th percentile is      100.00163426298006
93th percentile is      100.00277957745031
94th percentile is      100.0065334473732
95th percentile is      100.02732289123915
96th percentile is      100.19524819538748
97th percentile is      102.07391635027628
98th percentile is      113.59052406426372
99th percentile is      169.02264172090042
100th percentile is      9254.687475431878
```

In [46]:
```
k=99
l=[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,1]
```

```
for i in l:
    print(str(k+i)+'th percentile is      '+str(np.percentile(q,k+i)))
```

```
99.0th percentile is      169.02264172090042
99.1th percentile is      175.88577963591797
99.2th percentile is      198.53758645568678
99.3th percentile is      223.41210289109213
99.4th percentile is      245.9130782695837
99.5th percentile is      305.37021780616374
99.6th percentile is      394.37931711521514
99.7th percentile is      536.6762206010824
99.8th percentile is      587.8919121297967
99.9th percentile is      1102.831863417623
100th percentile is      9254.687475431878
```

In [54]:

```
k=99.9
l=[0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09]
for i in l:
    print(str(k+i)+'th percentile is      '+str(np.percentile(q,k+i)))
```

```
99.9th percentile is      1102.831863417623
99.91000000000001th percentile is      1104.6370384696156
99.92th percentile is      1165.0367551490817
99.93th percentile is      1312.535093554183
99.94000000000001th percentile is      1896.125292553941
99.95th percentile is      2131.877199219537
99.96000000000001th percentile is      2190.2509968870218
99.97th percentile is      3552.345130726965
99.98th percentile is      5147.332026528782
99.99000000000001th percentile is      7262.037731698806
```

In [60]:

```
j=np.percentile(q,100)-np.percentile(q,99.99)
print(j)
```

```
1992.6497437330818
```

In [62]:

```
ind=[]
features=count_vect.get_feature_names()
for i in range(xTrain2.shape[0]-1):
    if i>xTrain2.shape[0]:
        if q[i]>j:
            ind.append(i)
        break
print(len(ind))
print("features names are")
for j in ind:
    print(features[j])
```

```
0
features names are
```

[5.1.3] Feature Importance on BOW, SET 1

[5.1.3.1] Top 10 important features of positive class from SET 1

In [63]:

```
# Please write all the code with proper documentation
#https://stackoverflow.com/questions/11116697/how-to-get-most-informati
ve-features-for-scikit-learn-classifiers?answertab=votes#tab-top
def important_features(vectorizer,classifier,n=20):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()

    topn_class1 = sorted(zip(classifier.coef_[0], feature_names),revers
e=True)[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names),revers
e=False)[:n]
    print("Important words in positive reviews")

    for coef, feat in topn_class1:
        print(class_labels[1], coef, feat)
```

In [64]:

```
important_features(bow_vectorizer,mdl,10)
```

```
Important words in positive reviews
```

```
1 2.1712147686946175e-07 nigh
1 1.6753465864704413e-07 coumadin
1 1.3661527128184713e-07 malomars
1 1.3628618286186516e-07 giaia
1 1.2862405498855716e-07 teeny
1 1.2703235688789542e-07 weiner
1 1.2657130752550168e-07 coffeeshop
1 1.1854493401956663e-07 goosebumps
1 1.1722496740472135e-07 bikes
1 1.133561577997009e-07 protagonist
```

[5.1.3.2] Top 10 important features of negative class from SET 1

In [65]:

```
# Please write all the code with proper documentation
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers?answertab=votes#tab-top
def important_features(vectorizer,classifier,n=20):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()

    topn_class1 = sorted(zip(classifier.coef_[0], feature_names),reverse=True)[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names),reverse=False)[:n]
    print("Important words in negative reviews")
    for coef, feat in topn_class2:
        print(class_labels[0], coef, feat)
```

In [66]:

```
important_features(bow_vectorizer,mdl,10)
```

```
Important words in negative reviews
0 -1.8468898438650493e-07 everythin
0 -1.3788854494085125e-07 xxx
0 -1.3375842878093395e-07 nooo
0 -1.3055603189426877e-07 trys
0 -1.247085748101506e-07 glen
0 -1.1633098214531517e-07 recommemd
0 -1.0337878951400908e-07 klein
```

```
0 -1.022965053366915e-07 whiskeys
0 -1.0081620931921309e-07 energ
0 -1.0061562171622889e-07 pamphlet
```

[5.2] Logistic Regression on TFIDF, SET 2

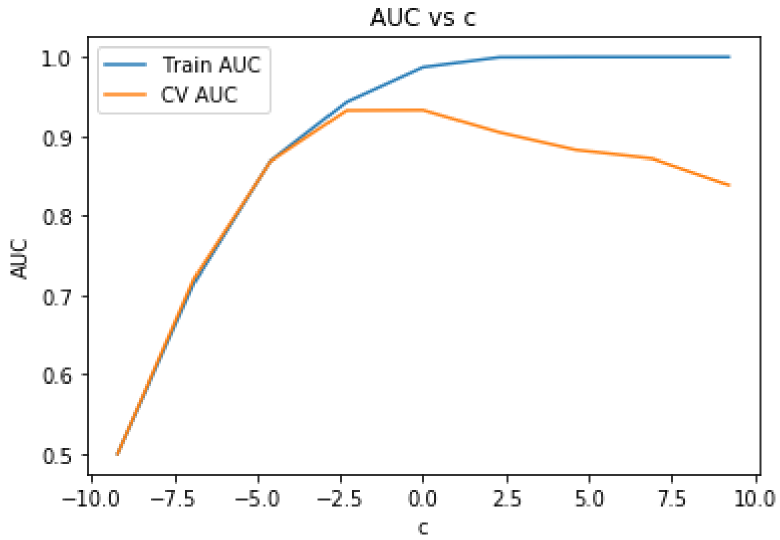
[5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

In [67]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
a=preprocessed_reviews
b=np.array(final['Score'])
tfidf_vectorizer=TfidfVectorizer()
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-an
d-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_select
ion.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3,random_state
=0)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)

#https://datascience.stackexchange.com/questions/12321/difference-betwe
en-fit-and-fit-transform-in-scikit-learn-models
#the above link is been used to clarify whether to use .fit() or .fit_t
ransform().I am using fit_transform() on train data and transform() on
 cv and test data
from sklearn.preprocessing import StandardScaler
xTrain2=bow_vectorizer.fit_transform(xTrain)
```

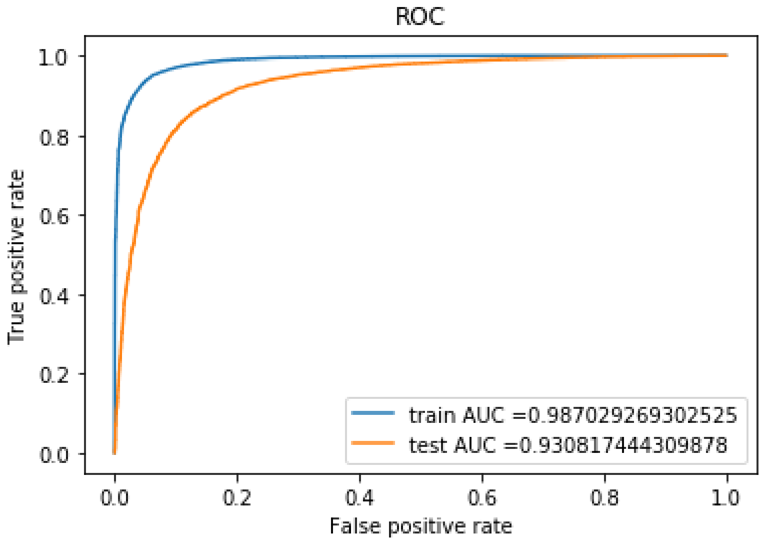
```
x_cv2=bow_vectorizer.transform(x_cv)
xTest2=bow_vectorizer.transform(xTest)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l1',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

```
best lambda is 1
```



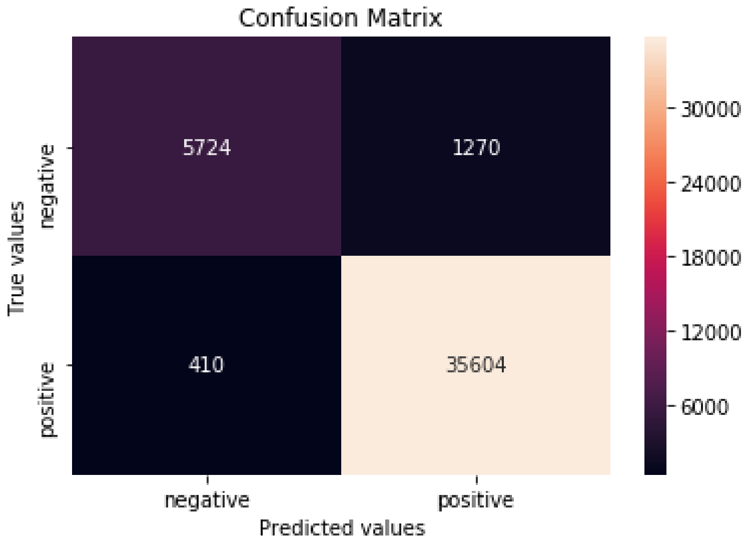
In [68]:

```
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()
```



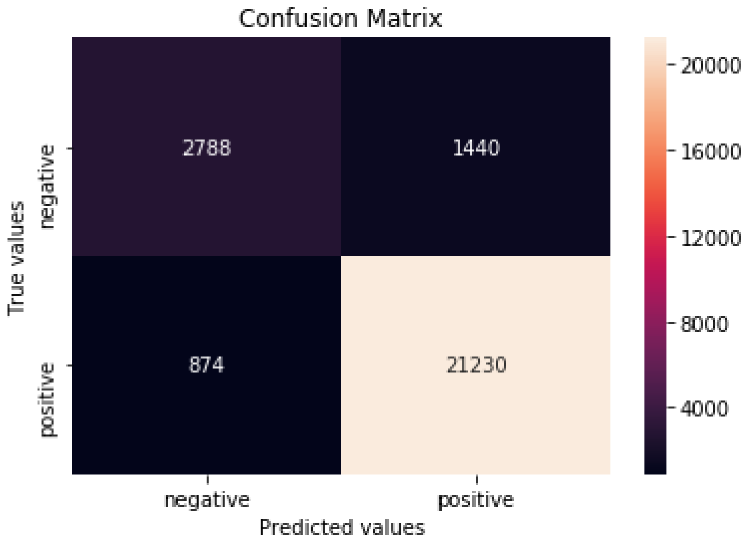
In [69]:

```
#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



In [70]:

```
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```

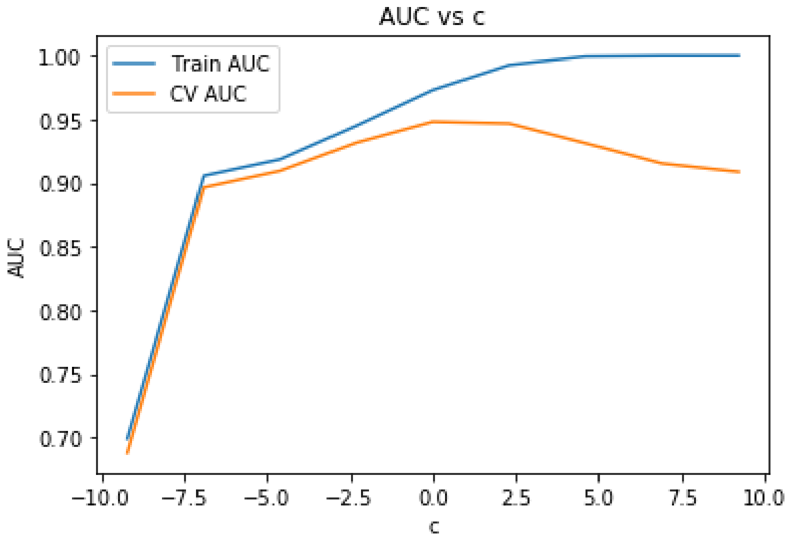


[5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

In [71]:

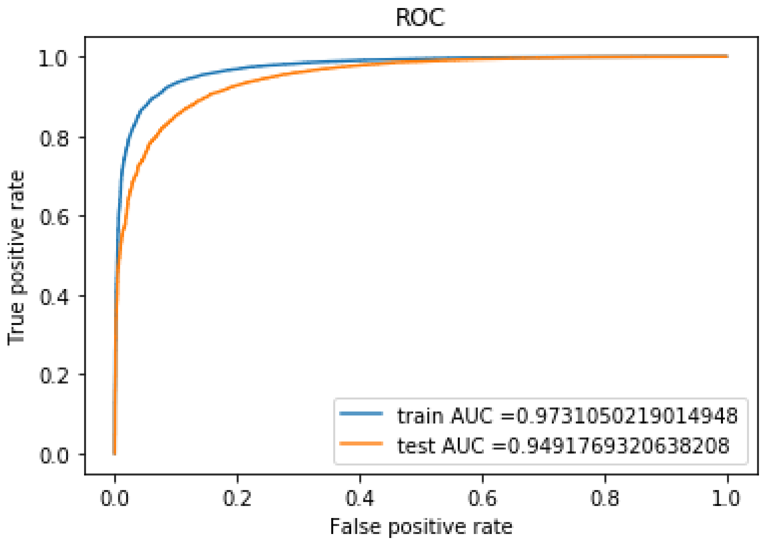
```
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
a=preprocessed_reviews
b=np.array(final['Score'])
tfidf_vectorizer=TfidfVectorizer()
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-an
d-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_select
ion.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3,random_state
=0)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)

#https://datascience.stackexchange.com/questions/12321/difference-betwe
en-fit-and-fit-transform-in-scikit-learn-models
#the above link is been used to clarify whether to use .fit() or .fit_t
ransform().I am using fit_transform() on train data and transform() on
 cv and test data
from sklearn.preprocessing import StandardScaler
xTrain2=tfidf_vectorizer.fit_transform(xTrain)
x_cv2=tfidf_vectorizer.transform(x_cv)
xTest2=tfidf_vectorizer.transform(xTest)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l2',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

```
best lambda is 1
```



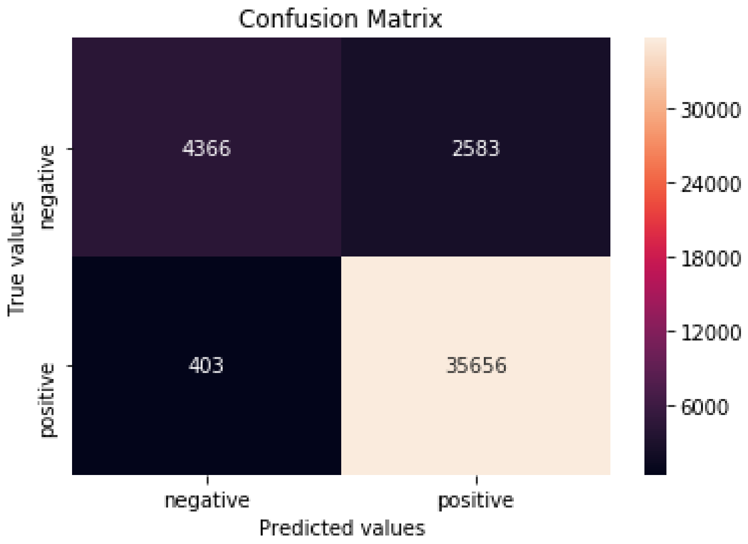
In [72]:

```
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()
```



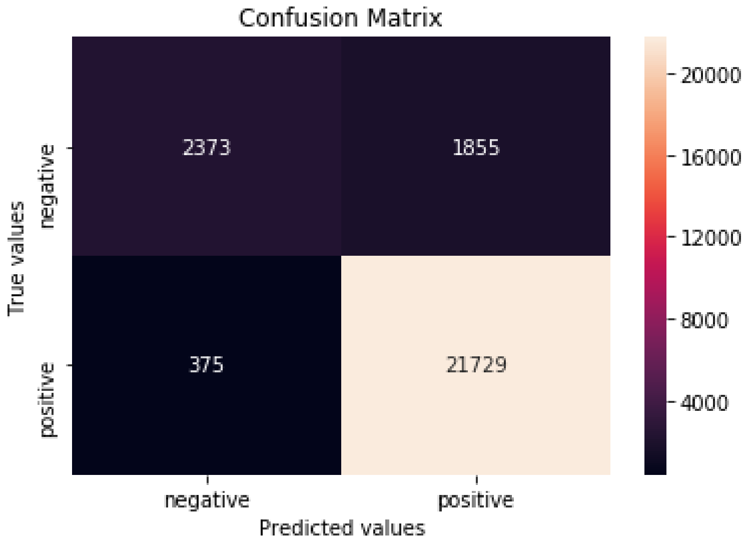
In [73]:

```
#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



In [74]:

```
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



[5.2.3] Feature Importance on TFIDF, SET 2

[5.2.3.1] Top 10 important features of positive class from SET 2

In [75]:
```
# Please write all the code with proper documentation
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers?answertab=votes#tab-top
def important_features(vectorizer,classifier,n=20):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()

    topn_class1 = sorted(zip(classifier.coef_[0], feature_names),reverse=True)[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names),reverse=False)[:n]
    print("Important words in positive reviews")
    for coef, feat in topn_class1:
        print(class_labels[1], coef, feat)
```

In [76]:
```
important_features(tfidf_vectorizer,mdl,10)
```

```
Important words in positive reviews
1 10.452901320905843 great
1 7.756952177614254 best
1 7.302834483389283 delicious
1 6.432852516653854 perfect
1 6.0954763816426585 good
1 5.971450420734337 love
1 5.887791768837542 loves
1 5.756981296939487 nice
1 5.591141209761712 excellent
1 5.306172199424622 wonderful
```

[5.2.3.2] Top 10 important features of negative class from SET 2

In [77]:
```
# Please write all the code with proper documentation
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers?answertab=votes#tab-top
def important_features(vectorizer,classifier,n=20):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()

    topn_class1 = sorted(zip(classifier.coef_[0], feature_names),reverse=True)[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names),reverse=False)[:n]
    print("Important words in negative reviews")

    for coef, feat in topn_class2:
        print(class_labels[0], coef, feat)
```

In [78]:
```
important_features(tfidf_vectorizer,mdl,10)
```

```
Important words in negative reviews
0 -7.356245124048191 not
0 -6.638390414696604 worst
0 -5.456943594152933 disappointed
0 -5.151589876291402 terrible
0 -5.02096943106699 awful
0 -4.979352839748957 disappointing
0 -4.87687509401607 disappointment
0 -4.754073820572765 horrible
0 -4.6066397290475285 unfortunately
0 -4.332741417666117 threw
```

[5.3] Logistic Regression on AVG W2V, SET 3

[5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

In [85]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
a=preprocessed_reviews
b=np.array(final['Score'])
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-an
d-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_select
ion.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)
#this code is used from Assignment_sample_solution.ipynb provided in th
e google classroom
list_of_sentance_train=[]
```

```
for sentance in xTrain:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(len(sent_vectors_train))
#print(sent_vectors_train[0])

list_of_sentance_cv=[]
for sentance in x_cv:
    list_of_sentance_cv.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_cv,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_cv= [];
for sent in tqdm(list_of_sentance_cv):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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)
sent_vectors_cv = np.array(sent_vectors_cv)
print(len(sent_vectors_cv))
#print(sent_vectors_cv[0])

list_of_sentance_test=[]
for sentance in xTest:
    list_of_sentance_test.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_test,min_count=5,size=50, workers=4
)
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_test= [];
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(len(sent_vectors_test))

train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l1',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

```
100%|██████████| 43008/43008 [00:49<00:00, 868.44it/s]
```

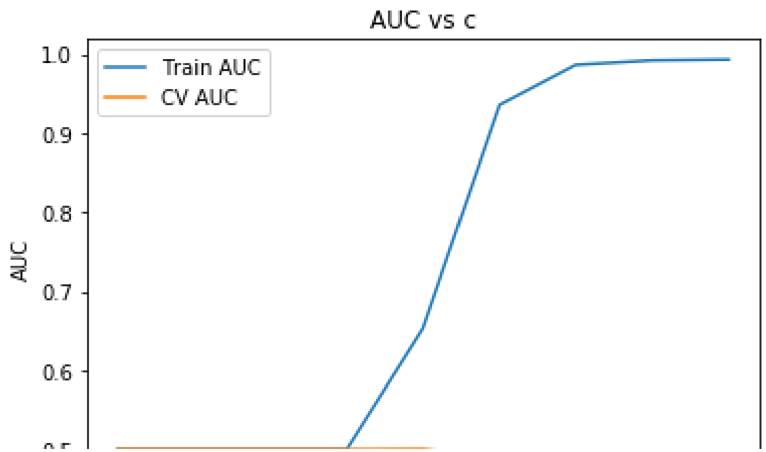
```
43008
```

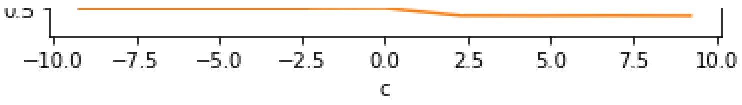
```
100%|██████████| 18433/18433 [00:18<00:00, 992.23it/s]
```

```
18433
```

```
100%|██████████| 26332/26332 [00:27<00:00, 959.49it/s]
```

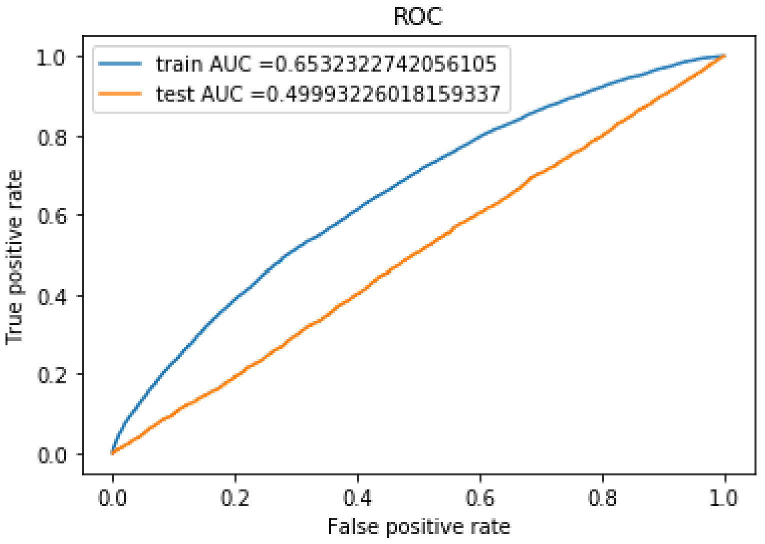
```
26332
best lambda is 1
```





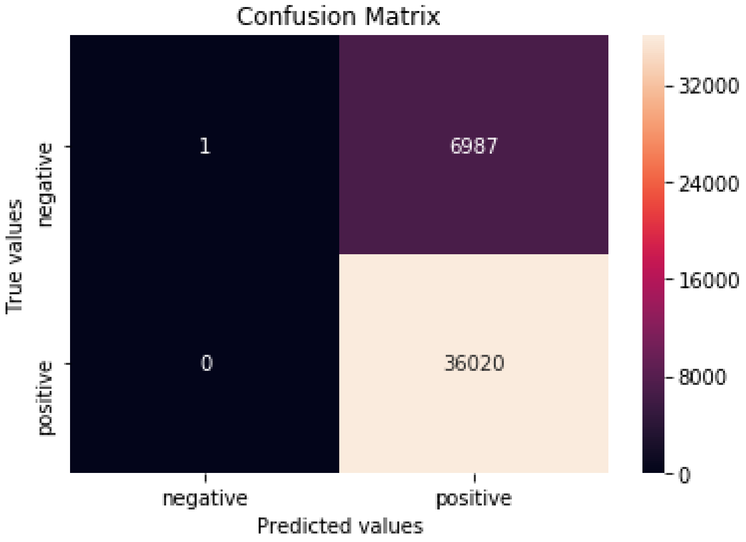
In [86]:

```python
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()
```



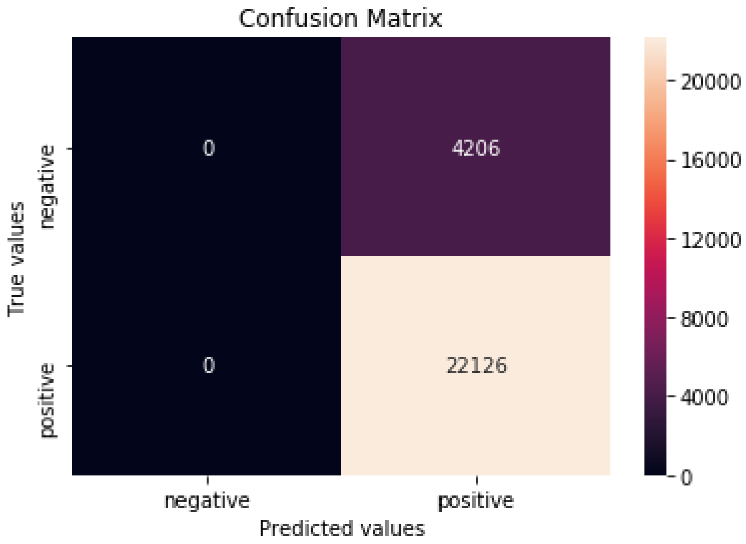
In [87]:

```
#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



In [88]:

```
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



[5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

In [89]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
a=preprocessed_reviews
b=np.array(final['Score'])
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)
#this code is used from Assignment_sample_solution.ipynb provided in the google classroom
list_of_sentance_train=[]
for sentance in xTrain:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
```

```
print(len(sent_vectors_train))
#print(sent_vectors_train[0])

list_of_sentance_cv=[]
for sentance in x_cv:
    list_of_sentance_cv.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_cv,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_cv= [];
for sent in tqdm(list_of_sentance_cv):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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)
sent_vectors_cv = np.array(sent_vectors_cv)
print(len(sent_vectors_cv))
#print(sent_vectors_cv[0])

list_of_sentance_test=[]
for sentance in xTest:
    list_of_sentance_test.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_test,min_count=5,size=50, workers=4
)
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_test= [];
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(len(sent_vectors_test))

train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l2',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

```
100%|██████████| 43008/43008 [00:48<00:00, 890.16it/s]
```

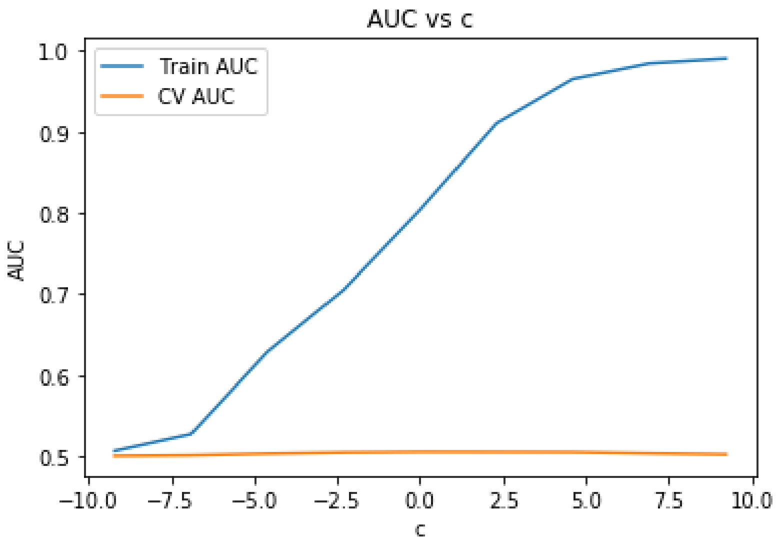
```
43008
```

```
100%|██████████| 18433/18433 [00:19<00:00, 923.64it/s]
```

```
18433
```

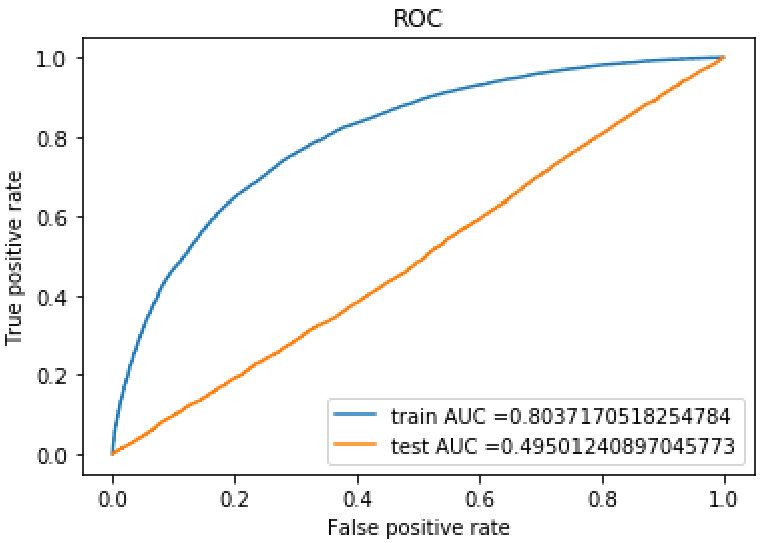
```
100%|██████████| 26332/26332 [00:29<00:00, 893.98it/s]
```

```
26332
best lambda is 1
```



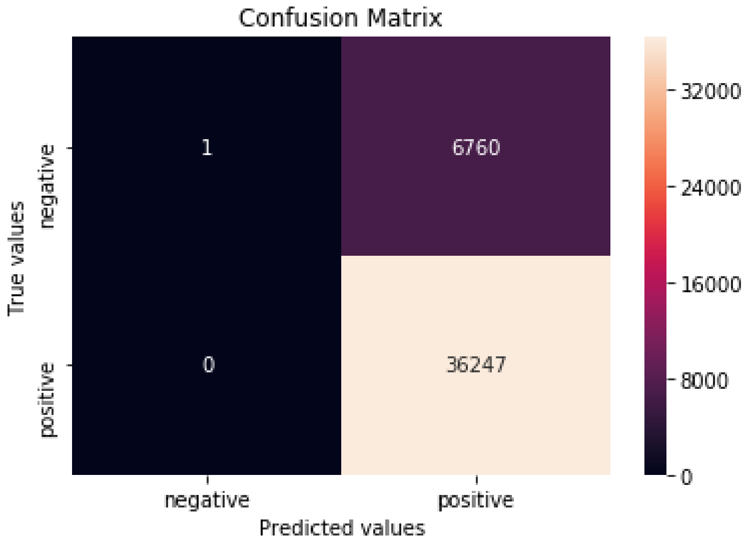
In [90]:

```
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()
```



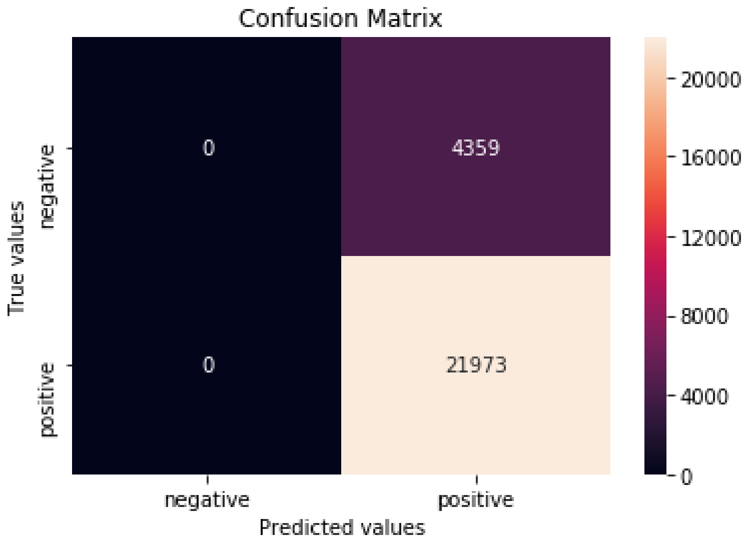
In [91]:

```
#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



In [92]:

```
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



[5.4] Logistic Regression on TFIDF W2V, SET 4

[5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

In [93]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
a=preprocessed_reviews
b=np.array(final['Score'])
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_select
```

```
ion.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)

#this code is used from Assignment_sample_solution.ipynb provided in th
e google classroom
list_of_sentance_train=[]
for sentance in xTrain:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(xTrain)
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
tfidf_feat = model.get_feature_names()

tfidf_vectors_train = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_vectors_train.append(sent_vec)
    row += 1
print(len(tfidf_vectors_train))

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

```
tfidf_vectors_cv = [];
row=0;
for sent in tqdm(list_of_sentance_cv):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_vectors_cv.append(sent_vec)
    row += 1
print(len(tfidf_vectors_cv))

list_of_sentance_test=[]
for sentance in xTest:
    list_of_sentance_test.append(sentance.split())
tfidf_vectors_test = [];
row=0;
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_vectors_test.append(sent_vec)
    row += 1
print(len(tfidf_vectors_test))

train_auc = []
cv_auc = []
```

```python
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l1',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

```
100%|██████████| 43008/43008 [08:38<00:00, 83.02it/s]
  0%|          | 12/18433 [00:00<03:13, 95.04it/s]
```

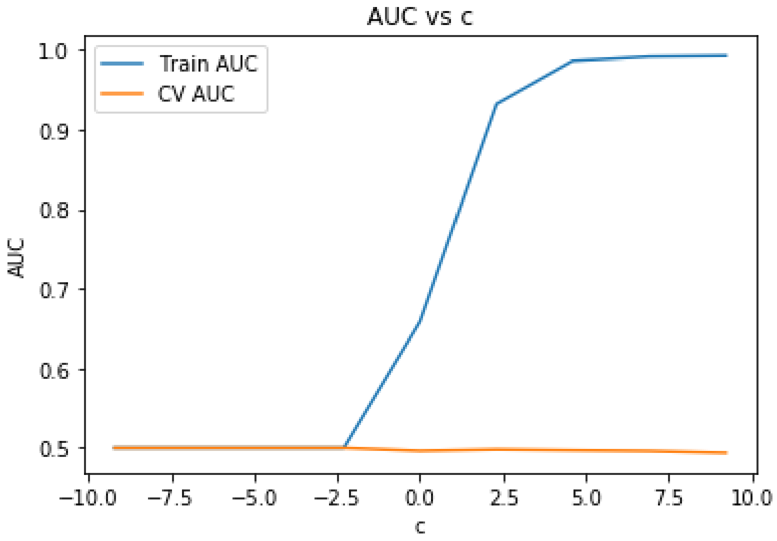
```
43008
```

```
100%|██████████| 18433/18433 [03:40<00:00, 83.59it/s]
  0%|          | 11/26332 [00:00<04:09, 105.38it/s]
```

```
18433
```

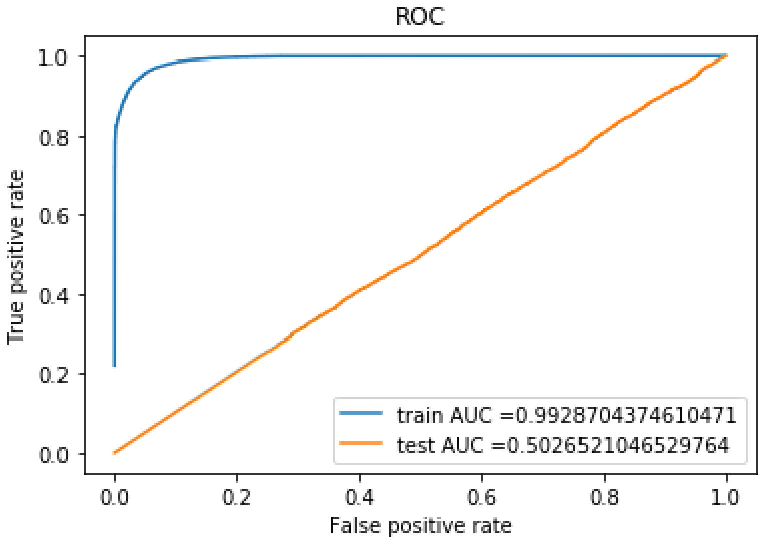
```
100%|██████████| 26332/26332 [05:13<00:00, 84.04it/s]
```

```
26332
best lambda is 9999.0
```



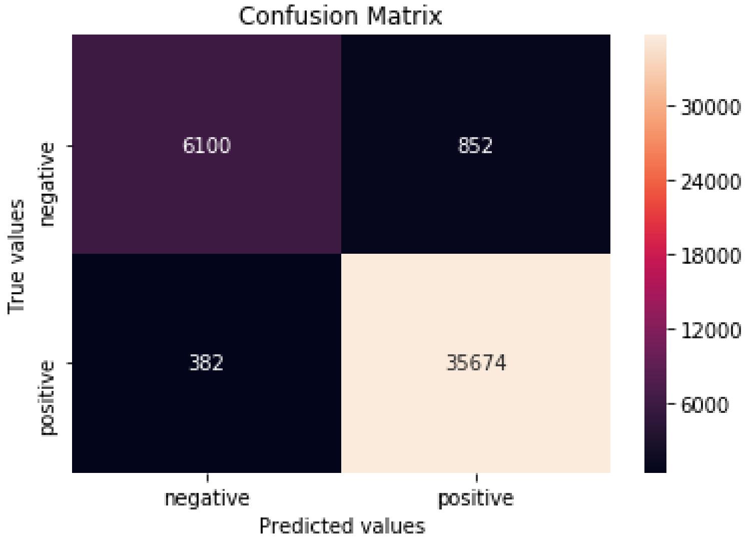
In [94]:

```
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()
```



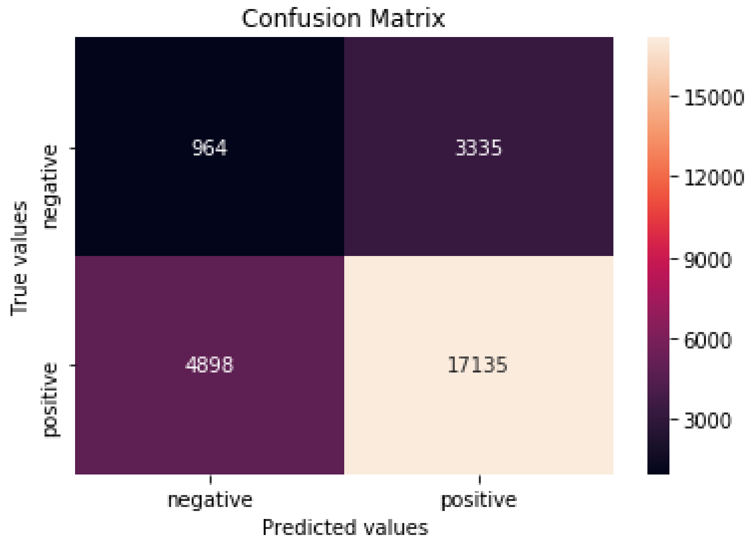
In [95]:

```
#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



In [96]:

```
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l1',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



[5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

In [97]:

```
# Please write all the code with proper documentation# Please write all the code with proper documentation
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
a=preprocessed_reviews
b=np.array(final['Score'])
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
```

```
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)

#this code is used from Assignment_sample_solution.ipynb provided in th
e google classroom
list_of_sentance_train=[]
for sentance in xTrain:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(xTrain)
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
tfidf_feat = model.get_feature_names()

tfidf_vectors_train = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_vectors_train.append(sent_vec)
    row += 1
print(len(tfidf_vectors_train))

list_of_sentance_cv=[]
for sentance in x_cv:
    list_of_sentance_cv.append(sentance.split())
tfidf_vectors_cv = [];
row=0;
for sent in tqdm(list_of_sentance_cv):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_vectors_cv.append(sent_vec)
    row += 1
print(len(tfidf_vectors_cv))

list_of_sentance_test=[]
for sentance in xTest:
    list_of_sentance_test.append(sentance.split())
tfidf_vectors_test = [];
row=0;
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_vectors_test.append(sent_vec)
    row += 1
print(len(tfidf_vectors_test))

train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l2',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

```
100%|██████████| 43008/43008 [10:48<00:00, 66.31it/s]
  0%|          | 0/18433 [00:00<?, ?it/s]
```

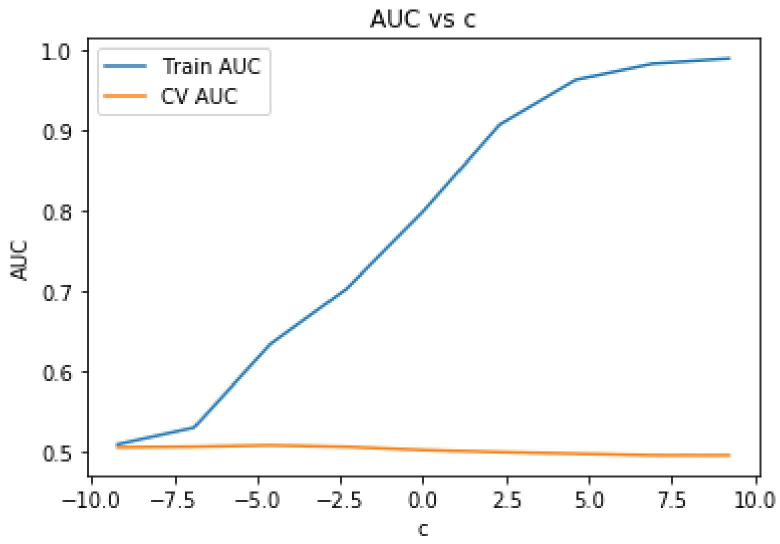
```
43008
```

```
100%|██████████| 18433/18433 [04:52<00:00, 62.99it/s]
  0%|          | 6/26332 [00:00<07:45, 56.53it/s]
```

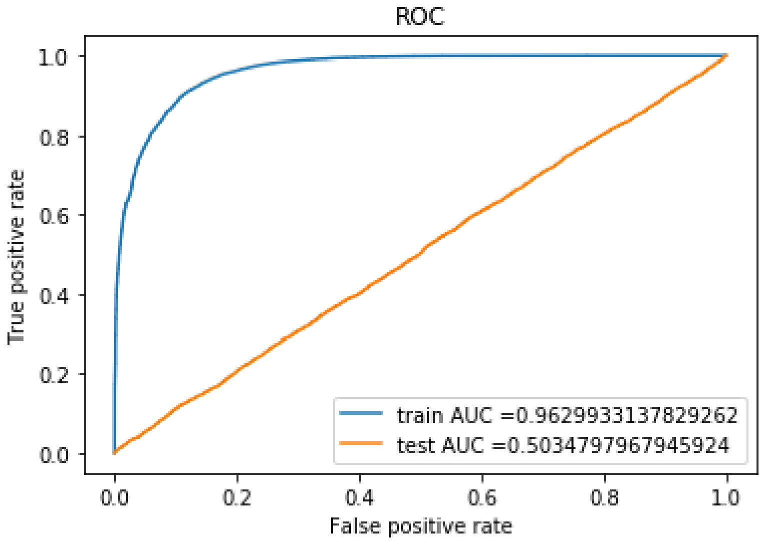
```
18433
```

```
100%|██████████| 26332/26332 [07:56<00:00, 55.24it/s]
```

```
26332
best lambda is 99.0
```

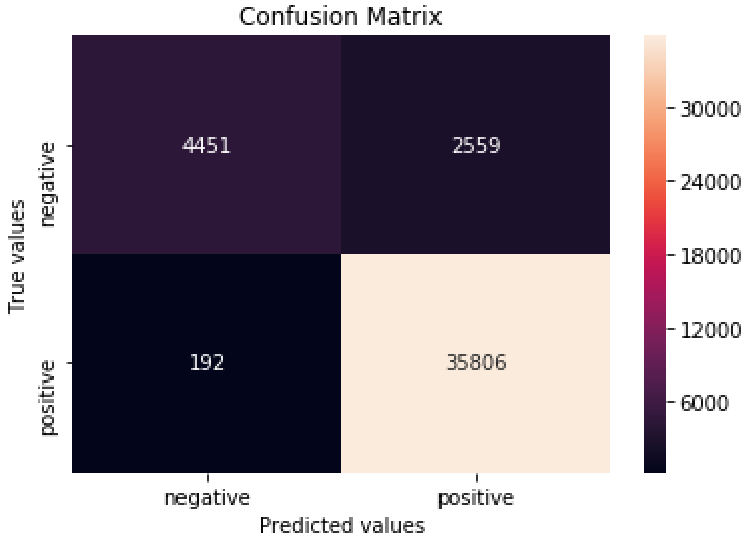


In [98]:
```
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(x
Train2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTes
t2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()
```



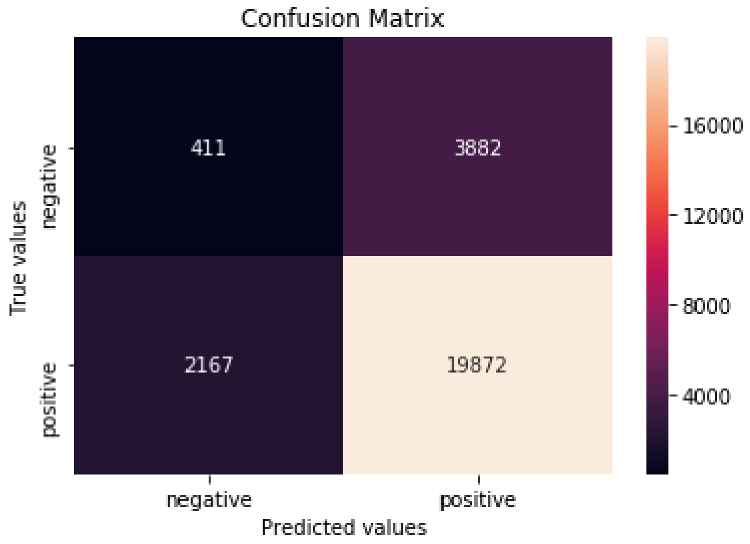
In [99]:

```
#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



In [100]:

```
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



Featuring engineering

```
In [101]: for i in range(len(preprocessed_reviews)):
              preprocessed_reviews[i]=preprocessed_reviews[i]+" "+str(len(final.T
          ext.iloc[i]))
              preprocessed_reviews[i]=preprocessed_reviews[i]+" "+preprocessed_su
          mmary[i]
          preprocessed_reviews[1500]
```

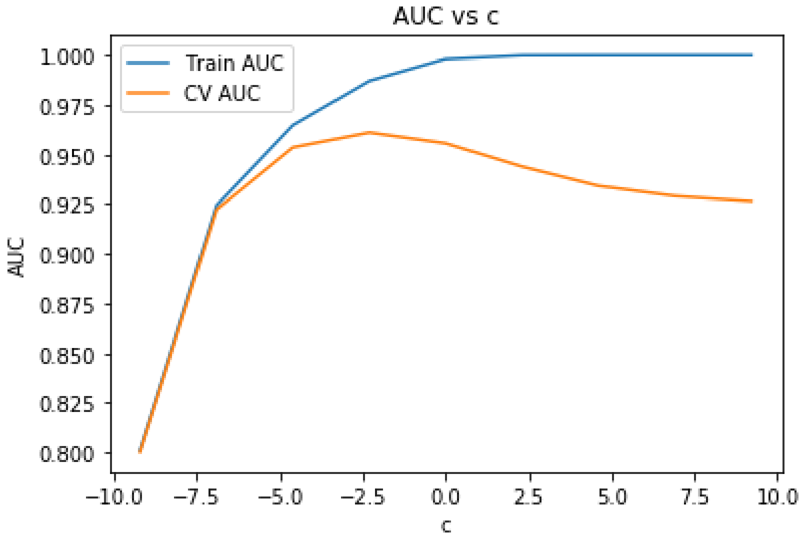
```
Out[101]: 'way hot blood took bite jig lol 59 hot stuff'
```

implementing bow with feature engineered reviews

```
In [102]: # Please write all the code with proper documentation
          import numpy as np
```

```python
import pandas as pd
import matplotlib.pyplot as plt
import math
a=preprocessed_reviews
b=np.array(final['Score'])
tfidf_vectorizer=CountVectorizer()
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3,random_state=0)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)

#https://datascience.stackexchange.com/questions/12321/difference-between-fit-and-fit-transform-in-scikit-learn-models
#the above link is been used to clarify whether to use .fit() or .fit_transform().I am using fit_transform() on train data and transform() on cv and test data
from sklearn.preprocessing import StandardScaler
xTrain2=tfidf_vectorizer.fit_transform(xTrain)
x_cv2=tfidf_vectorizer.transform(x_cv)
xTest2=tfidf_vectorizer.transform(xTest)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l2',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]
    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

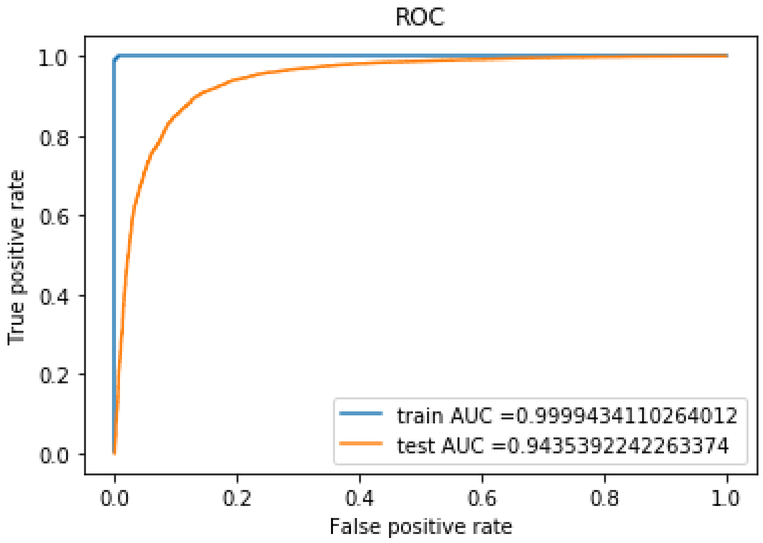
```
best lambda is 9.0
```



In [103]:

```
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l2',C=best_c)
```

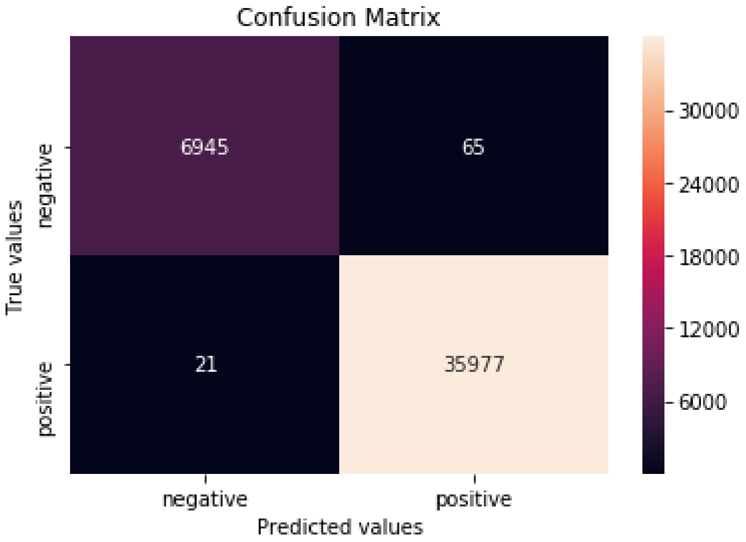
```
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()
```



In [104]:

```
#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
```

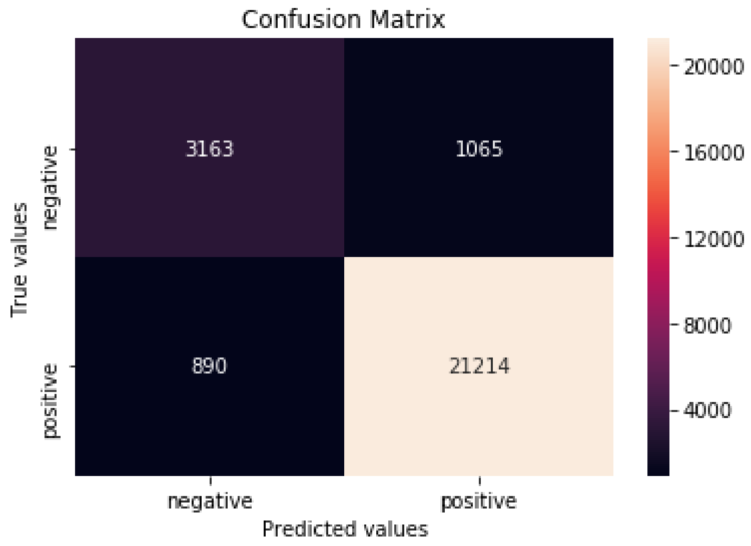
```
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



In [105]:

```
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
```

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```



In [106]:

```
# Please write all the code with proper documentation
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers?answertab=votes#tab-top
def important_features(vectorizer,classifier,n=20):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()

    topn_class1 = sorted(zip(classifier.coef_[0], feature_names),reverse=True)[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names),revers
e=False)[:n]
    print("Important words in positive reviews")

    for coef, feat in topn_class1:
        print(class_labels[1], coef, feat)

important_features(bow_vectorizer,mdl,10)

#### [5.1.3.2] Top 10 important features of negative class from<font co
lor='red'> SET 1</font>

# Please write all the code with proper documentation
#https://stackoverflow.com/questions/11116697/how-to-get-most-informati
ve-features-for-scikit-learn-classifiers?answertab=votes#tab-top
def important_features(vectorizer,classifier,n=20):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()

    topn_class1 = sorted(zip(classifier.coef_[0], feature_names),revers
e=True)[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names),revers
e=False)[:n]
    print("Important words in negative reviews")
    for coef, feat in topn_class2:
        print(class_labels[0], coef, feat)

important_features(bow_vectorizer,mdl,10)
```

```
Important words in positive reviews
1 4.571846704322702 ridder
1 3.997287380373501 moistening
1 3.903154724462512 overdose
1 3.7162695817405313 grateful
1 3.6622478832522134 neurons
1 3.590048353303167 glasses
1 3.5089680937177747 cutoff
1 3.5045857045298394 butsted
1 3.460701526267151 listmania
1 3.4272331285124533 argo
Important words in negative reviews
```

```
0 -5.109814161177726 dvsodium
0 -4.521247335195224 splendas
0 -4.335686463552218 tolorence
0 -4.098862481926717 weirder

0 -4.074089544126789 tumric
0 -3.989739879477029 raely
0 -3.9227295811947354 cheeselike
0 -3.9136949245645742 surging
0 -3.887723917291083 faleevs
0 -3.77298068233284 appreciationwhen
```

implementing tfidf with featured engineered reviews

In [107]:

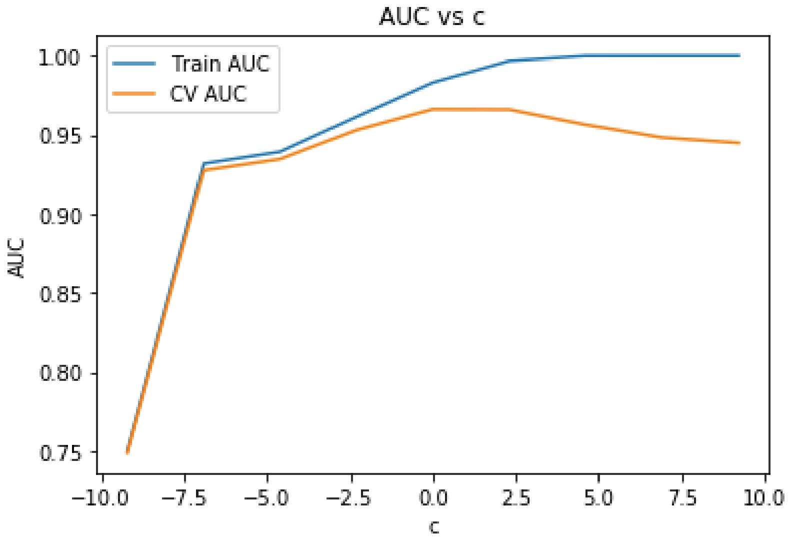
```
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
a=preprocessed_reviews
b=np.array(final['Score'])
tfidf_vectorizer=TfidfVectorizer()
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3,random_state=0)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)

#https://datascience.stackexchange.com/questions/12321/difference-between-fit-and-fit-transform-in-scikit-learn-models
#the above link is been used to clarify whether to use .fit() or .fit_t
```

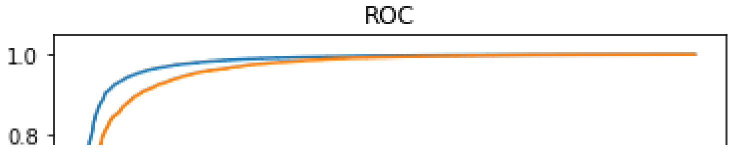
```
ransform().I am using fit_transform() on train data and transform() on
 cv and test data
from sklearn.preprocessing import StandardScaler
xTrain2=tfidf_vectorizer.fit_transform(xTrain)
x_cv2=tfidf_vectorizer.transform(x_cv)
xTest2=tfidf_vectorizer.transform(xTest)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l2',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()

from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l2',C=best_c)
```

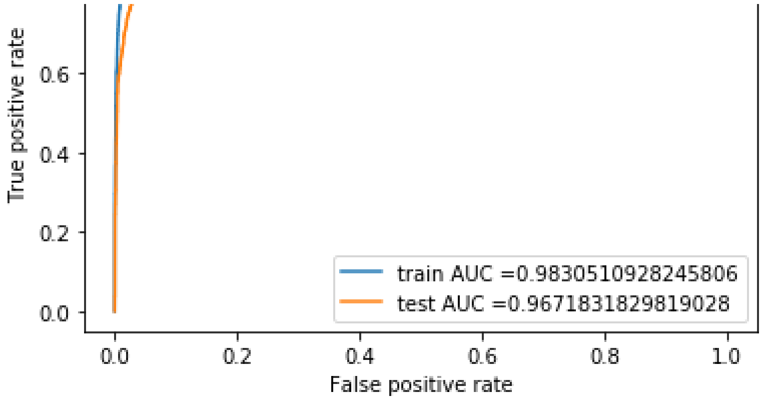
```python
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()

#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()

#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()

# Please write all the code with proper documentation
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers?answertab=votes#tab-top
def important_features(vectorizer,classifier,n=20):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()

    topn_class1 = sorted(zip(classifier.coef_[0], feature_names),reverse=True)[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names),reverse=False)[:n]
    print("Important words in positive reviews")

    for coef, feat in topn_class1:
        print(class_labels[1], coef, feat)

important_features(tfidf_vectorizer,mdl,10)

#### [5.2.3.2] Top 10 important features of negative class from<font color='red'> SET 2</font>

# Please write all the code with proper documentation
```

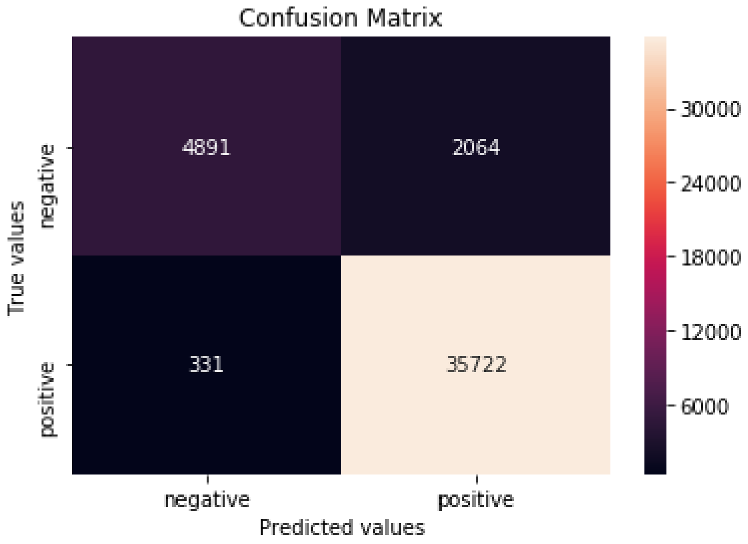
```
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers?answertab=votes#tab-top
def important_features(vectorizer,classifier,n=20):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()

    topn_class1 = sorted(zip(classifier.coef_[0], feature_names),reverse=True)[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names),reverse=False)[:n]
    print("Important words in negative reviews")

    for coef, feat in topn_class2:
        print(class_labels[0], coef, feat)

important_features(tfidf_vectorizer,mdl,10)
```

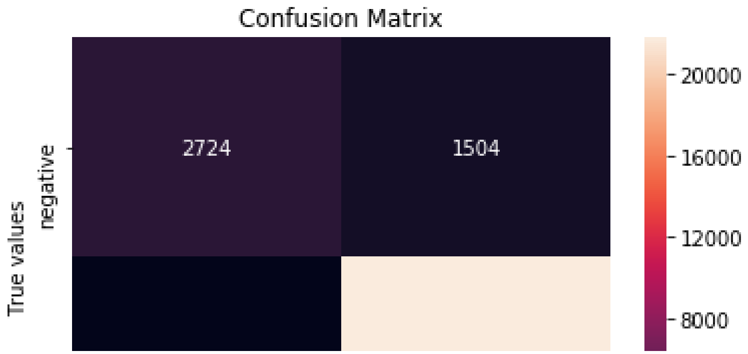
```
best lambda is 1
```

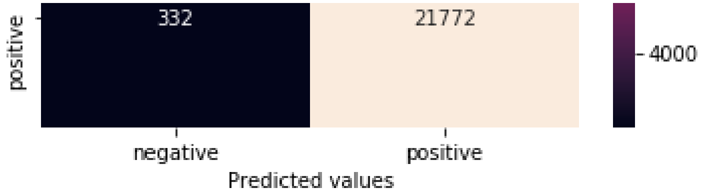





True positive rate
0.0
0.2
0.4
0.6
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate
train AUC =0.9830510928245806
test AUC =0.9671831829819028


Confusion Matrix
True values
negative
positive
4891
2064
331
35722
negative
positive
Predicted values
30000
24000
18000
12000
6000


Confusion Matrix
True values
negative
2724
1504
20000
16000
12000
8000



```
Important words in positive reviews
1 12.56668522708262 great
1 8.897149877433252 best
1 8.075747630474371 delicious
1 7.765936887542516 good
1 7.219318526553382 excellent
1 6.74059082984215 perfect
1 6.655310276230468 love
1 6.27330779731868 loves
1 6.021275901060111 nice
1 5.47730831364018 yummy
Important words in negative reviews
0 -9.80099029205899 not
0 -6.932016316581686 disappointed
0 -6.375400678232137 worst
0 -6.035880163775718 terrible
0 -5.721055824884405 disappointing
0 -5.650816259898504 horrible
0 -5.340090260810579 awful
0 -5.000744980167505 poor
0 -4.893960075382707 disappointment
0 -4.66685373660894 weak
```

implementing avg w2v with featured engineered reviews

In [108]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
a=preprocessed_reviews
b=np.array(final['Score'])
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-an
d-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_select
ion.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)
#this code is used from Assignment_sample_solution.ipynb provided in th
e google classroom
list_of_sentance_train=[]
for sentance in xTrain:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(len(sent_vectors_train))
#print(sent_vectors_train[0])

list_of_sentance_cv=[]
for sentance in x_cv:
    list_of_sentance_cv.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_cv,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_cv= [];
for sent in tqdm(list_of_sentance_cv):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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)
sent_vectors_cv = np.array(sent_vectors_cv)
print(len(sent_vectors_cv))
#print(sent_vectors_cv[0])

list_of_sentance_test=[]
for sentance in xTest:
    list_of_sentance_test.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_test,min_count=5,size=50, workers=4
)
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_test= [];
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(len(sent_vectors_test))

train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l2',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

```
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()

#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```

```
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusio
n-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```

```
100%|██████████| 43008/43008 [01:00<00:00, 709.61it/s]
```

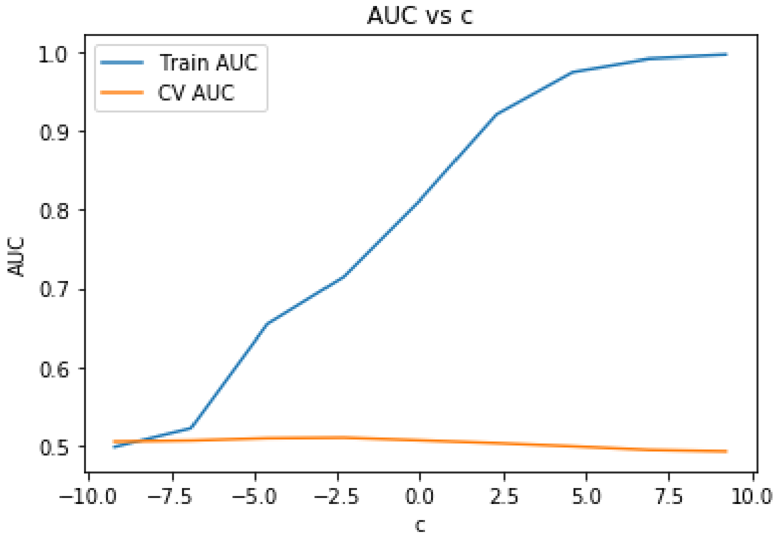
```
43008
```

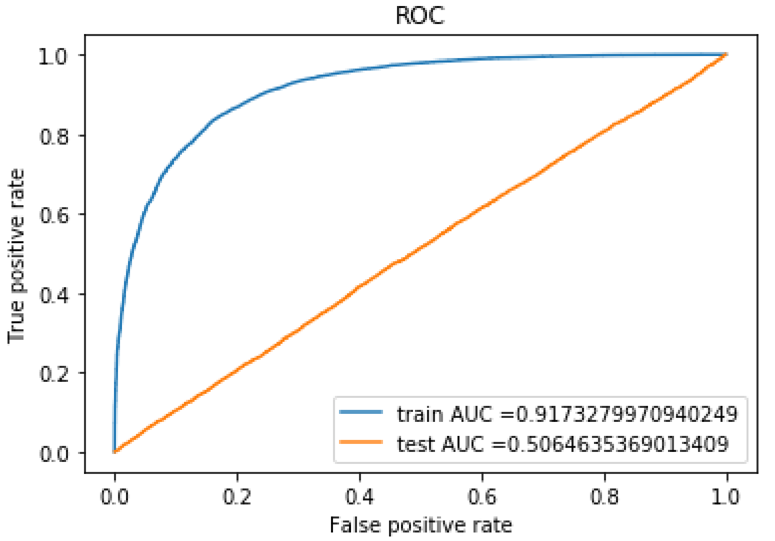
```
100%|██████████| 18433/18433 [00:22<00:00, 811.57it/s]
```

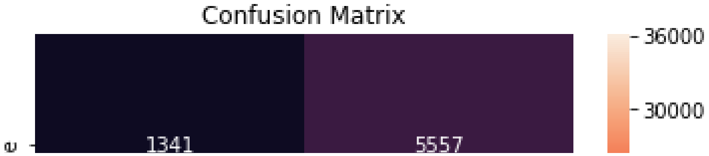
```
18433
```

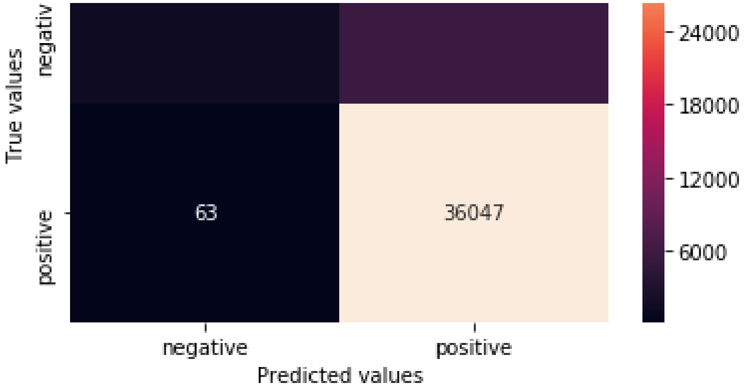
```
100%|██████████| 26332/26332 [00:32<00:00, 800.15it/s]
```

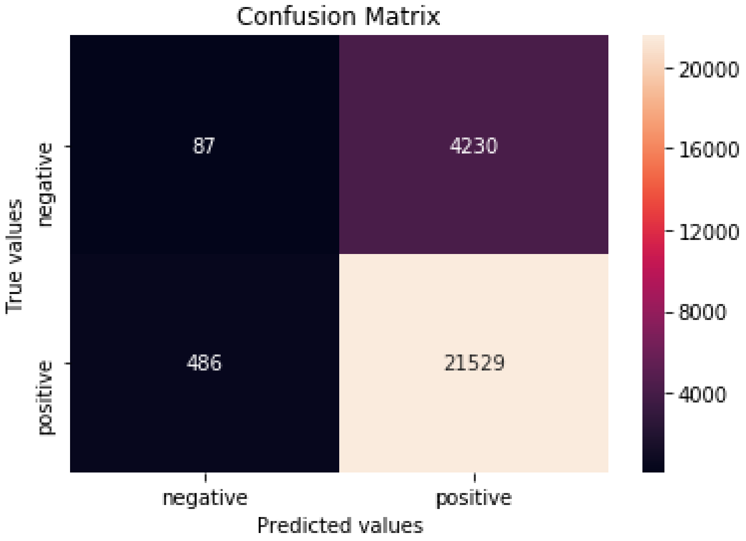
```
26332
best lambda is 9.0
```


AUC vs c
Train AUC
CV AUC
AUC
1.0
0.9
0.8
0.7
0.6
0.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
c


ROC
train AUC =0.9173279970940249
test AUC =0.5064635369013409
True positive rate
False positive rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Confusion Matrix
1341
5557
36000
30000





implementing tfidf w2v with featured engineered reviews

In [109]:

```
# Please write all the code with proper documentation# Please write all
```

```
 the code with proper documentation
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
a=preprocessed_reviews
b=np.array(final['Score'])
a=a[:100000]
b=b[:100000]
from sklearn.model_selection import train_test_split
#https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-an
d-test-datasets-using-scikit-learn-e7cf6eb5e0d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_select
ion.train_test_split.html
#used above references for train,text and cv splitting
from sklearn.model_selection import train_test_split
x, xTest, y, yTest = train_test_split(a,b, test_size = 0.3)
xTrain, x_cv, yTrain, y_cv= train_test_split(x, y, test_size =0.3)

#this code is used from Assignment_sample_solution.ipynb provided in th
e google classroom
list_of_sentance_train=[]
for sentance in xTrain:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(xTrain)
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
tfidf_feat = model.get_feature_names()

tfidf_vectors_train = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_vectors_train.append(sent_vec)
    row += 1
print(len(tfidf_vectors_train))

list_of_sentance_cv=[]
for sentance in x_cv:
    list_of_sentance_cv.append(sentance.split())
tfidf_vectors_cv = [];
row=0;
for sent in tqdm(list_of_sentance_cv):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_vectors_cv.append(sent_vec)
    row += 1
print(len(tfidf_vectors_cv))

list_of_sentance_test=[]
for sentance in xTest:
    list_of_sentance_test.append(sentance.split())
tfidf_vectors_test = [];
row=0;
for sent in tqdm(list_of_sentance_test):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_vectors_test.append(sent_vec)
    row += 1
print(len(tfidf_vectors_test))

train_auc = []
cv_auc = []
c = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in c:
    mdl = LogisticRegression(penalty='l2',C=i)
    mdl.fit(xTrain2,yTrain)

    y_train_pred =  mdl.predict_proba(xTrain2)[:,1]
    y_cv_pred =  mdl.predict_proba(x_cv2)[:,1]

    train_auc.append(roc_auc_score(yTrain,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

k= cv_auc.index(max(cv_auc))
print("best lambda is {}".format(1//c[k]))
best_c=1//c[k]
a_len=len(c)
for j in range(a_len):
    c[j]=math.log(c[j])
len(train_auc)
plt.plot(c,train_auc, label='Train AUC')
plt.plot(c,cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.show()
```

```python
from sklearn.metrics import roc_curve, auc
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
train_fpr, train_tpr, thresholds = roc_curve(yTrain,mdl.predict_proba(xTrain2)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(yTest,mdl.predict_proba(xTest2)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC")
plt.show()

#confusion matrix for train data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTrain2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTrain,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```

```python
#confusion matrix for test data
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
mdl=LogisticRegression(penalty='l2',C=best_c)
mdl.fit(xTrain2,yTrain)
acc3=mdl.predict(xTest2)
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
fig= confusion_matrix(yTest,acc3)
labels= ["negative", "positive"]
data= pd.DataFrame(fig, index = labels,columns = labels)
sns.heatmap(data,annot=True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```

```
100%|██████████| 43008/43008 [11:57<00:00, 59.93it/s]
  0%|          | 9/18433 [00:00<03:35, 85.33it/s]
```

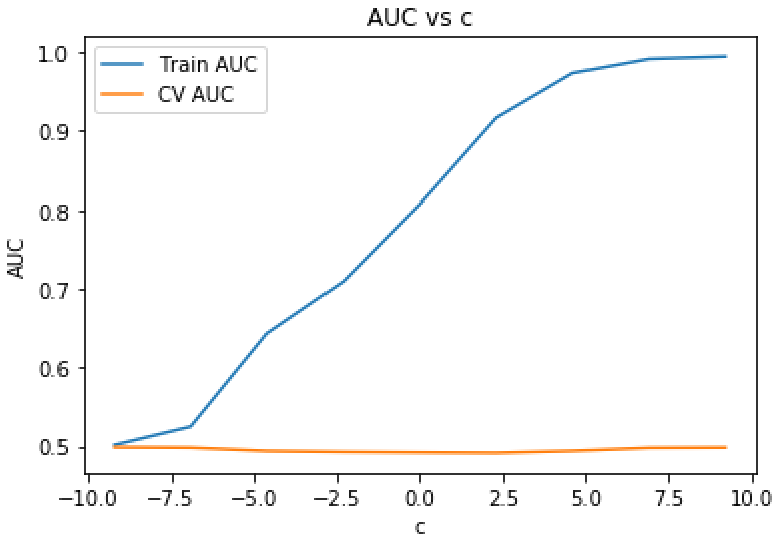
```
43008
```

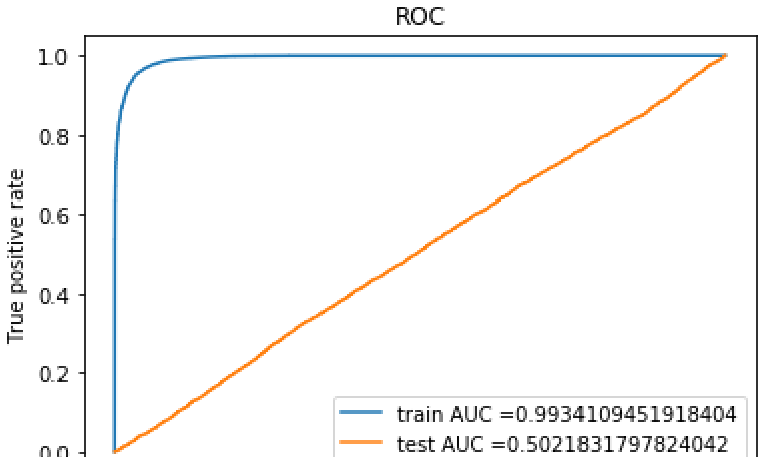
```
100%|██████████| 18433/18433 [04:42<00:00, 65.29it/s]
  0%|          | 10/26332 [00:00<04:42, 93.12it/s]
```

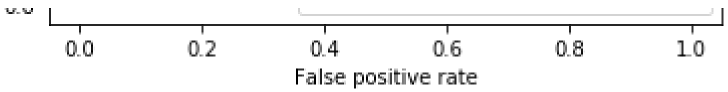
```
18433
```

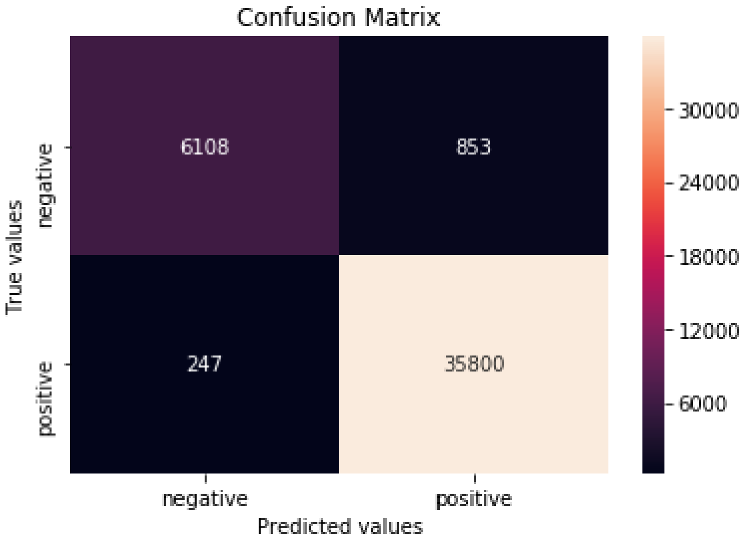
```
100%|██████████| 26332/26332 [06:41<00:00, 65.59it/s]
```

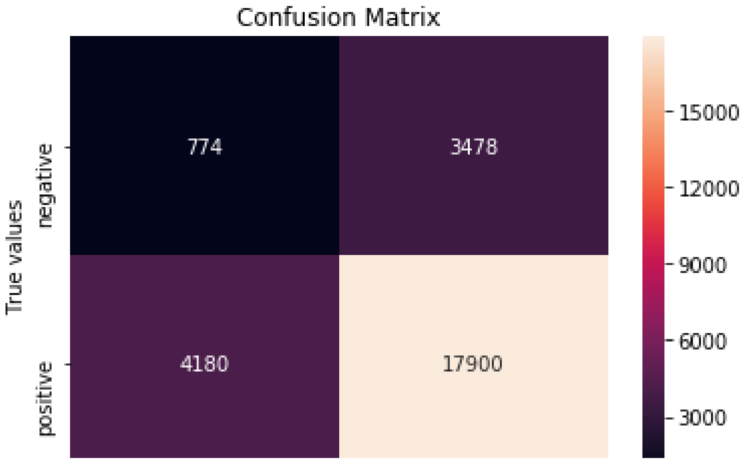
```
26332
best lambda is 9999.0
```


AUC vs c
Train AUC
CV AUC
AUC
1.0
0.9
0.8
0.7
0.6
0.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
c


ROC
True positive rate
1.0
0.8
0.6
0.4
0.2
0.0
train AUC =0.9934109451918404
test AUC =0.5021831797824042


0.0
0.2
0.4
0.6
0.8
1.0
False positive rate


Confusion Matrix
True values
negative
positive
6108
853
247
35800
negative
positive
Predicted values
30000
24000
18000
12000
6000


Confusion Matrix
True values
negative
positive
774
3478
4180
17900
15000
12000
9000
6000
3000

[6] Conclusions

In [111]:

```
# Please compare all your models using Prettytable library
#http://zetcode.com/python/prettytable/
#used the reference for pretty table representation


from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["MODEL","FEATURING ENGINEERING","C","AUC","REGULIZATIO
N"]

x.add_row(["BOW","NO",1,0.930,"L1"])
x.add_row(["BOW","NO",9,0.917,"L2"])
x.add_row(["TFIDF","NO",1,0.930,"L1"])
x.add_row(["TFIDF","NO",1,0.949,"L2"])
x.add_row(["AVG W2V","NO",1,0.499,"L1"])
x.add_row(["AVG W2V","NO",1,0.495,"L2"])
x.add_row(["TFIDF W2V","NO",4,0.502,"L1"])
x.add_row(["TFIDF W2V","NO",1,0.503,"L2"])
x.add_row(["BOW","YES",1,0.943,"L2"])
x.add_row(["TFIDF","YES",1,0.967,"L2"])
x.add_row(["AVG W2V","YES",1,0.506,"L2"])
x.add_row(["TFIDF W2V","YES",1,0.502,"L2"])


print(x)
```

```
+-----------+-----------------------+---+-------+-------------+
|   MODEL   | FEATURING ENGINEERING | C |  AUC  | REGULIZATION |
+-----------+-----------------------+---+-------+--------------+
|    BOW    |           NO          | 1 |  0.93 |      L1      |
|    BOW    |           NO          | 9 | 0.917 |      L2      |
|   TFIDF   |           NO          | 1 |  0.93 |      L1      |
|   TFIDF   |           NO          | 1 | 0.949 |      L2      |
|  AVG W2V  |           NO          | 1 | 0.499 |      L1      |
|  AVG W2V  |           NO          | 1 | 0.495 |      L2      |
| TFIDF W2V |           NO          | 4 | 0.502 |      L1      |
| TFIDF W2V |           NO          | 1 | 0.503 |      L2      |
|    BOW    |          YES          | 1 | 0.943 |      L2      |
|   TFIDF   |          YES          | 1 | 0.967 |      L2      |
|  AVG W2V  |          YES          | 1 | 0.506 |      L2      |
| TFIDF W2V |          YES          | 1 | 0.502 |      L2      |
+-----------+-----------------------+---+-------+--------------+
```