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.notebook import tqdm_notebook
from tqdm import tqdm
import os
```

In [40]:

```
# 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[40]:

	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	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2

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

In [5]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	(
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	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 [41]:

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

Out[41]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
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]:

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

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

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

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

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

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

```python
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
```

```
sent0 = 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

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 [01:33<00:00, 941.75it/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.
import warnings
warnings.filterwarnings("ignore")
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm_notebook(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())
```

In [25]:

```
preprocessed_summary[1500]
```

Out[25]:

```
'hot stuff'
```

[4] Featurization

[4.1] BAG OF WORDS

In [26]:

```
#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 [27]:

```python
#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 [28]:

```
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 [29]:

```
# 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 [30]:

```
# 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.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

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

```
[('fantastic', 0.8506678342819214), ('good', 0.826988697052002), ('awes
ome', 0.815509557723999), ('excellent', 0.8149617910385132), ('wonderfu
l', 0.7769489288330078), ('terrific', 0.7742729187011719), ('perfect',
0.7280958890914917), ('nice', 0.7102464437484741), ('amazing', 0.708073
9140510559), ('fabulous', 0.6975551843643188)]
==================================================
[('greatest', 0.8067414164543152), ('best', 0.7243658304214478), ('tast
iest', 0.6971726417541504), ('nastiest', 0.6742880344390869), ('closes
t', 0.6116491556167603), ('disgusting', 0.6098245978355408), ('terribl
e', 0.6080969572067261), ('horrible', 0.5934323072433472), ('nicest',
0.585574209690094), ('awful', 0.5843217372894287)]
```

In [31]:

```
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 [32]:

```
# 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 [08:26<00:00, 173.30it/s]
```

```
87773
50
```

[4.4.1.2] TFIDF weighted W2v

In [33]:

```
# 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 [34]:

```
# 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 [1:58:02<00:00, 12.39it/s]
```

[5] Assignment 7: SVM

1. **Apply SVM on these feature sets**

    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Procedure**

    - You need to work with 2 versions of SVM
        - Linear kernel
        - RBF kernel
    - When you are working with linear kernel, use SGDClassifier' with hinge loss because it is computationally less expensive.
    - When you are working with 'SGDClassifier' with hinge loss and trying to find the AUC score, you would have to use CalibratedClassifierCV

- Similarly, like kdtree of knn, when you are working with RBF kernel it's better to reduce the number of dimensions. You can put min_df = 10, max_features = 500 and consider a sample size of 40k points.

3. **Hyper paramter tuning (find best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**

   - Find the best hyper parameter which will give the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

4. **Feature importance**

   - When you are working on the linear kernel with BOW or TFIDF please print the top 10 best features for each of the positive and negative classes.

5. **Feature engineering**

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

6. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure. Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test. Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

7. [Conclusion]

- [You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link]

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

Applying SVM

[5.1] Linear SVM

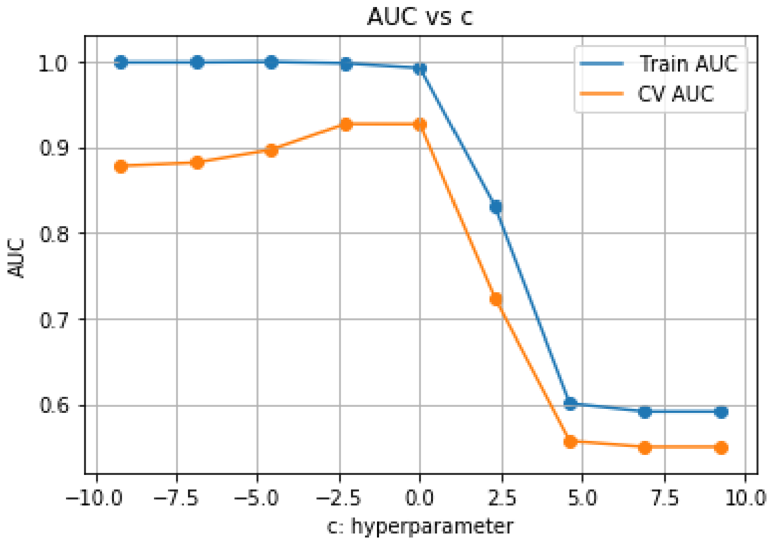
[5.1.1] Applying Linear SVM on BOW, SET 1

In [103]:

```
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import math
from sklearn.model_selection  import train_test_split
from sklearn.metrics import accuracy_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
bow_vect=CountVectorizer()
x=preprocessed_reviews
y=np.array(final['Score'])
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
x_train,x_cv,y_train,y_cv=train_test_split(x_train,y_train,test_size=0.3)

fbowx_tr=bow_vect.fit_transform(x_train)
fbowx_cv=bow_vect.transform(x_cv)
fbowx_te=bow_vect.transform(x_test)

std=StandardScaler(with_mean=False) #Standardizing Data
fbowx_tr=std.fit_transform(fbowx_tr)
fbowx_cv=std.transform(fbowx_cv)
fbowx_te=std.transform(fbowx_te)

svm_=SGDClassifier(alpha=0.0001)
svm=CalibratedClassifierCV(svm_,cv=3).fit(fbowx_tr,y_train)
```

In [115]:

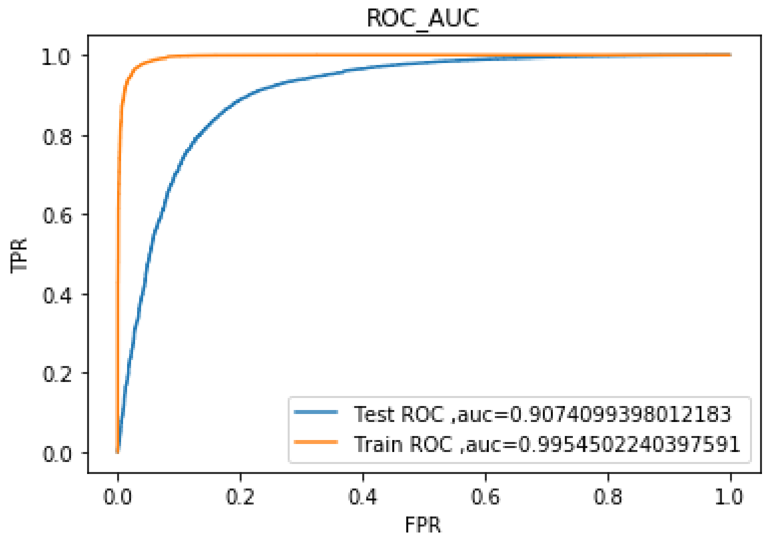
```
from sklearn.metrics import roc_auc_score
auc_cv=[]
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm_notebook(c): #simple cv using for loop
    svm_=SGDClassifier(alpha=i)
    svm=CalibratedClassifierCV(svm_,cv=3).fit(fbowx_tr,y_train)
    pred_cv = svm.predict_proba(fbowx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svm.predict_proba(fbowx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))
best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```



```
Best c value for max auc = 0.1
```

In [60]:
```
#Plotting ROC_AUC curve
svm_=SGDClassifier(alpha=best_c)
svm=CalibratedClassifierCV(svm_,cv=3).fit(fbowx_tr,y_train)
pred_te=svm.predict_proba(fbowx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
```

```
pred_tr=svm.predict_proba(fbowx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_tr)))
plt.title('ROC_AUC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```



In [51]:

```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t
```

```
def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

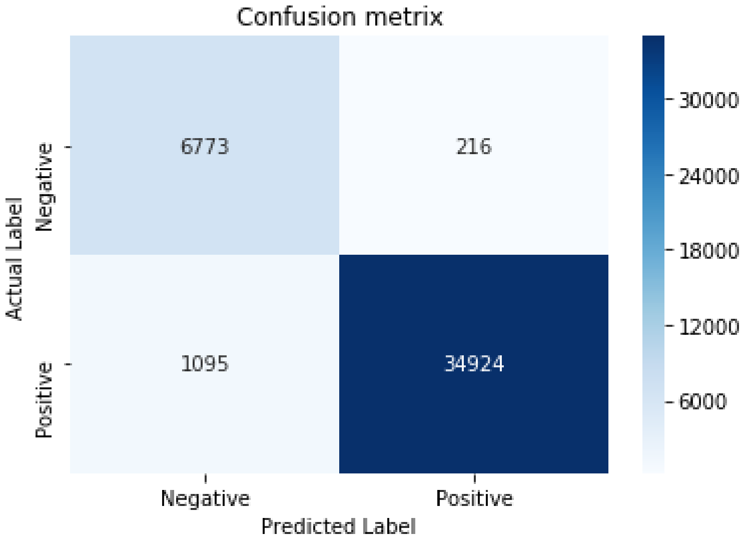
In [62]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9396332219075597 for threshold 0.74
Train confusion matrix
```

Out[62]:

```
Text(33,0.5,'Actual Label')
```



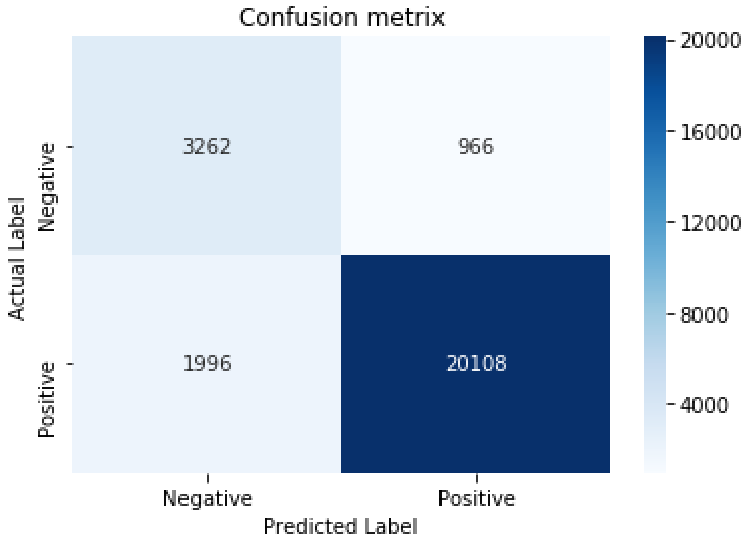
In [63]:

```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9396332219075597 for threshold 0.74
Train confusion matrix
```

Out[63]:

```
Text(33,0.5,'Actual Label')
```



Top 10 features

In [79]:

```
# Please write all the code with proper documentation
all_features = bow_vect.get_feature_names()
svm=SGDClassifier(alpha=0.1).fit(fbowx_tr,y_train)
weight=svm.coef_
positive=np.argsort(weight)[:,::-1]
negative=np.argsort(weight)

print('Top 10 positive features :')
for i in list(positive[0][0:10]):
    print(all_features[i])
```

```
Top 10 positive features :
great
good
best
love
delicious
loves
nice
```

```
wonderful
favorite
excellent
```

In [80]:

```
print('Top 10 negative features :')
for i in list(negative[0][0:10]):
    print(all_features[i])
```

```
Top 10 negative features :
not
disappointed
worst
terrible
awful
horrible
disappointing
threw
disappointment
waste
```

[5.1.2] Applying Linear SVM on TFIDF, SET 2

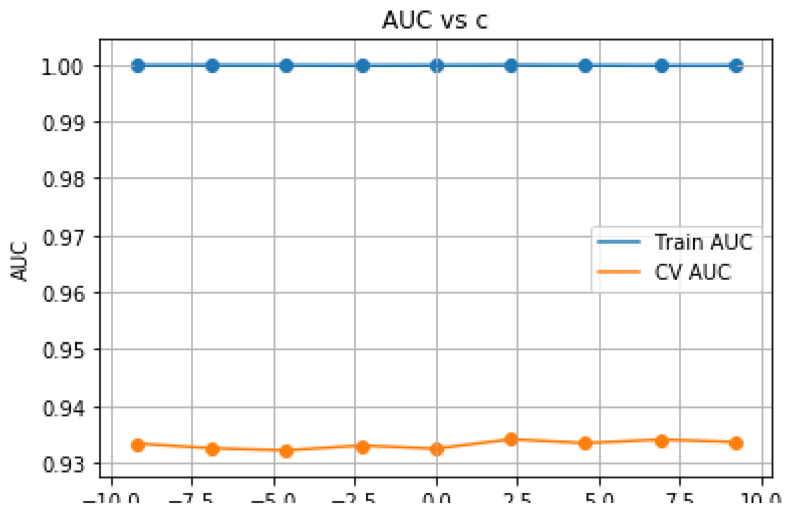
In [87]:

```
# Please write all the code with proper documentation
tf_vect=TfidfVectorizer(ngram_range=(1,2),min_df=10)
#tf_vect.fit(preprocessed_reviews)

ftfx_tr=tf_vect.fit_transform(x_train)
ftfx_cv=tf_vect.transform(x_cv)
ftfx_te=tf_vect.transform(x_test)

std = StandardScaler(with_mean=False)
ftfx_tr=std.fit_transform(ftfx_tr)#Standardizing Data
ftfx_cv=std.transform(ftfx_cv)
ftfx_te=std.transform(ftfx_te)
```

In [88]:

```
auc_cv=[]
auc_train=[]
```

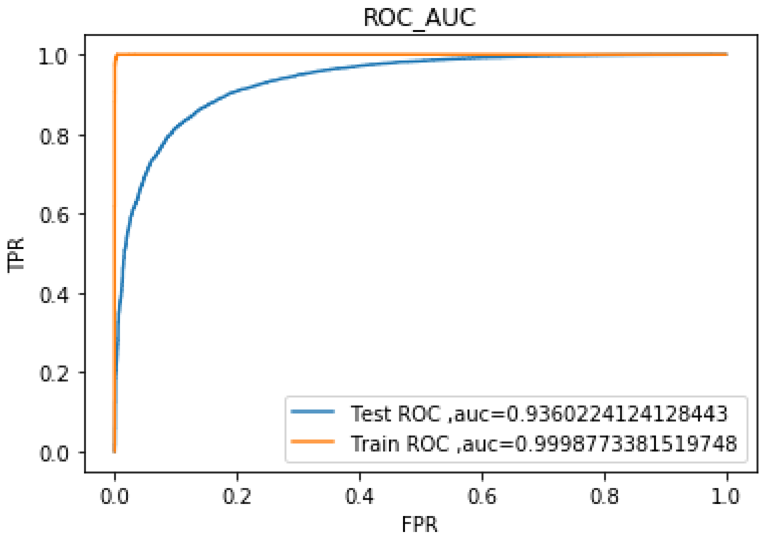
```
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm_notebook(c): #simple cv using for loop
    svm_=SGDClassifier(alpha=best_c)
    svm=CalibratedClassifierCV(svm_,cv=3).fit(ftfx_tr,y_train)
    pred_cv = svm.predict_proba(ftfx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svm.predict_proba(ftfx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))
best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```



```
Best c value for max auc = 10
```

In [74]:

```
#Plotting ROC_AUC curve
svm=CalibratedClassifierCV(svm_,cv=3).fit(ftfx_tr,y_train)
pred_te=svm.predict_proba(ftfx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svm.predict_proba(ftfx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_tes
t,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_tr
ain,pred_tr)))
plt.title('ROC_AUC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



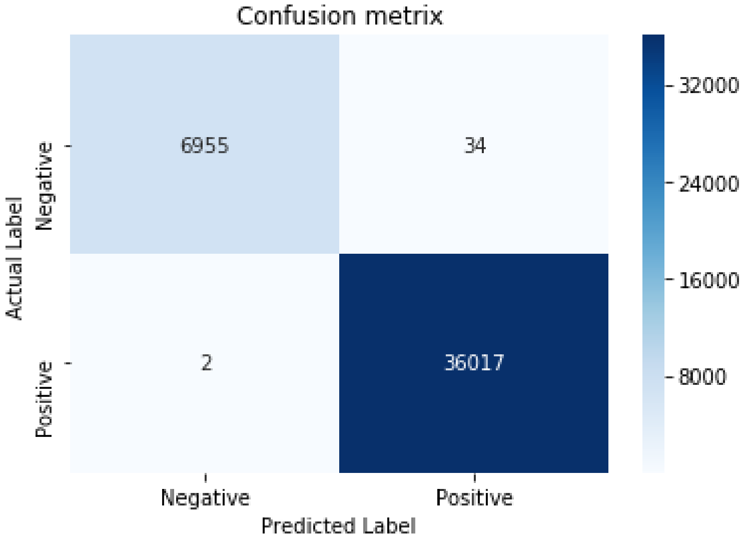
In [67]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9950799563501229 for threshold 0.587
Train confusion matrix
```

Out[67]:

```
Text(33,0.5,'Actual Label')
```



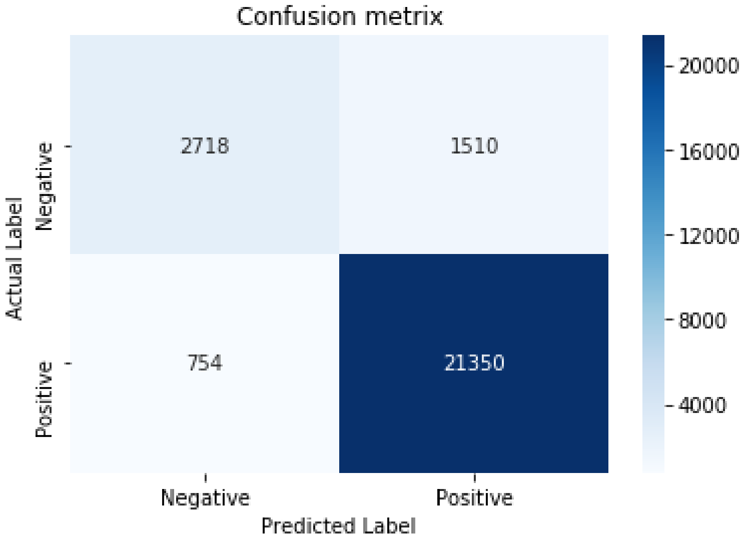
In [68]:

```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, b
est_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9950799563501229 for threshold 0.587
Train confusion matrix
```

Out[68]:

```
Text(33,0.5,'Actual Label')
```



[5.1.3] Applying Linear SVM on AVG W2V, SET 3

In [104]:

```
# Please write all the code with proper documentation
#Avg word2vec for train data
sent_train_list=[]
for sentence in x_train:
    sent_train_list.append(sentence.split())
w2v_model=Word2Vec(sent_train_list,min_count=5,size=50, workers=4)
```

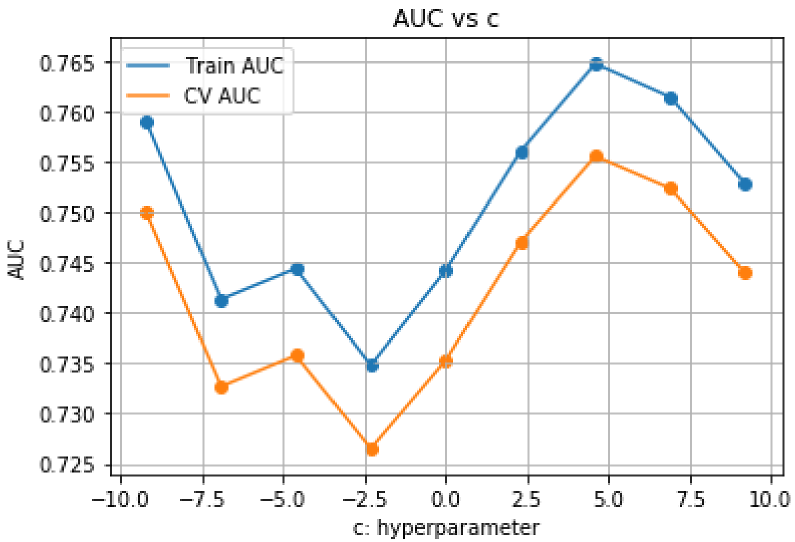
```
w2v_words = list(w2v_model.wv.vocab)

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

#Avg word2vec for cv data
sent_cv_list=[]
for sentence in x_cv:
    sent_cv_list.append(sentence.split())

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


#Avg word2vec for test data
sent_test_list=[]
for sentence in x_test:
    sent_test_list.append(sentence.split())

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


#This code is copied and modified from :https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW#scrollTo=3-XGItt4PSx0
```

```
43008
50


18433
50


26332
```

```
50
```

In [105]:

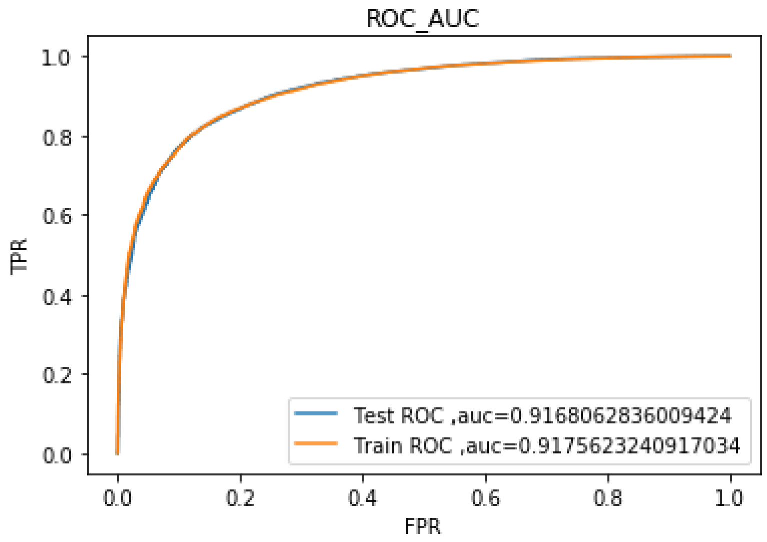
```
aw2vx_tr=sent_train_vectors
aw2vx_cv=sent_cv_vectors
aw2vx_te=sent_test_vectors

auc_cv=[]
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm_notebook(c):
    svm_=SGDClassifier(alpha=best_c)
    svm=CalibratedClassifierCV(svm_,cv=3).fit(aw2vx_tr,y_train)
    pred_cv= svm.predict_proba(aw2vx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svm.predict_proba(aw2vx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))
best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used
 to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```



```
Best c value for max auc = 100
```

In [103]:

```
#Plotting ROC_AUC curve
svm_=SGDClassifier(alpha=best_c)
svm=CalibratedClassifierCV(svm_,cv=3).fit(aw2vx_tr,y_train)
pred_te=svm.predict_proba(aw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svm.predict_proba(aw2vx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_tr)))
plt.title('ROC_AUC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



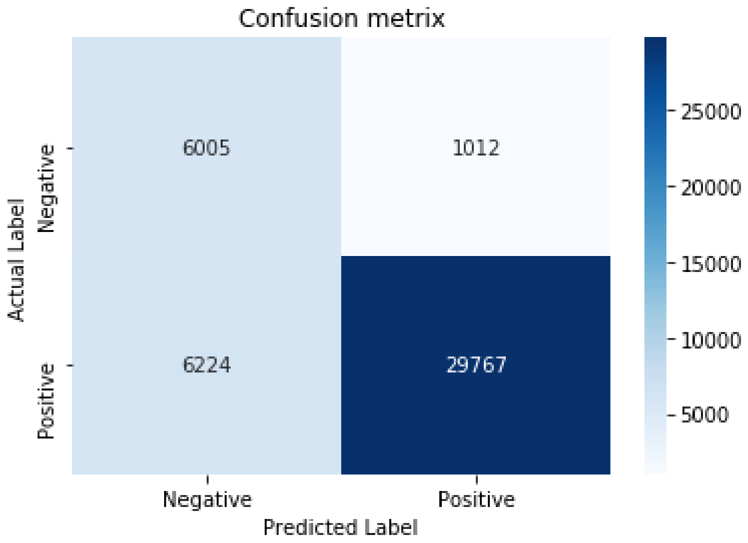
In [104]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.7077871751281446 for threshold 0.8
35
Train confusion matrix
```

Out[104]:

```
Text(33,0.5,'Actual Label')
```



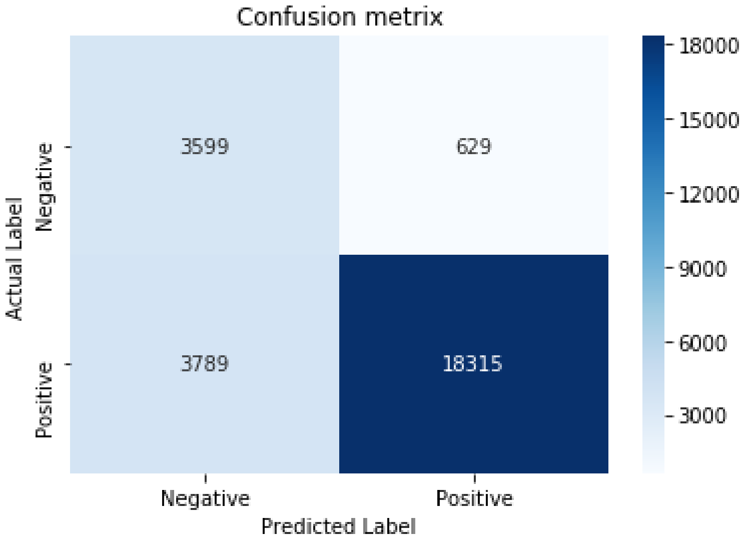
In [105]:

```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.7077871751281446 for threshold 0.835
Train confusion matrix
```

Out[105]:

```
Text(33,0.5,'Actual Label')
```



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

In [82]:

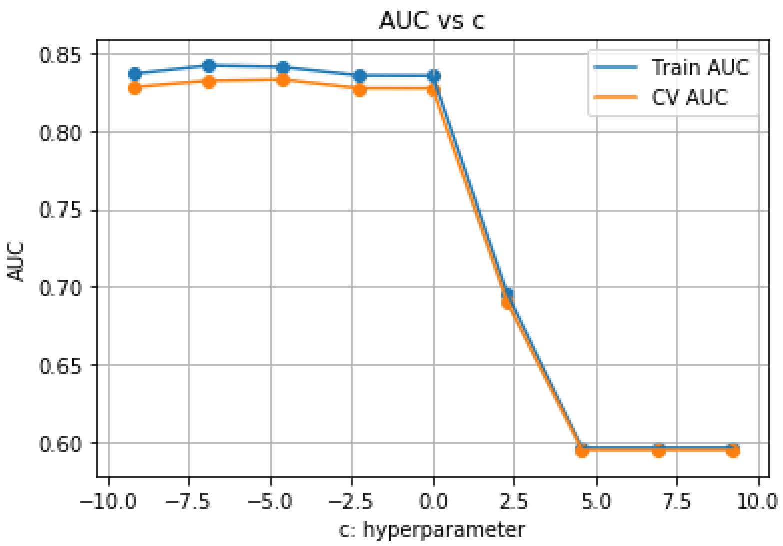
```
# Please write all the code with proper documentation
sent_train_list=[]
for sentence in x_train:
    sent_train_list.append(sentence.split())
w2v_model=Word2Vec(sent_train_list,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10, max_features
=500)
tf_idf_matrix=tf_idf_vect.fit_transform(x_train)
tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect
.idf_)))

#Train data
tfidf_sent_train_vectors = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm_notebook(sent_train_list): # for each review/sentence
```

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

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

#Test data
sent_test_list=[]
for sentence in x_test:
    sent_test_list.append(sentence.split())
tfidf_sent_test_vectors = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm_notebook(sent_test_list): # 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_test_vectors.append(sent_vec)
    row += 1
```

In [41]:

```
tfw2vx_tr=tfidf_sent_train_vectors
tfw2vx_cv=tfidf_sent_cv_vectors
tfw2vx_te=tfidf_sent_test_vectors
auc_cv=[]
```

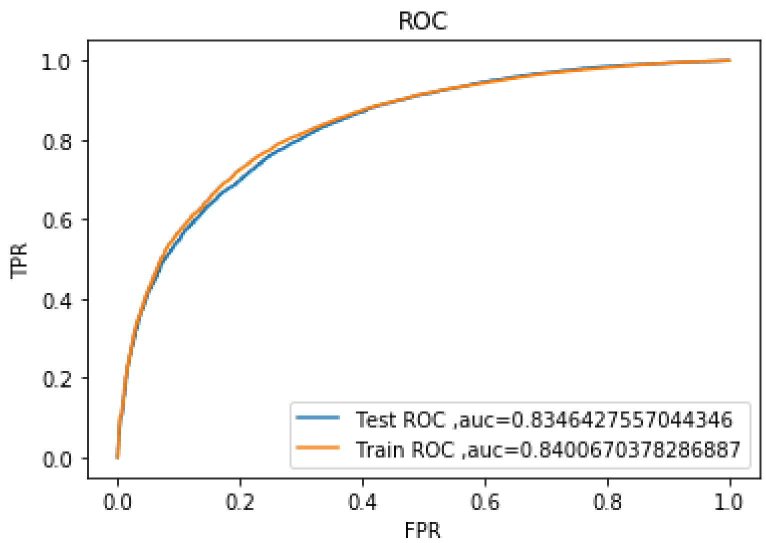
```
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm_notebook(c):
    svm_=SGDClassifier(alpha=i)
    svm=CalibratedClassifierCV(svm_,cv=3).fit(tfw2vx_tr,y_train)
    pred_cv = svm.predict_proba(tfw2vx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svm.predict_proba(tfw2vx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))

best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used
 to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```



```
Best c value for max auc = 0.01
```

In [43]:

```
#Plotting ROC Curve
svm_=SGDClassifier(alpha=best_c)
svm=CalibratedClassifierCV(svm_,cv=3).fit(tfw2vx_tr,y_train)
pred_te=svm.predict_proba(tfw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svm.predict_proba(tfw2vx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_tes
t,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_tr
ain,pred_tr)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



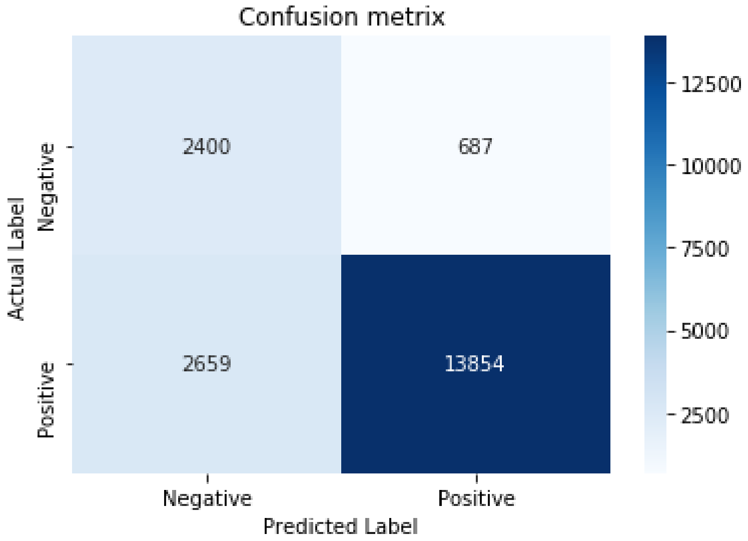
In [74]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.6522646085538402 for threshold 0.851
Train confusion matrix
```

Out[74]:

```
Text(33,0.5,'Actual Label')
```



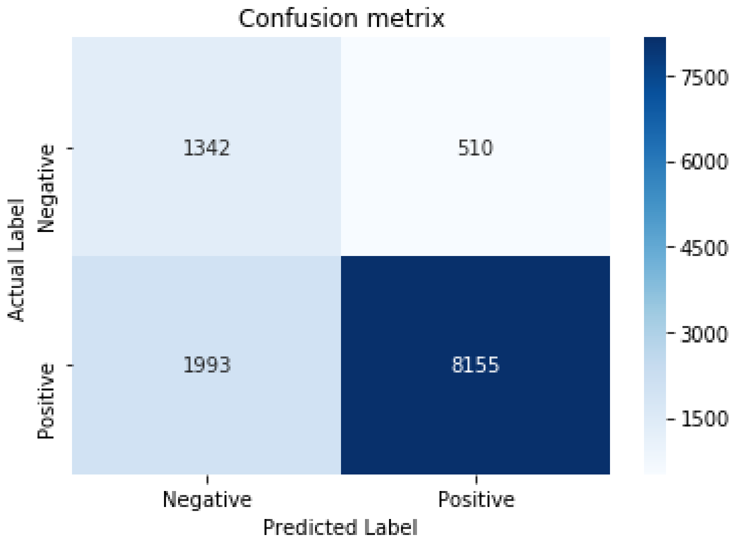
In [75]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, b
est_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.6522646085538402 for threshold 0.8
51
Train confusion matrix
```

Out[75]:

```
Text(33,0.5,'Actual Label')
```



[5.2] RBF SVM

[5.2.1] Applying RBF SVM on BOW, SET 1

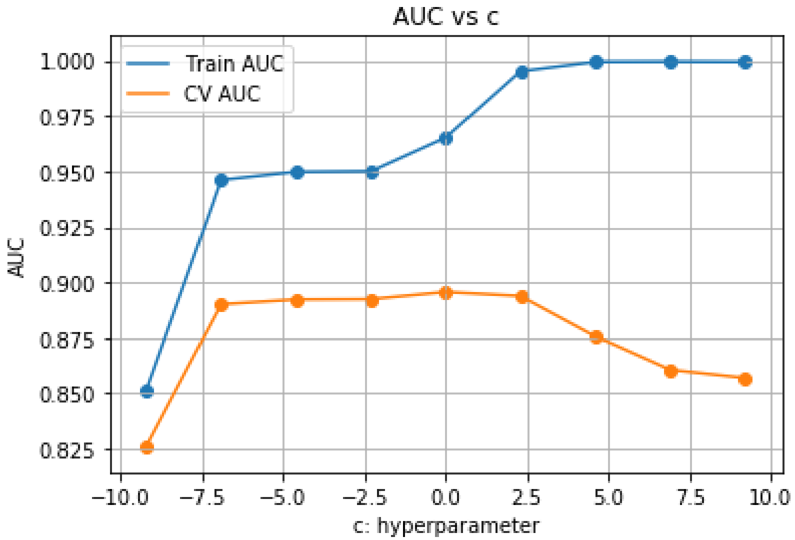
In [89]:

```
# Please write all the code with proper documentation
from sklearn.svm import SVC
bow_vect=CountVectorizer(min_df = 10, max_features = 500)
x=preprocessed_reviews[:40000]
y=np.array(final['Score'][:40000])
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
x_train,x_cv,y_train,y_cv=train_test_split(x_train,y_train,test_size=0.3)

fbowx_tr=bow_vect.fit_transform(x_train)
fbowx_cv=bow_vect.transform(x_cv)
fbowx_te=bow_vect.transform(x_test)
```

```
std=StandardScaler(with_mean=False) #Standardizing Data
fbowx_tr=std.fit_transform(fbowx_tr)
fbowx_cv=std.transform(fbowx_cv)
fbowx_te=std.transform(fbowx_te)

svc=SVC(C=1.0,probability=True).fit(fbowx_tr,y_train)
```

In [46]:

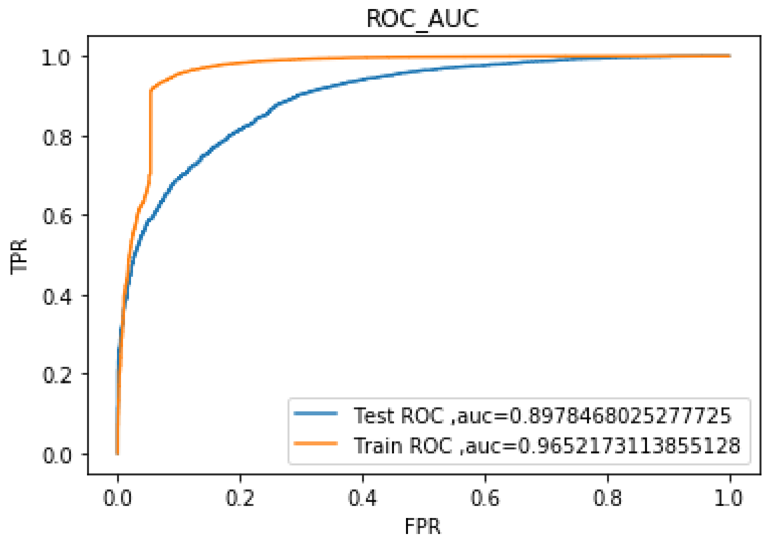
```
auc_cv=[]
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm_notebook(c): #simple cv using for loop
    svc=SVC(C=i,probability=True).fit(fbowx_tr,y_train)
    pred_cv = svc.predict_proba(fbowx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svc.predict_proba(fbowx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))
best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used
 to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```



```
Best c value for max auc = 1
```

In [49]:

```
#Plotting ROC_AUC curve
svc=SVC(C=best_c,probability=True).fit(fbowx_tr,y_train)
pred_te=svc.predict_proba(fbowx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svc.predict_proba(fbowx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_tr)))
plt.title('ROC_AUC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



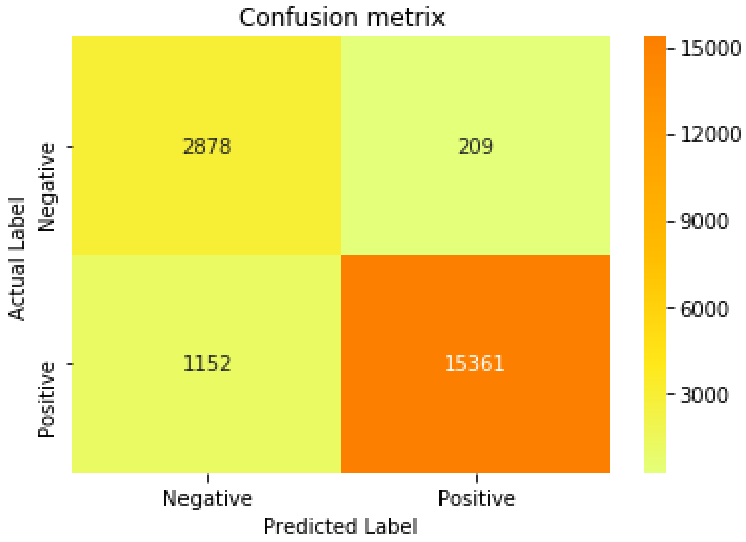
In [52]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Wistia")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.8672567093872756 for threshold 0.874
Train confusion matrix
```

Out[52]:

```
Text(33,0.5,'Actual Label')
```



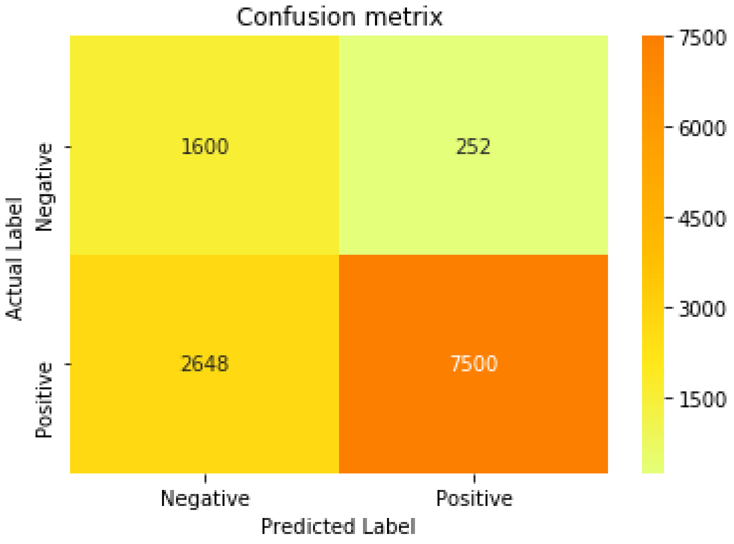
In [54]:

```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, b
est_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Wistia")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.8672567093872756 for threshold 0.874
Train confusion matrix
```

Out[54]:

```
Text(33,0.5,'Actual Label')
```



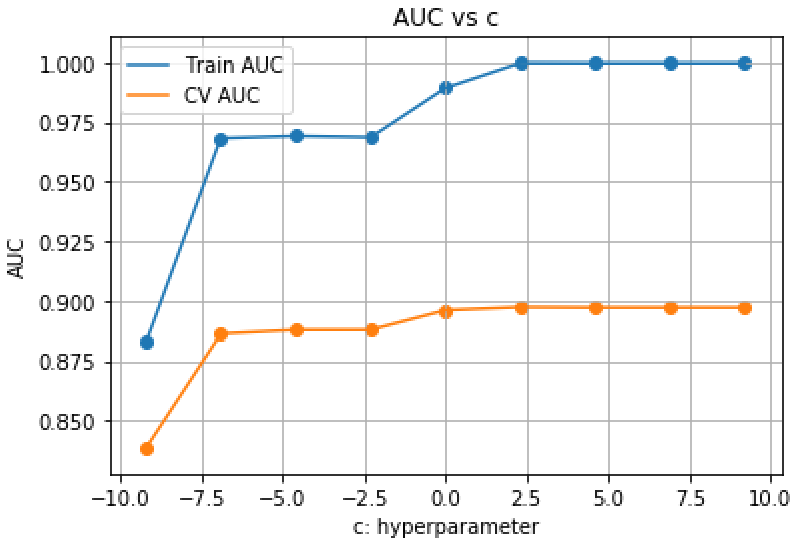
[5.2.2] Applying RBF SVM on TFIDF, SET 2

In [91]:

```
# Please write all the code with proper documentation
tf_vect=TfidfVectorizer(ngram_range=(1,2),min_df=10,max_features = 500)
tf_vect.fit(preprocessed_reviews)

ftfx_tr=tf_vect.fit_transform(x_train)
ftfx_cv=tf_vect.transform(x_cv)
ftfx_te=tf_vect.transform(x_test)

ftfx_tr=std.fit_transform(ftfx_tr)#Standardizing Data
ftfx_cv=std.transform(ftfx_cv)
ftfx_te=std.transform(ftfx_te)
```

In [92]:

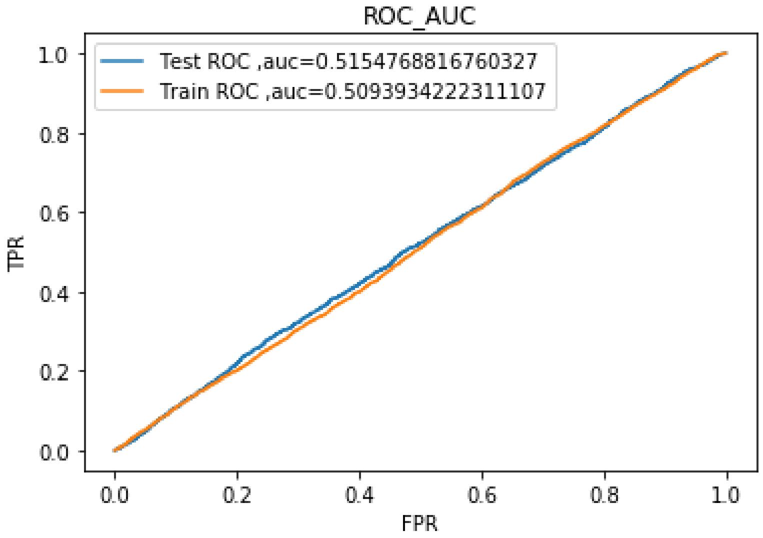
```
from sklearn.metrics import roc_auc_score
auc_cv=[]
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm_notebook(c): #simple cv using for loop
    svc=SVC(C=i,probability=True).fit(ftfx_tr,y_train)
    pred_cv = svc.predict_proba(ftfx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svc.predict_proba(ftfx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))
best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```



```
Best c value for max auc = 10
```

In [93]:

```
#Plotting ROC_AUC curve
svc=SVC(C=best_c,probability=True).fit(fbowx_tr,y_train)
pred_te=svc.predict_proba(ftfx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svc.predict_proba(ftfx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_tes
t,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_tr
ain,pred_tr)))
plt.title('ROC_AUC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



In [94]:

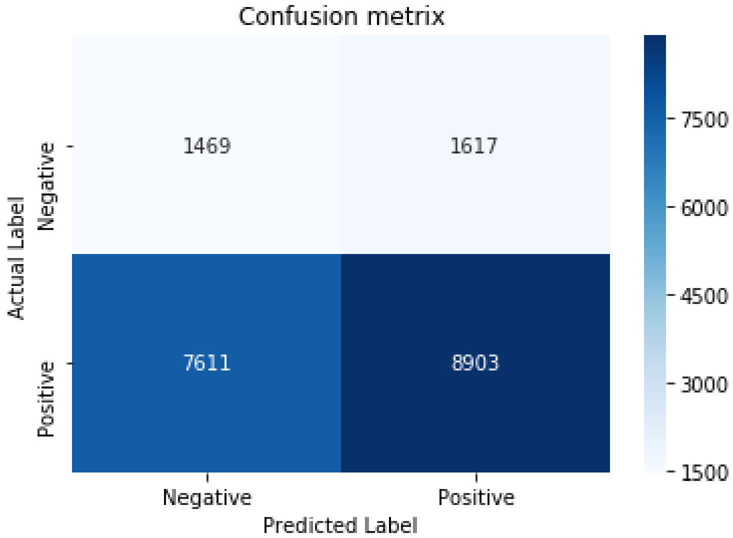
```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
```

```
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.25663150282903774 for threshold 0.85
6
Train confusion matrix
```

Out[94]:

```
Text(33,0.5,'Actual Label')
```

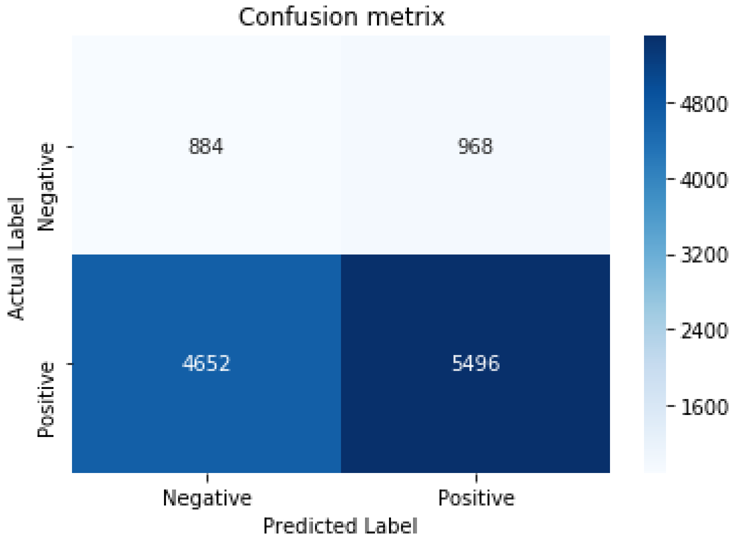


In [95]:

```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, b
est_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.25663150282903774 for threshold 0.85
6
Train confusion matrix
```

```
Out[95]: Text(33,0.5,'Actual Label')
```



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

```
In [60]: # Please write all the code with proper documentation

#Avg word2vec for train data
sent_train_list=[]
for sentence in x_train:
    sent_train_list.append(sentence.split())
w2v_model=Word2Vec(sent_train_list,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

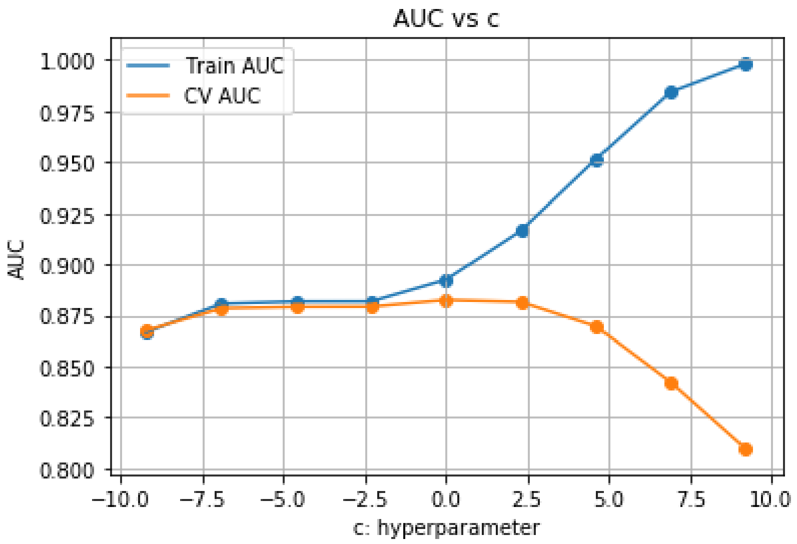
```
sent_train_vectors = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sent in tqdm_notebook(sent_train_list): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =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_train_vectors.append(sent_vec)
print(len(sent_train_vectors))
print(len(sent_train_vectors[0]))

#Avg word2vec for cv data
sent_cv_list=[]
for sentence in x_cv:
    sent_cv_list.append(sentence.split())

sent_cv_vectors = []; # the avg-w2v for each sentence/review is stored
 in this list
for sent in tqdm_notebook(sent_cv_list): # 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_cv_vectors.append(sent_vec)
```

```
print(len(sent_cv_vectors))
print(len(sent_cv_vectors[0]))


#Avg word2vec for test data
sent_test_list=[]
for sentence in x_test:
    sent_test_list.append(sentence.split())

sent_test_vectors = []; # the avg-w2v for each sentence/review is store
d in this list
for sent in tqdm_notebook(sent_test_list): # 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_test_vectors.append(sent_vec)
print(len(sent_test_vectors))
print(len(sent_test_vectors[0]))


#This code is copied and modified from :https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW#scrollTo=3-XGItt4PSx0
```

```
19600
50


8400
50


12000
50
```

In [61]:

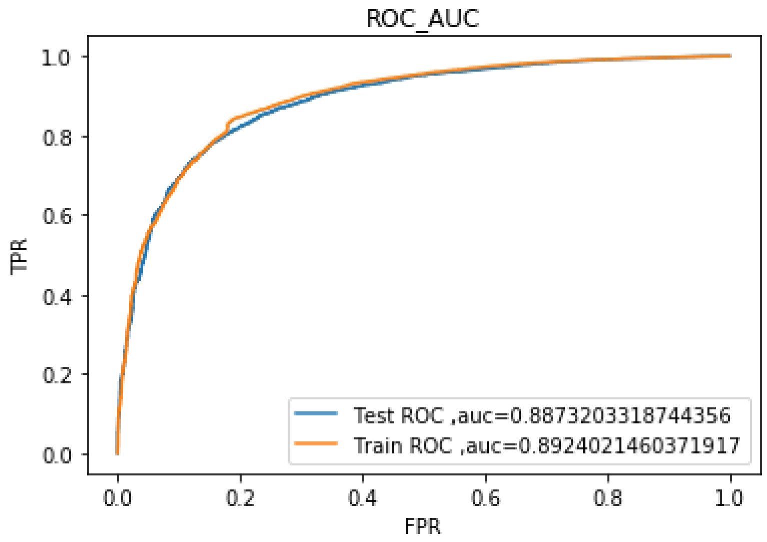
```
aw2vx_tr=sent_train_vectors
aw2vx_cv=sent_cv_vectors
aw2vx_te=sent_test_vectors

auc_cv=[]
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm_notebook(c):
    svc=SVC(C=i,probability=True).fit(aw2vx_tr,y_train)
    pred_cv= svc.predict_proba(aw2vx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svc.predict_proba(aw2vx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))
best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```



```
Best c value for max auc = 1
```

In [62]:

```
#Plotting ROC_AUC curve
svc=SVC(C=best_c,probability=True).fit(aw2vx_tr,y_train)
pred_te=svc.predict_proba(aw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svc.predict_proba(aw2vx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_tr)))
plt.title('ROC_AUC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



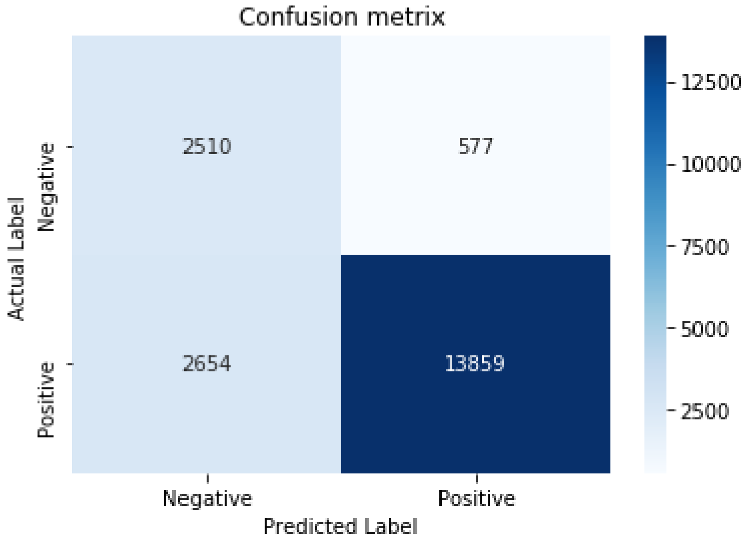
In [63]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.6824062658488721 for threshold 0.842
Train confusion matrix
```

Out[63]:

```
Text(33,0.5,'Actual Label')
```



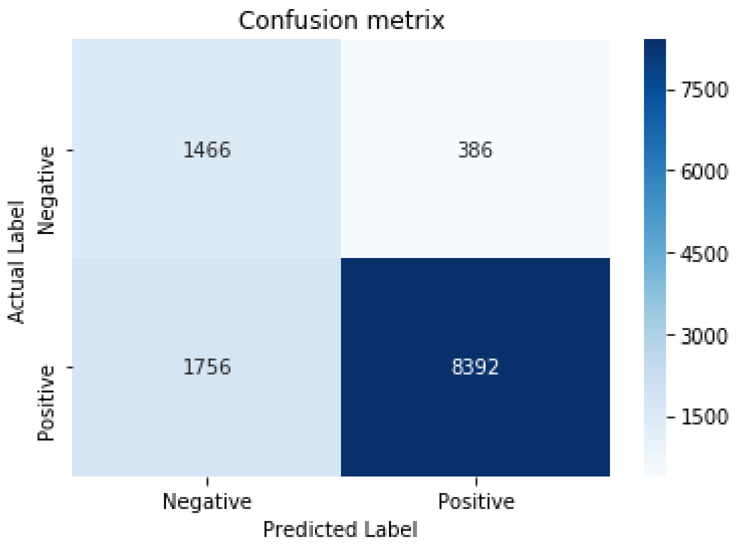
In [65]:

```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, b
est_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.6824062658488721 for threshold 0.8
42
Train confusion matrix
```

Out[65]:

```
Text(33,0.5,'Actual Label')
```



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

In [66]:

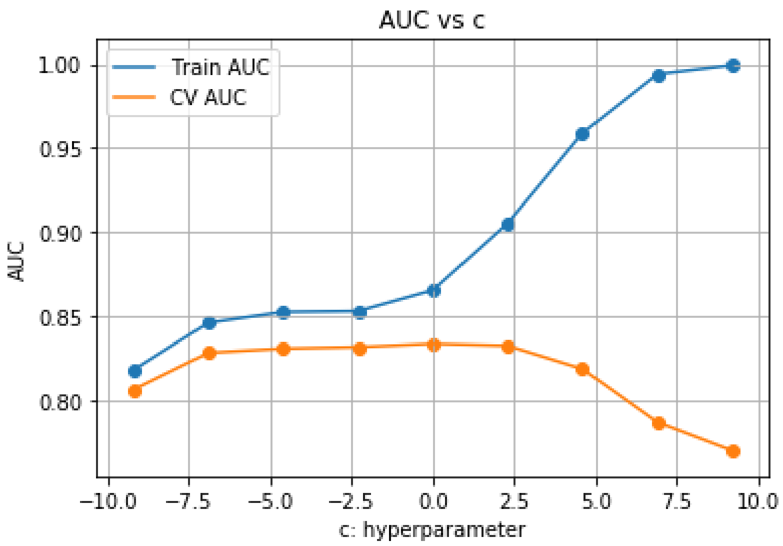
```
# Please write all the code with proper documentation
sent_train_list=[]
for sentence in x_train:
    sent_train_list.append(sentence.split())
w2v_model=Word2Vec(sent_train_list,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10, max_features
=500)
tf_idf_matrix=tf_idf_vect.fit_transform(x_train)
tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect
.idf_)))

#Train data
tfidf_sent_train_vectors = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
```

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

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

#Test data
sent_test_list=[]
for sentence in x_test:
    sent_test_list.append(sentence.split())
tfidf_sent_test_vectors = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm_notebook(sent_test_list): # 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_test_vectors.append(sent_vec)
    row += 1
```

In [67]:

```
# Please write all the code with proper documentation
tfw2vx_tr=tfidf_sent_train_vectors
tfw2vx_cv=tfidf_sent_cv_vectors
```

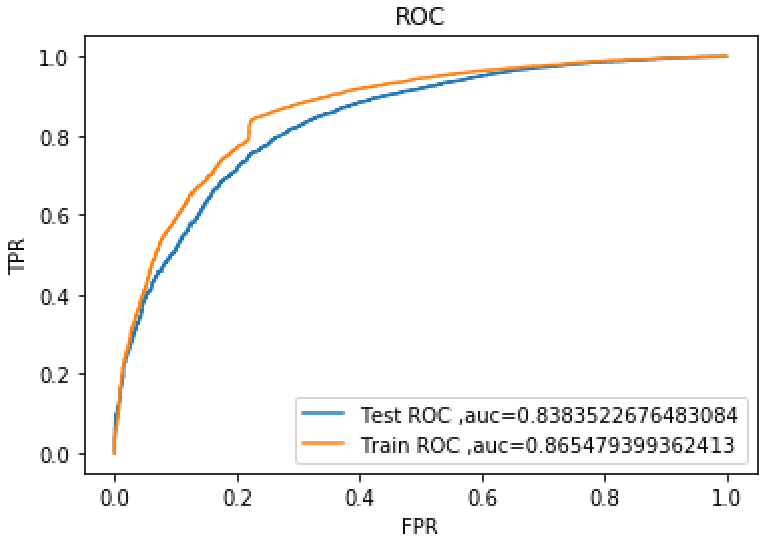
```
tfw2vx_te=tfidf_sent_test_vectors
auc_cv=[]
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm_notebook(c):
    svc=SVC(C=i,probability=True).fit(tfw2vx_tr,y_train)
    pred_cv = svc.predict_proba(tfw2vx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svc.predict_proba(tfw2vx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))

best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```



```
Best c value for max auc = 1
```

In [69]:

```python
#Plotting ROC Curve
svc=SVC(C=best_c,probability=True).fit(tfw2vx_tr,y_train)
pred_te=svc.predict_proba(tfw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svc.predict_proba(tfw2vx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_tr)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



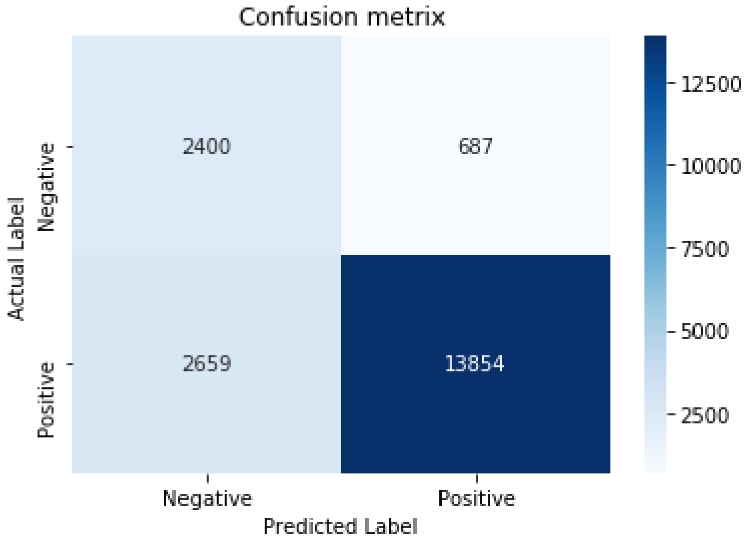
In [70]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.6522646085538402 for threshold 0.851
Train confusion matrix
```

Out[70]:

```
Text(33,0.5,'Actual Label')
```



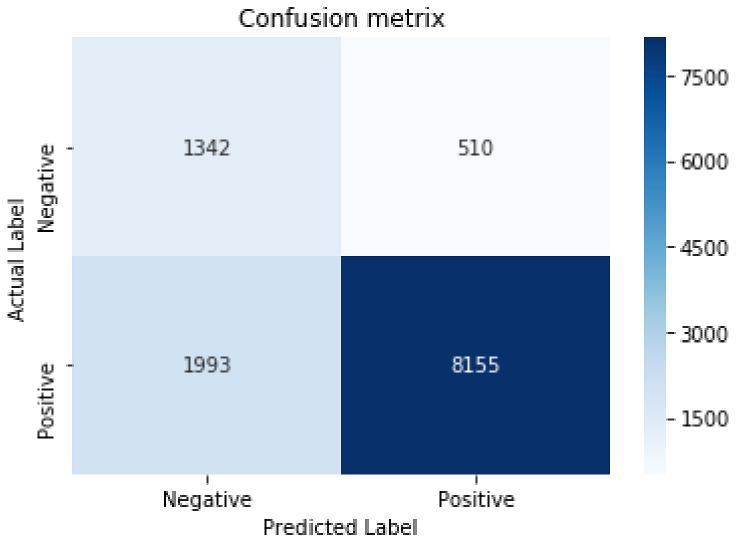
In [71]:

```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, b
est_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.6522646085538402 for threshold 0.8
51
Train confusion matrix
```

Out[71]:

```
Text(33,0.5,'Actual Label')
```



[5.1] Linear SVM with Feature Engineering

[5.1.1] Applying Linear SVM on BOW, SET 1

In [96]:
```
for i in range(len(preprocessed_reviews)):  #considering  some features
 from reviw summary and length of review text
    preprocessed_reviews[i]=preprocessed_reviews[i]+ ' '+preprocessed_s
ummary[i]+' '+str(len(final.Text.iloc[i]))
preprocessed_fe_reviews=preprocessed_reviews
```

In [97]:
```
preprocessed_fe_reviews[1500]
```

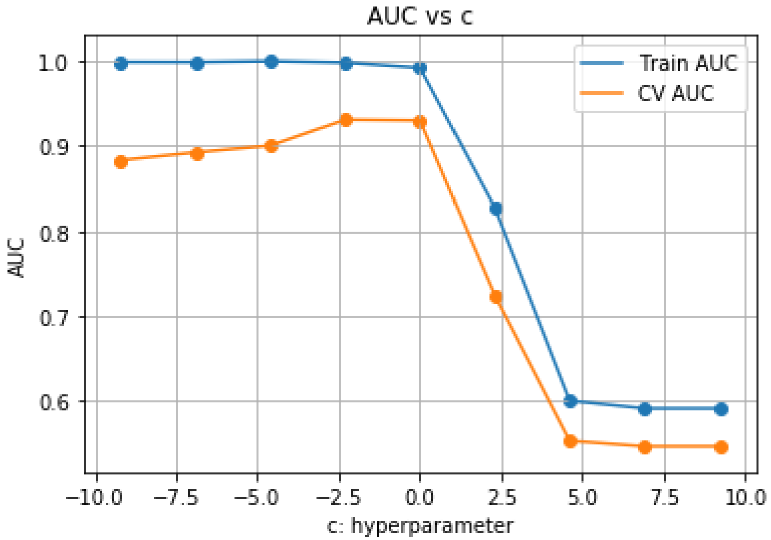
Out[97]:
```
'way hot blood took bite jig lol hot stuff 59'
```

In [98]:
```
# Please write all the code with proper documentation
import numpy as np
import pandas as pd
import math
from sklearn.model_selection  import train_test_split
from sklearn.metrics import accuracy_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
bow_vect=CountVectorizer()
x=preprocessed_fe_reviews
y=np.array(final['Score'])
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random
_state=0)
x_train,x_cv,y_train,y_cv=train_test_split(x_train,y_train,test_size=0.
3)

fbowx_tr=bow_vect.fit_transform(x_train)
fbowx_cv=bow_vect.transform(x_cv)
fbowx_te=bow_vect.transform(x_test)

std=StandardScaler(with_mean=False) #Standardizing Data
fbowx_tr=std.fit_transform(fbowx_tr)
fbowx_cv=std.transform(fbowx_cv)
fbowx_te=std.transform(fbowx_te)

svm_=SGDClassifier(alpha=0.0001)
svm=CalibratedClassifierCV(svm_,cv=3).fit(fbowx_tr,y_train)
```

In [84]:

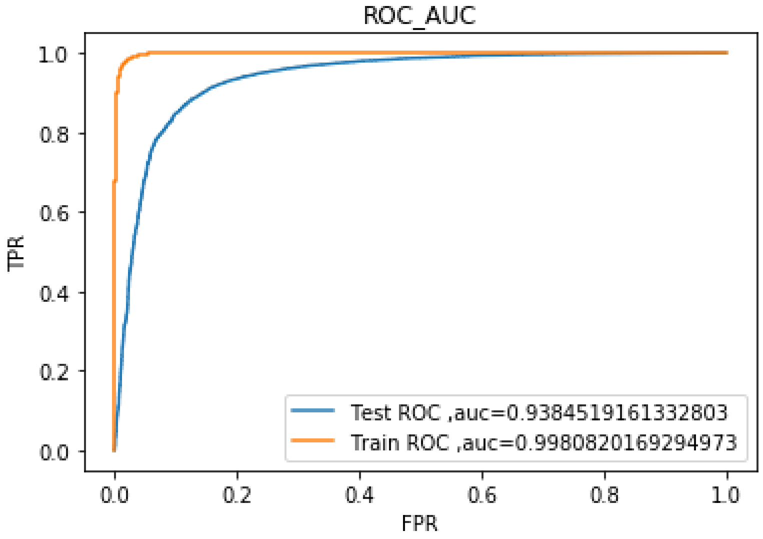
```
from sklearn.metrics import roc_auc_score
auc_cv=[]
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm(c): #simple cv using for loop
    svm_=SGDClassifier(alpha=i)
    svm=CalibratedClassifierCV(svm_,cv=3).fit(fbowx_tr,y_train)
    pred_cv = svm.predict_proba(fbowx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svm.predict_proba(fbowx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))
best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```

```
100%|██████████████████████████████████████████████| 9/9 [00:12<00:00,  1.41s/it]
```



```
Best c value for max auc = 0.1
```

In [85]:
```
#Plotting ROC_AUC curve
svm_=SGDClassifier(alpha=best_c)
svm=CalibratedClassifierCV(svm_,cv=3).fit(fbowx_tr,y_train)
pred_te=svm.predict_proba(fbowx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svm.predict_proba(fbowx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_tr)))
plt.title('ROC_AUC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



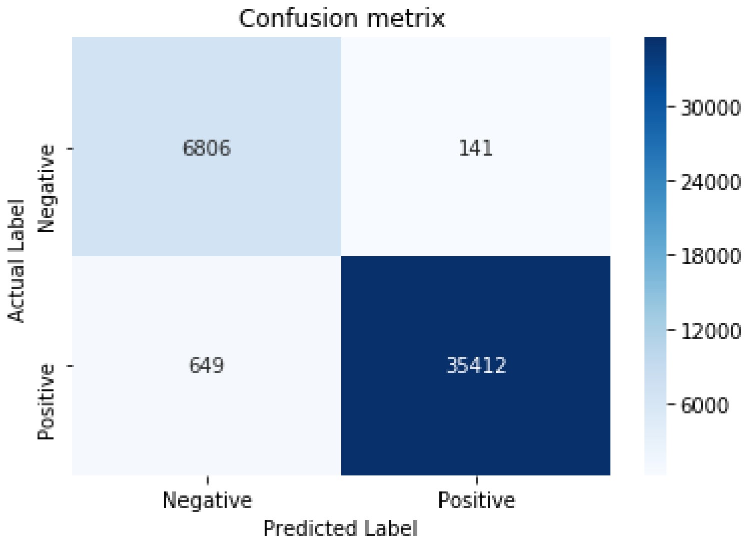
In [88]:
```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
```

```
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9620714691383078 for threshold 0.746
Train confusion matrix
```

Out[88]:

```
Text(33,0.5,'Actual Label')
```



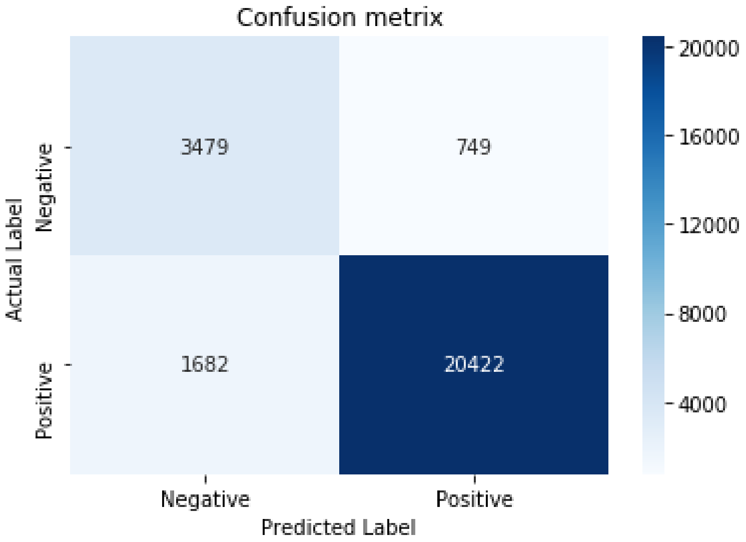
In [89]:

```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9620714691383078 for threshold 0.746
Train confusion matrix
```

Out[89]:

```
Text(33,0.5,'Actual Label')
```



[5.1.2] Applying Linear SVM on TFIDF, SET 2

In [90]:

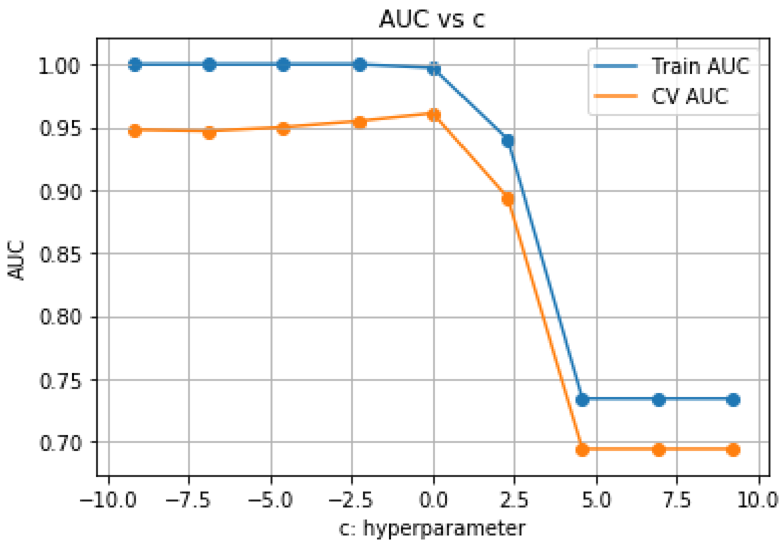
```
# Please write all the code with proper documentation
tf_vect=TfidfVectorizer(ngram_range=(1,2),min_df=10)
tf_vect.fit(preprocessed_reviews)

ftfx_tr=tf_vect.fit_transform(x_train)
ftfx_cv=tf_vect.transform(x_cv)
ftfx_te=tf_vect.transform(x_test)

ftfx_tr=std.fit_transform(ftfx_tr)#Standardizing Data
```

```
ftfx_cv=std.transform(ftfx_cv)
ftfx_te=std.transform(ftfx_te)
```

In [91]:

```
from sklearn.metrics import roc_auc_score
auc_cv=[]
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm(c): #simple cv using for loop
    svm_=SGDClassifier(alpha=i)
    svm=CalibratedClassifierCV(svm_,cv=3).fit(ftfx_tr,y_train)
    pred_cv = svm.predict_proba(ftfx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svm.predict_proba(ftfx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))
best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```

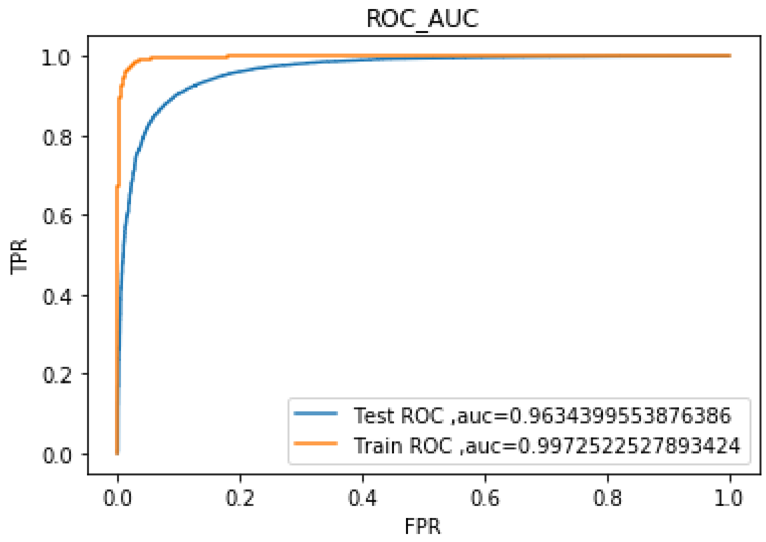
```
100%|████████████████████████████████████████████████████████████
████████████████| 9/9 [00:14<00:00,  1.66s/it]
```



```
Best c value for max auc = 1
```

In [92]:

```
#Plotting ROC_AUC curve
svm_=SGDClassifier(alpha=best_c)
svm=CalibratedClassifierCV(svm_,cv=3).fit(ftfx_tr,y_train)
pred_te=svm.predict_proba(ftfx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svm.predict_proba(ftfx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_tes
t,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_tr
ain,pred_tr)))
plt.title('ROC_AUC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



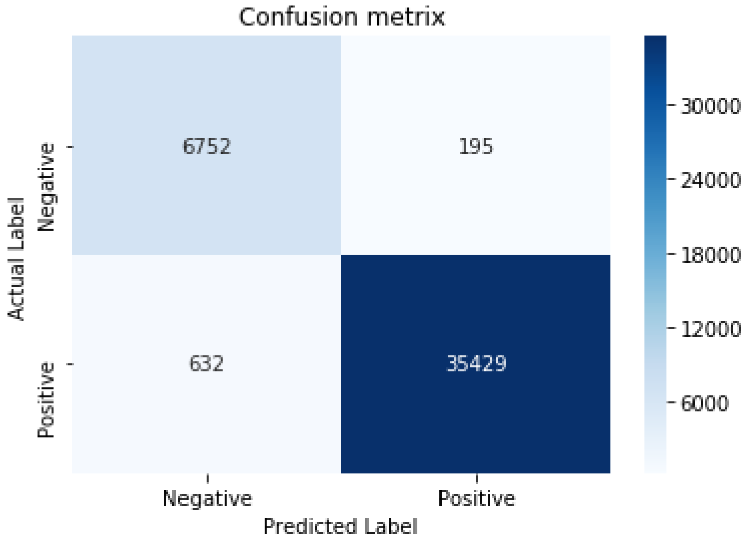
In [93]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9548964157613281 for threshold 0.742
Train confusion matrix
```

Out[93]:

```
Text(33,0.5,'Actual Label')
```



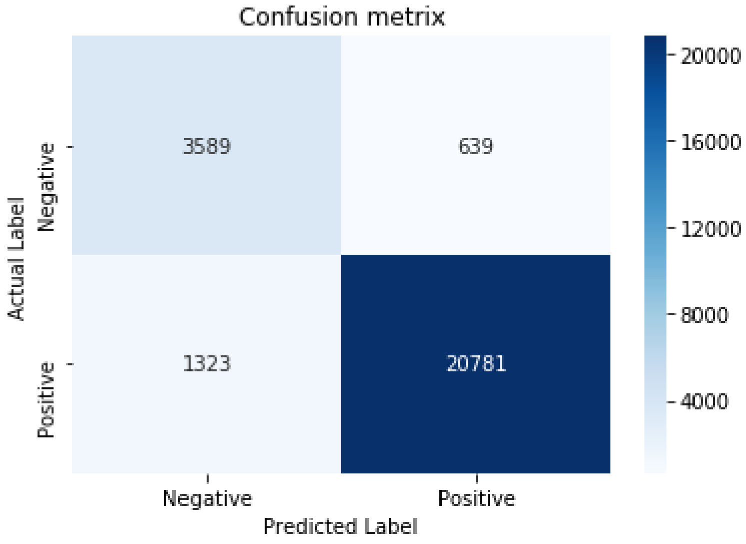
In [94]:

```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, b
est_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9548964157613281 for threshold 0.742
Train confusion matrix
```

Out[94]:

```
Text(33,0.5,'Actual Label')
```



[5.1.3] Applying Linear SVM on AVG W2V, SET 3

In [95]:

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


#Avg word2vec for train data
sent_train_list=[]
for sentence in x_train:
    sent_train_list.append(sentence.split())
w2v_model=Word2Vec(sent_train_list,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)

sent_train_vectors = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sent in tqdm(sent_train_list): # 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_train_vectors.append(sent_vec)
print(len(sent_train_vectors))
print(len(sent_train_vectors[0]))

#Avg word2vec for cv data
sent_cv_list=[]
for sentence in x_cv:
    sent_cv_list.append(sentence.split())

sent_cv_vectors = []; # the avg-w2v for each sentence/review is stored
 in this list
for sent in tqdm(sent_cv_list): # 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_cv_vectors.append(sent_vec)
print(len(sent_cv_vectors))
print(len(sent_cv_vectors[0]))


#Avg word2vec for test data
sent_test_list=[]
for sentence in x_test:
    sent_test_list.append(sentence.split())
```

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


#This code is copied and modified from :https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW#scrollTo=3-XGItt4PSx0
```

```
100%|██████████| 43008/43008 [04:51<00:00, 147.69it/s]
```

```
43008
50
```

```
100%|██████████| 18433/18433 [02:11<00:00, 140.45it/s]
```

```
18433
50
```

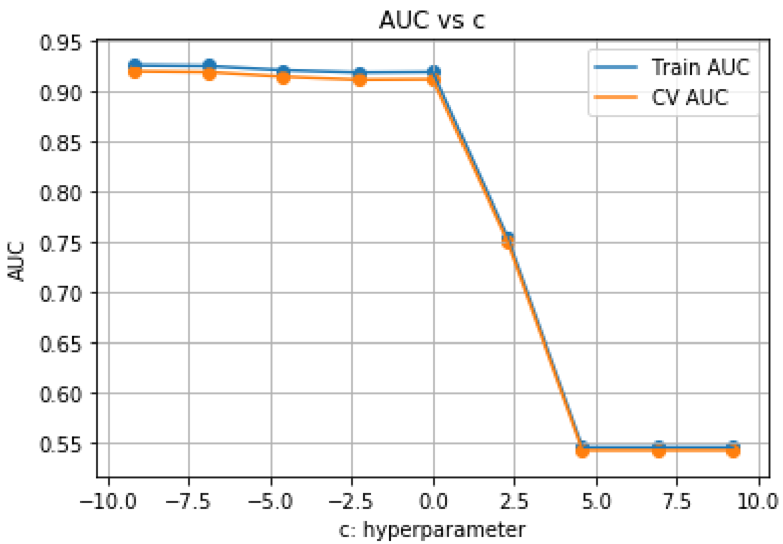
```
100%|██████████| 26332/26332 [03:15<00:00, 134.71it/s]
```

```
26332
50
```

In [96]:

```
aw2vx_tr=sent_train_vectors
```

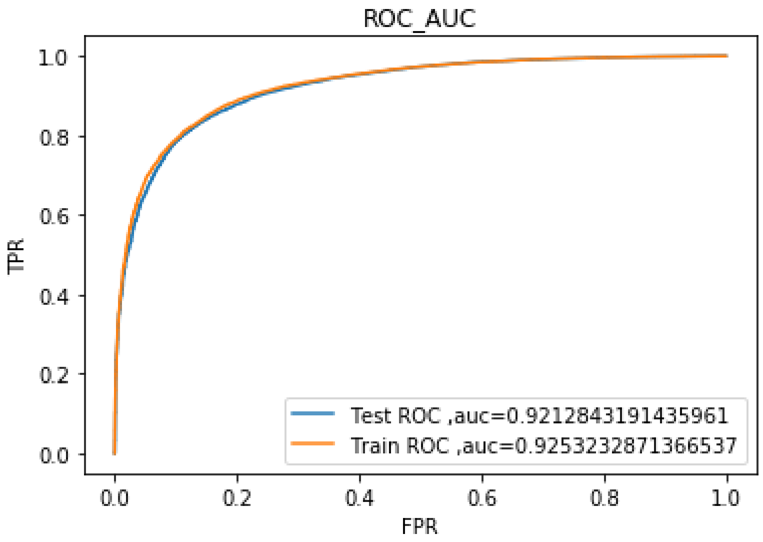
```
aw2vx_cv=sent_cv_vectors
aw2vx_te=sent_test_vectors

auc_cv=[]
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm_notebook(c):
    svm_=SGDClassifier(alpha=i)
    svm=CalibratedClassifierCV(svm_,cv=3).fit(aw2vx_tr,y_train)
    pred_cv= svm.predict_proba(aw2vx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svm.predict_proba(aw2vx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))
best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```



```
Best c value for max auc = 0.0001
```

In [97]:

```
#Plotting ROC_AUC curve
svm_=SGDClassifier(alpha=best_c)
svm=CalibratedClassifierCV(svm_,cv=3).fit(aw2vx_tr,y_train)
pred_te=svm.predict_proba(aw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svm.predict_proba(aw2vx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_tes
t,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_tr
ain,pred_tr)))
plt.title('ROC_AUC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



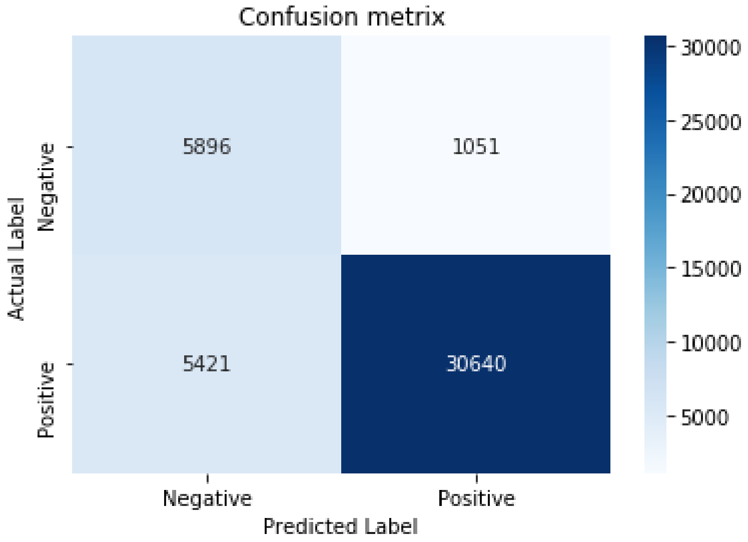
In [98]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.7211260279677327 for threshold 0.812
Train confusion matrix
```

Out[98]:

```
Text(33,0.5,'Actual Label')
```



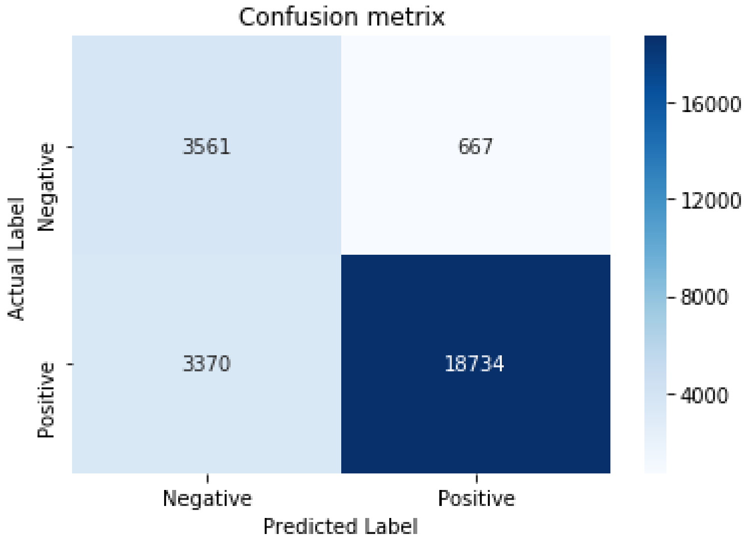
In [99]:

```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.7211260279677327 for threshold 0.812
Train confusion matrix
```

Out[99]:

```
Text(33,0.5,'Actual Label')
```



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

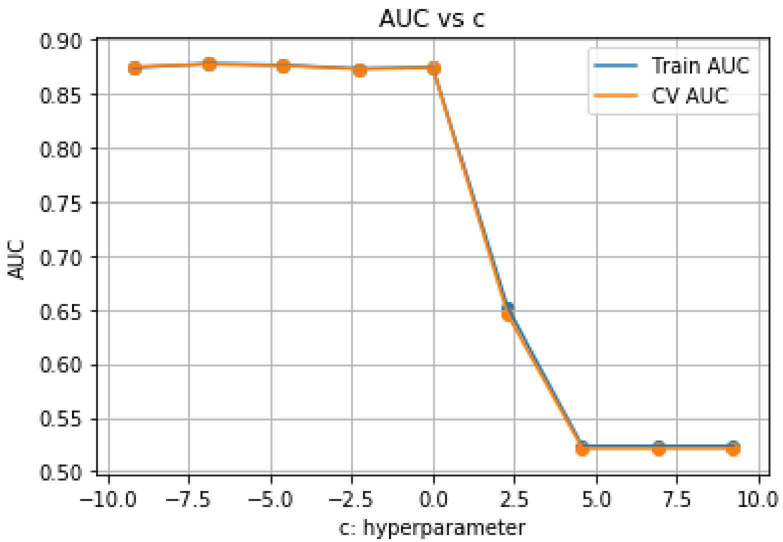
```
In [79]: # Please write all the code with proper documentation
sent_train_list=[]
for sentence in x_train:
    sent_train_list.append(sentence.split())
w2v_model=Word2Vec(sent_train_list,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10, max_features
=500)
tf_idf_matrix=tf_idf_vect.fit_transform(x_train)
tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect
.idf_)))

#Train data
tfidf_sent_train_vectors = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm_notebook(sent_train_list): # for each review/sentence
```

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

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

#Test data
sent_test_list=[]
for sentence in x_test:
    sent_test_list.append(sentence.split())
tfidf_sent_test_vectors = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm_notebook(sent_test_list): # 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_test_vectors.append(sent_vec)
    row += 1
```

In [48]:

```
tfw2vx_tr=tfidf_sent_train_vectors
tfw2vx_cv=tfidf_sent_cv_vectors
tfw2vx_te=tfidf_sent_test_vectors
auc_cv=[]
```

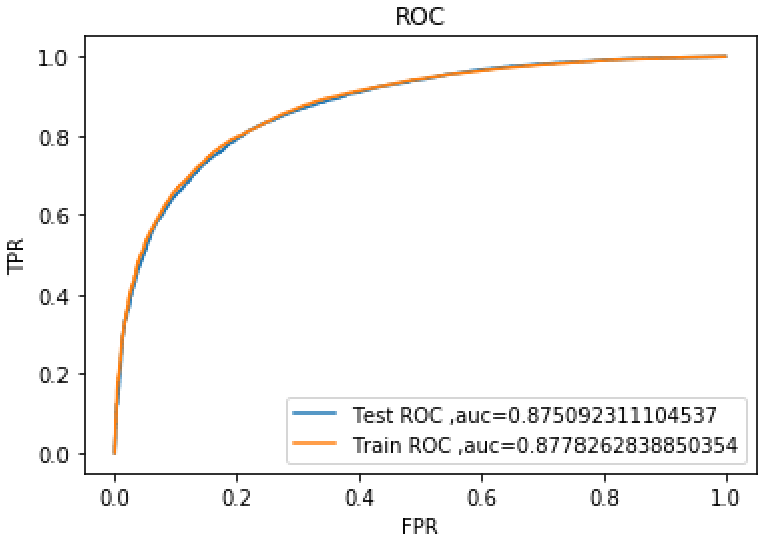
```
auc_train=[]
c=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in tqdm_notebook(c):
    svm_=SGDClassifier(alpha=i)
    svm=CalibratedClassifierCV(svm_,cv=3).fit(tfw2vx_tr,y_train)
    pred_cv = svm.predict_proba(tfw2vx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_cv))
    pred_tr=svm.predict_proba(tfw2vx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_tr))

best_c= c[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used
 to find best alpha
c=[math.log(j) for j in c]
plt.plot(c, auc_train, label='Train AUC')
plt.plot(c, auc_cv, label='CV AUC')

plt.scatter(c, auc_train)
plt.scatter(c, auc_cv)
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs c")
plt.grid()
plt.show()
print("Best c value for max auc =",best_c)
```



```
Best c value for max auc = 0.001
```

In [49]:

```
#Plotting ROC Curve
svm_=SGDClassifier(alpha=best_c)
svm=CalibratedClassifierCV(svm_,cv=3).fit(tfw2vx_tr,y_train)
pred_te=svm.predict_proba(tfw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=svm.predict_proba(tfw2vx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_tr)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_tes
t,pred_te)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_tr
ain,pred_tr)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



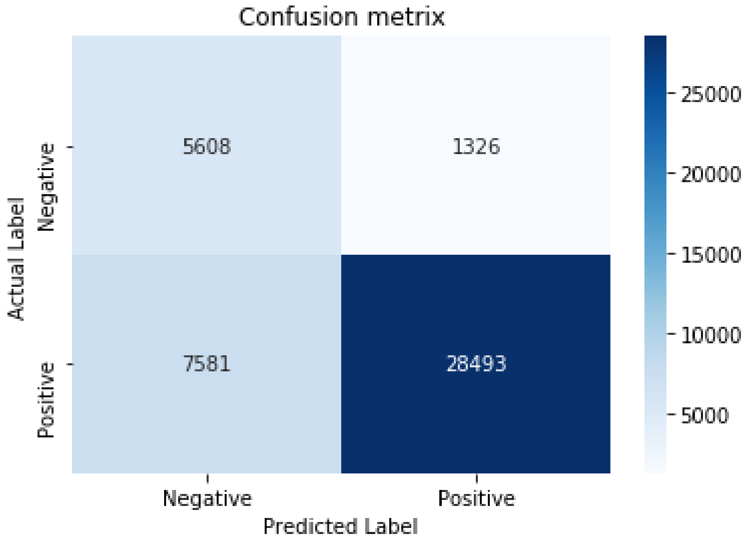
In [55]:

```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_tr,
best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.c
om/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.6388046146658218 for threshold 0.832
Train confusion matrix
```

Out[55]:

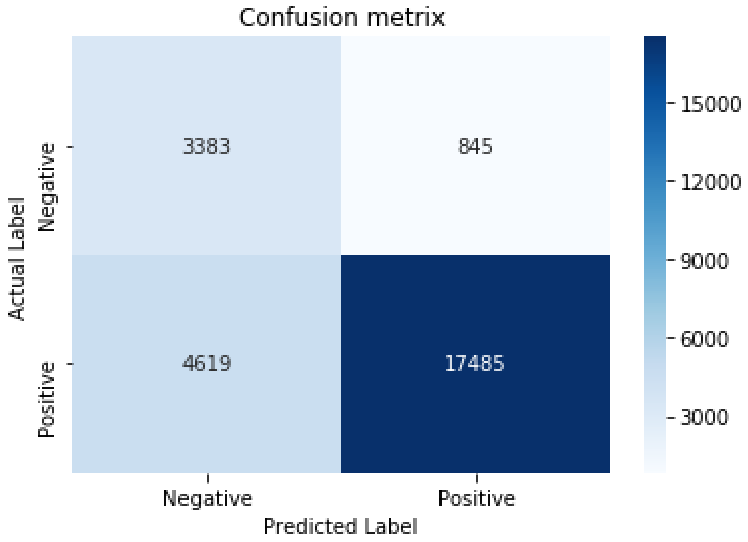
```
Text(33,0.5,'Actual Label')
```



In [54]:
```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.6388046146658218 for threshold 0.832
Train confusion matrix
```

Out[54]:
```
Text(33,0.5,'Actual Label')
```



[6] Conclusions

In [106]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=(['Vectorizer','Kernal','Hyperparameter','AUC','Feature E
ngineering'])
x.add_row(['BOW','Linear',0.1,0.907,'NO'])
x.add_row(['TF-IDF','Linear',10,0.936,'NO'])
x.add_row(['AW2V','Linear',100,0.916,'NO'])
x.add_row(['TF-IDF_w2v ','Linear',0.01,0.834,'NO'])
x.add_row(['BOW','RBF',1,0.897,'NO'])
x.add_row(['TF-IDF','RBF',10,0.515,'NO'])
x.add_row(['AW2V','RBF',1,0.887,'NO'])
x.add_row(['TF-IDF_w2v','RBF',1,0.838,'NO'])
x.add_row(['BOW','Linear',0.1,0.938,'YES'])
x.add_row(['TF-IDF','Linear',1,0.963,'YES'])
x.add_row(['AW2V','Linear',0.0001,0.921,'YES'])
x.add_row(['TF-IDF_w2v','Linear',0.001,0.875,'YES'])
print(x)
```

```
+-------------+--------+----------------+-------+---------------------+
|  Vectorizer | Kernal | Hyperparameter |  AUC  | Feature Engineering |
+-------------+--------+----------------+-------+---------------------+
|     BOW     | Linear |      0.1       | 0.907 |          NO         |
|    TF-IDF   | Linear |       10       | 0.936 |          NO         |
|     AW2V    | Linear |      100       | 0.916 |          NO         |
| TF-IDF_w2v  | Linear |      0.01      | 0.834 |          NO         |
|     BOW     |  RBF   |       1        | 0.897 |          NO         |
|    TF-IDF   |  RBF   |       10       | 0.515 |          NO         |
|     AW2V    |  RBF   |       1        | 0.887 |          NO         |
|  TF-IDF_w2v |  RBF   |       1        | 0.838 |          NO         |
|     BOW     | Linear |      0.1       | 0.938 |         YES         |
|    TF-IDF   | Linear |       1        | 0.963 |         YES         |
|     AW2V    | Linear |     0.0001     | 0.921 |         YES         |
|  TF-IDF_w2v | Linear |     0.001      | 0.875 |         YES         |
+-------------+--------+----------------+-------+---------------------+
```

It is observed that there is a slight increment in the accuracy after engineering