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 prettytable import PrettyTable
from tqdm import tqdm_notebook
import os
```

In [2]:

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

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

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", co
n)
# 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 score<3 a negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

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

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

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	130386240
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	134697600

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	121901760

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

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

In [5]:

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

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
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

[2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776

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='qui
cksort', na_position='last')
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=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	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883

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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because 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 were 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 smell. 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 these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely 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/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)
```

```
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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [16]:

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

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because 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 were 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 smell. 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 these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely worth it to buy a big bag if your dog eats them a lot.
```

In [17]:

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

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

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

In [18]:

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

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

In [19]:

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because 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', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', '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', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

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

```
100%|████████████████████████████████████████████████████████████████████████████| 87773/87773 [01:
35<00:00, 922.62it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

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

[3.2] Preprocessing Review Summary

In [25]:

```
## Similartly you can do preprocessing for review summary also.
import warnings
warnings.filterwarnings("ignore")

from tqdm import tqdm
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())
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

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

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

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

[4.2] Bi-Grams and n-Grams.

In [26]:

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

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

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

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

[4.3] TF-IDF

In [27]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[
1])
```

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

[4.4] Word2Vec

In [28]:

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

In [29]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.
# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
ue)
        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', 0.8369992971420288), ('awesome', 0.8270425796508789), ('good', 0.816173255443573),
('excellent', 0.8119032382965088), ('terrific', 0.7944000363349915), ('wonderful',
0.7765241861343384), ('perfect', 0.7585535645484924), ('amazing', 0.7363982796669006), ('nice', 0.
7208086252212524), ('fabulous', 0.6787959933280945)]
==================================================
[('greatest', 0.795633852481842), ('nastiest', 0.7171007394790649), ('tastiest',
0.7034794092178345), ('best', 0.7018686532974243), ('disgusting', 0.6479982137680054),
('terrible', 0.6330018043518066), ('surpass', 0.6329322457313538), ('horrible',
0.6324087977409363), ('coolest', 0.6309990882873535), ('awful', 0.607223391532898)]
```

In [30]:

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

```
number of words that occured minimum 5 times  17386
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buying', 'anymore',
'hard', 'find', 'products', 'made', 'usa', 'one', 'isnt', 'bad', 'good', 'take', 'chances',
'till', 'know', 'going', 'imports', 'love', 'saw', 'pet', 'store', 'tag', 'attached', 'regarding',
'satisfied', 'safe', 'infestation', 'literally', 'everywhere', 'flying', 'around', 'kitchen',
'bought', 'hoping', 'least', 'get', 'rid', 'weeks', 'fly', 'stuck', 'squishing', 'buggers', 'succe
ss', 'rate']
```

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

[4.4.1.1] Avg W2v

In [31]:

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

```
100%|██████████████████████████████████████████████████████████████████████████| 87773/87773 [11:
36<00:00, 126.06it/s]
```

```
87773
50
```

[4.4.1.2] TFIDF weighted W2v

In [32]:

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

In [33]:

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

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

```
100%|█████████████████████████████████████████████████████████████████████████| 87773/87773
[2:49:25<00:00,  8.63it/s]
```

[5] Assignment 9: Random Forests

1. **Apply Random Forests & GBDT 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. **The hyper paramter tuning (Consider two hyperparameters: n_estimators & max_depth)**

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

3. **Feature importance**

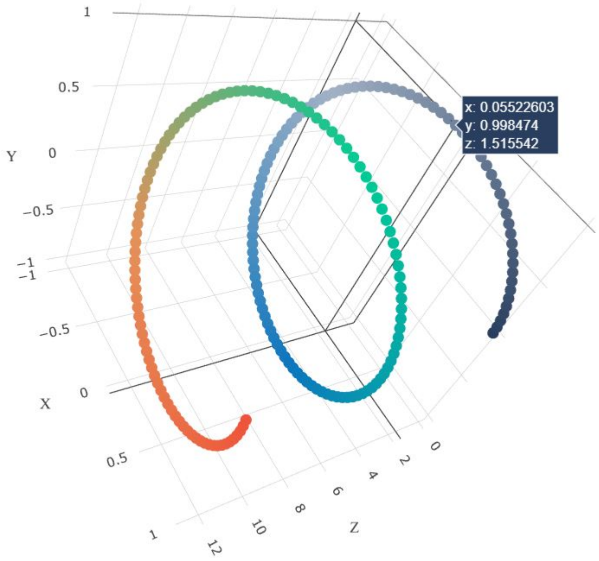
    - Get top 20 important features and represent them in a word cloud. Do this for BOW & TFIDF.

4. **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.

5. **Representation of results**

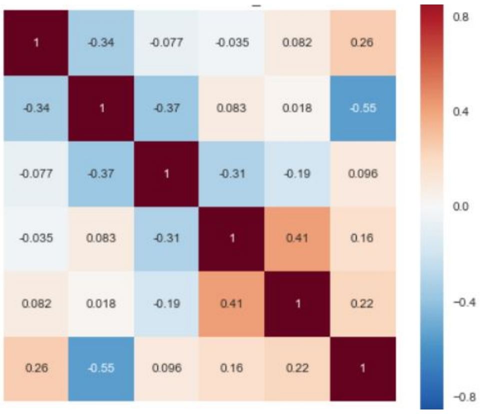
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

(or)

- 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



seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- 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.

6. **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

Note: Data Leakage

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

[5.1] Applying RF

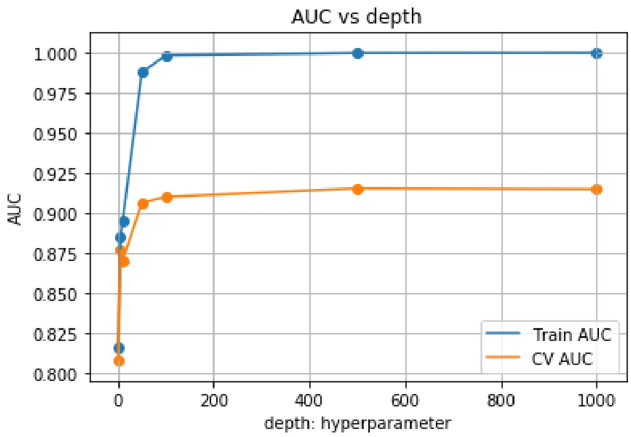
[5.1.1] Applying Random Forests on BOW, SET 1

In [34]:

```
# 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 import tree
from sklearn.ensemble import RandomForestClassifier

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)

rf=RandomForestClassifier(n_estimators=100,max_depth=None).fit(fbowx_tr,y_train)
```

In [35]:

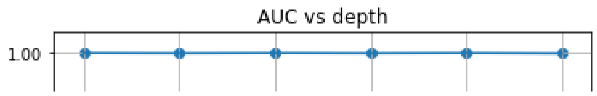
```
n_esti = [20,40,60,80,100,120]
depths=[1,5,10,50,100,500,1000]
best_n=[]
auc_train=[]
auc_cv=[]
for d in tqdm_notebook(depths):
    ns,rc=0,0
    #print(d)
    for n in n_esti:
        #print(m)
        rf=RandomForestClassifier(n_estimators=n,max_depth=d).fit(fbowx_tr,y_train)
        prob_c=rf.predict_proba(fbowx_cv)[:,1]
        val=roc_auc_score(y_cv,prob_c)
        if val>rc:
            rc=val
            ns=n
    rf=RandomForestClassifier(n_estimators=ns,max_depth=d).fit(fbowx_tr,y_train)
    probcv=rf.predict_proba(fbowx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,probcv))
    best_n.append(ns)
    probtr=rf.predict_proba(fbowx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,probtr))

best_depth= depths[auc_cv.index(max(auc_cv))]
best_estimator=best_n[auc_cv.index(max(auc_cv))]

plt.plot(depths, auc_train, label='Train AUC')
plt.plot(depths, auc_cv, label='CV AUC')

plt.scatter(depths, auc_train)
plt.scatter(depths, auc_cv)
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs depth")
plt.grid()
plt.show()
print("Best Depth value for max auc =",best_depth)
print("Best n_estimator value for max auc =",best_estimator)
```

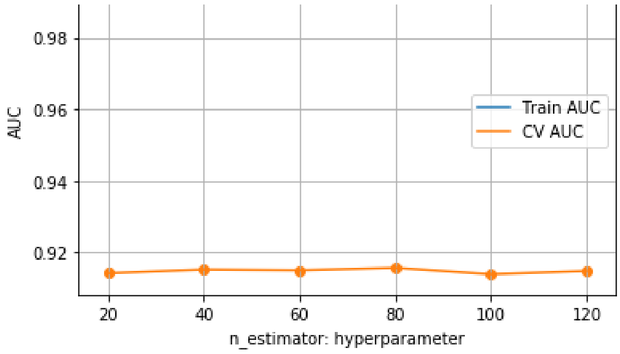


```
Best Depth value for max auc = 500
Best n_estimator value for max auc = 120
```

In [36]:

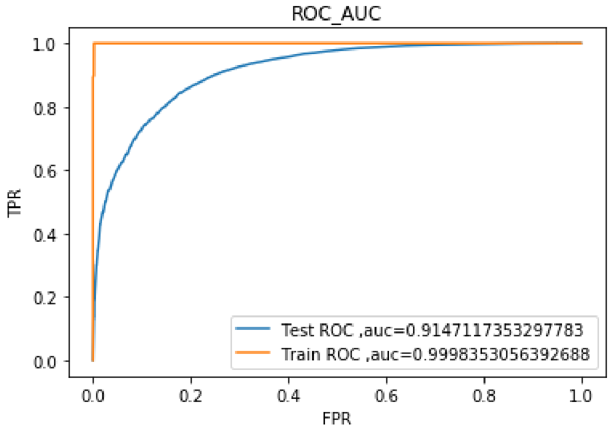
```
auc_train_n=[]
auc_cv_n=[]
for n in tqdm_notebook(n_esti):
    ms,rc=0,0
    #print(d)
    for d in (depths):
        #print(m)
        rf=RandomForestClassifier(n_estimators=n,max_depth=d).fit(fbowx_tr,y_train)
        prob_c=rf.predict_proba(fbowx_cv)[:,1]
        val=roc_auc_score(y_cv,prob_c)
        if val>rc:
            rc=val
            dep=d
    rf=RandomForestClassifier(n_estimators=ns,max_depth=dep).fit(fbowx_tr,y_train)
    probcv=rf.predict_proba(fbowx_cv)[:,1]
    auc_cv_n.append(roc_auc_score(y_cv,probcv))
    best_n.append(ns)
    probtr=rf.predict_proba(fbowx_tr)[:,1]
    auc_train_n.append(roc_auc_score(y_train,probtr))


plt.plot(n_esti, auc_train_n, label='Train AUC')
plt.plot(n_esti, auc_cv_n, label='CV AUC')

plt.scatter(n_esti, auc_train_n)
plt.scatter(n_esti, auc_cv_n)
plt.legend()
plt.xlabel("n_estimator: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs depth")
plt.grid()
plt.show()
```





In [37]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_estimator,max_depth=best_depth).fit(fbowx_tr,y_train)
pred_te=rf.predict_proba(fbowx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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_u
5LEf3yb0YJBVbZW
```



In [38]:

```
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.com/drive/1EkYHI-vGKnURqLL_u
5LEf3yb0YJBVbZW
```

In [39]:

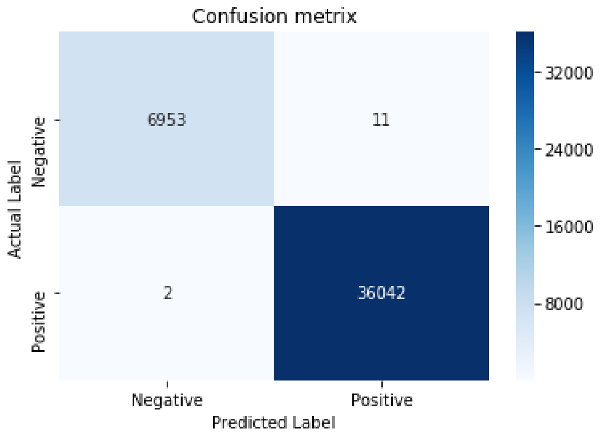
```
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
```

```
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) 0.9983650479269355 for threshold 0.725
Train confusion matrix
```

Out[39]:

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



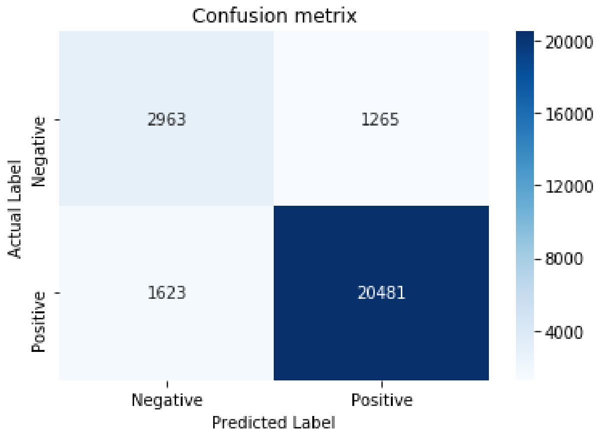
In [40]:

```
#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) 0.9983650479269355 for threshold 0.725
Train confusion matrix
```

Out[40]:

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



[5.1.2] Wordcloud of top 20 important features from SET 1

In [41]:

```
# Please write all the code with proper documentation
from wordcloud import WordCloud
all_features = bow_vect.get_feature_names()
tem=[]
rf=RandomForestClassifier(n_estimators=100,max_depth=None).fit(fbowx_tr,y_train)

features=np.argsort(rf.feature_importances_)[::-1]
for i in features[0:20]:
    tem.append(all_features[i])
print(tem)
words=str(tem) #converting list to string because wordcloud accepts data in string format

wordcloud = WordCloud( background_color ='black').generate(words)

# plot the WordCloud image
plt.figure(figsize = (5, 5), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
#https://www.geeksforgeeks.org/generating-word-cloud-python/
```

```
['not', 'great', 'worst', 'disappointed', 'terrible', 'bad', 'awful', 'horrible', 'money', 'waste'
, 'love', 'return', 'would', 'best', 'good', 'product', 'threw', 'stale', 'taste',
'disappointing']
```

horrible'
great'
best'
waste'
stale'
disappointed'
awful'
threw'
not'
disappointing'
taste'
return'
bad'
product'
love'
would'
worst'
good'
terrible'
money'

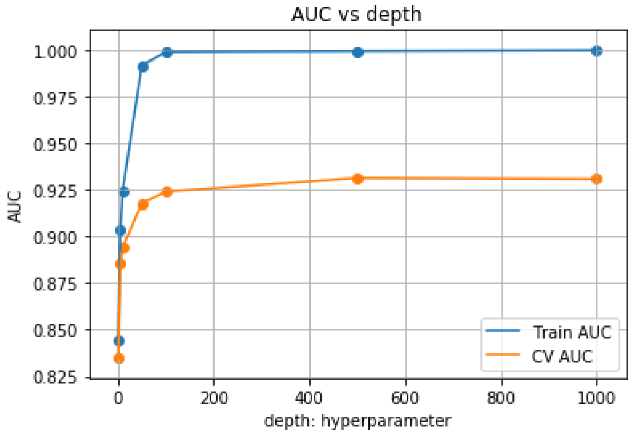
[5.1.3] Applying Random Forests on TFIDF, SET 2

In [61]:

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

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)

rf=RandomForestClassifier(n_estimators=100,max_depth=None).fit(ftfx_tr,y_train)
```

In [62]:

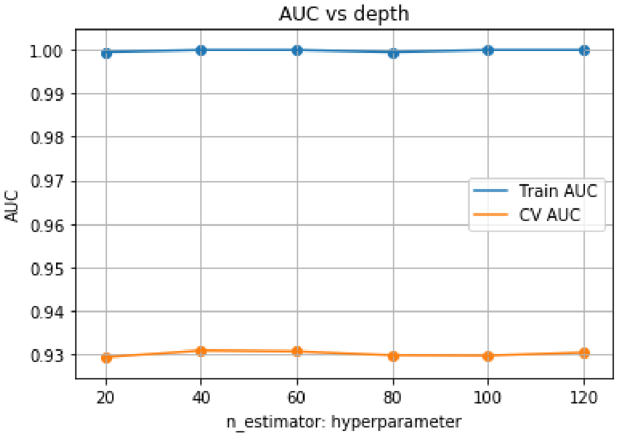
```
n_esti = [20,40,60,80,100,120]
depths=[1,5,10,50,100,500,1000]
best_n=[]
auc_train=[]
auc_cv=[]
for d in tqdm_notebook(depths):
    ns,rc=0,0
    #print(d)
    for n in n_esti:
        #print(m)
        rf=RandomForestClassifier(n_estimators=n,max_depth=d).fit(ftfx_tr,y_train)
        prob_c=rf.predict_proba(ftfx_cv)[:,1]
        val=roc_auc_score(y_cv,prob_c)
        if val>rc:
            rc=val
            ns=n
    rf=RandomForestClassifier(n_estimators=ns,max_depth=d).fit(ftfx_tr,y_train)
    probcv=rf.predict_proba(ftfx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,probcv))
    best_n.append(ns)
    probtr=rf.predict_proba(ftfx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,probtr))

best_depth= depths[auc_cv.index(max(auc_cv))]
best_estimator=best_n[auc_cv.index(max(auc_cv))]

plt.plot(depths, auc_train, label='Train AUC')
plt.plot(depths, auc_cv, label='CV AUC')

plt.scatter(depths, auc_train)
plt.scatter(depths, auc_cv)
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs depth")
plt.grid()
plt.show()
print("Best Depth value for max auc =",best_depth)
print("Best n_estimator value for max auc =",best_estimator)
```



```
Best Depth value for max auc = 500
Best n_estimator value for max auc = 120
```

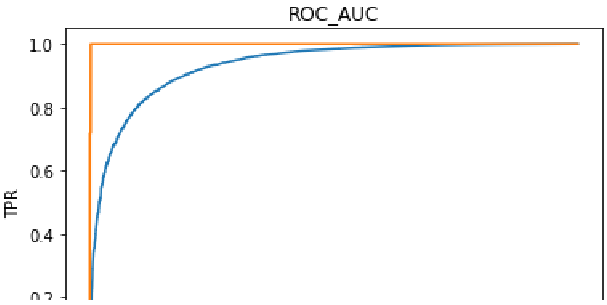
In [63]:

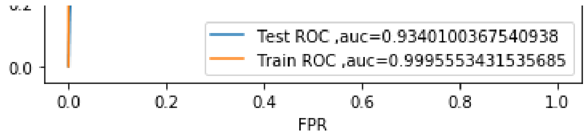
```
auc_train_n=[]
auc_cv_n=[]
for n in tqdm_notebook(n_esti):
    dep,rc=0,0

    for d in (depths):
        #print(m)
        rf=RandomForestClassifier(n_estimators=n,max_depth=d).fit(ftfx_tr,y_train)
        prob_c=rf.predict_proba(ftfx_cv)[:,1]
        val=roc_auc_score(y_cv,prob_c)
        if val>rc:
            rc=val
            dep=d
    rf=RandomForestClassifier(n_estimators=ns,max_depth=dep).fit(ftfx_tr,y_train)
    probcv=rf.predict_proba(ftfx_cv)[:,1]
    auc_cv_n.append(roc_auc_score(y_cv,probcv))
    best_n.append(ns)
    probtr=rf.predict_proba(ftfx_tr)[:,1]
    auc_train_n.append(roc_auc_score(y_train,probtr))


plt.plot(n_esti, auc_train_n, label='Train AUC')
plt.plot(n_esti, auc_cv_n, label='CV AUC')

plt.scatter(n_esti, auc_train_n)
plt.scatter(n_esti, auc_cv_n)
plt.legend()
plt.xlabel("n_estimator: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs depth")
plt.grid()
plt.show()
```



In [64]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_estimator,max_depth=best_depth).fit(ftfx_tr,y_train)
pred_te=rf.predict_proba(ftfx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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_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_u
5LEf3yb0YJBVbZW
```





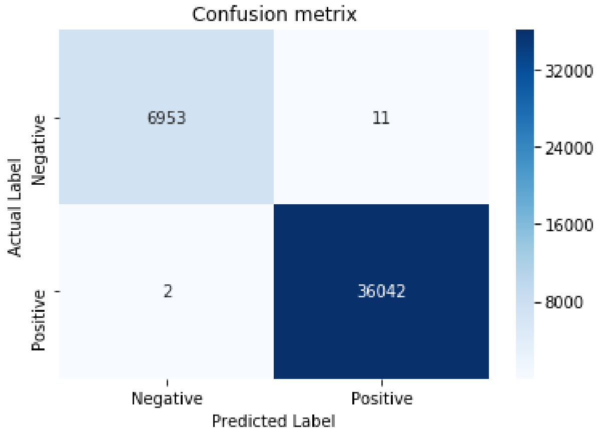
In [65]:

```
#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_u
5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9983650479269355 for threshold 0.667
Train confusion matrix
```

Out[65]:

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



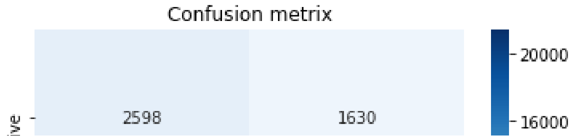
In [66]:

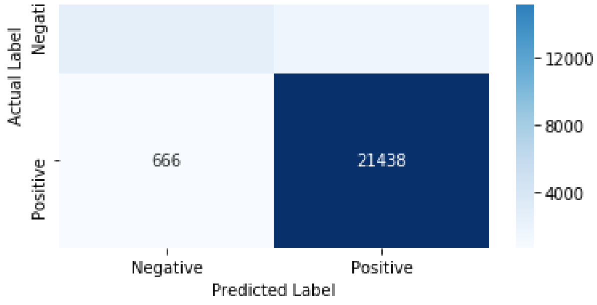
```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, best_t)),index=['Negative','P
ositive'],columns=['Negative','Positive'])
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_u
5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9983650479269355 for threshold 0.667
Train confusion matrix
```

Out[66]:

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



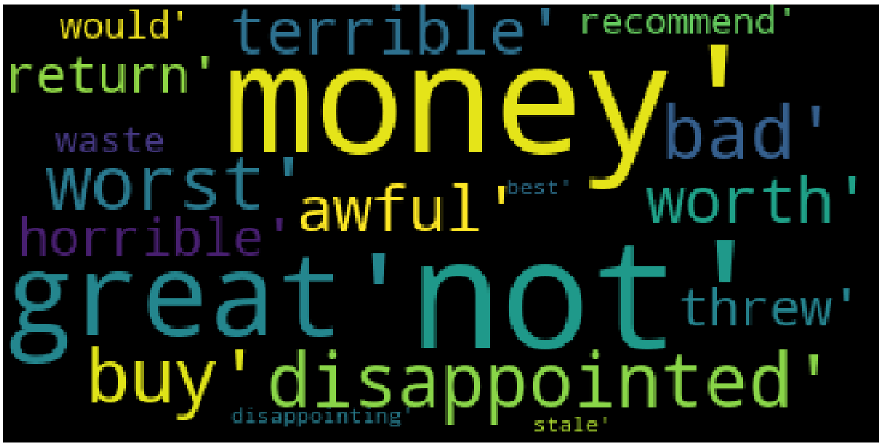


[5.1.4] Wordcloud of top 20 important features from SET 2

In [80]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
from wordcloud import WordCloud
all_features = tf_vect.get_feature_names()
tem=[]
rf=RandomForestClassifier(n_estimators=100,max_depth=None).fit(ftfx_tr,y_train)

features=np.argsort(rf.feature_importances_)[::-1]
for i in features[0:20]:
    tem.append(all_features[i])
print(tem)
words=str(tem) #converting list to string because wordcloud accepts data in string format

wordcloud = WordCloud( background_color ='black').generate(words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
#https://www.geeksforgeeks.org/generating-word-cloud-python/
```

```
['not', 'great', 'disappointed', 'worst', 'not buy', 'bad', 'terrible', 'awful', 'not worth', 'mon
ey', 'horrible', 'threw', 'return', 'not recommend', 'would not', 'waste money', 'would', 'disappo
inting', 'stale', 'best']
```



[5.1.5] Applying Random Forests on AVG W2V, SET 3

In [68]:

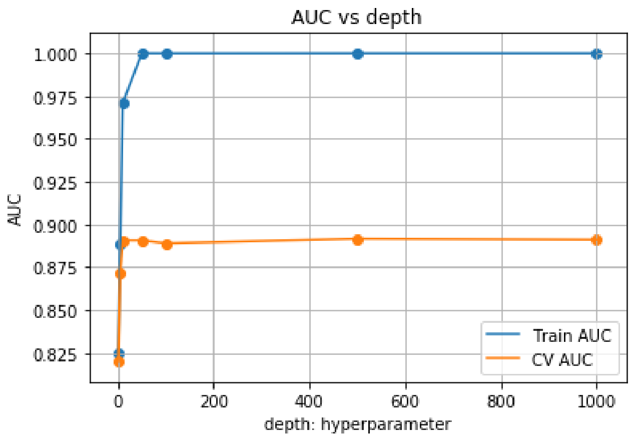
```
# Please write all the code with proper documentation
#Avg word2vec for train data
sent_train_list=[]
for sentence in x_train:
    sent_train_list.append(sentence.split())
w2v_model=Word2Vec(sent_train_list,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)

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_u
5LEf3yb0YJBVbZW#scrollTo=3-XGItt4PSx0
```

```
43008
50


18433
50


26332
```

```
26532
50
```

In [69]:

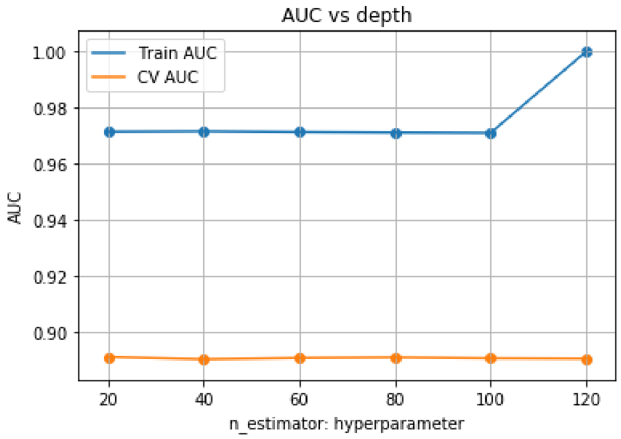
```
aw2vx_tr=sent_train_vectors
aw2vx_cv=sent_cv_vectors
aw2vx_te=sent_test_vectors
n_esti = [20,40,60,80,100,120]
depths=[1,5,10,50,100,500,1000]
best_n=[]
auc_train=[]
auc_cv=[]
for d in tqdm_notebook(depths):
    ms,rc=0,0
    #print(d)
    for n in n_esti:
        #print(m)
        rf=RandomForestClassifier(n_estimators=n,max_depth=d).fit(aw2vx_tr,y_train)
        prob_c=rf.predict_proba(aw2vx_cv)[:,1]
        val=roc_auc_score(y_cv,prob_c)
        if val>rc:
            rc=val
            ns=n
    rf=RandomForestClassifier(n_estimators=ns,max_depth=d).fit(aw2vx_tr,y_train)
    probcv=rf.predict_proba(aw2vx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,probcv))
    best_n.append(ns)
    probtr=rf.predict_proba(aw2vx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,probtr))

best_depth= depths[auc_cv.index(max(auc_cv))]
best_estimator=best_n[auc_cv.index(max(auc_cv))]

plt.plot(depths, auc_train, label='Train AUC')
plt.plot(depths, auc_cv, label='CV AUC')

plt.scatter(depths, auc_train)
plt.scatter(depths, auc_cv)
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs depth")
plt.grid()
plt.show()
print("Best Depth value for max auc =",best_depth)
print("Best n_estimator value for max auc =",best_estimator)
```



```
Best Depth value for max auc = 500
Best n_estimator value for max auc = 120
```

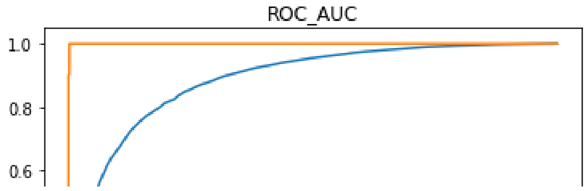
In [70]:

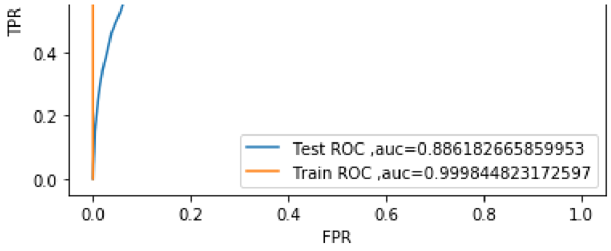
```
auc_train_n=[]
auc_cv_n=[]
for n in tqdm_notebook(n_esti):
    ms,rc=0,0
    #print(d)
    for d in (depths):
        #print(m)
        rf=RandomForestClassifier(n_estimators=n,max_depth=d).fit(aw2vx_tr,y_train)
        prob_c=rf.predict_proba(aw2vx_cv)[:,1]
        val=roc_auc_score(y_cv,prob_c)
        if val>rc:
            rc=val
            dep=d
    rf=RandomForestClassifier(n_estimators=ns,max_depth=dep).fit(aw2vx_tr,y_train)
    probcv=rf.predict_proba(aw2vx_cv)[:,1]
    auc_cv_n.append(roc_auc_score(y_cv,probcv))
    best_n.append(ns)
    probtr=rf.predict_proba(aw2vx_tr)[:,1]
    auc_train_n.append(roc_auc_score(y_train,probtr))


plt.plot(n_esti, auc_train_n, label='Train AUC')
plt.plot(n_esti, auc_cv_n, label='CV AUC')

plt.scatter(n_esti, auc_train_n)
plt.scatter(n_esti, auc_cv_n)
plt.legend()
plt.xlabel("n_estimator: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs depth")
plt.grid()
plt.show()
```



In [71]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_estimator,max_depth=best_depth).fit(aw2vx_tr,y_train)
pred_te=rf.predict_proba(aw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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()
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u
5LEf3yb0YJBVbZW
```





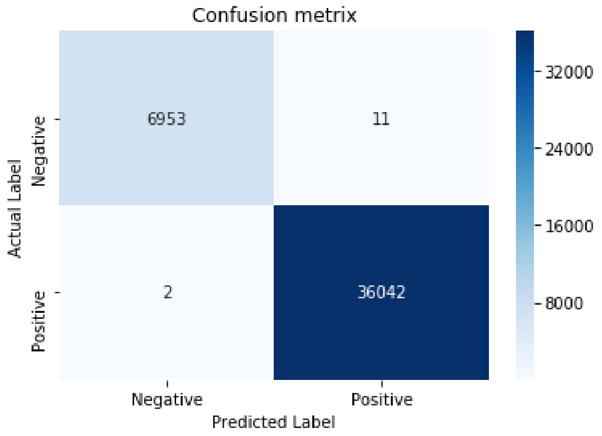
In [72]:

```
#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) 0.9983650479269355 for threshold 0.608
Train confusion matrix
```

Out[72]:

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



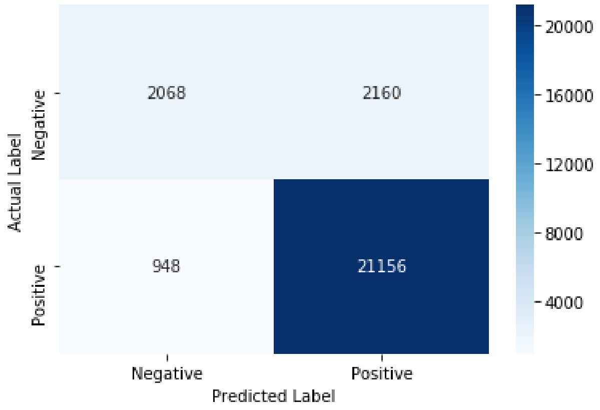
In [73]:

```
#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) 0.9983650479269355 for threshold 0.608
Train confusion matrix
```

Out[73]:

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



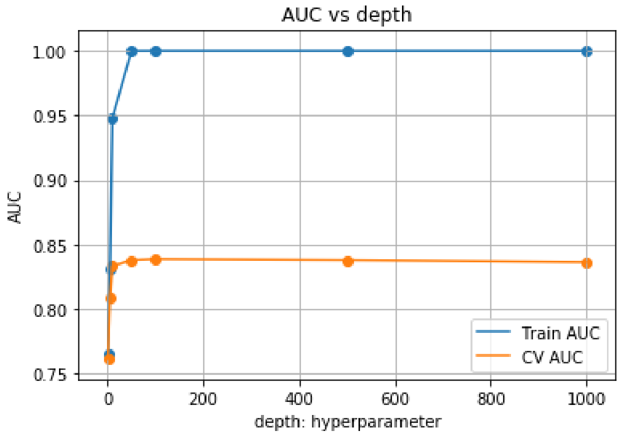
[5.1.6] Applying Random Forests on TFIDF W2V, SET 4

In [74]:

```
# 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
    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/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_test_vectors.append(sent_vec)
    row += 10
```

In [75]:

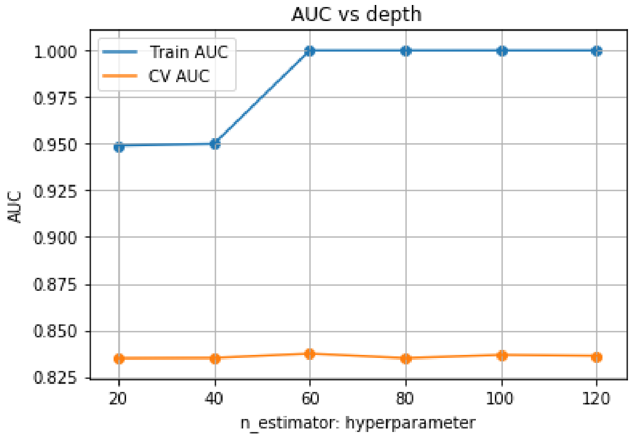
```
tfw2vx_tr=tfidf_sent_train_vectors
tfw2vx_cv=tfidf_sent_cv_vectors
tfw2vx_te=tfidf_sent_test_vectors
n_esti = [20,40,60,80,100,120]
depths=[1,5,10,50,100,500,1000]
best_n=[]
auc_train=[]
auc_cv=[]
for d in tqdm_notebook(depths):
    ms,rc=0,0
    #print(d)
    for n in n_esti:
        #print(m)
        rf=RandomForestClassifier(n_estimators=n,max_depth=d).fit(tfw2vx_tr,y_train)
        prob_c=rf.predict_proba(tfw2vx_cv)[:,1]
        val=roc_auc_score(y_cv,prob_c)
        if val>rc:
            rc=val
            ns=n
    rf=RandomForestClassifier(n_estimators=ns,max_depth=d).fit(tfw2vx_tr,y_train)
    probcv=rf.predict_proba(tfw2vx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,probcv))
    best_n.append(ns)
    probtr=rf.predict_proba(tfw2vx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,probtr))

best_depth= depths[auc_cv.index(max(auc_cv))]
best_estimator=best_n[auc_cv.index(max(auc_cv))]

plt.plot(depths, auc_train, label='Train AUC')
plt.plot(depths, auc_cv, label='CV AUC')

plt.scatter(depths, auc_train)
plt.scatter(depths, auc_cv)
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs depth")
plt.grid()
plt.show()
print("Best Depth value for max auc =",best_depth)
print("Best n_estimator value for max auc =",best_estimator)
```



```
Best Depth value for max auc = 100
Best n_estimator value for max auc = 120
```

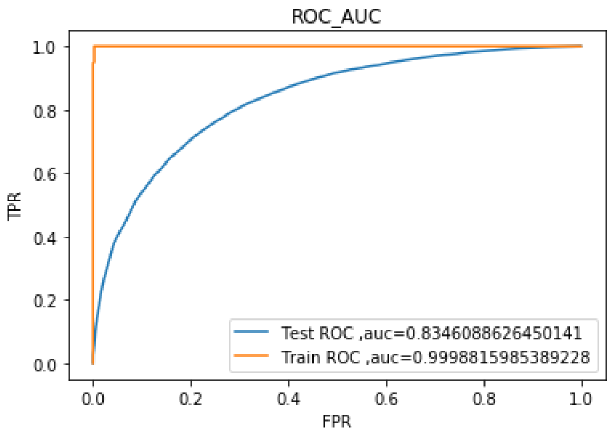
In [76]:

```
auc_train_n=[]
auc_cv_n=[]
for n in tqdm_notebook(n_esti):
    ms,rc=0,0
    #print(d)
    for d in (depths):
        #print(m)
        rf=RandomForestClassifier(n_estimators=n,max_depth=d).fit(tfw2vx_tr,y_train)
        prob_c=rf.predict_proba(tfw2vx_cv)[:,1]
        val=roc_auc_score(y_cv,prob_c)
        if val>rc:
            rc=val
            dep=d
    rf=RandomForestClassifier(n_estimators=ns,max_depth=dep).fit(tfw2vx_tr,y_train)
    probcv=rf.predict_proba(tfw2vx_cv)[:,1]
    auc_cv_n.append(roc_auc_score(y_cv,probcv))
    best_n.append(ns)
    probtr=rf.predict_proba(tfw2vx_tr)[:,1]
    auc_train_n.append(roc_auc_score(y_train,probtr))


plt.plot(n_esti, auc_train_n, label='Train AUC')
plt.plot(n_esti, auc_cv_n, label='CV AUC')

plt.scatter(n_esti, auc_train_n)
plt.scatter(n_esti, auc_cv_n)
plt.legend()
plt.xlabel("n_estimator: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs depth")
plt.grid()
plt.show()
```



In [77]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_estimator,max_depth=best_depth).fit(tfw2vx_tr,y_train)
pred_te=rf.predict_proba(tfw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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_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
```



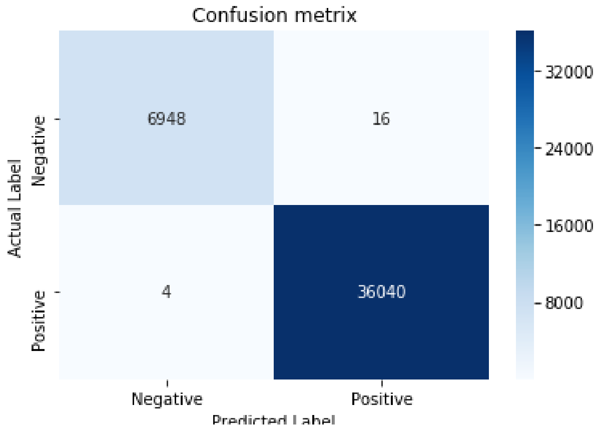
In [78]:

```
#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_u5LEf3yb0YJBVbZWb
```

```
the maximum value of tpr*(1-fpr) 0.9975917493399955 for threshold 0.633
Train confusion matrix
```

Out[78]:

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



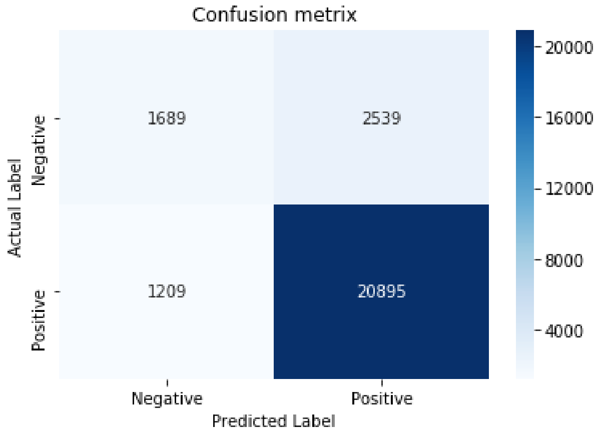
In [79]:

```
#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','P
ositive'],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_u
5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9975917493399955 for threshold 0.633
Train confusion matrix
```

Out[79]:

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



[5.2] Applying GBDT using XGBOOST

[5.2.1] Applying XGBOOST on BOW, SET 1

In [81]:

```
# Please write all the code with proper documentation
from xgboost import XGBClassifier
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)
xgb=XGBClassifier(booster='gbtree',max_depth=500,n_estimators=20).fit(fbowx_tr,y_train)
```

In [82]:

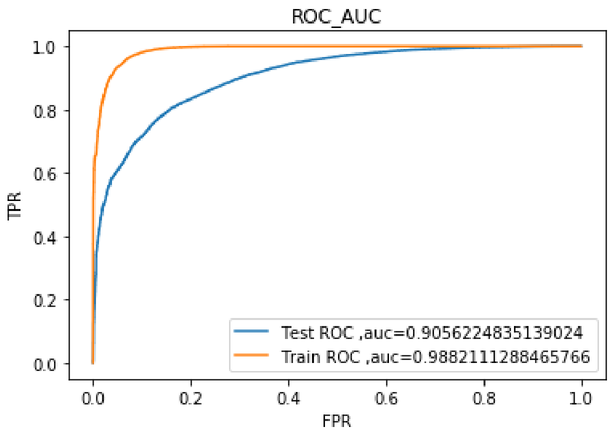
```
import warnings
warnings.filterwarnings("ignore")
import plotly.offline as offline
import plotly.graph_objs as go
n_esti=[20,40,60,80,100,120]
max_depth=[1,5,10,50,100,500,1000]
X=[]
Y=[]
auc_cv=[]
auc_train=[]
for n in tqdm_notebook(n_esti):
    for d in max_depth:
        xgb=XGBClassifier(booster='gbtree' ,max_depth=d,n_estimators=n).fit(fbowx_tr,y_train)
        pred_cv=xgb.predict_proba(fbowx_cv)[:,1]
        pred_tr=xgb.predict_proba(fbowx_tr)[:,1]
        X.append(n)
        Y.append(d)
        auc_cv.append(roc_auc_score(y_cv,pred_cv))
        auc_train.append(roc_auc_score(y_train,pred_tr))

best_depth=Y[auc_cv.index(max(auc_cv))]
best_n_estimator=X[auc_cv.index(max(auc_cv))]

print('optimal depth : ',best_depth)
print('optimal n_estimator : ',best_n_estimator)
offline.init_notebook_mode()
trace1 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_train, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_cv, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='n_estimators'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
optimal depth :  50
optimal n_estimator :  120
```

In [83]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_n_estimator,max_depth=best_depth).fit(fbowx_tr,y_train
)
```

```
,
pred_te=rf.predict_proba(fbowx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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_u
5LEf3yb0YJBVbZW
```



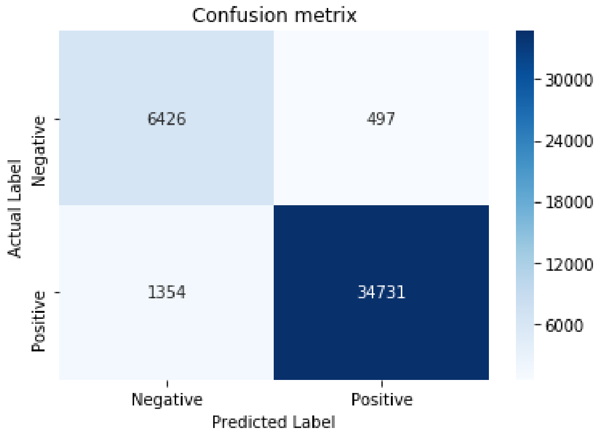
In [84]:

```
#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_u
5LEf3yb0YJBVbZWb
```

```
the maximum value of tpr*(1-fpr) 0.8933815268493822 for threshold 0.823
Train confusion matrix
```

Out[84]:

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



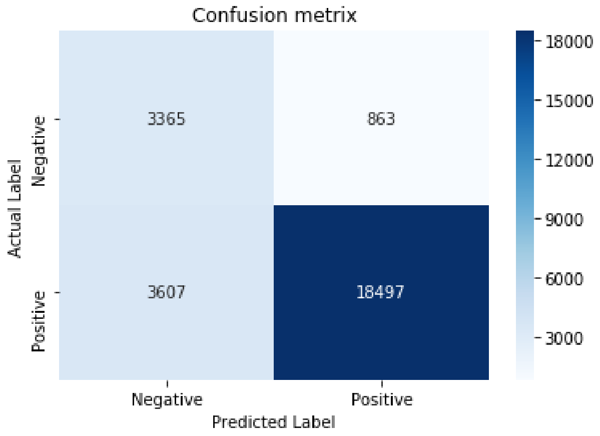
In [85]:

```
#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','P
ositive'],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_u
5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.8933815268493822 for threshold 0.823
Train confusion matrix
```

Out[85]:

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



[5.2.2] Applying XGBOOST on TFIDF, SET 2

In [86]:

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

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)

rf=RandomForestClassifier(n_estimators=100,max_depth=None).fit(ftfx_tr,y_train)
```

In [87]:

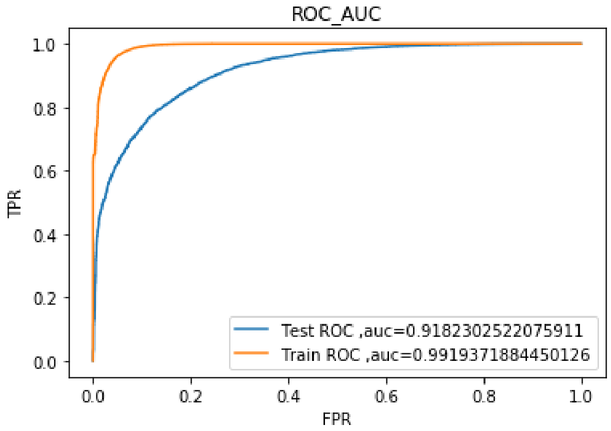
```
import warnings
warnings.filterwarnings("ignore")
import plotly.offline as offline
import plotly.graph_objs as go
import warnings
warnings.filterwarnings("ignore")
import plotly.offline as offline
import plotly.graph_objs as go
n_esti=[20,40,60,80,100,120]
max_depth=[1,5,10,50,100,500,1000]
X=[]
Y=[]
auc_cv=[]
auc_train=[]
```

```
for n in tqdm_notebook(n_esti):
    for d in max_depth:
        xgb=XGBClassifier(booster='gbtree' ,max_depth=d,n_estimators=n).fit(ftfx_tr,y_train)
        pred_cv=xgb.predict_proba(ftfx_cv)[:,1]
        pred_tr=xgb.predict_proba(ftfx_tr)[:,1]
        X.append(n)
        Y.append(d)
        auc_cv.append(roc_auc_score(y_cv,pred_cv))
        auc_train.append(roc_auc_score(y_train,pred_tr))

best_depth=Y[auc_cv.index(max(auc_cv))]
best_n_estimator=X[auc_cv.index(max(auc_cv))]

print('optimal depth : ',best_depth)
print('optimal n_estimator : ',best_n_estimator)
offline.init_notebook_mode()
trace1 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_train, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_cv, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='n_estimators'),
       zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
optimal depth :  50
optimal n_estimator :  120
```

In [88]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_n_estimator,max_depth=best_depth).fit(ftfx_tr,y_train)
pred_te=rf.predict_proba(ftfx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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_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_u
5LEf3yb0YJBVbZW
```



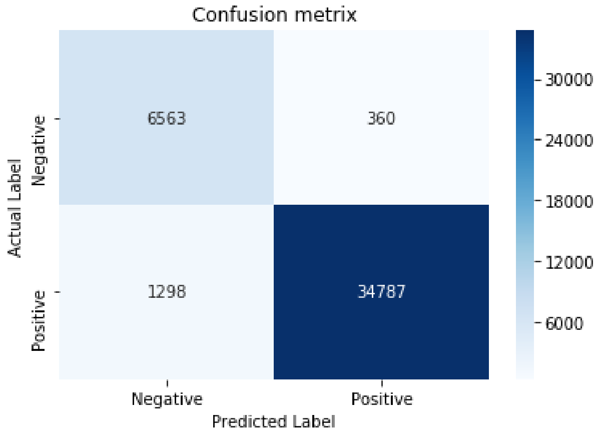
In [89]:

```
#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_u
5LEf3yb0YJBVbZWb
```

```
the maximum value of tpr*(1-fpr) 0.9138992905811588 for threshold 0.829
Train confusion matrix
```

Out[89]:

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



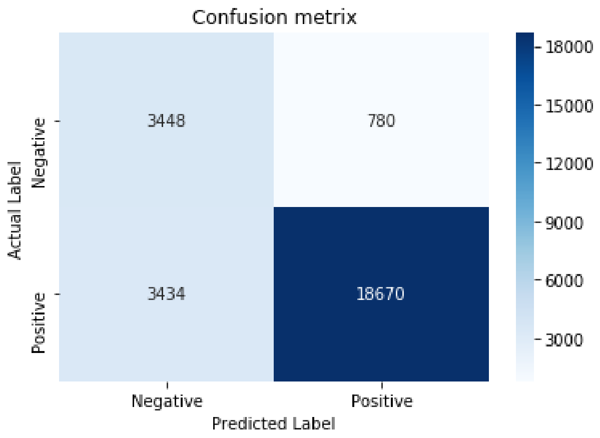
In [90]:

```
#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','P
ositive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Blues")
plt.title('Confusion metrix')
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_u
5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9138992905811588 for threshold 0.829
Train confusion matrix
```

Out[90]:

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



[5.2.3] Applying XGBOOST on AVG W2V, SET 3

In [91]:

```
# 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:
```

```
    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 googl'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_u
5LEf3yb0YJBVbZW#scrollTo=3-XGItt4PSx0
```

```
43008
50


18433
50


26332
50
```

In [92]:

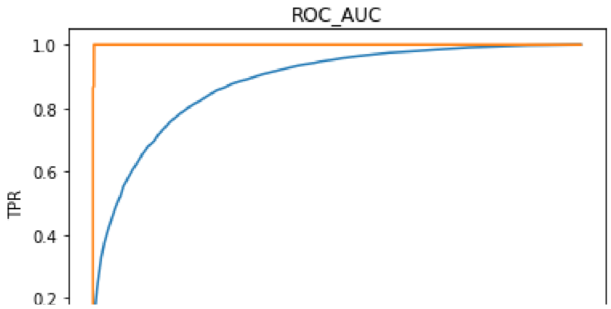
```
aw2vx_tr=np.array(sent_train_vectors)
aw2vx_cv=np.array(sent_cv_vectors)
aw2vx_te=np.array(sent_test_vectors)

n_esti=[20,40,60,80,100,120]
max_depth=[1,5,10,50,100,500,1000]
X=[]
Y=[]
auc_cv=[]
auc_train=[]
for n in tqdm_notebook(n_esti):
    for d in max_depth:
        xgb=XGBClassifier(booster='gbtree' ,max_depth=d,n_estimators=n).fit(aw2vx_tr,y_train)
        pred_cv=xgb.predict_proba(aw2vx_cv)[:,1]
        pred_tr=xgb.predict_proba(aw2vx_tr)[:,1]
        X.append(n)
        Y.append(d)
        auc_cv.append(roc_auc_score(y_cv,pred_cv))
        auc_train.append(roc_auc_score(y_train,pred_tr))

best_depth=Y[auc_cv.index(max(auc_cv))]
best_n_estimator=X[auc_cv.index(max(auc_cv))]

print('optimal depth : ',best_depth)
print('optimal n_estimator : ',best_n_estimator)
offline.init_notebook_mode()
trace1 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_train, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_cv, name = 'Cross validation')
data = [trace1, trace2]
```

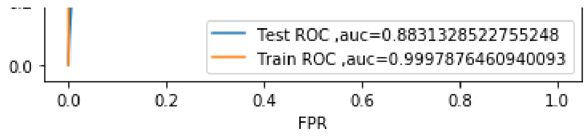
```
layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='n_estimators'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
optimal depth :  50
optimal n_estimator :  120
```

In [93]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_n_estimator,max_depth=best_depth).fit(aw2vx_tr,y_train
)
pred_te=rf.predict_proba(aw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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()
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u
5LEf3yb0YJBVbZW
```





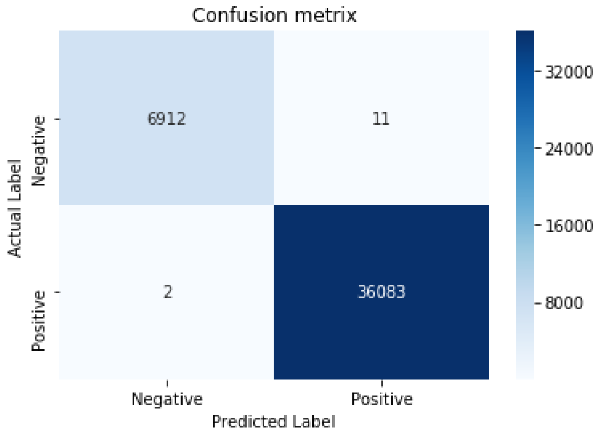
In [94]:

```
#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.com/drive/1EkYHI-vGKnURqLL_u5LEf3yb0YJBVbZWb
```

```
the maximum value of tpr*(1-fpr) 0.9983557568295491 for threshold 0.642
Train confusion matrix
```

Out[94]:

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



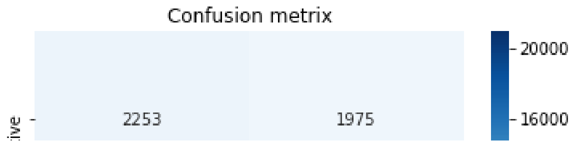
In [95]:

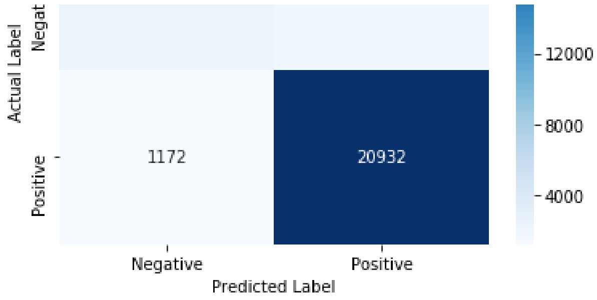
```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
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) 0.9983557568295491 for threshold 0.642
Train confusion matrix
```

Out[95]:

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





[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

In [96]:

```
# Please write all the code with proper documentation
# 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=[]
```

```
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/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_test_vectors.append(sent_vec)
    row += 10
```

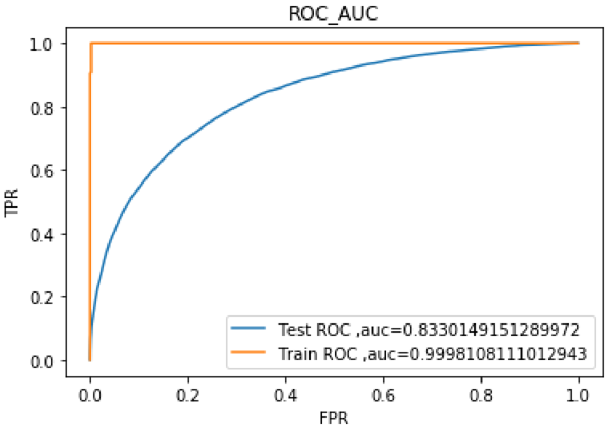
In [97]:

```
tfw2vx_tr=np.array(tfidf_sent_train_vectors)
tfw2vx_cv=np.array(tfidf_sent_cv_vectors)
tfw2vx_te=np.array(tfidf_sent_test_vectors)

n_esti=[20,40,60,80,100,120]
max_depth=[1,5,10,50,100,500,1000]
X=[]
Y=[]
auc_cv=[]
auc_train=[]
for n in tqdm_notebook(n_esti):
    for d in max_depth:
        xgb=XGBClassifier(booster='gbtree' ,max_depth=d,n_estimators=n).fit(tfw2vx_tr,y_train)
        pred_cv=xgb.predict_proba(tfw2vx_cv)[:,1]
        pred_tr=xgb.predict_proba(tfw2vx_tr)[:,1]
        X.append(n)
        Y.append(d)
        auc_cv.append(roc_auc_score(y_cv,pred_cv))
        auc_train.append(roc_auc_score(y_train,pred_tr))

best_depth=Y[auc_cv.index(max(auc_cv))]
best_n_estimator=X[auc_cv.index(max(auc_cv))]

print('optimal depth : ',best_depth)
print('optimal n_estimator : ',best_n_estimator)
offline.init_notebook_mode()
trace1 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_train, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_cv, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='n_estimators'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
optimal depth :  50
optimal n_estimator :  120
```

In [98]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_n_estimator,max_depth=best_depth).fit(tfw2vx_tr,y_train
)
pred_te=rf.predict_proba(tfw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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_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_u
5LEf3yb0YJBVbZW
```



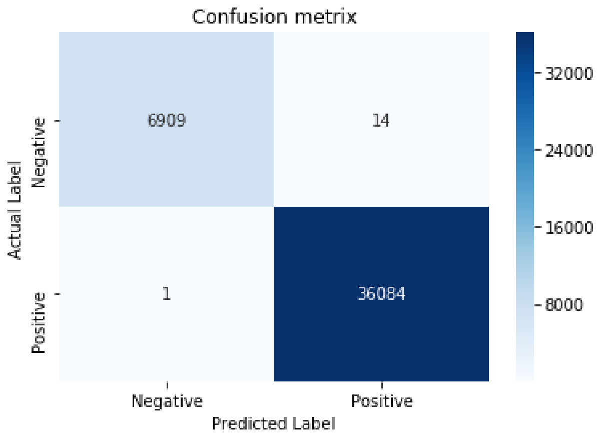
In [99]:

```
#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_u
5LEf3yb0YJBVbZWb
```

```
the maximum value of tpr*(1-fpr) 0.9979500990036865 for threshold 0.65
Train confusion matrix
```

Out[99]:

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



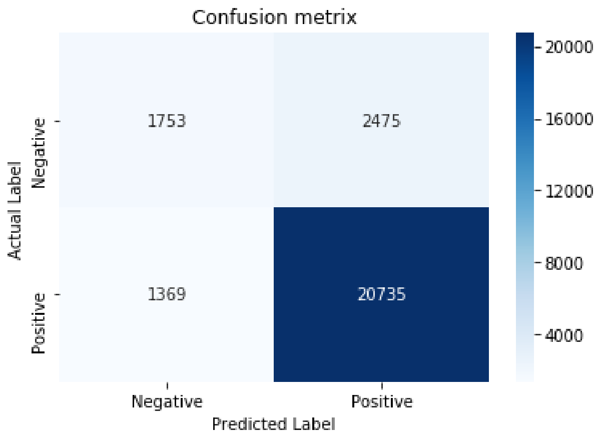
In [100]:

```
#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','P
ositive'],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_u
5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.9979500990036865 for threshold 0.65
Train confusion matrix
```

Out[100]:

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



[5.3] Feature Engineering

In [101]:

```
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_summary[i]+' '+str(len(final.Text.iloc[i]))
preprocessed_fe_reviews=preprocessed_reviews
```

In [102]:

```
preprocessed_fe_reviews[1500]
```

Out[102]:

```
'way hot blood took bite jig lol hot stuff 59'
```

[5.3.1] Applying XGBOOST on BOW, SET 1

In [103]:

```
# Please write all the code with proper documentation
from xgboost import XGBClassifier
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)
xgb=XGBClassifier(booster='gbtree',max_depth=500,n_estimators=20).fit(fbowx_tr,y_train)
```

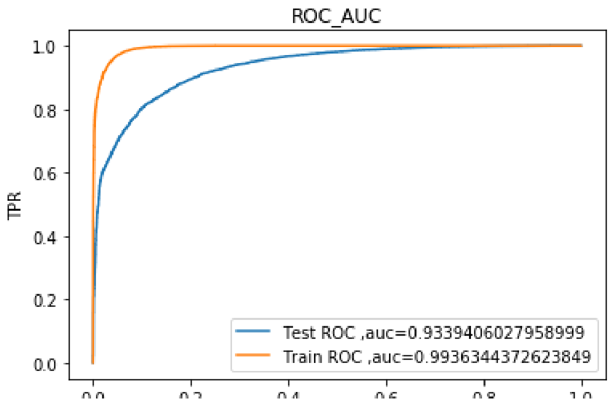
In [104]:

```
import warnings
warnings.filterwarnings("ignore")
import plotly.offline as offline
import plotly.graph_objs as go
n_esti=[20,40,60,80,100,120]
max_depth=[1,5,10,50,100,500,1000]
X=[]
Y=[]
auc_cv=[]
auc_train=[]
for n in tqdm_notebook(n_esti):
    for d in max_depth:
        xgb=XGBClassifier(booster='gbtree' ,max_depth=d,n_estimators=n).fit(fbowx_tr,y_train)
        pred_cv=xgb.predict_proba(fbowx_cv)[:,1]
        pred_tr=xgb.predict_proba(fbowx_tr)[:,1]
        X.append(n)
        Y.append(d)
        auc_cv.append(roc_auc_score(y_cv,pred_cv))
        auc_train.append(roc_auc_score(y_train,pred_tr))
        #print(XGBClassifier,Y,auc_train,auc_train)
best_depth=Y[auc_cv.index(max(auc_cv))]
best_n_estimator=X[auc_cv.index(max(auc_cv))]

print('optimal depth : ',best_depth)
print('optimal n_estimator : ',best_n_estimator)
offline.init_notebook_mode()
trace1 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_train, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_cv, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='n_estimators'),
        zaxis = dict(title='AUC'),))
        zaxis = dict(title='AUC')))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
optimal depth :  50
optimal n_estimator :  120
```

In [105]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_n_estimator,max_depth=best_depth).fit(fbowx_tr,y_train
)
pred_te=rf.predict_proba(fbowx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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_u
5LEf3yb0YJBVbZW
```



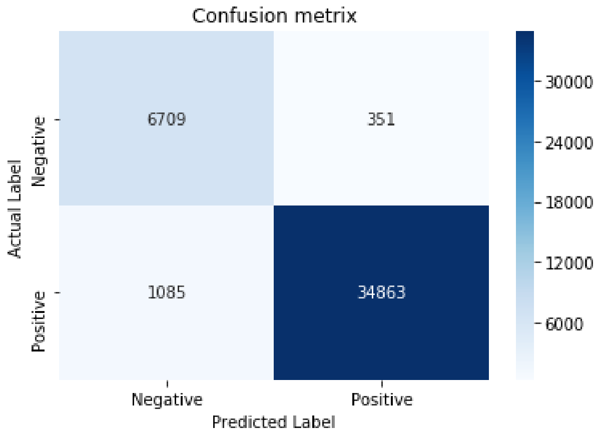
In [106]:

```
#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_u5LEf3yb0YJBVbZWb
```

```
the maximum value of tpr*(1-fpr) 0.9216013743175143 for threshold 0.817
Train confusion matrix
```

Out[106]:

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



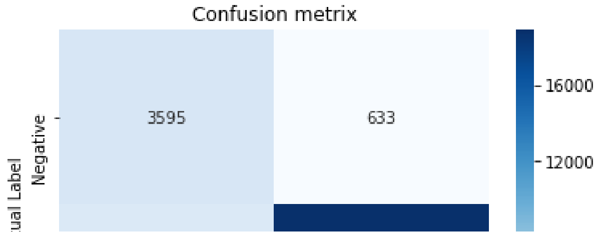
In [107]:

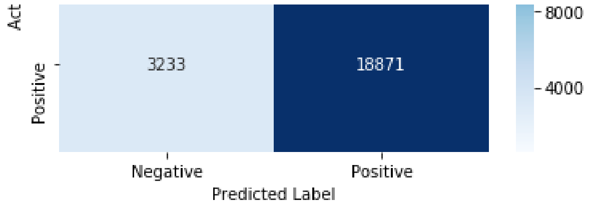
```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_te, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
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) 0.9216013743175143 for threshold 0.817
Train confusion matrix
```

Out[107]:

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





[5.2.2] Applying XGBOOST on TFIDF, SET 2

In [108]:

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

rf=RandomForestClassifier(n_estimators=100,max_depth=None).fit(ftfx_tr,y_train)
```

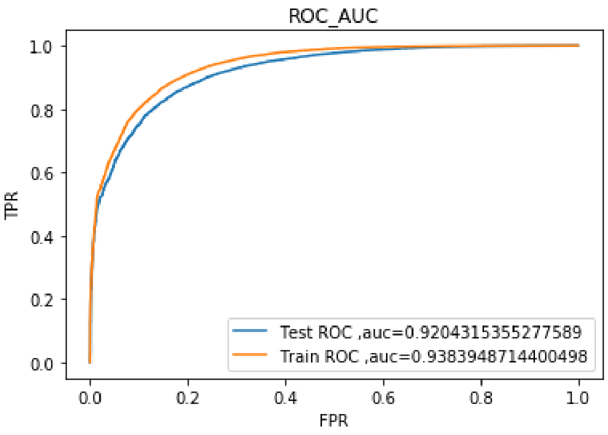
In [109]:

```
import warnings
warnings.filterwarnings("ignore")
import plotly.offline as offline
import plotly.graph_objs as go
n_esti=[20,40,60,80,100,120]
max_depth=[1,5,10,50,100,500,1000]
X=[]
Y=[]
auc_cv=[]
auc_train=[]
for n in tqdm_notebook(n_esti):
    for d in max_depth:
        xgb=XGBClassifier(booster='gbtree' ,max_depth=d,n_estimators=n).fit(ftfx_tr,y_train)
        pred_cv=xgb.predict_proba(ftfx_cv)[:,1]
        pred_tr=xgb.predict_proba(ftfx_tr)[:,1]
        X.append(n)
        Y.append(d)
        auc_cv.append(roc_auc_score(y_cv,pred_cv))
        auc_train.append(roc_auc_score(y_train,pred_tr))

best_depth=Y[auc_cv.index(max(auc_cv))]
best_n_estimator=X[auc_cv.index(max(auc_cv))]

print('optimal depth : ',best_depth)
print('optimal n_estimator : ',best_n_estimator)
offline.init_notebook_mode()
trace1 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_train, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_cv, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='n_estimators'),
       zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
optimal depth :  10
optimal n_estimator :  120
```

In [110]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_n_estimator,max_depth=best_depth).fit(ftfx_tr,y_train)
pred_te=rf.predict_proba(ftfx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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_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
```



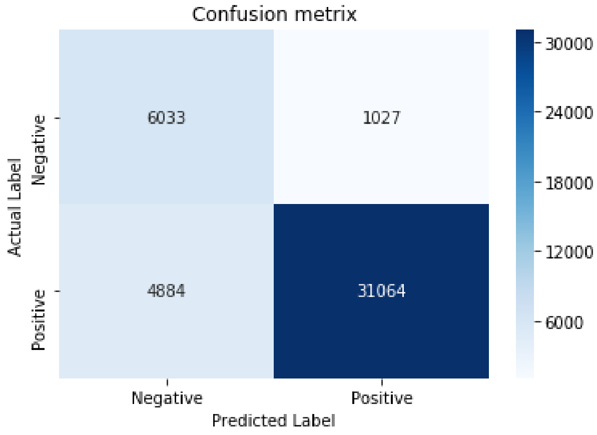
In [111]:

```
#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_u
5LEf3yb0YJBVbZWb
```

```
the maximum value of tpr*(1-fpr) 0.7384332925336597 for threshold 0.83
Train confusion matrix
```

Out[111]:

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



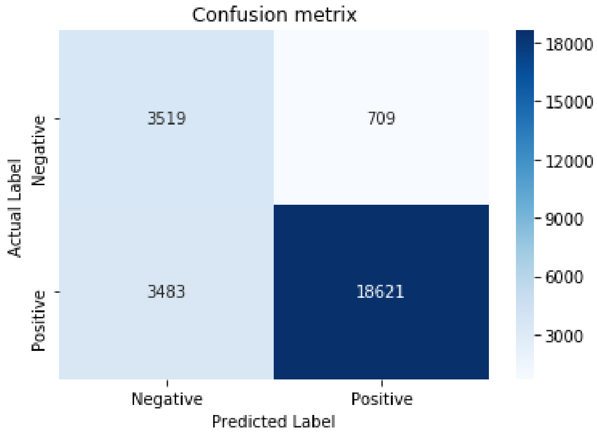
In [112]:

```
#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','P
ositive'],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_u
5LEf3yb0YJBVbZW
```

```
the maximum value of tpr*(1-fpr) 0.7384332925336597 for threshold 0.83
Train confusion matrix
```

Out[112]:

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



[5.2.3] Applying XGBOOST on AVG W2V, SET 3

In [113]:

```
# Please write all the code with proper documentation
#Avg word2vec for train data
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)

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_u
5LEf3yb0YJBVbZW#scrollTo=3-XGItt4PSx0
```

```
43008
50


18433
50


26332
50
```

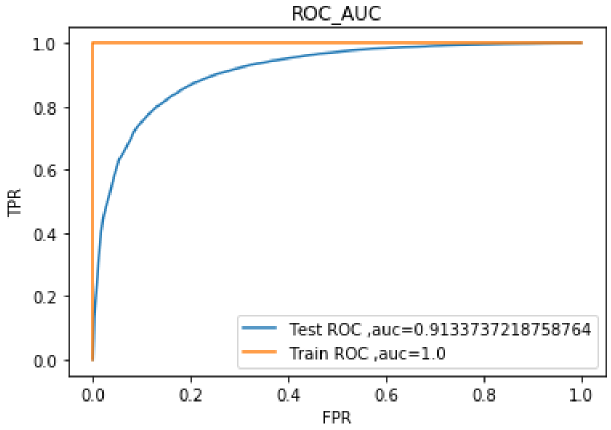
In [114]:

```
aw2vx_tr=np.array(sent_train_vectors)
aw2vx_cv=np.array(sent_cv_vectors)
aw2vx_te=np.array(sent_test_vectors)

n_esti=[20,40,60,80,100,120]
max_depth=[1,5,10,50,100,500,1000]
X=[]
Y=[]
auc_cv=[]
auc_train=[]
for n in tqdm_notebook(n_esti):
    for d in max_depth:
        xgb=XGBClassifier(booster='gbtree' ,max_depth=d,n_estimators=n).fit(aw2vx_tr,y_train)
        pred_cv=xgb.predict_proba(aw2vx_cv)[:,1]
        pred_tr=xgb.predict_proba(aw2vx_tr)[:,1]
        X.append(n)
        Y.append(d)
        auc_cv.append(roc_auc_score(y_cv,pred_cv))
        auc_train.append(roc_auc_score(y_train,pred_tr))
        #print(XGBClassifier,Y,auc_train,auc_train)
best_depth=Y[auc_cv.index(max(auc_cv))]
best_n_estimator=X[auc_cv.index(max(auc_cv))]

print('optimal depth : ',best_depth)
print('optimal n_estimator : ',best_n_estimator)
offline.init_notebook_mode()
trace1 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_train, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_cv, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='n_estimators'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
optimal depth :  50
optimal n_estimator :  120
```

In [115]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_n_estimator,max_depth=best_depth).fit(aw2vx_tr,y_train
)
pred_te=rf.predict_proba(aw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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()
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u
5LEf3yb0YJBVbZW
```



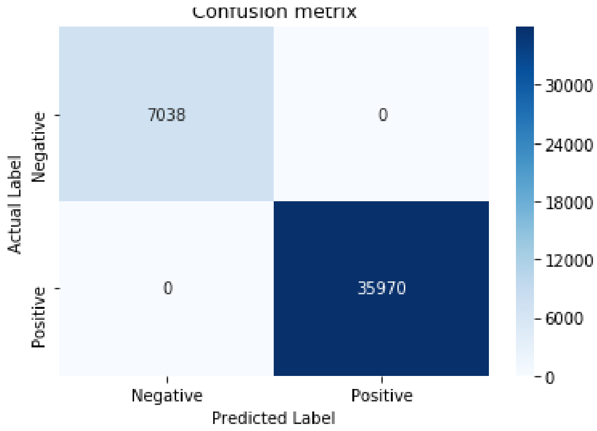
In [116]:

```
#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_u
5LEf3yb0YJBVbZWb
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 0.625
Train confusion matrix
```

Out[116]:

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



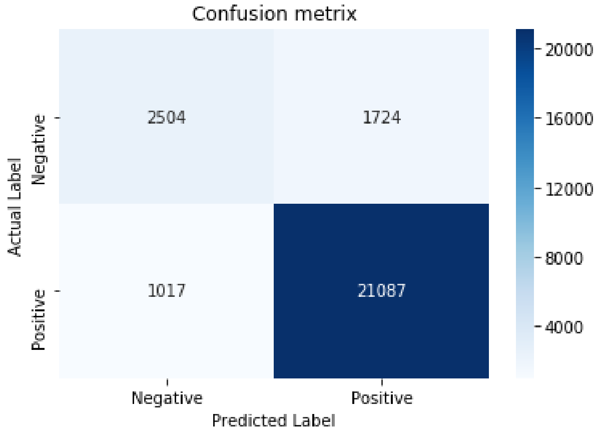
In [117]:

```
#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) 1.0 for threshold 0.625
Train confusion matrix
```

Out[117]:

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



[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

In [118]:

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

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)
```

```
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/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_test_vectors.append(sent_vec)
    row += 10
```

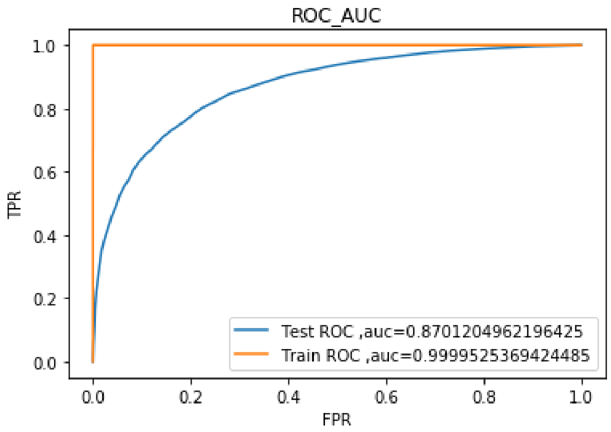
In [119]:

```
tfw2vx_tr=np.array(tfidf_sent_train_vectors)
tfw2vx_cv=np.array(tfidf_sent_cv_vectors)
tfw2vx_te=np.array(tfidf_sent_test_vectors)

n_esti=[20,40,60,80,100,120]
max_depth=[1,5,10,50,100,500,1000]
X=[]
Y=[]
auc_cv=[]
auc_train=[]
for n in tqdm_notebook(n_esti):
    for d in max_depth:
        xgb=XGBClassifier(booster='gbtree' ,max_depth=d,n_estimators=n).fit(tfw2vx_tr,y_train)
        pred_cv=xgb.predict_proba(tfw2vx_cv)[:,1]
        pred_tr=xgb.predict_proba(tfw2vx_tr)[:,1]
        X.append(n)
        Y.append(d)
        auc_cv.append(roc_auc_score(y_cv,pred_cv))
        auc_train.append(roc_auc_score(y_train,pred_tr))

best_depth=Y[auc_cv.index(max(auc_cv))]
best_n_estimator=X[auc_cv.index(max(auc_cv))]

print('optimal depth : ',best_depth)
print('optimal n_estimator : ',best_n_estimator)
offline.init_notebook_mode()
trace1 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_train, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=n_esti,z=auc_cv, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='n_estimators'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
optimal depth :  50
optimal n_estimator :  120
```

In [120]:

```
#Plotting ROC_AUC curve
rf=RandomForestClassifier(n_estimators=best_n_estimator,max_depth=best_depth).fit(tfw2vx_tr,y_train
)
pred_te=rf.predict_proba(tfw2vx_te)[:,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_te)
pred_tr=rf.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_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_u
5LEf3yb0YJBVbZW
```



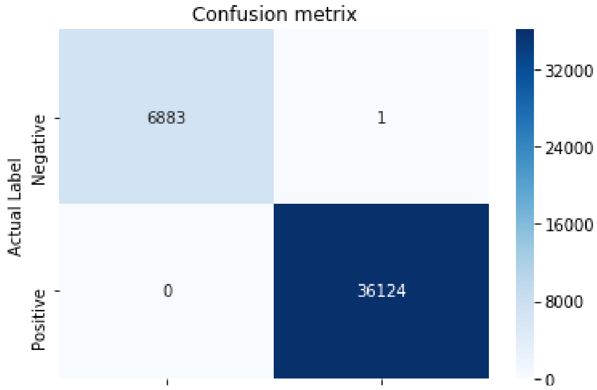
In [121]:

```
#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_u
5LEf3yb0YJBVbZWb
```

```
the maximum value of tpr*(1-fpr) 0.9998547356188263 for threshold 0.65
Train confusion matrix
```

Out[121]:

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



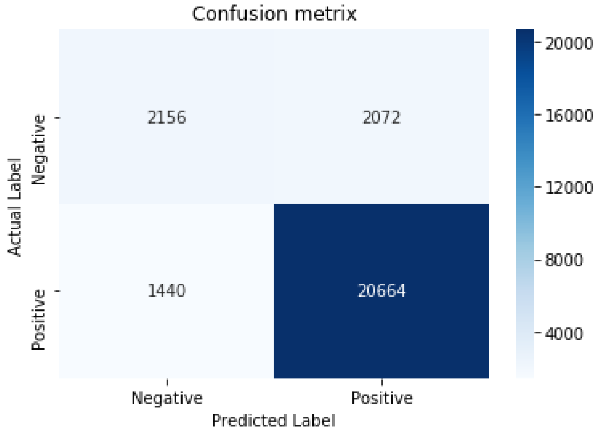
In [122]:

```
#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) 0.9998547356188263 for threshold 0.65
Train confusion matrix
```

Out[122]:

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



[6] Conclusions

In [125]:

```
# Please compare all your models using Prettytable library
# Please compare all your models using Prettytable library
x=PrettyTable()
x.field_names=(['Vectorizer','Model-Type','Best_Depth','Best_Split','AUC','Feature Engineering'])
x.add_row(['BOW','RF',500,120,0.914,'NO'])
x.add_row(['TF-IDF','RF',500,120,0.934,'NO'])
x.add_row(['AW2V','RF',500,120,0.886,'NO'])
x.add_row(['TF-IDF_w2v ','RF',100,120,0.834,'NO'])
x.add_row(['BOW','XGBOOST',50,120,0.905,'NO'])
x.add_row(['TF-IDF','XGBOOST',50,120,0.918,'NO'])
x.add_row(['AW2V','XGBOOST',50,120,0.883,'NO'])
x.add_row(['TF-IDF_w2v','XGBOOST',50,120,0.833,'NO'])
x.add_row(['BOW','XGBOOST',50,120,0.933,'Yes'])
x.add_row(['TF-IDF','XGBOOST',10,120,0.920,'Yes'])
x.add_row(['AW2V','XGBOOST',50,120,0.913,'Yes'])
x.add_row(['TF-IDF_w2v','XGBOOST',50,120,0.870,'Yes'])
print(x)
```

```
+-------------+------------+------------+------------+-------+---------------------+
|  Vectorizer | Model-Type | Best_Depth | Best_Split |  AUC  | Feature Engineering |
+-------------+------------+------------+------------+-------+---------------------+
|     BOW     |     RF     |    500     |    120     | 0.914 |          NO         |
|    TF-IDF   |     RF     |    500     |    120     | 0.934 |          NO         |
|     AW2V    |     RF     |    500     |    120     | 0.886 |          NO         |
|  TF-IDF_w2v |     RF     |    100     |    120     | 0.834 |          NO         |
|     BOW     |  XGBOOST   |     50     |    120     | 0.905 |          NO          |
|    TF-IDF   |  XGBOOST   |     50     |    120     | 0.918 |          NO          |
|     AW2V    |  XGBOOST   |     50     |    120     | 0.883 |          NO          |
|  TF-IDF_w2v |  XGBOOST   |     50     |    120     | 0.833 |          NO          |
|     BOW     |  XGBOOST   |     50     |    120     | 0.933 |         Yes          |
|    TF-IDF   |  XGBOOST   |     10     |    120     |  0.92 |         Yes          |
|     AW2V    |  XGBOOST   |     50     |    120     | 0.913 |         Yes          |
|  TF-IDF_w2v |  XGBOOST   |     50     |    120     |  0.87 |         Yes          |
+-------------+------------+------------+------------+-------+----------------------+
```

After feature engineering accuracy is increased