knnultimate_ipynb

February 12, 2019

1 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.

2 [1]. Reading Data

2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [2]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os
```

In [3]:
```
# 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 point
# 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 500
```

```
        # for tsne assignment you can take 5k data points



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

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

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

Out[3]:

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

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	\
0	1	1	1	1303862400	
1	0	0	0	1346976000	
2	1	1	1	1219017600	

	Summary	Text
0	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	"Delight" says it all	This is a confection that has been around a fe...

```
In [0]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
In [0]: display['COUNT(*)'].sum()
```

3 [2] Exploratory Data Analysis

3.1 [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 [0]: display= pd.read_sql_query("""
        SELECT *
```

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

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 [0]: #Sorting data according to ProductId in ascending order
        sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

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

```
Out[10]: (46072, 11)
```

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

```
Out[11]: 92.144
```

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 [0]: 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()
```

```
(46071, 11)
```

```
Out[13]: 5    31266
         4     7213
         1     4774
         2     2818
         Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [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
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
==================================================
```

```
In [15]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
         sent_0 = re.sub(r"http\S+", "", sent_0)
         sent_1000 = re.sub(r"http\S+", "", sent_1000)
```

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

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

In [16]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
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
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
```

In [0]:
```
# 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)
```

```
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
```

```
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
```

```
In [20]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
         print(sent_1500)
```

```
Great flavor low in calories high in nutrients high in protein Usually protein powders are high
```

```
In [0]: # 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
                    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
                    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
                    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
                    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
                    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
                    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any
                    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
                    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
```

```
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
                    '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 stopwo
             preprocessed_reviews.append(sentance.strip())
```

```
100%|| 46071/46071 [00:18<00:00, 2481.29it/s]
```

```
In [23]: preprocessed_reviews[1500]
```

```
Out[23]: 'great flavor low calories high nutrients high protein usually protein powders high pr
```

[3.2] Preprocessing Review Summary

```
In [0]: def func(x):
          if x>3:
            return 1
          else:
            return 0
```

```
In [0]: x=preprocessed_reviews
        y=final['Score'].apply(func)
```

```
In [0]: from sklearn.model_selection import train_test_split
        x1,xtest,y1,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

```
In [0]: xtrain,xcv,ytrain,ycv=train_test_split(x1,y1,test_size=0.2,random_state=1)
```

```
In [28]: print(len(xtrain))
         print(ytrain.shape)
         print(len(xtest))
         print(ytest.shape)
         print(len(xcv))
         print(ycv.shape)
```

```
25799
(25799,)
13822
(13822,)
6450
(6450,)
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [29]: from sklearn.feature_extraction.text import CountVectorizer
         count_vect=CountVectorizer()
         xtrainonehotencoding=count_vect.fit_transform(xtrain)
         xtestonehotencoding=count_vect.transform(xtest)
         xcvonehotencoding=count_vect.transform(xcv)
         print(xtrainonehotencoding.shape)
         print(xtestonehotencoding.shape)
         print(xcvonehotencoding.shape)
```

```
(25799, 29989)
(13822, 29989)
(6450, 29989)
```

```
In [0]: vect=CountVectorizer(min_df=10,max_features=50)
        xtrainonehotencoding1=vect.fit_transform(xtrain)
        xtestonehotencoding1=vect.transform(xtest)
        xcvonehotencoding1=vect.transform(xcv)
```

```
In [0]: xtrainonehotencoding11=xtrainonehotencoding1.toarray()
        xtestonehotencoding12=xtestonehotencoding1.toarray()
        xcvonehotencoding13=xcvonehotencoding1.toarray()
```

```
In [32]: print(type(xtrainonehotencoding11))
```

```
<class 'numpy.ndarray'>
```

5.2 [4.2] Bi-Grams and n-Grams.

```
In [32]: #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/mod

         # 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
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46072, 5000)
the number of unique words including both unigrams and bigrams  5000
```

5.3 [4.3] TF-IDF

```
In [65]: from sklearn.feature_extraction.text import TfidfVectorizer
         tfidf= TfidfVectorizer()
         xtraintfidfencoding=tfidf.fit_transform(xtrain)
         xtesttfidfencoding=tfidf.transform(xtest)
         xcvtfidfencoding=tfidf.transform(xcv)
         print(xtraintfidfencoding.shape)
         print(xtesttfidfencoding.shape)
         print(xcvtfidfencoding.shape)
```

```
(25799, 29989)
(13822, 29989)
(6450, 29989)
```

```
In [0]: vect=CountVectorizer(min_df=10,max_features=50)
        xtraintfidfencoding1=vect.fit_transform(xtrain)
        xtesttfidfencoding1=vect.transform(xtest)
        xcvtfidfencoding1=vect.transform(xcv)
```

```
In [0]: xtraintfidfencoding11=xtraintfidfencoding1.toarray()
        xtesttfidfencoding12=xtesttfidfencoding1.toarray()
        xcvtfidfencoding13=xcvtfidfencoding1.toarray()
```

5.4 [4.4] Word2Vec

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

```
In [77]: # 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=True
        want_to_use_google_w2v =True
        want_to_train_w2v = False

        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.bi
                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, t
```

```
you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v
```

```
In [78]: 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'))
```

```
[('good', 0.8157749772071838), ('wonderful', 0.8038014769554138), ('fantastic', 0.802417933940
==================================================
[('best', 0.7791042327880859), ('nastiest', 0.7564510107040405), ('hottest', 0.743679821491241
```

```
In [79]: 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  9654
sample words  ['bitters', 'go', 'style', 'called', 'drinks', 'food', 'dishes', 'old', 'fashion
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

```
In [80]: # 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(xtrain): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
             cnt_words =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
                 if word in w2v_words:
                     vec = w2v_model.wv[word]
                     sent_vec += vec
                     cnt_words += 1
             if cnt_words != 0:
                 sent_vec /= cnt_words
             sent_vectors.append(sent_vec)
         print(len(sent_vectors))
         print(len(sent_vectors[0]))
```

```
100%|| 25799/25799 [14:10<00:00, 30.32it/s]
```

```
25799
50
```

```
In [81]: # average Word2Vec
         # compute average word2vec for each review.
         sent_vectorstest = []; # the avg-w2v for each sentence/review is stored in this list
         for sent in tqdm(xtest): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
             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_vectorstest.append(sent_vec)
         print(len(sent_vectorstest))
         print(len(sent_vectorstest))
```

```
100%|| 13822/13822 [07:33<00:00, 30.51it/s]
```

```
13822
13822
```

```
In [82]: # average Word2Vec
         # compute average word2vec for each review.
         sent_vectorscv = []; # the avg-w2v for each sentence/review is stored in this list
         for sent in tqdm(xcv): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
             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_vectorscv.append(sent_vec)
         print(len(sent_vectorscv))
         print(len(sent_vectorscv))
```

```
100%|| 6450/6450 [03:31<00:00, 30.54it/s]
```

```
6450
6450
```

```
In [0]: xtrainy=sent_vectors
        xtesty=sent_vectorstest
```

[4.4.1.2] TFIDF weighted W2v

```
In [0]: model = TfidfVectorizer()
        xtraintfidfw2v = model.fit_transform(preprocessed_reviews)
        #xtesttfidfw2v=model.transform(xtest)
        #xcvtfidfw2v=model.transform(xcv)
        tfidf_feat = model.get_feature_names()
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [91]: xcvtfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in thi
         row=0;
         for sent in tqdm(xcv): # for each review/sentence
             sent_vec = np.zeros(50)
```

```
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent.split(' '): # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec /= weight_sum
            xcvtfidf_sent_vectors.append(sent_vec)
            row += 1
```

```
100%|| 6450/6450 [02:17<00:00, 46.66it/s]
```

```
In [92]: xtraintfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in
         row=0;
         for sent in tqdm(xtrain): # for each review/sentence
             sent_vec = np.zeros(50)
             weight_sum =0; # num of words with a valid vector in the sentence/review
             for word in sent.split(' '): # for each word in a review/sentence
                 if word in w2v_words and word in tfidf_feat:
                     vec = w2v_model.wv[word]
                     tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                     sent_vec += (vec * tf_idf)
                     weight_sum += tf_idf
             if weight_sum != 0:
                 sent_vec /= weight_sum
             xtraintfidf_sent_vectors.append(sent_vec)
             row += 1
```

```
100%|| 25799/25799 [09:19<00:00, 46.07it/s]
```

```
In [93]: xtesttfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in t
         row=0;
         for sent in tqdm(xtest): # for each review/sentence
             sent_vec = np.zeros(50)
             weight_sum =0; # num of words with a valid vector in the sentence/review
             for word in sent.split(' '): # for each word in a review/sentence
                 if word in w2v_words and word in tfidf_feat:
                     vec = w2v_model.wv[word]
                     tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                     sent_vec += (vec * tf_idf)
                     weight_sum += tf_idf
             if weight_sum != 0:
                 sent_vec /= weight_sum
             xtesttfidf_sent_vectors.append(sent_vec)
             row += 1
```

```
100%|| 13822/13822 [04:59<00:00, 46.22it/s]
```

6 [5] Assignment 3: KNN

```
<li><strong>Apply Knn(brute force version) on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>Apply Knn(kd tree version) on these feature sets</strong>
    <br><font color='red'>NOTE: </font>sklearn implementation of kd-tree accepts only dense mat
    <ul>
        <li><font color='red'>SET 5:</font>Review text, preprocessed one converted into vectors
        <pre>
        count_vect = CountVectorizer(min_df=10, max_features=500)
        count_vect.fit(preprocessed_reviews)
        </pre>
        </li>
        <li><font color='red'>SET 6:</font>Review text, preprocessed one converted into vectors
        <pre>
            tf_idf_vect = TfidfVectorizer(min_df=10, max_features=500)
            tf_idf_vect.fit(preprocessed_reviews)
        </pre>
        </li>
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning(find best K)</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
<br>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
```

```
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

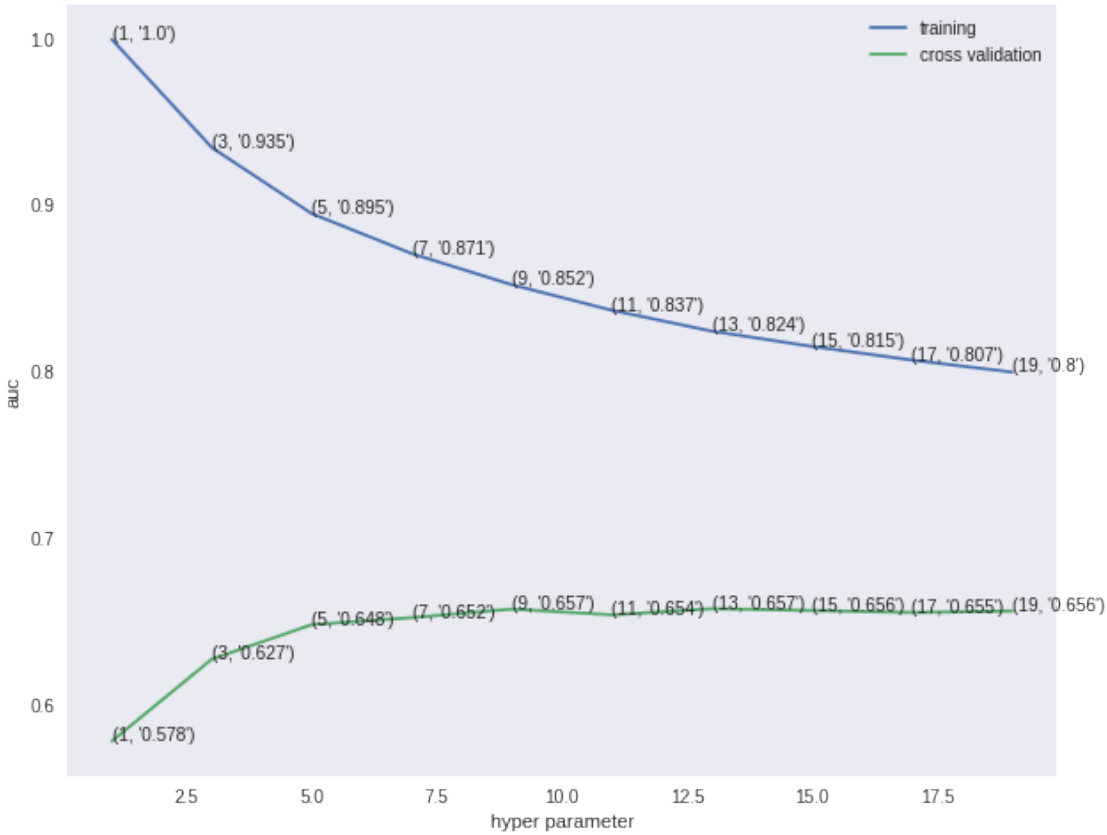
1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply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

6.1 [5.1] Applying KNN brute force

6.1.1 [5.1.1] Applying KNN brute force on BOW, SET 1

```
In [57]: #with hyper parameter tuning
         from sklearn.metrics import auc
         from sklearn.metrics import roc_auc_score
         from sklearn.neighbors import KNeighborsClassifier
         cvscores=[]
         cvscores1=[]

         alpha=[i for i in range(1,20,2)]
         for i in alpha:
             knnx=KNeighborsClassifier(n_neighbors=i,algorithm='brute')
             knnx.fit(xtrainonehotencoding,ytrain)
             predict1=knnx.predict_proba(xtrainonehotencoding)[:,1]
             cvscores.append(roc_auc_score(ytrain,predict1))
             predict2=knnx.predict_proba(xcvonehotencoding)[:,1]
             cvscores1.append(roc_auc_score(ycv,predict2))
             optimal_k=np.argmax(cvscores1)
         fig,ax=plt.subplots(figsize=(10,8))
         ax.plot(alpha,cvscores,label='training')
         for i,txt in enumerate(np.round(cvscores,3)):
           ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores[i]))
         plt.grid()
         ax.legend()
```

```
ax.plot(alpha,cvscores1,label='cross validation')
for i,txt in enumerate(np.round(cvscores1,3)):
  ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores1[i]))
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
print(alpha[optimal_k])
print(cvscores1)
```




```
13
[0.5778232271198394, 0.6271216643572783, 0.6478549375914228, 0.6520291396128493, 0.65718480293
```

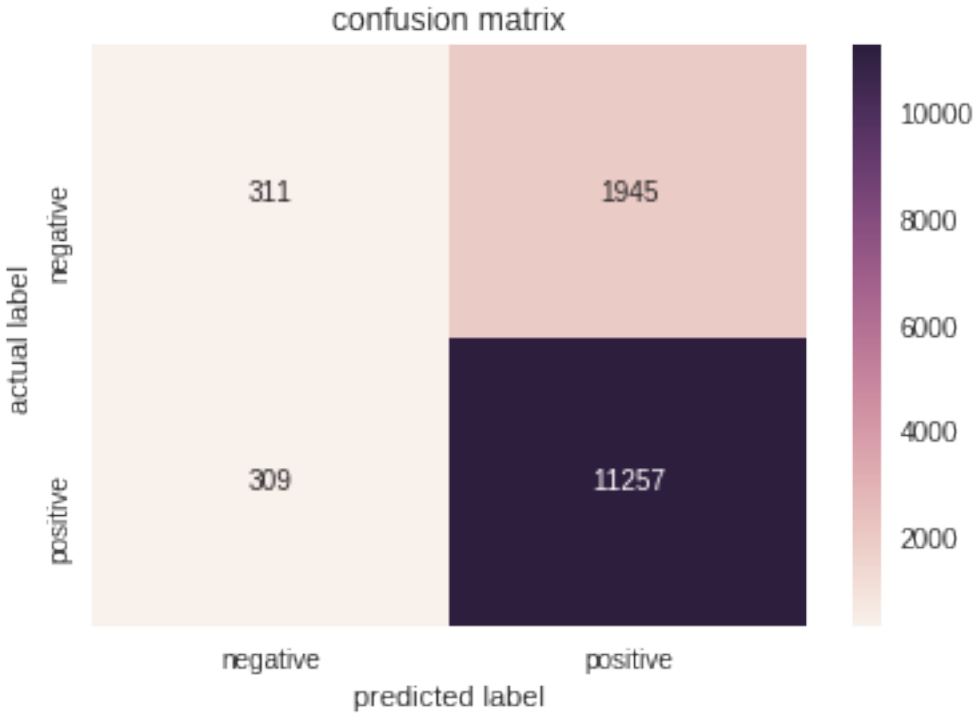
```
In [58]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import roc_auc_score
         knne=KNeighborsClassifier(n_neighbors=alpha[optimal_k],algorithm='brute')
         knne.fit(xtrainonehotencoding,ytrain)
         predictrain=knne.predict(xtrainonehotencoding)
```

```
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=knne.predict(xtestonehotencoding)
fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
auc = metrics.roc_auc_score(ytest, predic)
plt.plot(fpr,tpr,label="roc of test)")
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
print(roc_auc_score(ytest, predic))
```

```
0.5555691863409872
```

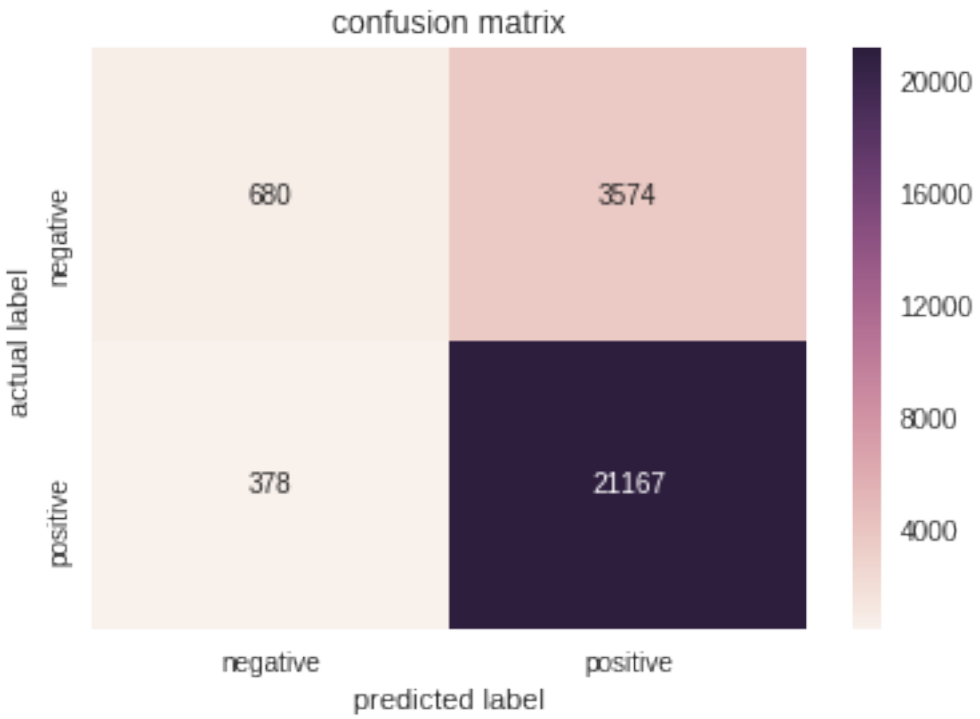
In [59]:
```
#plotting confusion matrix aftyer performing knn on top of svd data
from sklearn.metrics import confusion_matrix
rest=confusion_matrix(ytest,predic)
import seaborn as sns
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
```

```
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



In [60]:
```
print('train confusion matrix')
rest=confusion_matrix(ytrain,predictrain)
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```

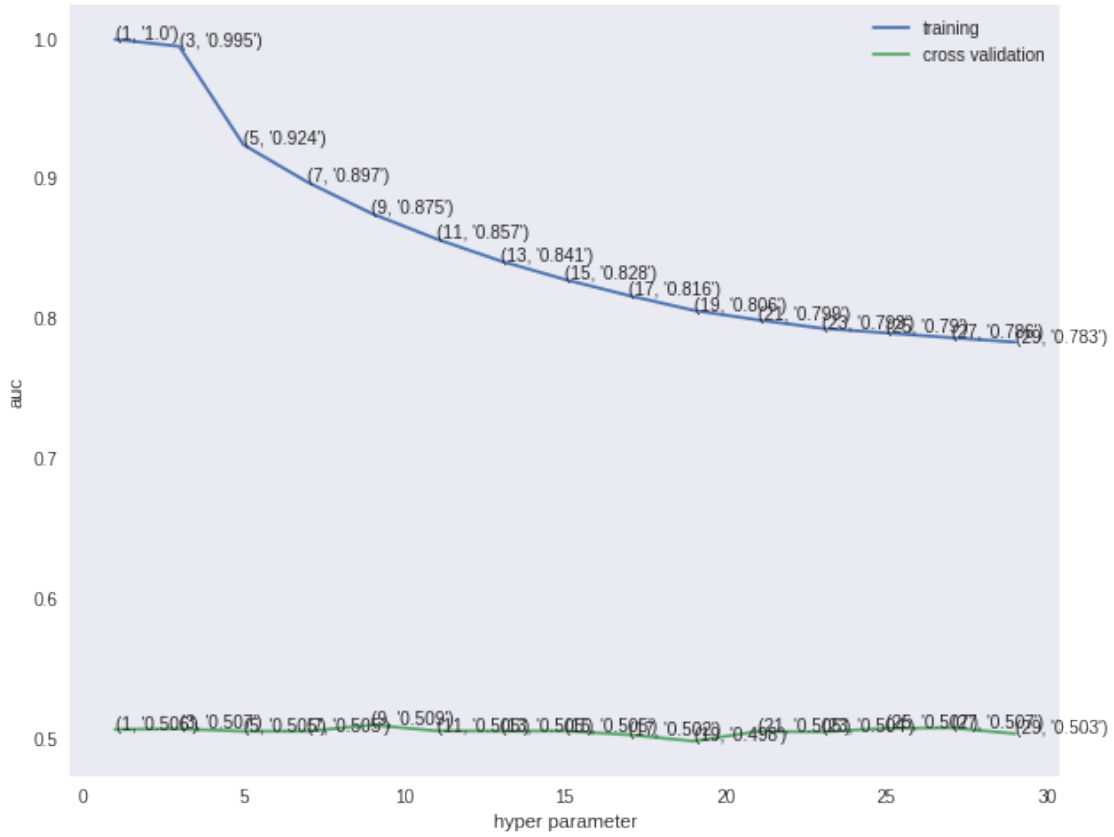
```
train confusion matrix
```



6.1.2 [5.1.2] Applying KNN brute force on TFIDF, SET 2

```
In [102]: #with hyper parameter tuning
          from sklearn.metrics import auc
          from sklearn.metrics import roc_auc_score
          from sklearn.neighbors import KNeighborsClassifier
          cvscores=[]
          cvscores1=[]

          alpha=[i for i in range(1,30,2)]
          for i in alpha:
              knnx=KNeighborsClassifier(n_neighbors=i,algorithm='brute')
              knnx.fit(xtraintfidfencoding,ytrain)
              predict1=knnx.predict_proba(xtraintfidfencoding)[:,1]
              cvscores.append(roc_auc_score(ytrain,predict1))
              predict2=knnx.predict_proba(xcvtfidfencoding)[:,1]
              cvscores1.append(roc_auc_score(ycv,predict2))
          optimal_k=np.argmax(cvscores1)
          fig,ax=plt.subplots(figsize=(10,8))
          ax.plot(alpha,cvscores,label='training')
          for i,txt in enumerate(np.round(cvscores,3)):
            ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores[i]))
          plt.grid()
          ax.legend()
```

```
ax.plot(alpha,cvscores1,label='cross validation')
for i,txt in enumerate(np.round(cvscores1,3)):
  ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores1[i]))
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
print(alpha[optimal_k])
print(cvscores1)
```




```
9
[0.506161142499814, 0.5065446536055381, 0.504868740203465, 0.504923576696023, 0.509295000702458
```

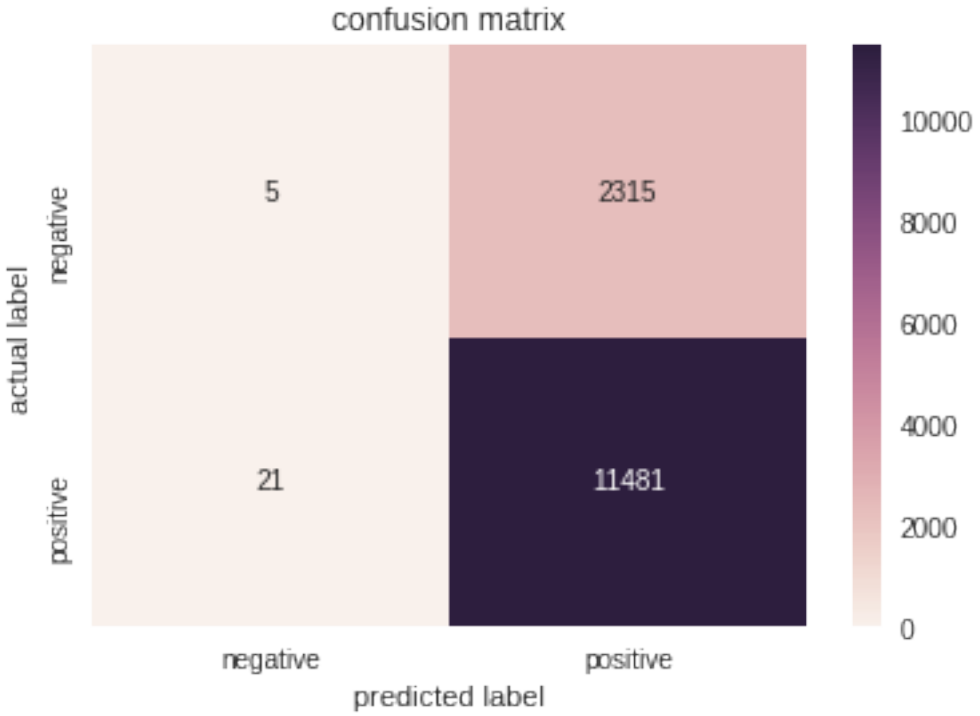
```
In [106]: from sklearn.neighbors import KNeighborsClassifier
          from sklearn.metrics import accuracy_score
          from sklearn.metrics import roc_auc_score
          knne=KNeighborsClassifier(n_neighbors=9,algorithm='brute')
          knne.fit(xtraintfidfencoding,ytrain)
          predictrain=knne.predict(xtraintfidfencoding)
```

```
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=knne.predict(xtesttfidfencoding)
fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
auc = metrics.roc_auc_score(ytest, predic)
plt.plot(fpr,tpr,label="roc of test)")
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
print(roc_auc_score(ytest, predic))
```

```
0.500178401048316
```

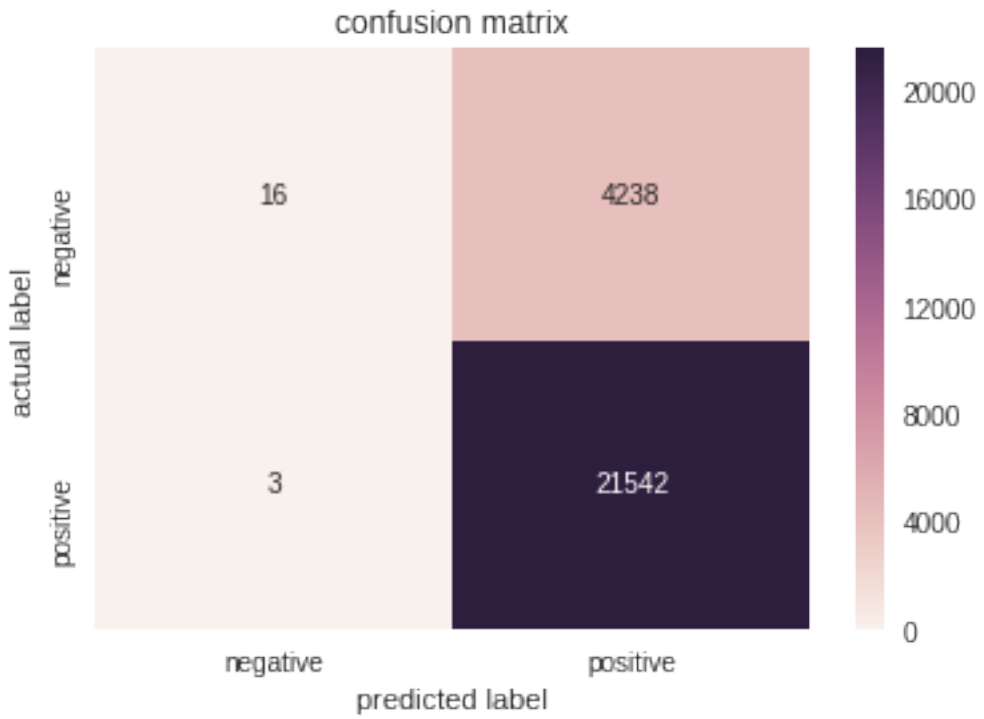

```
In [0]: #plotting confusion matrix aftyer performing knn on top of svd data
        from sklearn.metrics import confusion_matrix
        rest=confusion_matrix(ytest,predic)
        import seaborn as sns
        classlabel=['negative','positive']

        frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
```

```
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



In [107]:
```
print('train confusion matrix')
rest=confusion_matrix(ytrain,predictrain)
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```

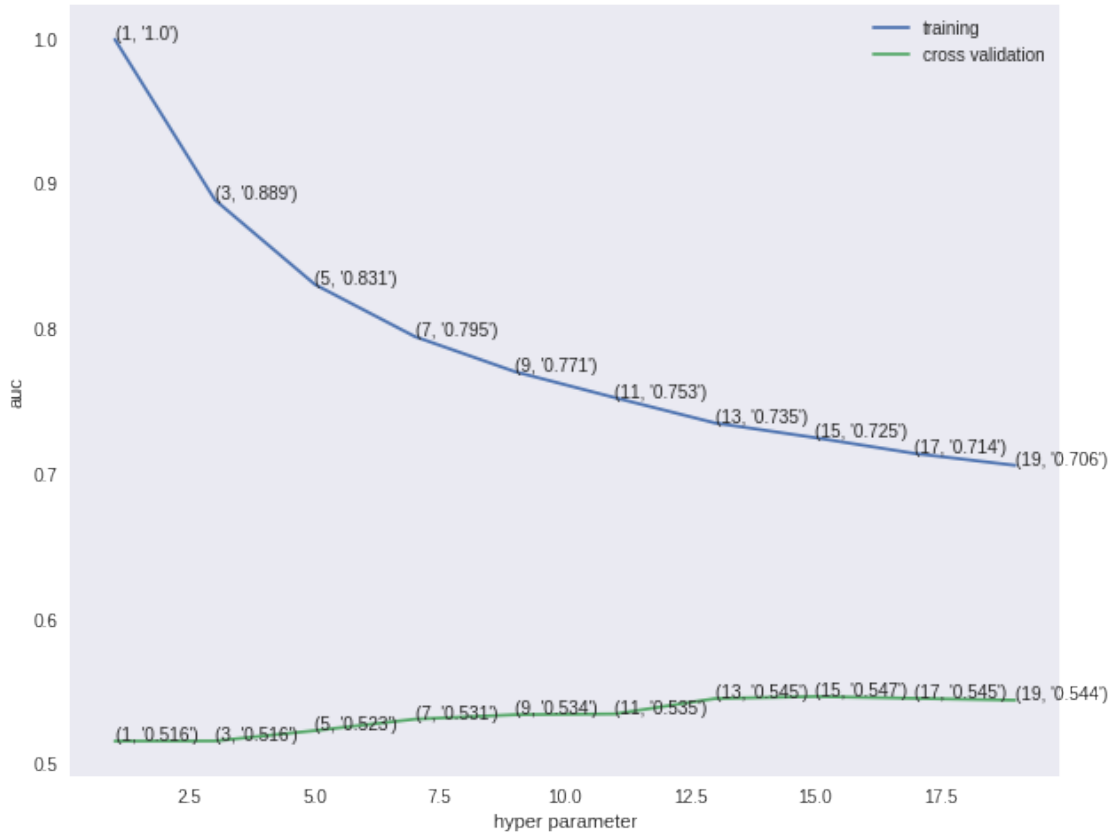
```
train confusion matrix
```



6.1.3 [5.1.3] Applying KNN brute force on AVG W2V, SET 3

```
In [48]: #with hyper parameter tuning
         from sklearn.metrics import auc
         from sklearn.metrics import roc_auc_score
         from sklearn.neighbors import KNeighborsClassifier
         cvscores=[]
         cvscores1=[]

         alpha=[i for i in range(1,20,2)]
         for i in alpha:
             knnx=KNeighborsClassifier(n_neighbors=i,algorithm='brute')
             knnx.fit(sent_vectors,ytrain)
             predict1=knnx.predict_proba(sent_vectors)[:,1]
             predictz=knnx.predict(sent_vectors)
             cvscores.append(roc_auc_score(ytrain,predict1))
             predict2=knnx.predict_proba(sent_vectorscv)[:,1]
             cvscores1.append(roc_auc_score(ycv,predict2))
             optimal_k=np.argmax(cvscores1)
         fig,ax=plt.subplots(figsize=(10,8))
         ax.plot(alpha,cvscores,label='training')
         for i,txt in enumerate(np.round(cvscores,3)):
           ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores[i]))
         plt.grid()
```

```
ax.legend()
ax.plot(alpha,cvscores1,label='cross validation')
for i,txt in enumerate(np.round(cvscores1,3)):
  ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores1[i]))
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
print(alpha[optimal_k])
print(cvscores1)
```

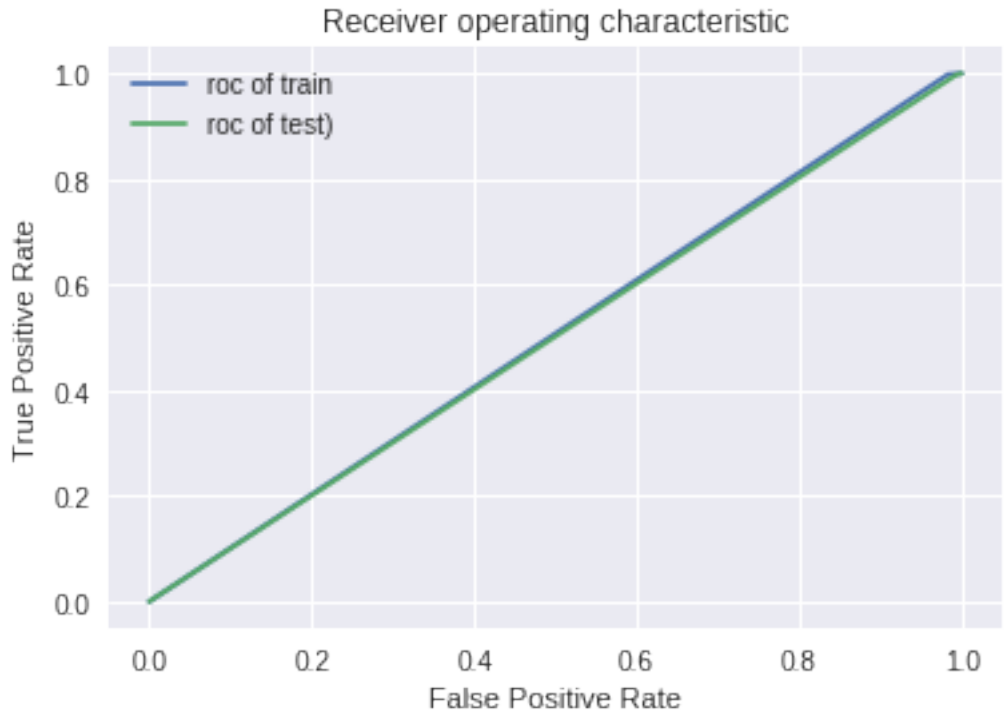



```
15
[0.515962359735011, 0.5160707422120141, 0.5231926164569716, 0.5310807460027573, 0.534165064573
```

```
In [84]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import roc_auc_score
         knne=KNeighborsClassifier(n_neighbors=15,algorithm='brute')
         knne.fit(sent_vectors,ytrain)
```

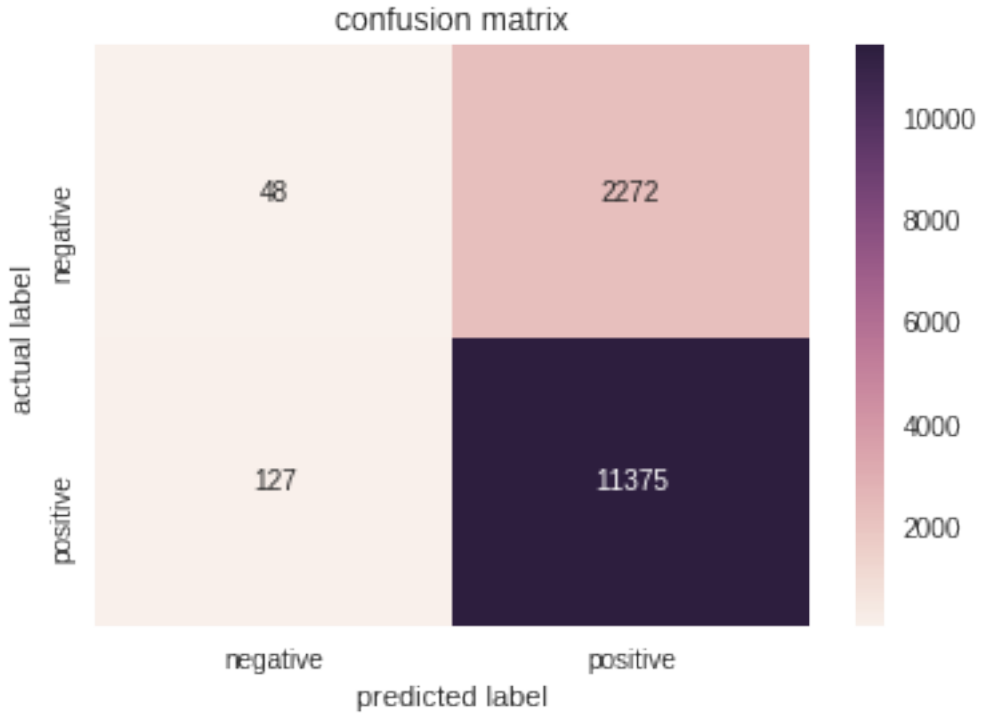
```
predictrain=knne.predict(sent_vectors)
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=knne.predict(sent_vectorstest)
fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
auc = metrics.roc_auc_score(ytest, predic)
plt.plot(fpr,tpr,label="roc of test)")
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
print(roc_auc_score(ytest, predic))
```

```
0.5020220829454883
```



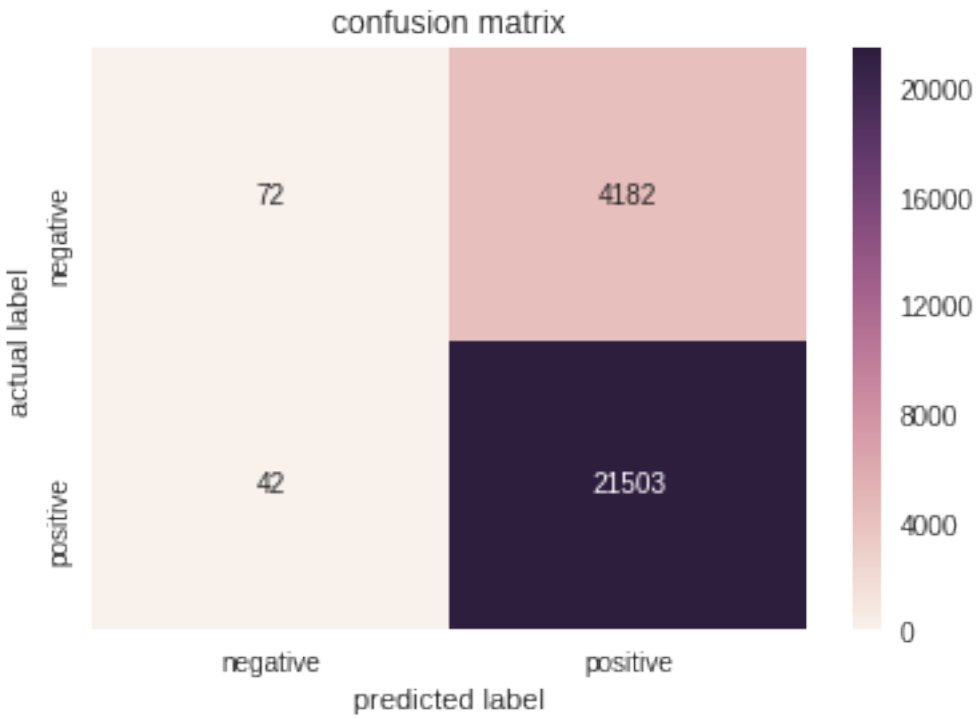

```
In [50]: #plotting confusion matrix aftyer performing knn on top of svd data
         from sklearn.metrics import confusion_matrix
         rest=confusion_matrix(ytest,predic)
         import seaborn as sns
         classlabel=['negative','positive']
```

```
frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



In [85]:
```
print('train confusion matrix')
rest=confusion_matrix(ytrain,predictrain)
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```

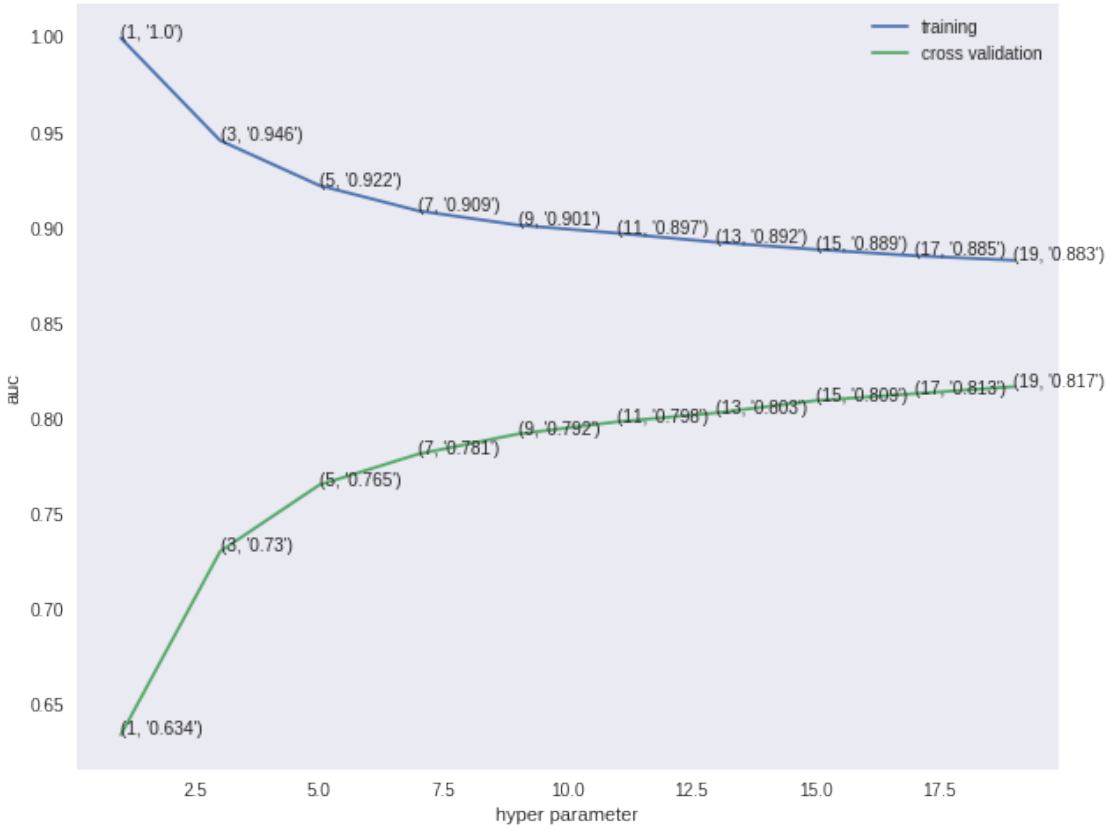
```
train confusion matrix
```



6.1.4 [5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

```
In [51]: #with hyper parameter tuning
         from sklearn.metrics import auc
         from sklearn.metrics import roc_auc_score
         from sklearn.neighbors import KNeighborsClassifier
         cvscores=[]
         cvscores1=[]
         alpha=[i for i in range(1,20,2)]
         for i in alpha:
             knnx=KNeighborsClassifier(n_neighbors=i,algorithm='brute')
             knnx.fit( xtraintfidf_sent_vectors,ytrain)
             predict1=knnx.predict_proba( xtraintfidf_sent_vectors)[:,1]
             predictz1=knnx.predict(xtraintfidf_sent_vectors)
             cvscores.append(roc_auc_score(ytrain,predict1))
             predict2=knnx.predict_proba( xcvtfidf_sent_vectors)[:,1]
             cvscores1.append(roc_auc_score(ycv,predict2))
             optimal_k=np.argmax(cvscores1)
         fig,ax=plt.subplots(figsize=(10,8))
         ax.plot(alpha,cvscores,label='training')
         for i,txt in enumerate(np.round(cvscores,3)):
           ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores[i]))
         plt.grid()
         ax.legend()
```

```
ax.plot(alpha,cvscores1,label='cross validation')
for i,txt in enumerate(np.round(cvscores1,3)):
  ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores1[i]))
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
print(alpha[optimal_k])
print(cvscores1)
```




```
19
[0.6342368966025882, 0.730344397216315, 0.7652461982863585, 0.7814750203062941, 0.791944238889
```

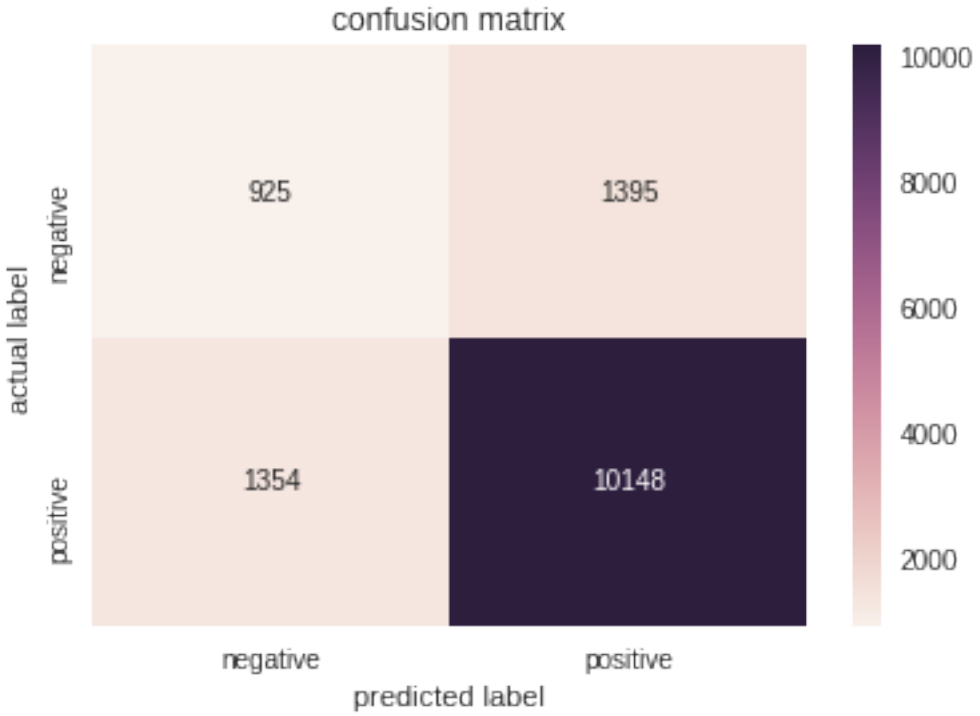
```
In [94]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import roc_auc_score
         knne=KNeighborsClassifier(n_neighbors=19,algorithm='brute')
         knne.fit( xtraintfidf_sent_vectors,ytrain)
         predictrain=knne.predict( xtraintfidf_sent_vectors)
```

```
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=knne.predict( xtesttfidf_sent_vectors)
fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
auc = metrics.roc_auc_score(ytest, predic)
plt.plot(fpr,tpr,label="roc of test)")
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
print(roc_auc_score(ytest, predic))
```

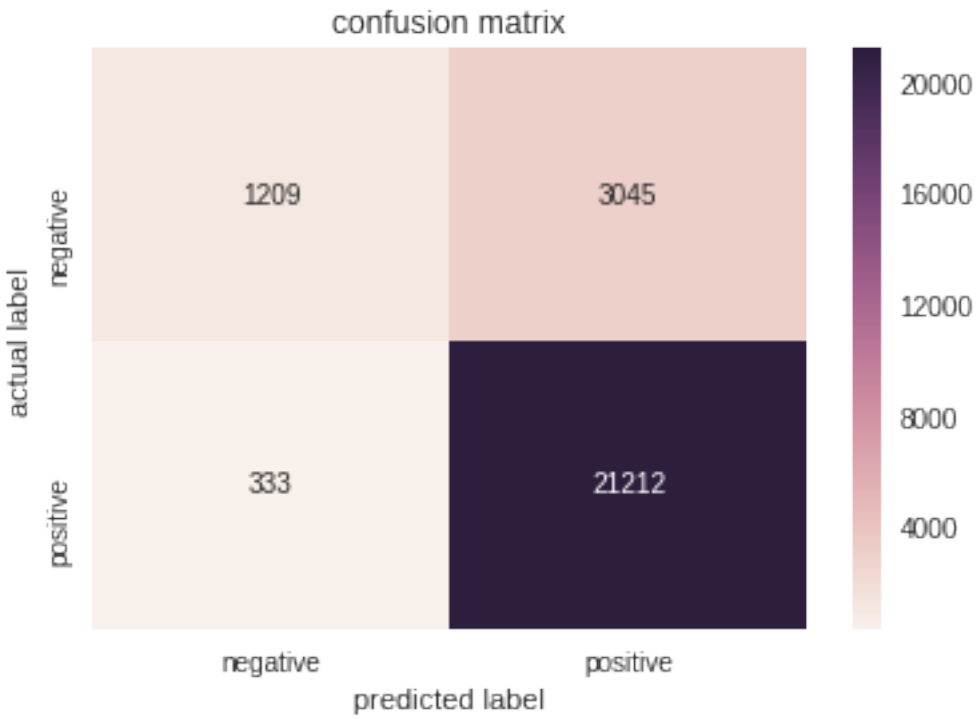
```
0.6098840466002701
```

In [53]:
```
#plotting confusion matrix aftyer performing knn on top of svd data
from sklearn.metrics import confusion_matrix
rest=confusion_matrix(ytest,predic)
import seaborn as sns
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
```

```
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



In [95]:
```
#plotting confusion matrix aftyer performing knn on top of svd data
from sklearn.metrics import confusion_matrix
rest=confusion_matrix(ytrain,predictrain)
import seaborn as sns
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



6.2 [5.2] Applying KNN kd-tree

6.2.1 [5.2.1] Applying KNN kd-tree on BOW, SET 5

In [36]:
```
print(ytrain.shape)
```

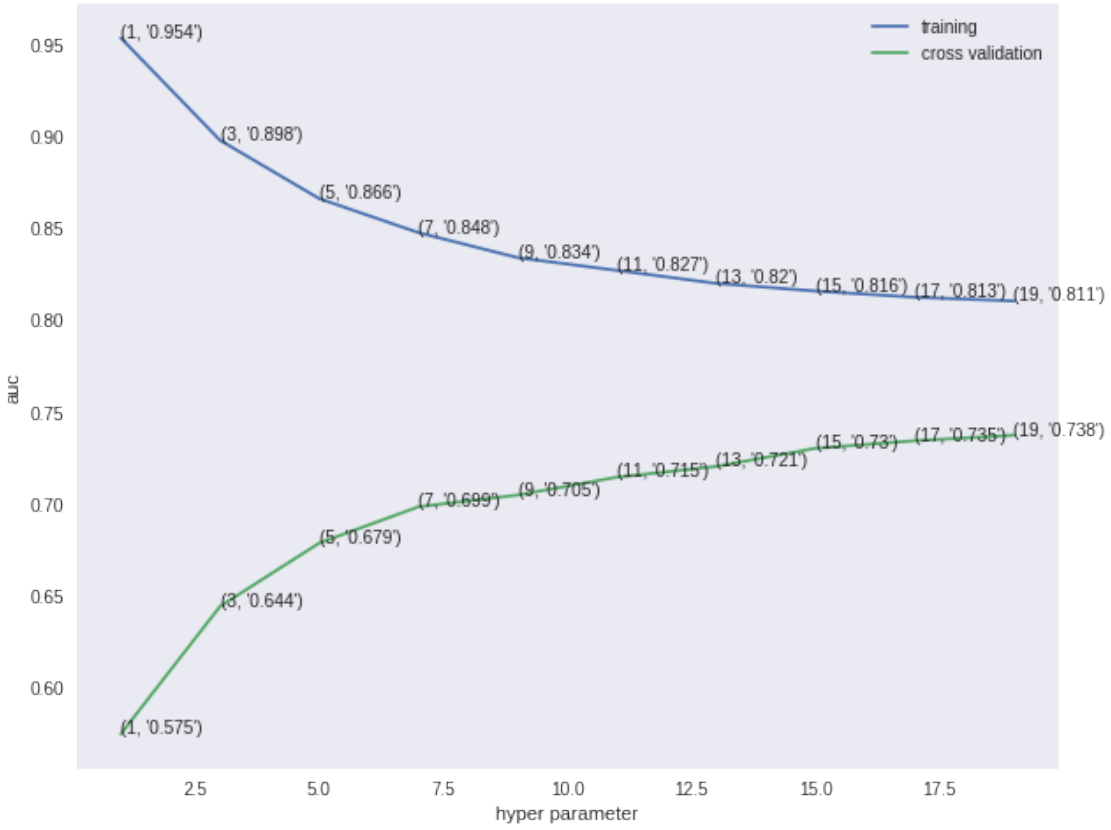
```
(25799,)
```

In [37]:
```
print(xtrainonehotencoding11.shape)
```

```
(25799, 50)
```

In [61]:
```
#@title Default title text
#with hyper parameter tuning
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
cvscores=[]
cvscores1=[]
alpha=[i for i in range(1,20,2)]
for i in alpha:
    knnx=KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    knnx.fit(xtrainonehotencoding11,ytrain)
```

```
    predict1=knnx.predict_proba(xtrainonehotencoding11)[:,1]
    cvscores.append(roc_auc_score(ytrain,predict1))
    predict2=knnx.predict_proba(xcvonehotencoding13)[:,1]
    cvscores1.append(roc_auc_score(ycv,predict2))
optimal_k=np.argmax(cvscores1)
fig,ax=plt.subplots(figsize=(10,8))
ax.plot(alpha,cvscores,label='training')
for i,txt in enumerate(np.round(cvscores,3)):
  ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores[i]))
plt.grid()
ax.legend()
ax.plot(alpha,cvscores1,label='cross validation')
for i,txt in enumerate(np.round(cvscores1,3)):
  ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores1[i]))
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
print(alpha[optimal_k])
print(cvscores1)
```

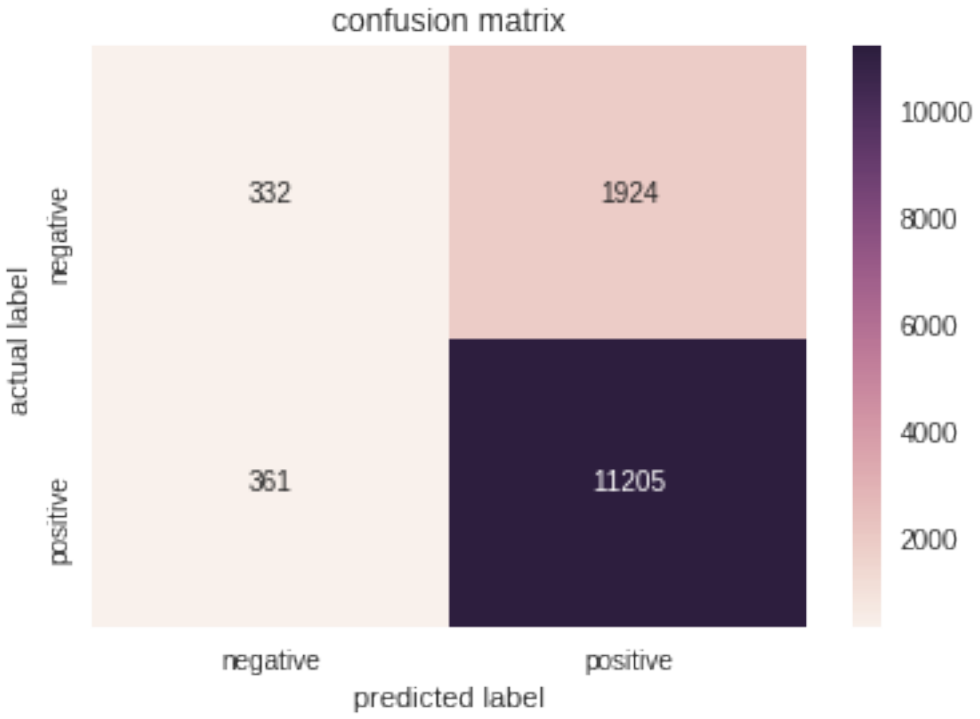


```
19
[0.5750669401202718, 0.6444938996338954, 0.6789287204795447, 0.6986661561219907, 0.70503183787
```

```
In [62]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import roc_auc_score
         knne=KNeighborsClassifier(n_neighbors=alpha[optimal_k],algorithm='kd_tree')
         knne.fit(xtrainonehotencoding11,ytrain)
         predictrain=knne.predict(xtrainonehotencoding11)
         fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
         auc = metrics.roc_auc_score(ytrain, predictrain)
         plt.plot(fpr,tpr,label="roc of train")
         plt.legend()
         predic=knne.predict(xtestonehotencoding12)
         fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
         auc = metrics.roc_auc_score(ytest, predic)
         plt.plot(fpr,tpr,label="roc of test)")
         plt.legend()
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('Receiver operating characteristic')
         print(roc_auc_score(ytest, predic))
```

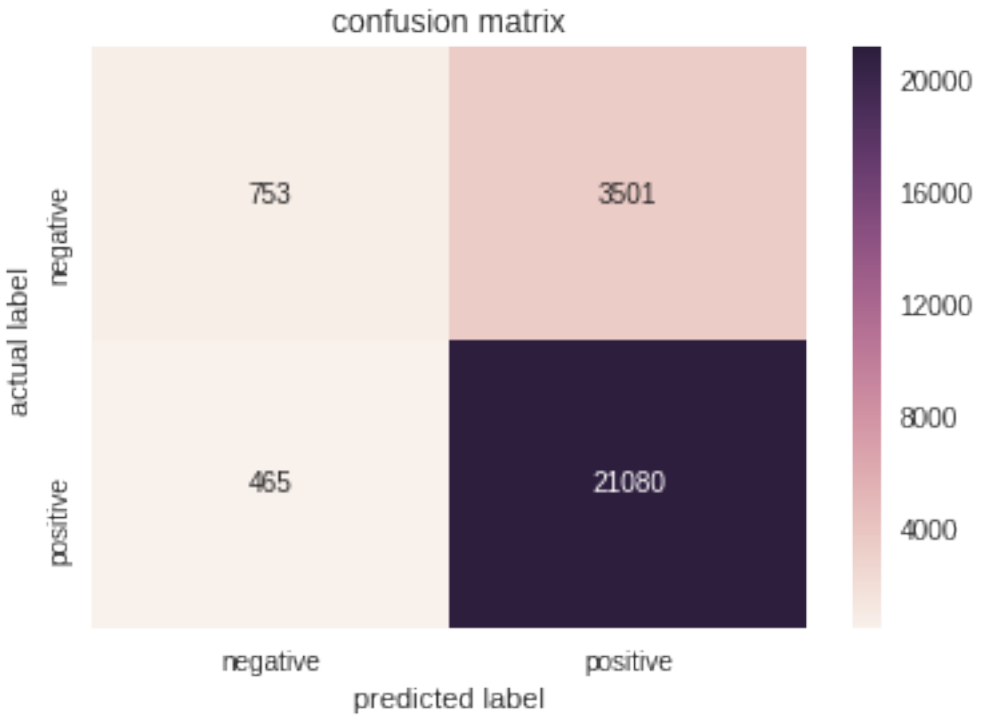
```
0.557975473477532
```

In [63]:
```
#plotting confusion matrix aftyer performing knn on top of svd data
from sklearn.metrics import confusion_matrix
rest=confusion_matrix(ytest,predic)
import seaborn as sns
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



In [64]:
```
print('train confusion matrix')
rest=confusion_matrix(ytrain,predictrain)
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
```

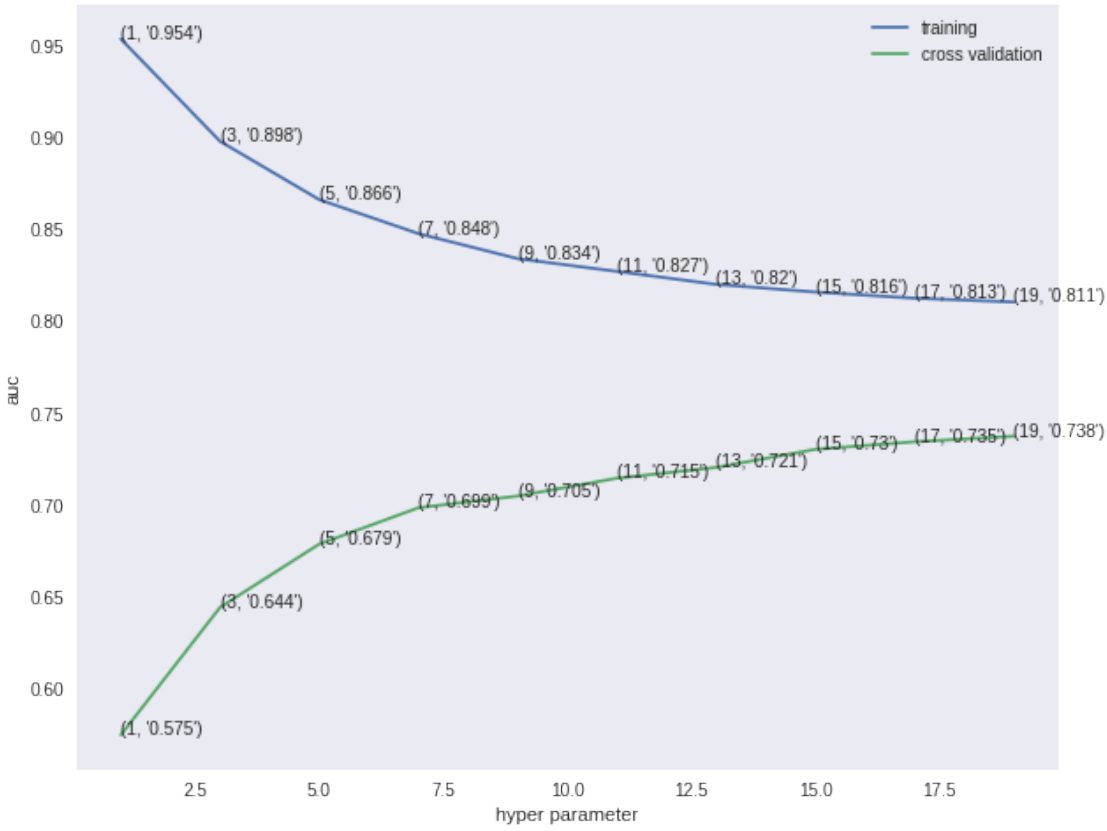
```
        plt.ylabel("actual label")
        plt.show()

train confusion matrix
```



6.2.2 [5.2.2] Applying KNN kd-tree on TFIDF, SET 6

```
In [52]: #with hyper parameter tuning
         from sklearn.metrics import auc
         from sklearn.metrics import roc_auc_score
         from sklearn.neighbors import KNeighborsClassifier
         cvscores=[]
         cvscores1=[]

         alpha=[i for i in range(1,20,2)]
         for i in alpha:
             knnx=KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
             knnx.fit(xtraintfidfencoding11,ytrain)
             predict1=knnx.predict_proba(xtraintfidfencoding11)[:,1]
             predix=knnx.predict(xtraintfidfencoding11)
             cvscores.append(roc_auc_score(ytrain,predict1))
             predict2=knnx.predict_proba(xcvtfidfencoding13)[:,1]
             cvscores1.append(roc_auc_score(ycv,predict2))
```

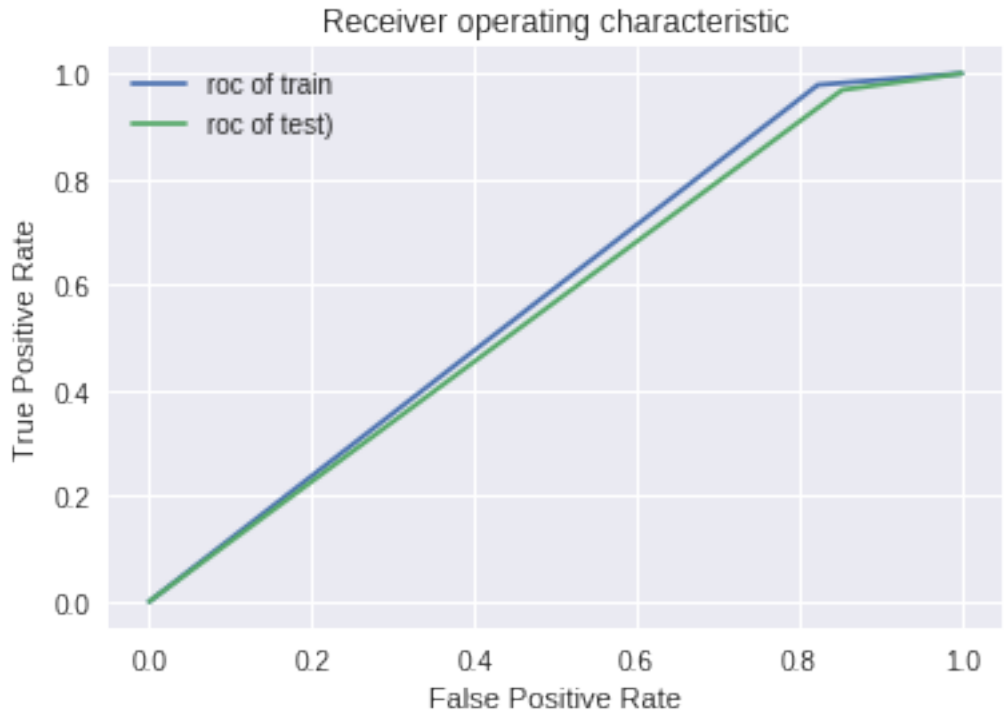
```
optimal_k=np.argmax(cvscores1)
fig,ax=plt.subplots(figsize=(10,8))
ax.plot(alpha,cvscores,label='training')
for i,txt in enumerate(np.round(cvscores,3)):
  ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores[i]))
plt.grid()
ax.legend()
ax.plot(alpha,cvscores1,label='cross validation')
for i,txt in enumerate(np.round(cvscores1,3)):
  ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores1[i]))
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
print(alpha[optimal_k])
print(cvscores1)
```




```
19
[0.5750669401202718, 0.6444938996338954, 0.6789287204795447, 0.6986661561219907, 0.70503183787
```

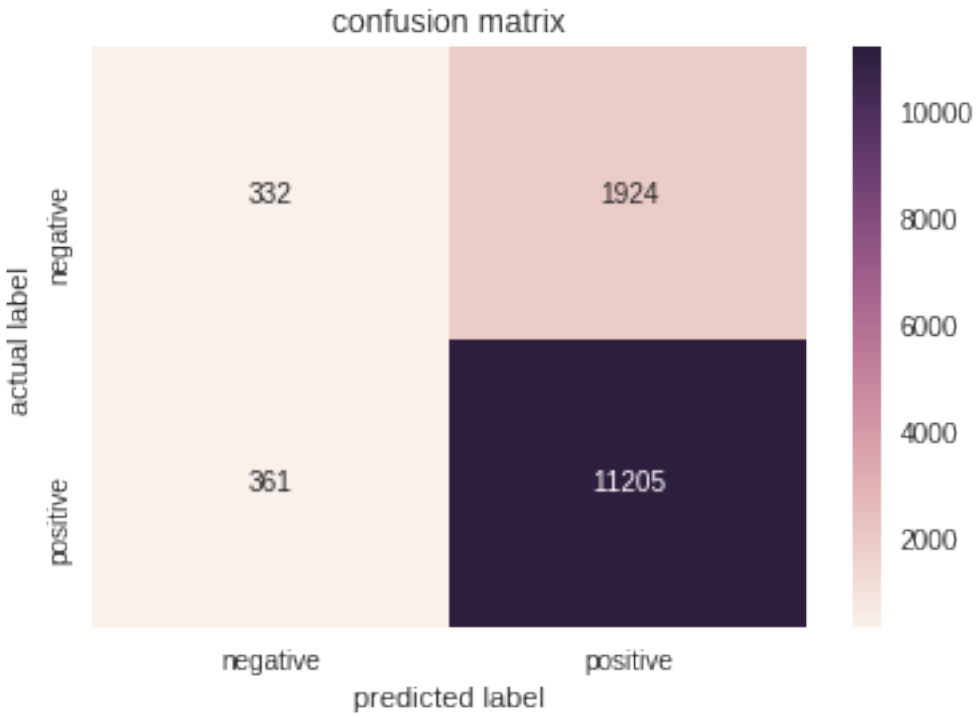
In [72]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
knne=KNeighborsClassifier(n_neighbors=19,algorithm='kd_tree')
knne.fit(xtraintfidfencoding11,ytrain)
predictrain=knne.predict(xtraintfidfencoding11)
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=knne.predict(xtesttfidfencoding12)
fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
auc = metrics.roc_auc_score(ytest, predic)
plt.plot(fpr,tpr,label="roc of test)")
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
print(roc_auc_score(ytest, predic))
```

```
0.557975473477532
```

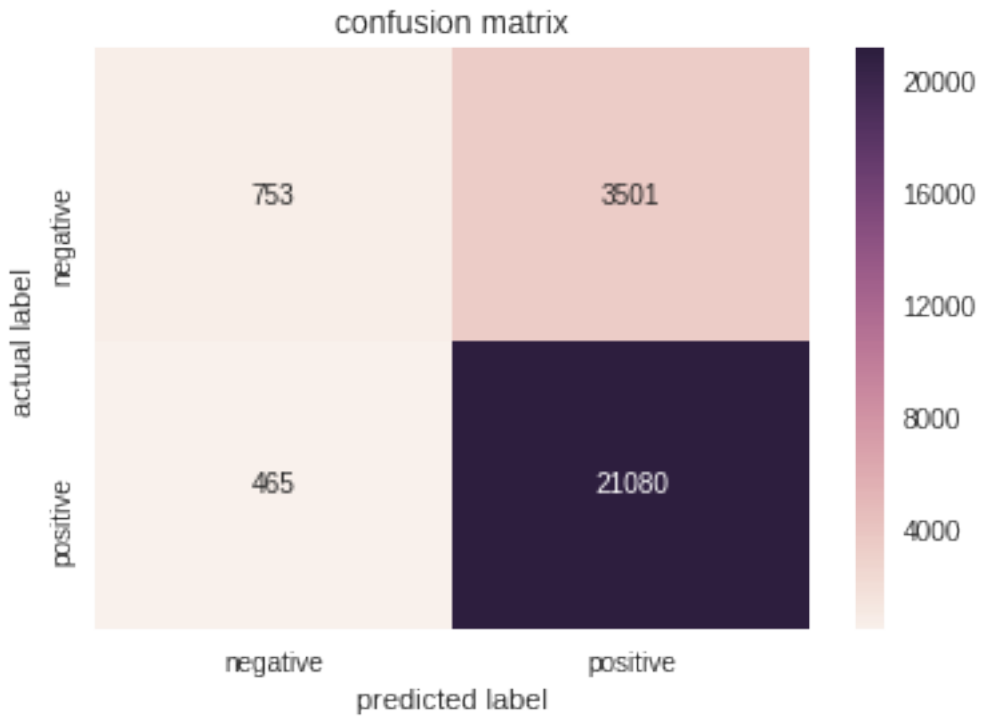


In [75]:
```
#plotting confusion matrix aftyer performing knn on top of svd data
from sklearn.metrics import confusion_matrix
```

```
rest=confusion_matrix(ytest,predic)
import seaborn as sns
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



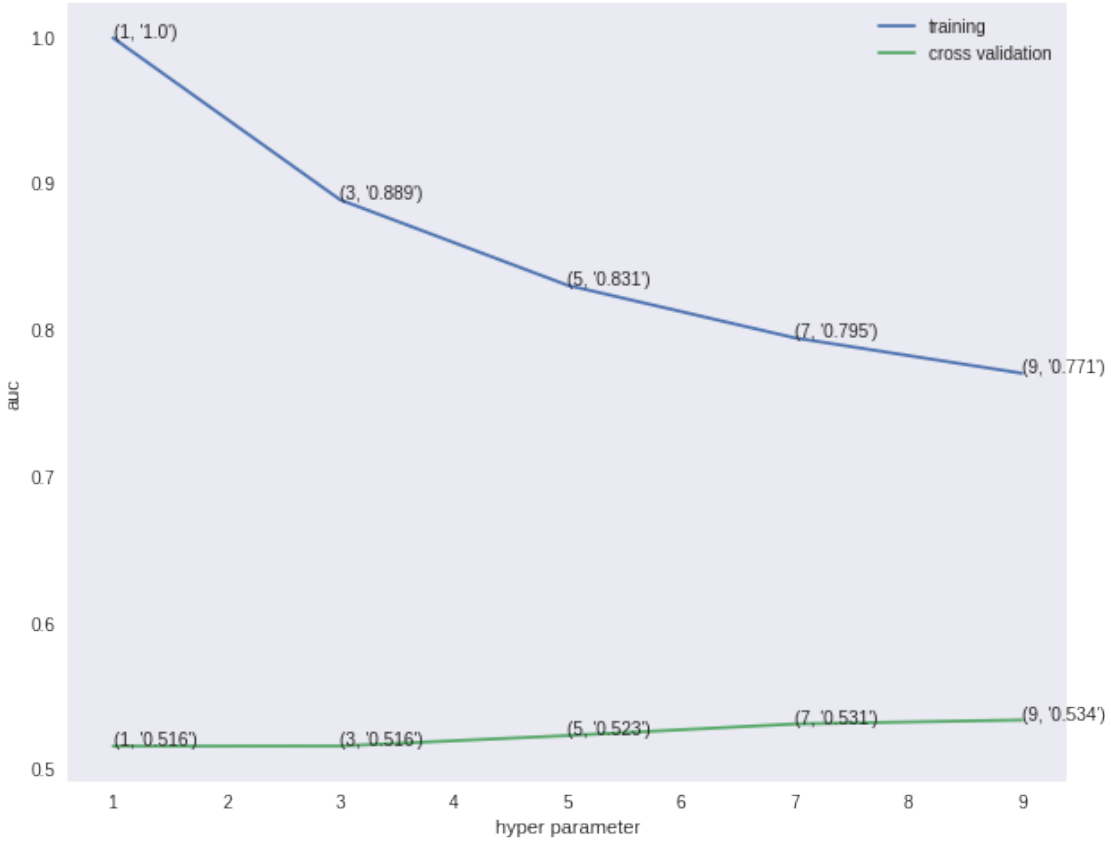
In [73]:
```
#plotting confusion matrix aftyer performing knn on top of svd data
from sklearn.metrics import confusion_matrix
rest=confusion_matrix(ytrain,predictrain)
import seaborn as sns
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



6.2.3 [5.2.3] Applying KNN kd-tree on AVG W2V, SET 7

```
In [54]: #with hyper parameter tuning
         from sklearn.metrics import auc
         from sklearn.metrics import roc_auc_score
         from sklearn.neighbors import KNeighborsClassifier
         cvscores=[]
         cvscores1=[]

         alpha=[i for i in range(1,10,2)]
         for i in alpha:
             knnx=KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
             knnx.fit(sent_vectors,ytrain)
             predict1=knnx.predict_proba(sent_vectors)[:,1]
             predicq=knnx.predict(sent_vectors)
             cvscores.append(roc_auc_score(ytrain,predict1))
             predict2=knnx.predict_proba(sent_vectorscv)[:,1]
             cvscores1.append(roc_auc_score(ycv,predict2))
             optimal_k=np.argmax(cvscores1)
         fig,ax=plt.subplots(figsize=(10,8))
         ax.plot(alpha,cvscores,label='training')
         for i,txt in enumerate(np.round(cvscores,3)):
           ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores[i]))
         plt.grid()
```

```
ax.legend()
ax.plot(alpha,cvscores1,label='cross validation')
for i,txt in enumerate(np.round(cvscores1,3)):
  ax.annotate((alpha[i],str(txt)), (alpha[i],cvscores1[i]))
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
print(alpha[optimal_k])
print(cvscores1)
```

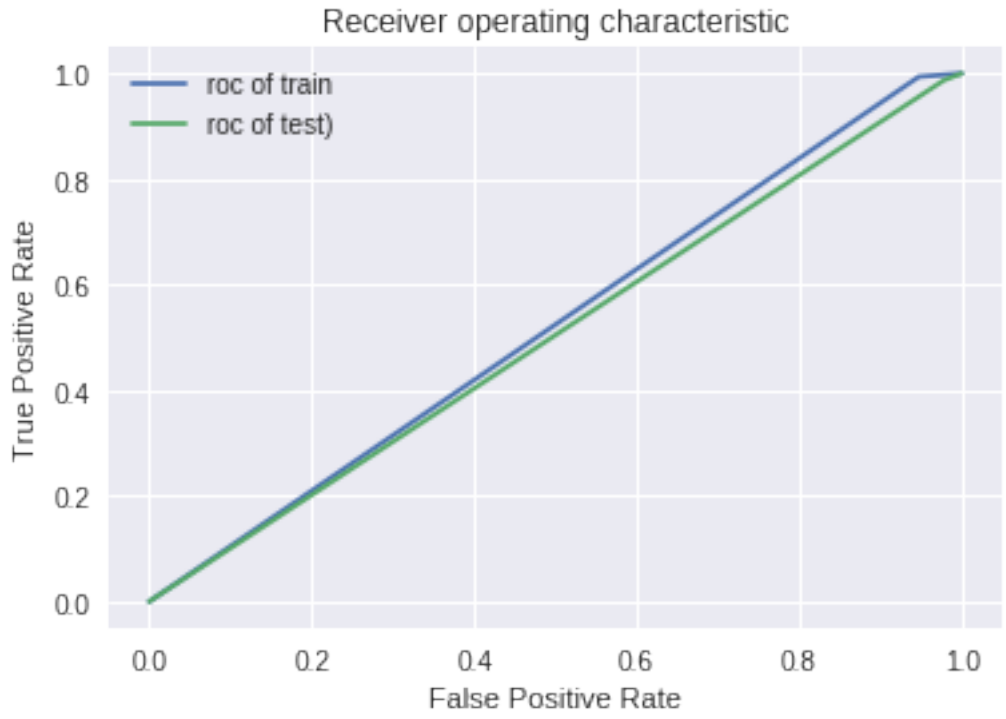



```
9
[0.515962359735011, 0.5160707422120141, 0.5231926164569716, 0.5310807460027573, 0.533819738616
```

```
In [86]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import roc_auc_score
         knne=KNeighborsClassifier(n_neighbors=9,algorithm='kd_tree')
         knne.fit(sent_vectors,ytrain)
```

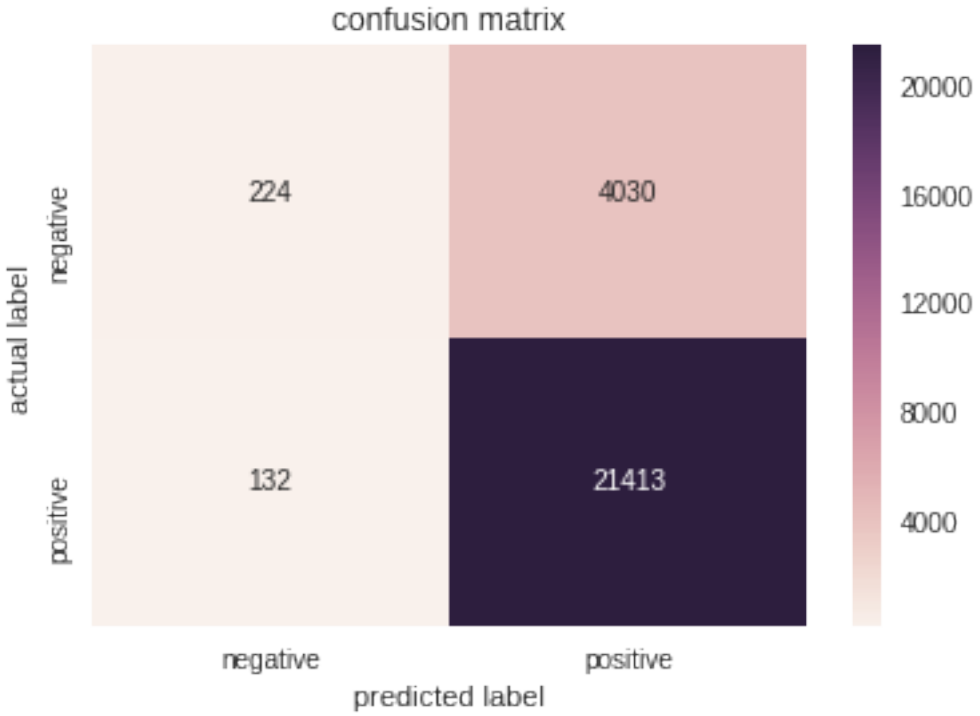
```
predictrain=knne.predict(sent_vectors)
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=knne.predict(sent_vectorstest)
fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
auc = metrics.roc_auc_score(ytest, predic)
plt.plot(fpr,tpr,label="roc of test)")
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
print(roc_auc_score(ytest, predic))
```

```
0.5045050959464216
```

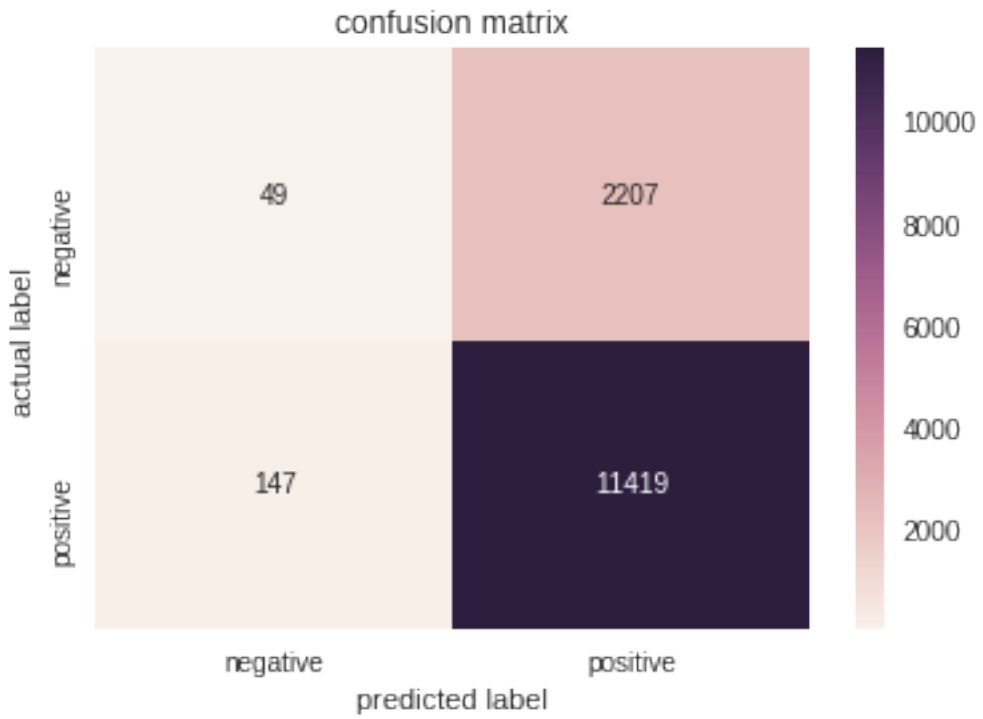


In [89]:
```
#plotting confusion matrix aftyer performing knn on top of svd data
from sklearn.metrics import confusion_matrix
rest=confusion_matrix(ytrain,predictrain)
import seaborn as sns
classlabel=['negative','positive']
```

```
frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



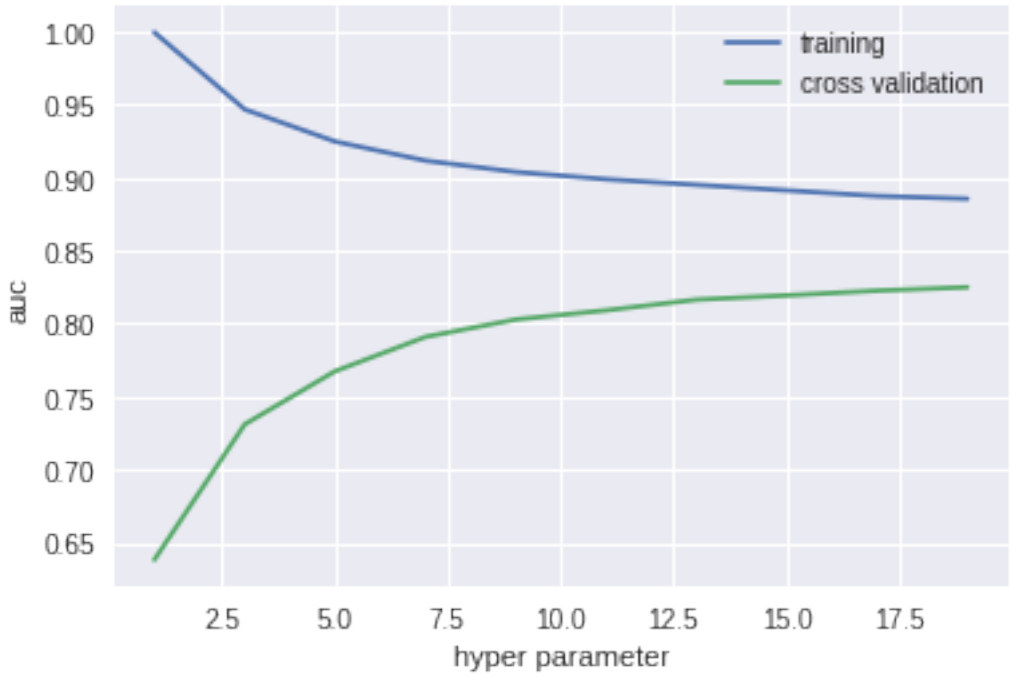
In [88]:
```
#plotting confusion matrix aftyer performing knn on top of svd data
from sklearn.metrics import confusion_matrix
rest=confusion_matrix(ytest,predic)
import seaborn as sns
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



6.2.4 [5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 8

```
In [96]: #with hyper parameter tuning
         from sklearn.metrics import auc
         from sklearn.metrics import roc_auc_score
         from sklearn.neighbors import KNeighborsClassifier
         cvscores=[]
         cvscores1=[]
         alpha=[i for i in range(1,20,2)]
         for i in alpha:
             knnx=KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
             knnx.fit( xtraintfidf_sent_vectors,ytrain)
             predict1=knnx.predict_proba( xtraintfidf_sent_vectors)[:,1]
             predictm=knnx.predict(sent_vectors)
             cvscores.append(roc_auc_score(ytrain,predict1))
             predict2=knnx.predict_proba( xcvtfidf_sent_vectors)[:,1]
             cvscores1.append(roc_auc_score(ycv,predict2))
             optimal_k=np.argmax(cvscores1)
         fig,ax=plt.subplots()
         ax.plot(alpha,cvscores,label='training')
         ax.legend()
         ax.plot(alpha,cvscores1,label='cross validation')
         ax.legend()
         plt.xlabel('hyper parameter')
```

```
plt.ylabel('auc')
plt.show()
print(cvscores1)
optimal_k=np.argmax(cvscores1)
optimal_k=alpha[optimal_k]
print(optimal_k)

print(cvscores1)
```

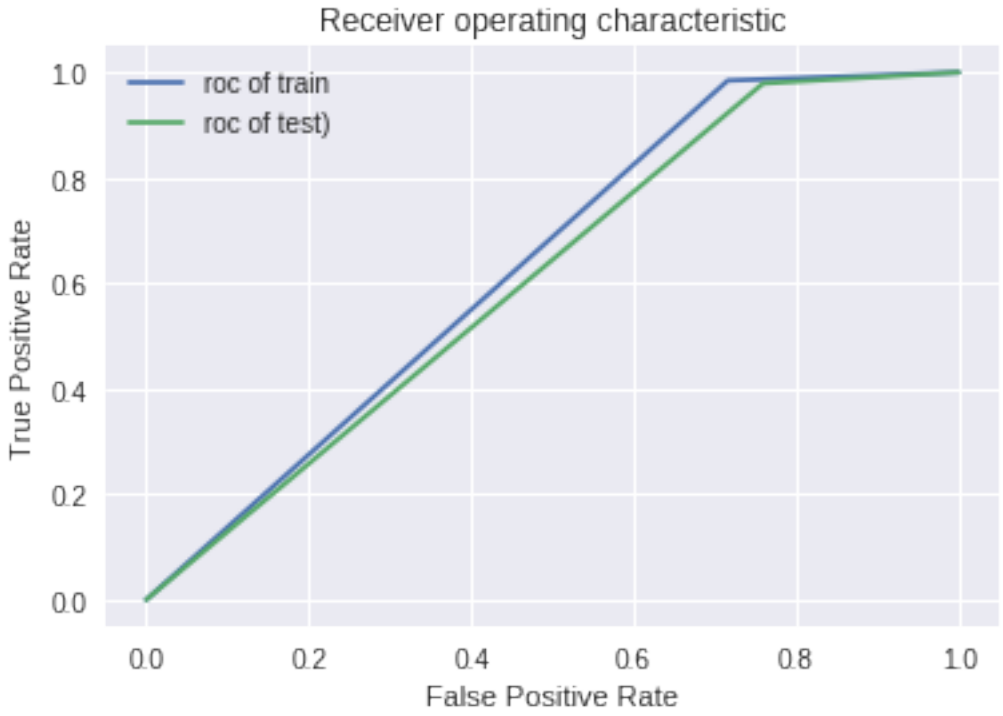



```
[0.6382805892934373, 0.7310453402238499, 0.7674201849255257, 0.790928081380454, 0.802872450834
19
[0.6382805892934373, 0.7310453402238499, 0.7674201849255257, 0.790928081380454, 0.802872450834
```

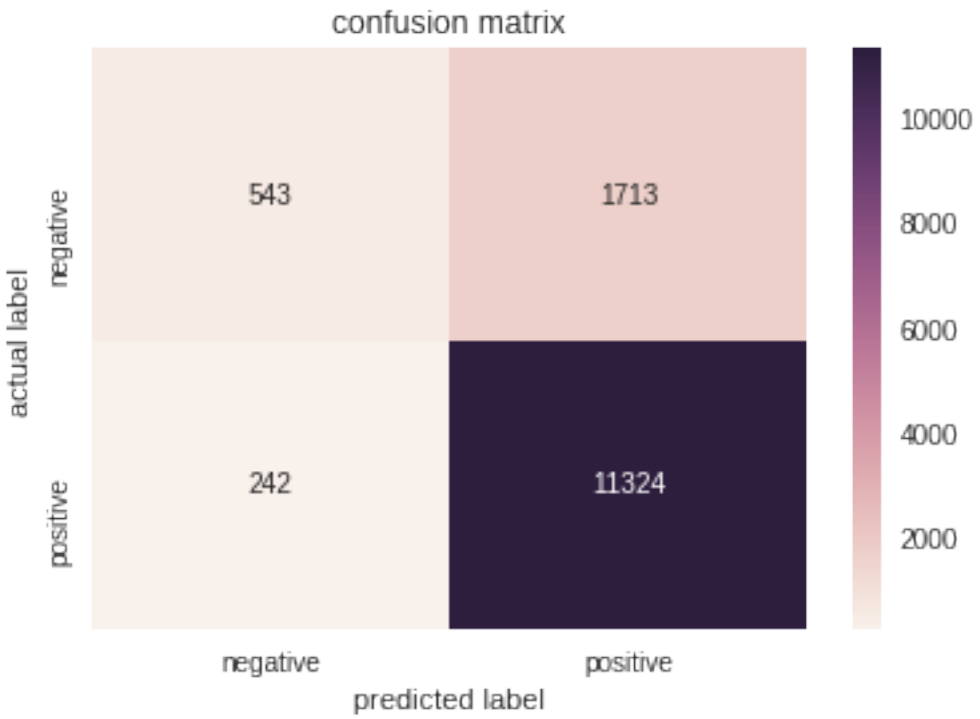
In [98]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
knne=KNeighborsClassifier(n_neighbors=19,algorithm='kd_tree')
knne.fit( xtraintfidf_sent_vectors,ytrain)
predictrain=knne.predict( xtraintfidf_sent_vectors)
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=knne.predict( xtesttfidf_sent_vectors)
```

```
fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
auc = metrics.roc_auc_score(ytest, predic)
plt.plot(fpr,tpr,label="roc of test)")
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
print(roc_auc_score(ytest, predic))
```

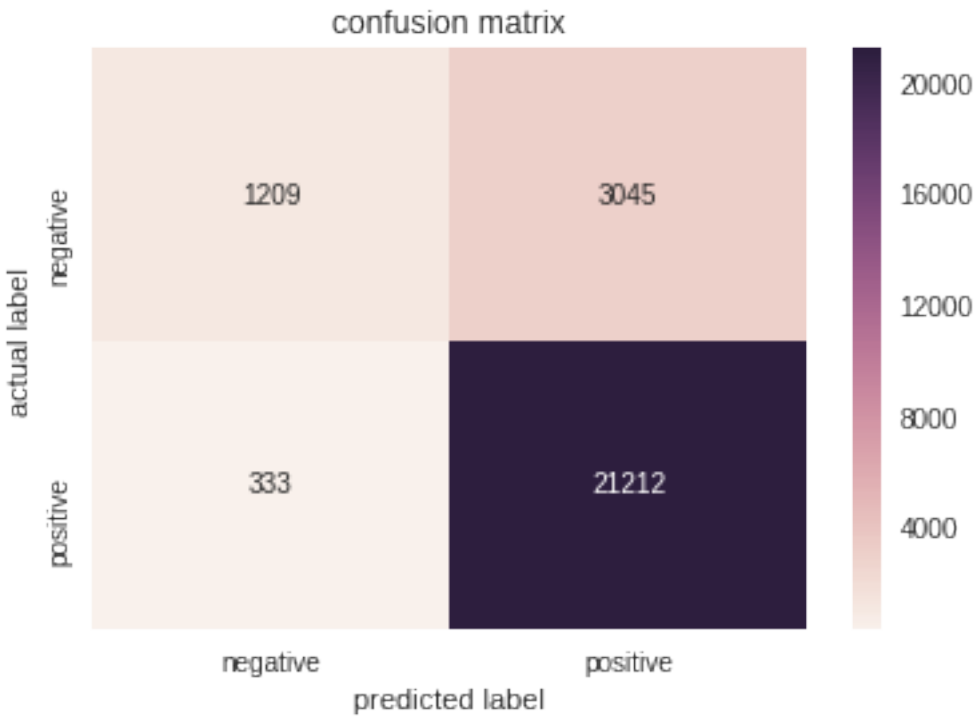
```
0.6098840466002701
```



In [99]:
```
#plotting confusion matrix aftyer performing knn on top of svd data
from sklearn.metrics import confusion_matrix
rest=confusion_matrix(ytest,predic)
import seaborn as sns
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



In [100]:
```
#plotting confusion matrix aftyer performing knn on top of svd data
from sklearn.metrics import confusion_matrix
rest=confusion_matrix(ytrain,predictrain)
import seaborn as sns
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



7 [6] Conclusions

```
In [108]: data = [['brute',13, 0.555], ['brute',9, 0.5],['brute',15, 0.502],['brute',19, 0.65]
          pd.DataFrame(data, columns=["model", "hyperparameter",'auc'],index=['bow','tfidf','w
```

Out[108]:

	model	hyperparameter	auc
bow	brute	13	0.555
tfidf	brute	9	0.500
word2vec	brute	15	0.502
averageword2vec	brute	19	0.650
bow	kd_tree	19	0.550
tfidf	kd_tree	19	0.550
word2vec	kd_tree	9	0.549
averageword2vec	kd_tree	19	0.650