# Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (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]:
```
# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
    100% |████████████████████████████████| 993kB 6.6MB/s ta 0:00:011
  Building wheel for PyDrive (setup.py) ... done
```

In [0]:
```
link = 'https://drive.google.com/open?id=1rPupavaoY_D5F-DDZLHatHqH8MCal0yo' # Th
```

In [3]:
```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
1rPupavaoY_D5F-DDZLHatHqH8MCal0yo
```

In [0]:
```
import pandas as pd
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('mydata.csv')
df3 = pd.read_csv('mydata.csv')
```

In [48]:
```
df3.shape
```

Out[48]:
```
(100000, 11)
```

In [0]:
```
filtered_data = df3
```

In [0]:

```
%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 [51]:

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

#filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIM

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a ne
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, 11)
```

Out[51]:

| | Unnamed: 0 | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | 1 |
| **1** | 1 | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | 0 |
| **2** | 2 | 3 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia Corres" | 1 | 1 |

In [0]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

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

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

# [2] Exploratory Data Analysis

## [2.1] Data Cleaning: Deduplication

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

In [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, inpla
```

In [53]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},
final.shape
```

Out[53]:
```
(87775, 11)
```

In [54]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[54]:
```
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 [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

In [0]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [57]:
```
#Before starting the next phase of preprocessing lets see the number of entries l
print(final.shape)

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

```
(87773, 11)
```

Out[57]:
```
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 [58]:
```
# 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 i
t anymore.  Its very hard to find any chicken products made in the USA but they
are out there, but this one isnt.  Its too bad too because its a good product b
ut I wont take any chances till they know what is going on with the china impor
ts.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy h
as 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 i
f 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 ove
r eating. Amazon's price was much more reasonable than any other retailer. You
can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at o
ther retailers. It's definitely worth it to buy a big bag if your dog eats them
a lot.
==================================================
```

In [59]:
```
# 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 i
t anymore.  Its very hard to find any chicken products made in the USA but they
are out there, but this one isnt.  Its too bad too because its a good product b
ut I wont take any chances till they know what is going on with the china impor
ts.
```

In [60]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove
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 i
t anymore.  Its very hard to find any chicken products made in the USA but they
are out there, but this one isnt.  Its too bad too because its a good product b
ut I wont take any chances till they know what is going on with the china impor
ts.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy h
as 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 i
f 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 ove
r eating. Amazon's price was much more reasonable than any other retailer. You
can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at o
ther retailers. It's definitely worth it to buy a big bag if your dog eats them
a lot.
```

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 [62]:
```
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 [63]:
```
#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 i
t anymore.  Its very hard to find any chicken products made in the USA but they
are out there, but this one isnt.  Its too bad too because its a good product b
ut I wont take any chances till they know what is going on with the china impor
ts.
```

In [64]:
```
#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 [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 st

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ou
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 't
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than'
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven','not','no', "haven't", 'isn', "is
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn
            'won', "won't", 'wouldn', "wouldn't"])
```

In [66]:
```
# 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
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:34<00:00, 2543.51it/s]
```

In [67]:
```
preprocessed_reviews[1500]
```

Out[67]:
```
'way hot blood took bite jig lol'
```

In [0]:
```
x=preprocessed_reviews
y=final['Score']
```

In [76]:
```
print(len(x))
print(len(y))
```

```
87773
87773
```

## [3.2] Preprocessing Review Summary

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 [85]:
```
print(len(xtrain))
print(ytrain.shape)
print(len(xtest))
print(ytest.shape)
print(len(xcv))
print(ycv.shape)
```

```
49152
(49152,)
26332
(26332,)
12289
(12289,)
```

In [0]:
```
wordslengthtrain=[len(mike.split()) for mike in xtrain]
wordslengthtest=[len(mike.split()) for mike in xtest]
wordlenghtcv=[len(mike.split()) for mike in xcv]
```

In [87]:
```
print(wordslengthtrain)
wordy=pd.DataFrame(wordslengthtrain)
wordy1=pd.DataFrame(wordslengthtest)
wordy2=pd.DataFrame(wordlenghtcv)
```

```
[11, 15, 46, 28, 12, 18, 125, 151, 14, 25, 12, 17, 20, 20, 55, 20, 16, 281, 6
6, 34, 33, 44, 12, 26, 10, 75, 13, 53, 10, 32, 12, 15, 16, 58, 31, 20, 27, 3
2, 54, 33, 43, 21, 18, 99, 25, 10, 33, 29, 33, 26, 29, 110, 82, 9, 116, 10, 2
4, 58, 13, 38, 29, 35, 45, 29, 31, 11, 39, 123, 17, 40, 40, 109, 57, 14, 36,
25, 76, 19, 33, 12, 16, 34, 15, 11, 17, 29, 21, 21, 42, 52, 36, 90, 0, 19, 2
7, 18, 53, 115, 11, 15, 27, 11, 11, 17, 17, 17, 94, 35, 59, 8, 97, 81, 37, 7
6, 42, 99, 52, 47, 14, 26, 17, 12, 10, 62, 38, 43, 54, 10, 52, 67, 9, 24, 38,
14, 32, 27, 38, 23, 12, 26, 15, 12, 24, 13, 62, 30, 15, 41, 35, 41, 133, 12,
16, 177, 71, 27, 27, 50, 35, 13, 11, 27, 86, 10, 19, 75, 40, 97, 100, 22, 18,
28, 74, 43, 29, 52, 36, 11, 16, 23, 26, 107, 12, 24, 28, 20, 45, 26, 28, 42,
12, 18, 21, 10, 57, 35, 52, 25, 31, 36, 34, 17, 88, 125, 52, 14, 22, 23, 29,
24, 90, 28, 84, 52, 12, 14, 24, 19, 75, 17, 15, 27, 19, 23, 8, 21, 16, 13, 3
3, 34, 77, 16, 49, 22, 57, 51, 13, 118, 125, 18, 124, 163, 41, 25, 20, 28, 1
6, 21, 50, 23, 8, 31, 20, 38, 11, 42, 72, 33, 168, 42, 47, 28, 51, 41, 17, 5
4, 85, 15, 115, 26, 70, 14, 27, 33, 30, 30, 37, 18, 38, 42, 17, 14, 25, 26, 2
0, 36, 51, 65, 31, 69, 86, 182, 12, 18, 10, 14, 39, 46, 22, 9, 11, 26, 55, 1
4, 7, 19, 21, 11, 18, 12, 20, 15, 57, 57, 28, 16, 11, 20, 31, 127, 405, 45, 3
8, 15, 49, 14, 106, 17, 74, 23, 9, 32, 5, 25, 10, 18, 22, 47, 70, 6, 165, 21,
29, 87, 18, 47, 101, 37, 10, 43, 98, 11, 24, 18, 32, 14, 34, 11, 32, 41, 26,
```

# [4] Featurization

## [4.1] BAG OF WORDS

In [185]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vect=CountVectorizer(min_df=2)
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)
```

```
(49152, 21730)
(26332, 21730)
(12289, 21730)
```

In [0]:
```
from scipy.sparse import hstack
xtrainonehotencoding1=hstack((xtrainonehotencoding,wordy))
xtestonehotencoding1=hstack((xtestonehotencoding,wordy1))
xcvonehotencoding1=hstack((xcvonehotencoding,wordy2))
```

In [132]:
```
print(xtrainonehotencoding1.shape)
print(xtestonehotencoding1.shape)
print(xcvonehotencoding1.shape)
```

```
(49152, 41229)
(26332, 41229)
(12289, 41229)
```

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

In [91]:
```
#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

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect1= CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect1.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_bi
```

```
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 [92]:
```
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)
```

```
(49152, 41228)
(26332, 41228)
(12289, 41228)
```

In [0]:
```
from scipy.sparse import hstack
xtraintfidfencoding1=hstack((xtraintfidfencoding,wordy))
xtesttfidfencoding1=hstack((xtesttfidfencoding,wordy1))
xcvtfidfencoding1=hstack((xcvtfidfencoding,wordy2))
```

In [94]:
```
print(xtraintfidfencoding1.shape)
print(xtesttfidfencoding1.shape)
print(xcvtfidfencoding1.shape)
```

```
(49152, 41229)
(26332, 41229)
(12289, 41229)
```

# [5] Assignment 4: Apply Naive Bayes

1. **Apply Multinomial NaiveBayes 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)

2. **The hyper paramter tuning(find best Alpha)**
    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
    - 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**
    - Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using values of `feature_log_prob_` parameter of MultinomialNB (https://scikit-

learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names

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. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.
      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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
      (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

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 (http://zetcode.com/python/prettytable/)

**Note: Data Leakage**

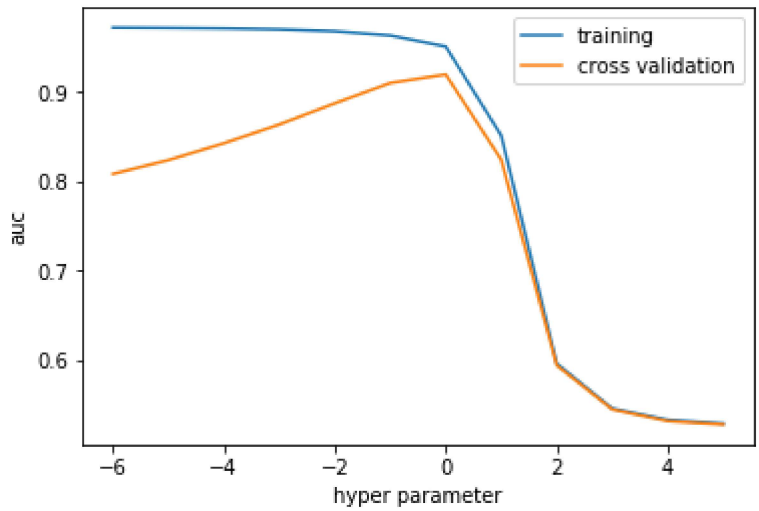
1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply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

# Applying Multinomial Naive Bayes

## [5.1] Applying Naive Bayes on BOW, SET 1

In [200]:
```
#with hyper parameter tuning
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
cvscores=[]
cvscores1=[]

alpha=[10**i for i in range(-6,6,1)]
for i in alpha:
    nbx=MultinomialNB(alpha=i,class_prior=[0.5,0.5])
    nbx.fit(xtrainonehotencoding,ytrain)
    predict1=nbx.predict_proba(xtrainonehotencoding)[:,1]
    cvscores.append(roc_auc_score(ytrain,predict1))
    predict2=nbx.predict_proba(xcvonehotencoding)[:,1]
    cvscores1.append(roc_auc_score(ycv,predict2))
    optimal_k=np.argmax(cvscores1)
fig,ax=plt.subplots()
ax.plot(np.log10(alpha),cvscores,label='training')
ax.legend()
ax.plot(np.log10(alpha),cvscores1,label='cross validation')
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
print(cvscores1)
optimal_k=np.argmax(cvscores1)
print(optimal_k)
print(alpha)
print(alpha[optimal_k])
print('optimum alpha value is ',alpha[optimal_k])
print(cvscores1)
```




```
[0.808111488393118, 0.823470096760744, 0.8423423469540978, 0.8634281712652299,
0.8871274161421835, 0.9098666931492114, 0.9193733514225886, 0.82397025218481,
0.5937298772637408, 0.5444419710501713, 0.5313975139169869, 0.5276286265777883]
6
[1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
1
optimum alpha value is  1
[0.808111488393118, 0.823470096760744, 0.8423423469540978, 0.8634281712652299,
```

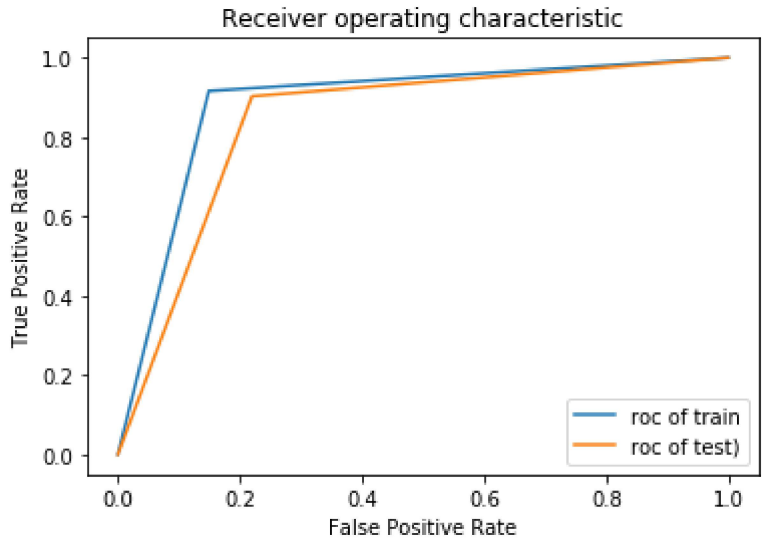
```
0.8871274161421835, 0.9098666931492114, 0.9193733514225886, 0.82397025218481,
0.5937298772637408, 0.5444419710501713, 0.5313975139169869, 0.5276286265777883]
```

In [201]:
```
print(alpha[optimal_k])
```

```
1
```

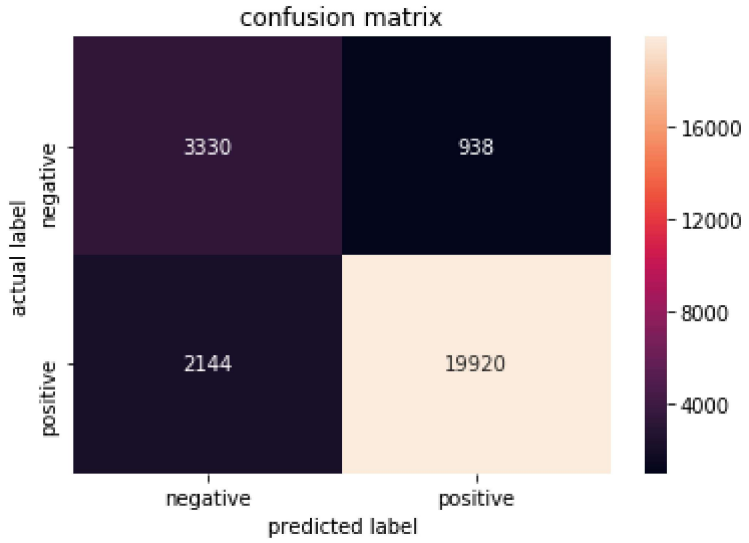
In [202]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
nbx=MultinomialNB(alpha=alpha[optimal_k],class_prior=[0.5,0.5])
nbx.fit(xtrainonehotencoding,ytrain)
predictrain=nbx.predict(xtrainonehotencoding)
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=nbx.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))
```

```
0.84152653302007
```



In [203]:
```
#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()
```




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

In [0]:
```
a=nbx.coef_
```

In [205]:
```
print(a.shape)
li=list(a)
```

```
(1, 21730)
```

In [0]:
```
x=np.argsort(li,axis=None)
feat=count_vect.get_feature_names()
```

In [207]:
```
print(feat)
print(len(feat))
```

```
['aa', 'aaa', 'aaaa', 'aafco', 'aahs', 'ab', 'aback', 'abandon', 'abandoned',
'abbott', 'abby', 'abc', 'abd', 'abdomen', 'abdominal', 'abide', 'abilities',
'ability', 'abit', 'ablaze', 'able', 'ablution', 'abnormal', 'abnormalities',
'abnormally', 'abounds', 'abowl', 'abrasive', 'abroad', 'abruptly', 'abs', 'a
bsence', 'absent', 'absolut', 'absolute', 'absolutely', 'absolutley', 'absolu
tly', 'absorb', 'absorbed', 'absorber', 'absorbing', 'absorbs', 'absorption',
'absoulte', 'absoultely', 'absoutly', 'absurd', 'absurdly', 'abuelita', 'abun
dance', 'abundant', 'abuse', 'abused', 'abut', 'abv', 'abysmal', 'ac', 'acaci
a', 'academy', 'acai', 'acana', 'accent', 'accented', 'accents', 'accentuat
e', 'accept', 'acceptable', 'acceptance', 'accepted', 'accepting', 'accepts',
'access', 'accessible', 'accessories', 'accessory', 'accident', 'accidental',
'accidentally', 'accidently', 'accidents', 'acclaimed', 'acclimate', 'acclima
ted', 'accolades', 'accommodate', 'accommodating', 'accomodate', 'accompanie
d', 'accompanies', 'accompaniment', 'accompany', 'accompanying', 'accomplis
h', 'accomplished', 'accomplishes', 'accomplishing', 'accomplishment', 'accor
dance', 'according', 'accordingly', 'account', 'accountable', 'accountant',
'accounting', 'accounts', 'accoutrements', 'accross', 'accumulate', 'accumula
ted', 'accumulating', 'accuracy', 'accurate', 'accurately', 'accusatory', 'ac
cused', 'accustom', 'accustomed', 'ace', 'acerola', 'acess', 'acesulfame', 'a
```

In [208]:
```
dd=np.argsort(a,axis=None)[::-1]
k=[feat[i] for i in dd]
print('top positive words arre',k[:10])
```

```
top positive words arre ['like', 'good', 'great', 'one', 'taste', 'coffee', 'lo
ve', 'flavor', 'would', 'product']
```

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

In [209]:
```
p=[feat[i] for i in x]
print('top negative words are',p[:10])
```

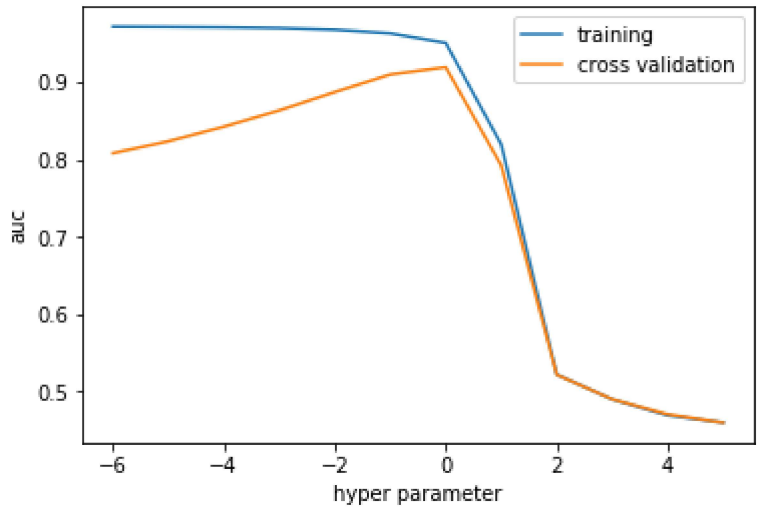
```
top negative words are ['milka', 'alka', 'whirlypop', 'silicones', 'unsatisfact
ory', 'microbial', 'tbhq', 'carmine', 'disapointment', 'horrors']
```

In [190]:
```
#with hyper parameter tuning
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score

cvscores=[]
cvscores1=[]

alpha=[10**i for i in range(-6,6,1)]
for i in alpha:
    nbx=MultinomialNB(alpha=i,class_prior=[0.5,0.5])
    nbx.fit(xtrainonehotencoding1,ytrain)
    predict1=nbx.predict_proba(xtrainonehotencoding1)[:,1]
    cvscores.append(roc_auc_score(ytrain,predict1))
    predict2=nbx.predict_proba(xcvonehotencoding1)[:,1]
    cvscores1.append(roc_auc_score(ycv,predict2))
    optimal_k=np.argmax(cvscores1)
fig,ax=plt.subplots()
ax.plot(np.log10(alpha),cvscores,label='training')
ax.legend()
ax.plot(np.log10(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)
```

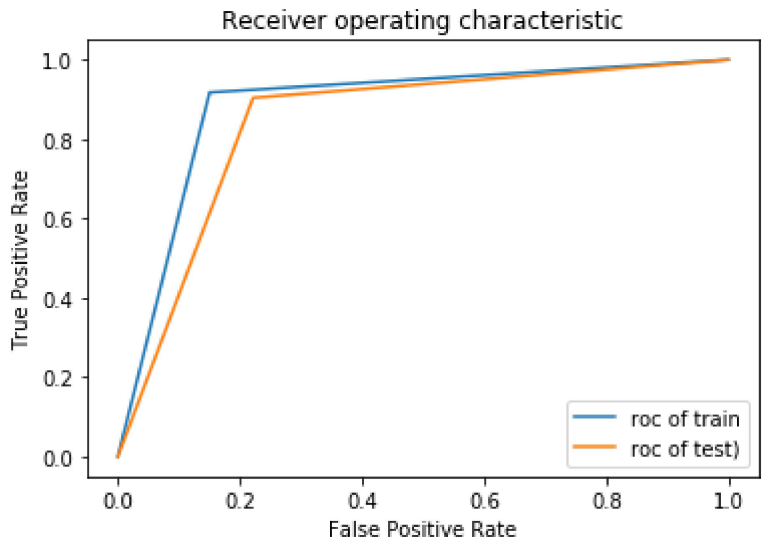



```
[0.8081313064241038, 0.823489500392558, 0.8423331326666285, 0.8634217115134327,
0.887122199587902, 0.9098596971161328, 0.9191980849492951, 0.7922852183717877,
0.5217141997142694, 0.4900607333675774, 0.4705393390596473, 0.459785363655262]
1
[0.8081313064241038, 0.823489500392558, 0.8423331326666285, 0.8634217115134327,
0.887122199587902, 0.9098596971161328, 0.9191980849492951, 0.7922852183717877,
0.5217141997142694, 0.4900607333675774, 0.4705393390596473, 0.459785363655262]
```

In [191]:

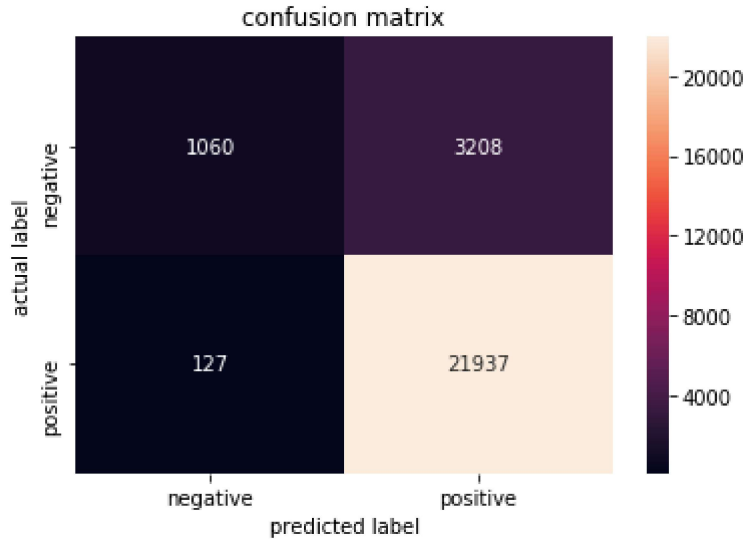
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
nbx=MultinomialNB(alpha=1,class_prior=[0.5,0.5])
nbx.fit(xtrainonehotencoding1,ytrain)
predictrain=nbx.predict(xtrainonehotencoding1)
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=nbx.predict(xtestonehotencoding1)
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))
```

```
0.8412238436637934
```



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()
```



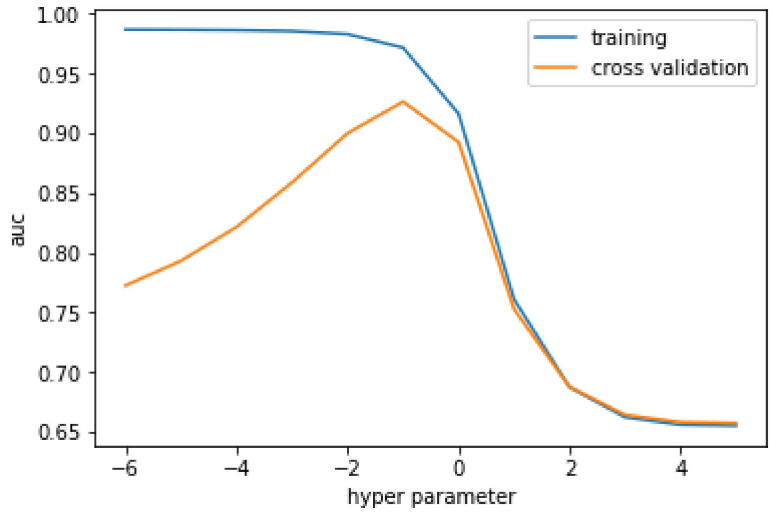

## [5.2] Applying Naive Bayes on TFIDF, SET 2

In [172]:
```
#with hyper parameter tuning
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score

cvscores=[]
cvscores1=[]

alpha=[10**i for i in range(-6,6,1)]
for i in alpha:
    nbx=MultinomialNB(alpha=i,class_prior=[0.5,0.5])
    nbx.fit(xtraintfidfencoding,ytrain)
    predict1=nbx.predict_proba(xtraintfidfencoding)[:,1]
    cvscores.append(roc_auc_score(ytrain,predict1))
    predict2=nbx.predict_proba(xcvtfidfencoding)[:,1]
    cvscores1.append(roc_auc_score(ycv,predict2))
    optimal_k=np.argmax(cvscores1)
fig,ax=plt.subplots()
ax.plot(np.log10(alpha),cvscores,label='training')
ax.legend()
ax.plot(np.log10(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)
```

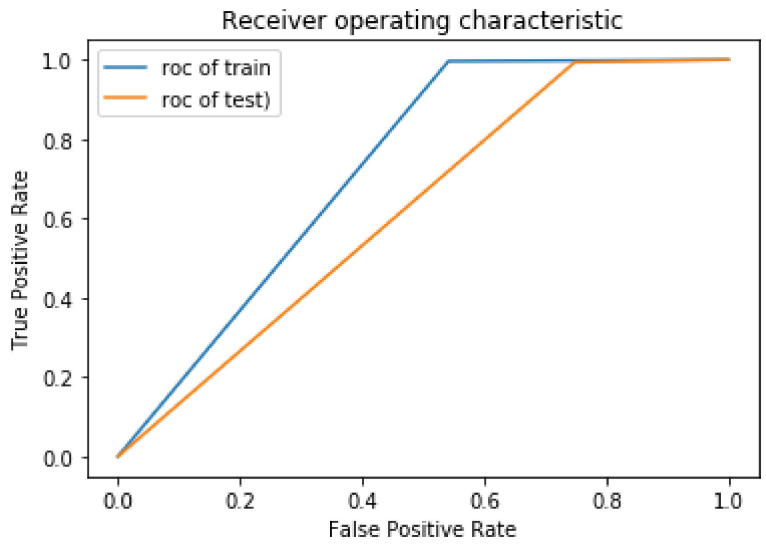



```
[0.7726895247299775, 0.7933391329716262, 0.8215448469609232, 0.858855398285382,
0.9000900172507063, 0.9264379066230933, 0.8926524784190661, 0.7528040685613191,
0.6878180391762252, 0.6643722603339453, 0.658195957570205, 0.6574325855427148]
0.1
[0.7726895247299775, 0.7933391329716262, 0.8215448469609232, 0.858855398285382,
0.9000900172507063, 0.9264379066230933, 0.8926524784190661, 0.7528040685613191,
0.6878180391762252, 0.6643722603339453, 0.658195957570205, 0.6574325855427148]
```

In [177]:

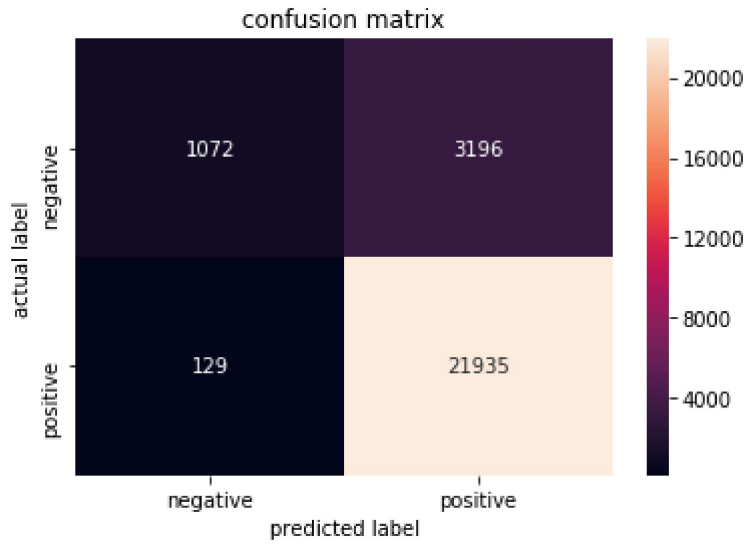
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
nbx=MultinomialNB(alpha=0.1)
nbx.fit(xtraintfidfencoding,ytrain)
predictrain=nbx.predict(xtraintfidfencoding)
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=nbx.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))
```

```
0.6226624404560848
```



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()
```

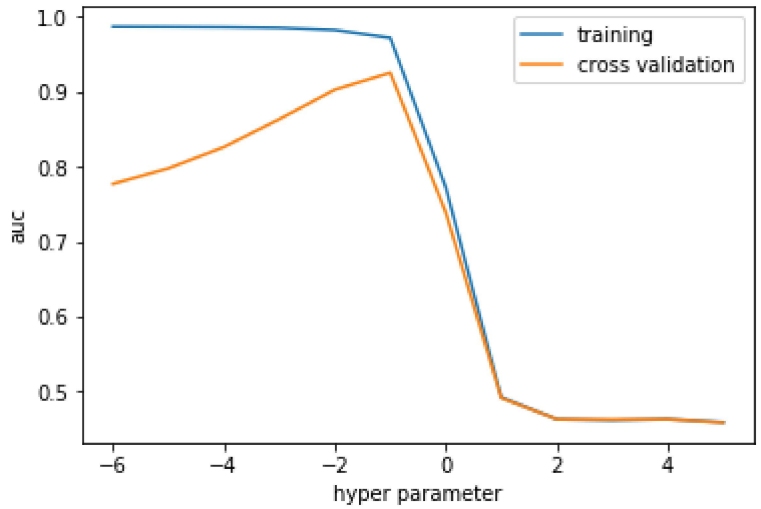


In [174]:
```
#with hyper parameter tuning
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score

cvscores=[]
cvscores1=[]

alpha=[10**i for i in range(-6,6,1)]
for i in alpha:
    nbx=MultinomialNB(alpha=i,class_prior=[0.5,0.5])
    nbx.fit(xtraintfidfencoding1,ytrain)
    predict1=nbx.predict_proba(xtraintfidfencoding1)[:,1]
    cvscores.append(roc_auc_score(ytrain,predict1))
    predict2=nbx.predict_proba(xcvtfidfencoding1)[:,1]
    cvscores1.append(roc_auc_score(ycv,predict2))
    optimal_k=np.argmax(cvscores1)
fig,ax=plt.subplots()
ax.plot(np.log10(alpha),cvscores,label='training')
ax.legend()
ax.plot(np.log10(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)
```

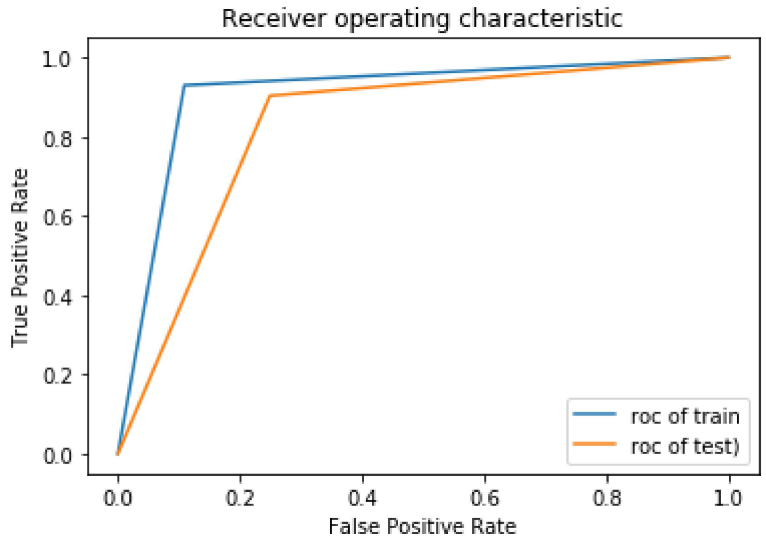



```
[0.7770694314599371, 0.7976826700801458, 0.8261125008921771, 0.863328471699754
2, 0.902725742242545, 0.9252676921144811, 0.7389852212579561, 0.491613243300481
73, 0.4631174986436959, 0.4627944866774079, 0.4632142486621244, 0.4586466434837
144]
0.1
[0.7770694314599371, 0.7976826700801458, 0.8261125008921771, 0.863328471699754
2, 0.902725742242545, 0.9252676921144811, 0.7389852212579561, 0.491613243300481
73, 0.4631174986436959, 0.4627944866774079, 0.4632142486621244, 0.4586466434837
144]
```

In [175]:

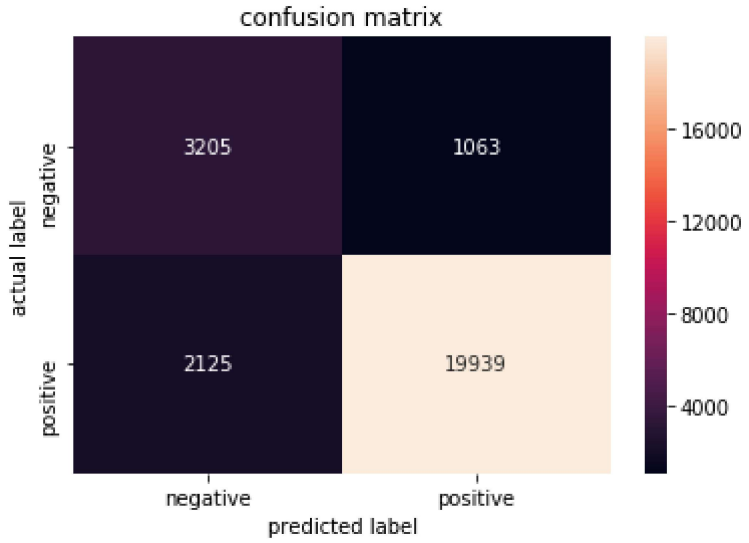
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
nbx=MultinomialNB(alpha=0.1,class_prior=[0.5,0.5])
nbx.fit(xtraintfidfencoding1,ytrain)
predictrain=nbx.predict(xtraintfidfencoding1)
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=nbx.predict(xtesttfidfencoding1)
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))
```

```
0.8273132373539903
```



In [176]:
```
#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()
```




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

In [0]:
```
featurs=tfidf.get_feature_names()
a=nbx.coef_
x=np.argsort(a,axis=None)
```

In [0]:
```
y=np.argsort(a,axis=None)[::-1]
k=[featurs[i] for i in y]
print('top positive words arre',k[:10])
```

```
top positive words arre ['great', 'good', 'like', 'coffee', 'love', 'tea', 'tas
te', 'one', 'product', 'flavor']
```

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

In [0]:
```
p=[feat[i] for i in x]
print('top negative words are',p[:10])
```

```
top negative words are ['unemployment', 'seder', 'stark', 'ketone', 'willies',
'stargazers', 'paused', 'battlefield', 'updatenormally', 'sedning']
```

# [6] Conclusions

In [211]:
```
data = [['no',1,0.84], ['yes',1,0.84],['no',0.1,0.62],['yes',0.1, 0.82]]
pd.DataFrame(data, columns=["feature engineering done", "hyperparameter",'auc'],
```

Out[211]:

| | feature engineering done | hyperparameter | auc |
|---|---|---|---|
| **bow** | no | 1.0 | 0.84 |
| **bow** | yes | 1.0 | 0.84 |
| **tfidf** | no | 0.1 | 0.62 |
| **tfidf** | yes | 0.1 | 0.82 |

##DOCUMENTATION

**WE HAVE USED THE NAIVE BAYES FOR THE BOTH COUNT VECTORIZER AND TFIDF VECTORIZER. WE HAVE PERFORMED THE HYPER PARAMETER TUNING AND OBTAINED THE BEST PARAMETERS AND OBTAINES THE ROC CURVE WHICH IS TPR VS FPR. WE HAVE OBTAINED THE AUC VALUES AND PLOTTED THE AUC CURVE. THE KEYTAKE AWAY FROM THIS MODEL IS WE HAVE DONE THE FEATURE ENGINEERING TECHNIQUE WE COUNTED THE NUMBER OF WORDS AND STACKED WITH THE FEATURES . IN CASE OF TFIDF VECTORIZER IT IMPROVED MODEL A LOT INTERMS OF AREA UNDER CURVE VALUES OBTAINED FRON 0.62 TO 0.82**

**WE HAVE ACHIEVED THE AUC VLAUES OF NEARLY 0.84 WHICH IS COMPARITIVEELY GOOD. AND WE KNOW THE NAIVE BAYES PERFORMS GOOD ON TEXT DATA AND SPECIALLY USED IN FAKE EMAIL DETECTION TASKS ETC.. AND ONE OF THE KEYTAKE AWAY IS OUR COUNT VECTORIZER MODEL PERFORMED GOOD THAN OUR TFIDF MODEL.AND ANOTHER IMPORTANT THING IS WE HAVE PLOTTED THE CONFUSON MATRIX DUE TO DATA IMBALANCE ACCURACY IS NOT RIGHT METRIC.WE HAVE INCREASED THE MODEL PERFORMANCE USING THE FEATURE ENGINEERING.**

In [0]: