

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 - unique 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 considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral 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 will be set to "positive". Otherwise, it will be set to "negative".

In [99]:

```
%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_notebook
import os

```

In [100]:

```

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

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

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", con)

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

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

```

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

Out[100]:

		Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	
				Natalia					

		Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
2	3		B0001QOCH0	ABXIMWJIXAIN	"Natalia Corres"	1	1	1	1219017600

In [101]:

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

In [102]:

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

(80668, 7)

Out[102]:

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

In [103]:

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

Out[103]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

In [104]:

```
display['COUNT(*)'].sum()
```

Out[104]:

393063

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove

duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [105]:

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

Out[105]:

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

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

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

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

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

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delete 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 [106]:

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

In [107]:

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

```
Out[107]:
(87775, 10)
```

```
In [108]:
```

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

```
Out[108]:
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 calculations

```
In [109]:
```

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

display.head()
```

```
Out[109]:
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	12248926
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128832

```
In [110]:
```

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [111]:
```

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

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

```
(87773, 10)
```

```
Out[111]:
```

```
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 observed to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [112]:

```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

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

=====

The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to it. Very little of the 2 lbs that I bought were eaten and I threw the rest away. I would not buy the candy again.

=====

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

=====

My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are really small in size. They are great for training. You can give your dog several of these without worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers. It's definitely worth it to buy a big bag if your dog eats them a lot.

In [113]:

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

In [114]:

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

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

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

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

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

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

=====

The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to it. Very little of the 2 lbs that I bought were eaten and I threw the rest away. I would not buy the candy again.

=====

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

=====

My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are really small in size. They are great for training. You can give your dog several of these without worrying about him over eating. Amazon's price was much more reasonable than any other retailer. You can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers. It's definitely worth it to buy a big bag if your dog eats them a lot.

In [115]:

```
# 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 [116]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
```

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

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

In [117]:

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

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

In [118]:

```
#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 [119]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've", \
    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
    'himself', \
    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
    'their', \
    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
    'these', 'those', \
    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
    'do', 'does', \
    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
    'before', 'after', \
    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further', \
    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more', \
    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "dc
esn't", 'hadn', \
    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
    "mightn't", 'mustn', \
    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
    "wasn't", 'weren', "weren't", \
    'won', "won't", 'wouldn', "wouldn't"])
```

In [120]:

```
# Combining all the above stundents
from tqdm import tqdm_notebook
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentence in tqdm_notebook(final['Text'].values):
    sentence = re.sub(r"http\S+", "", sentence)
    sentence = BeautifulSoup(sentence, 'lxml').get_text()
    sentence = decontracted(sentence)
```



```

sentence = re.sub("\S*\d\S*", "", sentence).strip()
sentence = re.sub('[^A-Za-z]+', ' ', sentence)
# https://gist.github.com/sebleier/554280
sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwords)
preprocessed_reviews.append(sentence.strip())

```

In [121]:

```
preprocessed_reviews[1500]
```

Out[121]:

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

[3.2] Preprocessing Review Summary

In [122]:

```

## Similarly you can do preprocessing for review summary also.
from tqdm import tqdm_notebook
preprocessed_summary = []
# tqdm is for printing the status bar
for sentence in tqdm_notebook(final['Summary'].values):
    sentence = re.sub(r"http\S+", "", sentence)
    sentence = BeautifulSoup(sentence, 'lxml').get_text()
    sentence = decontracted(sentence)
    sentence = re.sub("\S*\d\S*", "", sentence).strip()
    sentence = re.sub('[^A-Za-z]+', ' ', sentence)
    # https://gist.github.com/sebleier/554280
    sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwords)
    preprocessed_summary.append(sentence.strip())

```

In [123]:

```
preprocessed_summary[1500]
```

Out[123]:

```
'hot stuff'
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

```

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

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

```

```

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

```

[4.3] TF-IDF

In [27]:

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

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

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

[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](#) 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](#) 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](#) with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps](#).

6. Conclusion

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

Note: Data Leakage

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-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method `fit_transform()` on your train data, and apply the method `transform()` on cv/test data.
4. For more details please go through this [link](#).

Applying Multinomial Naive Bayes

[5.1] Applying Naive Bayes on BOW, SET 1

In [28]:

```
# Please write all the code with proper documentation
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
import math

bow_vect=CountVectorizer()

x=preprocessed_reviews
y=np.array(final['Score'])
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
x_train,x_cv,y_train,y_cv=train_test_split(x_train,y_train,test_size=0.3)
fbowx_tr=bow_vect.fit_transform(x_train)
fbowx_cv=bow_vect.transform(x_cv)
fbowx_te=bow_vect.transform(x_test)
nb=MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)
nb.fit(fbowx_tr,y_train)
```

Out[28]:

MultinomialNB()

In [29]:

```
print(fbowx_tr.shape, y_train.shape)
print(fbowx_cv.shape, y_cv.shape)
print(fbowx_te.shape, y_test.shape)
```

```
(43008, 39161) (43008,)
(18433, 39161) (18433,)
(26332, 39161) (26332,)
```

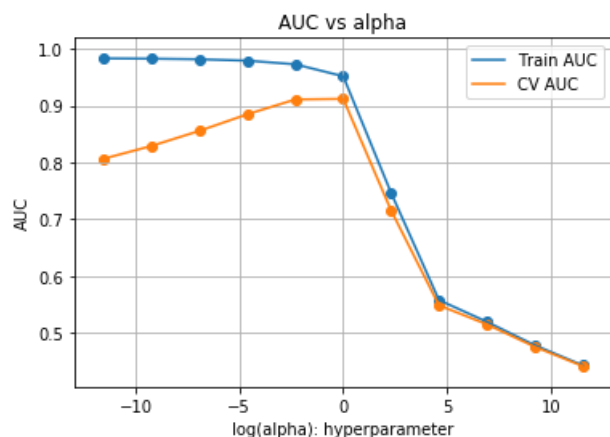
In [30]:

```
from sklearn.metrics import roc_auc_score
auc_cv=[]
auc_train=[]
alpha=[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
for i in tqdm_notebook(alpha): #simple cv using for loop
    nb=MultinomialNB(alpha=i, fit_prior=True, class_prior=None)
    nb.fit(fbowx_tr,y_train)
    pred_b = nb.predict_proba(fbowx_cv)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred_b))
    pred_b1=nb.predict_proba(fbowx_tr)[:,1]
    auc_train.append(roc_auc_score(y_train,pred_b1))

best_alpha= alpha[auc_cv.index(max(auc_cv))] #max value in auc_cv list is used to find best alpha
alpha=[math.log(j) for j in alpha] # some of the alpha values are almost close to zero so taking log values
plt.plot(alpha, auc_train, label='Train AUC')
plt.plot(alpha, auc_cv, label='CV AUC')

plt.scatter(alpha, auc_train)
plt.scatter(alpha, auc_cv)
plt.legend()
plt.xlabel("log(alpha): hyperparameter")
```

```
plt.ylabel("AUC")
plt.title("AUC vs alpha")
plt.grid()
plt.show()
print("Best alpha value for max auc =",best_alpha)
```

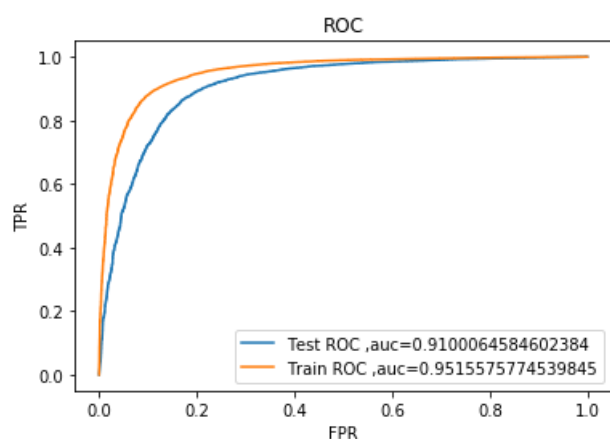


Best alpha value for max auc = 1

In [31]:

```
#Plotting ROC curve
nb=MultinomialNB(alpha=1, fit_prior=True, class_prior=None)
nb.fit(fbowx_tr,y_train)
pred_b=nb.predict_proba(fbowx_te)[:,-1]
fpr_te, tpr_te, thresholds_te = metrics.roc_curve(y_test, pred_b)
pred_b1=nb.predict_proba(fbowx_tr)[:,-1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_b1)

plt.plot(fpr_te, tpr_te, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_b)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_b1)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



In [32]:

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

def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
```

```

    if i>=threshold:
        predictions.append(1)
    else:
        predictions.append(0)
    return predictions
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u
5LEf3yb0YJBVbZW

```

In [33]:

```

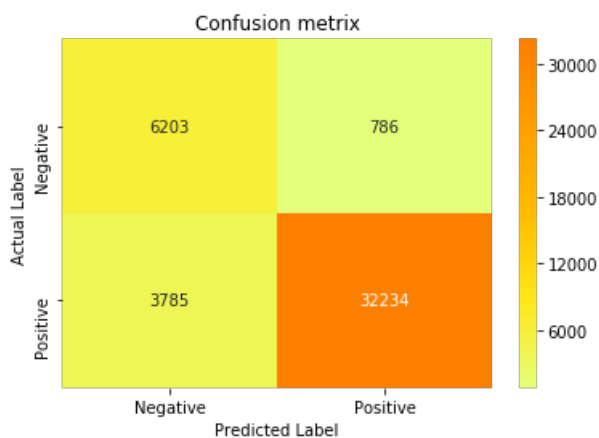
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_b1, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Wistia")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u
5LEf3yb0YJBVbZW

```

the maximum value of $tpr \cdot (1-fpr)$ 0.7942720696713735 for threshold 0.953
Train confusion matrix

Out[33]:

Text(33,0.5,'Actual Label')



In [34]:

```

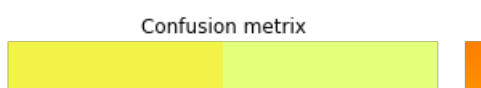
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Test confusion matrix")
df_b=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_b, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df_b,annot = True,fmt='d',cmap="Wistia")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
#This code is copied and modified from: https://colab.research.google.com/drive/1EkYHI-vGKnURqLL_u
5LEf3yb0YJBVbZW

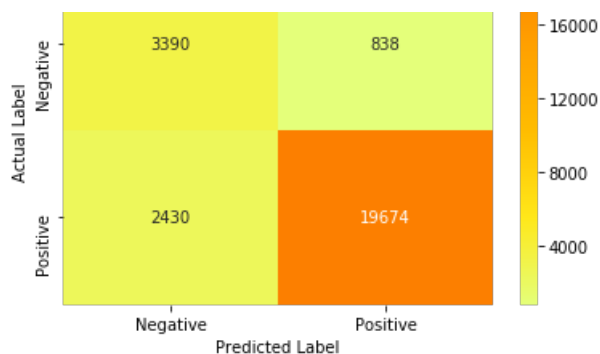
```

the maximum value of $tpr \cdot (1-fpr)$ 0.7942720696713735 for threshold 0.953
Test confusion matrix

Out[34]:

Text(33,0.5,'Actual Label')





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

In [35]:

```
# Please write all the code with proper documentation
print("Top 10 important features of positive class from:")
nb=MultinomialNB(alpha=1)
nb.fit(fbowx_tr,y_train)
feat=nb.feature_log_prob_
feature_names=bow_vect.get_feature_names()
neg_feat=np.argsort(feat[0])[:,::-1]
pos_feat=np.argsort(feat[1])[:,::-1]
for i in list(pos_feat[0:10]):
    print((feature_names[i]))
#This code is copied and modified from:https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-bayes
```

Top 10 important features of positive class from:

```
not
like
good
great
one
taste
coffee
love
would
flavor
```

In [88]:

```
print(type(feat))
```

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

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

In [36]:

```
# Please write all the code with proper documentation
print("Top 10 important features of negative class:")
for i in list(neg_feat[0:10]):
    print((feature_names[i]))
```

Top 10 important features of negative class:

```
not
like
would
product
taste
one
good
coffee
no
flavor
```

Feature Engineering

In [37]:

```
for i in range(len(preprocessed_reviews)): #considering length of review and some features from r  
ewiw summary  
    preprocessed_reviews[i]+' '+preprocessed_summary[i]+' '+str(len(final.Text.iloc[i]))  
  
preprocessedbow_reviews=preprocessed_reviews
```

In [38]:

```
preprocessedbow_reviews[1500]
```

Out[38]:

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

In [43]:

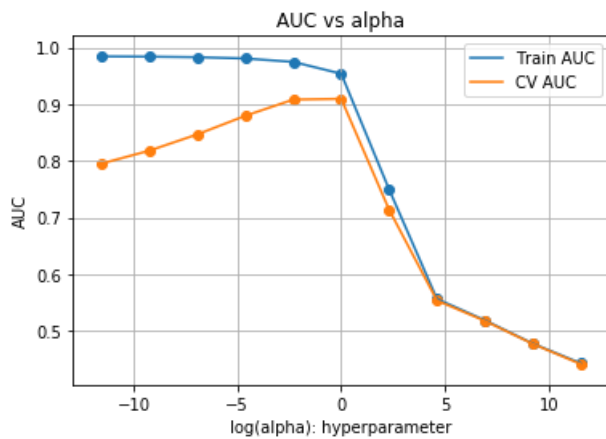
```
from sklearn.model_selection import train_test_split  
from sklearn.naive_bayes import MultinomialNB  
from sklearn.model_selection import cross_val_score  
import math  
  
bow_vect=CountVectorizer()  
  
x=preprocessedbow_reviews  
y=np.array(final['Score'])  
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)  
x_train,x_cv,y_train,y_cv=train_test_split(x_train,y_train,test_size=0.3)  
fbowx_tr=bow_vect.fit_transform(x_train)  
fbowx_cv=bow_vect.transform(x_cv)  
fbowx_te=bow_vect.transform(x_test)  
nb=MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)  
nb.fit(fbowx_tr,y_train)
```

Out[43]:

```
MultinomialNB()
```

In [45]:

```
from sklearn.metrics import roc_auc_score  
auc_cv=[]  
auc_train=[]  
alpha=[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]  
for i in tqdm_notebook(alpha): #simple cv using for loop  
    nb=MultinomialNB(alpha=i, fit_prior=True, class_prior=None)  
    nb.fit(fbowx_tr,y_train)  
    pred_b = nb.predict_proba(fbowx_cv)[:,-1]  
    auc_cv.append(roc_auc_score(y_cv,pred_b))  
    pred_b1=nb.predict_proba(fbowx_tr)[:,-1]  
    auc_train.append(roc_auc_score(y_train,pred_b1))  
  
best_alpha= alpha[auc_cv.index(max(auc_cv))]  
alpha=[math.log(j) for j in alpha] # some of the alpha values are almost close to zero so taking lo  
g values  
plt.plot(alpha, auc_train, label='Train AUC')  
plt.plot(alpha, auc_cv, label='CV AUC')  
  
plt.scatter(alpha, auc_train)  
plt.scatter(alpha, auc_cv)  
plt.legend()  
plt.xlabel("log(alpha): hyperparameter")  
plt.ylabel("AUC")  
plt.title("AUC vs alpha")  
plt.grid()  
plt.show()  
print("Best alpha value for max auc =",best_alpha)
```

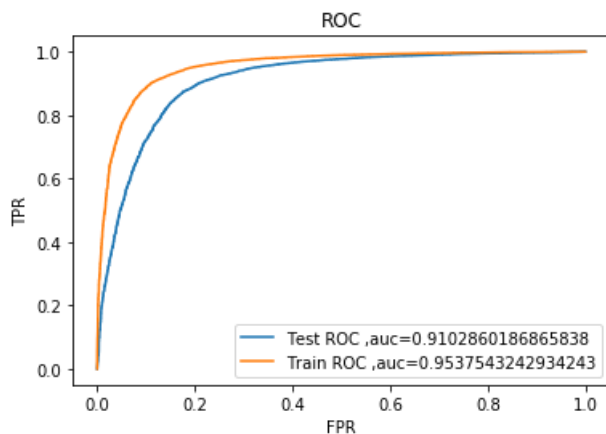


Best alpha value for max auc = 1

In [46]:

```
#Plotting ROC curve
nb=MultinomialNB(alpha=1, fit_prior=True, class_prior=None)
nb.fit(fbowx_tr,y_train)
pred_b=nb.predict_proba(fbowx_te)[:,1]
fpr_te, tpr_te, thresholds_te = metrics.roc_curve(y_test, pred_b)
pred_b1=nb.predict_proba(fbowx_tr)[:,1]
fpr_tr,tpr_tr,thresholds_tr=metrics.roc_curve(y_train,pred_b1)

plt.plot(fpr_te, tpr_te, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_b)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_b1)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```



In [47]:

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

def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

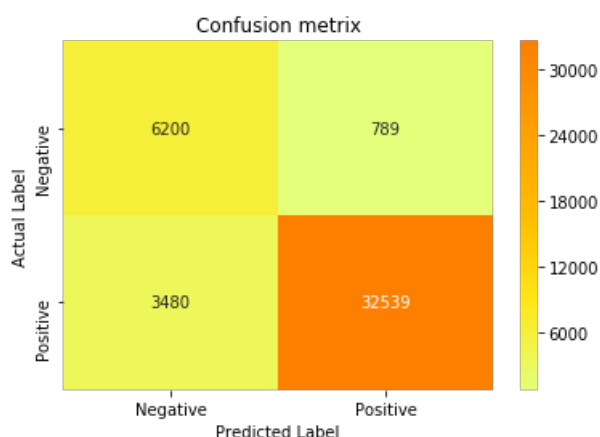

In [48]:

```
#Comfuiou matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_b1, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Wistia")
plt.title('Confusion matrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
```

the maximum value of $tpr \cdot (1 - fpr)$ 0.8013997445450871 for threshold 0.947
Train confusion matrix

Out[48]:

Text(33,0.5,'Actual Label')



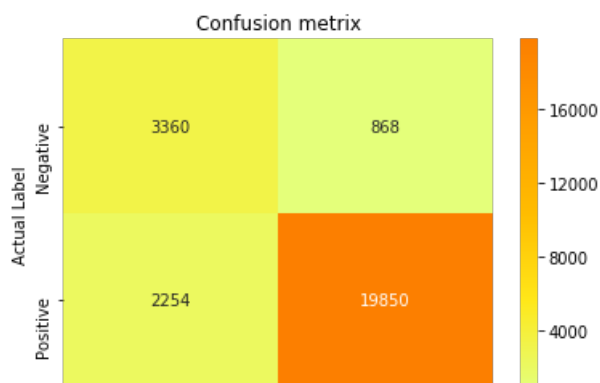
In [49]:

```
#Comfuiou matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Test confusion matrix")
df_b=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_b, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df_b,annot = True,fmt='d',cmap="Wistia")
plt.title('Confusion matrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
```

the maximum value of $tpr \cdot (1 - fpr)$ 0.8013997445450871 for threshold 0.947
Test confusion matrix

Out[49]:

Text(33,0.5,'Actual Label')



Negative Positive
Predicted Label

In [50]:

```
print("Top 10 important features of positive class from:")
nb=MultinomialNB(alpha=1)
nb.fit(fbowx_tr,y_train)
feat=nb.feature_log_prob_
feature_names=bow_vect.get_feature_names()
neg_feat=np.argsort(feat[0])[:,::-1]
pos_feat=np.argsort(feat[1])[:,::-1]
for i in list(pos_feat[0:10]):
    print((feature_names[i]))
```

Top 10 important features of positive class from:
not
like
good
great
one
taste
coffee
love
flavor
would

In [51]:

```
print("Top 10 important features of negative class:")
for i in list(neg_feat[0:10]):
    print((feature_names[i]))
```

Top 10 important features of negative class:
not
like
would
product
taste
one
coffee
good
flavor
no

[5.2] Applying Naive Bayes on TFIDF, SET 2

In [52]:

```
tf_vect=TfidfVectorizer(min_df=10)
tf_vect.fit(preprocessed_reviews)

ftfx_tr=tf_vect.fit_transform(x_train)
ftfx_cv=tf_vect.transform(x_cv)
ftfx_te=tf_vect.transform(x_test)
nb=MultinomialNB(alpha=1, fit_prior=True, class_prior=None)
nb.fit(ftfx_tr,y_train)
print(ftfx_tr.shape) # checking size to ensure correctly splitted
print(ftfx_cv.shape, y_cv.shape)
print(ftfx_te.shape, y_test.shape)
```

(43008, 8116) (43008,)
(18433, 8116) (18433,)
(26332, 8116) (26332,)

In [53]:

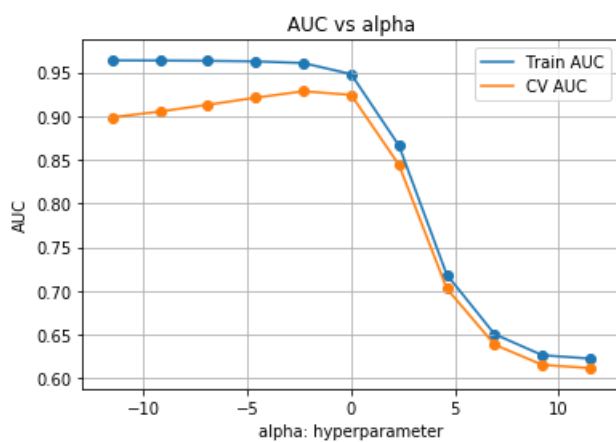
```
auc_cv=[]
auc_train=[]
alpha=[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
```

```

for i in tqdm_notebook(alpha):      #simple cv using for loop
    nb=MultinomialNB(alpha=i, fit_prior=True, class_prior=None)
    nb.fit(ftfx_tr,y_train)
    pred_tf = nb.predict_proba(ftfx_cv)[: ,1]
    auc_cv.append(roc_auc_score(y_cv,pred_tf))
    pred_tfl=nb.predict_proba(ftfx_tr)[: ,1]
    auc_train.append(roc_auc_score(y_train,pred_tfl))
best_alpha= alpha[auc_cv.index(max(auc_cv))]
alpha=[math.log(j) for j in alpha]
plt.plot(alpha, auc_train, label='Train AUC')
plt.plot(alpha, auc_cv, label='CV AUC')

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

```



Best alpha value for max auc = 0.1

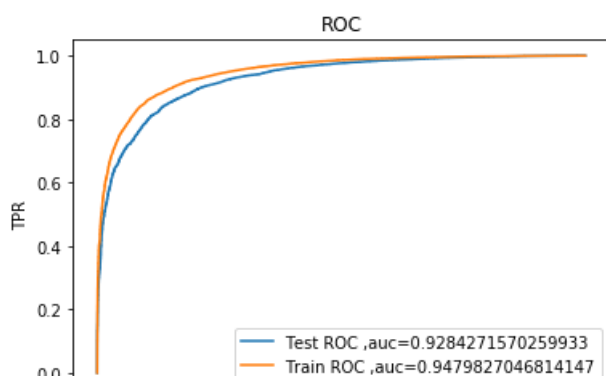
In [54]:

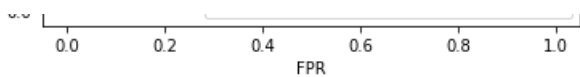
```

#Plotting ROC curve
nb=MultinomialNB(alpha=1, fit_prior=True, class_prior=None)
nb.fit(ftfx_tr,y_train)
pred_b=nb.predict_proba(ftfx_te)[: ,1]
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_b)
pred_b1=nb.predict_proba(ftfx_tr)[: ,1]
fpr_tr, tpr_tr, thresholds_tr=metrics.roc_curve(y_train,pred_b1)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_b)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_b1)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()

```





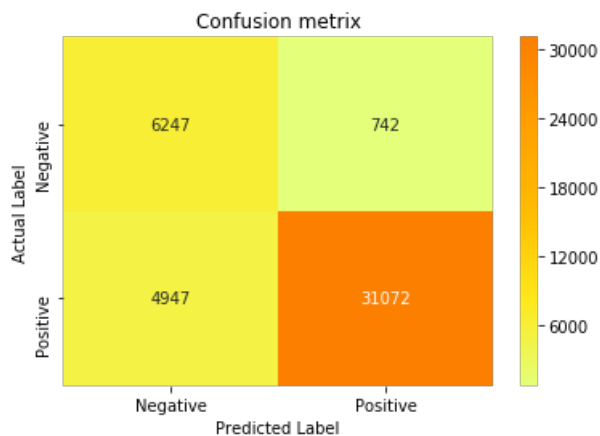
In [55]:

```
#Confuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_b1, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Wistia")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
```

the maximum value of $tpr \cdot (1 - fpr)$ 0.7710703835896597 for threshold 0.861
Train confusion matrix

Out[55]:

Text(33,0.5,'Actual Label')



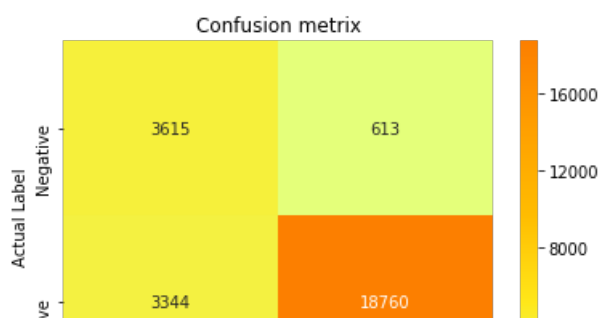
In [56]:

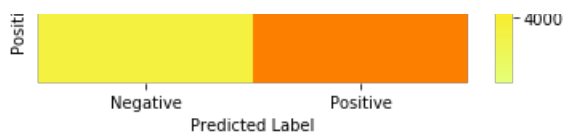
```
#Comfuion matrix for Test data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Test confusion matrix")
df_f=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_b, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df_f,annot = True,fmt='d',cmap="Wistia")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
```

the maximum value of $tpr \cdot (1 - fpr)$ 0.7710703835896597 for threshold 0.861
Test confusion matrix

Out[56]:

Text(33,0.5,'Actual Label')





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

In [57]:

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

print("Top 10 important features of positive class from:")
nb=MultinomialNB(alpha=0.1)
nb.fit(ftfx_tr,y_train)
feat=nb.feature_log_prob_
feature_names=tf_vect.get_feature_names()
neg_feat=np.argsort(feat[0])[:,::-1]
pos_feat=np.argsort(feat[1])[:,::-1]
for i in list(pos_feat[0:10]):
    print((feature_names[i]))
```

```
Top 10 important features of positive class from:
not
great
good
like
coffee
love
tea
taste
product
one
```

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

In [58]:

```
# Please write all the code with proper documentation
for i in list(neg_feat[0:10]):
    print((feature_names[i]))
```

```
not
like
product
taste
would
coffee
one
flavor
no
buy
```

Feature Engineering

In [59]:

```
for i in range(len(preprocessed_reviews)):
    preprocessed_reviews[i]+' '+preprocessed_summary[i]+' '+str(len(final.Text.iloc[i]))

preprocessedtf_reviews=preprocessed_reviews
```

In [60]:

```
preprocessedtf_reviews[1500]
```

Out [60]:

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

In [61]:

```
tf_vect=TfidfVectorizer(min_df=10)
tf_vect.fit(preprocessedtf_reviews)

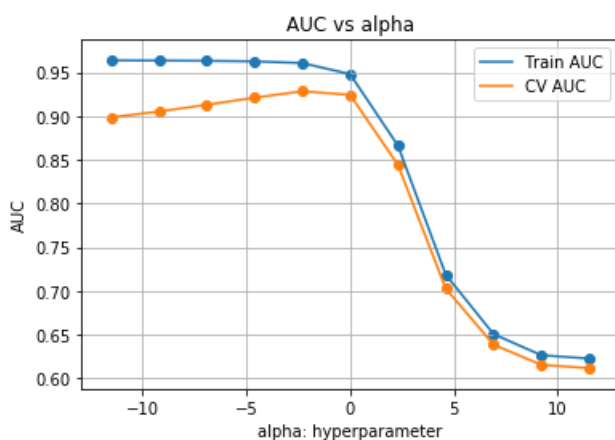
ftfx_tr=tf_vect.fit_transform(x_train)
ftfx_cv=tf_vect.transform(x_cv)
ftfx_te=tf_vect.transform(x_test)
nb=MultinomialNB(alpha=1, fit_prior=True, class_prior=None)
nb.fit(ftfx_tr,y_train)
print(ftfx_tr.shape, y_train.shape) # checking size to ensure correctly splitted
print(ftfx_cv.shape, y_cv.shape)
print(ftfx_te.shape, y_test.shape)

(43008, 8116) (43008,)
(18433, 8116) (18433,)
(26332, 8116) (26332,)
```

In [62]:

```
auc_cv=[]
auc_train=[]
alpha=[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
for i in tqdm_notebook(alpha): #simple cv using for loop
    nb=MultinomialNB(alpha=i, fit_prior=True, class_prior=None)
    nb.fit(ftfx_tr,y_train)
    pred_tf = nb.predict_proba(ftfx_cv)[: ,1]
    auc_cv.append(roc_auc_score(y_cv,pred_tf))
    pred_tfl=nb.predict_proba(ftfx_tr)[: ,1]
    auc_train.append(roc_auc_score(y_train,pred_tfl))
best_alpha= alpha[auc_cv.index(max(auc_cv))]
alpha=[math.log(j) for j in alpha]
plt.plot(alpha, auc_train, label='Train AUC')
plt.plot(alpha, auc_cv, label='CV AUC')

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



Best alpha value for max auc = 0.1

In [63]:

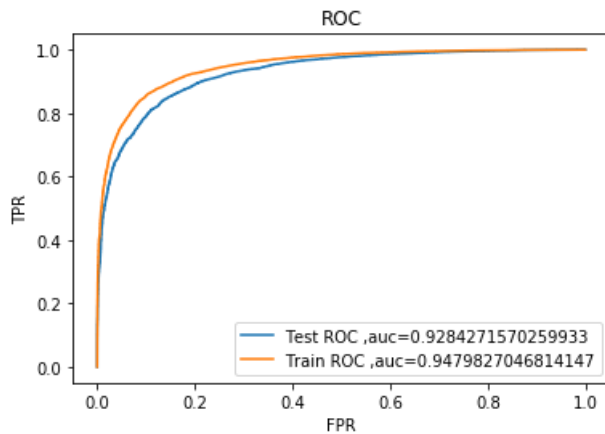
```
nb=MultinomialNB(alpha=1, fit_prior=True, class_prior=None)
nb.fit(ftfx_tr,y_train)
pred_b=nb.predict_proba(ftfx_te)[: ,1]
```

```

pred_b=nb.predict_proba(ftfx_te,1,1)
fpr_te, trp_te, thresholds_te = metrics.roc_curve(y_test, pred_b)
pred_b1=nb.predict_proba(ftfx_tr)[: ,1]
fpr_tr, tpr_tr, thresholds_tr=metrics.roc_curve(y_train,pred_b1)

plt.plot(fpr_te, trp_te, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_b)))
plt.plot(fpr_tr, tpr_tr, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_b1)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()

```



In [64]:

```

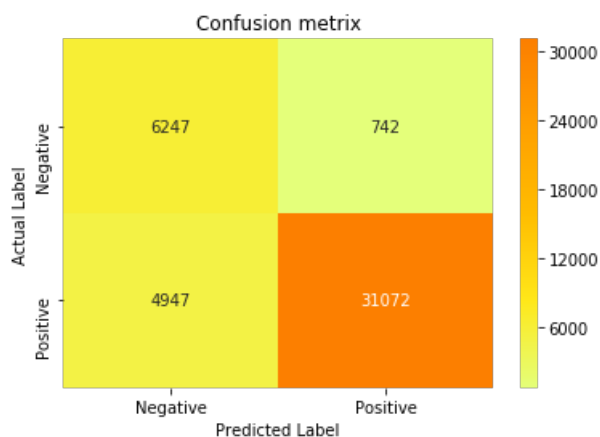
#Comfuion matrix for Train data
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds_tr, fpr_tr, tpr_tr)
print("Train confusion matrix")
df=pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(pred_b1, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df,annot = True,fmt='d',cmap="Wistia")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")

```

the maximum value of $tpr \cdot (1 - fpr)$ 0.7710703835896597 for threshold 0.861
Train confusion matrix

Out[64]:

Text(33,0.5,'Actual Label')



In [65]:

```

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

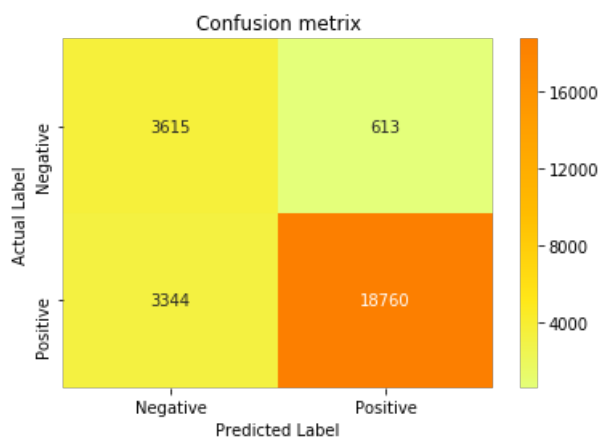
```

```
df_f=pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(pred_b, best_t)),index=['Negative','Positive'],columns=['Negative','Positive'])
sns.heatmap(df_f,annot = True,fmt='d',cmap="Wistia")
plt.title('Confusion metrix')
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
```

the maximum value of $tpr \cdot (1 - fpr)$ 0.7710703835896597 for threshold 0.861
Test confusion matrix

Out[65]:

Text(33,0.5,'Actual Label')



In [66]:

```
print("Top 10 important features of positive class from:")
nb=MultinomialNB(alpha=0.1)
nb.fit(ftfx_tr,y_train)
feat=nb.feature_log_prob_
feature_names=tf_vect.get_feature_names()
neg_feat=np.argsort(feat[0])[::-1]
pos_feat=np.argsort(feat[1])[::-1]
for i in list(pos_feat[0:10]):
    print((feature_names[i]))
```

Top 10 important features of positive class from:
not
great
good
like
coffee
love
tea
taste
product
one

In [67]:

```
for i in list(neg_feat[0:10]):
    print((feature_names[i]))
```

not
like
product
taste
would
coffee
one
flavor
no
buy

[6] Conclusions

In []:

```
# Please compare all your models using Prettytable library
```

In [92]:

```
from prettytable import PrettyTable

x=PrettyTable()
x.field_names=['Vectorizer','Feature Engineering','Hyperparameter','AUC']
x.add_row(["BOW","W/O FE",1,0.91])
x.add_row(["TF-IDF","W/O FE",0.1,0.92])
x.add_row(["BOW","With FE",1,0.91])
x.add_row(["TF-IDF","With FE",0.1,0.92])
print(x)
print("Een after some feature engineering, accuracy is almost same no much difference")
```

```
+-----+-----+-----+-----+
| Vectorizer | Feature Engineering | Hyperparameter | AUC |
+-----+-----+-----+-----+
| BOW        | W/O FE              | 1              | 0.91 |
| TF-IDF     | W/O FE              | 0.1            | 0.92 |
| BOW        | With FE             | 1              | 0.91 |
| TF-IDF     | With FE             | 0.1            | 0.92 |
+-----+-----+-----+-----+
```

Een after some feature engineering, accuracy is almost same no much difference