

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 - 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 wil be set to "positive". Otherwise, it will be set to "negative".

```
In [1]: %matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

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

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

from tqdm import tqdm
import os
```

```
In [2]: # using SQLite Table to read data.  
con = sqlite3.connect('database.sqlite')  
  
# filtering only positive and negative reviews i.e.  
# not taking into consideration those reviews with Score=3  
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points  
# you can change the number to any other number based on your computing power  
  
# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", con)  
# for tsne assignment you can take 5k data points  
  
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50000""", 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 (500000, 10)

Out[2]:

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

In [3]:

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

In [4]: `print(display.shape)
display.head()`

(80668, 7)

Out[4]:

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



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

Out[5]:

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



```
In [6]: display['COUNT(*)'].sum()
```

```
Out[6]: 393063
```

[2] Exploratory Data Analysis

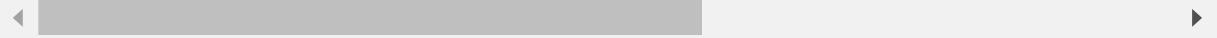
[2.1] Data Cleaning: Deduplication

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

```
In [7]: display= pd.read_sql_query("""  
SELECT *  
FROM Reviews  
WHERE Score != 3 AND UserId="AR5J8UI46CURR"  
ORDER BY ProductID  
""", con)  
display.head()
```

Out[7]:

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



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

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

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

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

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and 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 [8]: *#Sorting data according to ProductId in ascending order*
`sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')`

In [9]: *#Deduplication of entries*
`final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape`

Out[9]: (348262, 10)

In [10]: *#Checking to see how much % of data still remains*
`(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100`

Out[10]: 69.6524

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

◀ ▶

```
In [12]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [13]: #Before starting the next phase of preprocessing lets see the number of entries left
```

```
print(final.shape)
```

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

(348260, 10)

Out[13]: 1 293516

0 54744

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 [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)
```

This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and hugged it when told it was his to keep and he did not have to return it to the library.

I've purchased both the Expressione Espresso (classic) and the 100% Arabica. My vote is definitely with the 100% Arabica. The flavor has more bite and flavor (much more like European coffee than American).

This is a great product. It is very healthy for all of our dogs, and it is the first food that they all love to eat. It helped my older dog lose weight and my 10 year old lab gain the weight he needed to be healthy.

I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a tennis party, perfect! They were the size of malted milk balls. Unfortunately, they arrived 3 days after the party. The caveat here is, not everything from Amazon may arrive at an impressive 2 or 3 days. This shipment took 8 days from the Candy/Cosmetic Depot back east to southern California.

```
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)
```

This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and hugged it when told it was his to keep and he did not have to return it to the library.

```
In [16]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

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

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

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

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

This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and hugged it when told it was his to keep and he did not have to return it to the library.

I've purchased both the Expressione Espresso (classic) and the 100% Arabica. My vote is definitely with the 100% Arabica. The flavor has more bite and flavor (much more like European coffee than American).

This is a great product. It is very healthy for all of our dogs, and it is the first food that they all love to eat. It helped my older dog lose weight and my 10 year old lab gain the weight he needed to be healthy.

I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a tennis party, perfect! They were the size of malted milk balls. Unfortunately, they arrived 3 days after the party. The caveat here is, not everything from Amazon may arrive at an impressive 2 or 3 days. This shipment took 8 days from the Candy/Cosmetic Depot back east to southern California.

In [17]: [# https://stackoverflow.com/a/47091490/4084039](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)
```

This is a great product. It is very healthy for all of our dogs, and it is the first food that they all love to eat. It helped my older dog lose weight and my 10 year old lab gain the weight he needed to be healthy.

In [19]: [#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039](https://stackoverflow.com/a/18082370/4084039)

```
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

This book was purchased as a birthday gift for a year old boy. He squealed with delight and hugged it when told it was his to keep and he did not have to return it to the library.

In [20]: [#remove spacial character: https://stackoverflow.com/a/5843547/4084039](https://stackoverflow.com/a/5843547/4084039)

```
sent_1500 = re.sub('[^A-Za-z0-9]+', '', sent_1500)
print(sent_1500)
```

This is a great product It is very healthy for all of our dogs and it is the first food that they all love to eat It helped my older dog lose weight and my 10 year old lab gain the weight he needed to be healthy

```
In [21]: # https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have removed in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'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', 'each', 'few', 'mo
re', \
    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'how', 'd', 'll', 'm', 'o', 're', \
    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', \
    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', \
    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'were
n', "weren't", \
    'won', "won't", 'wouldn', "wouldn't"])
```

```
In [22]: # Combining all the above students
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentence in tqdm(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())
```

100% [██████████] 348260/348260 [02:33<00:00, 2265.54it/s]

In [23]: preprocessed_reviews[1500]

Out[23]: 'great product healthy dogs first food love eat helped older dog lose weight year old lab gain wei
ght needed healthy'

In [24]: *#replacing the reviews with preprocessed reviews*
`final['Text']=preprocessed_reviews`

```
positive = final[final["Score"] == 1].sample(n = 50000)
negative = final[final["Score"] == 0].sample(n = 50000)
final_posinegi = pd.concat([positive, negative])
final_posinegi.shape
```

Out[24]: (100000, 10)

In [25]: `fd = final_posinegi`

In [26]: *# Sorting based on time*
`fd['Time'] = pd.to_datetime(fd['Time'], origin='unix', unit='s')`
`fd= fd.sort_values('Time')`
`fd.shape`

Out[26]: (100000, 10)

In [27]: *#splitting data into X and Y*
`Y= fd.Score`
`X=fd.Text`
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
`from sklearn.model_selection import train_test_split`
`X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=False)# this is for time series split`

`print('Before Vectorization')`
`print('Train--->',X_train.shape, y_train.shape)`
`print('Test---->',X_test.shape, y_test.shape)`

Before Vectorization
Train---> (67000,) (67000,)
Test----> (33000,) (33000,)

[4] Featurization

[4.1] BAG OF WORDS

```
In [28]: from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max)
vectorizer.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_test_bow.shape, y_test.shape)
```

After vectorizations
(67000, 50308) (67000,)
(33000, 50308) (33000,)

[4.2] Bi-Grams and n-Grams.

```
In [29]: #splitting data into X and Y
Y= fd.Score
X=fd.Text
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=False)# this is for time series split
#bi-gram, tri-gram and n-gram

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

# you can choose these numbers 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(X_train)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of our text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_shape()[1])
```

the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of our text BOW vectorizer (67000, 5000)
the number of unique words including both unigrams and bigrams 5000

[4.3] TF-IDF

```
In [30]: #splitting data into X and Y
Y= fd.Score
X=fd.Text
# https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=False)# this is for time series split

print('Before Vectorization')
print('Train--->',X_train.shape, y_train.shape)
print('Test---->',X_test.shape, y_test.shape)
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df = 10)
tf_idf_vect.fit(X_train)

X_train_tfidf = tf_idf_vect.transform(X_train)
X_test_tfidf = tf_idf_vect.transform(X_test)

print("After vectorizations")
print(X_train_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_test.shape)
```

Before Vectorization
Train---> (67000,) (67000,)
Test----> (33000,) (33000,)
After vectorizations
(67000, 39628) (67000,)
(33000, 39628) (33000,)

[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>)
 (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)
 (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)
 (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)

6. Conclusion (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>).

(<https://seaborn.pydata.org/generated/seaborn.heatmap.html>).

- 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 (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>) 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-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. (<https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf>)

Applying Multinomial Naive Bayes

Function for finding the best alpha using gridsearch on multinomial naive bayes

```
In [68]: def NB(Xtrain,ytrain):
    # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
    from sklearn.naive_bayes import MultinomialNB
    from sklearn.metrics import roc_auc_score
    from sklearn.model_selection import GridSearchCV
    import matplotlib.pyplot as plt
    import numpy as np

    K =[10**x for x in range(-4,5)]

    nb = MultinomialNB()
    parameters = {'alpha':K}
    clf = GridSearchCV(nb, parameters, cv=4, scoring='roc_auc',return_train_score=True)
    clf.fit(Xtrain, ytrain)
    K = np.log10(K)

    train_auc= clf.cv_results_['mean_train_score']
    train_auc_std= clf.cv_results_['std_train_score']
    cv_auc = clf.cv_results_['mean_test_score']
    cv_auc_std= clf.cv_results_['std_test_score']
    best_a= clf.best_params_
    best_a = list(best_a.values())[0]

    plt.plot(K, train_auc, label='Train AUC')
    # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
    plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

    plt.plot(K, cv_auc, label='CV AUC')
    # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
    plt.gca().fill_between(K, cv_auc - cv_auc_std, cv_auc + cv_auc_std, alpha=0.2, color='darkorange')
    plt.scatter(K, train_auc, label='Train AUC points')
    plt.scatter(K, cv_auc, label='CV AUC points')
    plt.grid(True)
    plt.legend()
    plt.xlabel("K: hyperparameter (in log10())")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
    return best_a
```

Function for picking the best threshold that will give the least fpr

```
In [69]: # we are writing our own function for predict, with defined thresholds
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    import numpy as np
    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
```

Function for confusion matrix for best threshold values

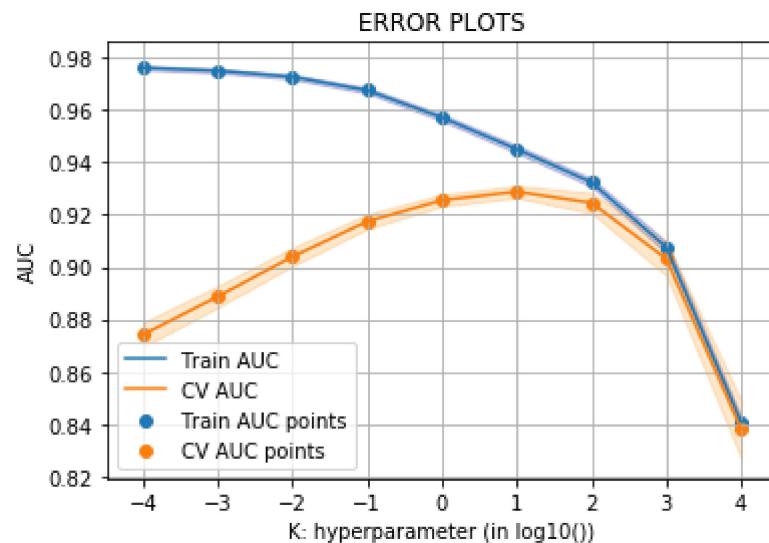
```
In [70]: def conf_matrix(Xtrain,ytrain,Xtest,ytest,best_t,model):
    #https://stackoverflow.com/questions/19233771/scikit-learn-plot-confusion-matrix-with-labels
    import seaborn as sns

    y_train_pred = model.predict_proba(Xtrain)[:,1]
    y_test_pred = model.predict_proba(Xtest)[:,1]
    from sklearn.metrics import confusion_matrix
    print('Train Data')
    cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
    cl = ['Negative', 'Positive']
    confusion_df = pd.DataFrame(cm, index=cl, columns=cl)
    sns.heatmap(confusion_df, annot=True, cbar = False,fmt = 'd'); #annot=True to annotate cells
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
    print('*'*50)
    print('Test Data')
    cm_test= (confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
    confusion_df_test = pd.DataFrame(cm_test, index=cl, columns=cl)
    sns.heatmap(confusion_df_test, annot=True, fmt = 'd'); #annot=True to annotate cells
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

[5.1] Applying Naive Bayes on BOW, SET 1

In [71]: %%time

```
best_alpha = NB(X_train_bow,y_train)
print('Best alpha is ',best_alpha)
```



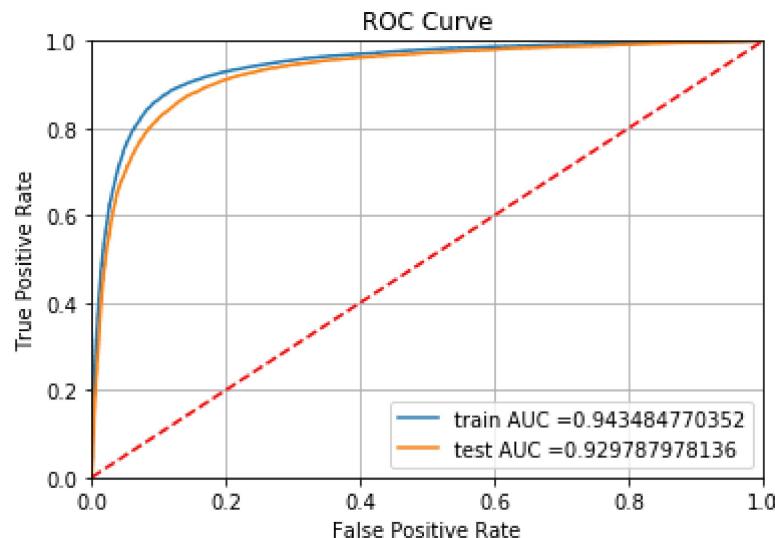
Best alpha is 10

Wall time: 4.57 s

```
In [72]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

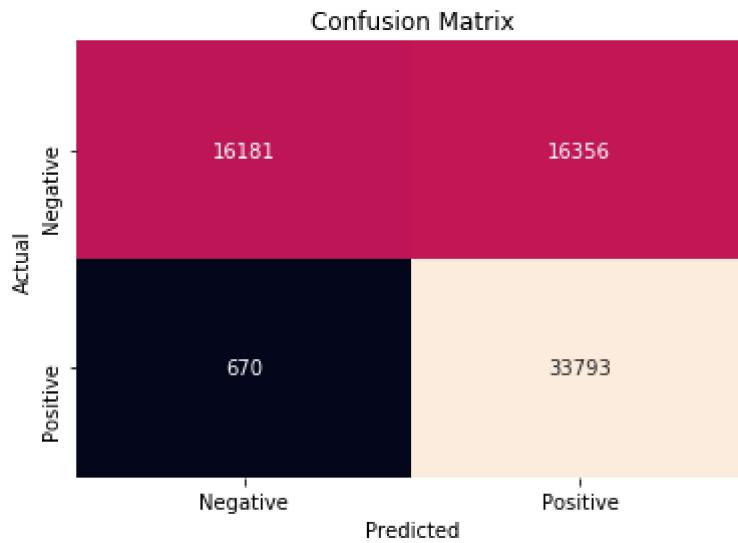
from sklearn.naive_bayes import MultinomialNB
naive = MultinomialNB(alpha=best_alpha,class_prior = [0.5,0.5])
naive.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(y_train, naive.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, naive.predict_proba(X_test_bow)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid(True)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.show()
```

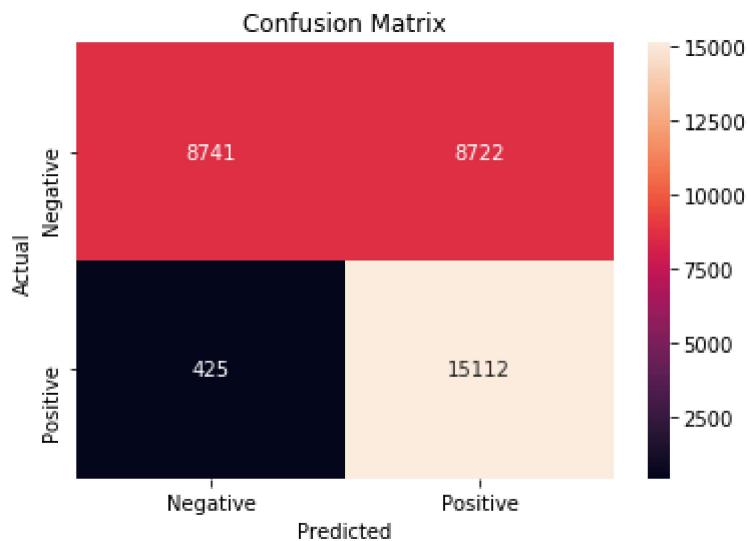


```
In [73]: best_t = find_best_threshold(thresholds, train_fpr, train_tpr)
conf_matrix(X_train_bow,y_train,X_test_bow,y_test,best_t,naive)
```

the maximum value of $tpr * (1 - fpr)$ 0.781875752582 for threshold 0.005
Train Data



=====
Test Data



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

```
In [74]: feature_prob = naive.feature_log_prob_
actual_features = vectorizer.get_feature_names()
feature_prob = pd.DataFrame(feature_prob, columns = actual_features)
feature_prob = feature_prob.T
positive_features = feature_prob[1].sort_values(ascending = False)[0:10]
print("\n\n Top 10 Positive Features:-\n",positive_features)
```

Top 10 Positive Features:-

```
not      -4.047057
like     -4.901421
good     -5.000662
great    -5.076965
one      -5.233155
taste    -5.289811
tea      -5.364296
flavor   -5.406657
love     -5.409227
product  -5.414634
```

Name: 1, dtype: float64

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

```
In [75]: negative_features = feature_prob[0].sort_values(ascending = False)[0:10]
print("Top 10 Negative Features:-\n",negative_features)
```

Top 10 Negative Features:-

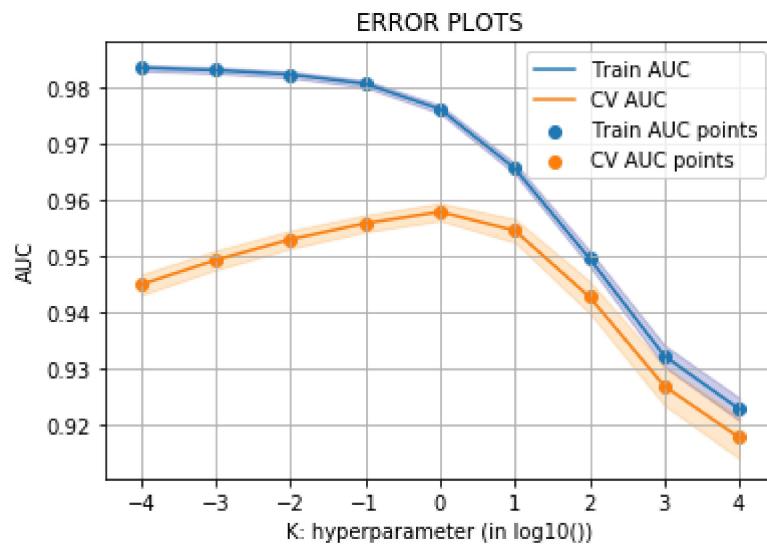
```
not      -3.583062
like     -4.695494
would   -4.969395
taste    -4.980637
product  -4.990442
one      -5.158570
good     -5.415730
flavor   -5.439582
no       -5.454022
coffee   -5.461134
```

Name: 0, dtype: float64

[5.2] Applying Naive Bayes on TFIDF, SET 2

In [76]: %%time

```
best_alpha_tfidf = NB(X_train_tfidf,y_train)
print('best alpha is ',best_alpha_tfidf)
```



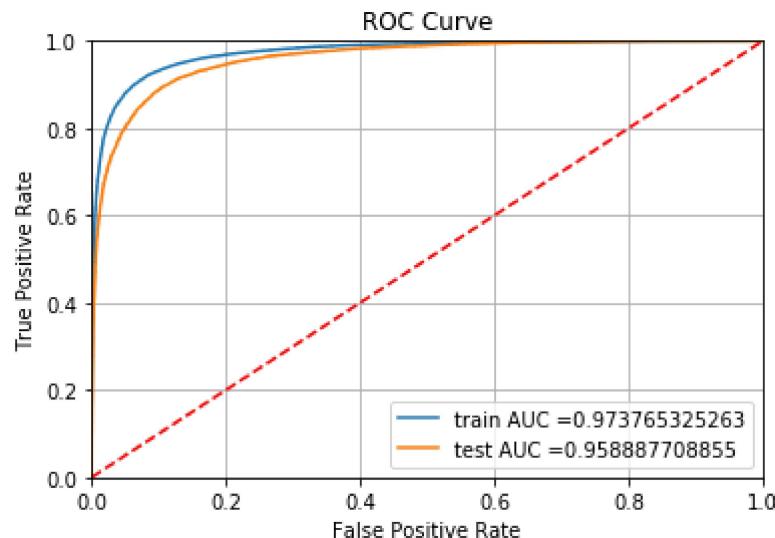
best alpha is 1

Wall time: 4.12 s

```
In [82]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

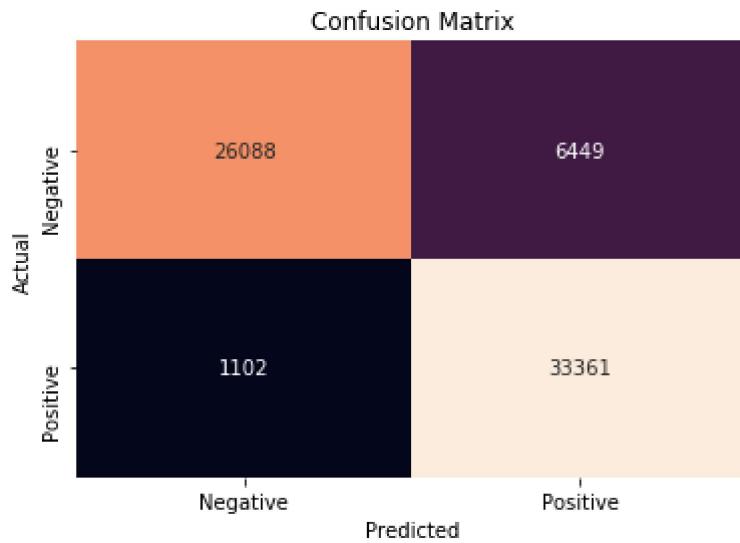
from sklearn.naive_bayes import MultinomialNB
naive_tfidf = MultinomialNB(alpha=best_alpha_tfidf, class_prior=[0.5,0.5])
naive_tfidf.fit(X_train_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(y_train, naive_tfidf.predict_proba(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, naive_tfidf.predict_proba(X_test_tfidf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid(True)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.show()
```

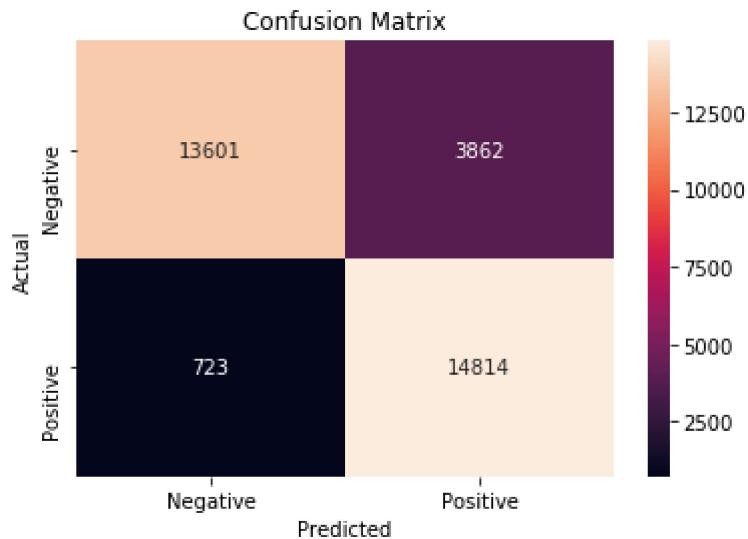


```
In [83]: best_t_tfidf = find_best_threshold(thresholds, train_fpr, train_tpr)
conf_matrix(X_train_tfidf,y_train,X_test_tfidf,y_test,best_t_tfidf,naive_tfidf)
```

the maximum value of $tpr * (1 - fpr)$ 0.844342154368 for threshold 0.348
Train Data



=====
Test Data



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

```
In [79]: feature_prob_tfidf = naive_tfidf.feature_log_prob_
actual_features_tfidf = tf_idf_vect.get_feature_names()
feature_prob_tfidf = pd.DataFrame(feature_prob_tfidf, columns = actual_features_tfidf)
feature_prob_tfidf = feature_prob_tfidf.T
positive_features_tfidf = feature_prob_tfidf[1].sort_values(ascending = False)[0:10]
print("\n\n Top 10 Positive Features:-\n",positive_features_tfidf)
```

Top 10 Positive Features:-

not	-5.561072
great	-5.730049
good	-5.854772
love	-5.988372
tea	-5.988811
like	-6.007123
coffee	-6.071754
product	-6.217123
flavor	-6.219069
one	-6.219920

Name: 1, dtype: float64

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

```
In [81]: negative_features_tfidf = feature_prob_tfidf[0].sort_values(ascending = False)[:10]
print("Top 10 Negative Features:-\n",negative_features_tfidf)
```

Top 10 Negative Features:-

not	-5.010610
like	-5.756092
product	-5.885677
taste	-5.887781
would	-5.934477
coffee	-6.127767
one	-6.133847
flavor	-6.266166
no	-6.295231
good	-6.338839

Name: 0, dtype: float64

[6] Conclusions

```
In [84]: # Creating table using PrettyTable library
from prettytable import PrettyTable
vectorizer = ['BOW','TFIDF']
AUC = [0.92,0.95]
Model = ['MultinomialNB','MultinomialNB']
hyperparameter = [10,1]
sno = [1,2]
# Initializing prettytable
ptable = PrettyTable()
# Adding columns
ptable.add_column("S.NO.",sno)
ptable.add_column("Vectorizer",vectorizer)
ptable.add_column("Model",Model)
ptable.add_column("Best Alpha",hyperparameter)
ptable.add_column("Test AUC",AUC)
# Printing the Table
print(ptable)
```

S.NO.	Vectorizer	Model	Best Alpha	Test AUC
1	BOW	MultinomialNB	10	0.92
2	TFIDF	MultinomialNB	1	0.95

```
In [ ]: !jupyter nbconvert
```