# 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 [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.model_selection import TimeSeriesSplit

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]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clien
t_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.co
m&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2
F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.co
m%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.re
adonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&respon
se_type=code (https://accounts.google.com/o/oauth2/auth?client_id=9473189898
03-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=
urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleap
is.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%
20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3
A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code)

Enter your authorization code:
··········
Mounted at /content/drive
```

In [34]:

```
# using SQLite Table to read data.
con = sqlite3.connect('drive/My Drive/FFRDB/database.sqlite')

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

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

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

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

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

```
Number of data points in our data (525814, 10)
```

Out[34]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenom |
|---|---|---|---|---|---|---|
| **0** | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | |
| **1** | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | |
| **2** | 3 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia Corres" | 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 [36]:

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

(80668, 7)

Out[36]:

| | 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 [37]:

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

Out[37]:

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

In [38]:

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

Out[38]:

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 [39]:

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

Out[39]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDen |
|---|---|---|---|---|---|---|
| **0** | 78445 | B000HDL1RQ | AR5J8UI46CURR | Geetha Krishnan | 2 | |
| **1** | 138317 | B000HDOPYC | AR5J8UI46CURR | Geetha Krishnan | 2 | |
| **2** | 138277 | B000HDOPYM | AR5J8UI46CURR | Geetha Krishnan | 2 | |
| **3** | 73791 | B000HDOPZG | AR5J8UI46CURR | Geetha Krishnan | 2 | |
| **4** | 155049 | B000PAQ75C | AR5J8UI46CURR | Geetha Krishnan | 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 delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [0]:

```
#Sorting data according to ProductId in ascending order

sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fal
```

In [41]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=

# sorting data based on time and taking first 100000 observations
final=final.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort
final=final[:100000]
final.shape
```

Out[41]:

```
(100000, 10)
```

In [42]:

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

Out[42]:

```
19.01813188694101
```

**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 [43]:

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

display.head()
```

Out[43]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDen |
|---|---|---|---|---|---|---|
| **0** | 64422 | B000MIDROQ | A161DK06JJMCYF | J. E. Stephens "Jeanne" | 3 | |
| **1** | 44737 | B001EQ55RW | A2V0I904FH7ABY | Ram | 3 | |

In [0]:

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

In [45]:

```
#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()
```

```
(99998, 10)
```

Out[45]:

```
1    87729
0    12269
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 [46]:

```
# 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 witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
==================================================
I finally ordered a couple products from this seller for myself(not as gift
s) and I am really happy. This Jade Bonsai is really cool and it arrived fas
t and in perfect condition. It's in my living room and I get tons of complim
ents. It's already grown some too and the pot it came in is really nice, loo
ks expensive! Much bigger than I thought it would be even.  Thanks again!!
==================================================
I bought some of this tea when I was in Seattle and I have been dying to get
more.  It really is the best tea I have ever had.  It is great hot or cold.
==================================================
I would prefer freshly made brown rice, but that takes a long time to make a
nd isn't easy. This makes it convenient, and takes all the guess work out of
making it. I generally have been buying frozen organic brown rice, but that
takes up lots of freezer space. The fact that this is easy to store at room
temperature is a big plus. I'll be buying more.
==================================================
```

In [47]:

```
# 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 witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
```

In [48]:

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

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

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

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

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

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
==================================================
I finally ordered a couple products from this seller for myself(not as gift
s) and I am really happy. This Jade Bonsai is really cool and it arrived fas
t and in perfect condition. It's in my living room and I get tons of complim
ents. It's already grown some too and the pot it came in is really nice, loo
ks expensive! Much bigger than I thought it would be even.  Thanks again!!
==================================================
I bought some of this tea when I was in Seattle and I have been dying to get
more.  It really is the best tea I have ever had.  It is great hot or cold.
==================================================
I would prefer freshly made brown rice, but that takes a long time to make a
nd isn't easy. This makes it convenient, and takes all the guess work out of
making it. I generally have been buying frozen organic brown rice, but that
takes up lots of freezer space. The fact that this is easy to store at room
temperature is a big plus. I'll be buying more.
```

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 [50]:

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

```
I bought some of this tea when I was in Seattle and I have been dying to get
more.  It really is the best tea I have ever had.  It is great hot or cold.
==================================================
```

In [51]:

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

```
this witty little book makes my son laugh at loud. i recite it in the car as
we're driving along and he always can sing the refrain. he's learned about w
hales, India, drooping roses:  i love all the new words this book  introduce
s and the silliness of it all.  this is a classic book i am  willing to bet
my son will STILL be able to recite from memory when he is  in college
```

In [52]:

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

```
I bought some of this tea when I was in Seattle and I have been dying to get
more It really is the best tea I have ever had It is great hot or cold
```

In [0]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "t
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", '
            'won', "won't", 'wouldn', "wouldn't"])
```

In [54]:

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

```
100%|██████████| 99998/99998 [00:39<00:00, 2532.56it/s]
```

In [55]:

```
preprocessed_reviews[1500]
```

Out[55]:

```
'bought tea seattle dying get really best tea ever great hot cold'
```

# [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)](https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

# Applying Multinomial Naive Bayes

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

In [118]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

## splitting with shuffle as false so time consistency is maintained as data is sorted

X_train_bow, X_test_bow, y_train_bow, y_test_bow = train_test_split(preprocessed_revie

BOW = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=50000)
BOW.fit_transform(X_train_bow)
X_train_bow = BOW.transform(X_train_bow)
X_test_bow = BOW.transform(X_test_bow)

scaler = StandardScaler(with_mean = False)
X_train_bow = scaler.fit_transform(X_train_bow)
X_test_bow = scaler.transform(X_test_bow)

print(X_train_bow.shape)
print(X_test_bow.shape)
```

```
(69998, 39325)
(30000, 39325)
```

In [119]:

```
#generating random alpha values between 10^-5 to 10^5
from numpy import random
alpha=list()
alpha.extend(list(map(lambda x,y:y*10**x,list(range(-5,5)),list(random.randint(0,9,siz
alpha.extend(list(map(lambda x,y:y*10**x,list(range(-5,5)),list(random.randint(0,9,siz
alpha.extend(list(map(lambda x,y:y*10**x,list(range(-5,5)),list(random.randint(0,9,siz
alpha.sort()
alpha
```

Out[119]:

```
[0.0,
 0,
 0.0,
 0,
 3.0000000000000004e-05,
 4e-05,
 8e-05,
 0.00030000000000000003,
 0.0006000000000000001,
 0.001,
 0.003,
 0.007,
 0.01,
 0.04,
 0.07,
 0.1,
 0.6000000000000001,
 2,
 4,
 6,
 10,
 30,
 200,
 700,
 800,
 3000,
 7000,
 10000,
 50000,
 60000]
```

In [120]:

```
#removing 0 as value of alpha
while True:
  if 0 in alpha:
    alpha.pop(0)
    continue
  break
alpha
```

Out[120]:

```
[3.0000000000000004e-05,
 4e-05,
 8e-05,
 0.00030000000000000003,
 0.0006000000000000001,
 0.001,
 0.003,
 0.007,
 0.01,
 0.04,
 0.07,
 0.1,
 0.6000000000000001,
 2,
 4,
 6,
 10,
 30,
 200,
 700,
 800,
 3000,
 7000,
 10000,
 50000,
 60000]
```

In [121]:

```
from sklearn.naive_bayes import MultinomialNB as MNB
from sklearn.model_selection import GridSearchCV

param={'alpha':alpha}
model = MNB(class_prior=[0.5,0.5])
gscv=GridSearchCV(model,param,cv=10,verbose=5,scoring='roc_auc',return_train_score=Tru
gscv.fit(X_train_bow,y_train_bow)
```

```
Fitting 10 folds for each of 26 candidates, totalling 260 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed:    2.0s
[Parallel(n_jobs=-1)]: Done 132 tasks      | elapsed:   10.5s
[Parallel(n_jobs=-1)]: Done 260 out of 260 | elapsed:   20.4s finished
```

Out[121]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=MultinomialNB(alpha=1.0, class_prior=[0.5, 0.5],
                                     fit_prior=True),
             iid='warn', n_jobs=-1,
             param_grid={'alpha': [3.0000000000000004e-05, 4e-05, 8e-05,
                                   0.00030000000000000003,
                                   0.0006000000000000001, 0.001, 0.003, 0.00
7,
                                   0.01, 0.04, 0.07, 0.1, 0.600000000000000
1, 2,
                                   4, 6, 10, 30, 200, 700, 800, 3000, 7000,
                                   10000, 50000, 60000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

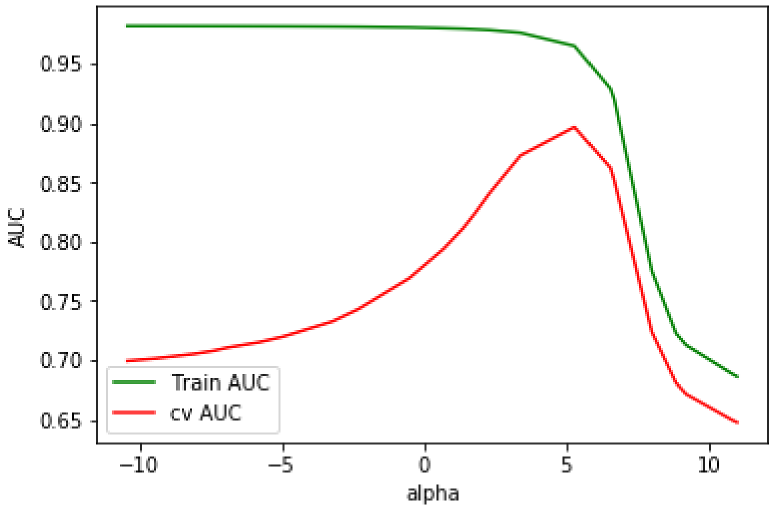
In [122]:

```
print("Best HyperParameter: ",gscv.best_params_)
Best_HyperParameter=gscv.best_params_

lines=plt.plot(np.log(param['alpha']),gscv.cv_results_['mean_train_score'],np.log(para
plt.setp(lines[0],color='g',label='Train AUC')
plt.setp(lines[1],color='r',label='cv AUC')

plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.show()
```

```
Best HyperParameter:  {'alpha': 200}
```



In [123]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,precision_score,r

MODEL = MNB(alpha=Best_HyperParameter['alpha'],class_prior=[0.5,0.5])
MODEL.fit(X_train_bow,y_train_bow)
y_pred_bow = MODEL.predict(X_test_bow)
y_pred_tr = MODEL.predict(X_train_bow)

acc=accuracy_score(y_test_bow, y_pred_bow)*100
ps=precision_score(y_test_bow, y_pred_bow)*100
rc=recall_score(y_test_bow, y_pred_bow)*100
f1=f1_score(y_test_bow, y_pred_bow)*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

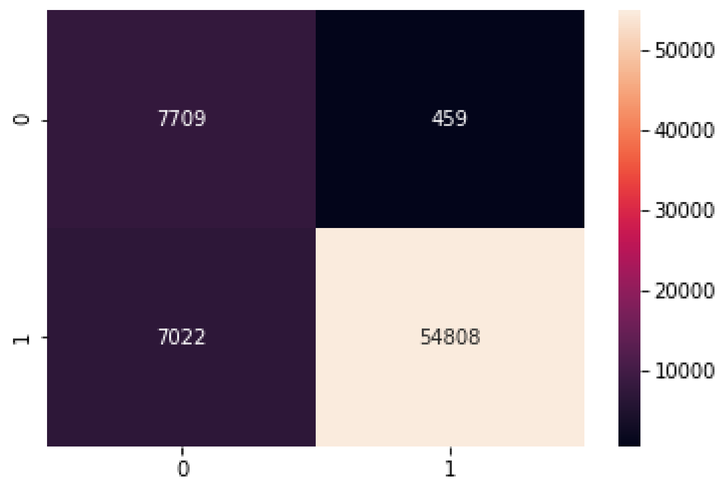
```
Accuracy on test set: 86.61%
Precision on test set: 97.15%
recall score on test set: 87.05%
f1 score on test set: 91.82%
```

In [124]:

```
cm = pd.DataFrame(confusion_matrix(y_train_bow, y_pred_tr), range(2),range(2))
sns.heatmap(cm, annot=True,fmt='g')
```

Out[124]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f967a96e320>
```

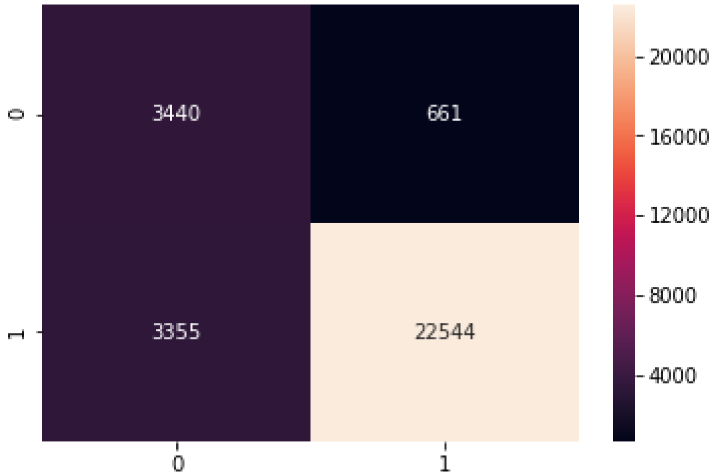


In [125]:

```
cm = pd.DataFrame(confusion_matrix(y_test_bow, y_pred_bow), range(2),range(2))
sns.heatmap(cm, annot=True,fmt='g')
```

Out[125]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f966ddeea90>
```

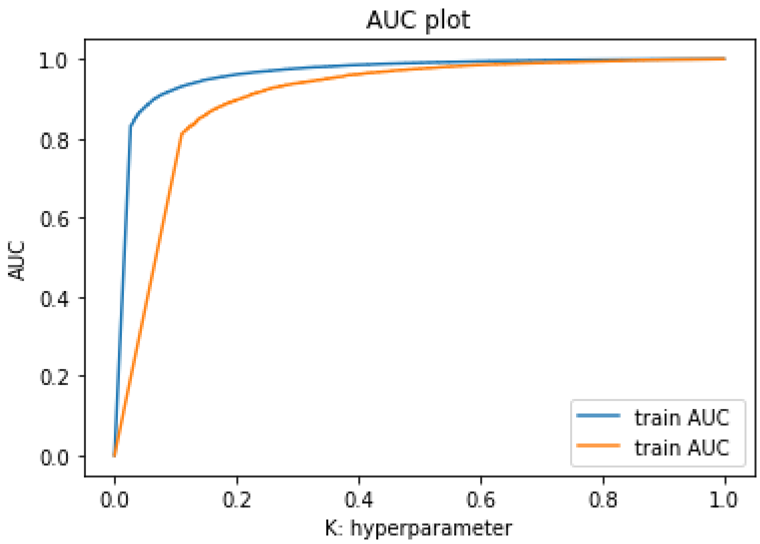


In [126]:

```
from sklearn.metrics import roc_auc_score
train_fpr, train_tpr, thresholds = roc_curve(y_train_bow, MODEL.predict_proba(X_train_
test_fpr, test_tpr, thresholds = roc_curve(y_test_bow, MODEL.predict_proba(X_test_bow)
print("train auc ={}".format(roc_auc_score(y_train_bow, MODEL.predict_proba(X_train_bo
print("test auc={}".format(roc_auc_score(y_test_bow, MODEL.predict_proba(X_test_bow)[:

plt.plot(train_fpr, train_tpr, label="train AUC ")
plt.plot(test_fpr, test_tpr, label="train AUC ")
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC plot")
plt.show()
```

```
train auc =0.9631827341896512
test auc=0.9015188321967882
```




## [5.1.1] Top 10 important features

In [127]:

```
neg_sorted = MODEL.feature_log_prob_[0, :].argsort()
pos_sorted = MODEL.feature_log_prob_[1, :].argsort()

print("Important features of Negative class\n")
print(np.take(BOW.get_feature_names(), neg_sorted[-10:]))
print("\n \n")
print("Important features of Positive class \n")
print(np.take(BOW.get_feature_names(), pos_sorted[-10:]))
print("\n \n")
```

```
Important features of Negative class

['would not' 'money' 'disappointed' 'bad' 'one' 'taste' 'product' 'like'
 'would' 'not']



Important features of Positive class

['product' 'flavor' 'best' 'taste' 'one' 'love' 'like' 'good' 'great'
 'not']
```

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

In [138]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split


## splitting with shuffle as false so time consistency is maintained as data is sorted

X_train_tfidf, X_test_tfidf, y_train_tfidf, y_test_tfidf = train_test_split(preprocess

tfidf = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tfidf.fit_transform(X_train_tfidf)
X_train_tfidf = tfidf.transform(X_train_tfidf)
X_test_tfidf = tfidf.transform(X_test_tfidf)

scaler = StandardScaler(with_mean = False)
X_train_tfidf = scaler.fit_transform(X_train_tfidf)
X_test_tfidf = scaler.transform(X_test_tfidf)

print(X_train_tfidf.shape)
print(X_test_tfidf.shape)
```

```
(69998, 39325)
(30000, 39325)
```

In [139]:

```
from numpy import random
alpha=list()
alpha.extend(list(map(lambda x,y:y*10**x,list(range(-5,5)),list(random.randint(0,9,siz
alpha.extend(list(map(lambda x,y:y*10**x,list(range(-5,5)),list(random.randint(0,9,siz
alpha.extend(list(map(lambda x,y:y*10**x,list(range(-5,5)),list(random.randint(0,9,siz
alpha.sort()
alpha
```

Out[139]:

```
[0.0,
 0.0,
 2e-05,
 4e-05,
 8e-05,
 0.0002,
 0.0006000000000000001,
 0.0008,
 0.005,
 0.006,
 0.01,
 0.01,
 0.08,
 0.30000000000000004,
 0.30000000000000004,
 1,
 2,
 5,
 10,
 50,
 50,
 300,
 500,
 800,
 4000,
 5000,
 6000,
 20000,
 30000,
 80000]
```

In [140]:

```
#removing 0 as value of alpha
while True:
  if 0 in alpha:
    alpha.pop(0)
    continue
  break
alpha

```

Out[140]:

```
[2e-05,
 4e-05,
 8e-05,
 0.0002,
 0.0006000000000000001,
 0.0008,
 0.005,
 0.006,
 0.01,
 0.01,
 0.08,
 0.30000000000000004,
 0.30000000000000004,
 1,
 2,
 5,
 10,
 50,
 50,
 300,
 500,
 800,
 4000,
 5000,
 6000,
 20000,
 30000,
 80000]
```

In [141]:

```
from sklearn.naive_bayes import MultinomialNB as MNB
from sklearn.model_selection import GridSearchCV

param={'alpha':alpha}
model = MNB(class_prior=[0.5,0.5])
gscv=GridSearchCV(model,param,cv=10,verbose=5,scoring='roc_auc',return_train_score=Tru
gscv.fit(X_train_tfidf,y_train_tfidf)
```

```
Fitting 10 folds for each of 28 candidates, totalling 280 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:    5.6s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:   13.0s
[Parallel(n_jobs=-1)]: Done 280 out of 280 | elapsed:   23.1s finished
```

Out[141]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=MultinomialNB(alpha=1.0, class_prior=[0.5, 0.5],
                                     fit_prior=True),
             iid='warn', n_jobs=-1,
             param_grid={'alpha': [2e-05, 4e-05, 8e-05, 0.0002,
                                   0.0006000000000000001, 0.0008, 0.005, 0.0
06,
                                   0.01, 0.01, 0.08, 0.30000000000000004,
                                   0.30000000000000004, 1, 2, 5, 10, 50, 50,
                                   300, 500, 800, 4000, 5000, 6000, 20000,
                                   30000, 80000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

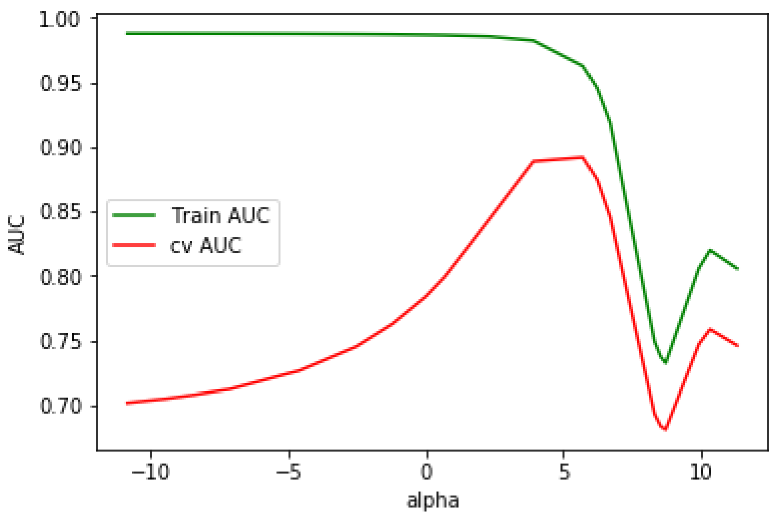
In [142]:

```
print("Best HyperParameter: ",gscv.best_params_)
Best_HyperParameter=gscv.best_params_

lines=plt.plot(np.log(param['alpha']),gscv.cv_results_['mean_train_score'],np.log(para
plt.setp(lines[0],color='g',label='Train AUC')
plt.setp(lines[1],color='r',label='cv AUC')

plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.show()
```

```
Best HyperParameter:  {'alpha': 300}
```



In [143]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,precision_score,r

MODEL = MNB(alpha=Best_HyperParameter['alpha'],class_prior=[0.5,0.5])
MODEL.fit(X_train_tfidf,y_train_tfidf)
y_pred_tfidf = MODEL.predict(X_test_tfidf)
y_pred_tr = MODEL.predict(X_train_tfidf)

acc=accuracy_score(y_test_tfidf, y_pred_tfidf)*100
ps=precision_score(y_test_tfidf, y_pred_tfidf)*100
rc=recall_score(y_test_tfidf, y_pred_tfidf)*100
f1=f1_score(y_test_tfidf, y_pred_tfidf)*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

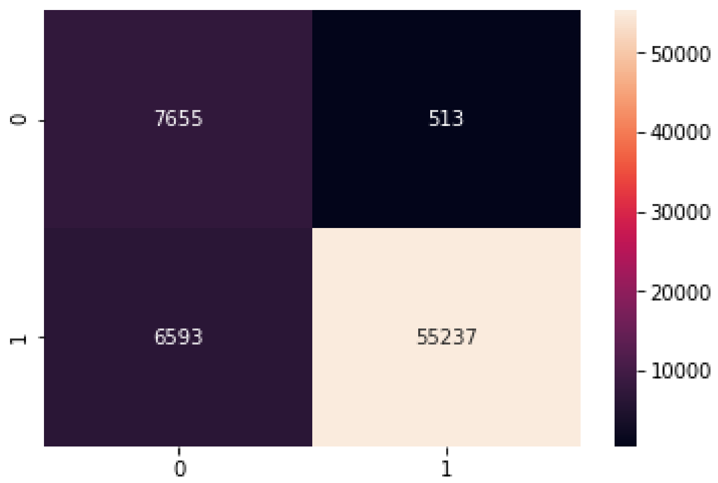
```
Accuracy on test set: 86.76%
Precision on test set: 96.77%
recall score on test set: 87.59%
f1 score on test set: 91.95%
```

In [144]:

```
cm = pd.DataFrame(confusion_matrix(y_train_tfidf, y_pred_tr), range(2),range(2))
sns.heatmap(cm, annot=True,fmt='g')
```

Out[144]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9679d575c0>
```

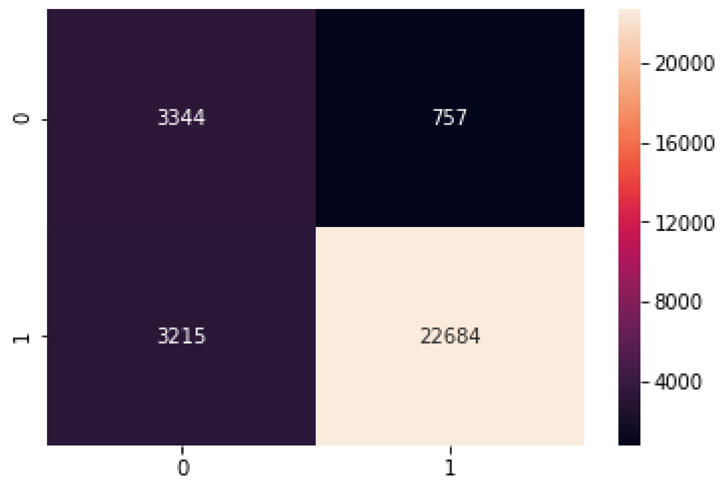


In [145]:

```
cm = pd.DataFrame(confusion_matrix(y_test_tfidf, y_pred_tfidf), range(2),range(2))
sns.heatmap(cm, annot=True,fmt='g')
```

Out[145]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f966b064ba8>
```

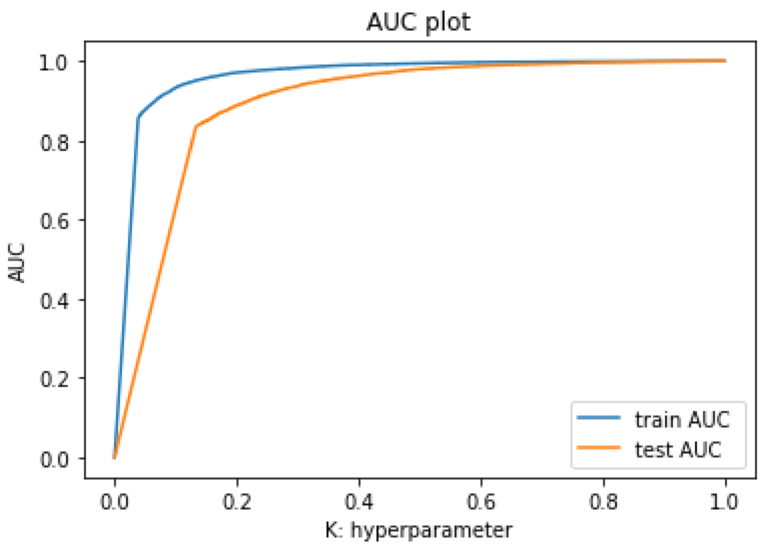


In [146]:

```
from sklearn.metrics import roc_auc_score
train_fpr, train_tpr, thresholds = roc_curve(y_train_tfidf, MODEL.predict_proba(X_trai
test_fpr, test_tpr, thresholds = roc_curve(y_test_tfidf, MODEL.predict_proba(X_test_tf
print("train auc ={}".format(roc_auc_score(y_train_tfidf, MODEL.predict_proba(X_train_
print("test auc={}".format(roc_auc_score(y_test_tfidf, MODEL.predict_proba(X_test_tfid

plt.plot(train_fpr, train_tpr, label="train AUC ")
plt.plot(test_fpr, test_tpr, label="test AUC ")
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC plot")
plt.show()
```

```
train auc =0.9619462914727959
test auc=0.893605219887105
```




## [5.2.1] Top 10 important features

In [147]:

```
neg_sorted = MODEL.feature_log_prob_[0, :].argsort()
pos_sorted = MODEL.feature_log_prob_[1, :].argsort()

print("Important features of Negative class\n")
print(np.take(tfidf.get_feature_names(), neg_sorted[-10:]))
print("\n \n")
print("Important features of Positive class \n")
print(np.take(tfidf.get_feature_names(), pos_sorted[-10:]))
print("\n \n")

```

```
Important features of Negative class

['money' 'would not' 'disappointed' 'bad' 'one' 'product' 'taste' 'like'
 'would' 'not']



Important features of Positive class

['best' 'product' 'flavor' 'love' 'taste' 'one' 'like' 'good' 'great'
 'not']
```

# [6] Conclusions

In [154]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["S.NO", "VECTORIZER","BEST ALPHA", "TEST AUC", "F1 SCORE"]

x.add_row(["1",'BOW',200,0.9015,91.82])
x.add_row(["2",'TFIDF',300,0.8936,91.95])

print(x)
```

```
+------+------------+------------+----------+----------+
| S.NO | VECTORIZER | BEST ALPHA | TEST AUC | F1 SCORE |
+------+------------+------------+----------+----------+
|  1   |    BOW     |    200     |  0.9015  |  91.82   |
|  2   |   TFIDF    |    300     |  0.8936  |  91.95   |
+------+------------+------------+----------+----------+
```

# 1. Naive bayes can be used a benchmark for high dimentional data as being used here.

# 2. Naive bayes has very low time complexcity

# 3. BOW vectoriztion is marginaly better than TFIDF vectorization

# 4. Test AUC and F1 score is even is similar