Assignment04 - Amazon Fine Food Reviews Analysis_NaiveBayes

May 16, 2019

1 Amazon Fine Food Reviews Analysis

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

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective: Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

2 [1]. Reading Data

2.1 [1.1] Loading the data

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

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

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

```
In [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.
        db_path = '/home/monodeepdas112/Datasets/amazon-fine-food-reviews/database.sqlite'
        con = sqlite3.connect(db_path)

        # 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 LIMIT 100

        # 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 (100000, 10)
```

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

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

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

```
In [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  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U    Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

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

```
Out[5]:               UserId   ProductId                      ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                               Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

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

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

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

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

```
In [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  \
        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
```

```
   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
1                       2      5  1199577600
2                       2      5  1199577600
3                       2      5  1199577600
4                       2      5  1199577600

                             Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                               Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

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

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

```
Out[9]: (87775, 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]: 87.775
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

```
In [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  \
         0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
         1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

            HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
         0                     3                       1      5  1224892800
         1                     3                       2      4  1212883200

                                              Summary  \
         0            Bought This for My Son at College
         1  Pure cocoa taste with crunchy almonds inside

                                                         Text
         0  My son loves spaghetti so I didn't hesitate or...
         1  It was almost a 'love at first bite' - the per...
```

```
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()
```

```
(87773, 10)
```

```
Out[13]: 1    73592
         0    14181
         Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

```
In [14]: # printing some random reviews
         sent_0 = final['Text'].values[0]
         print(sent_0)
         print("="*50)

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
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
==================================================
```

```
In [15]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
         sent_0 = re.sub(r"http\S+", "", sent_0)
         sent_1000 = re.sub(r"http\S+", "", sent_1000)
         sent_150 = re.sub(r"http\S+", "", sent_1500)
         sent_4900 = re.sub(r"http\S+", "", sent_4900)

         print(sent_0)
```

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

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

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
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
```

```
In [17]: # 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)

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


In [19]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)

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


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

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


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 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', "
                     'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', '
                     'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
                     'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
                     'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
                     'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
                     'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
                     's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
                     've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
                     "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
                     "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                     'won', "won't", 'wouldn', "wouldn't"])

In [22]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_reviews = []
         # tqdm is for printing the status bar
         for sentance in tqdm(final['Text'].values):
             sentance = re.sub(r"http\S+", "", sentance)
             sentance = BeautifulSoup(sentance, 'lxml').get_text()
             sentance = decontracted(sentance)
             sentance = re.sub("\S*\d\S*", "", sentance).strip()
             sentance = re.sub('[^A-Za-z]+', ' ', sentance)
             # https://gist.github.com/sebleier/554280
             sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwo
             preprocessed_reviews.append(sentance.strip())
```

```
100%|| 87773/87773 [00:45<00:00, 1910.01it/s]
```

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

```
Out[23]: 'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

```
In [24]: ## Similartly you can do preprocessing for review summary also.
         from tqdm import tqdm
         preprocessed_review_summaries = []
         # 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_review_summaries.append(sentance.strip())
```

```
100%|| 87773/87773 [00:38<00:00, 2293.78it/s]
```

5 [4] Featurization

5.1 [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])
```

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

```
In [26]: # #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/m

         # # you can choose these numebrs min_df=10, max_features=5000, of your choice
         # count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
         # final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
         # print("the type of count vectorizer ",type(final_bigram_counts))
         # print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
         # print("the number of unique words including both unigrams and bigrams ", final_bigr
```

5.3 [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_na
         # 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_i
```

5.4 [4.4] Word2Vec

```
In [28]: # # Train your own Word2Vec model using your own text corpus
         # i=0
         # list_of_sentance=[]
         # for sentance in preprocessed_reviews:
         #     list_of_sentance.append(sentance.split())
```

```
In [29]: # # Using Google News Word2Vectors

         # # in this project we are using a pretrained model by google
         # # its 3.3G file, once you load this into your memory
         # # it occupies ~9Gb, so please do this step only if you have >12G of ram
         # # we will provide a pickle file wich contains a dict ,
         # # and it contains all our courpus words as keys and  model[word] as values
         # # To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
         # # from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
         # # it's 1.9GB in size.


         # # http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
         # # you can comment this whole cell
         # # or change these varible according to your need

         # is_your_ram_gt_16g=False
         # want_to_use_google_w2v = False
         # want_to_train_w2v = True

         # if want_to_train_w2v:
         #     # min_count = 5 considers only words that occured atleast 5 times
         #     w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
         #     print(w2v_model.wv.most_similar('great'))
         #     print('='*50)
         #     print(w2v_model.wv.most_similar('worst'))

         # elif want_to_use_google_w2v and is_your_ram_gt_16g:
         #     if os.path.isfile('GoogleNews-vectors-negative300.bin'):
         #         w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300
         #         print(w2v_model.wv.most_similar('great'))
         #         print(w2v_model.wv.most_similar('worst'))
         #     else:
         #         print("you don't have gogole's word2vec file, keep want_to_train_w2v = True
```

In [30]:
```
# w2v_words = list(w2v_model.wv.vocab)
# print("number of words that occured minimum 5 times ",len(w2v_words))
# print("sample words ", w2v_words[0:50])
```

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

[4.4.1.1] Avg W2v

In [31]:
```
# # average Word2Vec
# # compute average word2vec for each review.
# sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
# for sent in tqdm(list_of_sentance): # for each review/sentence
#     sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need
#     cnt_words =0; # num of words with a valid vector in the sentence/review
#     for word in sent: # for each word in a review/sentence
#         if word in w2v_words:
#             vec = w2v_model.wv[word]
#             sent_vec += vec
#             cnt_words += 1
#     if cnt_words != 0:
#         sent_vec /= cnt_words
#     sent_vectors.append(sent_vec)
# print(len(sent_vectors))
# print(len(sent_vectors[0]))
```

[4.4.1.2] TFIDF weighted W2v

```
In [32]: # # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
         # model = TfidfVectorizer()
         # tf_idf_matrix = model.fit_transform(preprocessed_reviews)
         # # we are converting a dictionary with word as a key, and the idf as a value
         # dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [33]: # # TF-IDF weighted Word2Vec
         # tfidf_feat = model.get_feature_names() # tfidf words/col-names
         # # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfi

         # tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this
         # row=0;
         # for sent in tqdm(list_of_sentance): # for each review/sentence
         #     sent_vec = np.zeros(50) # as word vectors are of zero length
         #     weight_sum =0; # num of words with a valid vector in the sentence/review
         #     for word in sent: # for each word in a review/sentence
         #         if word in w2v_words and word in tfidf_feat:
         #             vec = w2v_model.wv[word]
         # #               tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
         #             # to reduce the computation we are
         #             # dictionary[word] = idf value of word in whole courpus
         #             # sent.count(word) = tf valeus of word in this review
         #             tf_idf = dictionary[word]*(sent.count(word)/len(sent))
         #             sent_vec += (vec * tf_idf)
         #             weight_sum += tf_idf
         #     if weight_sum != 0:
         #         sent_vec /= weight_sum
         #     tfidf_sent_vectors.append(sent_vec)
         #     row += 1
```

6 [5] Assignment 4: Apply Naive Bayes

```
<li><strong>Apply Multinomial NaiveBayes on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vector
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning(find best Alpha)</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001</
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
```

```
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>Find the top 10 features of positive class and top 10 features of negative class for both i
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engine
        <ul>
        <li>Taking length of reviews as another feature.</li>
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

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

7 Applying Multinomial Naive Bayes

```
In [34]: from sklearn.model_selection import train_test_split
         from sklearn.metrics import confusion_matrix
```

```
        from sklearn.metrics import roc_curve, auc
        from sklearn.metrics import roc_auc_score
        import pprint
        from sklearn.pipeline import Pipeline
        import os.path
        import pickle
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.model_selection import RandomizedSearchCV
        from random import randint
        from sklearn.model_selection import StratifiedKFold
        from prettytable import PrettyTable
```

7.1 [5.1] Applying Naive Bayes on BOW, SET 1

7.1.1 [5.0.0] Splitting up the Dataset into D_train and D_test

```
In [35]: Dx_train, Dx_test, Dy_train, Dy_test = train_test_split(preprocessed_reviews[:100000]
```

```
In [36]: prettytable_data = []
```

7.1.2 [5.0.1] Defining some functions to increase code reusability and readability

```
In [37]: '''Initializing and training the vectorizer'''
        def get_vectorizer(vectorizer, data):
            if(vectorizer=='BOW'):
                vectorizer = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
            if(vectorizer=='TFIDF'):
                vectorizer=TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
            vectorizer.fit(data)
            return vectorizer
```

```
In [38]: def perform_hyperparameter_tuning(X, Y, vectorizer, vec_name):
            #If the pandas dataframe with the hyperparameter info exists then return it
            results_name = 'saved_models/Assignment4-Results/{0}_multi_nb_results.csv'.format
            if(os.path.exists(results_name)):
                return pd.read_csv(results_name)

            #else perform hyperparameter tuning
            parameters_grid = {
                'nb__alpha' : [0.00001, 0.000025, 0.00005, 0.000075, 0.0001, 0.00025, 0.0005,
                               0.0075, 0.01, 0.025, 0.05, 0.075, 0.1, 0.025, 0.5, 0.075,
                               1, 2.5, 5, 7.5, 10, 25, 50, 75, 100, 250, 500, 750, 1000, 2500
            }

            alpha = []
            train_scores = []
            test_scores = []

            train_mean_score = []
```

```
test_mean_score = []

# Initializing KFold
skf = StratifiedKFold(n_splits=10)
X = np.array(X)
Y = np.array(Y)

for _alpha in tqdm(parameters_grid['nb__alpha']):
    #Performing Cross Validation
    for train_index, test_index in skf.split(X, Y):
        Dx_train, Dx_cv = X[train_index], X[test_index]
        Dy_train, Dy_cv = Y[train_index], Y[test_index]

        #Initializing the Vectorizer
        vectorizer = get_vectorizer(vectorizer, Dx_train.tolist())

        #Transforming the data to features
        x_train = vectorizer.transform(Dx_train.tolist())
        x_cv = vectorizer.transform(Dx_cv.tolist())

        #Initializing the MultinomialNB model
        mnb = MultinomialNB(fit_prior=True, class_prior=None, alpha=_alpha)

        #Training the model
        mnb.fit(x_train, Dy_train)

        #Prediction
        train_results = mnb.predict_proba(x_train)
        cv_results = mnb.predict_proba(x_cv)

        try:
            train_score = roc_auc_score(Dy_train, train_results[:, 1])
            test_score = roc_auc_score(Dy_cv, cv_results[:, 1])

            #storing the results to form a dataframe
            train_scores.append(train_score)
            test_scores.append(test_score)

        except Exception as e:
            print('Error Case : ', e)
            print(('Actual, Predicted'))
            [print((Dy_cv[i], cv_results[i, 1])) for i in range(len(Dy_cv))]


    train_mean_score.append(sum(train_scores)/len(train_scores))
    test_mean_score.append(sum(test_scores)/len(test_scores))
    alpha.append(_alpha)
```

```
                 train_scores = []
                 test_scores = []

             results_df = pd.DataFrame({'alpha' : alpha, 'train_score' : train_mean_score,
                                        'cv_score': test_mean_score})
             #writing the results to csv after performing hyperparameter tuning
             results_df.to_csv(results_name)
             return results_df
```

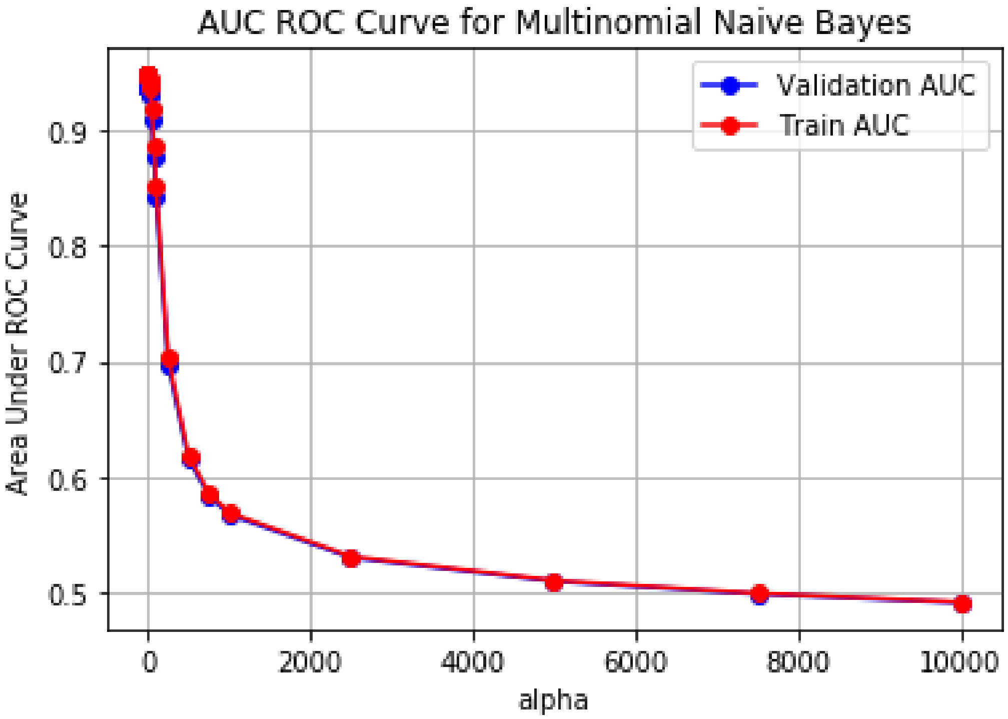
```
In [39]: def analyse_results(df):
             # Sorting the dataframe by the number of neighbours
             df = df.sort_values(by=['alpha', 'cv_score'], ascending=[True, False])

             #plotting the uniform weighted measure K-NN results
             fig = plt.figure()
             ax = fig.gca()
             plt.plot(df.alpha, df.cv_score, '-o', c='b', label='Validation AUC')
             plt.plot(df.alpha, df.train_score, '-o', c='r', label='Train AUC')
             plt.grid(True)
             plt.xlabel('alpha')
             plt.ylabel('Area Under ROC Curve')
             plt.title('AUC ROC Curve for Multinomial Naive Bayes')
             plt.legend(loc='best')
             plt.show()

         def selecting_best_hyperparameters(df):
             #Selecting the max score and its corresponding characteristics
             tmp = df.sort_values(by=['cv_score', 'alpha'], ascending=[False, True])
             #Printing best 5 scores and their params
             print(tmp.iloc[0:15,:].to_string())
```

```
In [40]: def retrain_with_best_hyperparameters(X, Y, best_alpha, vectorizer):
             vectorizer = get_vectorizer(vectorizer, X)

             x_train = vectorizer.transform(X)
             y_train = np.array(Y)

             mnb = MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=None)

             #Training the model
             mnb.fit(x_train, y_train)
             return mnb, vectorizer #returning the vectorizer here so as to avoid having to re
```

```
In [41]: def plot_confusion_matrix(model, data, labels, dataset_label):
             pred = model.predict(data)
             conf_mat = confusion_matrix(labels, pred)

             strings = strings = np.asarray([['TN = ', 'FP = '],
                                             ['FN = ', 'TP = ']])

             labels = (np.asarray(["{0}{1}".format(string, value)
                                   for string, value in zip(strings.flatten(),
                                                            conf_mat.flatten())])
                      ).reshape(2, 2)

             fig, ax = plt.subplots()
             ax.set(xlabel='Predicted', ylabel='Actual', title='Confusion Matrix : {0}'.format
             sns.heatmap(conf_mat, annot=labels, fmt="", cmap='YlGnBu', ax=ax)
             ax.set_xlabel('Predicted')
             ax.set_ylabel('Actual')
             ax.set_xticklabels(['False', 'True'])
             ax.set_yticklabels(['False', 'True'])
             plt.show()
```

```
In [42]: def plot_AUC_ROC(mnb, vectorizer, Dx_train, Dx_test, Dy_train, Dy_test):

             #predicting probability of Dx_test, Dx_train
             test_score = mnb.predict_proba(vectorizer.transform(Dx_test))
             train_score = mnb.predict_proba(vectorizer.transform(Dx_train))

             #Finding out the ROC_AUC_SCORE
             train_roc_auc_score = roc_auc_score(np.array(Dy_train), train_score[:, 1])
             print('Area Under the Curve for Train : ', train_roc_auc_score)
             test_roc_auc_score = roc_auc_score(np.array(Dy_test), test_score[:, 1])
             print('Area Under the Curve for Test : ', test_roc_auc_score)

             #Plotting with matplotlib.pyplot
             #ROC Curve for D-train
             train_fpr, train_tpr, thresholds = roc_curve(np.array(Dy_train), train_score[:, 1
             plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))

             # #ROC Curve for D-test
             test_fpr, test_tpr, thresholds = roc_curve(np.array(Dy_test), test_score[:, 1])
             plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))

             plt.legend()
             plt.xlabel("Alpha: hyperparameter")
             plt.ylabel("AUC")
             plt.title("Area Under ROC Curve")
             plt.show()

             plot_confusion_matrix(mnb, vectorizer.transform(Dx_train), np.array(Dy_train), 'T
             plot_confusion_matrix(mnb, vectorizer.transform(Dx_test), np.array(Dy_test), 'Tes
```

```
In [43]: # Please write all the code with proper documentatio
         results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='BOW', vec
         # Analysing results
         analyse_results(results)

         # Selecting best hyperparameters
         selecting_best_hyperparameters(results)
```

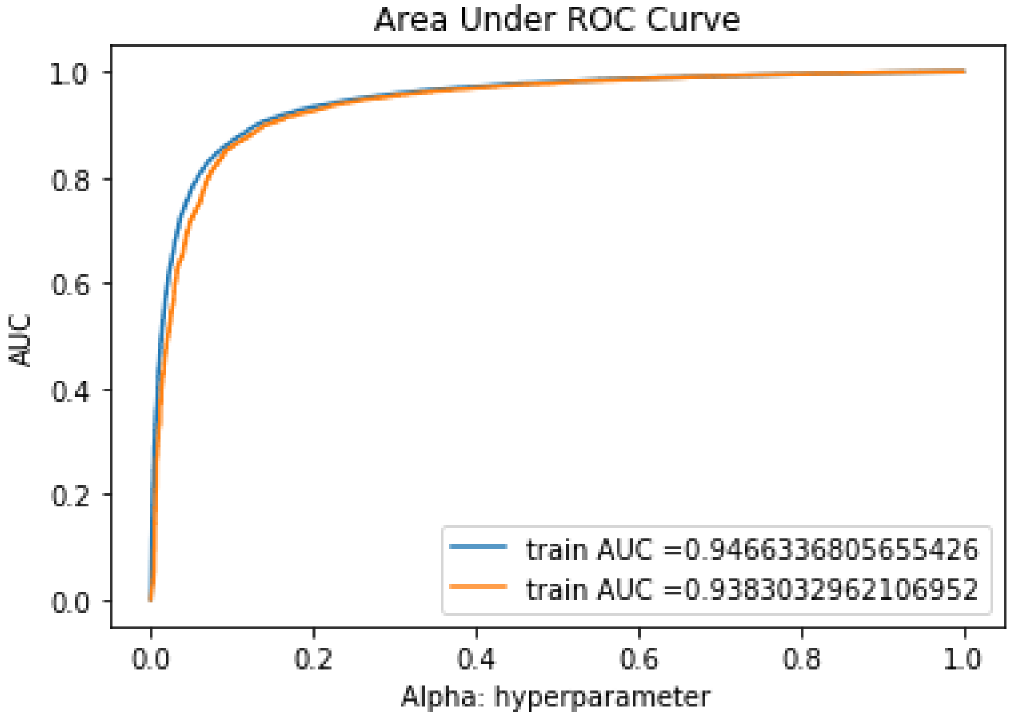


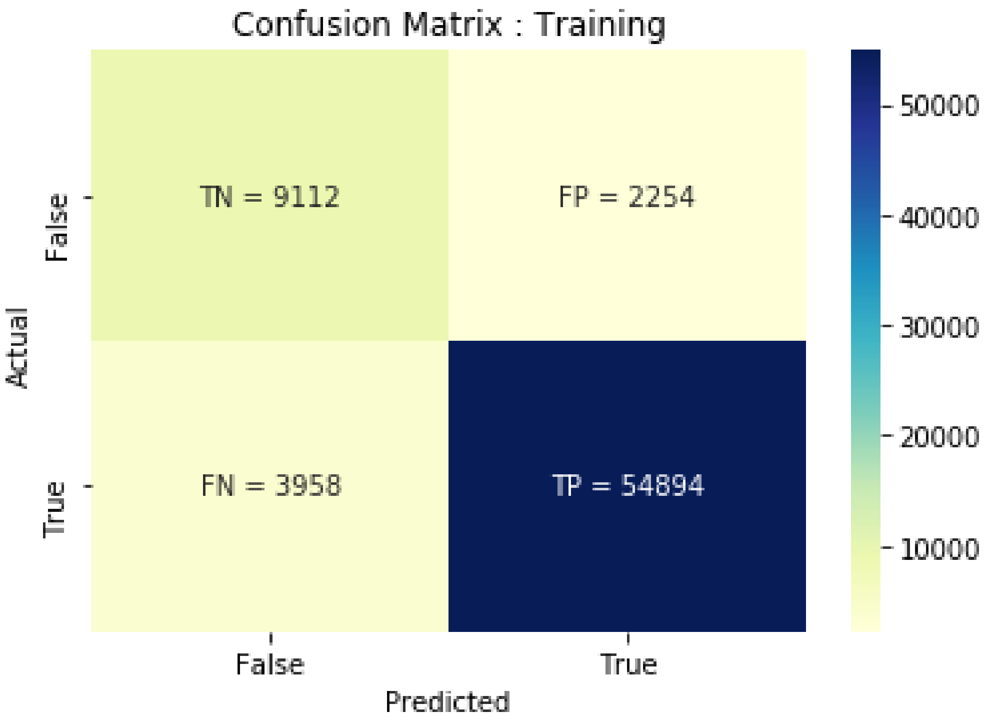
```
    Unnamed: 0    alpha  train_score  cv_score
20          20  1.00000     0.947270  0.938017
18          18  0.50000     0.947502  0.938015
16          16  0.10000     0.947748  0.937941
15          15  0.07500     0.947772  0.937921
19          19  0.07500     0.947772  0.937921
14          14  0.05000     0.947801  0.937896
21          21  2.50000     0.946656  0.937860
13          13  0.02500     0.947839  0.937849
17          17  0.02500     0.947839  0.937849
12          12  0.01000     0.947876  0.937779
11          11  0.00750     0.947885  0.937756
10          10  0.00500     0.947897  0.937721
9            9  0.00250     0.947914  0.937661
8            8  0.00100     0.947932  0.937579
7            7  0.00075     0.947937  0.937551
```

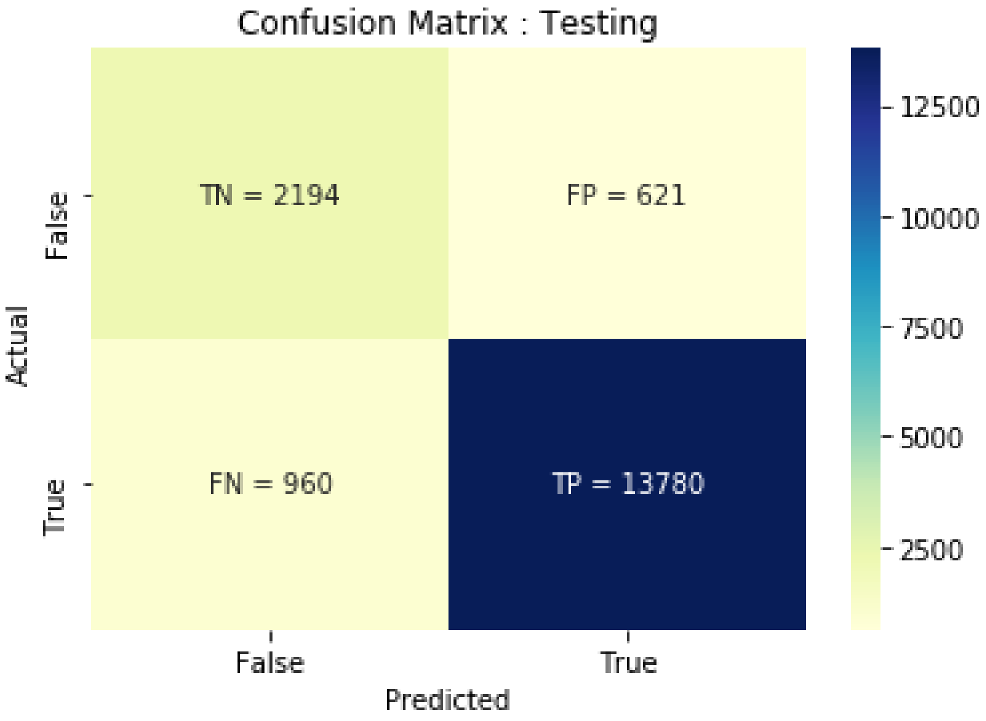
We can select alpha = 1 as the best hyper-parameter

```
In [44]: mnb, vectorizer = retrain_with_best_hyperparameters(X=Dx_train, Y=Dy_train,
                                                             best_alpha = 1,
                                                             vectorizer='BOW')
         plot_AUC_ROC(mnb, vectorizer, Dx_train, Dx_test, Dy_train, Dy_test)
```

```
Area Under the Curve for Train :  0.9466336805655426
Area Under the Curve for Test :  0.9383032962106952
```




Confusion Matrix : Training
TN = 9112
FP = 2254
FN = 3958
TP = 54894
Actual
False
True
Predicted
False
True
50000
40000
30000
20000
10000


Confusion Matrix : Testing
TN = 2194
FP = 621
FN = 960
TP = 13780
Actual
False
True
Predicted
False
True
12500
10000
7500
5000
2500

```
In [45]: prettytable_data.append(['BOW', 'MultinomialNB','alpha = 1', 0.9466336805655426, 0.93
```

7.1.3 [5.1.1] Top 10 important features of positive class from SET 1

```
In [46]: # Please write all the code with proper documentation
         pos_class_prob_sorted = mnb.feature_log_prob_[1, :].argsort()
         pos_class_prob_sorted = pos_class_prob_sorted[::-1]
         print(np.take(vectorizer.get_feature_names(), pos_class_prob_sorted[:10]))
```

```
['not' 'like' 'good' 'great' 'one' 'taste' 'coffee' 'flavor' 'would'
 'love']
```

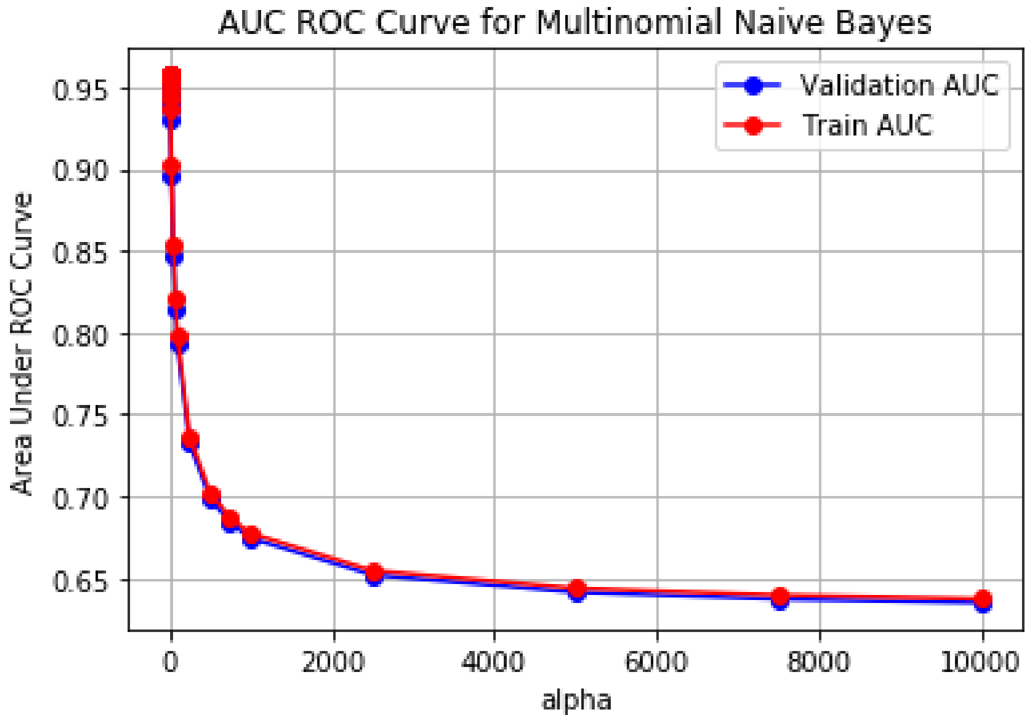
7.1.4 [5.1.2] Top 10 important features of negative class from SET 1

```
In [47]: # Please write all the code with proper documentation
         neg_class_prob_sorted = mnb.feature_log_prob_[0, :].argsort()
         neg_class_prob_sorted = neg_class_prob_sorted[::-1]
         print(np.take(vectorizer.get_feature_names(), neg_class_prob_sorted[:10]))
```

```
['not' 'like' 'would' 'taste' 'product' 'one' 'good' 'coffee' 'no'
 'flavor']
```

7.2 [5.2] Applying Naive Bayes on TFIDF, SET 2

```
In [48]: # Please write all the code with proper documentation
         results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='TFIDF', ve

         # Analysing results
         analyse_results(results)

         # Selecting best hyperparameters
         selecting_best_hyperparameters(results)
```

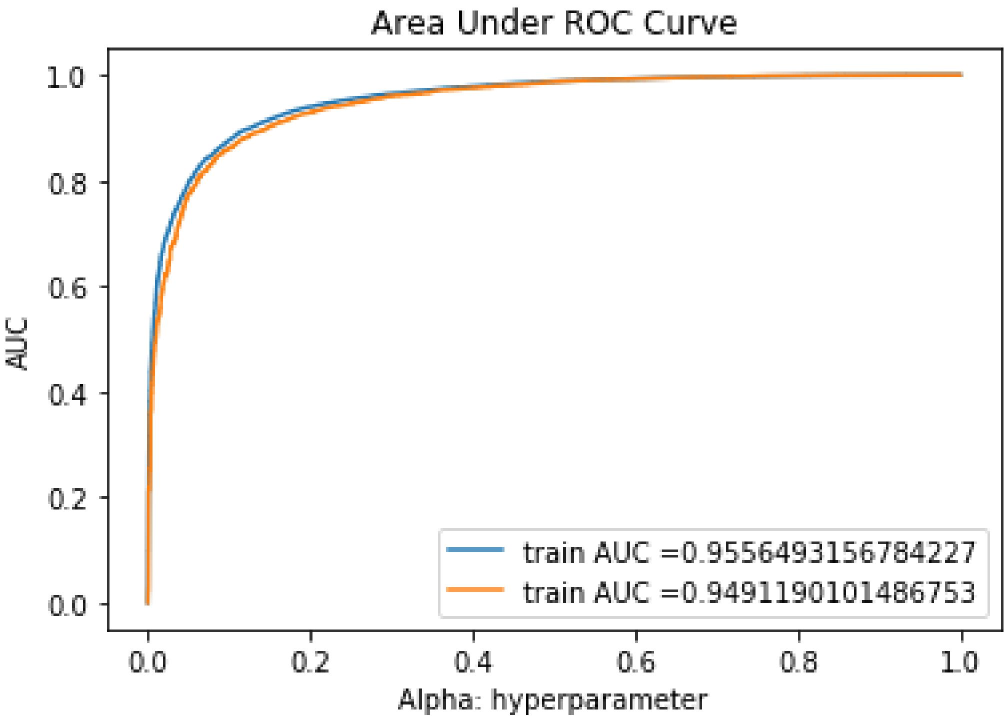


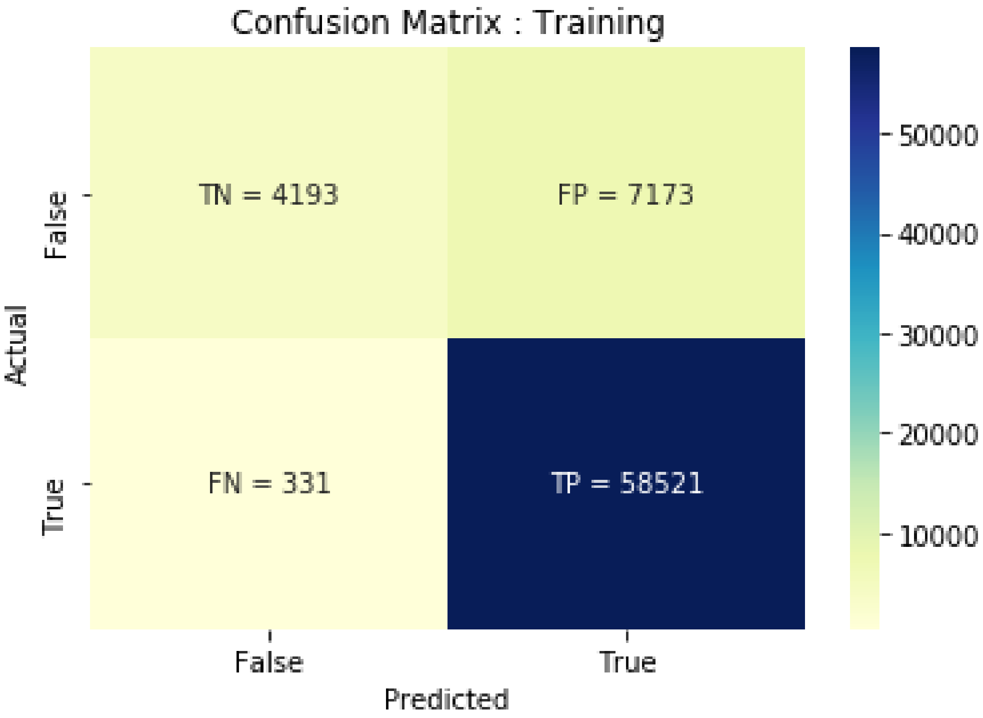
```
    Unnamed: 0    alpha  train_score  cv_score
18          18  0.50000     0.956951  0.948060
20          20  1.00000     0.956136  0.947802
16          16  0.10000     0.957526  0.947782
15          15  0.07500     0.957563  0.947726
19          19  0.07500     0.957563  0.947726
14          14  0.05000     0.957605  0.947652
13          13  0.02500     0.957654  0.947542
17          17  0.02500     0.957654  0.947542
12          12  0.01000     0.957697  0.947420
11          11  0.00750     0.957707  0.947385
10          10  0.00500     0.957719  0.947338
9            9  0.00250     0.957736  0.947253
8            8  0.00100     0.957754  0.947146
7            7  0.00075     0.957758  0.947114
6            6  0.00050     0.957764  0.947068
```

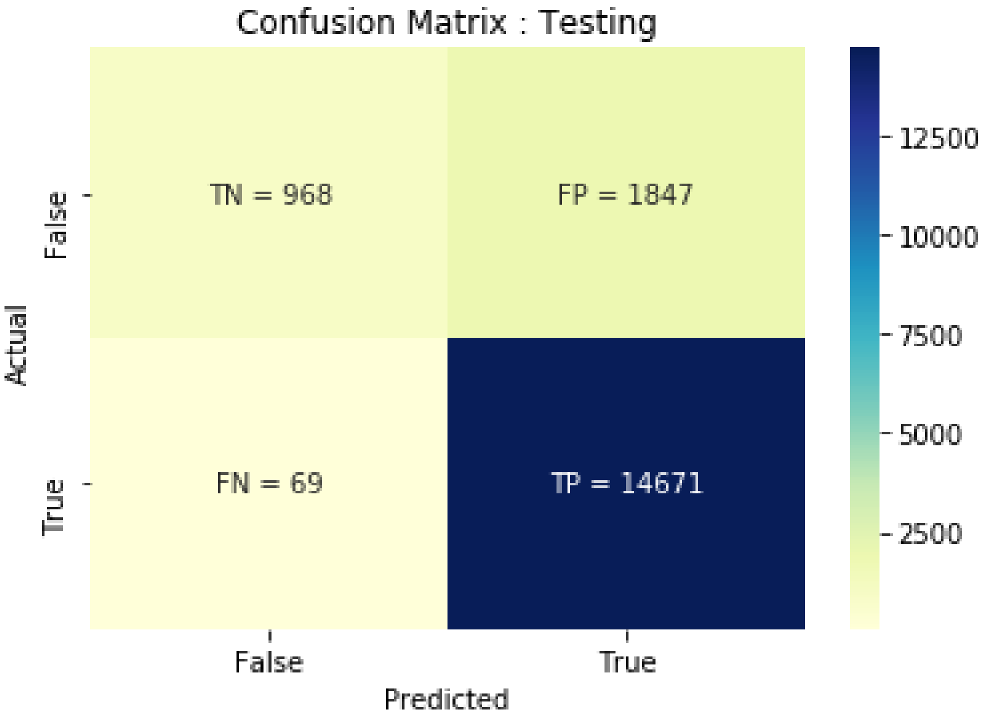
We can select alpha = 1 as the best hyper-parameter

```
In [49]: mnb, vectorizer = retrain_with_best_hyperparameters(X=Dx_train, Y=Dy_train,
                                                              best_alpha = 1,
                                                              vectorizer='TFIDF')
         plot_AUC_ROC(mnb, vectorizer, Dx_train, Dx_test, Dy_train, Dy_test)
```

```
Area Under the Curve for Train :  0.9556493156784227
Area Under the Curve for Test :  0.9491190101486753
```







```
In [50]: prettytable_data.append(['TF-IDF', 'MultinomialNB','alpha = 1', 0.9556493156784227, 0
```

7.2.1 [5.2.1] Top 10 important features of positive class from SET 2

```
In [51]: # this code snippet below has been taken from stackoverflow
```

```
In [52]: # Please write all the code with proper documentation
         pos_class_prob_sorted = mnb.feature_log_prob_[1, :].argsort()
         print(np.take(vectorizer.get_feature_names(), pos_class_prob_sorted[:10]))
```

```
['returns' 'inedible' 'waste money' 'not impressed' 'really wanted' 'fda'
 'expired' 'ruined' 'quality control' 'soybean']
```

7.2.2 [5.2.2] Top 10 important features of negative class from SET 2

```
In [53]: # Please write all the code with proper documentation
         neg_class_prob_sorted = mnb.feature_log_prob_[0, :].argsort()
         print(np.take(vectorizer.get_feature_names(), neg_class_prob_sorted[:10]))
```

```
['coffee smooth' 'no bitter' 'no bitterness' 'not regret' 'best gluten'
 'goes long' 'glad amazon' 'say enough' 'fast delivery' 'every penny']
```

8 [6] Conclusions

```
In [54]: # Please compare all your models using Prettytable library
         x = PrettyTable()

         x.field_names = ["Vectorizer", "Model", "Hyper parameter", "Train AUC", "Test AUC"]
         [x.add_row(i) for i in prettytable_data]
         print(x)
```

```
+------------+---------------+-----------------+--------------------+--------------------+
| Vectorizer |     Model     | Hyper parameter |     Train AUC      |      Test AUC      |
+------------+---------------+-----------------+--------------------+--------------------+
|    BOW     | MultinomialNB |    alpha = 1    | 0.9466336805655426 | 0.9383032962106952 |
|   TF-IDF   | MultinomialNB |    alpha = 1    | 0.9556493156784227 | 0.9491190101486753 |
+------------+---------------+-----------------+--------------------+--------------------+
```

Naive Bayes is insanely fast as can be felt while implementing!