AmazonFineFoodReviewsAnalysisNaiveBayes

January 25, 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 [32]:
```
%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 TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score
from scipy.sparse import hstack

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.stem import SnowballStemmer

from tqdm import tqdm
import os

from sklearn.preprocessing import StandardScaler

import pickle as pkl
from prettytable import PrettyTable
from sklearn.naive_bayes import MultinomialNB

from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)
```

In [4]:
```
# Read the Amazon fine food review data from database using sqlite
con = sqlite3.connect('database.sqlite')

# Select all reviews where score is not 3 (neutral)
review_data = pd.read_sql_query("""SELECT * FROM Reviews WHERE Score != 3""", con)

# Assign positive class if score >=4 else assign negative class
score = review_data['Score']
```

```
PN_score = score.map(lambda x: "Positive" if x>=4 else "Negative")
review_data['Score'] = PN_score


print("Shape of review data is {}".format(review_data.shape))
review_data.head(3)
```

```
Shape of review data is (525814, 10)
```

Out[4]:

```
   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  Positive  1303862400
1                     0                       0  Negative  1346976000
2                     1                       1  Positive  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 [5]:

```
#Trying to visualize the duplicate data before removal
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [7]:

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

```
(80668, 7)
```

Out[7]:

```
               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 [8]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[8]:
```
                  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 [9]:
```
display['COUNT(*)'].sum()
```

Out[9]:
```
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 [12]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[12]:
```
       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=Fa

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

In [13]:
```
#Remove the duplicate entries from the data

sorted_data = review_data.sort_values('ProductId')
final = sorted_data.drop_duplicates(subset=["UserId", "Time", "Summary"])
print(final.shape)
```

```
(363186, 10)
```

In [15]:
```
#Checking to see how much % of data still remains
print((final['Id'].size*1.0)/(review_data['Id'].size*1.0)*100)
```

```
69.07119247490557
```

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 [16]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[16]:
```
      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 [18]:
```
# Removing the reviews where HelpfullnessNumerator > HelpfulnessDenominator

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

In [20]:
```
#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?
print(final['Score'].value_counts())
```

```
(363184, 10)
Positive    306173
Negative     57011
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 [42]:
```python
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
from bs4 import BeautifulSoup
```

In [47]:
```python
# 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 [43]:
```python
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
```

In [116]:
```python
# 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', 'oursel
            "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'
                         "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 [55]: #Intializing SnowballStemmer
         snow_stemmer = SnowballStemmer('english')

         #Using Stemmer on a word
         print(snow_stemmer.stem('Moves'))
```

```
move
```

```
In [48]: # Combining all the above to clean reviews
         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 stopwo
             preprocessed_reviews.append(sentence.strip())
```

```
100%|| 363184/363184 [04:45<00:00, 1270.29it/s]
```

```
In [64]: # Storing the preprocessed reviews and stemmed preprocessed reviews seperately.
         # We have performed the cleaning on the whole data so we can use it later on
         # models other than KNN that can handle high dimensional data gracefully.

         ############---- storing the data into .sqlite file ------########################
         # Reviews are present in preprocessed_reviews

         final['CleanedText'] = preprocessed_reviews

         #Store the data into a sqlite database
         if not os.path.isfile('final.sqlite'):
             conn = sqlite3.connect('final.sqlite')
```

```
    c = conn.cursor()
    conn.text_factory = str
    final.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
                 index=True, index_label=None, chunksize=None, dtype=None)
    conn.close()
```

In [91]:
```
# Performing stemming on the preprocessed reviews
final['CleanedText'] = preprocessed_reviews
stemmed_reviews = []

for sentence in final['CleanedText'].values:
    sentence = b' '.join((snow_stemmer.stem(word)).encode('utf8') for word in sentence
    stemmed_reviews.append(sentence)
```

In [85]:
```
final['CleanedText'] = stemmed_reviews
final['CleanedText'] = final['CleanedText'].str.decode("utf-8")

if not os.path.isfile('final_stemmedreviews.sqlite'):
    conn = sqlite3.connect('final_stemmedreviews.sqlite')
    c = conn.cursor()
    conn.text_factory = str
    final.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
                 index=True, index_label=None, chunksize=None, dtype=None)
    conn.close()
```

In [2]:
```
# Load the preprocessed dataset from the database final.sqlite
# Data is ordered by time stamps to facilitate time base splitting
# of data for cross validation

conn = sqlite3.connect('final.sqlite')
final = pd.read_sql_query("""SELECT * From Reviews ORDER BY Time""", conn)
conn.close()

conn = sqlite3.connect('final_stemmedreviews.sqlite')
final_stemmed = pd.read_sql_query("""SELECT * From Reviews ORDER BY Time""", conn)
conn.close()
```

In [3]:
```
# There is an extra index column in the data
final.head(1)
```

Out[3]:
```
    index      Id   ProductId         UserId      ProfileName  \
0  138706  150524  0006641040  ACITT7DI6IDDL  shari zychinski

   HelpfulnessNumerator  HelpfulnessDenominator     Score       Time  \
0                     0                       0  Positive  939340800

                    Summary  \
0  EVERY book is educational
```

```
                                                     Text  \
0  this witty little book makes my son laugh at l...

                                              CleanedText
0  witty little book makes son laugh loud recite ...
```

```
In [4]: #Removing the index column from data
        clean_data = final.drop(['index'], axis=1)

        #clean_data_stemmed = final_stemmed.drop(['index'], axis=1)

        # Map postive to 1 and negative to 0 in Score column
        score = clean_data['Score']
        bin_score = score.map(lambda x: 1 if x == "Positive" else 0)
        clean_data['Score'] = bin_score

        # Add stemmed reviews as an extra column in the data
        # This will be in addition to the preprocessed non stemmed
        # reviews which are stored in the CleanedText column.

        stemmed_reviews = final_stemmed['CleanedText']
        clean_data['StemmedText'] = stemmed_reviews
```

```
In [5]: # Adding another feature into the data
        # we will find the length of the each review
        # and add that as a feature into the existing
        # dataframe.

        clean_data['Reviewlen'] = clean_data['StemmedText'].apply(len)
```

```
In [6]: # Split the dataset in training and test dataset
        # We will use the training data for cross validation and training.
        # Test data will not be known to model and will be used
        # to calculate the accuracy.

        # Data is split in 70-30 train-test split using slicing since
        # data is sorted in ascending time order

        # Instead of splitting the data and then sampling
        # let's try to split the 100k samples directly and
        # then just simple time split the data in 70-30k

        data = clean_data.iloc[:,:]
        subset_data = data.iloc[100000:200000,:]

        train_cv_split = 70000

        train = subset_data.iloc[:train_cv_split,:]
```

```
test = subset_data.iloc[train_cv_split:,:]

print(train.shape , '\n', test.shape)
```

```
(70000, 13)
 (30000, 13)
```

```
In [7]: print(train[train['Score'] == 0].shape)
        print(test[test['Score'] == 0].shape)
```

```
(11235, 13)
(4961, 13)
```

```
In [8]: # Seperating the Score column from rest of the data
        columns = list(clean_data.columns)
        columns = [column for column in columns if column != 'Score']

        X_train = train[columns]
        y_train = train['Score']

        X_test = test[columns]
        y_test = test['Score']

        print(X_train.shape , y_train.shape, '\n', X_test.shape, y_test.shape)
```

```
(70000, 12) (70000,)
 (30000, 12) (30000,)
```

```
In [9]: # Save the y_train and y_test so we
        # can directly use it later rather than rerunning
        # the splitting steps again

        pkl.dump(y_train, open("y_train.pkl", 'wb'))
        pkl.dump(y_test, open("y_test.pkl", 'wb'))
```

[3.2] Preprocessing Review Summary

```
In [6]: ## Similartly you can do preprocessing for review summary also.
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [105]: # Running count vectorizer on training data only
          # to avoid data leakage
          # we will use the uni-grams & bi-grams in BoW embedding
```

```
# count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
count_vec = CountVectorizer(ngram_range=(1,2), min_df=10)

X_train_bow = count_vec.fit_transform(X_train['StemmedText'].values)
X_test_bow = count_vec.transform(X_test['StemmedText'].values)

# Save the training and test BOW vectors in pickle files
# We can simply load this data later and use it

pkl.dump(X_train_bow, open("train_bow.pkl", 'wb'))
pkl.dump(X_test_bow, open("test_bow.pkl", 'wb'))
pkl.dump(count_vec, open("count_vec.pkl", 'wb'))
```

5.2 [4.2] TF-IDF

In [106]:
```
# Apply tfidf vectorizer to convert text to vectors

tf_idf = TfidfVectorizer(ngram_range=(1,2), min_df=10)

X_train_tfidf = tf_idf.fit_transform(X_train['StemmedText'].values)
X_test_tfidf = tf_idf.transform(X_test['StemmedText'].values)

# Save the training, CV and test TFIDF vectors in pickle files
# We can simply load this data later and use it

pkl.dump(X_train_tfidf, open("train_tfidf.pkl", 'wb'))
pkl.dump(X_test_tfidf, open("test_tfidf.pkl", 'wb'))
pkl.dump(count_vec, open("tf_idf.pkl", 'wb'))
```

In [107]:
```
# Creating a dictionary with word as key and it's tfidf representation as value
dictionary = dict(zip(tf_idf.get_feature_names(), list(tf_idf.idf_)))

pkl.dump(dictionary, open("tfidf_dictionary.pkl", 'wb'))
```

5.3 Utility Functions used in Multinomial Naive Bayes classification

In [29]:
```
# This function takes the vector representation of review data
# and returns the optimal alpha for MultinomailNB classification
# using 5-fold cross validation.

# Code below splits the TimeSeries data in linear fashion including
# another split of data progressively with each iteration.

def get_optimal_alpha(X_train_data, y_train_data, vect, flag, n_splits=5):
    auc_scores_cv = []
    auc_scores_train = []
    alpha_values = [10**i for i in range(-5,6)]
```

```
lot_size = int(X_train_data.shape[0] / n_splits)
X_train_start = 0
X_train_end = 0
X_cv_start = 0
X_cv_end = 0

for alpha in alpha_values:
    avg_scores_cv = []
    avg_scores_train = []
    for i in range(1, n_splits):
        X_train_end = lot_size*i
        X_cv_start = X_train_end
        X_cv_end = X_cv_start + lot_size
        #print(X_train_start, X_train_end, X_cv_start, X_cv_end)
        X_train = X_train_data.iloc[X_train_start:X_train_end, :]
        X_cv = X_train_data.iloc[X_cv_start:X_cv_end, :]
        y_train = y_train_data.iloc[X_train_start:X_train_end]
        y_cv = y_train_data.iloc[X_cv_start:X_cv_end]
        #print(y_train.shape, y_cv.shape)

        X_train_revlen = np.array(X_train['Reviewlen'].values).astype(float)
        X_cv_revlen = np.array(X_cv['Reviewlen'].values).astype(float)

        # Running vectorizer on X_train to tranform text to vector
        if vect == "bow":
            count_vec = CountVectorizer(ngram_range=(1,2), min_df=10)
            X_train_vec = count_vec.fit_transform(X_train['StemmedText'].values)
            X_cv_vec = count_vec.transform(X_cv['StemmedText'].values)
        else:
            tf_idf = TfidfVectorizer(ngram_range=(1,2), min_df=10)
            X_train_vec = tf_idf.fit_transform(X_train['StemmedText'].values)
            X_cv_vec = tf_idf.transform(X_cv['StemmedText'].values)

        # hstack here helps in adding the new review length column to the
        # existing text vectors. Based of flag value we add it if we are
        # considering review length as an additional feature.
        if flag == 1:
            X_train_vec = hstack((X_train_vec,X_train_revlen[:,None])).tocsr()
            X_cv_vec = hstack((X_cv_vec,X_cv_revlen[:,None])).tocsr()

        std = StandardScaler(with_mean=False)

        # Standardizing the vectors
        X_train = std.fit_transform(X_train_vec)
        X_cv = std.transform(X_cv_vec)

        nb_clf = MultinomialNB(alpha=alpha)
        nb_clf.fit(X_train, y_train)
```

```
                y_pred_cv = nb_clf.predict_proba(X_cv)[:,1]
                y_pred_train = nb_clf.predict_proba(X_train)[:,1]

                fpr_cv, tpr_cv, thresholds_cv = roc_curve(y_cv, y_pred_cv)
                avg_score_cv = auc(fpr_cv, tpr_cv)

                fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_pred_train)
                avg_score_train = auc(fpr_train, tpr_train)

                avg_scores_cv.append(avg_score_cv)
                avg_scores_train.append(avg_score_train)

            auc_score = round(sum(avg_scores_cv) / float(len(avg_scores_cv)), 2)
            auc_scores_cv.append(auc_score)

            auc_score = round(sum(avg_scores_train) / float(len(avg_scores_train)), 2)
            auc_scores_train.append(auc_score)

            #print("Accuracy on CV data with k = {} is {}%".format(k, round(auc_score, 2)
        return alpha_values[auc_scores_cv.index(max(auc_scores_cv))],\
                    zip(alpha_values, auc_scores_train), zip(alpha_values, auc_scores_
```

In [26]:
```
# Running MultinomialNB with given alpha
# returns a tuple indicating AUC obtained for
# test data along with the confusion matrix along with the classifier
# object same function can be used on all vectorized data irrespective
# of vectorizer

def run_nb(X_train, y_train, X_test, y_test, alpha):
    nb_clf = MultinomialNB(alpha=alpha)
    nb_clf.fit(X_train, y_train)

    y_pred_test = nb_clf.predict_proba(X_test)
    y_pred_train = nb_clf.predict_proba(X_train)

    y_pred_test_prob = y_pred_test[:,1]
    y_pred_test_label = np.argmax(y_pred_test, axis=1)

    y_pred_train_prob = y_pred_train[:,1]
    y_pred_train_label = np.argmax(y_pred_train, axis=1)

    fpr_test, tpr_test, thresholds_test = roc_curve(y_test,\
                                                    y_pred_test_prob)
    auc_score_test = auc(fpr_test, tpr_test)

    fpr_train, tpr_train, thresholds_train = roc_curve(y_train, \
                                                       y_pred_train_prob)
```

```
        auc_score_train = auc(fpr_train, tpr_train)

        conf_mat = confusion_matrix(y_test, y_pred_test_label)

        plt.figure()
        plt.plot(fpr_train, tpr_train, color='darkorange', lw=1, \
                 label='Train ROC curve (area = %0.2f)' % auc_score_train)
        plt.plot(fpr_test, tpr_test, color='navy', lw=1, \
                 label='Test ROC curve (area = %0.2f)' % auc_score_test)
        plt.plot([0, 1], [0, 1], color='black', lw=1, linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic')
        plt.legend(loc="lower right")
        plt.show()

        return nb_clf, auc_score_test, conf_mat
```

```
In [27]: def plot_confusion_matrix(cm):
             labels = ['Negative', 'Positive']
             confmat = pd.DataFrame(cm, index = labels, columns = labels)
             sns.heatmap(confmat, annot = True, fmt = 'd', cmap="Greens")
             plt.title("Confusion Matrix")
             plt.xlabel("Predicted Label")
             plt.ylabel("True Label")
             plt.show()
```

```
In [30]: # All the results will be stored in the results dataframe and
         # later in we will use this dataframe to print the results
         # in tabular format

         results = pd.DataFrame(columns=['Features-Used', 'Vectorizer', 'Alpha', 'AUC'])
```

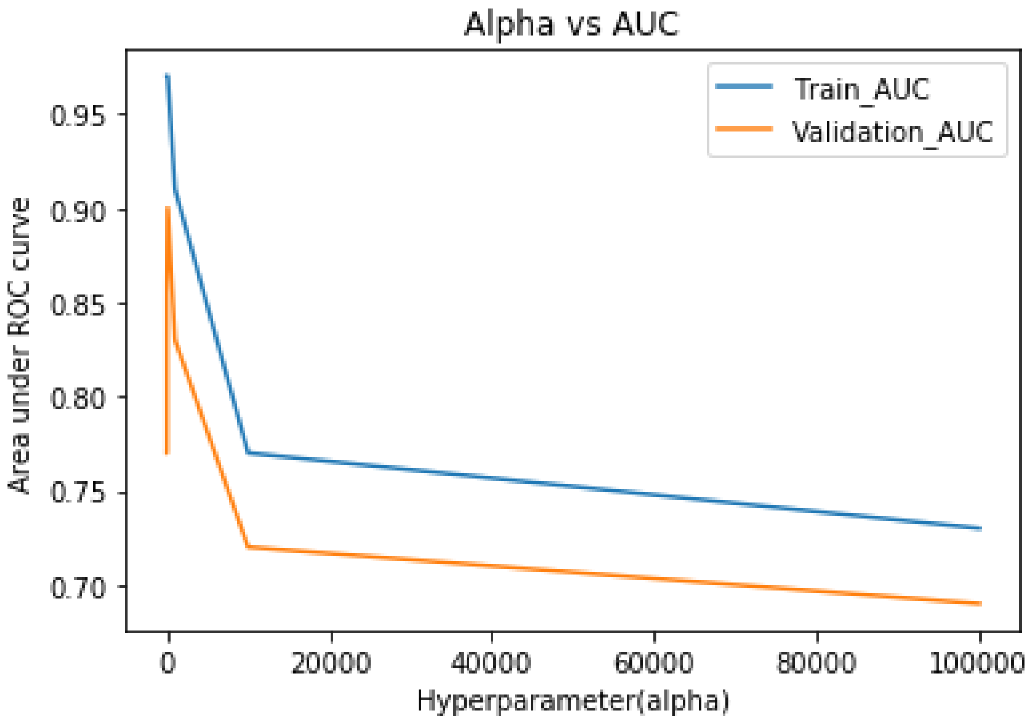
```
In [31]: import prettytable as pt
         # function to print the results obtained in a table format
         def print_results(data):
             result = PrettyTable(hrules=pt.ALL,
                                  vrules=pt.ALL, padding_width=5)
             result.field_names = list(data.columns)
             for i in range(0, data.shape[0]):
                 result.add_row(data.iloc[i])

             #result.align["Vectorizer"] = "l"
             print(result)
```

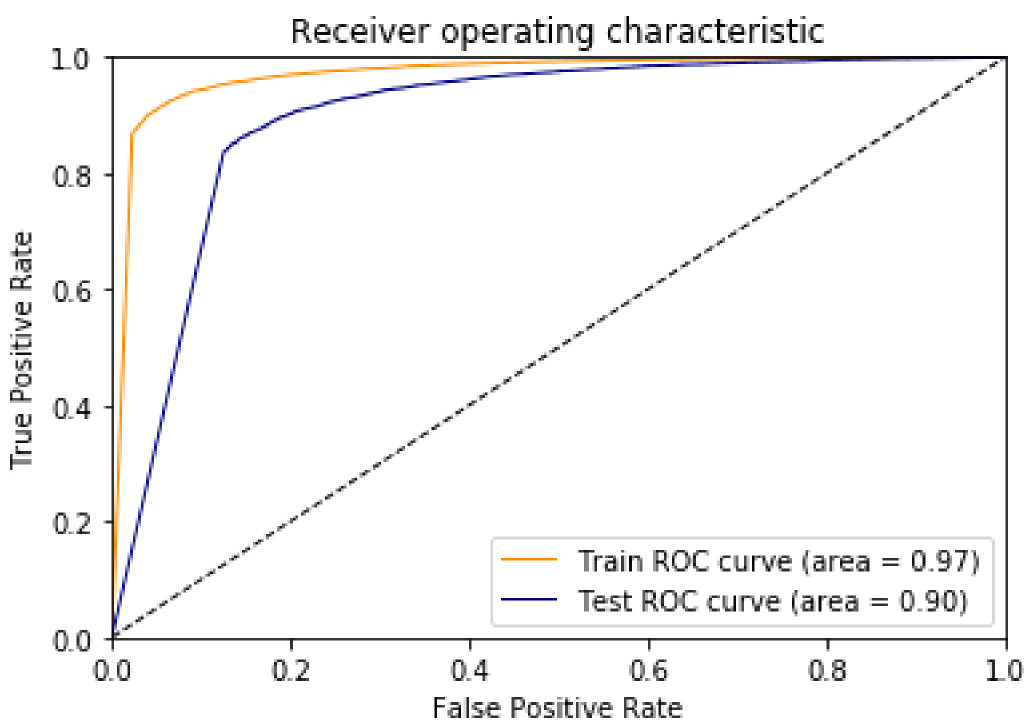
6 Applying Multinomial Naive Bayes

6.1 [5.1.1] Applying Naive Bayes on BOW, SET 1

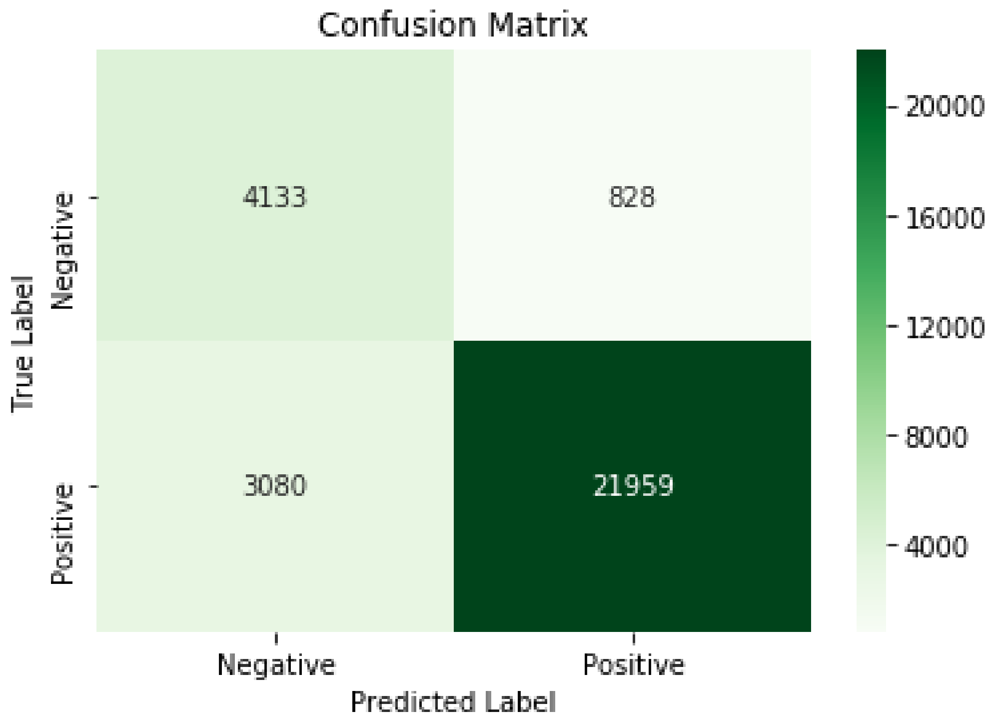
In [33]:
```
# Load the saved vectorized data for train-test datapoints
X_train_bow = pkl.load(open('train_bow.pkl', 'rb'))
X_test_bow = pkl.load(open('test_bow.pkl', 'rb'))

std = StandardScaler( with_mean=False)

# Standardizing the vectors
X_train_bow_std = std.fit_transform(X_train_bow)
X_test_bow_std = std.transform(X_test_bow)

# Getting an optimal value of hyperparameter alpha and AUC scores
# This data is used to plot a graph of alpha-values vs AUC
# To tackle the problem of data leakage, while trying hyperparameter
# tuning using k-fold cv, we will vectorize the data after split into
# train and cv inside get_optimal_alpha()

optimal_alpha, alpha_auc_train, alpha_auc_cv = get_optimal_alpha(X_train, y_train, "bo
print("Optimal value of alpha : {}".format(optimal_alpha))

train_auc = [(alpha, train_auc) for alpha, train_auc in alpha_auc_train]
cv_auc = [(alpha, cv_auc) for alpha, cv_auc in alpha_auc_cv]

# Plotting K values vs AUC scores

plt.title("Alpha vs AUC")
plt.xlabel("Hyperparameter(alpha)")
plt.ylabel("Area under ROC curve")
plt.plot(*(zip(*train_auc)), label='Train_AUC')
plt.plot(*(zip(*cv_auc)), label='Validation_AUC')
plt.legend()
plt.show()

# Running Multinomial NB with optimal alpha value obtained
nb_clf, auc_score, conf_mat = run_nb(X_train_bow_std, y_train,
                                      X_test_bow_std, y_test, optimal_alpha)

print("AUC score:\n {:.2f}".format(auc_score))

# Plotting confusion matrix
plot_confusion_matrix(conf_mat)

auc_score = '%0.2f' % auc_score
# Adding the results to our results dataframe
results.loc[results.shape[0]] = ["Review Text", "BoW", optimal_alpha, auc_score]
```

```
Optimal value of alpha : 100
```





```
AUC score:
 0.90
```



6.2 [5.1.2] Top 10 important features of positive class from SET 1

In [34]:
```
# Loading the saved count vectorizer object
count_vec = pkl.load(open("count_vec.pkl", 'rb'))
```

In [38]:
```
# Top 10 features in positive class
max_ind_pos=np.argsort((nb_clf.feature_log_prob_)[1])[::-1][0:10]
print(np.take(count_vec.get_feature_names(), max_ind_pos))
```

```
['not' 'love' 'great' 'good' 'like' 'tast' 'one' 'tri' 'use' 'flavor']
```

6.3 [5.1.3] Top 10 important features of negative class from SET 1

In [39]:
```
# Top 10 features in negative class
max_ind_neg=np.argsort((nb_clf.feature_log_prob_)[0])[::-1][0:10]
print(np.take(count_vec.get_feature_names(), max_ind_neg))
```

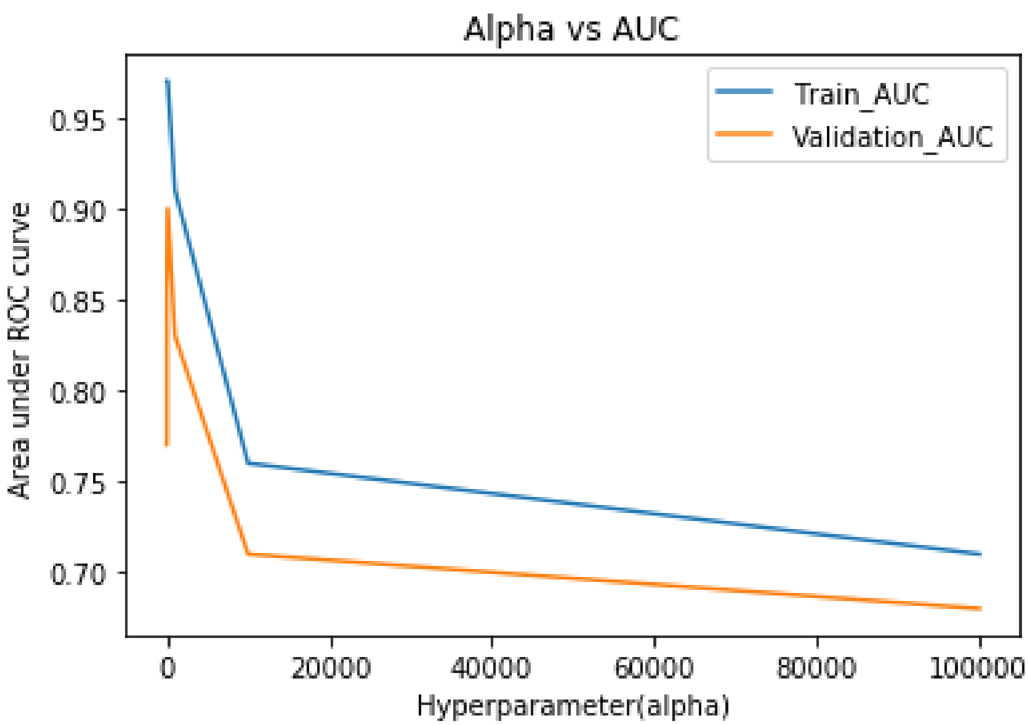
```
['not' 'tast' 'like' 'disappoint' 'product' 'would' 'bad' 'buy' 'not buy'
 'one']
```

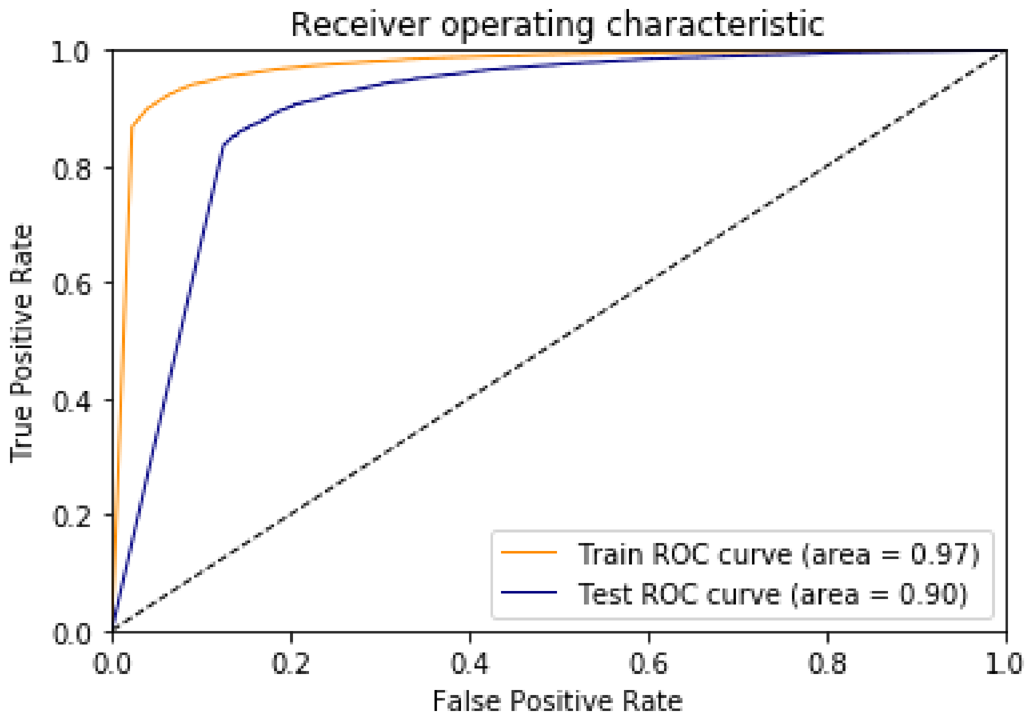
6.4 [5.1.2] Naive Bayes on BoW with Review length as new feature SET 1

In [40]:
```
# Load the saved vectorized data for train-test datapoints
X_train_bow = pkl.load(open('train_bow.pkl', 'rb'))
X_test_bow = pkl.load(open('test_bow.pkl', 'rb'))

# We will add review length as new feature with our vectorized
# data and train the model on this. We have calculated
# review length in main dataframe and has been added as a
# seperate column and can be accessed with X_train['Reviewlen']

X_train_revlen = np.array(X_train['Reviewlen'].values)
X_test_revlen = np.array(X_test['Reviewlen'].values)

# hstack here helps in adding the new review length column to the
# existing text vectors.

X_train_bow = hstack((X_train_bow,X_train_revlen[:,None])).tocsr()
X_test_bow = hstack((X_test_bow,X_test_revlen[:,None])).tocsr()

# We will standardize the whole data including review length
std = StandardScaler( with_mean=False)

# Standardizing the vectors
X_train_bow_std = std.fit_transform(X_train_bow)
X_test_bow_std = std.transform(X_test_bow)

# Getting an optimal value of hyperparameter alpha and AUC scores
# This data is used to plot a graph of alpha-values vs AUC
optimal_alpha, alpha_auc_train, alpha_auc_cv = get_optimal_alpha(X_train, y_train, 'b
print("Optimal value of alpha : {}".format(optimal_alpha))

train_auc = [(alpha, train_auc) for alpha, train_auc in alpha_auc_train]
cv_auc = [(alpha, cv_auc) for alpha, cv_auc in alpha_auc_cv]

# Plotting Alpha hyperparameter values vs AUC scores

plt.title("Alpha vs AUC")
plt.xlabel("Hyperparameter(alpha)")
plt.ylabel("Area under ROC curve")
plt.plot(*(zip(*train_auc)), label='Train_AUC')
plt.plot(*(zip(*cv_auc)), label='Validation_AUC')
plt.legend()
plt.show()
```

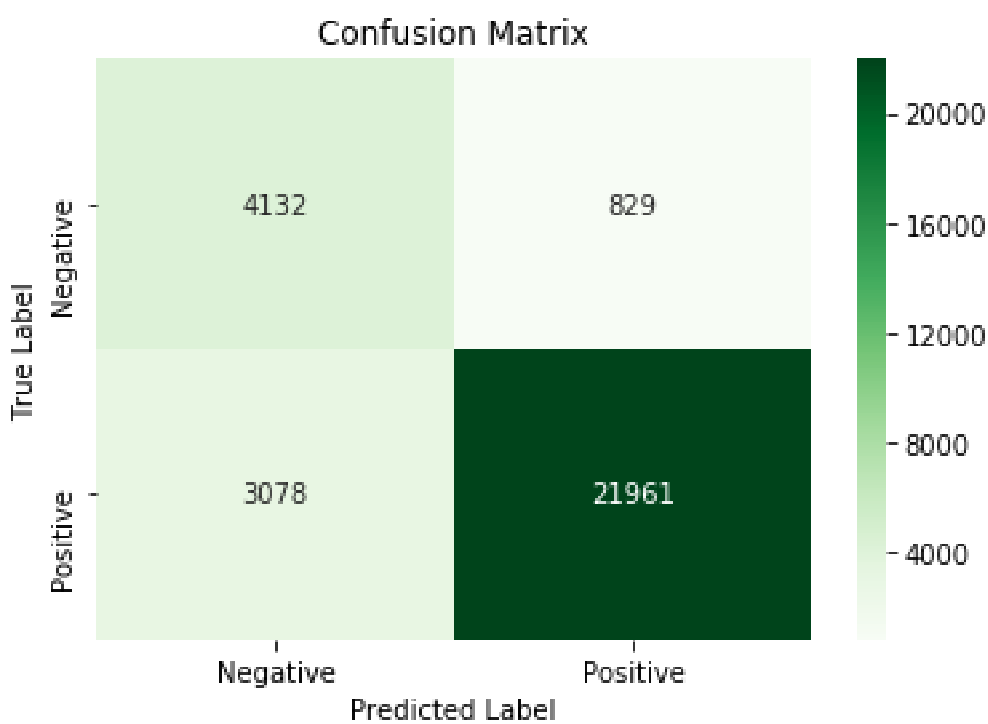
```
# Running Multinomial NB with optimal alpha value obtained
nb_clf, auc_score, conf_mat = run_nb(X_train_bow_std, y_train,
                                      X_test_bow_std, y_test, optimal_alpha)

print("AUC score:\n {:.2f}".format(auc_score))

# Plotting confusion matrix
plot_confusion_matrix(conf_mat)

auc_score = '%0.2f' % auc_score

# Adding the results to our results dataframe
results.loc[results.shape[0]] = ["Review Text/ Review Length", "BoW", optimal_alpha, a
```

```
Optimal value of alpha : 100
```





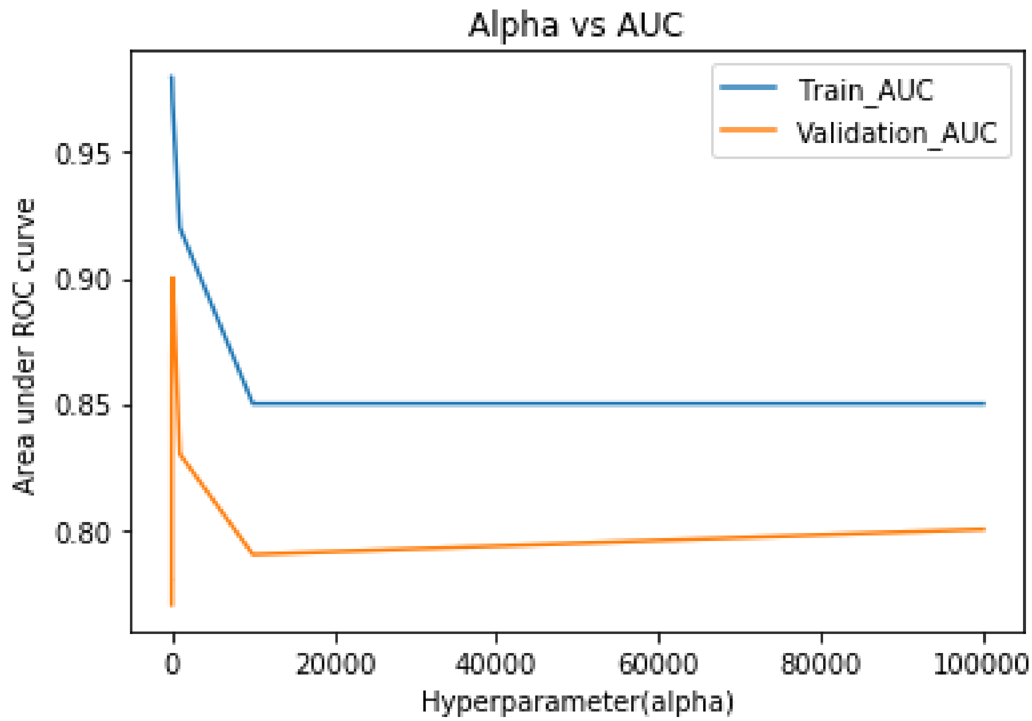

```
AUC score:
 0.90
```

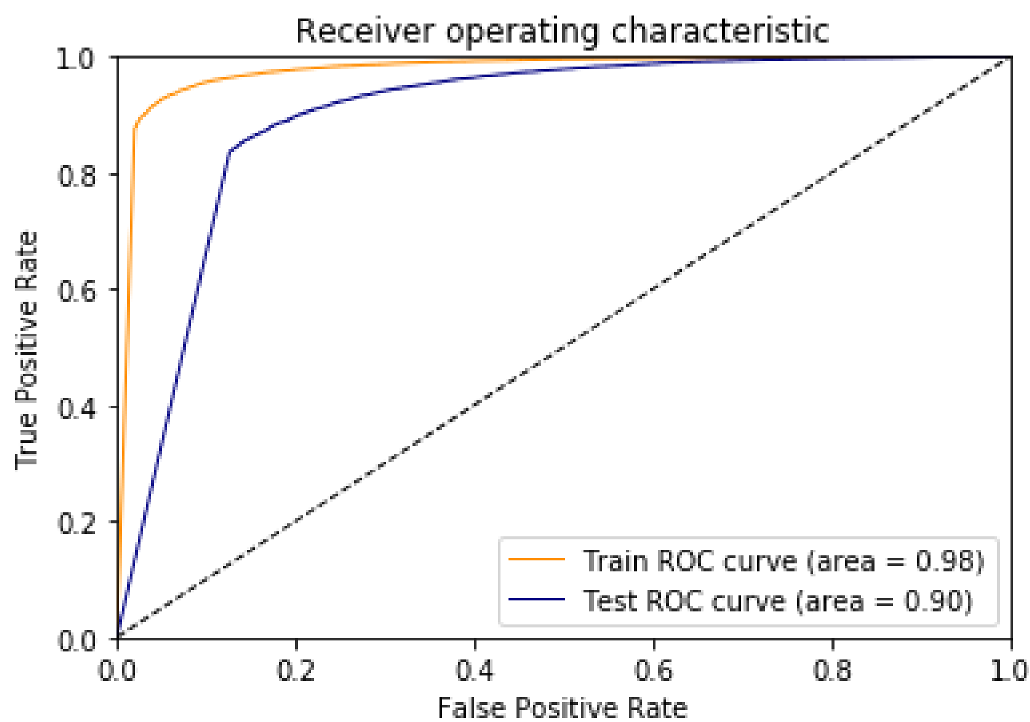


6.5 [5.2.1] Applying Naive Bayes on TFIDF, SET 2

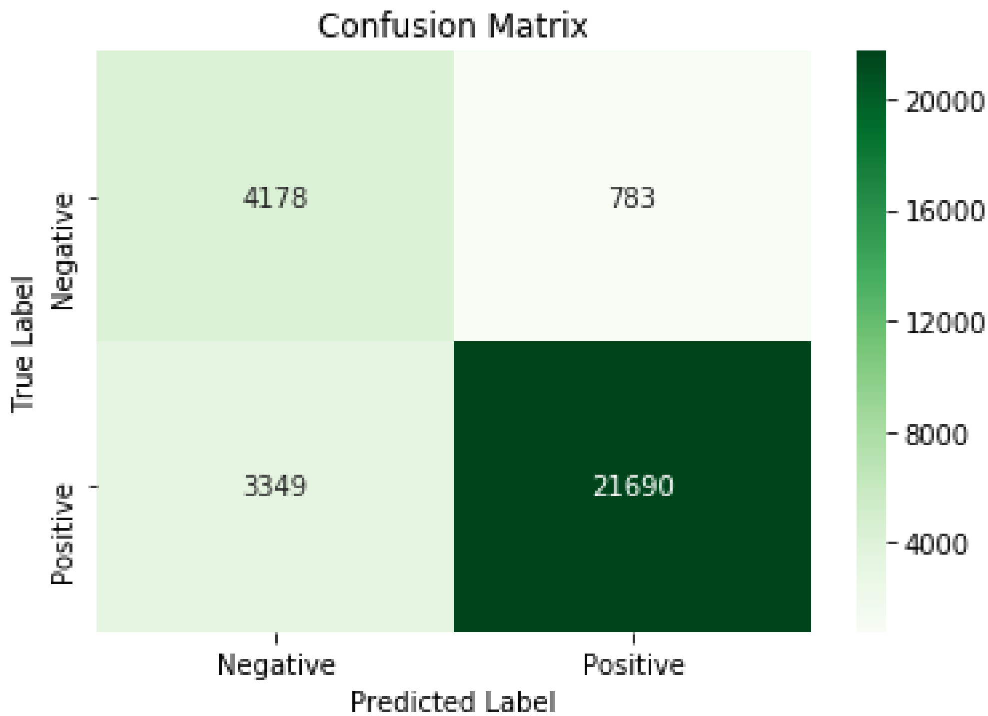
In [41]:
```
# Load the saved vectorized data for train-test datapoints
X_train_tfidf = pkl.load(open('train_tfidf.pkl', 'rb'))
X_test_tfidf = pkl.load(open('test_tfidf.pkl', 'rb'))

std = StandardScaler( with_mean=False)

# Standardizing the vectors
X_train_tfidf_std = std.fit_transform(X_train_tfidf)
X_test_tfidf_std = std.transform(X_test_tfidf)

# Getting an optimal value of hyperparameter alpha and AUC scores
# This data is used to plot a graph of alpha-values vs AUC
optimal_alpha, alpha_auc_train, alpha_auc_cv = get_optimal_alpha(X_train, y_train, 't
print("Optimal value of alpha : {}".format(optimal_alpha))

train_auc = [(alpha, train_auc) for alpha, train_auc in alpha_auc_train]
cv_auc = [(alpha, cv_auc) for alpha, cv_auc in alpha_auc_cv]

# Plotting K values vs AUC scores

plt.title("Alpha vs AUC")
plt.xlabel("Hyperparameter(alpha)")
plt.ylabel("Area under ROC curve")
plt.plot(*(zip(*train_auc)), label='Train_AUC')
plt.plot(*(zip(*cv_auc)), label='Validation_AUC')
plt.legend()
plt.show()

# Running Multinomial NB with optimal alpha value obtained
nb_clf, auc_score, conf_mat = run_nb(X_train_tfidf_std, y_train,
                                      X_test_tfidf_std, y_test, optimal_alpha)

print("AUC score:\n {:.2f}".format(auc_score))

# Plotting confusion matrix
plot_confusion_matrix(conf_mat)

auc_score = '%0.2f' % auc_score

# Adding the results to our results dataframe
results.loc[results.shape[0]] = ["Review Text", "TF-IDF", optimal_alpha, auc_score]
```

```
Optimal value of alpha : 100
```


Alpha vs AUC
Train_AUC
Validation_AUC
0.95
0.90
0.85
0.80
Area under ROC curve
0
20000
40000
60000
80000
100000
Hyperparameter(alpha)


Receiver operating characteristic
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
Train ROC curve (area = 0.98)
Test ROC curve (area = 0.90)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

```
AUC score:
 0.90
```



6.6 [5.2.2] Top 10 important features of positive class from SET 2

```
In [42]: # Load the tf-idf vectorizer we saved earlier
         tf_idf = pkl.load(open('tf_idf.pkl', 'rb'))
         # Top 10 features in positive class
         max_ind_pos=np.argsort((nb_clf.feature_log_prob_)[1])[::-1][0:10]
         print(np.take(count_vec.get_feature_names(), max_ind_pos))
```

```
['not' 'love' 'like' 'great' 'good' 'tast' 'flavor' 'one' 'use' 'tri']
```

6.7 [5.2.3] Top 10 important features of negative class from SET 2

```
In [43]: # Top 10 features in negative class
         max_ind_neg=np.argsort((nb_clf.feature_log_prob_)[0])[::-1][0:10]
         print(np.take(count_vec.get_feature_names(), max_ind_neg))
```

```
['not' 'tast' 'like' 'would' 'product' 'disappoint' 'bad' 'one' 'not buy'
 'tri']
```

6.8 [5.2.2] Naive Bayes on TFIDF with Review length as new feature SET 2

In [ ]:
```
# Load the saved vectorized data for train-test datapoints
X_train_tfidf = pkl.load(open('train_tfidf.pkl', 'rb'))
X_test_tfidf = pkl.load(open('test_tfidf.pkl', 'rb'))

# We will add review length as new feature with our vectorized
# data and train the model on this. We have calculated
# review length in main dataframe and has been added as a
# seperate column and can be accessed with X_train['Reviewlen']

X_train_revlen = np.array(X_train['Reviewlen'].values)
X_test_revlen = np.array(X_test['Reviewlen'].values)

# hstack here helps in adding the new review length column to the
# existing text vectors.

X_train_tfidf = hstack((X_train_tfidf,X_train_revlen[:,None])).tocsr()
X_test_tfidf = hstack((X_test_tfidf,X_test_revlen[:,None])).tocsr()

# We will standardize the whole data including review length
std = StandardScaler( with_mean=False)

# Standardizing the vectors
X_train_tfidf_std = std.fit_transform(X_train_tfidf)
X_test_tfidf_std = std.transform(X_test_tfidf)

# Getting an optimal value of hyperparameter alpha and AUC scores
# This data is used to plot a graph of alpha-values vs AUC
optimal_alpha, alpha_auc_train, alpha_auc_cv = get_optimal_alpha(X_train, y_train, 'tf
print("Optimal value of alpha : {}".format(optimal_alpha))

train_auc = [(alpha, train_auc) for alpha, train_auc in alpha_auc_train]
cv_auc = [(alpha, cv_auc) for alpha, cv_auc in alpha_auc_cv]

# Plotting K values vs AUC scores

plt.title("Alpha vs AUC")
plt.xlabel("Hyperparameter(alpha)")
plt.ylabel("Area under ROC curve")
plt.plot(*(zip(*train_auc)), label='Train_AUC')
plt.plot(*(zip(*cv_auc)), label='Validation_AUC')
plt.legend()
plt.show()

# Running Multinomial NB with optimal alpha value obtained
nb_clf, auc_score, conf_mat = run_nb(X_train_tfidf_std, y_train,
                                     X_test_tfidf_std, y_test, optimal_alpha)
```

```
print("AUC score:\n {:.2f}".format(auc_score))

# Plotting confusion matrix
plot_confusion_matrix(conf_mat)

auc_score = '%0.2f' % auc_score

# Adding the results to our results dataframe
results.loc[results.shape[0]] = ["Review Text/ Review Length", "TF-IDF", optimal_alpha
```

7 [6] Conclusions

1. AUC score in MultinomialNB for both BoW and TF-IDF is same i.e. 0.90 for this dataset
2. Adding review length as an extra feature does not improve or worsen the model. AUC stays at the same 0.90.
3. Even though the AUC and alpha looks same for all cases, there are only marginal differences in confusion matrices.
4. Top postive words for BoW

   'not' 'love' 'great' 'good' 'like' 'tast' 'one' 'tri' 'use' 'flavor'
5. Top negative words for BoW

   'not' 'tast' 'like' 'disappoint' 'product' 'would' 'bad' 'buy' 'not buy' 'one'
6. Top postive words for TF-IDF

   'not' 'love' 'like' 'great' 'good' 'tast' 'flavor' 'one' 'use' 'tri'
7. Top negative words for TF-IDF

   'not' 'tast' 'like' 'would' 'product' 'disappoint' 'bad' 'one' 'not buy' 'tri'

```
In [159]: print_results(results)
```

Features-Used	Vectorizer	Alpha	AUC
Review Text	BoW	100	0.90
Review Text/ Review Length	BoW	100	0.90
Review Text	TF-IDF	100	0.90
Review Text/ Review Length	TF-IDF	100	0.90