04 Amazon Fine Food Reviews Analysis_NaiveBayes

January 9, 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.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data 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 500

# 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 (5000, 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-R1105J5ZVQE25C  B005HG9ET0            Penguin Chick  1346889600      5
4  #oc-R12KPBODL2B5ZD  B0070SBE1U   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]:
```python
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]:
```python
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]:
```python
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=Fals
```

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

Out[9]: (4986, 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]: 99.72

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()
```

```
(4986, 10)
```

```
Out[13]: 1    4178
         0     808
         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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />http://www.amazon.c
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the oth
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great coffee
==================================================
```

```
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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victor
```

```
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)
```

```
Why is this $[...] when the same product is available for $[...] here? />The Victor M380 and M5
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the oth
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.  dcaf
```

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)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the otl
==================================================
```

```
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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
```

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

```
Wow So far two two star reviews One obviously had no idea what they were ordering the other wa
```

```
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%|| 4986/4986 [00:01<00:00, 3137.37it/s]
```

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

```
Out[23]: 'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey
```

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

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

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdominal
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

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/modu
```

```
        # 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_bigram_
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

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_name
         print('='*50)

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

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

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 [42]: # 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.bi
        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, t
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.99460321664810
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.999275088310
```

In [36]:
```
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])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby
```

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

[4.4.1.1] Avg W2v

In [38]:
```
# 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 t
    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]))
```

```
100%|| 4986/4986 [00:03<00:00, 1330.47it/s]


4986
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [39]: # 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 [41]: # 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 = tfidf

         tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this l
         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
```

```
100%|| 4986/4986 [00:20<00:00, 245.63it/s]
```

6 [5] Assignment 4: Apply Naive Bayes

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)
The hyper paramter tuning(find best Alpha)
Find the best hyper parameter which will give the maximum AUC value
Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
Find the best hyper paramter using k-fold cross validation or simple cross validation data
Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

```
</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

7.1 after preprocessing

```
In [2]: # Split the data into train , test and crossvalidation datasets
        # load "preprocessed.pkl" data frame
        df = pd.read_pickle("files/preprocessed.pkl")
        df.head(1)
```

```
Out[2]:              Id   ProductId         UserId      ProfileName  \
        138706  150524  0006641040  ACITT7DI6IDDL  shari zychinski

                HelpfulnessNumerator  HelpfulnessDenominator  Score       Time  \
        138706                     0                       0      1  939340800

                                   Summary  \
        138706  EVERY book is educational

                                                           Text  \
        138706  this witty little book makes my son laugh at l...

                                                    CleanedText  \
        138706  witty little book makes son laugh loud recite ...

                          CleanedSummary
        138706  every book educational
```

```
In [3]: df.shape
```

```
Out[3]: (364171, 12)
```

```
In [4]: # take 100k sample data randomly
        sample_data = df.sample(100000)
        sample_data.shape
```

Out[4]: (100000, 12)

In [5]:
```
# sorted the data using time based
sorted_data = sample_data.sort_values('Time', axis=0, inplace=False)
sorted_data.shape
```

Out[5]: (100000, 12)

In [6]:
```
sorted_data['Score'].value_counts()
```

Out[6]:
```
1    84518
0    15482
Name: Score, dtype: int64
```

In [7]:
```
from sklearn.model_selection import train_test_split
```

In [8]:
```
X = np.array(sorted_data['CleanedText'])
y = np.array(sorted_data['Score'])
print(X.shape)
print(y.shape)
```

```
(100000,)
(100000,)
```

In [9]:
```
# Simple cross validation
# split the data sent into train and test
train , test , train_y , test_y = train_test_split(X, y, test_size = 0.3, random_state=

# split the train data set into cross validation train and cross validation test
train, cv , train_y, cv_y = train_test_split(train, train_y, test_size=0.3, random_sta

print("train data = ", train.shape)
print("cros validation = ", cv.shape)
print("test data = ", test.shape)
```

```
train data =  (49000,)
cros validation =  (21000,)
test data =  (30000,)
```

In [ ]:

7.2 [5.1] Applying Naive Bayes on BOW, SET 1

In [10]:
```
# Please write all the code with proper documentation
#BoW
count_vect = CountVectorizer(min_df=15, ngram_range=(1,2)) #in scikit-learn
count_vect.fit(train)
print("some feature names ", count_vect.get_feature_names()[:10])
```

```
        print('='*50)

        bow_train = count_vect.fit_transform(train)
        bow_cv = count_vect.transform(cv)
        bow_test = count_vect.transform(test)
        print("============Train Data=========")
        print("the type of count vectorizer ",type(bow_train))
        print("the shape of out text BOW vectorizer ",bow_train.get_shape())
        print("the number of unique words ", bow_train.get_shape()[1])
        print("============Cross validation Data=========")
        print("the type of count vectorizer ",type(bow_cv))
        print("the shape of out text BOW vectorizer ",bow_cv.get_shape())
        print("the number of unique words ", bow_cv.get_shape()[1])
        print("============Test Data=========")
        print("the type of count vectorizer ",type(bow_test))
        print("the shape of out text BOW vectorizer ",bow_test.get_shape())
        print("the number of unique words ", bow_test.get_shape()[1])
```

```
some feature names  ['ability', 'able', 'able buy', 'able drink', 'able eat', 'able enjoy', 'a
==================================================
============Train Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (49000, 18510)
the number of unique words  18510
============Cross validation Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (21000, 18510)
the number of unique words  18510
============Test Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (30000, 18510)
the number of unique words  18510
```

```
In [11]: from sklearn.naive_bayes import MultinomialNB
         from sklearn.metrics import accuracy_score , f1_score , confusion_matrix
         from collections import Counter
         from sklearn.metrics import accuracy_score, roc_auc_score , roc_curve
         from sklearn.model_selection import train_test_split
```

```
In [12]: bow_train_auc = []
         bow_cv_auc = []
         hyperparameters = [pow(10,j) for j in range(-4,4,1)]
         for i in hyperparameters:
             mnb = MultinomialNB(alpha=i)
             mnb.fit(bow_train, train_y)
             # train data
             y_prob_train = mnb.predict_proba(bow_train)[:,1]
```

```
        y_pred = np.where(y_prob_train > 0.5, 1, 0)
        auc_roc_train = roc_auc_score(train_y , y_prob_train)
        print('\nTrain AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_train * float(1
        bow_train_auc.append(auc_roc_train)
        # CV
        y_prob_cv = mnb.predict_proba(bow_cv)[:,1]
        y_pred = np.where(y_prob_cv > 0.5, 1, 0)
        auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
        print('\nCV AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_cv * float(100))))
        bow_cv_auc.append(auc_roc_cv)
        print("="*50)
```

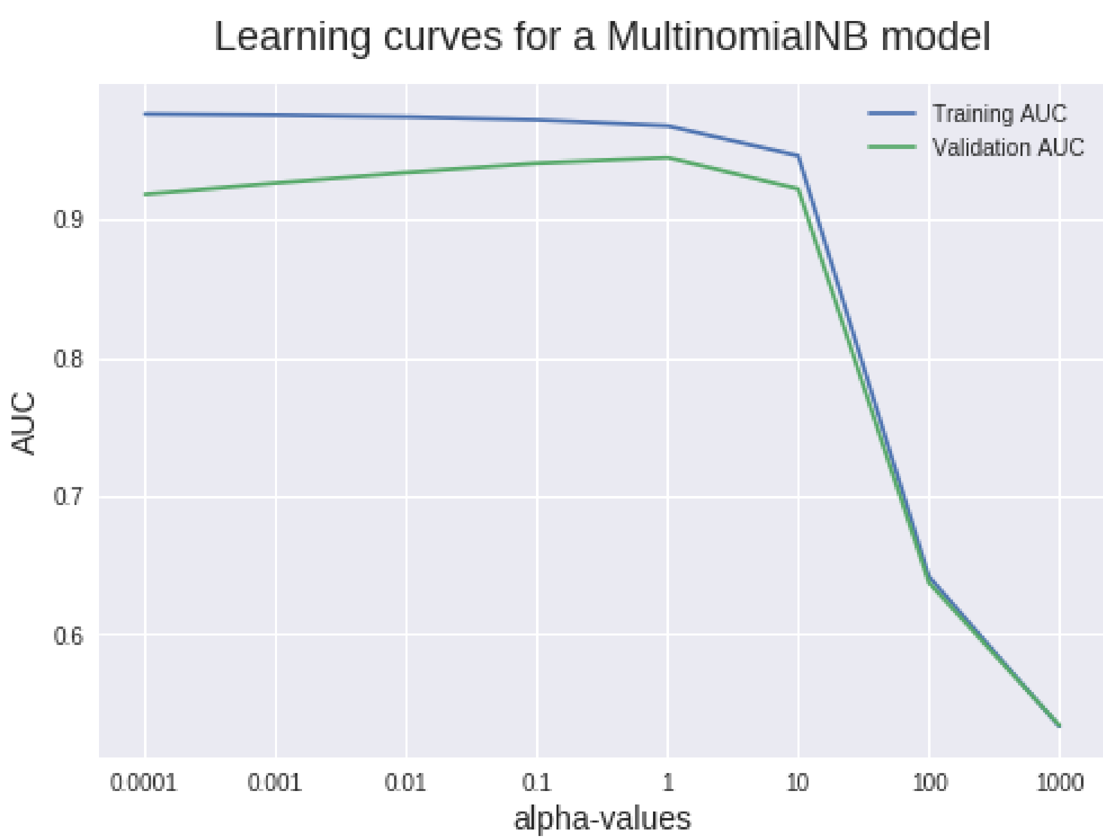
```
Train AUC for alpha = 0.0001 is 97.59%

CV AUC for alpha = 0.0001 is 91.79%
==================================================

Train AUC for alpha = 0.001 is 97.52%

CV AUC for alpha = 0.001 is 92.60%
==================================================

Train AUC for alpha = 0.01 is 97.39%

CV AUC for alpha = 0.01 is 93.37%
==================================================

Train AUC for alpha = 0.1 is 97.17%

CV AUC for alpha = 0.1 is 94.04%
==================================================

Train AUC for alpha = 1 is 96.74%

CV AUC for alpha = 1 is 94.45%
==================================================

Train AUC for alpha = 10 is 94.58%

CV AUC for alpha = 10 is 92.18%
==================================================

Train AUC for alpha = 100 is 64.19%

CV AUC for alpha = 100 is 63.74%
==================================================
```

```
Train AUC for alpha = 1000 is 53.37%

CV AUC for alpha = 1000 is 53.44%
==================================================
```

```
In [13]: hyper = [str(pow(10,j)) for j in range(-4,4)]
```

```
In [15]: # https://www.dataquest.io/blog/learning-curves-machine-learning/

         import matplotlib.pyplot as plt
         %matplotlib inline

         plt.style.use('seaborn')

         plt.plot(hyper,bow_train_auc,label = 'Training AUC')
         plt.plot(hyper, bow_cv_auc, label = 'Validation AUC')

         plt.ylabel('AUC', fontsize = 14)
         plt.xlabel('alpha-values', fontsize = 14)
         plt.title('Learning curves for a MultinomialNB model', fontsize = 18, y = 1.03)
         plt.legend()
```

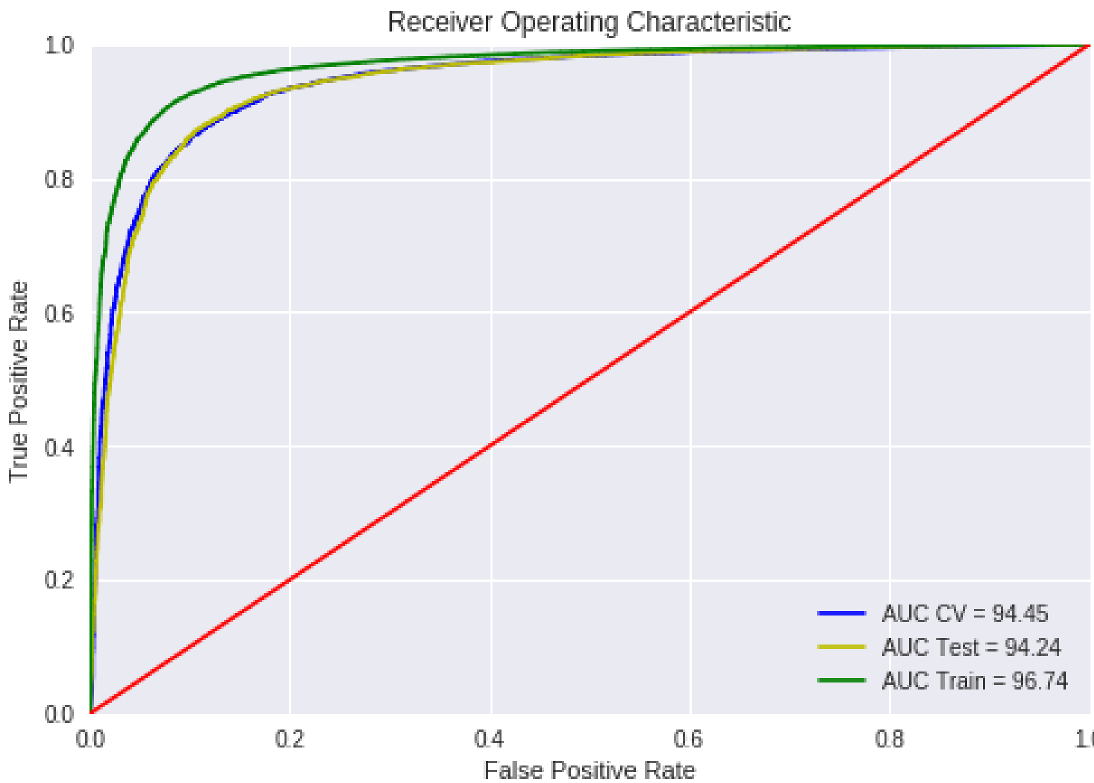
```
Out[15]: <matplotlib.legend.Legend at 0x7f7747574860>
```



In [16]:
```
i = 1
mnb = MultinomialNB(alpha=i)
mnb.fit(bow_train, train_y)
# train data
y_prob_train = mnb.predict_proba(bow_train)[:,1]
fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_train * float(100)))
# CV
y_prob_cv = mnb.predict_proba(bow_cv)[:,1]
fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_cv * float(100))))
# Test
y_prob_test = mnb.predict_proba(bow_test)[:,1]
fprts, tprts, thrsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , y_prob_test)
print('\nTest AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_test * float(100))))
```

```
Train AUC for alpha = 1 is 96.74%

CV AUC for alpha = 1 is 94.45%

Test AUC for alpha = 1 is 94.24%
```

In [17]:
```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100)))
plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

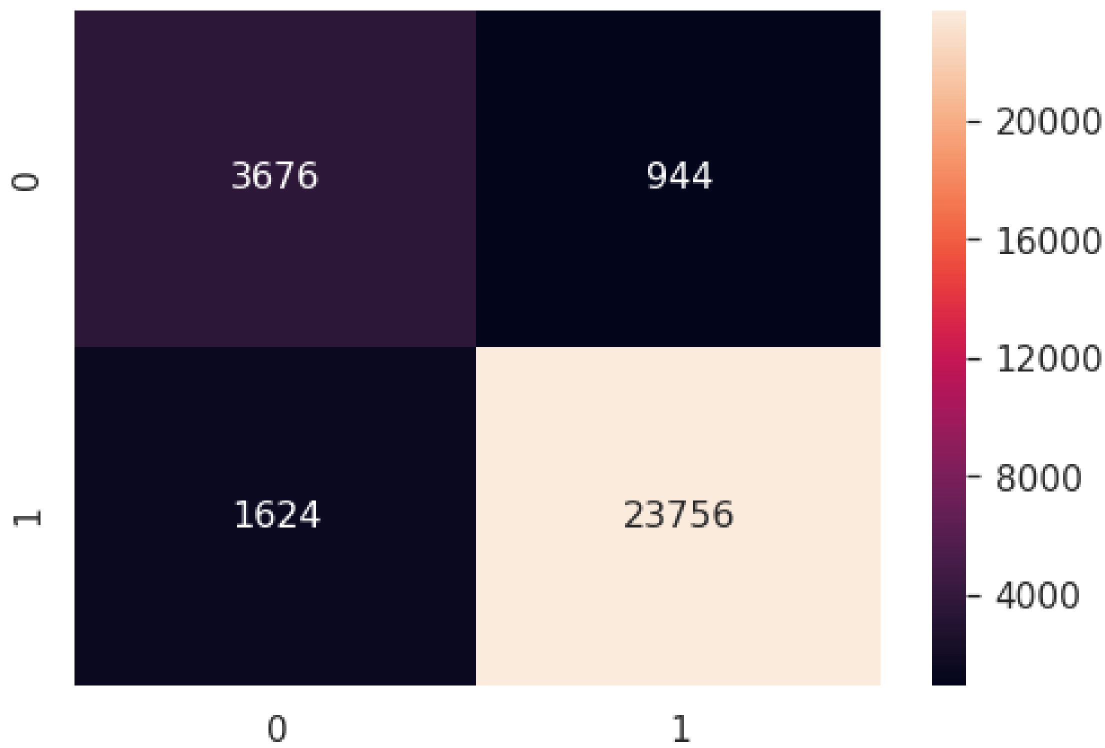



```
In [18]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.95
```

```
In [19]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
         sns.set(font_scale=1.4)
         sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7746c0b240>
```



In [ ]:

7.2.1 [5.1.1] Top 10 important features of positive class from SET 1

In [20]:
```
# Please write all the code with proper documentation
class_labels = mnb.classes_
feature_names = count_vect.get_feature_names()
positive_class = sorted(zip(mnb.feature_count_[1] ,  feature_names ,mnb.feature_log_p
print("words" +"\t\t" +"word count" + "\t"+ "log prob")
print("="*50)
for coef, feat, log in positive_class:
    print(feat + "\t\t" + str(coef) + "\t\t"+ str(round(log,3)))
```

```
words                   word count          log prob
==================================================
not                   38424.0               -3.924
like                   16934.0               -4.743
good                   14731.0               -4.882
great                   13778.0               -4.949
one                   11747.0               -5.109
taste                   11185.0               -5.158
product                   10088.0               -5.261
flavor                   10059.0               -5.264
```

```
tea                  9831.0               -5.287
love                  9818.0               -5.288
```

7.2.2 [5.1.2] Top 10 important features of negative class from SET 1

```
In [21]: # Please write all the code with proper documentation
         class_labels = mnb.classes_
         feature_names = count_vect.get_feature_names()
         positive_class = sorted(zip(mnb.feature_count_[0] ,  feature_names ,mnb.feature_log_pr
         print("words" +"\t\t" +"word count" + "\t"+ "log prob")
         print("="*50)
         for coef, feat, log in positive_class:
             print(feat + "\t\t" + str(coef) + "\t\t"+ str(round(log,3)))
```

```
words                word count        log prob
==================================================
not                  12212.0              -3.489
like                  3999.0              -4.605
product                  3189.0              -4.831
would                  3086.0              -4.864
taste                  2953.0              -4.908
one                  2521.0              -5.066
no                  1947.0              -5.325
coffee                  1920.0              -5.339
good                  1889.0              -5.355
flavor                  1747.0              -5.433
```

7.3 [5.2] Applying Naive Bayes on TFIDF, SET 2

```
In [22]: model = TfidfVectorizer(min_df=15, ngram_range=(1,2))
         #tf_idf_matrix = model.fit_transform(train)

         print("================Train Data===========")
         tf_idf_train = model.fit_transform(train)
         print("the type of count vectorizer ",type(tf_idf_train))
         print("the shape of out text TFIDF vectorizer ",tf_idf_train.get_shape())
         print("the number of unique words including both unigrams and bigrams ",tf_idf_train.g
         print("================CV Data===========")
         tf_idf_cv = model.transform(cv)
         print("the type of count vectorizer ",type(tf_idf_cv))
         print("the shape of out text TFIDF vectorizer ",tf_idf_cv.get_shape())
         print("the number of unique words including both unigrams and bigrams ",tf_idf_cv.get_
         print("================Test Data===========")
         tf_idf_test = model.transform(test)
         print("the type of count vectorizer ",type(tf_idf_test))
         print("the shape of out text TFIDF vectorizer ",tf_idf_test.get_shape())
```

```
print("the number of unique words including both unigrams and bigrams ", tf_idf_test.g

# 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_)))
```

```
================Train Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (49000, 18510)
the number of unique words including both unigrams and bigrams  18510
================CV Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (21000, 18510)
the number of unique words including both unigrams and bigrams  18510
================Test Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (30000, 18510)
the number of unique words including both unigrams and bigrams  18510
```

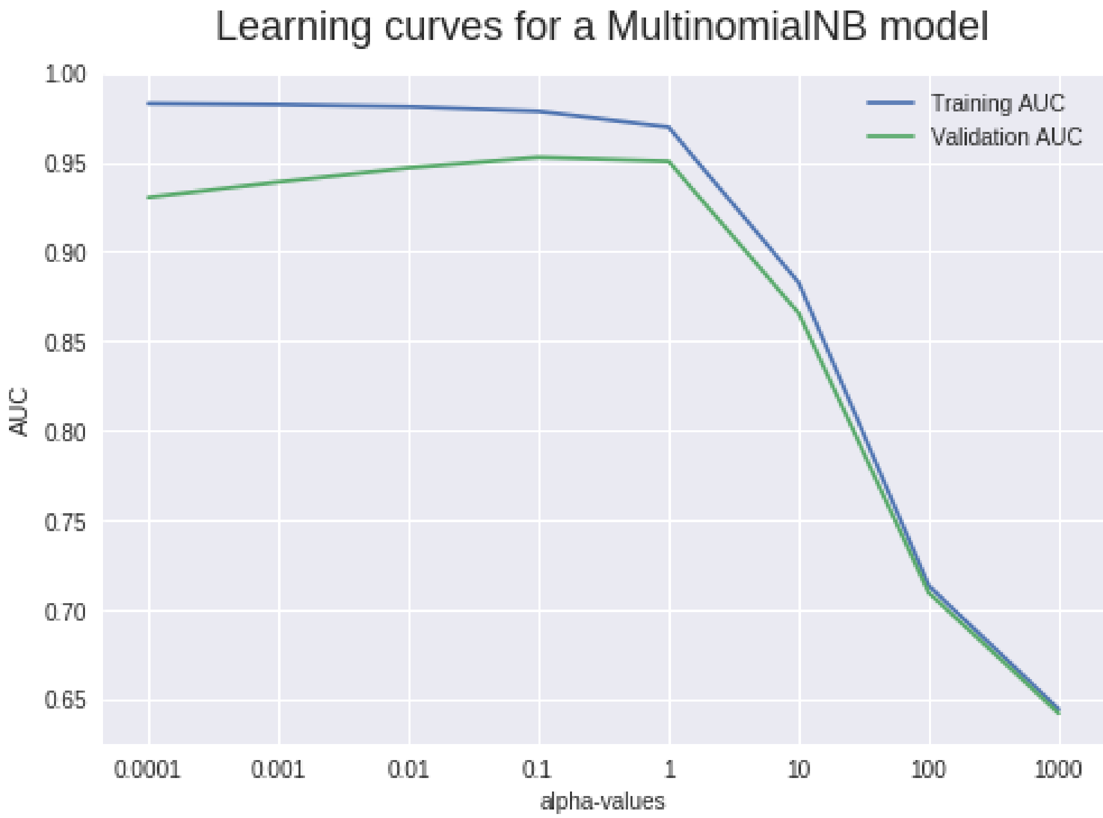
```
In [23]: tfidf_train_auc = []
         tfidf_cv_auc = []
         hyperparameters = [pow(10,j) for j in range(-4,4,1)]
         for i in hyperparameters:
             mnb = MultinomialNB(alpha=i)
             mnb.fit(tf_idf_train, train_y)
             # train data
             y_prob_train = mnb.predict_proba(tf_idf_train)[:,1]
             y_pred = np.where(y_prob_train > 0.5, 1, 0)
             auc_roc_train = roc_auc_score(train_y , y_prob_train)
             print('\nTrain AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_train * float(1
             tfidf_train_auc.append(auc_roc_train)
             # CV
             y_prob_cv = mnb.predict_proba(tf_idf_cv)[:,1]
             y_pred = np.where(y_prob_cv > 0.5, 1, 0)
             auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
             print('\nCV AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_cv * float(100)))
             tfidf_cv_auc.append(auc_roc_cv)
             print("="*50)
```

```
Train AUC for alpha = 0.0001 is 98.27%

CV AUC for alpha = 0.0001 is 93.04%
==================================================

Train AUC for alpha = 0.001 is 98.21%

CV AUC for alpha = 0.001 is 93.90%
```

```
==================================================

Train AUC for alpha = 0.01 is 98.09%

CV AUC for alpha = 0.01 is 94.68%
==================================================

Train AUC for alpha = 0.1 is 97.84%

CV AUC for alpha = 0.1 is 95.28%
==================================================

Train AUC for alpha = 1 is 96.96%

CV AUC for alpha = 1 is 95.06%
==================================================

Train AUC for alpha = 10 is 88.29%

CV AUC for alpha = 10 is 86.58%
==================================================

Train AUC for alpha = 100 is 71.36%

CV AUC for alpha = 100 is 70.96%
==================================================

Train AUC for alpha = 1000 is 64.47%

CV AUC for alpha = 1000 is 64.23%
==================================================
```

In [24]:
```
# https://www.dataquest.io/blog/learning-curves-machine-learning/

import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('seaborn')

plt.plot(hyper,tfidf_train_auc,label = 'Training AUC')
plt.plot(hyper, tfidf_cv_auc, label = 'Validation AUC')

plt.ylabel('AUC')
plt.xlabel('alpha-values', fontsize = 10)
plt.title('Learning curves for a MultinomialNB model', fontsize = 18, y = 1.03)
plt.legend()
```

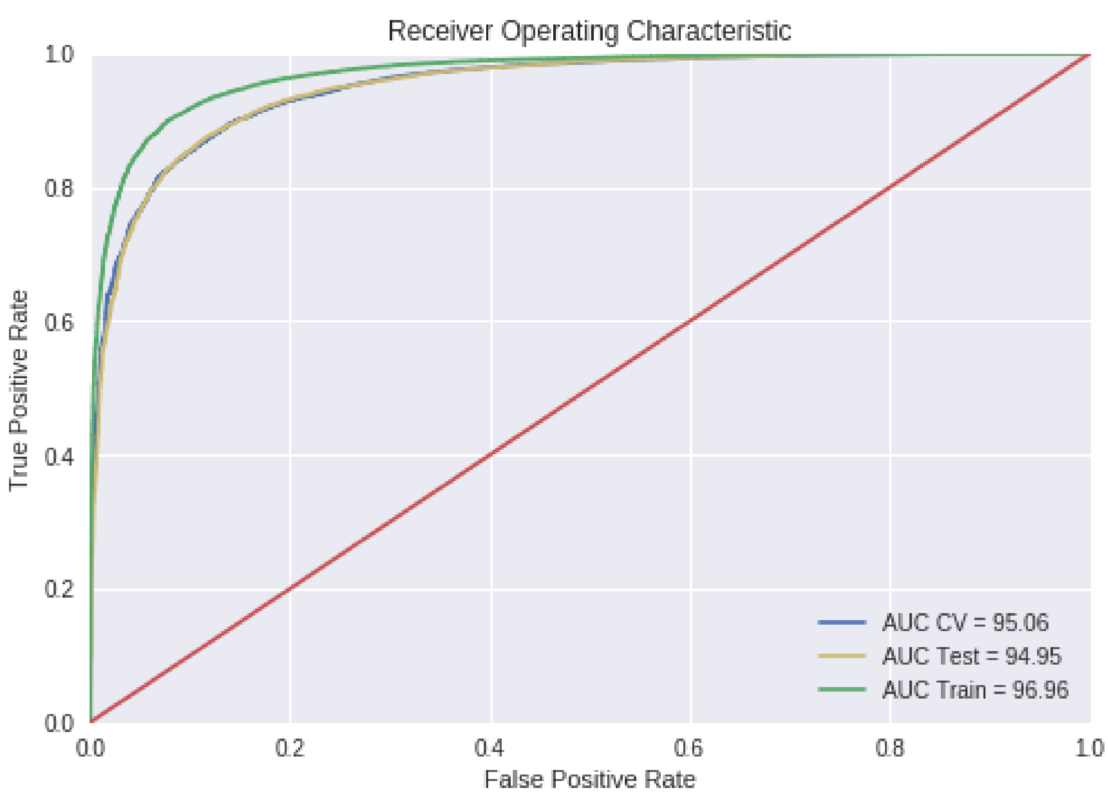
Out[24]:
```
<matplotlib.legend.Legend at 0x7f7744ffe160>
```



In [25]:
```
i = 1
mnb = MultinomialNB(alpha=i)
mnb.fit(tf_idf_train, train_y)
# train data
y_prob_train = mnb.predict_proba(tf_idf_train)[:,1]
fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_train * float(100)))
# CV
y_prob_cv = mnb.predict_proba(tf_idf_cv)[:,1]
fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_cv * float(100))))
# Test
y_prob_test = mnb.predict_proba(tf_idf_test)[:,1]
fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , y_prob_test)
print('\nTest AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_test * float(100))))
```

```
Train AUC for alpha = 1 is 96.96%

CV AUC for alpha = 1 is 95.06%

Test AUC for alpha = 1 is 94.95%
```

```
In [26]: # https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

         import matplotlib.pyplot as plt
         plt.clf()
         plt.title('Receiver Operating Characteristic')
         plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
         plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100)))
         plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
         plt.legend(loc = 'lower right')
         plt.plot([0, 1], [0, 1],'r')
         plt.xlim([0, 1])
         plt.ylim([0, 1])
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.show()
```

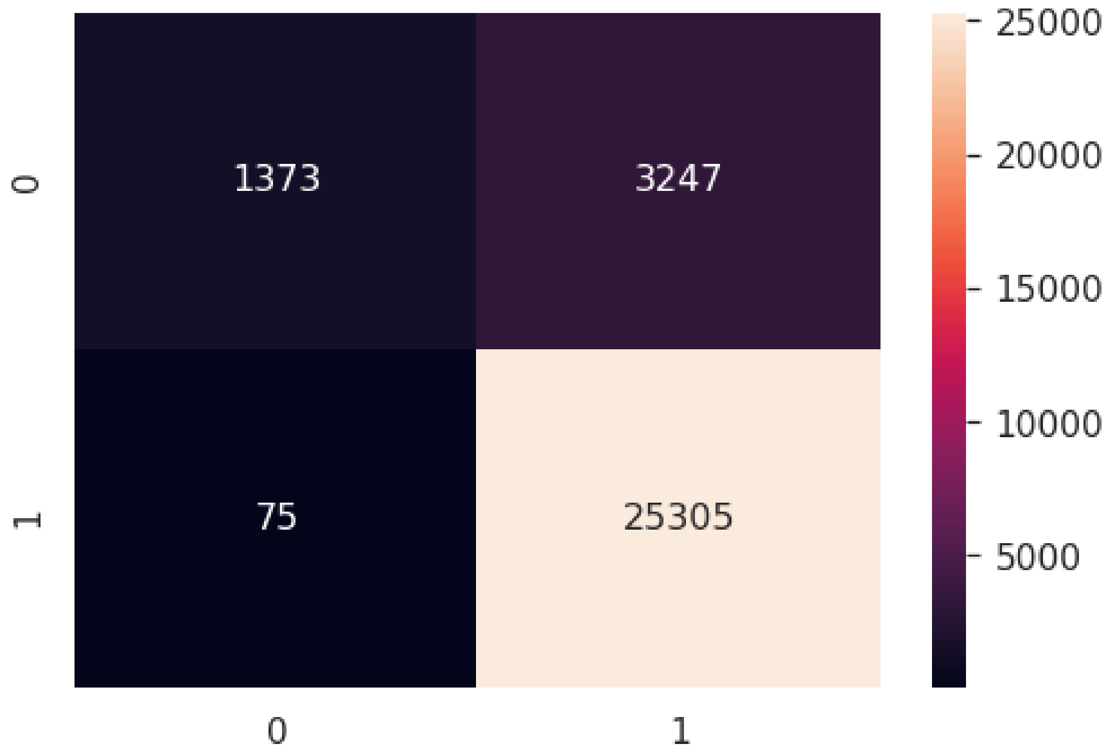


In [27]:
```
print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.94
```

In [28]:
```
df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f77466480b8>
```



7.3.1 [5.2.1] Top 10 important features of positive class from SET 2

In [29]:
```
# Please write all the code with proper documentation
class_labels = mnb.classes_
feature_names = count_vect.get_feature_names()
positive_class = sorted(zip(mnb.feature_count_[1] ,  feature_names ,mnb.feature_log_p
print("words" +"\t\t" +"word count" + "\t"+ "log prob")
print("="*50)
for coef, feat, log in positive_class:
    print(feat + "\t\t" + str(coef) + "\t\t"+ str(round(log,3)))
```

```
words                   word count        log prob
==================================================
not                   1305.1887177121612                  -5.237
great                   922.2006848624757                  -5.584
good                  866.2782934796204                  -5.647
like                  818.7008215983421                  -5.703
coffee                   733.5843747559663                  -5.813
tea                   727.4206070558514                  -5.821
love                  723.7406761001374                  -5.826
product                   672.7526060531211                  -5.899
taste                   647.4250733460474                  -5.938
one                   632.1127473412029                  -5.962
```

7.3.2 [5.2.1] Top 10 important features of Negative class from SET 1

In [30]:
```
# Please write all the code with proper documentation
class_labels = mnb.classes_
feature_names = count_vect.get_feature_names()
positive_class = sorted(zip(mnb.feature_count_[0] ,  feature_names ,mnb.feature_log_p
print("words" +"\t\t" +"word count" + "\t"+ "log prob")
print("="*50)
for coef, feat, log in positive_class:
    print(feat + "\t\t" + str(coef) + "\t\t"+ str(round(log,3)))
```

```
words                   word count        log prob
==================================================
not                   436.1979416712799                  -4.943
like                   194.74406883968157                  -5.746
product                   184.71170902827004                  -5.799
would                   173.88832243172237                  -5.859
taste                   170.1450210474986                  -5.88
coffee                   132.72534322120146                  -6.127
one                   132.2469254494228                  -6.131
no                   119.98539473149333                  -6.227
flavor                   103.94290988532542                  -6.37
tea                   100.92137268789133                  -6.399
```

7.3.3 Feature engineering

Taking length of reviews as another feature

In [31]:
```
df.head(1)
```

Out[31]:
```
            Id   ProductId          UserId      ProfileName  \
138706  150524  0006641040  ACITT7DI6IDDL  shari zychinski

        HelpfulnessNumerator  HelpfulnessDenominator  Score       Time  \
```

```
138706                        0                       0      1  939340800

                           Summary  \
138706  EVERY book is educational


                                                     Text  \
138706  this witty little book makes my son laugh at l...

                                              CleanedText  \
138706  witty little book makes son laugh loud recite ...

                 CleanedSummary
138706  every book educational
```

In [32]:
```python
# new featrue function text leangth
def get_text_length(x):
    return np.array([len(t) for t in x]).reshape(-1, 1)
```

In [36]:
```python
# adding a new feature to dataframe as text_length
df['text_length'] = get_text_length(df["CleanedText"].values)
```

In [38]:
```python
df.tail()
```

Out[38]:
```
            Id   ProductId          UserId    ProfileName  \
178145  193174  B009RSR8HO   A4P6AN2L435PV         romarc
173675  188389  B009SF0TN6  A1L0GWGRK4BYPT  Bety Robinson
204727  221795  B009SR4OQ2  A32A6X5KCP7ARG        sicamar
5259      5703  B009WSNWC4   AMP7K1O84DH1T           ESTY
302474  327601  B009WVB40S  A3ME78KVX31T21           K'la

        HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
178145                     0                       0      1  1350432000
173675                     0                       0      1  1350518400
204727                     1                       1      1  1350604800
5259                       0                       0      1  1351209600
302474                     0                       0      1  1351123200

                                    Summary  \
178145                         LOVE!!  LOVE!!
173675  Amazing!! Great sauce for everything!
204727                          Awesome Taste
5259                                DELICIOUS
302474                                 Tasty!

                                                     Text  \
178145  LOVE, LOVE this sweetener!!  I use it in all m...
173675  You have to try this sauce to believe it! It s...
204727  I bought this Hazelnut Paste (Nocciola Spread)...
5259    Purchased this product at a local store in NY ...
```

```
302474  I purchased this to send to my son who's away ...

                                              CleanedText  \
178145  love love sweetener use baking unsweetened fla...
173675  try sauce believe starts little sweet honey ta...
204727  bought hazelnut paste nocciola spread local sh...
5259    purchased product local store ny kids love qui...
302474  purchased send son away college delivered righ...

                      CleanedSummary  text_length
178145                     love love          374
173675  amazing great sauce everything          241
204727                 awesome taste          100
5259                       delicious          114
302474                         tasty          176
```

In [39]:
```python
# take 100k sample data randomly
sample_data = df.sample(100000)
sample_data.shape

# sorted the data using time based
sorted_data = sample_data.sort_values('Time', axis=0, inplace=False)
sorted_data.shape
```

Out[39]: (100000, 13)

In [40]:
```python
sorted_data['Score'].value_counts()

from sklearn.model_selection import train_test_split
text_length = np.array(sorted_data['text_length'])
X = np.array(sorted_data['CleanedText'])
y = np.array(sorted_data['Score'])
print(X.shape)
print(y.shape)
print(text_length.shape)
```

```
(100000,)
(100000,)
(100000,)
```

In [53]:
```python
# Simple cross validation
# split the data sent into train and test
train , test, train_text, test_text , train_y , test_y = train_test_split(X,text_leng

# split the train data set into cross validation train and cross validation test
train, cv , train_text, cv_text, train_y, cv_y = train_test_split(train, train_text, 

print("train data = ", train.shape)
```

```
print("cros validation = ", cv.shape)
print("test data = ", test.shape)
print("train text = ",train_text.shape)
print("cv text = ",cv_text.shape)
print("test text = ", test_text.shape)
```

```
train data =  (49000,)
cros validation =  (21000,)
test data =  (30000,)
train text =  (49000,)
cv text =  (21000,)
test text =  (30000,)
```

```
In [55]: #BoW
         count_vect = CountVectorizer(min_df=15, ngram_range=(1,1)) #in scikit-learn
         count_vect.fit(train)
         print("some feature names ", count_vect.get_feature_names()[:10])
         print('='*50)

         bow_train = count_vect.fit_transform(train)
         bow_cv = count_vect.transform(cv)
         bow_test = count_vect.transform(test)
         print("============Train Data=========")
         print("the type of count vectorizer ",type(bow_train))
         print("the shape of out text BOW vectorizer ",bow_train.get_shape())
         print("the number of unique words ", bow_train.get_shape()[1])
         print("============Cross validation Data=========")
         print("the type of count vectorizer ",type(bow_cv))
         print("the shape of out text BOW vectorizer ",bow_cv.get_shape())
         print("the number of unique words ", bow_cv.get_shape()[1])
         print("============Test Data=========")
         print("the type of count vectorizer ",type(bow_test))
         print("the shape of out text BOW vectorizer ",bow_test.get_shape())
         print("the number of unique words ", bow_test.get_shape()[1])
```

```
some feature names  ['ability', 'able', 'absolute', 'absolutely', 'absorb', 'absorbed', 'absor
==================================================
============Train Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (49000, 7033)
the number of unique words  7033
============Cross validation Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (21000, 7033)
the number of unique words  7033
============Test Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

```
the shape of out text BOW vectorizer  (30000, 7033)
the number of unique words  7033
```

```
In [56]: # adding a text_length feature to sparse matrix
         from scipy.sparse import hstack
         bow_train = hstack((bow_train,train_text[:,None]))
         bow_cv = hstack((bow_cv,cv_text[:,None]))
         bow_test = hstack((bow_test,test_text[:,None]))
```

```
In [57]: #after adding new feature to sparce matrix
         print("bow train shape = ", bow_train.shape)
         print("bow cv shape = ", bow_cv.shape)
         print("bow text shape = ",bow_test.shape)
```

```
bow train shape =  (49000, 7034)
bow cv shape =  (21000, 7034)
bow text shape =  (30000, 7034)
```

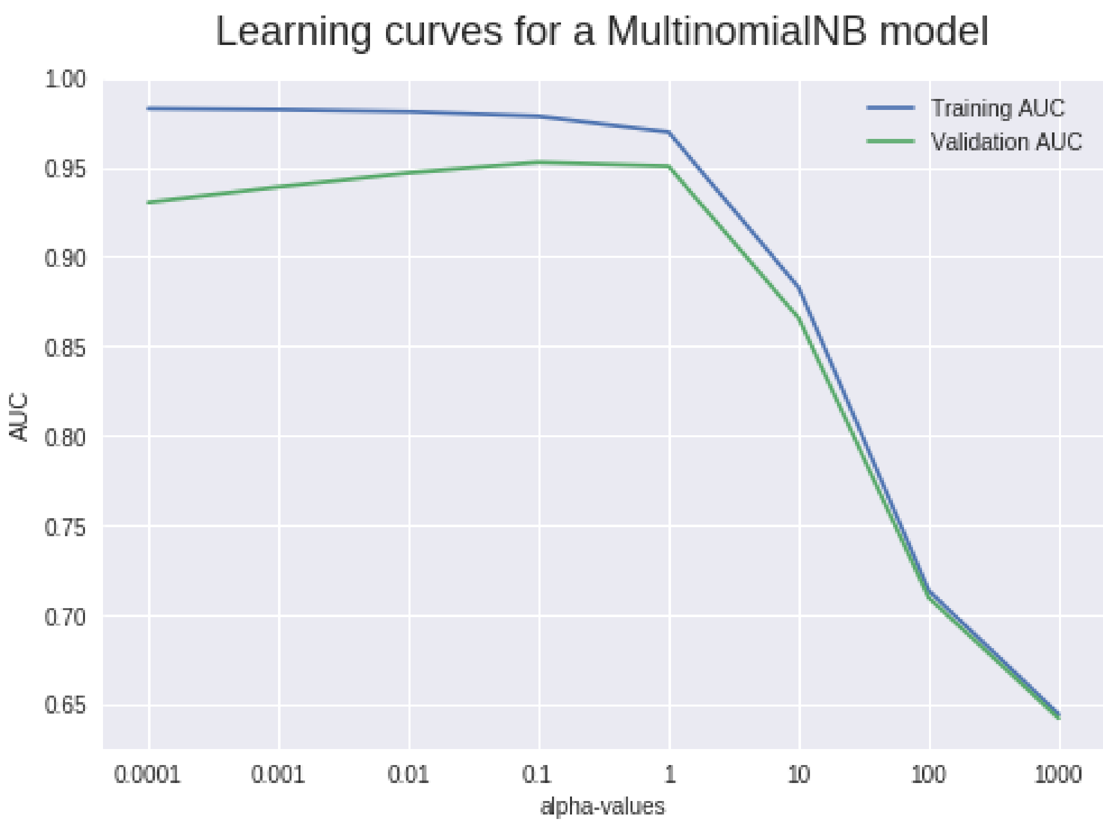
```
In [58]: bow_train_auc = []
         bow_cv_auc = []
         hyperparameters = [pow(10,j) for j in range(-4,4,1)]
         for i in hyperparameters:
             mnb = MultinomialNB(alpha=i)
             mnb.fit(bow_train, train_y)
             # train data
             y_prob_train = mnb.predict_proba(bow_train)[:,1]
             y_pred = np.where(y_prob_train > 0.5, 1, 0)
             auc_roc_train = roc_auc_score(train_y , y_prob_train)
             print('\nTrain AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_train * float(1
             bow_train_auc.append(auc_roc_train)
             # CV
             y_prob_cv = mnb.predict_proba(bow_cv)[:,1]
             y_pred = np.where(y_prob_cv > 0.5, 1, 0)
             auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
             print('\nCV AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_cv * float(100)))
             bow_cv_auc.append(auc_roc_cv)
             print("="*50)
```

```

Train AUC for alpha = 0.0001 is 94.79%

CV AUC for alpha = 0.0001 is 91.35%
==================================================

Train AUC for alpha = 0.001 is 94.77%

CV AUC for alpha = 0.001 is 91.76%
```

```
==================================================

Train AUC for alpha = 0.01 is 94.74%

CV AUC for alpha = 0.01 is 92.13%
==================================================

Train AUC for alpha = 0.1 is 94.67%

CV AUC for alpha = 0.1 is 92.47%
==================================================

Train AUC for alpha = 1 is 94.46%

CV AUC for alpha = 1 is 92.81%
==================================================

Train AUC for alpha = 10 is 92.63%

CV AUC for alpha = 10 is 91.40%
==================================================

Train AUC for alpha = 100 is 53.84%

CV AUC for alpha = 100 is 54.48%
==================================================

Train AUC for alpha = 1000 is 49.98%

CV AUC for alpha = 1000 is 50.06%
==================================================
```

In [59]:
```python
# https://www.dataquest.io/blog/learning-curves-machine-learning/

import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('seaborn')

plt.plot(hyper,tfidf_train_auc,label = 'Training AUC')
plt.plot(hyper, tfidf_cv_auc, label = 'Validation AUC')

plt.ylabel('AUC')
plt.xlabel('alpha-values', fontsize = 10)
plt.title('Learning curves for a MultinomialNB model', fontsize = 18, y = 1.03)
plt.legend()
```

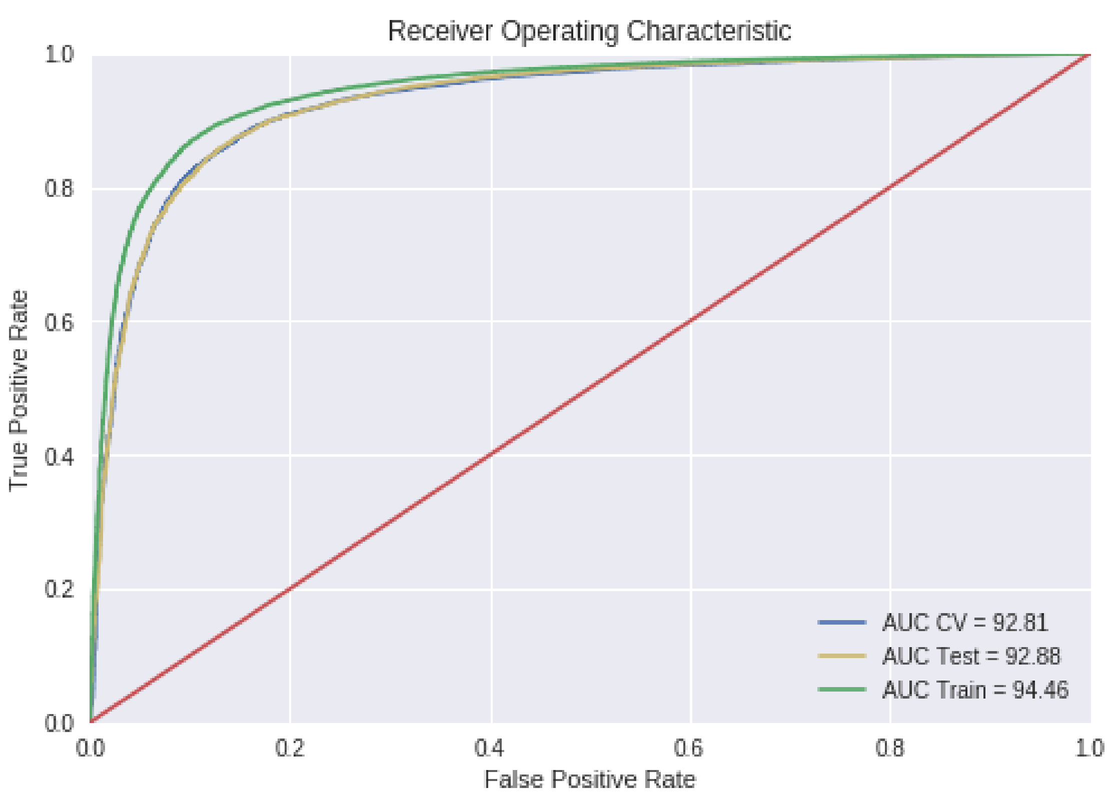
Out[59]: <matplotlib.legend.Legend at 0x7f774945dd30>



In [60]:
```
i = 1
mnb = MultinomialNB(alpha=i)
mnb.fit(bow_train, train_y)
# train data
y_prob_train = mnb.predict_proba(bow_train)[:,1]
fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_train * float(100)))
# CV
y_prob_cv = mnb.predict_proba(bow_cv)[:,1]
fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_cv * float(100))))
# Test
y_prob_test = mnb.predict_proba(bow_test)[:,1]
fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , y_prob_test)
print('\nTest AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_test * float(100))))
```

```
Train AUC for alpha = 1 is 94.46%

CV AUC for alpha = 1 is 92.81%

Test AUC for alpha = 1 is 92.88%
```

```
In [61]: # https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

         import matplotlib.pyplot as plt
         plt.clf()
         plt.title('Receiver Operating Characteristic')
         plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
         plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100)))
         plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
         plt.legend(loc = 'lower right')
         plt.plot([0, 1], [0, 1],'r')
         plt.xlim([0, 1])
         plt.ylim([0, 1])
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.show()
```

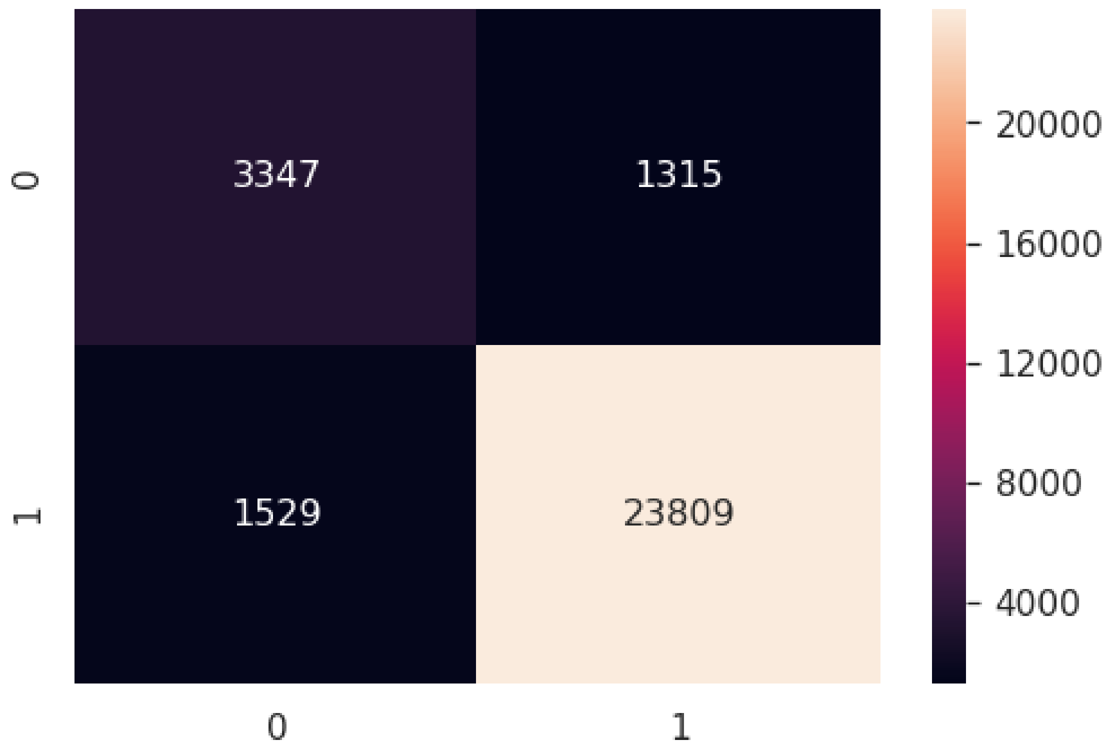


```
In [62]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.94
```

```
In [63]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
         sns.set(font_scale=1.4)
         sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x7f774eefec18>
```



7.3.4 Considering some features from review summary as well.

taking review summary as second model train bow model and take average averages of review text model and summary model

```
In [64]: X_review = np.array(sorted_data['CleanedText'])
         X_summary = np.array(sorted_data['Summary'])
         y = np.array(sorted_data['Score'])
         print(X_review.shape)
         print(X_summary.shape)
         print(y.shape)
```

```
(100000,)
(100000,)
(100000,)
```

```
In [66]: # Simple cross validation
         # split the data sent into train and test
         train_review , test_review, train_summary, test_summary , train_y , test_y = train_tes

         # split the train data set into cross validation train and cross validation test
         train_review, cv_review, train_summary, cv_summary , train_y, cv_y = train_test_split

         print("train review", train_review.shape)
         print("cv review", cv_review.shape)
         print("test review", test_review.shape)
         print("train summary", train_summary.shape)
         print("cv summary", cv_summary.shape)
         print("test summary", test_summary.shape)
```

```
train review (49000,)
cv review (21000,)
test review (30000,)
train summary (49000,)
cv summary (21000,)
test summary (30000,)
```

```
In [67]: # bow for summary
         # Please write all the code with proper documentation
         #BoW
         count_vect = CountVectorizer(min_df=15, ngram_range=(1,2)) #in scikit-learn
         count_vect.fit(train_summary)
         print("some feature names ", count_vect.get_feature_names()[:10])
         print('='*50)

         bow_train_summary = count_vect.fit_transform(train_summary)
         bow_cv_summary = count_vect.transform(cv_summary)
         bow_test_summary = count_vect.transform(test_summary)
         print("============Train Data=========")
         print("the type of count vectorizer ",type(bow_train_summary))
         print("the shape of out text BOW vectorizer ",bow_train_summary.get_shape())
         print("the number of unique words ", bow_train_summary.get_shape()[1])
         print("============Cross validation Data=========")
         print("the type of count vectorizer ",type(bow_cv_summary))
         print("the shape of out text BOW vectorizer ",bow_cv_summary.get_shape())
         print("the number of unique words ", bow_cv_summary.get_shape()[1])
         print("============Test Data=========")
         print("the type of count vectorizer ",type(bow_test_summary))
```

```
        print("the shape of out text BOW vectorizer ",bow_test_summary.get_shape())
        print("the number of unique words ", bow_test_summary.get_shape()[1])
```

```
some feature names  ['10', '100', '11', '12', '16', '24', '50', 'about', 'about this', 'absolu
==================================================
============Train Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (49000, 2269)
the number of unique words  2269
============Cross validation Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (21000, 2269)
the number of unique words  2269
============Test Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (30000, 2269)
the number of unique words  2269
```

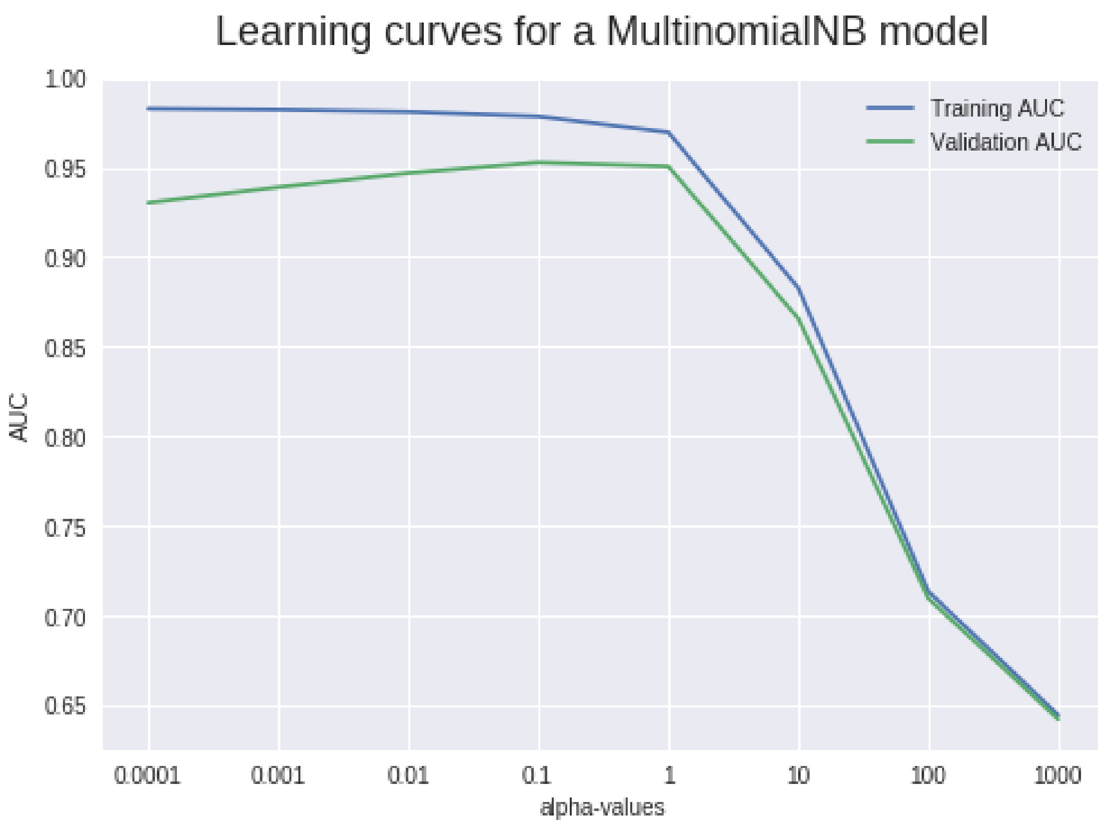
```
In [68]: bow_train_auc = []
         bow_cv_auc = []
         hyperparameters = [pow(10,j) for j in range(-4,4,1)]
         for i in hyperparameters:
             mnb = MultinomialNB(alpha=i)
             mnb.fit(bow_train_summary, train_y)
             # train data
             y_prob_train = mnb.predict_proba(bow_train_summary)[:,1]
             y_pred = np.where(y_prob_train > 0.5, 1, 0)
             auc_roc_train = roc_auc_score(train_y , y_prob_train)
             print('\nTrain AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_train * float(10
             bow_train_auc.append(auc_roc_train)
             # CV
             y_prob_cv = mnb.predict_proba(bow_cv_summary)[:,1]
             y_pred = np.where(y_prob_cv > 0.5, 1, 0)
             auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
             print('\nCV AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_cv * float(100)))
             bow_cv_auc.append(auc_roc_cv)
             print("="*50)
```

```
Train AUC for alpha = 0.0001 is 93.47%

CV AUC for alpha = 0.0001 is 90.88%
==================================================

Train AUC for alpha = 0.001 is 93.46%

CV AUC for alpha = 0.001 is 91.04%
```

```
=====================================================

Train AUC for alpha = 0.01 is 93.43%

CV AUC for alpha = 0.01 is 91.29%
=====================================================

Train AUC for alpha = 0.1 is 93.33%

CV AUC for alpha = 0.1 is 91.60%
=====================================================

Train AUC for alpha = 1 is 92.95%

CV AUC for alpha = 1 is 91.68%
=====================================================

Train AUC for alpha = 10 is 91.19%

CV AUC for alpha = 10 is 90.39%
=====================================================

Train AUC for alpha = 100 is 82.42%

CV AUC for alpha = 100 is 81.68%
=====================================================

Train AUC for alpha = 1000 is 73.45%

CV AUC for alpha = 1000 is 72.97%
=====================================================
```

In [69]:
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('seaborn')

plt.plot(hyper,tfidf_train_auc,label = 'Training AUC')
plt.plot(hyper, tfidf_cv_auc, label = 'Validation AUC')

plt.ylabel('AUC')
plt.xlabel('alpha-values', fontsize = 10)
plt.title('Learning curves for a MultinomialNB model', fontsize = 18, y = 1.03)
plt.legend()
```

Out[69]: <matplotlib.legend.Legend at 0x7f774f1e0dd8>




```
In [76]: # summary text
         i = 1
         mnb = MultinomialNB(alpha=i)
         mnb.fit(bow_train_summary, train_y)
         # Test
         y_prob_test_summary = mnb.predict_proba(bow_test_summary)[:,1]
         fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test_summary)
         y_pred_test = np.where(y_prob_test_summary > 0.5, 1, 0)
         auc_roc_test = roc_auc_score(test_y , y_prob_test_summary)
         print('\nTest AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_test * float(100))))
```

```
Test AUC for alpha = 1 is 91.57%
```

```
In [72]: # bow for review
         # Please write all the code with proper documentation
         #BoW
         count_vect = CountVectorizer(min_df=15, ngram_range=(1,2)) #in scikit-learn
         count_vect.fit(train_review)
         print("some feature names ", count_vect.get_feature_names()[:10])
```

```
print('='*50)

bow_train_review = count_vect.fit_transform(train_review)
bow_cv_review = count_vect.transform(cv_review)
bow_test_review = count_vect.transform(test_review)
print("============Train Data=========")
print("the type of count vectorizer ",type(bow_train_review))
print("the shape of out text BOW vectorizer ",bow_train_review.get_shape())
print("the number of unique words ", bow_train_review.get_shape()[1])
print("============Cross validation Data=========")
print("the type of count vectorizer ",type(bow_cv_review))
print("the shape of out text BOW vectorizer ",bow_cv_review.get_shape())
print("the number of unique words ", bow_cv_review.get_shape()[1])
print("============Test Data=========")
print("the type of count vectorizer ",type(bow_test_review))
print("the shape of out text BOW vectorizer ",bow_test_review.get_shape())
print("the number of unique words ", bow_test_review.get_shape()[1])
```

```
some feature names  ['ability', 'able', 'able buy', 'able eat', 'able enjoy', 'able find', 'ab
==================================================
============Train Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (49000, 18386)
the number of unique words  18386
============Cross validation Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (21000, 18386)
the number of unique words  18386
============Test Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (30000, 18386)
the number of unique words  18386
```

In [75]:
```
# review text
i = 1
mnb = MultinomialNB(alpha=i)
mnb.fit(bow_train_review, train_y)
# Test
y_prob_test_review = mnb.predict_proba(bow_test_review)[:,1]
fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test_review)
y_pred_test = np.where(y_prob_test_review > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , y_prob_test_review)
print('\nTest AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_test * float(100))))
```

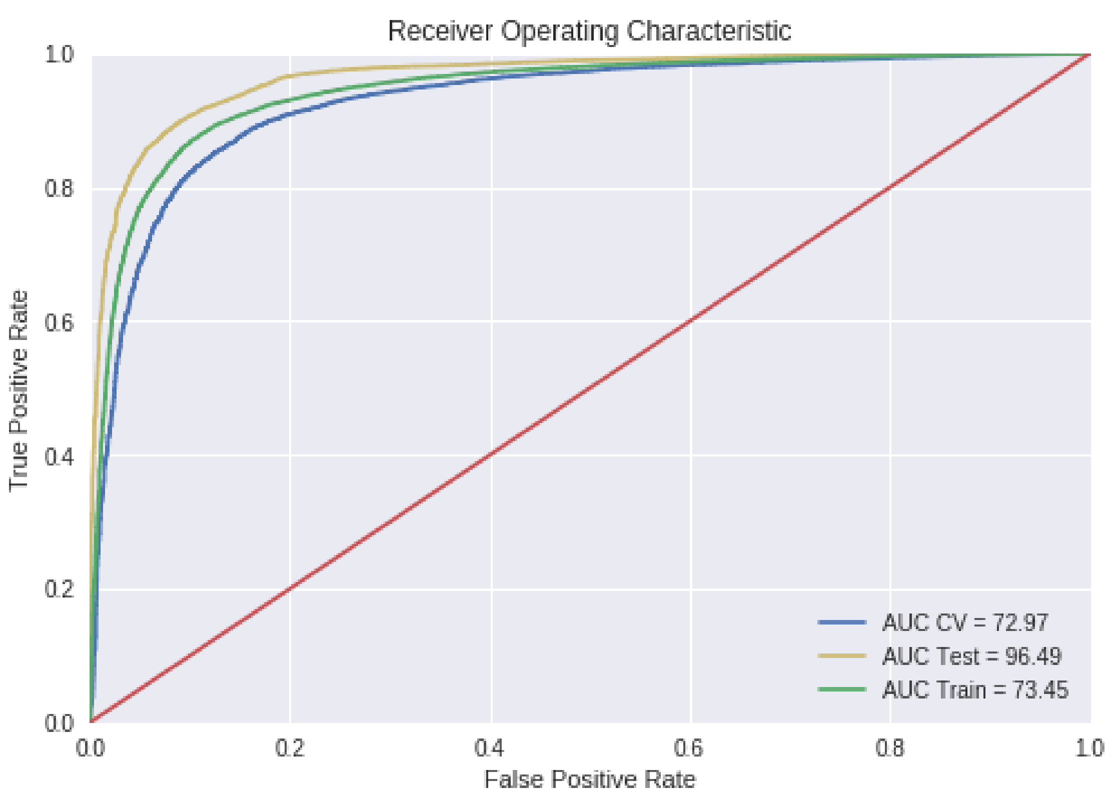
```
Test AUC for alpha = 1 is 94.42%
```

In [77]:
```
# adding both summary and review test probabilitys and average
new_proba = (y_prob_test_review + y_prob_test_summary) / 2
```

In [78]:
```
fprts, tprts, throsholdts = roc_curve(test_y, new_proba)
y_pred_test = np.where(new_proba > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , new_proba)
print('\nTest AUC for alpha = %s is %0.2f%%' % (str(i), (auc_roc_test * float(100))))
```

```
Test AUC for alpha = 1 is 96.49%
```

In [79]:
```
import matplotlib.pyplot as plt
plt.clf()
plt.title('Receiver Operating Characteristic')
plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * float(100)))
plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test * float(100)))
plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train * float(100)))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

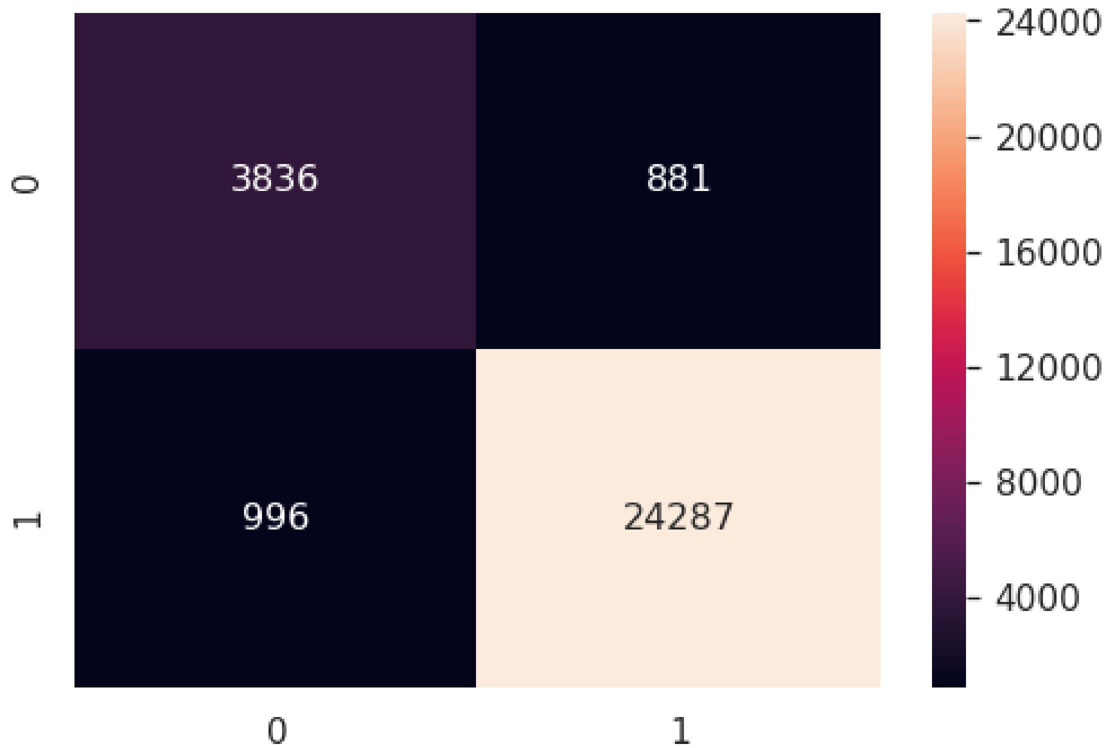


```
In [80]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.96
```

```
In [81]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
         sns.set(font_scale=1.4)
         sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x7f774a288eb8>
```



Observation : : adding two models we get better results. we see the 96.49% AUC and F1 score is 0.96

8 [6] Conclusions

```
In [84]: from prettytable import PrettyTable

         x = PrettyTable(["Vectorizer" , "Hyper parameter alpha", "AUC", "F1 Score"])
```

```
x.add_row(["BOW", 1, "94.24%", 0.95])
x.add_row(["TFIDF",1 , "94.95%", 0.94])

print(x.get_string(title="MNB Model"))
```

```
+------------+----------------------+--------+----------+
| Vectorizer | Hyper parameter alpha |  AUC   | F1 Score |
+------------+----------------------+--------+----------+
|    BOW     |          1           | 94.24% |   0.95   |
|   TFIDF    |          1           | 94.95% |   0.94   |
+------------+----------------------+--------+----------+
```

8.1 Feature engineering

```
In [86]: x = PrettyTable(["Feature", "Vectorizer" , "Hyper parameter alpha", "AUC", "F1 Score"]

         x.add_row(["Test length", "BOW", 1, "92.88%", 0.94])
         x.add_row(["Summary + Review", "BOW",1 , "96.49%", 0.96])

         print(x.get_string(title="MNB Model"))
```

```
+------------------+------------+----------------------+--------+----------+
|     Feature      | Vectorizer | Hyper parameter alpha |  AUC   | F1 Score |
+------------------+------------+----------------------+--------+----------+
|   Test length    |    BOW     |          1           | 92.88% |   0.94   |
| Summary + Review |    BOW     |          1           | 96.49% |   0.96   |
+------------------+------------+----------------------+--------+----------+
```

```
In [ ]:
```