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.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```python
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]:

```python
# using SQLite Table to read data.
con = sqlite3.connect('D:/AAIC/Data_Sets/amazon-fine-food-reviews/database.sqlite')

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

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", co
n)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", con)

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

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

```
Number of data points in our data (500000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	130386240
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	134697600

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	121901760

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

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

In [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(

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=False, kind='qui
cksort', na_position='last')
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=False)
final.shape
```

Out[9]:

```
(348262, 10)
```

In [10]:

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

Out[10]:

```
69.6524
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from 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	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883

In [12]:

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

In [13]:

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

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

```
(348260, 10)
```

Out[13]:

```
1    293516
0     54744
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [14]:

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

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

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

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

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and hugg
ed it when told it was his to keep and he did not have to return it to the library.
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Arabica.  My vote is definitel
y with the 100% Arabica.  The flavor has more bite and flavor (much more like European coffee than
American).
==================================================
This is a great product. It is very healthy for all of our dogs, and it is the first food that the
y all love to eat. It helped my older dog lose weight and my 10 year old lab gain the weight he ne
eded to be healthy.
==================================================
I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a tenn
is party, perfect! They were the size of malted milk balls. Unfortunately, they arrived 3 days aft
er the party. The caveat here is, not everything from Amazon may arrive at an impressive 2 or 3 da
ys. This shipment took 8 days from the Candy/Cosmetic Depot back east to southern California.
==================================================
```

In [15]:

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

print(sent_0)
```

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

In [16]:

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

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

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

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

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

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and hugg
ed it when told it was his to keep and he did not have to return it to the library.
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Arabica.  My vote is definitel
y with the 100% Arabica.  The flavor has more bite and flavor (much more like European coffee than
American).
==================================================
This is a great product. It is very healthy for all of our dogs, and it is the first food that the
y all love to eat. It helped my older dog lose weight and my 10 year old lab gain the weight he ne
eded to be healthy.
==================================================
I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a tenn
is party, perfect! They were the size of malted milk balls. Unfortunately, they arrived 3 days aft
er the party. The caveat here is, not everything from Amazon may arrive at an impressive 2 or 3 da
ys. This shipment took 8 days from the Candy/Cosmetic Depot back east to southern California.
```

In [17]:

```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 [18]:

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

```
This is a great product. It is very healthy for all of our dogs, and it is the first food that the
```

```
y all love to eat. It helped my older dog lose weight and my 10 year old lab gain the weight he ne
eded to be healthy.
==================================================
```

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)
```

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

In [20]:

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

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

In [21]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren'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 stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████| 348260/348260 [03:13<00:00, 1802.26it/s]
```

[3.2] Preprocessing Review Summary

In [23]:

```
#preprocessing review summary
preprocessed_review_summary=[]
for sentance in tqdm(final['Summary'].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 stopwords)
    preprocessed_review_summary.append(sentance.strip())
```

```
100%|██████████████████████████████████| 348260/348260 [02:00<00:00, 2883.72it/s]
```

In [24]:

```
sample = preprocessed_reviews[:50]
sample_sum = preprocessed_review_summary[:50]

a = preprocessed_reviews[0] + preprocessed_review_summary[0]
```

[4] Featurization

[4.1] Applying Naive Bayes on BOW, SET 1

In [56]:

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

#Breaking into Train and test
X_train, X_test, Y_train, Y_test = train_test_split(preprocessed_reviews,final['Score'].values,test
_size=0.3,shuffle=False)

#Text -> Uni gram Vectors ---- Featurization
uni_gram = CountVectorizer()
X_train = uni_gram.fit_transform(X_train)
#Normalize Data
X_train = preprocessing.normalize(X_train)
print("Train Data Size: ",X_train.shape)
X_test = uni_gram.transform(X_test)
#Normalize Data
X_test = preprocessing.normalize(X_test)
print("Test Data Size: ",X_test.shape)
```

```
Train Data Size:  (243782, 94165)
Test Data Size:  (104478, 94165)
```

In [26]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_validate

MNB = MultinomialNB()
```

In [27]:

```
%%time
Alpha_val = [1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0003, 0.0001,0.00001]
scores = []

for i in tqdm(Alpha_val):
    MNB.alpha = i
    score = cross_validate(MNB, X_train, Y_train, cv=10, scoring=
('precision','roc_auc'),return_train_score=True)
    scores.append(score)
```

```
100%|████████████████████████████████████████| 17/17 [01:58<00:00,  7.01s/it]
```

```
Wall time: 1min 58s
```

In [51]:

```
#Val_MSE = []
Val_prec =[]
for_opt_hyp = []
train_prec = []
Val_Roc_AUC_bow=[]
train_Roc_AUC_bow=[]

for i in tqdm(scores):
    #Val_MSE.append(-i.get('test_neg_mean_squared_error').mean()*100)
    Val_prec.append(i.get('test_precision').mean()*100)
    #train_MSE.append(-i.get('train_neg_mean_squared_error').mean()*100)
    train_prec.append(i.get('train_precision').mean()*100)
    Val_Roc_AUC_bow.append(i.get('test_roc_auc').mean()*100)
    train_Roc_AUC_bow.append(i.get('train_roc_auc').mean()*100)
#am = MNB.predict(X_test)
```

```
100%|████████████████████████████████████████| 17/17 [00:00<00:00, 8484.43it/s]
```

In [54]:

```
Best_Alpha_bow = Alpha_val[train_Roc_AUC_bow.index(max(train_Roc_AUC_bow))]
print("The hypermeter with highest AUC value is:", Best_Alpha,"at",max(Val_Roc_AUC_bow))
```

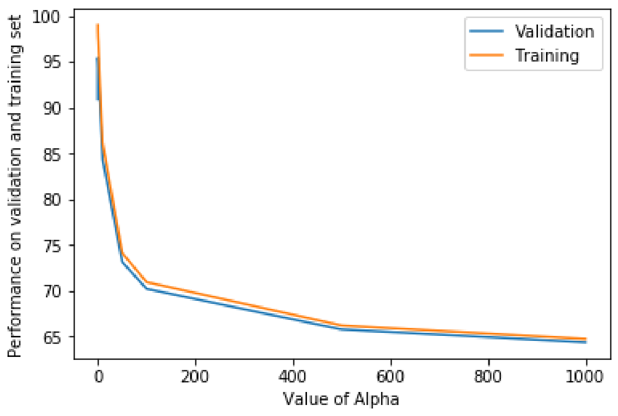
```
The hypermeter with highest AUC value is: 1e-05 at 95.42193624634383
```

In [55]:

```
plt.plot(Alpha_val, Val_Roc_AUC_bow, label="Validation")
plt.plot(Alpha_val, train_Roc_AUC_bow, label="Training")
plt.xlabel("Value of Alpha");
plt.ylabel("Performance on validation and training set")
plt.legend()
```

Out[55]:

```
<matplotlib.legend.Legend at 0x44b48f6cc0>
```



Multinomial NB for prediction on test data

In [57]:

```
MNB_for_pred = MultinomialNB(alpha=Best_Alpha)
MNB_for_pred.fit(X_train,Y_train)
Pred_test = MNB_for_pred.predict(X_test)
Pred_Prob_test = MNB_for_pred.predict_proba(X_test)[:,1]
```

Top 10 and least 10 important features of positive class from SET 1

In [58]:

```
#Class 0 features - the most 20 importan
def show_most_informative_features(vectorizer, clf, n=20):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.feature_log_prob_[0], feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t%.4f\t%-15s"%(coef_1, fn_1, coef_2, fn_2))
```

The show_most_informative_features function is referred from here

In [59]:

```
show_most_informative_features(uni_gram, MNB_for_pred)
```

```
 -23.6690 aaaa            -3.2155 not
 -23.6690 aaaaaa          -4.3836 like
 -23.6690 aaaaaaaaaaa     -4.5407 product
 -23.6690 aaaaaaaaaaaa    -4.6238 taste
 -23.6690 aaaaaaaaaaaaa   -4.6577 would
 -23.6690 aaaaaaaaaaaaaaa  -4.8480 one
 -23.6690 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  -5.0342 good
 -23.6690 aaaaaaaaaaaaaaaaaaaaaargh  -5.0930 no
 -23.6690 aaaaaaaaaagghh   -5.1540 flavor
 -23.6690 aaaaaaah         -5.3076 buy
 -23.6690 aaaaaaand        -5.3344 coffee
 -23.6690 aaaaah           -5.3841 tea
 -23.6690 aaaaahhhhhhhhhhhhhhh  -5.3915 get
 -23.6690 aaaaallll        -5.4289 amazon
 -23.6690 aaaaawsome       -5.4313 even
 -23.6690 aaaah            -5.4570 food
 -23.6690 aaaahhhhhh       -5.5193 much
 -23.6690 aaaahhhhhhhhhhh  -5.5675 bought
 -23.6690 aaaallll         -5.5765 really
 -23.6690 aaaand           -5.6373 box
```

Top 10 and least 10 important features of negative class from SET 1

In [60]:

```
def show_most_informative_features(vectorizer, clf, n=20):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.feature_log_prob_[1], feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t%.4f\t%-15s"%(coef_1, fn_1, coef_2, fn_2))
```

In [61]:

```
show_most_informative_features(uni_gram, MNB_for_pred)
```

```
 -25.3500 aaaaaaarrrrrggghhh  -3.7704 not
 -25.3500 aaaaaahhhhhyaaaaaa  -4.4728 great
 -25.3500 aachen          -4.5287 good
 -25.3500 aargh           -4.5980 like
 -25.3500 aarrgh          -4.8335 love
 -25.3500 aauces          -4.8713 product
 -25.3500 abandonment     -4.9202 one
 -25.3500 abbazabba       -4.9376 taste
 -25.3500 abbreviating    -4.9791 tea
 -25.3500 abel            -5.0480 flavor
 -25.3500 aberrations     -5.1334 coffee
 -25.3500 abhorrent       -5.1808 would
 -25.3500 abjectly        -5.2630 best
 -25.3500 abolitionists   -5.2924 amazon
 -25.3500 abominable      -5.3352 get
 -25.3500 abominably      -5.3779 use
 -25.3500 abominated      -5.3930 find
 -25.3500 abouthis        -5.3932 really
 -25.3500 abovei          -5.3991 price
 -25.3500 abreviating     -5.4032 no
```

In [65]:

```python
from sklearn.metrics import roc_auc_score, roc_curve
fpr_tr, tpr_tr, threshol_tr = roc_curve(Y_train, MNB_for_pred.predict(X_train))
fpr_ts, tpr_ts, threshold_ts = roc_curve(Y_test, Pred_Prob_test)
#print("The accuracy score is:",roc_auc_score(fpr,tpr))
```

In [67]:

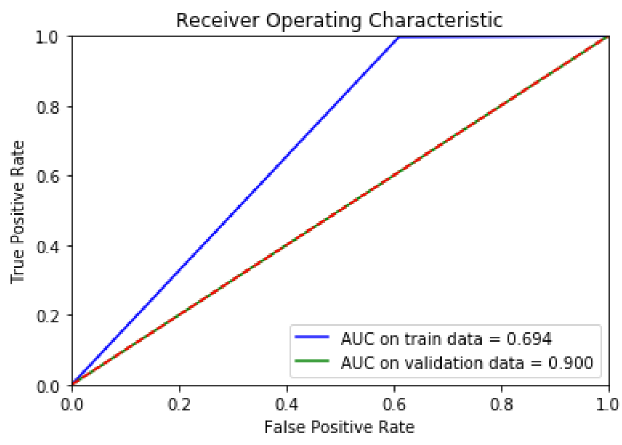
```python
AUC_Bow_tr = metrics.auc(fpr_tr, tpr_tr)
AUC_Bow_ts = metrics.auc(fpr_ts, tpr_ts)
print("The AUC value of train set is:",AUC_Bow_tr)
print("The AUC value of test set is:",AUC_Bow_ts)
```

```
The AUC value of train set is: 0.6935739366248278
The AUC value of test set is: 0.8998794355369393
```

In [69]:

```python
#Plotting ROC Curve
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr_tr, tpr_tr, 'b', label = 'AUC on train data = %0.3f' % AUC_Bow_tr)

plt.plot(fpr_ts, fpr_ts, 'g', label = 'AUC on validation data = %0.3f' % AUC_Bow_ts)
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()
```

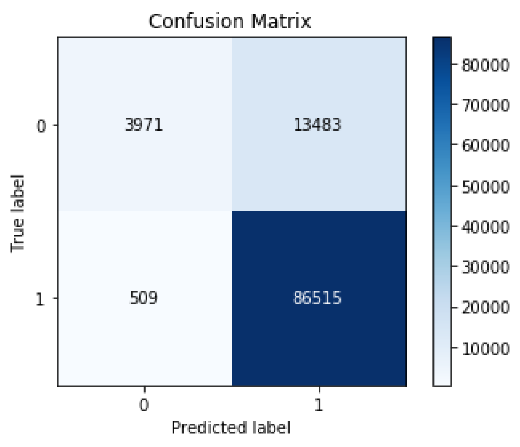


In [71]:

```
import scikitplot.metrics as skplt
skplt.plot_confusion_matrix(Y_test, Pred_test)

print("The precision score is:",round(metrics.precision_score(Y_test,Pred_test),4))
print("The recall value is:",round(metrics.recall_score(Y_test,Pred_test),4))
print("The F1 Score is:",round(metrics.f1_score(Y_test,Pred_test),4))
```

```
The precision score is: 0.8652
The recall value is: 0.9942
The F1 Score is: 0.9252
```



[4.2] Bi-Grams and n-Grams.

In [241]:

```
#bi-gram, tri-gram and n-gram

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

# you can choose these 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_counts.get_s
hape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (348260, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [258]:

```
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_names()[0:10])
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.get_shape()[
1])
```

```
some sample features(unique words in the corpus) ['aa', 'aaa', 'aaaaa', 'aaah', 'aafco', 'ab', 'ab
ack', 'abandon', 'abandoned', 'abbey']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (348260, 194764)
the number of unique words including both unigrams and bigrams  194764
```

In [72]:

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

#Breaking into Train and test
X_train, X_test, Y_train, Y_test = train_test_split(preprocessed_reviews,final['Score'].values,test
_size=0.3,shuffle=False)

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

#Text -> Uni/bi gram Vectors ---- Featurization
X_train = tf_idf_vect.fit_transform(X_train)

#Normalize Data - to get rid of outliers
X_train = preprocessing.normalize(X_train)
print("Train Data Size: ",X_train.shape)

X_test = tf_idf_vect.transform(X_test)
#Normalize Data
X_test = preprocessing.normalize(X_test)
print("Test Data Size: ",X_test.shape)
```

```
Train Data Size:  (243782, 136004)
Test Data Size:  (104478, 136004)
```

In [31]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_validate

MNB_tf = MultinomialNB()
```

In [32]:

```
%%time
Alpha_val = [1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0003, 0.0001,0.00001]
scores = []

for i in tqdm(Alpha_val):
    MNB_tf.alpha = i
    score = cross_validate(MNB_tf, X_train, Y_train, cv=10, scoring=
('precision','roc_auc'),return_train_score=True)
    scores.append(score)
```

```
100%|██████████████████████████████████████████| 17/17 [02:52<00:00,  9.96s/it]
```

```
Wall time: 2min 52s
```

In [33]:

```
Val_prec =[]
for_opt_hyp = []
train_prec = []
Val_Roc_AUC_tf=[]
train_Roc_AUC_tf=[]

for i in tqdm(scores):
    Val_prec.append(i.get('test_precision').mean()*100)
    train_prec.append(i.get('train_precision').mean()*100)
    Val_Roc_AUC_tf.append(i.get('test_roc_auc').mean()*100)
    train_Roc_AUC_tf.append(i.get('train_roc_auc').mean()*100)
```

```
100%|███████████████████████████████████████| 17/17 [00:00<00:00, 16997.18it/s]
```

In [35]:

```
Best_Alpha_tf = Alpha_val[train_Roc_AUC_tf.index(max(train_Roc_AUC_tf))]
print("The hypermeter with highest AUC value is:", Best_Alpha_tf,"at",max(train_Roc_AUC_tf))
```

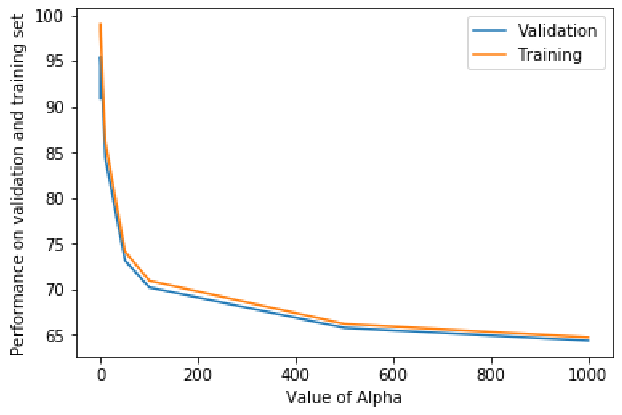
```
The hypermeter with highest AUC value is: 1e-05 at 99.01623668660926
```

In [36]:

```
plt.plot(Alpha_val, Val_Roc_AUC_tf, label="Validation")
plt.plot(Alpha_val, train_Roc_AUC_tf, label="Training")
plt.xlabel("Value of Alpha");
plt.ylabel("Performance on validation and training set")
plt.legend()
```

Out[36]:

```
<matplotlib.legend.Legend at 0x44435ce9e8>
```



Multinomial NB for prediction on test data

In [37]:

```
MNB_for_pred = MultinomialNB(alpha=Best_Alpha_tf)
MNB_for_pred.fit(X_train,Y_train)
Pred_test = MNB_for_pred.predict(X_test)
Pred_Prob_test = MNB_for_pred.predict_proba(X_test)[:,1]
```

Top 10 and least 10 important features of positive class from SET 2

In [38]:

```
#Class 0 features - the most 20 importan
def show_most_informative_features(vectorizer, clf, n=20):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.feature_log_prob_[0], feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t%.4f\t%-15s"%(coef_1, fn_1, coef_2, fn_2))
```

The show_most_informative_features function is referred from here

In [39]:

```
show_most_informative_features(tf_idf_vect, MNB_for_pred)
```

```
 -23.8713 able brew          -4.9275 not
 -23.8713 able carry         -5.7703 like
 -23.8713 able come          -5.8010 product
 -23.8713 able destroy       -5.8902 would
 -23.8713 able leave         -5.8943 taste
 -23.8713 able offer         -6.1366 one
 -23.8713 able offer       -6.15[illegible] one
 -23.8713 able produce     -6.2475 no
 -23.8713 able protect     -6.3253 coffee
 -23.8713 able smell       -6.3559 tea
 -23.8713 able snack       -6.3592 flavor
 -23.8713 able something   -6.3889 buy
 -23.8713 able still       -6.3952 good
 -23.8713 able stock       -6.4677 food
 -23.8713 able stop        -6.4823 even
 -23.8713 able store       -6.5273 box
 -23.8713 able teach       -6.5434 get
 -23.8713 able track       -6.5495 bought
 -23.8713 abs              -6.5676 amazon
 -23.8713 absolute delight  -6.5699 bad
 -23.8713 absolute favorites  -6.6280 could
```

Top 10 and least 10 important features of negative class from SET 2

In [40]:

```python
def show_most_informative_features(vectorizer, clf, n=20):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.feature_log_prob_[1], feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t%.4f\t%-15s"%(coef_1, fn_1, coef_2, fn_2))
```

In [41]:

```python
show_most_informative_features(tf_idf_vect, MNB_for_pred)
```

```
 -25.5533 absolutely worst  -5.5121 not
 -25.5533 according amazon  -5.8532 great
 -25.5533 allow return     -5.9262 good
 -25.5533 almost undrinkable  -5.9854 tea
 -25.5533 amazon ashamed   -6.0150 like
 -25.5533 avoid costs      -6.1010 love
 -25.5533 away guess       -6.1537 coffee
 -25.5533 awful feeding    -6.1664 product
 -25.5533 awful no         -6.2347 one
 -25.5533 awful texture    -6.2409 taste
 -25.5533 bad diarrhea     -6.2846 flavor
 -25.5533 bad threw        -6.4088 best
 -25.5533 balance blue     -6.4375 would
 -25.5533 beaks chicken    -6.4714 amazon
 -25.5533 believe positive  -6.5013 food
 -25.5533 bite threw       -6.5057 use
 -25.5533 bites threw      -6.5094 price
 -25.5533 box sticky       -6.5176 get
 -25.5533 buyers beware    -6.5206 find
 -25.5533 commercials cats  -6.5320 really
```

In [45]:

```python
from sklearn.metrics import roc_auc_score, roc_curve
fpr_tr, tpr_tr, threshol_tr = roc_curve(Y_train, MNB_for_pred.predict(X_train))
fpr_ts, tpr_ts, threshold_ts = roc_curve(Y_test, Pred_Prob_test)
#print("The accuracy score is:",roc_auc_score(fpr,tpr))
```

In [46]:

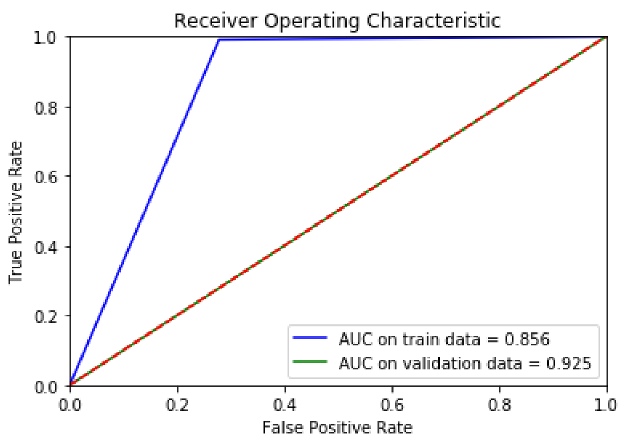
```python
AUC_Bow_tr_tf = metrics.auc(fpr_tr, tpr_tr)
AUC_Bow_ts_tf = metrics.auc(fpr_ts, tpr_ts)
print("The AUC value is:",AUC_Bow_tr_tf)
print("The AUC value is:",AUC_Bow_ts_tf)
```

```
The AUC value is: 0.8560828515244714
The AUC value is: 0.9250156981596971
```

In [47]:

```
#Plotting ROC Curve
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr_tr, tpr_tr, 'b', label = 'AUC on train data = %0.3f' % AUC_Bow_tr_tf)

plt.plot(fpr_ts, fpr_ts, 'g', label = 'AUC on validation data = %0.3f' % AUC_Bow_ts_tf)
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()
```

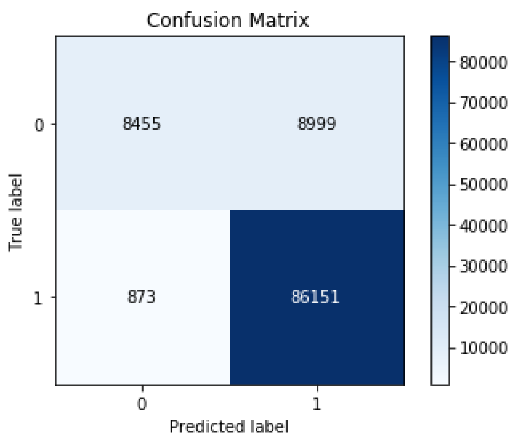


In [49]:

```
import scikitplot.metrics as skplt
skplt.plot_confusion_matrix(Y_test, Pred_test)

print("The precision score is:",round(metrics.precision_score(Y_test,Pred_test),4))
print("The recall value is:",round(metrics.recall_score(Y_test,Pred_test),4))
print("The F1 Score is:",round(metrics.f1_score(Y_test,Pred_test),4))
```

```
The precision score is: 0.9054
The recall value is: 0.99
The F1 Score is: 0.9458
```



In [73]:

```
from prettytable import PrettyTable
X = PrettyTable()

X.field_names = ["vectorizer","Optimal-Alpha Value","AUC on train","AUC on Test"]
X.add_row(["BoW",Best_Alpha_bow, AUC_Bow_tr,AUC_Bow_ts])
X.add_row(["TfIdf",Best_Alpha_tf,AUC_Bow_tr_tf, AUC_Bow_ts_tf])
print(X)
```

```
+------------+---------------------+--------------------+--------------------+
| vectorizer | Optimal-Alpha Value |    AUC on train    |    AUC on Test     |
+------------+---------------------+--------------------+--------------------+
|    BoW     |        1e-05        | 0.6935739366248278 | 0.8998794355369393 |
|   TfIdf    |        1e-05        | 0.8560828515244714 | 0.9250156981596971 |
+------------+---------------------+--------------------+--------------------+
```

[6] Conclusions

1. Both Bow and TfIdf are tuned for same alpha value
2. Compared to KNN Naive bayes is way better in terms of computation speed, takes less time to predict
3. The precision has come very well for tfidf vectorized(0.9054) data compared to BagOfWords(0.8652 - has more false positives). Hence shows the amount of truly predicted positives.
4. The recall is almost same for both (~0.0042)
5. (From AUC score)The TfIdf model works really good on both test data and train data whereas the Bow Model worked well on Test data but not much on train data.
6. F1 score is better for TfIdf model than Bow mode (~0.2)

---

[5] Assignment 4: Apply Naive Bayes

1. **Apply Multinomial NaiveBayes on these feature sets**

   - SET 1:Review text, preprocessed one converted into vectors using (BOW)
   - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)

2. **The hyper paramter tuning(find best Alpha)**

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

3. **Feature importance**

   - Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using values of `feature_log_prob_` parameter of MultinomialNB and print their corresponding feature names

4. **Feature engineering**

   - To increase the performance of your model, you can also experiment with with feature engineering like :
     - Taking length of reviews as another feature.
     - Considering some features from review summary as well.

5. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

6. **Conclusion**

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

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.