Amazon Fine Food Reviews Analysis

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

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:

Id

ProductId - unique identifier for the product

UserId - unqiue identifier for the user

ProfileName

HelpfulnessNumerator - number of users who found the review helpful

HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not

Score - rating between 1 and 5

Time - timestamp for the review

Summary - brief summary of the review

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).

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer #BOWs
from sklearn.feature_extraction.text import TfidfVectorizer #Tfidf
from sklearn import metrics
from sklearn.metrics import confusion_matrix

import nltk
from nltk.stem.porter import PorterStemmer

import re
```

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947

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

```
!cp "/content/drive/My Drive/database.sqlite" "database.sqlite"
```

1. Reading Data

```
con=sqlite3.connect("database.sqlite")
```

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
				Natalia		

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

```
actual_score=filtered_data['Score']
posnegative=actual_score.map(partition)
filtered_data['Score']=posnegative
filtered_data.head(5)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	

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.

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId',axis=0,ascending=True)
sorted_data.shape
```

```
(525814, 10)
```

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

```
(364173, 10)
```

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

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

```
(364171, 10)
```

```
final['Score'].value_counts()
```

```
1    307061
0     57110
Name: Score, dtype: int64
```

3. Preprocessing

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

1.Begin by removing the linksand html tags

2.Expand English language contractions and 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)

```
#print(link)
for i in range(0,1000):
  links=re.findall(r'http\S+',final['Text'].values[i])
  for link in links:
    print(link)
```

```
http://www.amazon.com/gp/product/B0002DGRSY">Pro-Treat
http://www.amazon.com/gp/product/B001905Z0Q">Charlee
http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<br
http://www.amazon.com/gp/product/B0088EDMMS">Hocus
http://www.amazon.com/gp/product/B001AGXEAG">Beetlejuice
http://www.amazon.com/gp/product/B001AGXEAG">here</a>.
http://www.amazon.com/gp/product/B001AGXEA6">here</a>.
http://www.amazon.com/gp/product/0790700506">Gremlins</a><br
http://www.amazon.com/gp/product/6301871952">Gremlins
http://www.amazon.com/gp/product/6303347657">Mask</a><br
http://www.amazon.com/gp/product/6304826141">Rocketman</a><br
http://www.amazon.com/gp/product/B001B504LI">The
http://www.amazon.com/gp/product/B001AGXEA6">Beetlejuice
http://www.amazon.com/gp/product/B00004RAMX">Victor
http://www.amazon.com/gp/product/B00004RAMY">Victor
http://www.amazon.com/gp/product/B00004RAMY">Victor
http://www.amazon.com/gp/product/B001VJ3FP6">SentryHOME
http://www.amazon.com/gp/product/B0051GCTAW">Fiproguard
http://www.amazon.com/gp/product/B000668Z96">Victor
http://www.amazon.com/gp/product/B000668Z96">Victor
http://www.amazon.com/gp/product/B000BQRQ8C">Rescue
http://www.amazon.com/gp/product/B00004RBDZ">Victor
http://www.amazon.com/gp/product/B00005344V">Traditional
http://www.amazon.com/gp/product/B00005C2M2">Astronaut
```

```
for i in range(0,364171):
  final['Text'].values[i] = re.sub(r"http\S+", "", final['Text'].values[i])
```

```
#Remove HTML tags
from bs4 import BeautifulSoup
for i in range(1,364171):
```

```
  soup = BeautifulSoup(final['Text'].values[i],'lxml')
  text = soup.get_text()
  final['Text'].values[i]=text


#Expanding English language contractions
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



for i in range(0,364171):
  #print(final['Text'].values[i])
  final['Text'].values[i]=decontracted(final['Text'].values[i])
  #print(final['Text'].values[i])


#remove words with numbers python
for i in range(1,364171):
  #print(final['Text'].values[i])
  final['Text'].values[i] = re.sub("\S*\d\S*", "", final['Text'].values[i]).strip()
  #print(final['Text'].values[i])


#remove spacial character:
for i in range(1,364171):
  final['Text'].values[i]= re.sub('[^A-Za-z0-9]+', ' ', final['Text'].values[i])


#Remove punctution
for i in range(1,364171):
  final['Text'].values[i]= re.sub(r'[!|#|$\%|&|*|?|,|.|\'|/|"|)|(]',r'', final['Text'].valu


# 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"])
```

```
#tqdm
#Instantly make your loops show a smart progress meter
# just wrap any iterable with tqdm(iterable), and you’re done!
from tqdm import tqdm
preprocessed_reviews = []
for i in tqdm(range(0,364171)):
  sentence=""
  #print(final['Text'].values[i])
  for word in final['Text'].values[i].split():
    #print(word)
    word =word.lower()
    if word not in stopwords:
      sentence+=" "+word
  #print(sentence)
  preprocessed_reviews.append(sentence.strip())
```

```
100%|██████████| 364171/364171 [00:15<00:00, 23191.87it/s]
```

```
preprocessed_reviews[364170]
```

```
'purchased send son away college delivered right dorm room fast shipping loved much
```

```
final['Cleaned_text']=preprocessed_reviews
final.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	He
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	

```
#Randomly sample Data 60k points
random_sample_data = final.sample(n=100000)
```

```
final_sorted_time=random_sample_data.sort_values('Time',ascending=True,axis=0)
```

Data Splitting

```
y_Train=final_sorted_time['Score'][0:70000]
y_train=final_sorted_time['Score'][0:49000]
y_cv=final_sorted_time['Score'][49000:70000]
y_test=final_sorted_time['Score'][70000:100000]
```

```
Train_data=final_sorted_time['Cleaned_text'][0:70000]
train_data=final_sorted_time['Cleaned_text'][0:49000]
cv_data=final_sorted_time['Cleaned_text'][49000:70000]
test_data=final_sorted_time['Cleaned_text'][70000:100000]
```

4. Bag of Words

```
from sklearn.naive_bayes import MultinomialNB
```

```
count_vector=CountVectorizer()
train_bows=count_vector.fit_transform(train_data)
cv_bows=count_vector.transform(cv_data)
```

```
from sklearn.metrics import roc_curve, auc
```

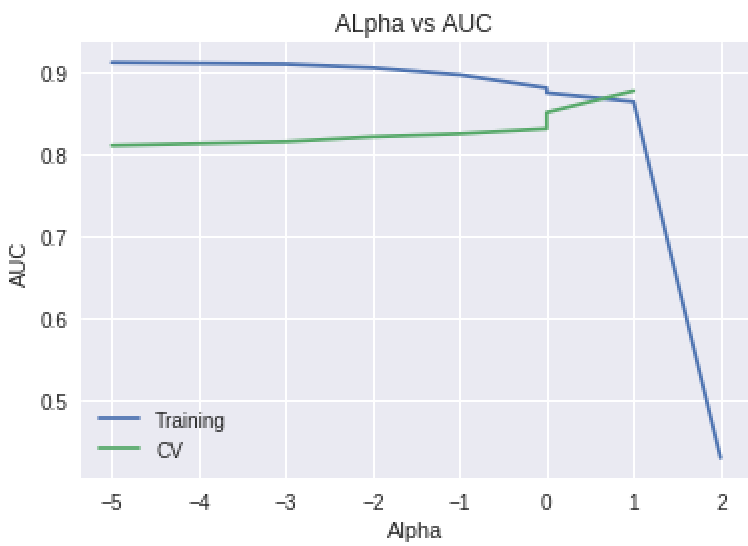
```
import math
```

```
#Simple CrossValidation
AUC_training=[]
AUC_cv=[]
ALPHA=[]
alpha=0.00001
while(alpha<1000000):
  ALPHA.append(math.ceil(math.log(alpha,10)))
  clf=MultinomialNB(alpha=alpha)
  clf.fit(train_bows , y_train)

  #Training Curve
  y_predict_training=clf.predict(train_bows)
  fpr, tpr, thresholds = roc_curve(y_predict_training, y_train)
  AUC_training.append(metrics.auc(fpr, tpr))
```

```
  #CV Curve
  y_predict_cv=clf.predict(cv_bows)
  fpr, tpr, thresholds = roc_curve(y_predict_cv, y_cv)
  AUC_cv.append(metrics.auc(fpr, tpr))
  alpha*=10


plt.plot(ALPHA[0:9],AUC_training[0:9],label='Training')
plt.plot(ALPHA[0:9],AUC_cv[0:9],label="CV")
plt.ylabel('AUC')
plt.xlabel('Alpha')
plt.title('ALpha vs AUC ')
plt.legend()
plt.show()
```



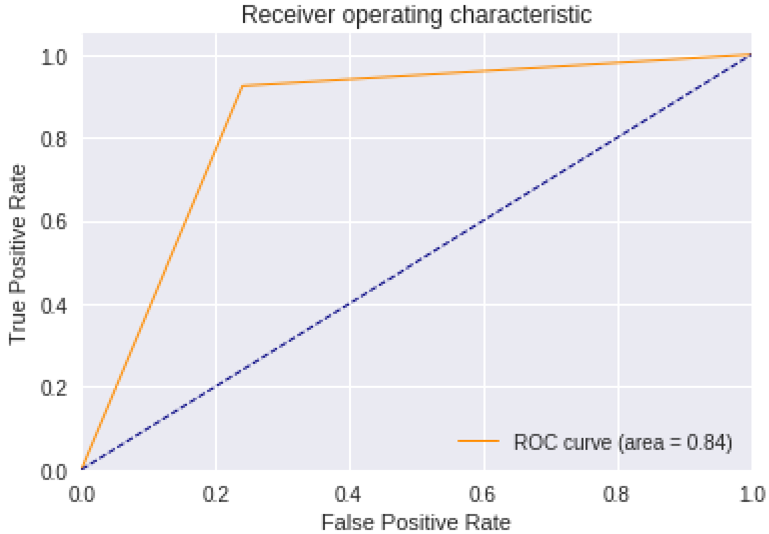
Observation : Optimal Alpha value is 1 having AUC=0.88

```
#For Optimal on test data
count_vectorizer=CountVectorizer()
Train_bows=count_vectorizer.fit_transform(Train_data)
test_bows=count_vectorizer.transform(test_data)



clf=MultinomialNB(alpha=1.0)
clf.fit(Train_bows,y_Train)
y_pred=clf.predict(test_bows)
 #Drawing ROC curve
fpr, tpr, thresholds = roc_curve(y_pred, y_test)
roc_auc = auc(fpr, tpr)
print(' AUC = ',metrics.auc(fpr, tpr))
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
print("\n\n")
print(metrics.classification_report(y_test,y_pred))
```

```
AUC =  0.8418896168504832
```




```
              precision    recall  f1-score   support

           0       0.76      0.63      0.69      5191
           1       0.92      0.96      0.94     24809

   micro avg       0.90      0.90      0.90     30000
   macro avg       0.84      0.79      0.81     30000
weighted avg       0.90      0.90      0.90     30000
```

Top Feature for negative

```
import operator
```

```
feature_names=count_vectorizer.get_feature_names()
a=getattr(clf, 'feature_log_prob_')
top =zip(a[0], feature_names)
top=list(top)
top.sort(key=lambda x: x[0])
l=len(top)
for i in range(1,20):
  print(top[l-i][1],end="\t")
```

```
not     like    product would   taste   one     good    no      coffee  flavor  tea
```

Top Feature for positive

```
feature_names=count_vectorizer.get_feature_names()
a=getattr(clf, 'feature_log_prob_')
top =zip(a[1], feature_names)
top=list(top)
top.sort(key=lambda x: x[0])
l=len(top)
for i in range(1,50):
  print(top[l-i][1],end="\t")
```

```
coffee  would  get  amazon  no  really  use  food  best  also  mu
```

TFIDF

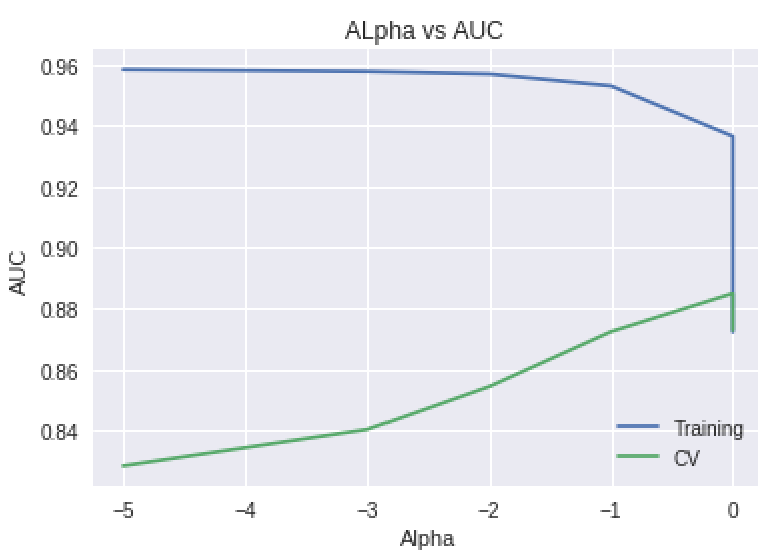
```
tfidf_vector=TfidfVectorizer()
train_tfidf=tfidf_vector.fit_transform(train_data)
cv_tfidf=tfidf_vector.transform(cv_data)


#Simple CrossValidation
AUC_training=[]
AUC_cv=[]
ALPHA=[]
alpha=0.00001
while(alpha<1000000):
  ALPHA.append(math.ceil(math.log(alpha,10)))
  clf=MultinomialNB(alpha=alpha)
  clf.fit(train_tfidf , y_train)

  #Training Curve
  y_predict_training=clf.predict(train_tfidf)
  fpr, tpr, thresholds = roc_curve(y_predict_training, y_train)
  AUC_training.append(metrics.auc(fpr, tpr))

  #CV Cuve
  y_predict_cv=clf.predict(cv_tfidf)
  fpr, tpr, thresholds = roc_curve(y_predict_cv, y_cv)
  AUC_cv.append(metrics.auc(fpr, tpr))
  alpha*=10


plt.plot(ALPHA[0:9],AUC_training[0:9],label='Training')
plt.plot(ALPHA[0:9],AUC_cv[0:9],label="CV")
plt.ylabel('AUC')
plt.xlabel('Alpha')
plt.title('ALpha vs AUC ')
plt.legend()
plt.show()
```

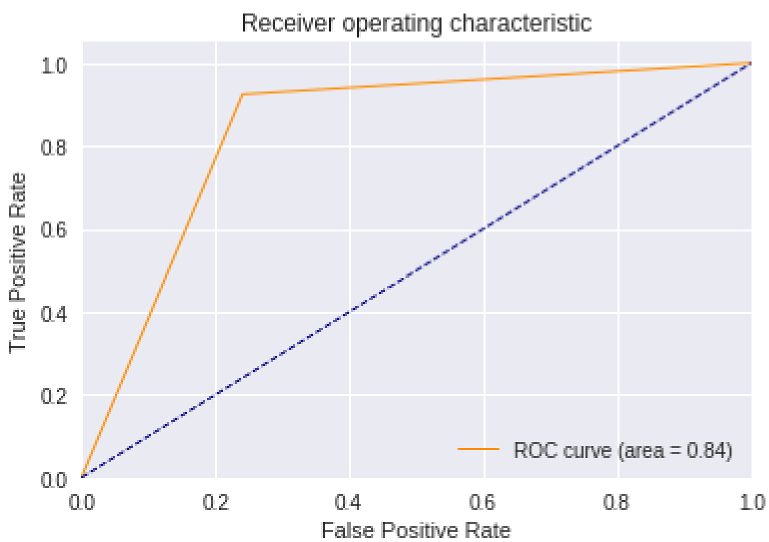


Observation : Optimal Alpha value is 1 having AUC=0.89

```
#For Optimal on test data
tfidf_vectorizer=CountVectorizer()
Train_tfidf=tfidf_vectorizer.fit_transform(Train_data)
test_tfidf=tfidf_vectorizer.transform(test_data)


clf=MultinomialNB(alpha=1.0)
clf.fit(Train_tfidf,y_Train)
y_pred=clf.predict(test_tfidf)
 #Drawing ROC curve
fpr, tpr, thresholds = roc_curve(y_pred, y_test)
roc_auc = auc(fpr, tpr)
print(' AUC = ',metrics.auc(fpr, tpr))
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
print("\n\n")
print(metrics.classification_report(y_test,y_pred))
```

```
AUC =  0.8418896168504832
```




```
              precision    recall  f1-score   support

           0       0.76      0.63      0.69      5191
           1       0.92      0.96      0.94     24809

   micro avg       0.90      0.90      0.90     30000
   macro avg       0.84      0.79      0.81     30000
weighted avg       0.90      0.90      0.90     30000
```

Top Feature for positive

```
feature_names=count_vectorizer.get_feature_names()
a=getattr(clf, 'feature_log_prob_')
top =zip(a[1], feature_names)
top=list(top)
top.sort(key=lambda x: x[0])
l=len(top)
for i in range(1,50):
  print(top[l-i][1],end="\t")
```

```
not	like	good	great	one	taste	tea	flavor	product love	coff
```

Top Feature for negative

```
feature_names=count_vectorizer.get_feature_names()
a=getattr(clf, 'feature_log_prob_')
top =zip(a[0], feature_names)
top=list(top)
top.sort(key=lambda x: x[0])
l=len(top)
for i in range(1,20):
  print(top[l-i][1],end="\t")
```

```
not	like	product would	taste	one	good	no	coffee	flavor	tea
```