Amazon Fine Food Reviews Analysis

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

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (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 the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

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 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 id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [56]:

```python
# for ploting inline
%matplotlib inline

# for ignore warnings
import warnings
warnings.filterwarnings("ignore")



import sqlite3  # to handle light database
import pandas as pd # for data processing
import numpy as np # for Numerical Operation
import nltk # Natural Language ToolKit
import string # for String handling
import matplotlib.pyplot as plt # for visualization
import seaborn as sns #  build on top of matplotlib

# for text feature extraction
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Metrics
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

# for stemming the word import PorterStemmer
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

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
```

[1]. Reading Data

In [57]:
```python
# using the 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 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""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50000""", con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
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 (50000, 10)
```

Out[57]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	This is a confection that has been around a fe...

In [58]:
```
# save data temporarily
filtered_data.to_csv('filtered_data.csv',index=False)
```

In [ ]:

In [59]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [60]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[60]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [61]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[61]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [62]:
```
display['COUNT(*)'].sum()
```

Out[62]:

```
393063
```

Exploratory Data Analysis

[2] 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 [63]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[63]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	T
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIO WAFER FIND TH EUROPE WAFERS
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIO WAFER FIND TH EUROPE WAFERS
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIO WAFER FIND TH EUROPE WAFERS
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIO WAFER FIND TH EUROPE WAFERS
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIO WAFER FIND TH EUROPE WAFERS

As can be seen above the same user has multiple reviews of the with the 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 [64]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

In [65]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[65]:
```
(46072, 10)
```

In [66]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[66]:
```
92.144
```

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 [67]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[67]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College	My son loves spaghetti so I didn't hesitate or...
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside	It was almost a 'love at first bite' - the per...

In [68]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [69]:
```
#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()
```

```
(46071, 10)
```

Out[69]:
```
1    38479
0     7592
Name: Score, dtype: int64
```

We Observed that data is highly Imbalanced

[3]. Text Preprocessing.

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 [70]:

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

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

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

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

My dogs loves this chicken but its a product from China, so we wont be buying it anymore. Its very hard to find any chicken products made in the USA but they are out there, but this one isnt. Its too bad too because its a good product but I wont take any chances till they know what is going on with the china imports.

==

this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is better, but a heck of a lot more work. this is great to have on hand for last minute dessert needs where you really want to impress wih your creativity in cooking! recommended.

==

Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are high priced and high in calories, this one is a great bargain and tastes great, I highly recommend for the lady gym rats, probably not "macho" enough for guys since it is soy based...

==

For those of you wanting a high-quality, yet affordable green tea, you should definitely give this one a try. Let me first start by saying that everyone is looking for something different for their ideal tea, and I will attempt to briefly highlight what makes this tea attractive to a wide range of tea drinkers (whether you are a beginner or long-time tea enthusiast). I have gone through over 12 boxes of this tea myself, and highly recommend it for the following reasons:

-Quality: First, this tea offers a smooth quality without any harsh or bitter after tones, which often turns people off from many green teas. I've found my ideal brewing time to be between 3-5 minutes, giving you a light but flavorful cup of tea. However, if you get distracted or forget about your tea and leave it brewing for 20+ minutes like I sometimes do, the quality of this tea is such that you still get a smooth but deeper flavor without the bad after taste. The leaves themselves are whole leaves (not powdered stems, branches, etc commonly found in other brands), and the high-quality nylon bags also include chunks of tropical fruit and other discernible ingredients. This isn't your standard cheap paper bag with a mix of unknown ingredients that have been ground down to a fine powder, leaving you to wonder what it is you are actually drinking.

-Taste: This tea offers notes of real pineapple and other hints of tropical fruits, yet isn't sweet or artificially flavored. You have the foundation of a high-quality young hyson green tea for those true "tea flavor" lovers, yet the subtle hints of fruit make this a truly unique tea that I believe most will enjoy. If you want it sweet, you can add sugar, splenda, etc but this really is not necessary as this tea offers an inherent warmth of flavor through it's ingredients.

-Price: This tea offers an excellent product at an exceptional price (especially when purchased at the prices Amazon offers). Compared to other brands which I believe to be of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), Revolution offers a superior product at an outstanding price. I have been purchasing this through Amazon for less per box than I would be paying at my local grocery store for Lipton, etc.

Overall, this is a wonderful tea that is comparable, and even better than, other teas that are priced much higher. It offers a well-balanced cup of green tea that I believe many will enjoy. In terms of taste, quality, and price, I would argue you won't find a better combination that that offered by Revolution's Tropical Green Tea.

==

In [71]:
```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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very hard to find any c
hicken products made in the USA but they are out there, but this one isnt.  Its too bad too because its a good product
but I wont take any chances till they know what is going on with the china imports.
```

In [72]:

```
# 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)
```

My dogs loves this chicken but its a product from China, so we wont be buying it anymore. Its very hard to find any chicken products made in the USA but they are out there, but this one isnt. Its too bad too because its a good product but I wont take any chances till they know what is going on with the china imports.

==

this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is better, but a heck of a lot more work. this is great to have on hand for last minute dessert needs where you really want to impress wih your creativity in cooking! recommended.

==

Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are high priced and high in calories, this one is a great bargain and tastes great, I highly recommend for the lady gym rats, probably not "macho" enough for guys since it is soy based...

==

For those of you wanting a high-quality, yet affordable green tea, you should definitely give this one a try. Let me first start by saying that everyone is looking for something different for their ideal tea, and I will attempt to briefly highlight what makes this tea attractive to a wide range of tea drinkers (whether you are a beginner or long-time tea enthusiast). I have gone through over 12 boxes of this tea myself, and highly recommend it for the following reasons:-Quality: First, this tea offers a smooth quality without any harsh or bitter after tones, which often turns people off from many green teas. I've found my ideal brewing time to be between 3-5 minutes, giving you a light but flavorful cup of tea. However, if you get distracted or forget about your tea and leave it brewing for 20+ minutes like I sometimes do, the quality of this tea is such that you still get a smooth but deeper flavor without the bad after taste. The leaves themselves are whole leaves (not powdered stems, branches, etc commonly found in other brands), and the high-quality nylon bags also include chunks of tropical fruit and other discernible ingredients. This isn't your standard cheap paper bag with a mix of unknown ingredients that have been ground down to a fine powder, leaving you to wonder what it is you are actually drinking.-Taste: This tea offers notes of real pineapple and other hints of tropical fruits, yet isn't sweet or artificially flavored. You have the foundation of a high-quality young hyson green tea for those true "tea flavor" lovers, yet the subtle hints of fruit make this a truly unique tea that I believe most will enjoy. If you want it sweet, you can add sugar, splenda, etc but this really is not necessary as this tea offers an inherent warmth of flavor through it's ingredients.-Price: This tea offers an excellent product at an exceptional price (especially when purchased at the prices Amazon offers). Compared to other brands which I believe to be of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), Revolution offers a superior product at an outstanding price. I have been purchasing this through Amazon for less per box than I would be paying at my local grocery store for Lipton, etc.Overall, this is a wonderful tea that is comparable, and even better than, other teas that are priced much higher. It offers a well-balanced cup of green tea that I believe many will enjoy. In terms of taste, quality, and price, I would argue you won't find a better combination that that offered by Revolution's Tropical Green Tea.

In [73]:
```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 [74]:
```python
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are high priced and high in calories, this one is a great bargain and tastes great, I highly recommend for the lady gym rats, probably not "macho" enough for guys since it is soy based...
==================================================
```

In [75]:
```python
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very hard to find any c
hicken products made in the USA but they are out there, but this one isnt.  Its too bad too because its a good product
but I wont take any chances till they know what is going on with the china imports.
```

In [76]:
```python
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Great flavor low in calories high in nutrients high in protein Usually protein powders are high priced and high in cal
ories this one is a great bargain and tastes great I highly recommend for the lady gym rats probably not macho enough
for guys since it is soy based
```

In [77]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'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', 'each', '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', "doesn'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', "were
n't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [78]:

```python
# 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%|██████████████████████████████| 46071/46071 [01:59<00:00, 384.20it/
s]
```

In [79]:
```
preprocessed_reviews[1500]
```
Out[79]:
```
'great flavor low calories high nutrients high protein usually protein powders high priced high calories one great bar
gain tastes great highly recommend lady gym rats probably not macho enough guys since soy based'
```

[3.2] Preprocess Summary

In [80]:
```
## Similarly you can do preprocessing for review summary also.
preprocessed_summary = []
# tqdm is for printing the status bar
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_summary.append(sentance.strip())
```
```
100%|████████████████████████████████████████████████████| 46071/46071 [00:34<00:00, 1348.94it/
s]
```

In [81]:
```
preprocessed_summary[1500]
```
Out[81]:
```
'gym rat bargain'
```

In [82]:
```
final.iloc[1500]
```
Out[82]:
```
Id                                                          27047
ProductId                                              B00024D628
UserId                                             A1TUTMN6KI1PW7
ProfileName                                    Amy M. Nissen "amn"
HelpfulnessNumerator                                            1
HelpfulnessDenominator                                          1
Score                                                           1
Time                                                   1245456000
Summary                                           gym rat bargain!
Text                 Great flavor, low in calories, high in nutrien...
Name: 24760, dtype: object
```

In [ ]:

[4] Featurization

[4.1] BAG OF WORDS

In [83]:
```
#BoW
count_vect = CountVectorizer(max_features=5000) #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  ['ability', 'able', 'absolute', 'absolutely', 'absorb', 'absorbed', 'acai', 'accept', 'acceptabl
e', 'accepted']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 5000)
the number of unique words  5000
```

[4.2] Bi-Grams and n-Grams.

```
In [84]: #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_ext
raction.text.CountVectorizer.html
# you can choose these numebrs min_df=10, max_features=10000, 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_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

```
In [85]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
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) ['ability', 'able', 'able buy', 'able find', 'able get', 'absolute',
'absolute favorite', 'absolutely', 'absolutely best', 'absolutely delicious']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (46071, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.4] Word2Vec

In [86]:
```
# 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 [87]:

```
# 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.bin', binary=True)
        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, to train your own w2v ")
```

```
[('awesome', 0.8466016054153442), ('terrific', 0.8083761930465698), ('fantastic', 0.7963908910751343), ('good', 0.7941110134124756), ('excellent', 0.7757711410522461), ('amazing', 0.759486973285675), ('wonderful', 0.7581246495246887), ('perfect', 0.749467134475708), ('decent', 0.6931086182594299), ('nice', 0.6757321357727051)]
==================================================
[('nastiest', 0.7269057035446167), ('best', 0.712761402130127), ('greatest', 0.7010270953178406), ('tastiest', 0.6726614832878113), ('experienced', 0.6657267212867737), ('awful', 0.6627132892608643), ('hottest', 0.6474201679229736), ('closest', 0.6374478340148926), ('ive', 0.606102466583252), ('eaten', 0.5973571538925171)]
```

In [88]:

```
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  12798
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buying', 'anymore', 'hard', 'find', 'product
s', 'made', 'usa', 'one', 'isnt', 'bad', 'good', 'take', 'chances', 'till', 'know', 'going', 'imports', 'love', 'saw',
'pet', 'store', 'tag', 'attached', 'regarding', 'satisfied', 'safe', 'available', 'victor', 'traps', 'unreal', 'cours
e', 'total', 'fly', 'pretty', 'stinky', 'right', 'nearby', 'used', 'bait', 'seasons', 'ca', 'not', 'beat', 'great']
```

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

[4.4.1.1] Avg W2v

In [34]:

```
# 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 to change this to 300 if you use go
ogle's w2v
    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%|██████████████████████████████████████████████████| 46071/46071 [03:23<00:00, 226.27it/
s]

46071
50
```

[4.4.1.2] TFIDF weighted W2v

In [39]:
```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(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 [42]:
```python
#dictionary  # uncomment this to see the dictionary
```

In [43]:
```python
# 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 list
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%|██████████████████████████████| 46071/46071 [1:16:43<00:00, 10.01it/
s]
```

In [ ]:

Creating ML Model

importing necessary libraries

source code: [https://www.machinelearningmastery.com (https://www.machinelearningmastery.com)](https://www.machinelearningmastery.com)

In [89]:
```
import numpy as np                    # for numerical computation in python
import matplotlib.pyplot as plt       # for Data Visualization
import seaborn as sns                 # Built on top of matplotlib for Data Viz
import pandas as pd                   # for Data Munging,Manipulation etc.
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

In [90]:
```
final.head()
```

Out[90]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Te
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	1	0	1192060800	made in china	My dog loves th chicke but its produ f
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	0	1	1195948800	Dog Lover Delites	Our dog just lov them saw the in a pet
2546	2774	B00002NCJC	A196AJHU9EASJN	Alex Chaffee	0	0	1	1282953600	thirty bucks?	Why this $[. when th sam product av
2547	2775	B00002NCJC	A13RRPGE79XFFH	reader48	0	0	1	1281052800	Flies Begone	We hav used th Victor f bait for seasons
1145	1244	B00002Z754	A3B8RCEI0FXFI6	B G Chase	10	10	1	962236800	WOW Make your own 'slickers' !	I ju receive n shipme and cou hardly w

In [91]:
```
final_counts
```

Out[91]:
```
<46071x5000 sparse matrix of type '<class 'numpy.int64'>'
	with 1389857 stored elements in Compressed Sparse Row format>
```

BOW ML model

In [92]:
```
# using BOW to create the model

X = final_counts[0:5000].toarray()
# selecting 5000 rows because of having memory error
Y = final['Score'].iloc[0:5000].values
```

In [ ]:

In [93]:
```
# shape of the dataset
print(X.shape)
print(Y.shape)
```

```
(5000, 5000)
(5000,)
```

In [94]:
```
# create a validation dataset
validation_size = 0.20
seed=7
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=validation_size,random_state=seed)
```

In [95]:
```
X[0]
```

Out[95]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [96]:
```
Y[0]
```

Out[96]:
```
0
```

In [97]:
```
X.size
```

Out[97]:
```
25000000
```

In [98]:
```
25000000/(1024*1024)
```

Out[98]:
```
23.84185791015625
```

Spot-Check Algorithms

In [99]:
```python
# Spot-Check Algorithms

models = []
models.append(('LR',LogisticRegression()))
models.append(('LDA',LinearDiscriminantAnalysis()))
models.append(('KNN',KNeighborsClassifier()))
models.append(('CART',DecisionTreeClassifier()))
models.append(('NB',GaussianNB()))
models.append(('SVM',SVC()))

# evaluate each model in turn
results = []
names = []

for name, model in models:
    kfold = KFold(n_splits = 10, random_state=seed)
    cv_results = cross_val_score(model,X_train,y_train,cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
    print(name, cv_results.mean()*100.0, "(",cv_results.std()*100.0,")")
```

```
LR 89.225 ( 1.1036870027322054 )
LDA 63.324999999999996 ( 1.8474644786842322 )
KNN 84.72500000000001 ( 1.2064928512013675 )
CART 82.45000000000002 ( 1.7705931209625783 )
NB 69.525 ( 2.39648179630057 )
SVM 85.32499999999999 ( 1.7466038474708574 )
```

In [100]:
```python
model = LogisticRegression() # creating the model Logistic Regression
```

In [101]:
```python
model.fit(X_train,y_train) # fitting the model
```

Out[101]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [102]:
```python
predictions = model.predict(X_test)
```

In [103]:
```python
accuracy_score(predictions,y_test)
```

Out[103]:
```
0.893
```

In [104]:
```
confusion_matrix(predictions,y_test)
```

Out[104]:
```
array([[ 58,  36],
       [ 71, 835]], dtype=int64)
```

In [105]:
```
print(classification_report(predictions,y_test))
```

```
              precision    recall  f1-score   support

           0       0.45      0.62      0.52        94
           1       0.96      0.92      0.94       906

    accuracy                           0.89      1000
   macro avg       0.70      0.77      0.73      1000
weighted avg       0.91      0.89      0.90      1000
```

In [106]:
```
# help(LogisticRegression())
```

cross-validation, Hyperparameter tuning

In [107]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [108]:
```
for i in [0.0001,0.001,0.01,0.1,1.0,5,10,100,1000,10000]:
    model = LogisticRegression(C=i,class_weight='balanced')
    model.fit(X_train,y_train)
    print("i =",i," accuracy: ",accuracy_score(model.predict(X_test),y_test))
```

```
i = 0.0001  accuracy:  0.557
i = 0.001  accuracy:  0.748
i = 0.01  accuracy:  0.835
i = 0.1  accuracy:  0.87
i = 1.0  accuracy:  0.873
i = 5  accuracy:  0.877
i = 10  accuracy:  0.875
i = 100  accuracy:  0.869
i = 1000  accuracy:  0.865
i = 10000  accuracy:  0.86
```

In [109]:
```
for i in range(1,11):
    model = LogisticRegression(C=i,class_weight='balanced')
    model.fit(X_train,y_train)
    print("i =",i," accuracy: ",accuracy_score(model.predict(X_test),y_test))
```

```
i = 1  accuracy:  0.873
i = 2  accuracy:  0.877
i = 3  accuracy:  0.879
i = 4  accuracy:  0.879
i = 5  accuracy:  0.877
i = 6  accuracy:  0.877
i = 7  accuracy:  0.876
i = 8  accuracy:  0.877
i = 9  accuracy:  0.875
i = 10  accuracy:  0.875
```

Our final model is

In [111]:
```
final_model = LogisticRegression(C=4.0,class_weight='balanced')
```

In [112]:
```
final_model.fit(X_train,y_train)
```

Out[112]:
```
LogisticRegression(C=4.0, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [113]:
```
predictions = final_model.predict(X_test)
```

In [114]:
```
print("accuracy_score:\n")
print(accuracy_score(predictions,y_test))
```

```
accuracy_score:

0.879
```

In [115]:
```
print("confusion matrix\n")
print(confusion_matrix(predictions,y_test))
```

```
confusion matrix

[[ 73  65]
 [ 56 806]]
```

In [116]:
```
print(classification_report(predictions,y_test))
```

```
              precision    recall  f1-score   support

           0       0.57      0.53      0.55       138
           1       0.93      0.94      0.93       862

    accuracy                           0.88      1000
   macro avg       0.75      0.73      0.74      1000
weighted avg       0.88      0.88      0.88      1000
```

Bi-gram , n-gram ML model

In [117]:
```
X = final_bigram_counts[0:5000].toarray()
# selecting 5000 rows because of having memory error
Y = final['Score'].iloc[0:5000].values
```

In [118]:
```
# create a validation dataset
validation_size = 0.20
seed=7
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=validation_size,random_state=seed)
```

In [119]:
```
# Spot-Check Algorithms

models = []
models.append(('LR',LogisticRegression()))
models.append(('LDA',LinearDiscriminantAnalysis()))
models.append(('KNN',KNeighborsClassifier()))
models.append(('CART',DecisionTreeClassifier()))
models.append(('NB',GaussianNB()))
models.append(('SVM',SVC()))

# evaluate each model in turn
results = []
names = []

for name, model in models:
    kfold = KFold(n_splits = 10, random_state=seed)
    cv_results = cross_val_score(model,X_train,y_train,cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
    print(name, cv_results.mean()*100.0, "(",cv_results.std()*100.0,")")
```

```
LR 89.2 ( 1.0770329614269014 )
LDA 65.60000000000001 ( 2.660357118884606 )
KNN 84.87500000000001 ( 1.179247641507077 )
CART 82.325 ( 1.4188463623662713 )
NB 78.47500000000001 ( 1.039531144314591 )
SVM 85.32499999999999 ( 1.7466038474708574 )
```

still LogisticRegression is performing well on n-grams data

In [120]:
```
n_gram_LR_model = LogisticRegression()
```

In [121]:
```
n_gram_LR_model.fit(X_train,y_train)
```

Out[121]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [122]:
```
predictions_ngram = n_gram_LR_model.predict(X_test)
```

In [123]:
```
accuracy_score(predictions_ngram,y_test)
```
Out[123]:
```
0.896
```

In [125]:
```
print(classification_report(predictions_ngram,y_test))
```
```
              precision    recall  f1-score   support

           0       0.45      0.64      0.53        91
           1       0.96      0.92      0.94       909

    accuracy                           0.90      1000
   macro avg       0.71      0.78      0.73      1000
weighted avg       0.92      0.90      0.90      1000
```

In [126]:
```
print(confusion_matrix(predictions_ngram,y_test))
```
```
[[ 58  33]
 [ 71 838]]
```

TF-IDF ML model

In [129]:
```
type(final_tf_idf)
```
Out[129]:
```
scipy.sparse.csr.csr_matrix
```

In [130]:
```
X = final_tf_idf[0:5000].toarray()
# selecting 5000 rows because of having memory error
Y = final['Score'].iloc[0:5000].values
```

In [132]:
```
X_train,X_test,y_train,y_test = train_test_split(X,Y, random_state=seed,test_size=validation_size)
```

In [133]:
```
# Spot-Check Algorithms

models = []
models.append(('LR',LogisticRegression()))
models.append(('LDA',LinearDiscriminantAnalysis()))
# models.append(('KNN',KNeighborsClassifier()))
models.append(('CART',DecisionTreeClassifier()))
models.append(('NB',GaussianNB()))
# models.append(('SVM',SVC()))

# evaluate each model in turn
results = []
names = []

for name, model in models:
    kfold = KFold(n_splits = 10, random_state=seed)
    cv_results = cross_val_score(model,X_train,y_train,cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
    print(name, cv_results.mean()*100.0, "(",cv_results.std()*100.0,")")
```

```
LR 86.99999999999999 ( 1.5929532322073983 )
LDA 71.825 ( 2.395438373241943 )
CART 83.35000000000001 ( 2.233830790368867 )
NB 78.82499999999999 ( 1.1183134623172504 )
```

In [134]:
```
tfidf_LR_model = LogisticRegression()
```

In [135]:
```
tfidf_LR_model.fit(X_train,y_train)
```

Out[135]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [136]:
```
tfidf_predictions = tfidf_LR_model.predict(X_test)
```

In [139]:
```
def metrics(y_predicted, y_actual):

    print("="*50)
    print("accuracy score")
    print(accuracy_score(y_predicted,y_actual))

    print("="*50)
    print("confusion matrix")
    print(confusion_matrix(y_predicted,y_actual))

    print("="*50)
    print("classification report")
    print(classification_report(y_predicted,y_actual))
```

In [140]:
```
metrics(tfidf_predictions, y_test)
```

```
==================================================
accuracy score
0.89
==================================================
confusion matrix
[[ 20   1]
 [109 870]]
==================================================
classification report
              precision    recall  f1-score   support

           0       0.16      0.95      0.27        21
           1       1.00      0.89      0.94       979

    accuracy                           0.89      1000
   macro avg       0.58      0.92      0.60      1000
weighted avg       0.98      0.89      0.93      1000
```

In [141]:
```
tfidf_DT_model = DecisionTreeClassifier()
```

In [142]:
```
tfidf_DT_model.fit(X_train,y_train)
```

Out[142]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [143]:
```
tfidf_dt_predictions = tfidf_DT_model.predict(X_test)
```

In [144]:
```
metrics(tfidf_dt_predictions,y_test)
```

```
==================================================
accuracy score
0.834
==================================================
confusion matrix
[[ 51  88]
 [ 78 783]]
==================================================
classification report
              precision    recall  f1-score   support

           0       0.40      0.37      0.38       139
           1       0.90      0.91      0.90       861

    accuracy                           0.83      1000
   macro avg       0.65      0.64      0.64      1000
weighted avg       0.83      0.83      0.83      1000
```

In [ ]: