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 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.

In [ ]:
```
# [1]. Reading Data
```

In [ ]:
```
## [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.
<br>

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 recommendati
```

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
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
import scipy as sp
from tqdm import tqdm
import os
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import normalize
from sklearn import datasets, neighbors
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
```

```
C:\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
C:\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect('C:\Python\Amazon review\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""", 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(1)
```

```
Number of data points in our data (525814, 10)
```

Out[2]:

	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...

In [3]:
```
#  [2] Exploratory Data Analysis
```

In [4]:
```
ication


e 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:
```

```
  File "<ipython-input-4-946a5d04d6db>", line 3
    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:
                ^
SyntaxError: invalid syntax
```

In [5]:
```
#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 [6]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[6]:
```
(364173, 10)
```

In [7]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

[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 [8]:

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

# 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', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])

# 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())


## Similartly you can do preprocessing for review summary also.
preprocessed_sum = []
# 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_sum.append(sentance.strip())


# adding the new preprocessed data as new columns to our final dataframe.

ps = pd.Series(preprocessed_sum)
final['Summary_new']=ps.values

pr = pd.Series(preprocessed_reviews)
final['Text_new']=pr.values

print('Shape of final',final.shape)
print(final['Score'].value_counts())
final.head(1)


# Saving the final data frame, prerocessed reviews and summary,
# so that we can resume directly from here without doing preprocessing again
# https://www.datacamp.com/community/tutorials/reading-writing-files-python

with open("final.txt", "wb") as file:
    pickle.dump(final, file)

with open("preprocessed_reviews.txt", "wb") as file:
    pickle.dump(preprocessed_reviews, file)

with open("preprocessed_summary.txt", "wb") as file:
    pickle.dump(preprocessed_sum, file)


sorted_data = final.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')

with open("final_sorted.txt", "wb") as file:
    pickle.dump(sorted_data, file)
```

```
should probably open this file and pass the filehandle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
 92%|████████████████████████████▍   | 335126/364171 [01:58<00:10, 2716.85it/s]C:\Anaconda3\lib\site-packages\bs4\__init__.py:219: UserWarning: "b'...'" looks like a filename, not markup. You
should probably open this file and pass the filehandle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
 97%|█████████████████████████████▊  | 354901/364171 [02:05<00:03, 2843.48it/s]C:\Anaconda3\lib\site-packages\bs4\__init__.py:219: UserWarning: "b'.'" looks like a filename, not markup. You sh
ould probably open this file and pass the filehandle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
 99%|██████████████████████████████▍ | 358963/364171 [02:07<00:01, 2634.33it/s]C:\Anaconda3\lib\site-packages\bs4\__init__.py:219: UserWarning: "b'...'" looks like a filename, not markup. You s
hould probably open this file and pass the filehandle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
 99%|██████████████████████████████▍ | 359231/364171 [02:07<00:02, 2398.42it/s]C:\Anaconda3\lib\site-packages\bs4\__init__.py:219: UserWarning: "b'...'" looks like a filename, not markup. You s
hould probably open this file and pass the filehandle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
100%|██████████████████████████████| 364171/364171 [02:09<00:00, 2812.24it/s]
```

```
Shape of final (364171, 12)
1    307061
0     57110
Name: Score, dtype: int64
```

In [9]:
```
my_final = sorted_data[:100000]
my_final['Score'].value_counts()
```

Out[9]:
```
1    87729
0    12271
Name: Score, dtype: int64
```

In [10]:
```
from sklearn.model_selection import train_test_split
x = my_final['Text_new'].values
y = my_final['Score']

# split the data set into train and test
X_1, X_test, y_1, Y_test = cross_validation.train_test_split(x, y, test_size=0.3, random_state=0)

# split the train data set into cross validation train and cross validation test
X_train, X_cv, Y_train, Y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3,random_state=0)
```

In [11]:
```
#https://stackoverflow.com/questions/10741346/numpy-most-efficient-frequency-counts-for-unique-values-in-an-array
# Printing the frequency of positive and negative values in Train, CV and Test data set
unique, counts = np.unique(Y_train, return_counts=True)

np.asarray((unique, counts)).T
```

Out[11]:
```
array([[    0,  6013],
       [    1, 42987]], dtype=int64)
```

In [12]:
```
unique, counts = np.unique(Y_test, return_counts=True)

np.asarray((unique, counts)).T
```

Out[12]:
```
array([[    0,  3665],
       [    1, 26335]], dtype=int64)
```

In [13]:
```
unique, counts = np.unique(Y_cv, return_counts=True)

np.asarray((unique, counts)).T
```

Out[13]:
```
array([[    0,  2593],
       [    1, 18407]], dtype=int64)
```

Bag of Words

In [14]:
```
# Please write all the code with proper documentation
# Bag of Words
count_vect = CountVectorizer() #in scikit-learn
train_bow = count_vect.fit_transform(X_train)
test_bow = count_vect.transform(X_test)
cv_bow = count_vect.transform(X_cv)

```

In [15]:
```
from scipy.sparse import save_npz
save_npz('train_bow.npz', train_bow)
```

In [16]:
```
save_npz('test_bow.npz', test_bow)
save_npz('cv_bow.npz', cv_bow)
```

Bag of Words with max_feature

In [26]:
```
# Please write all the code with proper documentation
# Bag of Words
count_vect = CountVectorizer(min_df=50, max_features=2000) #in scikit-learn
train_bowl = count_vect.fit_transform(X_train)
test_bowl = count_vect.transform(X_test)
cv_bowl = count_vect.transform(X_cv)

```

In [27]:
```
from scipy.sparse import save_npz
save_npz('train_bow_lim.npz', train_bowl)
```

In [28]:
```
save_npz('test_bow_lim.npz', test_bowl)
save_npz('cv_bow_lim.npz', cv_bowl)
```

TFIDF

In [17]:
```
# Please write all the code with proper documentation
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)

final_tf_idf = tf_idf_vect.fit_transform(X_train)

train_tfidf = tf_idf_vect.fit_transform(X_train)
test_tfidf = tf_idf_vect.transform(X_test)
cv_tfidf = tf_idf_vect.transform(X_cv)
```

In [18]:
```
save_npz('train_tfidf.npz', train_tfidf)
save_npz('test_tfidf.npz', test_tfidf)
save_npz('cv_tfidf.npz', cv_tfidf)
```

TFIDF with max feature

In [29]:
```
# Please write all the code with proper documentation
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=50, max_features=2000)

final_tf_idf = tf_idf_vect.fit_transform(X_train)

train_tfidfl = tf_idf_vect.fit_transform(X_train)
test_tfidfl = tf_idf_vect.transform(X_test)
cv_tfidfl = tf_idf_vect.transform(X_cv)
```

In [30]:
```
save_npz('train_tfidf_lim.npz', train_tfidfl)
save_npz('test_tfidf_lim.npz', test_tfidfl)
save_npz('cv_tfidf_lim.npz', cv_tfidfl)
```

W2V

In [20]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in X_train:
    list_of_sentance.append(sentance.split())

 # min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))

def avgwtv(X_test):
    '''
    returns average woed2vec
    '''
    i=0
    list_of_sentance=[]
    for sentance in X_test:
        list_of_sentance.append(sentance.split())
    test_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 google'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
        test_vectors.append(sent_vec)
    return test_vectors

train_avgw2v = avgwtv(X_train)
cv_avgw2v = avgwtv(X_cv)
test_avgw2v = avgwtv(X_test)
```

```
number of words that occured minimum 5 times  13481

100%|██████████████████████████████| 49000/49000 [02:02<00:00, 398.75it/s]
100%|██████████████████████████████| 21000/21000 [00:55<00:00, 376.72it/s]
100%|██████████████████████████████| 30000/30000 [01:19<00:00, 376.48it/s]
```

In [22]:
```
with open("train_avgw2v.txt", "wb") as file:
    pickle.dump(train_avgw2v, file)
with open("cv_avgw2v.txt", "wb") as file:
    pickle.dump(cv_avgw2v, file)
with open("test_avgw2v.txt", "wb") as file:
    pickle.dump(test_avgw2v, file)
```

TFIDF W2V

In [21]:
```
# Please write all the code with proper documentation
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(X_train)
# 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_)))

# 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

#standardized_weight_w2v = StandardScaler().fit_transform(tfidf_sent_vectors)
#print(standardized_weight_w2v.shape)

def tfidfw2v(test):
    '''
    Returns tfidf word2vec
    '''
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
    i=0
    list_of_sentance=[]
    for sentance in test:
        list_of_sentance.append(sentance.split())

    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 = 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)

    return tfidf_sent_vectors

train_tfw2v = tfidfw2v(X_train)
cv_tfw2v = tfidfw2v(X_cv)
test_tfw2v = tfidfw2v(X_test)
```

```
100%|██████████████████████████████| 49000/49000 [30:48<00:00, 26.51it/s]
100%|██████████████████████████████| 21000/21000 [14:41<00:00, 15.27it/s]
100%|██████████████████████████████| 30000/30000 [18:19<00:00, 27.29it/s]
```

In [23]:
```
with open("train_tfw2v.txt", "wb") as file:
    pickle.dump(train_tfw2v, file)
with open("cv_tfw2v.txt", "wb") as file:
    pickle.dump(cv_tfw2v, file)
with open("test_tfw2v.txt", "wb") as file:
    pickle.dump(test_tfw2v, file)
```

In [24]:
```
with open("X_test.txt", "wb") as file:
    pickle.dump(X_test, file)
with open("X_train.txt", "wb") as file:
    pickle.dump(X_train, file)
with open("X_cv.txt", "wb") as file:
    pickle.dump(X_cv, file)
```

In [25]:
```
with open("Y_test.txt", "wb") as file:
    pickle.dump(Y_test, file)
with open("Y_train.txt", "wb") as file:
    pickle.dump(Y_train, file)
with open("Y_cv.txt", "wb") as file:
    pickle.dump(Y_cv, file)
```

In [ ]:
```

```