AMZN_Food_Reviews_Data_Preparation

April 14, 2019

1 [7] 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:

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.

1.1 [7.1] 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 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 [1]:
```
import pandas as pd
import numpy as np

# import database related packages
import sqlite3

# import text processing related packages
import nltk
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

# W2V Related Packages
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# for saving model
import pickle
import os

# regular expression for handling strings
import scipy.sparse
import re
import string

# Visualization related packages
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

In [2]:
```
# using the SQLite Table to read data.
con = sqlite3.connect('/media/amd_3/20DAD539DAD50BC2/DSET_REPO/DataSets/CS01-AMZN_FOOD_R

#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
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
filtered_data['Score'] = filtered_data['Score'].apply(lambda x : 1 if x > 3  else 0)
filtered_data.rename(columns={'Score' : 'Label'}, inplace=True)

print(filtered_data.shape) #looking at the number of attributes and size of the data
filtered_data.head()
```

```
(525814, 10)
```

```
Out[2]:    Id   ProductId          UserId                      ProfileName  \
        0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
        1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
        2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
        3   4  B000UA0QIQ  A395BORC6FGVXV                             Karl
        4   5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"

           HelpfulnessNumerator  HelpfulnessDenominator  Label        Time  \
        0                     1                       1      1  1303862400
        1                     0                       0      0  1346976000
        2                     1                       1      1  1219017600
        3                     3                       3      0  1307923200
        4                     0                       0      1  1350777600

                         Summary                                               Text
        0  Good Quality Dog Food  I have bought several of the Vitality canned d...
        1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
        2  "Delight" says it all  This is a confection that has been around a fe...
        3         Cough Medicine  If you are looking for the secret ingredient i...
        4            Great taffy  Great taffy at a great price.  There was a wid...
```

2 Exploratory Data Analysis

2.1 [7.1.2] 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 [3]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
        ORDER BY ProductID
        """, con)
        display.head()
```

```
Out[3]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
        0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
        1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
        2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
        3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
        4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

           HelpfulnessDenominator  Score        Time  \
        0                       2      5  1199577600
        1                       2      5  1199577600
        2                       2      5  1199577600
```

```
3                         2      5  1199577600
4                         2      5  1199577600

                                 Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                                Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As 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 [4]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False
```

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

Out[5]:
```
(364173, 10)
```

In [6]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[6]:
```
69.25890143662969
```

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

        display.head()
```

```
Out[7]:       Id   ProductId          UserId             ProfileName  \
        0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
        1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

           HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
        0                     3                       1      5  1224892800
        1                     3                       2      4  1212883200

                                               Summary  \
        0             Bought This for My Son at College
        1  Pure cocoa taste with crunchy almonds inside

                                                        Text
        0  My son loves spaghetti so I didn't hesitate or...
        1  It was almost a 'love at first bite' - the per...
```

```
In [8]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [9]: #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['Label'].value_counts()
```

```
(364171, 10)
```

```
Out[9]: 1    307061
        0     57110
        Name: Label, dtype: int64
```

2.2 7.2.3 Text Preprocessing: Stemming, stop-word removal and Lemmatization.

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

2.3 Declare a stemmer & set of stopwords to remove

In [10]:
```
# Global declaration of two variables

# set a stemmer for finding root words
snowball_stemmer = nltk.stem.SnowballStemmer('english')

# get a set of stop words in english, 3 common stop words are excluded from this set be
# it helps to identify the customer liked the product or not, less than length 3 stop w
# are eliminated from this list because all words with length less than 3 are removed f
# before applying stop word removal
stop_words_set = set(stopwords.words('english')) - {'not', 'too', 'very'}
stop_words_set = set(filter(lambda x : len(x) > 2, stop_words_set))
print('Total stop words with more than two letters', len(stop_words_set))
```

```
Total stop words with more than two letters 142
```

In [11]:
```
def decontract_phrase(phrase):
    # https://stackoverflow.com/a/47091490/4084039
    """
    This function decontracts the common phrases into its full form.
    """

    # 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 [12]:
```
def remove_repeated_letter(word):
    """
```

```
    This function truncates repeated letters (letters that occur in consequtive locatio
    three times into 3 letters.

    eg: 'The price is toooooooooooooooooooooooooooooooooooooooooooooo high !!!!' u
    to 'The price is too high !!!'

    """

    word_size = len(word)

    # if the word size is less than three return as it is
    if word_size < 3:
        return word

    # consider first two letter of the word as it is
    new_word_letters_list = list(word[0:2])

    # for every letter starting from the third position
    for letter in word[2:]:

        # case 1: the current letter we are adding is a duplicate
        if (letter == new_word_letters_list[-1]) and (letter == new_word_letters_list[-
            continue

        # case 2: the current letter we are adding is not a duplicate
        new_word_letters_list.append(letter)


    return ''.join(new_word_letters_list)
```

```
In [13]: def clean_text(text_str):
    """
    This function takes a raw text and clean it using various operations like removing
    removing special characters, removing stop words etc and returns the cleaned text.
    """

    ########### CLEAN the test ################################

    # step 1: Remove HTML tags
    html_cleanr = re.compile('<.*?>')
    text_str = re.sub(html_cleanr, ' ', text_str)

    # step 2: Convert to lower case letters
    text_str = text_str.lower()

    # step 3 : Decontract the word
    text_str = decontract_phrase(text_str)
```

```
        # step 4: Remove any consequitve occurence of letter in a word & also remove multip
        # spaces with a single space
        text_str = ' '.join(list(map(remove_repeated_letter, text_str.split())))

        # step 5: Remove special characters
        text_str = re.sub(r'[^a-z ]', r' ', text_str)

        # convert text into list of words
        text_words_list = text_str.split()

        # step 6: Remove all words with size less than three
        text_words_list = list(filter(lambda x : len(x) > 2, text_words_list))

        # step 7: remove unwanted stop words decalred in the global set of stop words
        text_words_list = list(filter(lambda x : x not in stop_words_set , text_words_list)

        # step 8: Stemming of words to find the root word
        text_words_list = list(map(snowball_stemmer.stem, text_words_list))

        # step 9 : join all words with space
        text_str = ' '.join(text_words_list)

        return text_str
```

```
In [14]: test_text = """adsFFFdfad <body> price is             tooooooooooo deeeeeee\'t Much!!!
                    loved ### $$$ it liking !!!!!!1"""
         clean_text(test_text)
```

```
Out[14]: 'adsffdfad price too dee not much adfdsf good love like'
```

3 Clean the text

```
In [15]: final['CleanedText'] = final['Text'].apply(clean_text)
         print('After cleaning size:', final.shape)

         # remove any empty review rows presnet after cleaning operation. The size of review sho
         final = final[final['CleanedText'].apply(len) > 2]
         print('After empty review removal size:', final.shape)

         # keep only the required columns
         final = final[['Id', 'ProductId', 'UserId', 'Time', 'ProfileName', 'HelpfulnessNumerato
                        'HelpfulnessDenominator', 'CleanedText', 'Summary', 'Label']]
         # sort the data frame in ascending order of time stamp, this is for time based partitio
         final = final.sort_values(['Time'])
         final.head()
```

```
After cleaning size: (364171, 11)
After empty review removal size: (364169, 11)
```

```
Out[15]:             Id   ProductId         UserId       Time  \
        138706  150524  0006641040   ACITT7DI6IDDL  939340800
        138683  150501  0006641040   AJ46FKXOVC7NR  940809600
        417839  451856  B00004CXX9   AIUWLEQ1ADEG5  944092800
        346055  374359  B00004CI84  A344SMIA5JECGM  944438400
        417838  451855  B00004CXX9   AJH6LUC1UT1ON  946857600

                          ProfileName  HelpfulnessNumerator  \
        138706        shari zychinski                     0
        138683     Nicholas A Mesiano                     2
        417839       Elizabeth Medina                     0
        346055        Vincent P. Ross                     1
        417838  The Phantom of the Opera                  0

                HelpfulnessDenominator  \
        138706                       0
        138683                       2
        417839                       0
        346055                       2
        417838                       0

                                                     CleanedText  \
        138706  witti littl book make son laugh loud recit car...
        138683  rememb see show air televis year ago child sis...
        417839  beetlejuic well written movi everyth excel act...
        346055  twist rumplestiskin captur film star michael k...
        417838  beetlejuic excel funni movi keaton hilari wack...

                                                         Summary  Label
        138706                          EVERY book is educational      1
        138683  This whole series is great way to spend time w...      1
        417839                               Entertainingl Funny!      1
        346055                            A modern day fairy tale      1
        417838                                         FANTASTIC!      1
```

In [16]:
```
# store final table into an SQlLite table for future.
final_db_path = '/home/amd_3/AAIC/ASM_REPO/Processed_data/AMZN_FOOD_REVIW/final.sqlite'

conn = sqlite3.connect(final_db_path)
c=conn.cursor()
conn.text_factory = str
final.to_sql('Reviews', conn,  schema=None, if_exists='replace', index=False,
             index_label=None, chunksize=None, dtype=None)
```

3.1 Read the final Database

In [17]:
```
## Read back the data
# using the SQLite Table to read data.
```

```
con = sqlite3.connect(final_db_path)

final_data = pd.read_sql_query(""" SELECT * FROM Reviews""", con)
# add review length
final_data['Review_Length'] = final_data['CleanedText'].apply(lambda x: len(x.split()))
final_data.head()
```

Out[17]:
```
       Id   ProductId          UserId       Time                ProfileName  \
0  150524  0006641040   ACITT7DI6IDDL  939340800            shari zychinski
1  150501  0006641040   AJ46FKXOVC7NR  940809600         Nicholas A Mesiano
2  451856  B00004CXX9   AIUWLEQ1ADEG5  944092800           Elizabeth Medina
3  374359  B00004CI84  A344SMIA5JECGM  944438400            Vincent P. Ross
4  451855  B00004CXX9   AJH6LUC1UT1ON  946857600  The Phantom of the Opera

   HelpfulnessNumerator  HelpfulnessDenominator  \
0                     0                       0
1                     2                       2
2                     0                       0
3                     1                       2
4                     0                       0

                                         CleanedText  \
0  witti littl book make son laugh loud recit car...
1  rememb see show air televis year ago child sis...
2  beetlejuic well written movi everyth excel act...
3  twist rumplestiskin captur film star michael k...
4  beetlejuic excel funni movi keaton hilari wack...

                                         Summary  Label  Review_Length
0                      EVERY book is educational      1             35
1  This whole series is great way to spend time w...      1             33
2                           Entertainingl Funny!      1             13
3                        A modern day fairy tale      1             21
4                                     FANTASTIC!      1             25
```

In [18]:
```
final_data.shape
```

Out[18]:
```
(364169, 11)
```

3.2 split data into train, validation, test

In [19]:
```
final_data[0:237800]['Label'].value_counts()
```

Out[19]:
```
1    202803
0     34997
Name: Label, dtype: int64
```

In [20]:
```
def get_train_test_split(final_df):
```

```
            # consider first 237800 points for generating train sample and remaining for  test
            # within 237800 points we have 35000 - ve samples and others are +ve, from this set
            # can take a sample of 35000 +ve, so we will have a balanced data set having 35K +v
            # points which is apt for training the model

            # partiton the data for train, test data set generation
            final_df_train = final_df[0:237800]
            final_df_test = final_df[237800:]

            # partition the data frame to positive and negative
            final_df_positive = final_df_train[final_df_train['Label'] == 1]
            final_df_negative = final_df_train[final_df_train['Label'] == 0]

            # since positive sample is dominating we select 30K samples randomly from the posit
            #  take whole negative samples
            final_df_positive = final_df_positive.sample(n=35000)

            # form train sample set
            final_train_df = final_df_positive.append(final_df_negative)
            final_train_df = final_train_df.sample(frac=1.0)
            final_train_df = final_train_df.reset_index(drop=True)

            # sample 30K points for testing
            final_test_df = final_df_test.sample(n=30000)
            final_test_df = final_test_df.reset_index(drop=True)

            print('Final train df statistics:\n', final_train_df['Label'].value_counts())
            print('\n\nFinal test df statistics:\n', final_test_df['Label'].value_counts())

            return (final_train_df, final_test_df,)
```

In [21]:
```
final_train_df, final_test_df = get_train_test_split(final_data)
```

```
Final train df statistics:
 1    35000
0    34997
Name: Label, dtype: int64


Final test df statistics:
 1    24754
0     5246
Name: Label, dtype: int64
```

In [22]:
```
final_train_df.head()
```

Out[22]:

```
       Id   ProductId          UserId        Time     ProfileName  \
0  456873  B000LKTZSM  AY6TK80W3N0KF  1266451200      Rutherford
```

```
1   81416  B001JTIFUI  A215TX3PTHM32D  1295395200     Beth "Beth"
2  519340  B003VD9MPW  A2TLXWT5XDS9PX  1299542400  Karla Robinett
3  340949  B000UJTZ80  A2BPUJL5Z0011X  1313712000     Mr. Johnson
4  453782  B00684ILVW   AV4MG4PDBLD0H  1325635200      Barbaramom

   HelpfulnessNumerator  HelpfulnessDenominator  \
0                     3                       7
1                     0                       1
2                     1                       1
3                     0                       0
4                     0                       0

                                         CleanedText  \
0  chose tomato label organ sinc discov muir glen...
1  nasti chemic aftertast smell artifici disappoi...
2  not think would like decaf starbuck decid give...
3  trap hard set trip easili push ground veri fru...
4     order item famili receiv contain one pig error

                                             Summary  Label  Review_Length
0  Dishonest labeling of "organic" when cans cont...      0            136
1                                        Undrinkable      0             11
2                                    Best Decaf Ever      1             35
3                                    Junk Do not buy      0             10
4                       Item stated 2 only received 1      0              8
```

In [23]:
```
final_test_df.head()
```

Out[23]:
```
       Id   ProductId          UserId        Time          ProfileName  \
0  401792  B0035QJARK  A34TQDJ94475A0  1330819200  Jay Endo "Jay Endo"
1  210928  B0025VF8TK   A1N7R0YP7TGWI  1328054400        JoeSchmoe155
2  565701  B002GKEK7G  A18LM9AWCHBL8U  1341878400          Christy M.
3  315160  B002BEKJUY  A22OUEFLM49JH6  1346198400      Steve and Dana
4  277135  B004CJUE8I  A36XETPK7F42BY  1348704000     Just Falcon "sf"

   HelpfulnessNumerator  HelpfulnessDenominator  \
0                     0                       0
1                     2                       4
2                     0                       0
3                     0                       0
4                     0                       0

                                         CleanedText  \
0  use fri fish catfish flounder even sea bass al...
1  far open one let put way not know salmon would...
2  weight loss fit blogger runner tri mani protei...
3  good eat straight bag believ chocol compani se...
4  love take packet travel great slushi twist lim...
```

```
                                             Summary  Label  Review_Length
0                                        Great Stuff      1             20
1  Doesn't look like a salmon, doesn't taste like...      0             73
2                         Best Protein Drink Out There!   1             46
3                                              Yummy      1             10
4  Great in bottled water, you may not need the w...      1             43
```

3.3 Observation

The train sample is almost balanced
The test sample is unbalanced like in the real case scenario (-ve : +ve = : 1: 5)
Train Test split is done by the time based splitting method
The cleaned text doesnt contain any unwanted letters

3.4 [7.2.4] Bi-Grams and n-Grams.

Motivation
Now that we have our list of words describing positive and negative reviews lets analyse them.
We begin analysis by getting the frequency distribution of the words as shown below

```
In [24]: positive_df = final_data[final_data['Label'] == 1]
         negative_df = final_data[final_data['Label'] == 0]
```

```
In [25]: all_positive_words = list()
         for item in positive_df['CleanedText'].tolist():
             all_positive_words += item.split()
         print('Fetched positvie words')

         all_negative_words = list()
         for item in negative_df['CleanedText'].tolist():
             all_negative_words += item.split()
         print('Fetched negative words')
```

```
Fetched positvie words
Fetched negative words
```

```
In [26]: freq_dist_positive=nltk.FreqDist(all_positive_words)
         freq_dist_negative=nltk.FreqDist(all_negative_words)
         print("Most Common Positive Words : ",freq_dist_positive.most_common(25))
         print("Most Common Negative Words : ",freq_dist_negative.most_common(25))
```

```
Most Common Positive Words :  [('not', 289276), ('like', 141054), ('tast', 131285), ('good', 113
Most Common Negative Words :  [('not', 94104), ('tast', 35188), ('like', 32791), ('product', 286
```

Observation:- From the above it can be seen that the most common positive and the negative words overlap for eg. 'like' could be used as 'not like' etc. So, it is a good idea to consider pairs of consequent words (bi-grams) or q sequnce of n consecutive words (n-grams)

```
In [27]: #bi-gram, tri-gram and n-gram
         count_vect = CountVectorizer(ngram_range=(1,2), min_df=0.001, max_df=0.95, max_features

         # Fit on train data
         count_vect.fit(final_train_df['CleanedText'].values)
```

```
Out[27]: CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=0.95, max_features=500, min_df=0.001,
                 ngram_range=(1, 2), preprocessor=None, stop_words=None,
                 strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                 tokenizer=None, vocabulary=None)
```

```
In [28]: print('Final vocabulary size:',len(count_vect.vocabulary_))
```

```
Final vocabulary size: 500
```

```
In [29]: print('Number of Words which are ignored due to appearing in almost all documents or it
               len(count_vect.stop_words_))
```

```
Number of Words which are ignored due to appearing in almost all documents or it is very rare 10
```

```
In [30]: print('all bigram features', list(filter(lambda x: len(x.split()) > 1, count_vect.get_f
```

```
all bigram features ['could not', 'flavor not', 'gluten free', 'green tea', 'groceri store', 'hi
```

```
In [31]: len(count_vect.get_feature_names())
```

```
Out[31]: 500
```

3.5 Observation

We got some interesting bigrams like not buy, high recommend, wast money, veri disappoint etc.

3.6 Function for saving the sparse matrix (BoW features n-grams)

```
In [32]: def save_bow_features(count_vect, df, partition_name):
             """
             This function takes a data frame and featurize the text data using
             bag of words method. This also saves the featurized data frame and the
             bag of words model to specific location.
             """

             # set output base directory
             out_base_dir = '/home/amd_3/AAIC/ASM_REPO/Processed_data/AMZN_FOOD_REVIW/BOW/'

             # vectorize the data using BOW and save the sparse matrix
```

```
            final_bigram_counts = count_vect.transform(df['CleanedText'].values)

            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])

            # create a data frame for bow features
            bow_bigram_bow = pd.DataFrame(final_bigram_counts.todense(), columns=count_vect.get
            # add the review length as additional column
            bow_bigram_bow['Review_Length'] = df['Review_Length']
            bow_bigram_bow['Label'] = df['Label']
            bow_bigram_bow['Id'] = df['Id']

            # write to disk
            bow_bigram_bow.to_csv(os.path.join(out_base_dir, partition_name + '_bow_bigram.csv'
                                  index=False)


            # dump the bow model as pickle file
            bow_model_file = open(os.path.join(out_base_dir, 'bow_model.pickle'), 'wb')
            pickle.dump(count_vect, bow_model_file)
            bow_model_file.close()
```

```
In [33]: # vectorize train, test data and save it
         save_bow_features(count_vect, final_train_df, 'train')
         save_bow_features(count_vect, final_test_df, 'test')
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (69997, 500)
the number of unique words including both unigrams and bigrams  500
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (30000, 500)
the number of unique words including both unigrams and bigrams  500
```

4 [7.2.5] TF-IDF

```
In [34]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=0.001, max_df=0.95, max_feature
         final_tf_idf = tf_idf_vect.fit_transform(final_train_df['CleanedText'].values)
```

```
In [35]: def save_tfidf_features(tfidf_vect, df, partition_name):
             """
             This function takes a data frame and featurize the text data using
             TF-IDF method. This also saves the featurized data frame and the
             TF-IDF model to specific location.
             """
```

```
    # set output base directory
    out_base_dir = '/home/amd_3/AAIC/ASM_REPO/Processed_data/AMZN_FOOD_REVIW/TFIDF/'

    # vectorize the data using BOW and save the sparse matrix
    final_bigram_tfidf = tfidf_vect.transform(df['CleanedText'].values)

    print("Type of count vectorizer ", type(final_bigram_tfidf))
    print("Shape of out text BOW vectorizer ", final_bigram_tfidf.get_shape())
    print("Number of unique words including both unigrams and bigrams ",
          final_bigram_tfidf.get_shape()[1])

    # create a data frame for bow features
    tfidf_df = pd.DataFrame(final_bigram_tfidf.todense(), columns=tfidf_vect.get_featur
    # add review length as additional column
    tfidf_df['Review_Length'] = df['Review_Length']
    tfidf_df['Label'] = df['Label']
    tfidf_df['Id'] = df['Id']

    # write to disk
    tfidf_df.to_csv(os.path.join(out_base_dir, partition_name + '_bigram_tfidf.csv'),
                    index=False)

    # dump the bow model as pickle file
    tfidf_model_file = open(os.path.join(out_base_dir, 'tfidf_model.pickle'), 'wb')
    pickle.dump(tfidf_vect, tfidf_model_file)
    tfidf_model_file.close()
```

In [36]:
```
# vectorize train, validation, test data and save it
save_tfidf_features(tf_idf_vect, final_train_df, 'train')
save_tfidf_features(tf_idf_vect, final_test_df, 'test')
```

```
Type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
Shape of out text BOW vectorizer  (69997, 500)
Number of unique words including both unigrams and bigrams  500
Type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
Shape of out text BOW vectorizer  (30000, 500)
Number of unique words including both unigrams and bigrams  500
```

5 [7.2.6] Word2Vec

In [37]:
```
# get a list of list for training the word to vec model
w2vec_train_data = final_train_df['CleanedText'].apply(lambda x : x.split())
# print sample training data for w2v
w2vec_train_data[0:3]
```

Out[37]:
```
0    [chose, tomato, label, organ, sinc, discov, mu...
1    [nasti, chemic, aftertast, smell, artifici, di...
```

```
2    [not, think, would, like, decaf, starbuck, dec...
Name: CleanedText, dtype: object
```

5.1 Train our own Word2Vec

```
In [38]: # consider only those words which appeared 5 times
         w2v_model = Word2Vec(w2vec_train_data, min_count=5, size=50, workers=4)
```

```
In [39]: 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  11211
sample words  ['chose', 'tomato', 'label', 'organ', 'sinc', 'discov', 'muir', 'glen', 'can', 'co
```

```
In [40]: w2v_model.wv.most_similar('tasti')
```

```
Out[40]: [('yummi', 0.7804615497589111),
          ('delici', 0.7529192566871643),
          ('satisfi', 0.7077146768569946),
          ('nutriti', 0.6981334686279297),
          ('nice', 0.6340436935424805),
          ('versatil', 0.6295880079269409),
          ('tastey', 0.6231692433357239),
          ('crunchi', 0.6142654418945312),
          ('good', 0.5960522890090942),
          ('dens', 0.5911023616790771)]
```

```
In [41]: w2v_model.wv.most_similar('like')
```

```
Out[41]: [('okay', 0.7673594951629639),
          ('weird', 0.7271952629089355),
          ('remind', 0.6655181646347046),
          ('gross', 0.6525304317474365),
          ('funki', 0.6406497955322266),
          ('alright', 0.640399158000946),
          ('appeal', 0.6389076709747314),
          ('resembl', 0.6381198167800903),
          ('funni', 0.6374648809432983),
          ('aw', 0.6307752132415771)]
```

6 [7.2.7] Avg W2V Vectorization

```
In [42]: def get_avg_w2v_features(w2v_model, df, partition_type):

             # set output base directory
             out_base_dir = '/home/amd_3/AAIC/ASM_REPO/Processed_data/AMZN_FOOD_REVIW/AVG_W2V/'
```

```
    # average Word2Vec
    list_of_sent = (df['CleanedText'].apply(lambda x : x.split())).tolist()

    # compute average word2vec for each review.
    sent_vectors = list() # the avg-w2v for each sentence/review is stored in this list

    # convert each review into vector format
    for sent in list_of_sent: # for each review/sentence

        sent_vec = np.zeros(50) # as word vectors are of zero length
        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

            try:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
            except:
                pass

        if cnt_words != 0:
            sent_vec /= cnt_words

        sent_vectors.append(sent_vec)

    # create a data frame
    col_names = ['dim_' + str(item) for item in range(1, 51)]
    feature_df = pd.DataFrame(sent_vectors, columns=col_names)
    feature_df['Label'] = df['Label'].tolist()
    feature_df['Id'] = df['Id']

    # write to disk
    feature_df.to_csv(os.path.join(out_base_dir, partition_type +'_avg_w2v.csv'),
                      index=False)
    # dump the bow model as pickle file
    w2v_model_file = open(os.path.join(out_base_dir, 'avg_w2v_model.pickle'), 'wb')
    pickle.dump(w2v_model, w2v_model_file)
    w2v_model_file.close()


    return feature_df
```

In [43]:
```
# create data frames for train, validation, test
# 1) Train
train_w2v_data = get_avg_w2v_features(w2v_model, final_train_df, 'train')
# 2) Test
```

```
test_w2v_data = get_avg_w2v_features(w2v_model, final_test_df, 'test')
```

6.1 TF_IDF w2v vectorization

```
In [44]: def get_tfidf_w2v_features(tf_idf_vect, w2v_model, df, partition_type):

             # set output base directory
             out_base_dir = '/home/amd_3/AAIC/ASM_REPO/Processed_data/AMZN_FOOD_REVIW/TFIDF_W2V/

             # form idf dictionary
             idf_dictionary = dict(zip(tf_idf_vect.get_feature_names(), tf_idf_vect.idf_))

             # average Word2Vec
             list_of_sent = (df['CleanedText'].apply(lambda x : x.split())).tolist()

             # TF-IDF weighted Word2Vec
             tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names


             tfidf_sent_vectors = list(); # the tfidf-w2v for each sentence/review is stored in

             for sent in list_of_sent: # 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

                 # process each review
                 for word in sent: # for each word in a review/sentence
                     try:
                         vec = w2v_model.wv[word]
                         # obtain the tf_idfidf of a word in a sentence/review
                         tf_idf = sent.count(word) * idf_dictionary[word]
                         sent_vec += (vec * tf_idf)
                         weight_sum += tf_idf

                     except:
                         pass

                 if weight_sum != 0:
                     sent_vec /= weight_sum

                 # update list
                 tfidf_sent_vectors.append(sent_vec)

             # create the feature df
             col_names = ['dim_' + str(item) for item in range(1, 51)]
             feature_df = pd.DataFrame(tfidf_sent_vectors, columns=col_names)
```

```
        feature_df['Label'] = df['Label'].tolist()
        feature_df['Id'] = df['Id']

        # write to disk
        feature_df.to_csv(os.path.join(out_base_dir, partition_type + '_tf_w2v.csv'),
                          index=False)
        # dump the bow model as pickle file
        w2v_model_file = open(os.path.join(out_base_dir, 'tfidfw2v_model.pickle'), 'wb')
        pickle.dump(w2v_model, w2v_model_file)
        w2v_model_file.close()

        return feature_df
```

```
In [45]: # create data frames for train, validation, test
         # 1) Train
         train_tfw2v_data = get_tfidf_w2v_features(tf_idf_vect, w2v_model,
                                                    final_train_df, 'train')
         # 3) Test
         test_tfw2v_data = get_tfidf_w2v_features(tf_idf_vect, w2v_model,
                                                   final_test_df, 'test')
```

7 Steps Followed

Basic EDA such as class distribution, common word presence analysis in +ve, -ve reviews

Cleaning of raw review text using methods like html tag removal, special charactes removal, stop words removal, stemming etc.

Featurization of cleaned review using four different methods 1) Bag of Words, 2) TFIDF, 3) Avg-W2v and 4) TFIDF w2v