

# Amazon Fine Food Reviews Analysis

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

EDA: <https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/>

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454

Number of users: 256,059

Number of products: 74,258

Timespan: Oct 1999 - Oct 2012

Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unique 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 considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

## [1]. Reading Data

### [1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

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

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

```

import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

```

```

C:\Users\nisha\Anaconda3\lib\site-packages\gensim\utils.py:1212: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")

```

In [2]:

```

# using 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 5000""", con)

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

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

```

Number of data points in our data (5000, 10)

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400

1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000
		ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600

In [3]:

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

In [4]:

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

(80668, 7)

Out[4]:

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

In [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

393063

## [2] Exploratory Data Analysis

### [2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out [7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delete the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

In [9]:

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

Out[9]:

(4986, 10)

In [10]:

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

Out[10]:

99.72

**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 calculations

In [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	12248926
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128832

In [12]:

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

In [13]:

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

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

```
(4986, 10)
```

```
Out[13]:
```

```
1    4178
0     808
Name: Score, dtype: int64
```

## [3] Preprocessing

### [3.1]. Preprocessing Review Text

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

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

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

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

```
In [14]:
```

```
# 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)
```

Why is this \$[...] when the same product is available for \$[...] here?  
http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<br /><br />The Victor M380 and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.

=====

I recently tried this flavor/brand and was surprised at how delicious these chips are. The best thing was that there were a lot of "brown" chips in the bsg (my favorite), so I bought some more through amazon and shared with family and friends. I am a little disappointed that there are not, so far, very many brown chips in these bags, but the flavor is still very good. I like them better than the yogurt and green onion flavor because they do not seem to be as salty, and the onion flavor is better. If you haven't eaten Kettle chips before, I recommend that you try a bag before buying bulk. They are thicker and crunchier than Lays but just as fresh out of the bag.

=====

Wow. So far, two two-star reviews. One obviously had no idea what they were ordering; the other wants crispy cookies. Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look before ordering.<br /><br />These are chocolate-oatmeal cookies. If you don't like that combination, don't order this type of cookie. I find the combo quite nice, really. The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-type consistency. Now let's also remember that tastes differ; so, I've given my opinion.<br /><br />Then, these are soft, chewy cookies -- as advertised. They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy." I happen to like raw cookie dough; however, I don't see where these taste like raw cookie dough. Both are soft, however, so is this the confusion? And, yes, they stick together. Soft cookies tend to do that. They aren't individually wrapped, which would add to the cost. Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if you want

something hard and crisp, I suggest Nabiso's Ginger Snaps. If you want a cookie that's soft, chewy and tastes like a combination of chocolate and oatmeal, give these a try. I'm here to place my second order.

love to order my coffee on amazon. easy and shows up quickly.<br />This k cup is great coffee. dcaf is very good as well

In [15]:

```
# 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)
```

Why is this \$[...] when the same product is available for \$[...] here?<br /> /><br />The Victor M380 and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.

In [16]:

```
# 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)
```

Why is this \$[...] when the same product is available for \$[...] here? />The Victor M380 and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.

I recently tried this flavor/brand and was surprised at how delicious these chips are. The best thing was that there were a lot of "brown" chips in the bsg (my favorite), so I bought some more through amazon and shared with family and friends. I am a little disappointed that there are not, so far, very many brown chips in these bags, but the flavor is still very good. I like them better than the yogurt and green onion flavor because they do not seem to be as salty, and the onion flavor is better. If you haven't eaten Kettle chips before, I recommend that you try a bag before buying bulk. They are thicker and crunchier than Lays but just as fresh out of the bag.

Wow. So far, two two-star reviews. One obviously had no idea what they were ordering; the other wants crispy cookies. Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look before ordering. These are chocolate-oatmeal cookies. If you don't like that combination, don't order this type of cookie. I find the combo quite nice, really. The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-type consistency. Now let's also remember that tastes differ; so, I've given my opinion. Then, these are soft, chewy cookies -- as advertised. They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy." I happen to like raw cookie dough; however, I don't see where these taste like raw cookie dough. Both are soft, however, so is this the confusion? And, yes, they stick together. Soft cookies tend to do that. They aren't individually wrapped, which would add to the cost. Oh yeah, chocolate chip cookies tend to be somewhat sweet. So, if you want something hard and crisp, I suggest Nabiso's Ginger Snaps. If you want a cookie that's soft, chewy and tastes like a combination of chocolate and oatmeal, give these a try. I'm here to place my second order.

love to order my coffee on amazon. easy and shows up quickly. This k cup is great coffee. dcaf is very good as well

In [17]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can't", "can not", phrase)

    # general
    phrase = re.sub(r"n't", " not", phrase)
    phrase = re.sub(r"'re", " are", phrase)
    phrase = re.sub(r"'s", " is", phrase)
    phrase = re.sub(r"'d", " would", phrase)
    phrase = re.sub(r"'ll", " will", phrase)
    phrase = re.sub(r"'t", " not", phrase)
    phrase = re.sub(r"'ve", " have", phrase)
    phrase = re.sub(r"'m", " am", phrase)
    return phrase
```

In [18]:

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

Wow. So far, two two-star reviews. One obviously had no idea what they were ordering; the other wants crispy cookies. Hey, I am sorry; but these reviews do nobody any good beyond reminding us to look before ordering.<br /><br />These are chocolate-oatmeal cookies. If you do not like that combination, do not order this type of cookie. I find the combo quite nice, really. The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-type consistency. Now let is also remember that tastes differ; so, I have given my opinion.<br /><br />Then, these are soft, chewy cookies -- as advertised. They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy." I happen to like raw cookie dough; however, I do not see where these taste like raw cookie dough. Both are soft, however, so is this the confusion? And, yes, they stick together. Soft cookies tend to do that. They are not individually wrapped, which would add to the cost. Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if you want something hard and crisp, I suggest Nabisco is Ginger Snaps. If you want a cookie that is soft, chewy and tastes like a combination of chocolate and oatmeal, give these a try. I am here to place my second order.

=====

In [19]:

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

Why is this \$[...] when the same product is available for \$[...] here?<br /><br />The Victor and traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.

In [20]:

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

Wow So far two two star reviews One obviously had no idea what they were ordering the other wants crispy cookies Hey I am sorry but these reviews do nobody any good beyond reminding us to look before ordering br br These are chocolate oatmeal cookies If you do not like that combination do not order this type of cookie I find the combo quite nice really The oatmeal sort of calms the rich chocolate flavor and gives the cookie sort of a coconut type consistency Now let is also remember that tastes differ so I have given my opinion br br Then these are soft chewy cookies as advertised They are not crispy cookies or the blurb would say crispy rather than chewy I happen to like raw cookie dough however I do not see where these taste like raw cookie dough Both are soft however so is this the confusion And yes they stick together Soft cookies tend to do that They are not individually wrapped which would add to the cost Oh yeah chocolate chip cookies tend to be somewhat sweet br br So if you want something hard and crisp I suggest Nabisco is Ginger Snaps If you want a cookie that is soft chewy and tastes like a combination of chocolate and oatmeal give these a try



I am here to place my second order

In [21]:

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

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

In [22]:

[illegible]

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey sorry review  
s nobody good beyond reminding us look ordering chocolate oatmeal cookies not like combination not  
order type cookie find combo quite nice really oatmeal sort calms rich chocolate flavor gives  
cookie sort coconut type consistency let also remember tastes differ given opinion soft chewy cook  
ies advertised not crispy cookies blurb would say crispy rather chewy happen like raw cookie dough  
however not see taste like raw cookie dough soft however confusion yes stick together soft cookies  
tend not individually wrapped would add cost oh yeah chocolate chip cookies tend somewhat sweet wa  
nt something hard crisp suggest nabiso ginger snaps want cookie soft chewy tastes like combination  
chocolate oatmeal give try place second order'

## [3.2] Preprocessing Review Summary

In [6]:

```
## Similarly you can do preprocessing for review summary also.
```

## [4] Featurization

### [4.1] BAG OF WORDS

In [24]:

```
#BoW
count_vect = CountVectorizer() #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 ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdominal',
'abiding', 'ability']
=====
```

```
the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer (4986, 12997)
the number of unique words 12997
```

### [4.2] Bi-Grams and n-Grams.

In [25]:

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

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

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer (4986, 3144)
the number of unique words including both unigrams and bigrams 3144
```

### [4.3] TF-IDF

In [26]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
```

```
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get',
'absolute', 'absolutely', 'absolutely delicious', 'absolutely love', 'absolutely no', 'according']
=====
the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer (4986, 3144)
the number of unique words including both unigrams and bigrams 3144
```

## [4.4] Word2Vec

In [27]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentence=[]
for sentence in preprocessed_reviews:
    list_of_sentence.append(sentence.split())
```

In [28]:

```
# 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 variable 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 occurred atleast 5 times
    w2v_model=Word2Vec(list_of_sentence,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 ")
```

```
[('especially', 0.9960894584655762), ('alternative', 0.9959992170333862), ('regular',
0.9959608316421509), ('right', 0.9958057403564453), ('think', 0.9957183599472046), ('crisp',
0.9956017136573792), ('tasty', 0.9955894947052002), ('greasy', 0.9955078363418579), ('amazing', 0.
9954730868339539), ('feel', 0.995471715927124)]
=====
```

```
[('popcorn', 0.9995096921920776), ('become', 0.999442458152771), ('world', 0.999393105506897), ('g
ain', 0.9993913173675537), ('none', 0.9993846416473389), ('simply', 0.9993757605552673),
('tomatoes', 0.9993616342544556), ('opinion', 0.9993601441383362), ('wow', 0.9993371367454529), ('
come', 0.9993346929550171)]
```

```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occurred minimum 5 times ", len(w2v_words))
print("sample words ", w2v_words[0:50])
```

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

```
# 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_sentence): # 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
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

4986  
50

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(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)))
```

```
# 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_sentence): # 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)]
```

[illegible]

```
# utility functions
from wordcloud import WordCloud
import matplotlib.pyplot as plt
stopwords = set(STOPWORDS)
import seaborn as sns;

def cluster_wordcloud_generated_image_fun(words_list):
    wordcloud = WordCloud(max_font_size=50, max_words=50 ,background_color="white").generate(" ".join(words_list))
    plt.figure()
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()

def gettopfeatures(data,max_feat):
    tf_idf_vect = TfidfVectorizer(max_features=max_feat)
    tfidf_vec = tf_idf_vect.fit_transform(data)
    # print("the type of count vectorizer :",type(tfidf_vec))
    print("the shape of TFIDF vectorizer : ",tfidf_vec.get_shape())
    print("the number of unique words :", tfidf_vec.get_shape()[1])

    # tfidf frequencies of top 'n' words
    idf_freq = tf_idf_vect.idf_

    return tf_idf_vect.get_feature_names() , idf_freq

# Computing the co-occurrence matrix with value of neighbourhood as neighbour_num
def cooccurrenceMatrix(data, neighbour_num , list_words):

    # Storing all words with their indices in the dictionary
    corpus = dict()
```

```

    # List of all words in the corpus
doc = []
index = 0
for sent in data:
    for word in sent.split():
        doc.append(word)
        corpus.setdefault(word, [])
        corpus[word].append(index)
        index += 1

# Co-occurrence matrix
matrix = []
# rows in co-occurrence matrix
for row in list_words:
    # row in co-occurrence matrix
    temp = []
    # column in co-occurrence matrix
    for col in list_words :
        if( col != row):
            # No. of times col word is in neighbourhood of row word
            count = 0
            # Value of neighbourhood
            num = neighbour_num
            # Indices of row word in the corpus
            positions = corpus[row]
            for i in positions:
                if i<(num-1):
                    # Checking for col word in neighbourhood of row
                    if col in doc[i:i+num]:
                        count +=1
                elif (i>=(num-1)) and (i<=(len(doc)-num)):
                    # Check col word in neighbour of row
                    if (col in doc[i-(num-1):i+1]) and (col in doc[i:i+num]):
                        count +=2
                    # Check col word in neighbour of row
                    elif (col in doc[i-(num-1):i+1]) or (col in doc[i:i+num]):
                        count +=1
                else :
                    if (col in doc[i-(num-1):i+1]):
                        count +=1

            # appending the col count to row of co-occurrence matrix
            temp.append(count)
        else:
            # Append 0 in the column if row and col words are equal
            temp.append(0)
    # appending the row in co-occurrence matrix
    matrix.append(temp)
# Return covariance matrix
return np.array(matrix)

def plotCumulativeVariance(co_occurrence_matrix):
    #Applying TruncatedSVD
    from sklearn.decomposition import TruncatedSVD
    max_features = co_occurrence_matrix.shape[1]-1

    svd = TruncatedSVD(n_components=max_features)
    svd_data = svd.fit_transform(co_occurrence_matrix)

    percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_variance_)
    cum_var_explained = np.cumsum(percentage_var_explained)
    # Plot the TruncatedSVD spectrum
    plt.figure(1, figsize=(6, 4))
    plt.clf()
    plt.plot(cum_var_explained, linewidth=2)
    plt.axis('tight')
    plt.grid()
    plt.xlabel('n_components')
    plt.ylabel('Cumulative_explained_variance')
    plt.title("Cumulative_explained_variance VS n_components")
    plt.show()

```

In [66]:

```
# Please write all the code with proper documentation
top_features,idf_freq = gettopfeatures(preprocessed_reviews,2000)
```

the shape of TFIDF vectorizer : (4986, 2000)  
the number of unique words : 2000

## [5.2] Calculation of Co-occurrence matrix

In [78]:

```
# calculating co-occurrence with the top selected features and it returns the covariance matrix
covariance_matrix = cooccurrenceMatrix(preprocessed_reviews,5,top_features)
```

In [381]:

```
covariance_matrix.shape
```

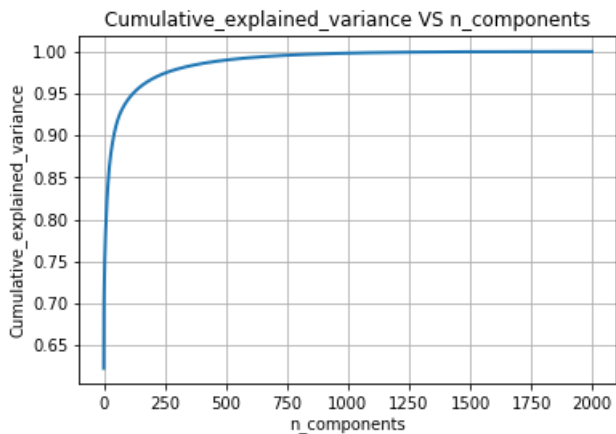
Out[381]:

(2000, 2000)

## [5.3] Finding optimal value for number of components (n) to be retained.

In [82]:

```
#choosing the n_components in truncated svd, with maximum explained variance using calculated covariance matrix
plotCumulativeVariance(covariance_matrix)
```



As we can see that at 250 we are getting 98% variance so its good to take 250 n\_components

In [86]:

```
from sklearn.decomposition import TruncatedSVD
svd_trunc = TruncatedSVD(n_components=250)
svd_transform = svd_trunc.fit_transform(covariance_matrix)
```

## [5.4] Applying k-means clustering

### [5.4.1] k-means clustering hyperparameter tuning

- choose the best number of clusters based on elbow method

In [109]:

```
# After calculating svd using optimal n components with maximum explained variance
```





\*\*\*\*\*

\*\*\*\*\*



\*\*\*\*\*



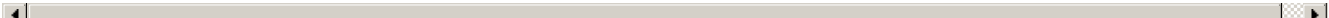
\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*



\*\*\*\*\*

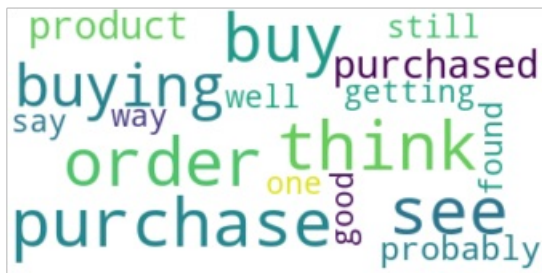




```

order - 0.886417880111772
purchase - 0.8844080813925498
however - 0.8804333125297161
think - 0.8797739789952139
see - 0.8728667974770125
buying - 0.8666778553195906
purchased - 0.8626471055429804
product - 0.8616723086131861

```



In [376]:

```

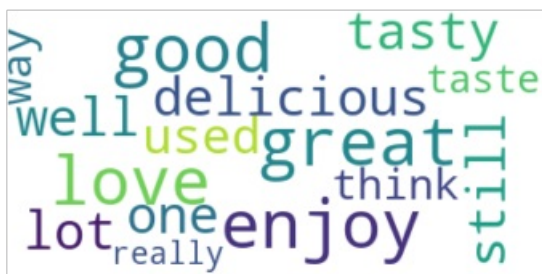
similar_words('love',svd_transform,top_features,10)
similar_words('love',svd_transform,top_features,20,False)

```

```

love - 0.9999999999999999
good - 0.890462442882289
great - 0.8896032112168769
enjoy - 0.8892855322670199
also - 0.874793130264399
delicious - 0.870395634160912
lot - 0.8681202024391418
get - 0.8643152483344635
tasty - 0.8636576007960628
like - 0.8613325762862605

```



In [380]:

```

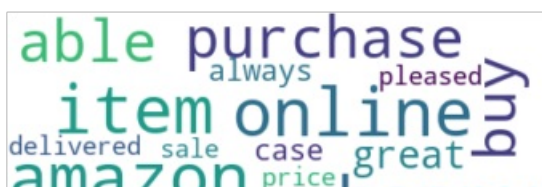
similar_words('amazon',svd_transform,top_features,10)
similar_words('amazon',svd_transform,top_features,20,False)

```

```

amazon - 0.9999999999999996
online - 0.8489445072999579
happy - 0.7946388478258145
item - 0.7736262662553128
product - 0.7676414148602599
purchase - 0.7653963989195209
able - 0.7642410922963652
get - 0.7545394925466021
buy - 0.7536814361418721
around - 0.7534842658974868

```



## [6] Conclusions

### Step-By-Step Process

1. After Text Preprocessing(i.e cleand text means we cleaned our data by removing stops words, other this which are going to affect our model) and then we will take only data not their respective class lable.
2. After Text Preprocessing Take top 2000 features from tf-idf vectorizers.
3. Computing co-occurrence matrix using these top n Features which will return covariance\_matrix
4. Doing Hyperparameter tuning for getting best n\_components with maximum explained variance using cumulative\_explained\_variance VS n\_components plot.
5. Applying TruncatedSVD on covariance\_matrix with best n\_components in order to find matrix of word-vectors .
6. Apply k-means clustering on this matrix of word-vector and choose the best number of clusters based on elbow method.
7. As we get optimal n\_clusters showing wordclouds of each clusters
8. Function that returns similar words using cosine similarity between the vectors