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 - 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.

[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 wil 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
```

```
D:\AAnaconda\lib\site-packages\gensim\utils.py:1212: UserWarning: detected Windows; aliasing chunkize t
o 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	Ti
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017

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

In [5]:

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

Out[5]:

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

In [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	
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995

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 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 [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quickso
rt', 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 calcualtions

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	
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	12248
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128

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 obsereved 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?<br />http://www.amazon.com/VICTO
R-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 amaz
on and shared with family and friends.  I am a little disappointed that there are not, so far, very man
y 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 hav
en't eaten Kettle chips before, I recommend that you try a bag before buying bulk.  They are thicker an
d 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  befo
re 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 advertis
ed.  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, ho
wever, so is this the confusion?  And, yes, they stick together.  Soft cookies tend to do that.  They a
```

```
wever, so is this the confusion?  And, yes, they stick together.  Soft cookies tend to do that.  They a
ren't individually wrapped, which would add to the cost.  Oh yeah, chocolate chip cookies tend to be so
mewhat 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 i
s very good as well
==================================================
```

In [15]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

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

In [16]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-elem
ent
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 amaz
on and shared with family and friends.  I am a little disappointed that there are not, so far, very man
y 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 hav
en't eaten Kettle chips before, I recommend that you try a bag before buying bulk.  They are thicker an
d 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  befo
re 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 f
lavor and gives the cookie sort of a coconut-type consistency.  Now let's also remember that tastes dif
fer; so, I've given my opinion.Then, these are soft, chewy cookies -- as advertised.  They are not "cri
spy" 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 c
onfusion?  And, yes, they stick together.  Soft cookies tend to do that.  They aren't individually wrap
ped, 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, chew
y and tastes like a combination of chocolate and oatmeal, give these a try.  I'm here to place my secon
d 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  bef
ore 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 r
ich 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 a
dvertised.  They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy."  I hap
pen 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 t
o be somewhat sweet.<br /><br />So, if you want something hard and crisp, I suggest Nabiso is Ginger Sn
aps.  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("\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  t
raps 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 crisp
y cookies Hey I am sorry but these reviews do nobody any good beyond reminding us to look before orderi
ng 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 gi
ves 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 t
he blurb would say crispy rather than chewy I happen to like raw cookie dough however I do not see wher
e these taste like raw cookie dough Both are soft however so is this the confusion And yes they stick t
ogether Soft cookies tend to do that They are not individually wrapped which would add to the cost Oh y
eah chocolate chip cookies tend to be somewhat sweet br br So if you want something hard and crisp I su
ggest Nabiso is Ginger Snaps If you want a cookie that is soft chewy and tastes like a combination of c
hocolate 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', "you'r
e", "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', 't
heir',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'whil
e', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'bef
ore', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'a
gain', '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"])
```

In [22]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 4986/4986 [00:02<00:00, 2235.07it/s]
```

In [23]:

```
print(preprocessed_reviews[4322])
final['Cleaned_Text'] = preprocessed_reviews
final['Cleaned_Text'].isnull().values.any()
```

```
pleased stuff smells good roommies love sniff shower ive first one got lost mail customer service sent
second one upgraded overnight delivery gets stars
```

Out[23]:

```
False
```

[3.2] Preprocessing Review Summary

In [69]:

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

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

```
#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', 'abid
ing', '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 [26]:

```
#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/skl
earn.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 [27]:

```
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 [24]:

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

In [25]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own
w2v ")
```

```
[('think', 0.9932986497879028), ('right', 0.9932862520217896), ('regular', 0.992827832698822), ('pretty
', 0.9925357699394226), ('wonderful', 0.9922983646392822), ('bad', 0.9922630190849304), ('easy', 0.9922
041296958923), ('alternative', 0.9920534491539001), ('enjoy', 0.992050290107727), ('want', 0.9920144677
16217)]
==================================================
[('gourmet', 0.9994463324546814), ('experience', 0.9993977546691895), ('must', 0.9993831515312195), ('t
ogether', 0.999355673789978), ('awful', 0.9993512630462646), ('similar', 0.9993462562561035), ('normal'
, 0.9993374347686768), ('become', 0.9993196129798889), ('major', 0.9993147253990173), ('heavy', 0.99931
15067481995)]
```

In [26]:

```
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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby', 'used'
, 'ca', 'not', 'beat', 'great', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call
', 'instead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows', 'beautif
ully', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'like', 'tv', 'computer', 'really', 'goo
d', 'idea', 'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'made']
```

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

[4.4.1.1] Avg W2v

In [27]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 3
00 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]))
```

```
100%|██████████| 4986/4986 [00:05<00:00, 887.82it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [28]:

```
# 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_)))
```

In [29]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 4986/4986 [00:29<00:00, 169.29it/s]
```

[5] Assignment 10: K-Means, Agglomerative & DBSCAN Clustering

1. **Apply K-means Clustering on these feature sets:**
   - SET 1:Review text, preprocessed one converted into vectors using (BOW)
   - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
   - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
   - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)
   - Find the best ‘k’ using the elbow-knee method (plot k vs inertia_)
   - Once after you find the k clusters, plot the word cloud per each cluster so that at a single go we can analyze the words in a cluster.

2. **Apply Agglomerative Clustering on these feature sets:**
   - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
   - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)
   - Apply agglomerative algorithm and try a different number of clusters like 2,5 etc.
   - Same as that of K-means, plot word clouds for each cluster and summarize in your own words what that cluster is representing.
   - You can take around 5000 reviews or so(as this is very computationally expensive one)

3. **Apply DBSCAN Clustering on these feature sets:**
   - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
   - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)
   - Find the best ‘Eps’ using the elbow-knee method.
   - Same as before, plot word clouds for each cluster and summarize in your own words what that cluster is representing.
   - You can take around 5000 reviews for this as well.

[5.1] K-Means Clustering

[5.1.1] Applying K-Means Clustering on BOW, SET 1

In [3]:

```
# Please write all the code with proper documentation
```

[5.1.2] Wordclouds of clusters obtained after applying k-means on BOW SET 1

In [3]:

```
# Please write all the code with proper documentation
```

[5.1.3] Applying K-Means Clustering on TFIDF, SET 2

In [3]:

```
# Please write all the code with proper documentation
```

[5.1.4] Wordclouds of clusters obtained after applying k-means on TFIDF SET 2

In [3]:

```
# Please write all the code with proper documentation
```

[5.1.5] Applying K-Means Clustering on AVG W2V, SET 3

In [3]:

```
# Please write all the code with proper documentation
```

[5.1.6] Wordclouds of clusters obtained after applying k-means on AVG W2V SET 3

In [3]:

```
# Please write all the code with proper documentation
```

[5.1.7] Applying K-Means Clustering on TFIDF W2V, SET 4

In [3]:

```
# Please write all the code with proper documentation
```

[5.1.8] Wordclouds of clusters obtained after applying k-means on TFIDF W2V SET 4

In [3]:

```
# Please write all the code with proper documentation
```

[5.2] Agglomerative Clustering

[5.2.1] Applying Agglomerative Clustering on AVG W2V, SET 3

In [127]:

```
# Please write all the code with proper documentation
from sklearn.cluster import AgglomerativeClustering
Agg_avgw2v = AgglomerativeClustering(n_clusters=3)
Agg_avgw2v.fit_predict(sent_vectors)
#centers = Agg_avgw2v.cluster_centers_
label = Agg_avgw2v.labels_.tolist()
#print(centers)
#print(label)
#add that cluster label to original dataframe as a new column
final['Agg_avgw2v_label'] = label
#final.loc[final['Kmeans_bow_label'] == 0, 'CleanedText']
```

[5.2.2] Wordclouds of clusters obtained after applying Agglomerative Clustering on AVG W2V SET 3

In [165]:

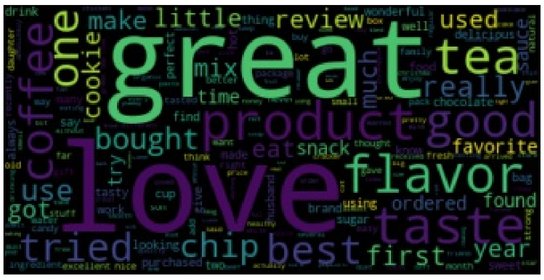
```
print(final['Agg_avgw2v_label'].max())
print(final['Agg_avgw2v_label'].unique())
#data = final.loc[final['Agg_avgw2v_label'] == 1, 'Cleaned_Text']
#print(data)
#print(final['Cleaned_Text'])
```

```
2
[2 0 1]
```

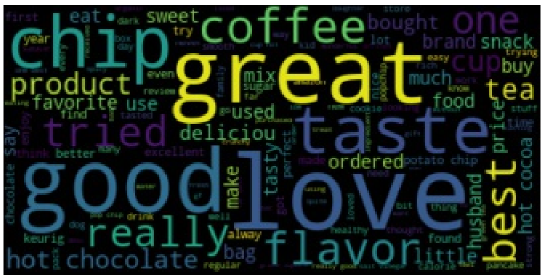
In [133]:

```python
# Please write all the code with proper documentation
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
for i in range(0,3):
    data = final.loc[final['Agg_avgw2v_label'] == i, 'Cleaned_Text']
    dataset = data.to_string()
    print("Cluster",i,":")
    wordcloud = WordCloud().generate(dataset)
    #Display the generated image:
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```

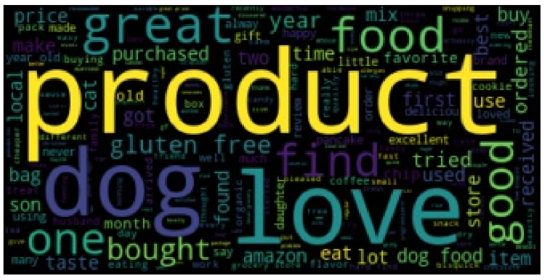
```
Cluster 0 :
```



```
Cluster 1 :
```



```
Cluster 2 :
```



[5.2.3] Applying Agglomerative Clustering on TFIDF W2V, SET 4

In [135]:

```python
# Please write all the code with proper documentation
from sklearn.cluster import AgglomerativeClustering
Agg_tfidfw2v = AgglomerativeClustering(n_clusters=5)
Agg_tfidfw2v.fit_predict(tfidf_sent_vectors)
label = Agg_tfidfw2v.labels_.tolist()
#print(centers)
#print(label)
```

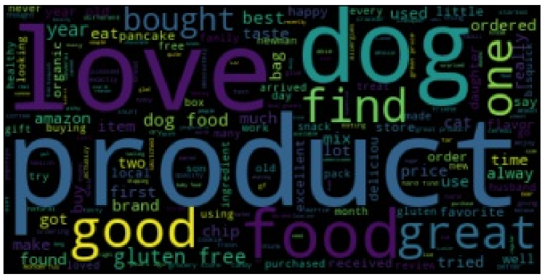
```
#add that cluster label to original dataframe as a new column
final['Agg_tfidfw2v_label'] = label
#final.loc[final['Kmeans_bow_label'] == 0, 'CleanedText']
```

[5.2.4] Wordclouds of clusters obtained after applying Agglomerative Clustering on TFIDF W2V SET 4

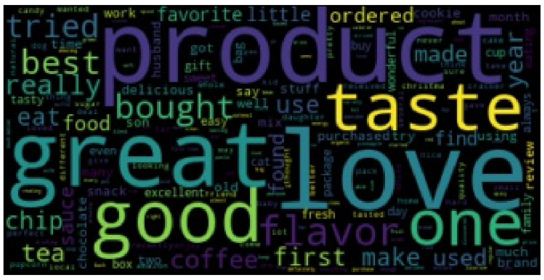
In [137]:

```
# Please write all the code with proper documentation
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
for i in range(0,5):
    data = final.loc[final['Agg_tfidfw2v_label'] == i, 'Cleaned_Text']
    dataset = data.to_string()
    print("Cluster",i,":")
    wordcloud = WordCloud().generate(dataset)
    #Display the generated image:
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```

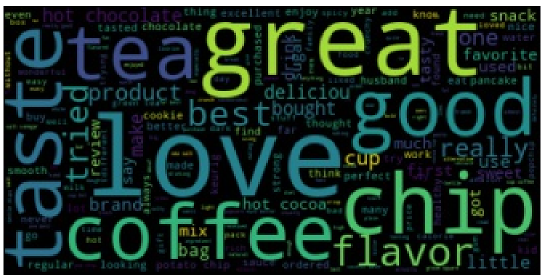
```
Cluster 0 :
```



```
Cluster 1 :
```



```
Cluster 2 :
```

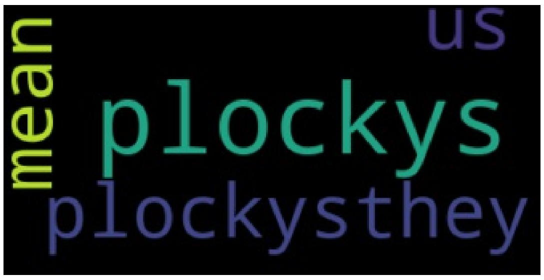


```
Cluster 3 :
```

love
best
flavor
taste
one
use
time
got

bought
great
purchased
coffee
good
ordered
product
used
review
never
found
food
buy
chocolate
made
tasty

```
Cluster 4 :
```



[5.3] DBSCAN Clustering

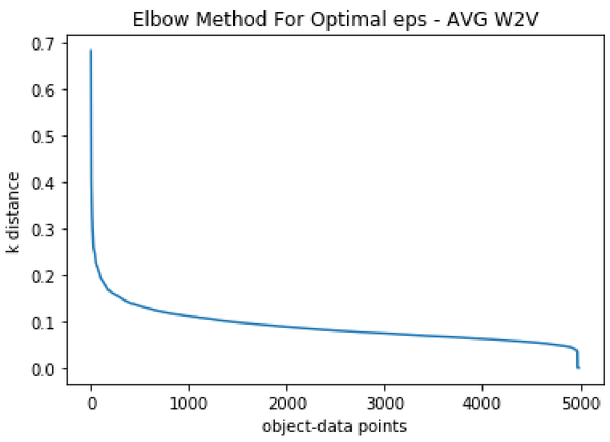
[5.3.1] Applying DBSCAN on AVG W2V, SET 3

In [53]:

```
# Please write all the code with proper documentation
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
ns = 4
nbrs = NearestNeighbors(n_neighbors=ns).fit(sent_vectors)
distances, indices = nbrs.kneighbors(sent_vectors)
distanceDec = sorted(distances[:,ns-1], reverse=True)
#plt.plot(indices[:,0], distanceDec)
plt.plot(list(range(1,4986+1)), distanceDec)
plt.xlabel('object-data points')
plt.ylabel('k distance')
plt.title('Elbow Method For Optimal eps - AVG W2V')
```

Out[53]:

```
Text(0.5, 1.0, 'Elbow Method For Optimal eps - AVG W2V')
```



In [55]:

```
# Please write all the code with proper documentation
from sklearn.cluster import DBSCAN
DBSCAN_avgw2v = DBSCAN(eps=0.17,n_jobs=-1).fit(sent_vectors)
DBSCAN_avgw2v.fit_predict(sent_vectors)
label = DBSCAN_avgw2v.labels_.tolist()
#print(label)
final['DBSCAN_avgw2v_label'] = label
#final.loc[final['Kmeans_bow_label'] == 0, 'CleanedText']
```

```
print(final['DBSCAN_avgw2v_label'].unique())
```

```
[ 0  1 -1]
```

[5.3.2] Wordclouds of clusters obtained after applying DBSCAN on AVG W2V SET 3

In [57]:

```
# Please write all the code with proper documentation
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
for i in range(0,1):
    data = final.loc[final['DBSCAN_avgw2v_label'] == i, 'Cleaned_Text']
    dataset = data.to_string()
    print("Cluster",i,":")
    wordcloud = WordCloud().generate(dataset)
    #Display the generated image:
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```

```
Cluster 0 :
```

product
love
good
great
taste
one
coffee
flavor
chip
best
bought
tried
little
tea
food

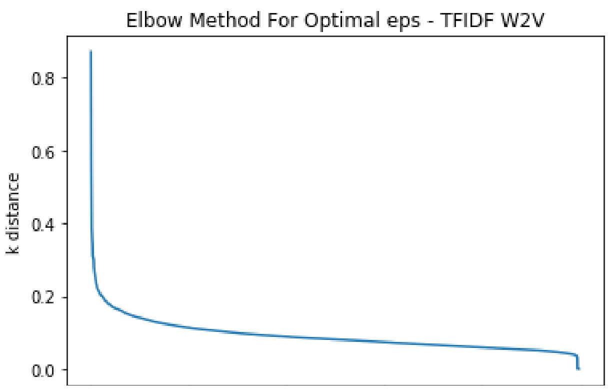
[5.3.3] Applying DBSCAN on TFIDF W2V, SET 4

In [58]:

```
# Please write all the code with proper documentation
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
ns = 4
nbrs = NearestNeighbors(n_neighbors=ns).fit(tfidf_sent_vectors)
distances, indices = nbrs.kneighbors(tfidf_sent_vectors)
distanceDec = sorted(distances[:,ns-1], reverse=True)
#plt.plot(indices[:,0], distanceDec)
plt.plot(list(range(1,4986+1)), distanceDec)
plt.xlabel('object-data points')
plt.ylabel('k distance')
plt.title('Elbow Method For Optimal eps - TFIDF W2V')
```

Out[58]:

```
Text(0.5, 1.0, 'Elbow Method For Optimal eps - TFIDF W2V')
```



In [51]:

```
# Please write all the code with proper documentation
from sklearn.cluster import DBSCAN
DBSCAN_tfidfw2v = DBSCAN(eps = 0.2,n_jobs=-1).fit(tfidf_sent_vectors)
DBSCAN_tfidfw2v.fit_predict(tfidf_sent_vectors)
label = DBSCAN_tfidfw2v.labels_.tolist()
#print(label)
final['DBSCAN_tfidfw2v_label'] = label
#final.loc[final['Kmeans_bow_label'] == 0, 'CleanedText']
print(final['DBSCAN_tfidfw2v_label'].unique())
```

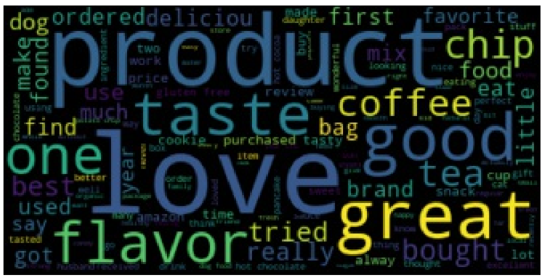
```
[ 0  1 -1  2  4  3]
```

[5.3.4] Wordclouds of clusters obtained after applying DBSCAN on TFIDF W2V SET 4

In [59]:

```
# Please write all the code with proper documentation
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
for i in range(0,1):
    data = final.loc[final['DBSCAN_tfidfw2v_label'] == i, 'Cleaned_Text']
    dataset = data.to_string()
    print("Cluster",i,":")
    wordcloud = WordCloud().generate(dataset)
    #Display the generated image:
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```

```
Cluster 0 :
```



[6] Conclusions

In [60]:

```
# Please compare all your models using Prettytable library.
# You can have 3 tables, one each for kmeans, agglomerative and dbscan
# Please compare all your models using Prettytable library

from prettytable import PrettyTable
print("Agglomerative Clustering results:")
table = PrettyTable(["model","n_clusters"])
table.add_row(["Agglomerative using AVG W2V", "3"])
table.add_row(["Agglomerative using TFIDF W2V", "5"])
print(table)

print("DBSCAN Clustering results:")
table = PrettyTable(["model","n_clusters","eps"])
table.add_row(["DBSCAN using AVG W2V", "1","0.17"])
table.add_row(["DBSCAN using TFIDF W2V", "1","0.2"])
print(table)
```

```
Agglomerative Clustering results:
```

```
+--------------------------------+------------+
|             model              | n_clusters |
+--------------------------------+------------+
|  Agglomerative using AVG W2V   |     3      |
| Agglomerative using TFIDF W2V  |     5      |
+--------------------------------+------------+
DBSCAN Clustering results:
+------------------------+------------+------+
|         model          | n_clusters | eps  |
+------------------------+------------+------+
|  DBSCAN using AVG W2V  |     1      | 0.17 |
| DBSCAN using TFIDF W2V |     1      | 0.2  |
+------------------------+------------+------+
```

Observation: I have chosen 5000 data points to perform Agglomerative and DBSCAN clustering. Agglomerative Clustering: TFIDF W2V vectorization technique is segregating the data into 5 clusters, whereas AVG W2V technique is separating into 2 clusters only. DBSCAN Clustering: For eps value 0.1, the model predicts most of the reviews as noise/outliers. And for eps value of 0.17 and 0.2,DBSCAN returns only one cluster with both TFIDF W2V and AVG W2V techniques.