11 Amazon Fine Food Reviews Analysis_Truncated SVD

May 13, 2019

1 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.

2 [1]. Reading Data

2.1 [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
```

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 point
# 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 5
```

```
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 200(

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
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 (200000, 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"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      1  1303862400
1                     0                       0      0  1346976000
2                     1                       1      1  1219017600

                 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...
```

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  \
0  #oc-R115TNMSPFT9I7  B005ZBZLT4                 Breyton  1331510400      2
```

```
1  #oc-R11D9D7SHXIJB9  B005HG9ESG  Louis E. Emory "hoppy"  1342396800      5
2  #oc-R11DNU2NBKQ23Z  B005ZBZLT4        Kim Cieszykowski  1348531200      1
3  #oc-R11O5J5ZVQE25C  B005HG9ESG           Penguin Chick  1346889600      5
4  #oc-R12KPBODL2B5ZD  B007OSBEV0   Christopher P. Presta  1348617600      1

                                                Text  COUNT(*)
0  Overall its just OK when considering the price...         2
1  My wife has recurring extreme muscle spasms, u...         3
2  This coffee is horrible and unfortunately not ...        2
3  This will be the bottle that you grab from the...         3
4  I didnt like this coffee. Instead of telling y...         2
```

In [5]:
```python
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:
```
              UserId   ProductId                     ProfileName        Time  \
80638  AZY10LLTJ71NX  B001ATMQK2  undertheshrine "undertheshrine"  1296691200

       Score                                               Text  COUNT(*)
80638      5  I bought this 6 pack because for the price tha...         5
```

In [6]:
```python
display['COUNT(*)'].sum()
```

Out[6]: 393063

3 [2] Exploratory Data Analysis

3.1 [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]:
```python
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  \
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 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=Fals
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
final.shape
```

Out[9]: (160178, 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]: 80.089

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  \
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 [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()
```

```
(160176, 10)
```

Out[13]:
```
1    134799
0     25377
Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [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)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as good as I rememb
==================================================
The qualitys not as good as the lamb and rice but it didn't seem to bother his stomach, you get
==================================================
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loan-word, and&quot;ko-&quo
==================================================
What can I say... If Douwe Egberts was good enough for my dutch grandmother, it's perfect for m
==================================================
```

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)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as good as I rememb
```

In [16]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-
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)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as good as I rememb
==================================================
The qualitys not as good as the lamb and rice but it didn't seem to bother his stomach, you get
==================================================
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loan-word, and"ko-"  is "ch
==================================================
What can I say... If Douwe Egberts was good enough for my dutch grandmother, it's perfect for m
```

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)
```

```
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loan-word, and&quot;ko-&quo
==================================================
```

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)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as good as I rememb
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
This is the Japanese version of breadcrumb pan bread a Portuguese loan word and quot ko quot is
```

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', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', '
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```
# Combining all the above stundents
from tqdm import tqdm
prepr_rev = []
# 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 stopwo
            prepr_rev.append(sentance.strip())
```

```
100%|| 160176/160176 [01:48<00:00, 1475.27it/s]
```

```
In [23]: prepr_rev[1500]
```

```
Out[23]: 'japanese version breadcrumb pan bread portuguese loan word ko child derived panko use
```

```
In [24]: print(len(prepr_rev))
         final.shape
```

```
160176
```

```
Out[24]: (160176, 10)
```

[3.2] Preprocessing Review Summary

```
In [26]: ## Similartly you can do preprocessing for review summary also.
         preprocessed_summary = []
         # tqdm is for printing the status bar
         for summary in tqdm(final['Summary'].values):
             summary = re.sub(r"http\S+", "", summary) # remove urls from text python: https:/
             summary = BeautifulSoup(summary, 'lxml').get_text() # https://stackoverflow.com/q
             summary = decontracted(summary)
             summary = re.sub("\S*\d\S*", "", summary).strip() #remove words with numbers pytho
             summary = re.sub('[^A-Za-z]+', ' ', summary) #remove spacial character: https://s
             # https://gist.github.com/sebleier/554280
             summary = ' '.join(e.lower() for e in summary.split() if e.lower() not in stopword
             preprocessed_summary.append(summary.strip())
```

```
 58%|    | 92304/160176 [00:25<00:18, 3680.91it/s]/Volumes/Saida/Applications/Anaconda/anaconda
  ' Beautiful Soup.' % markup)
100%|| 160176/160176 [00:45<00:00, 3552.28it/s]
```

```
In [27]: prepr_rev = [i + ' ' + j for i, j in zip(prepr_rev,preprocessed_summary)]
         print(prepr_rev[1500])
```

```
japanese version breadcrumb pan bread portuguese loan word ko child derived panko used katsudon
```

```
In [28]: final ['CleanText']= prepr_rev
         final.head(5)
```

Out[28]:
```
             Id   ProductId          UserId                      ProfileName  \
138695  150513  0006641040   ASH0DZQQF6AIZ                         tessarat
138707  150525  0006641040  A2QID6VCFTY51R                             Rick
138708  150526  0006641040  A3E9QZFE9KXH8J                      R. Mitchell
138686  150504  0006641040   AQEYF1AXARWJZ      Les Sinclair "book maven"
138685  150503  0006641040  A3R5XMPFU8YZ4D  Her Royal Motherliness "Nana"

        HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
138695                     0                       0      1  1325721600
138707                     1                       2      1  1025481600
138708                    11                      18      0  1129507200
138686                     1                       1      1  1212278400
138685                     1                       1      1  1233964800

                                                   Summary  \
138695                                           A classic
138707  In December it will be, my snowman's anniversa...
138708                                awesome book poor size
138686                                Chicken Soup with Rice
138685                                        so fun to read

                                                      Text  \
138695  I remembered this book from my childhood and g...
138707  My daughter loves all the "Really Rosie" books...
138708  This is one of the best children's books ever ...
138686  A very entertaining rhyming story--cleaver and...
138685  This is my grand daughter's and my favorite bo...

                                                 CleanText
138695  remembered book childhood got kids good rememb...
138707  daughter loves really rosie books introduced r...
138708  one best children books ever written mini vers...
138686  entertaining rhyming story cleaver catchy illu...
138685  grand daughter favorite book read loves rhythm...
```

In [30]:
```
##Sorting data for Time Based Splitting
final = final.sort_values('Time',axis= 0,inplace = False , na_position = 'last',ascend
X_train = final['CleanText'].values
X_train = X_train[:50000]
print(X_train.shape)
```

```
(50000,)
```

4.2 TF-IDF

In [31]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,1),use_idf=True, min_df=10, max_features
tf_idf_vect.fit(X_train)
```

```
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names
print('='*50)

final_tf_idf = tf_idf_vect.transform(X_train)
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
```

```
some sample features(unique words in the corpus) ['able', 'absolute', 'absolutely', 'according
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (50000, 2000)
the number of unique words including both unigrams and bigrams  2000
```

5 [5] Assignment 11: Truncated SVD

```
<li><strong>Apply Truncated-SVD on only this feature set:</strong>
    <ul>
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vector
<br>
<li><strong>Procedure:</strong>
    <ul>
<li>Take top 2000 or 3000 features from tf-idf vectorizers using idf_ score.</li>
<li>You need to calculate the co-occurrence matrix with the selected features (Note: X.X^T
```

doesn't give the co-occurrence matrix, it returns the covariance matrix, check these bolgs blog-1, blog-2 for more information)

```
        <li>You should choose the n_components in truncated svd, with maximum explained
```

variance. Please search on how to choose that and implement them. (hint: plot of cumulative explained variance ratio)

```
        <li>After you are done with the truncated svd, you can apply K-Means clustering and ch
```

the best number of clusters based on elbow method.

```
        <li> Print out wordclouds for each cluster, similar to that in previous assignment. </
        <li>You need to write a function that takes a word and returns the most similar words
```

cosine similarity between the vectors(vector: a row in the matrix after truncatedSVD)

```
    </ul>
</li>
<br>
```

5.1 Truncated-SVD

5.1.1 Taking top features from TFIDF

In [32]:
```
# Please write all the code with proper documentation final_tf_idf
def top_ft(tf_idf_vect, ft_names, top_n_ft):
    # returns top n features :
    ft_index = np.argsort(tf_idf_vect.idf_)
    top_ft = [(ft_names[i], tf_idf_vect.idf_[i]) for i in ft_index[:top_n_ft]]
    data_topFt = pd.DataFrame(data=top_ft, columns = ['Top Words','TFIDF Values'])
    return data_topFt

#Get TF-IDF Scores mean:
tfidf_mean = np.mean(final_tf_idf, axis = 0)
tfidf_mean = np.array(tfidf_mean)[0].tolist()

#List feature names:
ft_names = tf_idf_vect.get_feature_names()

#Top TFIDF words with their scores:
top_n_ft = 2000
top_ft = top_ft(tf_idf_vect, ft_names, top_n_ft)

#Print the top 20 features.

top_ft.head(20)
```

Out[32]:

	Top Words	TFIDF Values
0	not	1.630192
1	great	2.124504
2	good	2.192394
3	like	2.241902
4	taste	2.477640
5	one	2.527141
6	product	2.590931
7	love	2.612964
8	flavor	2.683212
9	would	2.703000
10	best	2.764592
11	no	2.888378
12	get	2.922526
13	really	2.981101
14	amazon	3.018178
15	much	3.045021
16	also	3.095266
17	find	3.111159
18	time	3.116458
19	little	3.149198

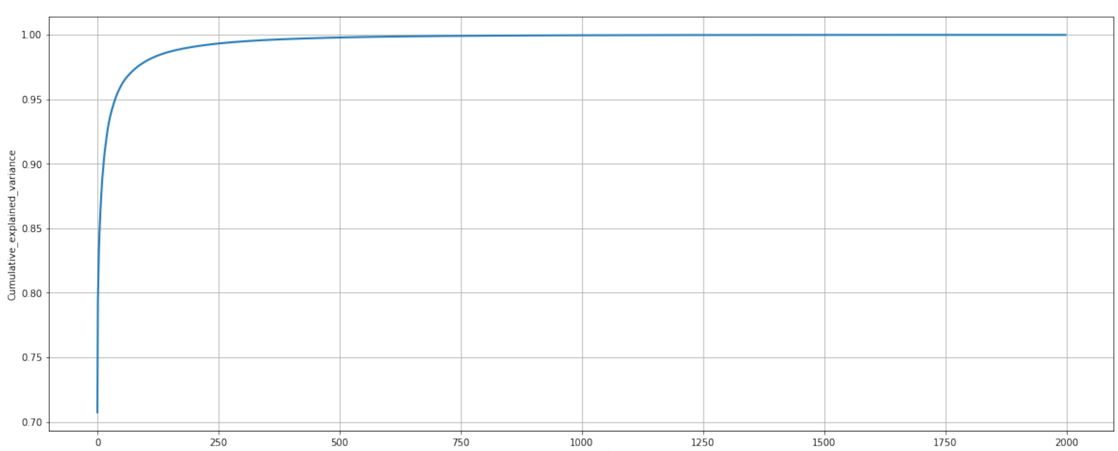
5.1.2 Calulation of Co-occurrence matrix

In [33]:
```python
# Please write all the code with proper documentation
def co_Mat(X_train, top_ft, window):
    print("Generate the Co-Occurence Matrix....")
    dim=top_ft.shape[0]
    square_matrix = np.zeros((dim,dim),int)

    values = [i for i in range(0,top_ft.shape[0])]
    keys = [str(i) for i in top_ft['Top Words']]
    lookup_dict = dict(zip(keys,values))

    top_words= keys

    for row in tqdm(X_train):
        #Split each review into words
        words_in_row = row.split()
        lnt = len(words_in_row)
        for i in range(0,len(words_in_row),1):
            idx_of_neigbors= []
            if((i-window >= 0) and (i+window < lnt)):
                idx_of_neigbors = np.arange(i-window,i+window+1)
            elif((i-window < 0) and (i+window < lnt)):
                idx_of_neigbors = np.arange(0, i+window+1)
            elif((i-window >= 0) and (i+window >= lnt)):
                idx_of_neigbors = np.arange(i-window, lnt)
            else:
                pass

            #neigh = [words_in_row[x] for x in idx_of_neigbors]
            #print(words_in_row[i],"********",neigh)
            #print(idx_of_neigbors)


            for j in idx_of_neigbors:
                if((words_in_row[j] in top_words) and (words_in_row[i] in top_words))
                    row_idx = lookup_dict[words_in_row[i]]
                    col_idx = lookup_dict[words_in_row[j]]
                    square_matrix[row_idx,col_idx] += 1
                else:
                    pass

    np.fill_diagonal(square_matrix, 0)
    print("Generate Co-Oc Matrix")

    #Create a co-occurence df.
    co_Mat_df=pd.DataFrame(data=square_matrix, index=keys, columns=keys)
    return co_Mat_df
```

```
co_Mat = co_Mat(X_train, top_ft, window=5)

co_Mat.to_csv('coc_Mat.csv')
```

```
  0%|          | 6/50000 [00:00<15:53, 52.43it/s]

Generate the Co-Occurence Matrix...

100%|| 50000/50000 [09:26<00:00, 88.30it/s]

Generate Co-Oc Matrix
```

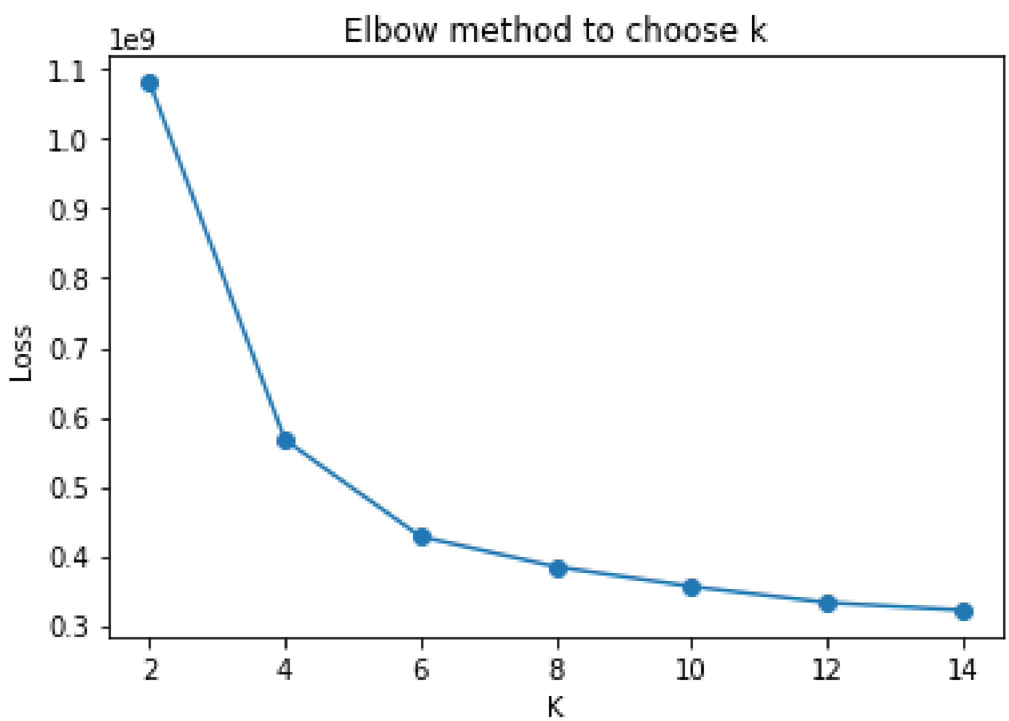
```
In [34]: from sklearn.decomposition import TruncatedSVD

         n = co_Mat.shape[0]-1

         #Inititalizing truncated SVD.
         svd = TruncatedSVD(n_components=n,
                            algorithm='randomized',
                            n_iter=7,
                            random_state=42)
         svd_tfidf=svd.fit_transform(co_Mat)

         cum_var_explained = np.cumsum(svd.explained_variance_ratio_)

         # Plot the SVD
         plt.figure(1, figsize=(20, 8))
         plt.plot(cum_var_explained, linewidth=2)
         plt.grid()
         plt.axis('tight')
         plt.xlabel('n_components')
         plt.ylabel('Cumulative_explained_variance')
         plt.show()
```



5.1.3 [5.4] Applying k-means clustering

In [35]:
```
# Please write all the code with proper documentation
# Elbow method to find K
from sklearn.cluster import KMeans
def find_optimal_k(svd_tfidf):
    inertia = []
    k = [2,4,6,8,10,12,14]
    for i in k:
        km = KMeans(n_clusters = i)
        km.fit(svd_tfidf)
        inertia.append(km.inertia_)
    plt.plot(k, inertia, "-o")
    plt.title("Elbow method to choose k")
    plt.xlabel("K")
    plt.ylabel("Loss")
    plt.show()
```

In [36]:
```
find_optimal_k(svd_tfidf)
```



In [37]:
```
model = KMeans(n_clusters=50, random_state=42, n_init=50)
model.fit(svd_tfidf)
```

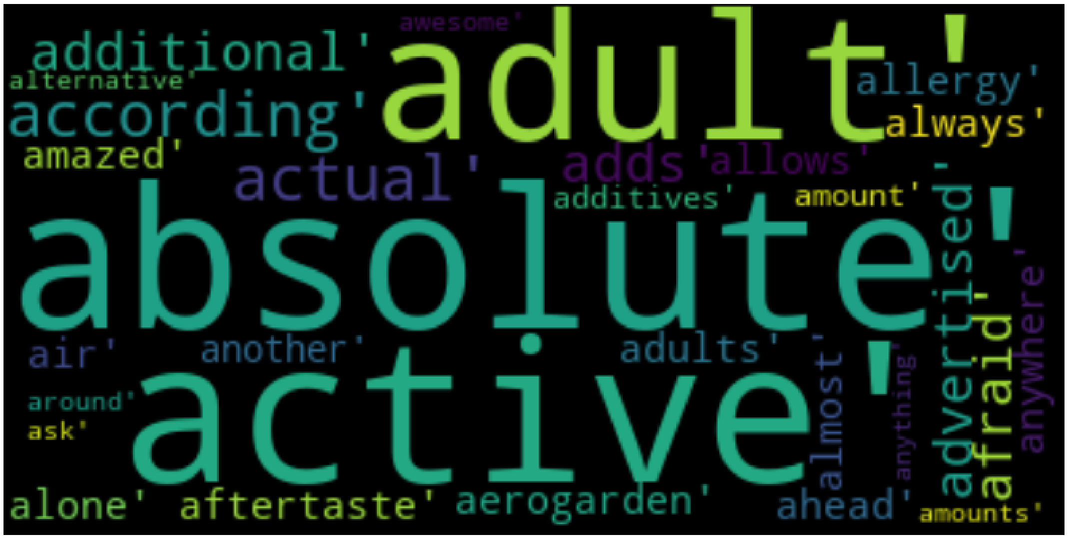
```
#Get the cluster centroid values.
print("The cluster centroids:")
print(model.cluster_centers_)
```

```
The cluster centroids:
[[ 9.15202997e+03 -1.74910147e+03  1.84845344e+03 ... -1.30726180e-04
  -1.22164771e-04  5.63185185e-05]
 [ 1.74707988e+02 -2.29916145e+01  3.43285369e-01 ...  2.73083536e-04
  -6.61958219e-05  6.03000328e-05]
 [ 2.83763346e+03 -1.18421300e+02 -2.00524913e+02 ...  3.74600772e-05
  -6.33542560e-06  5.91428468e-06]
 ...
 [ 2.71189592e+03 -3.29429147e+02 -6.77229807e+01 ... -1.64089237e-04
   2.21422475e-04 -8.81004841e-06]
 [ 3.69052478e+03  1.13113830e+02  1.86938780e+02 ... -5.49999154e-05
   8.51940711e-05  9.36505068e-05]
 [ 1.47205739e+03 -5.61471668e+01  3.74837857e+01 ...  3.06392562e-04
   2.93683420e-05 -5.57502115e-05]]
```

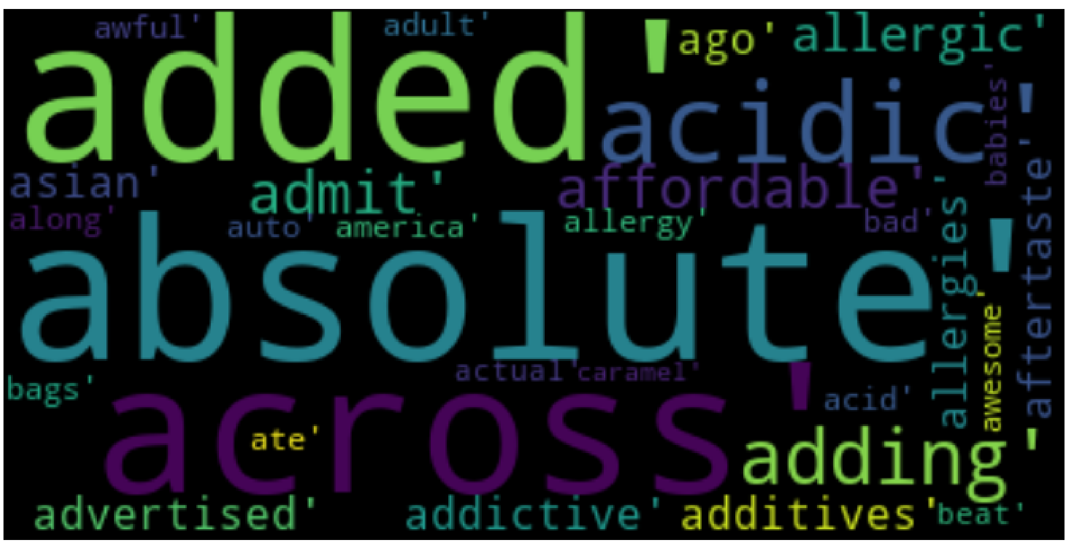
5.1.4 [5.5] Wordclouds of clusters obtained in the above section

```
In [38]: # Please write all the code with proper documentation
         from wordcloud import WordCloud


         centroids = model.cluster_centers_.argsort() # function for printing top 30 feature n

         terms = tf_idf_vect.get_feature_names()

         list1 = []

         for i in range(10):
             print("Cluster %d:" % i, end='')
             for j in centroids[i, :30]:
                 list1.append(terms[j])
             wc = WordCloud(background_color="black", max_words=len(str(list1)))
             wc.generate(str(list1))
             print("Word Cloud for KMeans Cluster:", i)
             plt.figure(figsize = (12,12), facecolor = None)
             plt.imshow(wc, interpolation='bilinear')
             plt.axis("off")
             plt.show()
             list1.clear()
```

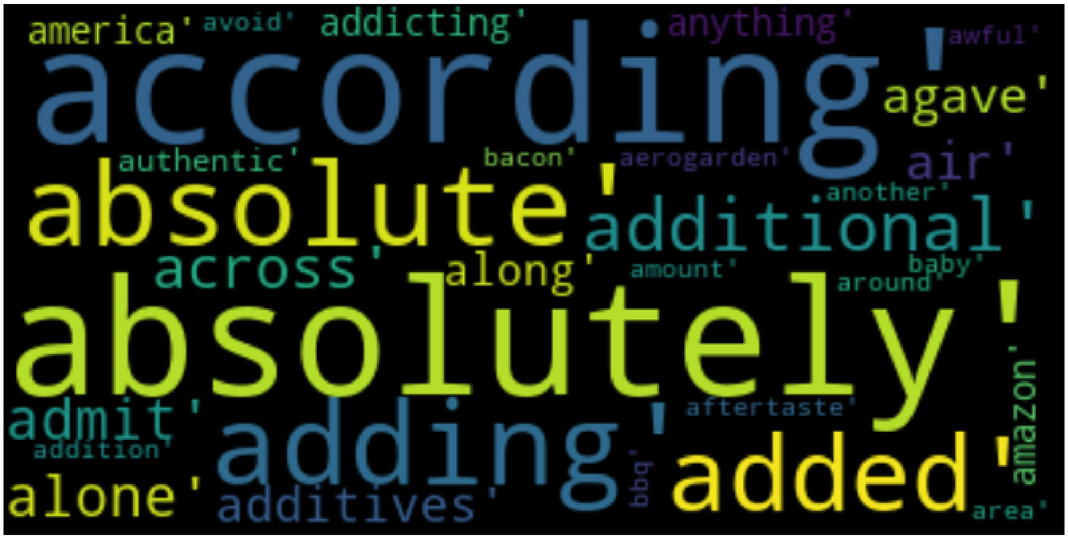
```
Cluster 0:Word Cloud for KMeans Cluster: 0
```



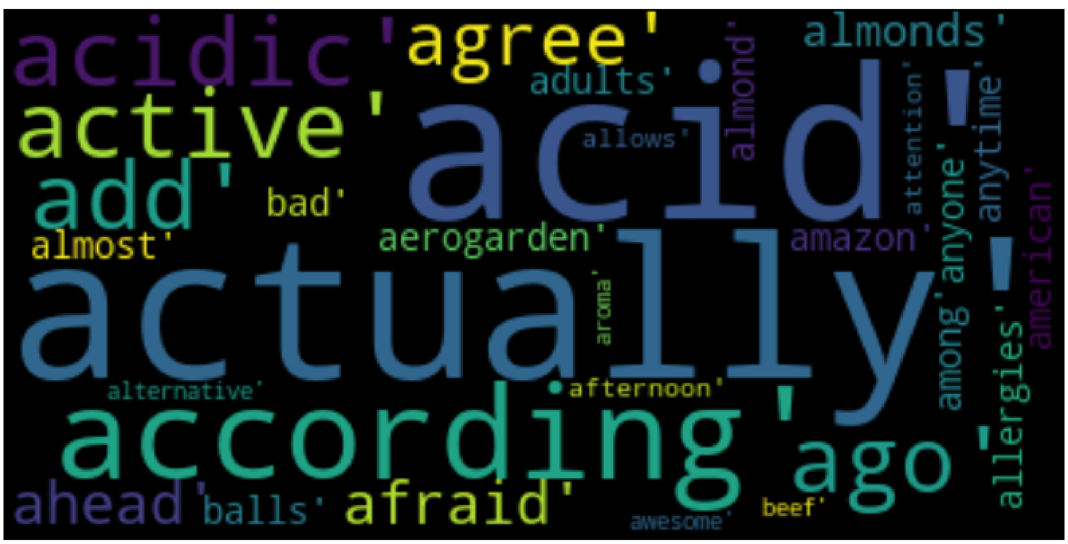
Cluster 1:Word Cloud for KMeans Cluster: 1



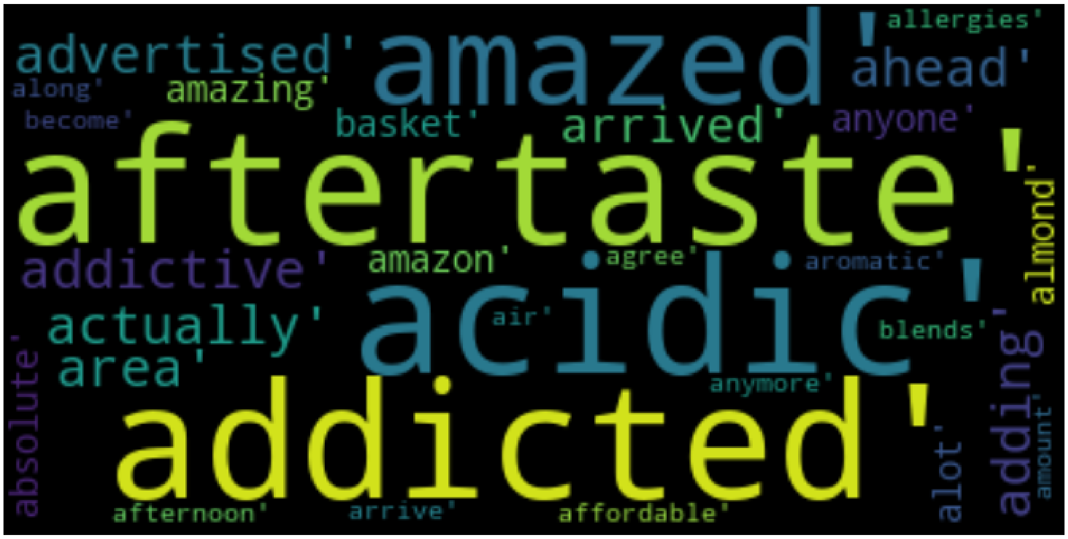
Cluster 2:Word Cloud for KMeans Cluster: 2



Cluster 3:Word Cloud for KMeans Cluster: 3



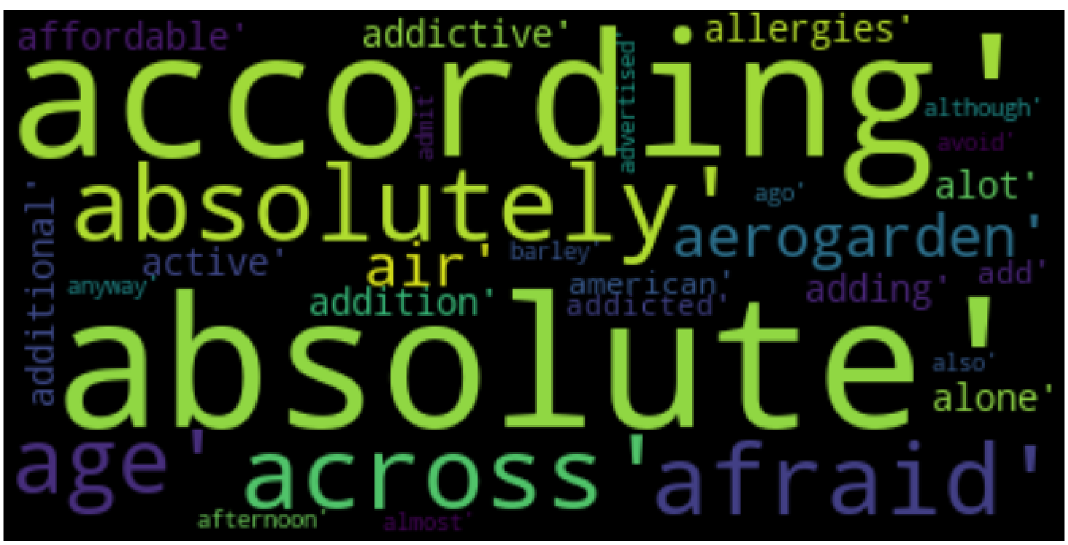
Cluster 4:Word Cloud for KMeans Cluster: 4



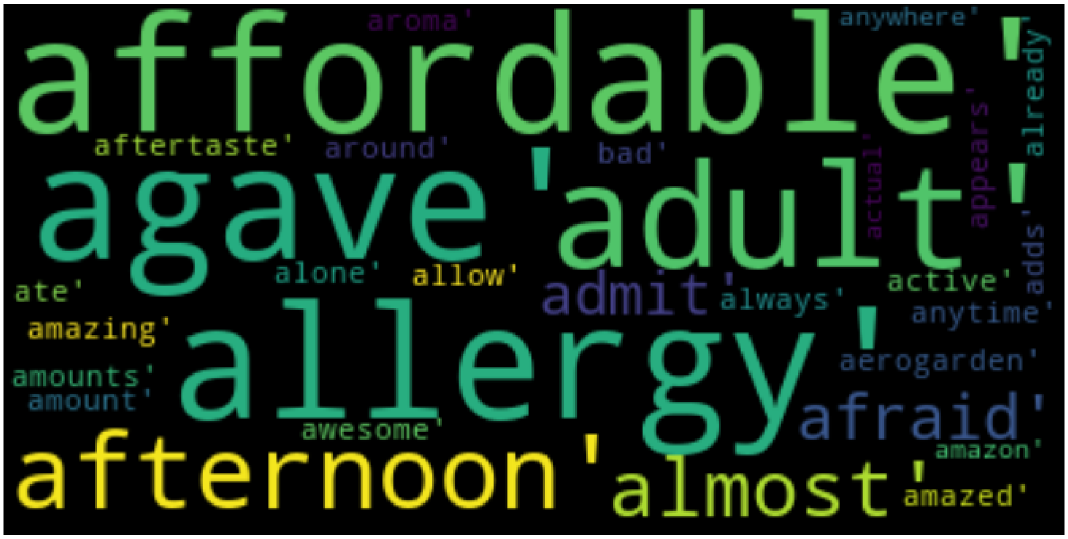
Cluster 5:Word Cloud for KMeans Cluster: 5

Cluster 6:Word Cloud for KMeans Cluster: 6

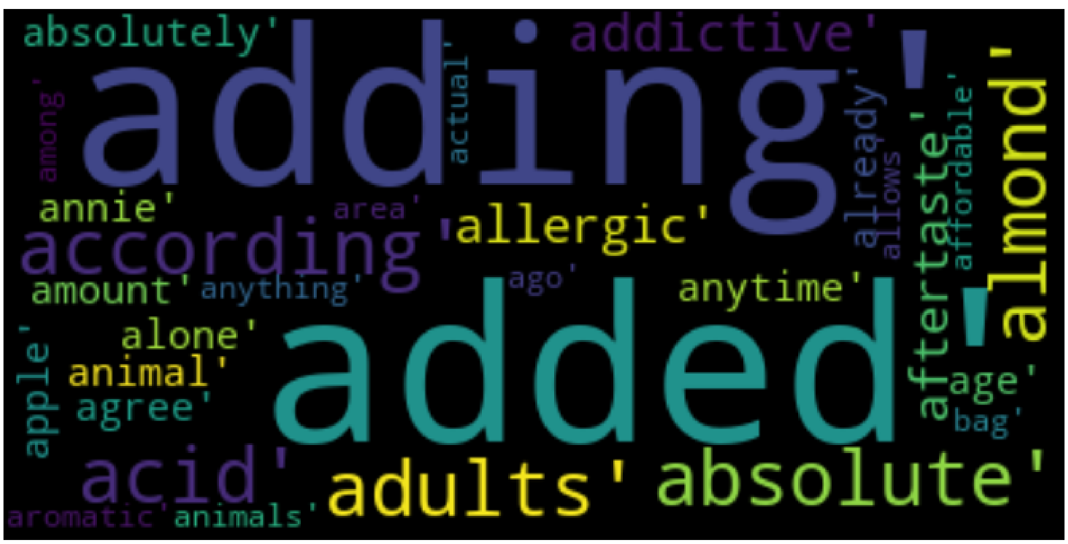
Cluster 7:Word Cloud for KMeans Cluster: 7



Cluster 8:Word Cloud for KMeans Cluster: 8




```
Cluster 9:Word Cloud for KMeans Cluster: 9
```



5.1.5 Function that returns most similar words for a given word.

```
In [39]: top_ft = tf_idf_vect.get_feature_names()
```

```
In [44]: # Please write all the code with proper documentation
         from sklearn.metrics.pairwise import cosine_similarity
```

```python
def similar_word(word):
    similarity = cosine_similarity(co_Mat)
    word_vect = similarity[top_ft.index(word)]
    print("Similar Word to",word)
    index = word_vect.argsort()[::-1][1:21]
    for j in range(len(index)):
        print((j+1),"Word",top_ft[index[j]] ,"is similar to",word,"\n")
```

In [45]: `similar_word(top_ft[1])`

```
Similar Word to absolute
1 Word amazon is similar to absolute

2 Word child is similar to absolute

3 Word busy is similar to absolute

4 Word celiac is similar to absolute

5 Word absolutely is similar to absolute

6 Word apples is similar to absolute

7 Word lab is similar to absolute

8 Word ago is similar to absolute

9 Word issues is similar to absolute

10 Word although is similar to absolute

11 Word barely is similar to absolute

12 Word glad is similar to absolute

13 Word breads is similar to absolute

14 Word afternoon is similar to absolute

15 Word difficult is similar to absolute

16 Word allergy is similar to absolute

17 Word average is similar to absolute

18 Word active is similar to absolute

19 Word cannot is similar to absolute
```

```
20 Word area is similar to absolute
```

6 [6] Conclusions

6.0.1 Please write down few lines about what you observed from this assignment.

6.0.2 Also please do mention the optimal values that you obtained for number of components & number of clusters.

-The optimal value of K from the elbow plot is 6.
-the most important features using TF-IDF vectorizer along with their tf_idf scores.
-Computed the co-occurence matrix from the tf-idf vectors and applied truncated SVD on it.
-Applied K means with optimal k value by using the elbow method.
-I have shown words clouds.
-The model can be improved more.