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 [2]:
```
%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:\ProgramData\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWa
rning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_seria
l")
```

In [3]:

```
# using SQLite Table to read data.
con = sqlite3.connect('C:/Users/Excel/Desktop/vins/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 50
0000 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 Sco
re != 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 sc
ore<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[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [4]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [5]:
```
print(display.shape)
display.head()
```

(80668, 7)

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU

	UserId	ProductId	ProfileName	Time	Score	Text	COU
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 [6]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[6]:

	UserId	ProductId	ProfileName	Time	Score	Text

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

In [7]:
```
display['COUNT(*)'].sum()
```

Out[7]: 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 [8]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

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 [9]:
```
#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 [10]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[10]:
```
(4986, 10)
```

In [11]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[11]:
```
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 [12]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [13]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [14]:
```
#Before starting the next phase of preprocessing lets see the number of
 entries left
print(final.shape)
```

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

```
(4986, 10)
```

Out[14]:

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

In [15]:

```python
final['Time']=pd.to_datetime(final['Time'],unit='s')
final=final.sort_values(by='Time')
```

[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 [16]:

```python
# printing some random reviews
```

```python
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)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
==================================================
These are thin,crisp, fragrant cookies and are very delicious and tast
y. They are excellent with a glass of cold almond milk or hot herbal te
a. (my choices) If you like ginger snaps you will love Lars ginger snap
s.
==================================================
Green Mountain "Nantucket Blend" K-Cups make a very good cup of coffee
in my <a href="http://www.amazon.com/gp/product/B000AQPMHA">Keurig B-40
B40 Elite Gourmet Single-Cup Home-Brewing System</a>. This is a very sm
ooth tasting brew that my wife prefers over the <a href="http://www.ama
zon.com/gp/product/B0029XDZIK">Coffee People, Donut Shop K-Cups for Keu
rig Brewers (Pack of 50) [Amazon Frustration-Free Packaging</a>] I gene
rally drink in the morning.<br /><br />These are good on both "Small" a
nd "Large" cup settings as well.<br /><br />Highly Recommended!<br /><b
r />CFH
==================================================
Besides being smaller than runts, they look the same and have the same
consistency.  Unfortunately, they taste nothing like banana runts...nor
do they even taste good.  Yucky stuff.  Trying to return with vendor.
==================================================
```

In [17]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
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)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
```

In [18]:

```python
# 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)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
==================================================
```

```
These are thin,crisp, fragrant cookies and are very delicious and tast
y. They are excellent with a glass of cold almond milk or hot herbal te
a. (my choices) If you like ginger snaps you will love Lars ginger snap
s.
==================================================
Green Mountain "Nantucket Blend" K-Cups make a very good cup of coffee
in my Keurig B-40 B40 Elite Gourmet Single-Cup Home-Brewing System. Thi
s is a very smooth tasting brew that my wife prefers over the Coffee Pe
ople, Donut Shop K-Cups for Keurig Brewers (Pack of 50) [Amazon Frustra
tion-Free Packaging] I generally drink in the morning.These are good on
both "Small" and "Large" cup settings as well.Highly Recommended!CFH
==================================================
Besides being smaller than runts, they look the same and have the same
consistency.  Unfortunately, they taste nothing like banana runts...nor
do they even taste good.  Yucky stuff.  Trying to return with vendor.
```

In [19]:

```
# 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 [20]:

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

```
Green Mountain "Nantucket Blend" K-Cups make a very good cup of coffee
in my <a href="http://www.amazon.com/gp/product/B000AQPMHA">Keurig B-40
B40 Elite Gourmet Single-Cup Home-Brewing System</a>. This is a very sm
ooth tasting brew that my wife prefers over the <a href="http://www.ama
zon.com/gp/product/B0029XDZIK">Coffee People, Donut Shop K-Cups for Keu
rig Brewers (Pack of 50) [Amazon Frustration-Free Packaging</a>] I gene
rally drink in the morning.<br /><br />These are good on both "Small" a
nd "Large" cup settings as well.<br /><br />Highly Recommended!<br /><b
r />CFH
==================================================
```

In [21]:

```
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
```

In [22]:

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

```
Green Mountain Nantucket Blend K Cups make a very good cup of coffee in
my a href http www amazon com gp product B000AQPMHA Keurig B 40 B40 Eli
te Gourmet Single Cup Home Brewing System a This is a very smooth tasti
ng brew that my wife prefers over the a href http www amazon com gp pro
duct B0029XDZIK Coffee People Donut Shop K Cups for Keurig Brewers Pack
of 50 Amazon Frustration Free Packaging a I generally drink in the morn
ing br br These are good on both Small and Large cup settings as well b
r br Highly Recommended br br CFH
```

In [23]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <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', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', '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', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', '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', "is
n'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 [24]:

```
# 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, 187
8.57it/s]
```

In [25]:
```
preprocessed_reviews[1500]
```

Out[25]:
```
'green mountain nantucket blend k cups make good cup coffee generally d
rink morning good small large cup settings well highly recommended cfh'
```

[3.2] Preprocessing Review Summary

In [26]:
```
final['preprocessed_reviews']=preprocessed_reviews
```

In [27]:
```
X=final['preprocessed_reviews'].values
```

In [29]:
```
from sklearn.preprocessing import StandardScaler
sent_x = []
for sent in X :
    sent_x.append(sent.split())


# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(sent_x,min_count=5,size=50, workers=4)

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

# compute average word2vec for each review for sent_x .
train_vectors = [];
```

```
for sent in tqdm(sent_x):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)

data = StandardScaler().fit_transform(train_vectors)
```

```
number of words that occured minimum 5 times  3817
```

```
100%|██████████████████████████████████████| 4986/4986 [00:06<00:00, 81
7.87it/s]
```

In [37]:

```
data_avg=data
```

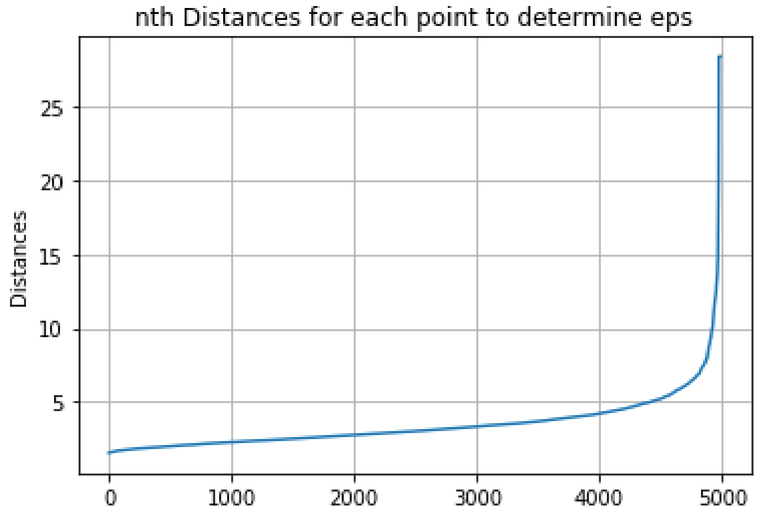
In [68]:

```
def db(data,min_point):
    distances = []
    for xi in tqdm(data):
        tmp = np.linalg.norm(data-xi,axis=1)
        tmp = np.sort(tmp)
        distances.append(tmp[min_point])
    distances = np.sort(distances)
    plt.plot(distances)
    plt.grid(True)
    plt.ylabel("Distances")
    plt.title("nth Distances for each point to determine eps")
    plt.show()
```

In [39]:

```
db(data=data_avg,min_point= 2*data_avg.shape[1])
```

```
100%|██████████████████████████████████████| 4986/4986 [00:14<00:00, 34
9.63it/s]
```



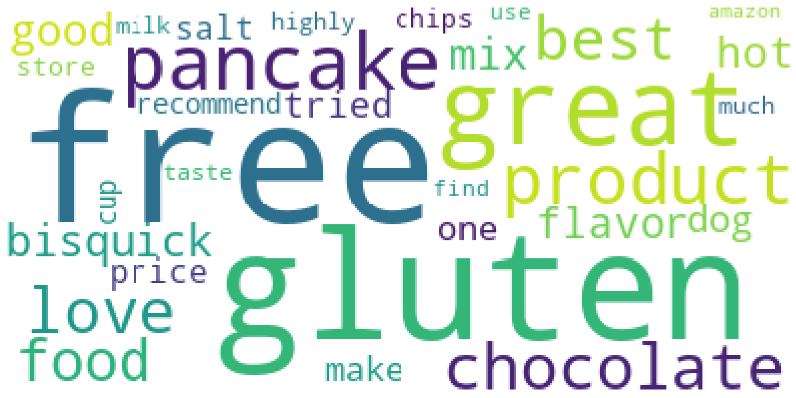
In [45]:
```
def cloud(cluster,i):
    wordcloud = WordCloud(collocations=False, background_color ='white'
,
                    min_font_size = 10).generate(str(list(cluster)))
    plt.figure(figsize = (6,6), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.title("Cluster "+str(i),size= 30,)
    plt.show()
```

In [46]:
```
from wordcloud import WordCloud
from sklearn.cluster import DBSCAN
epsilon = [5,6.5,7.5]
for ep in epsilon:
    print("Epsilon Value = ",ep)
    db = DBSCAN(eps=ep, min_samples=2*data_avg.shape[1]).fit(data_avg)
    labels = db.labels_ + 1 # to avoid -1 a for outliers
    clust = [ [] for i in range(len(set(labels))) ]  # this is the list
 of clusters
    for i in range(labels.shape[0]):
        clust[labels[i]].append(X[i])
```

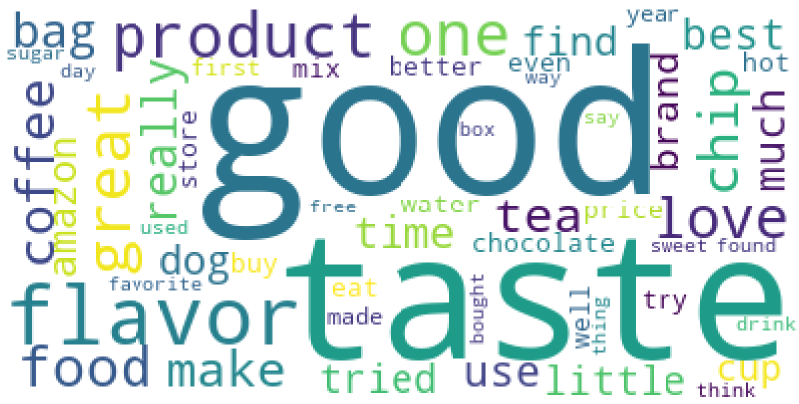
```
    i = 1
    for cl in clust:
        cloud(cl,i)
        i += 1
```

```
Epsilon Value =  5
```

Cluster 1



Cluster 2




```
Epsilon Value =  6.5
```

Cluster 1

Cluster 2

Epsilon Value = 7.5

Cluster 1

Cluster 2

1. epsilon-5 cluster1:pancake,great,gluten,chocolate. cluster2:bag,product,good,taste,flavor.
2. epsilon-6.5 cluster1:gluten,pancake,chocolate,free. cluster2:taste,good,one,love,product,chip.
3. epsilon-7.5 cluster1:bisquick,great,food,dog,product. cluster2:good,flavor,food,taste,great.

TF-IDF W2V

In [47]:

```
model = TfidfVectorizer()
model.fit(X)
# we are converting a dictionary with word as a key, and the idf as a v
alue
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 ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(sent_x): # 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/r
eview
    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
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
data_tf=tfidf_sent_vectors
```

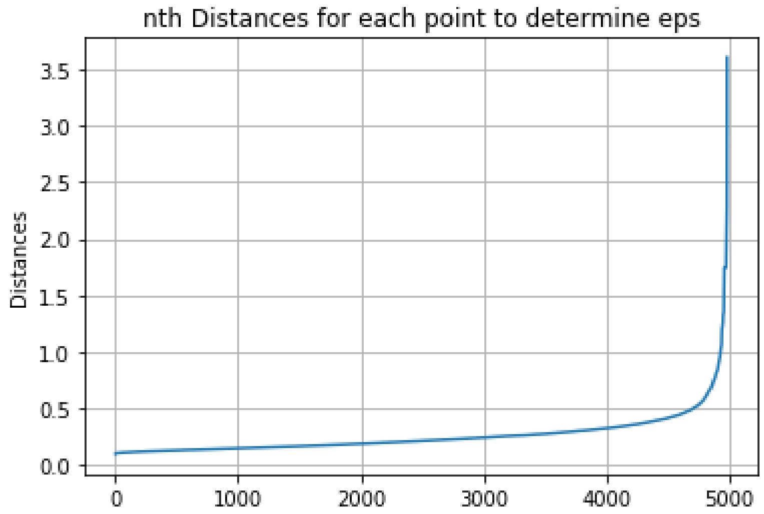
```
100%|██████████████████████████████| 4986/4986 [00:31<00:00, 15
8.95it/s]
```

```
0.95it/s]
```

In [69]:
```python
min_points=2*np.shape(data_tf)[1]
```

In [70]:
```python
db(data=data_tf,min_point=min_points)
```

```
100%|██████████████████████████████████████| 4986/4986 [00:37<00:00, 13
1.62it/s]
```



In [74]:
```python
w=len(set(labels))
w
```

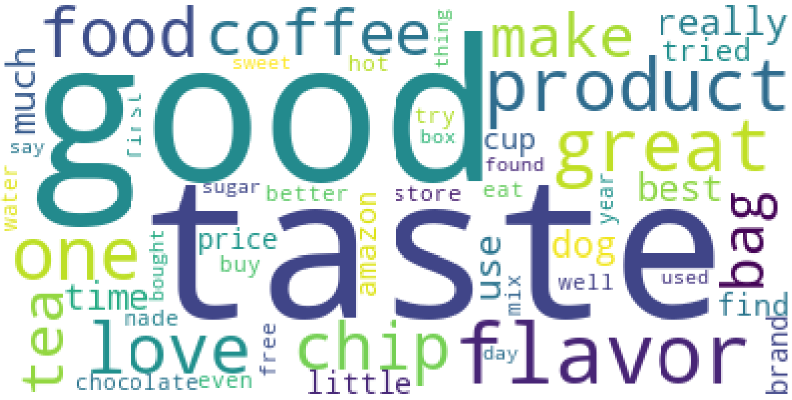
Out[74]: 1

In [57]:
```python
from wordcloud import WordCloud
from sklearn.cluster import DBSCAN
epsilon = [0.5,0.7,0.9]
for ep in epsilon:
    print("Epsilon Value = ",ep)
    db = DBSCAN(eps=ep, min_samples=2*data_avg.shape[1]).fit(data_avg)
    labels = db.labels_ + 1 # to avoid -1 a for outliers
    clust = [ [] for i in range(len(set(labels))) ]  # this is the list
 of clusters
```

```
    for i in range(labels.shape[0]):
        clust[labels[i]].append(X[i])
    i = 1
    for cl in clust:
        cloud(cl,i)
        i += 1
```

```
Epsilon Value =  0.5
```

Cluster 1




```
Epsilon Value =  0.7
```

Cluster 1


```
Epsilon Value =  0.9
```

Cluster 1

1. epsilon-0.5 cluster1:food,coffee,make,really,product,taste,chip.
2. epsilon-0.7 cluster1:food,good,flavor,product,tea,taste.
3. epsilon-0.9 cluster1:taste,good,product,love,chip.

In [ ]:
```
from tabulate import tabulate
print(tabulate ([['BOW',5],['TF-IDF',4],['AVG-W2V',4] , ['TFIDF-W2V',4
]],    headers=['Vectorizer', 'CLUSTER']))
```

CONCLUSION

In [3]:
```
from tabulate import tabulate
print(tabulate ([['AVG-W2V',5,1],['AVG-W2V',6.5,1],['AVG-W2V',7.5,1],[
'TFIDF-W2V',0.5,1],['TF-IDF-W2V',0.7,1],['TF-IDFW2V',0.9,1]], headers=[
'Vectorizer','epsilon' ,'CLUSTER']))
```

```
Vectorizer      epsilon    CLUSTER
------------  ---------  ---------
AVG-W2V             5            1
AVG-W2V             6.5          1
AVG-W2V             7.5          1
TFIDF-W2V           0.5          1
TF-IDF-W2V          0.7          1
TF-IDFW2V           0.9          1
```

[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 [3]:
```
# Please write all the code with proper documentation
```

[5.2.2] Wordclouds of clusters obtained after applying Agglomerative

Clustering on AVG W2V SET 3

In [3]:
```
# Please write all the code with proper documentation
```

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

In [3]:
```
# Please write all the code with proper documentation
```

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

In [3]:
```
# Please write all the code with proper documentation
```

[5.3] DBSCAN Clustering

[5.3.1] Applying DBSCAN on AVG W2V, SET 3

In [3]:
```
# Please write all the code with proper documentation
```

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

In [2]:
```
# Please write all the code with proper documentation
```

[5.3.3] Applying DBSCAN on TFIDF W2V, SET 4

In [3]:
```
# Please write all the code with proper documentation
```

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

In [3]:
```
# Please write all the code with proper documentation
```

[6] Conclusions

In [4]:
```
# Please compare all your models using Prettytable library.
# You can have 3 tables, one each for kmeans, agglomerative and dbscan
```