Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (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]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee649
1hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fww
w.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.
com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
#Libraries for getting date and time
from datetime import datetime
from pytz import timezone

india_tz=timezone('Asia/Kolkata')
```

In [0]:
```
%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

from sklearn.cluster import KMeans

from wordcloud import WordCloud

from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
```

In [0]:
```
# 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[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summar
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertise
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

In [0]:

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

In [0]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[0]:

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

In [0]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[0]:

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

In [0]:
```
display['COUNT(*)'].sum()
```

Out[0]: 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 [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Sun
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF

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

In [0]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[0]: (4986, 10)

In [0]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[0]: 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 [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summa
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste wi crunchy almond: inside

In [0]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [0]:
```
#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[0]:
```
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 [0]:
```
# 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/VICTOR-FLY-MAGNET-B
AIT-REFILL/dp/B00004RBDY<br /><br />The Victor M380 and M502 traps are unreal, of course -- total fly genocide. Prett
y stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The best thing was that there
were a lot of "brown" chips in the bsg (my favorite), so I bought some more through amazon and shared with family and
friends.  I am a little disappointed that there are not, so far, very many brown chips in these bags, but the flavor
is still very good.  I like them better than the yogurt and green onion flavor because they do not seem to be as salt
y, and the onion flavor is better.  If you haven't eaten Kettle chips before, I recommend that you try a bag before b
uying bulk.  They are thicker and crunchier than Lays but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the other wants crispy cookie
s.  Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look  before ordering.<br /><br />The
se are chocolate-oatmeal cookies.  If you don't like that combination, don't order this type of cookie.  I find the c
ombo quite nice, really.  The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconu
t-type consistency.  Now let's also remember that tastes differ; so, I've given my opinion.<br /><br />Then, these ar
e soft, chewy cookies -- as advertised.  They are not "crispy" cookies, or the blurb would say "crispy," rather than
"chewy."  I happen to like raw cookie dough; however, I don't see where these taste like raw cookie dough.  Both are
soft, however, so is this the confusion?  And, yes, they stick together.  Soft cookies tend to do that.  They aren't
individually wrapped, which would add to the cost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><
br />So, if you want something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you want a cookie that's soft, ch
ewy and tastes like a combination of chocolate and oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great coffee.  dcaf is very good as
well
==================================================
```

In [0]:

```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 and M502 traps a
re unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.
```

In [0]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
Why is this $[...] when the same product is available for $[...] here? />The Victor M380 and M502 traps are unreal, o
f course -- total fly genocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The best thing was that there
were a lot of "brown" chips in the bsg (my favorite), so I bought some more through amazon and shared with family and
friends.  I am a little disappointed that there are not, so far, very many brown chips in these bags, but the flavor
is still very good.  I like them better than the yogurt and green onion flavor because they do not seem to be as salt
y, and the onion flavor is better.  If you haven't eaten Kettle chips before, I recommend that you try a bag before b
uying bulk.  They are thicker and crunchier than Lays but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the other wants crispy cookie
s.  Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look  before ordering.These are choco
late-oatmeal cookies.  If you don't like that combination, don't order this type of cookie.  I find the combo quite n
ice, really.  The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-type consi
stency.  Now let's also remember that tastes differ; so, I've given my opinion.Then, these are soft, chewy cookies --
as advertised.  They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy."  I happen to lik
e raw cookie dough; however, I don't see where these taste like raw cookie dough.  Both are soft, however, so is this
the confusion?  And, yes, they stick together.  Soft cookies tend to do that.  They aren't individually wrapped, whic
h would add to the cost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.So, if you want something hard an
d crisp, I suggest Nabiso's Ginger Snaps.  If you want a cookie that's soft, chewy and tastes like a combination of c
hocolate and oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.  dcaf is very good as well
```

In [0]:

```
# 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 [0]:

```
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 cookie
s.  Hey, I am sorry; but these reviews do nobody any good beyond reminding us to look  before ordering.<br /><br />Th
ese are chocolate-oatmeal cookies.  If you do not like that combination, do not order this type of cookie.  I find th
e combo quite nice, really.  The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coc
onut-type consistency.  Now let is also remember that tastes differ; so, I have given my opinion.<br /><br />Then, th
ese are soft, chewy cookies -- as advertised.  They are not "crispy" cookies, or the blurb would say "crispy," rather
than "chewy."  I happen to like raw cookie dough; however, I do not see where these taste like raw cookie dough.  Bot
h are soft, however, so is this the confusion?  And, yes, they stick together.  Soft cookies tend to do that.  They a
re not individually wrapped, which would add to the cost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.
<br /><br />So, if you want something hard and crisp, I suggest Nabiso is Ginger Snaps.  If you want a cookie that is
soft, chewy and tastes like a combination of chocolate and oatmeal, give these a try.  I am here to place my second o
rder.
==================================================
```

In [0]:
```
#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  traps are unrea
l, of course -- total fly genocide. Pretty stinky, but only right nearby.
```

In [0]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

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

In [0]:
```
# 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're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'furthe
r',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mor
e',\
            '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', "were
n't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [0]:
```python
# 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 [00:01<00:00, 3137.37it/
s]
```

In [0]:
```python
preprocessed_reviews[1500]
```

Out[0]:
```
'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey sorry reviews nobody good beyon
d reminding us look ordering chocolate oatmeal cookies not like combination not order type cookie find combo quite ni
ce really oatmeal sort calms rich chocolate flavor gives cookie sort coconut type consistency let also remember taste
s differ given opinion soft chewy cookies advertised not crispy cookies blurb would say crispy rather chewy happen li
ke raw cookie dough however not see taste like raw cookie dough soft however confusion yes stick together soft cookie
s tend not individually wrapped would add cost oh yeah chocolate chip cookies tend somewhat sweet want something hard
crisp suggest nabiso ginger snaps want cookie soft chewy tastes like combination chocolate oatmeal give try place sec
ond order'
```

[3.2] Preprocessing Review Summary

In [0]:
```python
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [0]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdominal', 'abiding', 'abilit
y']
==================================================
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 [0]:
```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.feature_ex
traction.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 [0]:
```
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', 'absolutel
y', '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 [0]:
```
# 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 [0]:
```
# 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 ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.9946032166481018), ('excellent', 0.9
944332838058472), ('especially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted', 0.994047224521637),
('alternative', 0.9937226176261902), ('tasty', 0.9936816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.9992750883102417), ('de', 0.9992610
216140747), ('miss', 0.9992451071739197), ('melitta', 0.999218761920929), ('choice', 0.9992102384567261), ('america
n', 0.9991837739944458), ('beef', 0.9991780519485474), ('finish', 0.9991567134857178)]
```

In [0]:

```
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', 'eas
ily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot',
'fun', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstanding', 'window', 'everybod
y', 'asks', 'bought', 'made']
```

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

[4.4.1.1] Avg W2v

In [0]:
```
# 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 300 if you use g
oogle'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 [00:03<00:00, 1330.47it/
s]

4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:
```
# 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 [0]:
```
# 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
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████| 4986/4986 [00:20<00:00, 245.63it/
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. (https://stackoverflow.com/questions/12893492/choosing-eps-and-minpts-for-dbscan-r/48558030#48558030)
- 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.

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/preprocessed_reviews.pkl','rb') as f:
  preprocessed_reviews=pickle.load(f)
with open('/content/drive/My Drive/Applied AI assignments/list_of_scores.pkl','rb') as f:
  list_of_scores=pickle.load(f)
```

[5.1] K-Means Clustering

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

In [0]:
```
X=preprocessed_reviews[:50000]
```

In [0]:
```
#vectorizing using BoW
bow_vectorizer=CountVectorizer()
X_bow=bow_vectorizer.fit_transform(X)
```

In [0]:
```
start_time=datetime.now(india_tz)
print(start_time)
```

```
2019-03-14 10:47:48.361585+05:30
```

In [0]:
```
n_cluster_list=[2,3,5,7,9,10]
inertias=[]

for k in n_cluster_list:
  km=KMeans(n_clusters=k)
  km.fit(X_bow)
  inertias.append(km.inertia_)
```

In [0]:
```
end_time=datetime.now(india_tz)
print(end_time)
```

```
2019-03-14 11:38:51.756451+05:30
```

In [0]:
```
#saving the model
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_bow_model.pkl','wb') as f:
  pickle.dump(km,f)

#saving the inertias
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_bow_inertias.pkl','wb') as f:
  pickle.dump(inertias,f)
```

In [0]:
```
#plotting no of clusters vs inertia
plt.plot(n_cluster_list,inertias)
plt.xlabel('No of clusters (k)')
plt.ylabel('Inertia')
plt.title('No of clusters vs Inertia')
plt.show()
```

In [0]:
```
best_n_clusters=5
```

In [0]:
```
km=KMeans(n_clusters=best_n_clusters)
km.fit(X_bow)
```

Out[0]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_bow_trained_model.pkl','wb') as f:
  pickle.dump(km,f)
```

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_bow_trained_model.pkl','rb') as f:
  km=pickle.load(f)
```

In [0]:
```
cluster_centers=km.cluster_centers_
labels=km.labels_

with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_bow_cluster_centers.pkl','wb') as f:
  pickle.dump(cluster_centers,f)
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_bow_labels.pkl','wb') as f:
  pickle.dump(labels,f)
```

In [0]:
```
cluster_predictions=km.predict(X_bow)

with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_bow_cluster_predictions.pkl','wb') as f:
  pickle.dump(cluster_predictions,f)
```

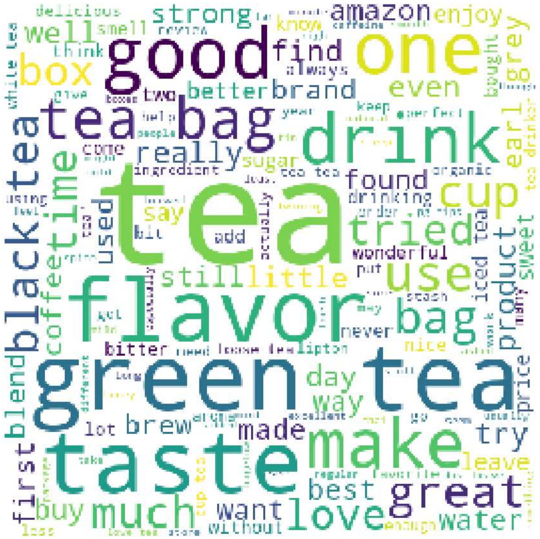
In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_bow_labels.pkl','rb') as f:
  lbls=pickle.load(f)
```

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_bow_cluster_predictions.pkl','rb') as f:
  cluster_predictions=pickle.load(f)
```

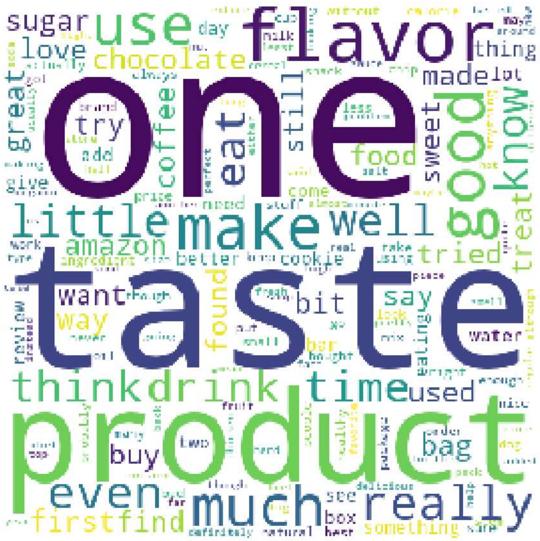
[5.1.2] Wordclouds of clusters obtained after applying k-means on BOW SET 1

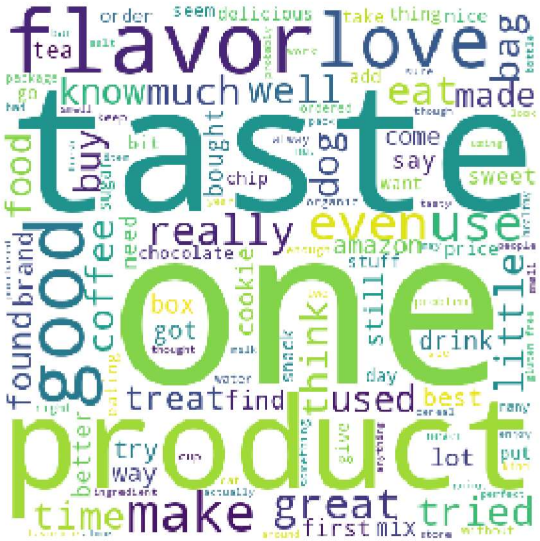
In [0]:
```
#function to generate word clouds
#Parameters
#X: text
#predictions: cluster predictions made on given data
#n_clusters: no of clusters


def generate_wordclouds(X,predictions,n_clusters):
  cluster_dict={}

  #creating a dictionary where keys are cluster names and values are text correspoding to cluster
  for i in range(n_clusters):
    cluster_dict['c'+str(i)]=[]

  idx=0
  for l in predictions:
    cluster_dict['c'+str(l)].append(X[idx])
    idx+=1

  wordcloud_objects=[]
  for c in list(cluster_dict.keys()):
    temp=WordCloud(width = 500, height = 500,background_color ='white',min_font_size = 10).generate(str(cluster_dict[c
]))
    wordcloud_objects.append(temp)

  n_cols=2

  if n_clusters%2==0:
    n_rows=n_clusters/2
  else:
    n_rows=(n_clusters+1)/2

  plt.figure(figsize=(12,12))

  for i in range(n_clusters):
    plt.subplot(n_rows,n_cols,i+1)
    # plot the WordCloud image
    plt.imshow(wordcloud_objects[i])
    plt.axis("off")
    plt.tight_layout(pad = 0)
```

In [19]:
```
generate_wordclouds(X,cluster_predictions,best_n_clusters)
```

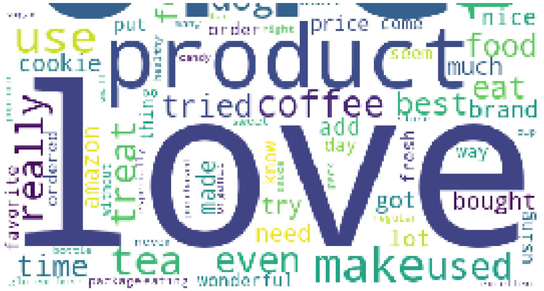

tea
green tea
flavor
taste
good
one
drink
bag
make
black tea
great
love
much
tried
use
cup
strong
amazon
enjoy
box
well
coffee
time
brand
even
found
really
little
still
bag
product
iced tea
try
water
buy
first
blend
brew
made
day
way
best
want
earl grey
sugar
find
better
say
price
wonderful
never
nice


food
dog
cat
dog food
cat food
eat
one
product
even
seem
good
great
love
price
chicken
day
brand
time
much
ingredient
healthy
well
really
bag
feeding
look
best
little
found
treat
know
diet
feed
first
better
two
lot
problem
dry food
need
year
give
find
tried
eating
never
amazon
still
many


one
taste
product
flavor
use
sugar
love
chocolate
good
make
little
well
think
drink
time
used
much
really
even
first
find
great
try
coffee
eat
still
sweet
food
know
treat
made
amazon
better
cookie
tried
want
way
bit
say
found
buy
bag
water


taste
one
product
flavor
love
know
much
well
eat
made
bag
good
coffee
food
buy
really
even
use
amazon
price
chocolate
sweet
little
drink
still
think
cookie
found
brand
treat
find
used
best
make
great
tried
time
first
mix
try
way
lot
box
got
dog
come
say
want


one
good
taste
flavor
great
delicious
little
still
perfect
drink
tasty
mix
bag
well
dog
better
give
box
first
year
think
chocolate
buy
keep
snack
work
say



Observations

1) **Cluster 1 (1st row 1st column)** contains words related to **tea products** (like green tea, black tea, tea bag). Observing other words like (taste, great,good,want), we can say that tea products has good reviews. The reviews in this cluster belong to the tea products
2) **Cluster 2 (1st row 2nd column)** contains words like dog,cat,food. So this cluster belongs to **animal food**. Seeing words like (seem, better), we may say that customers are just satisifed with these products
3) **Cluster 3 (2nd row 1st column)** contains words like (chocolate, cookie, coffee,sugar and sweet). So this cluster may contain reviews related to **confectionery**
4) **Cluster 4 (2nd row 2nd column)** contains words like (chocolate, cookie, coffee)
5) **Cluster 5 (3rd row 1st column)** contains words related to **beverages** (coffee and tea) and observing words like (delicious, great,good) we may say that beverages have more good reviews

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

In [0]:
```
X=preprocessed_reviews[:50000]
```

In [0]:
```
tfidf_vectorizer=TfidfVectorizer(ngram_range=(1,2),min_df=10)
X_tfidf=tfidf_vectorizer.fit_transform(X)
```

In [0]:
```
start_time=datetime.now(india_tz)
print(start_time)
```

```
2019-03-14 16:35:03.399231+05:30
```

In [0]:
```
n_cluster_list=[2,3,5,7,9,10]
inertias=[]

for k in n_cluster_list:
  km=KMeans(n_clusters=k)
  km.fit(X_tfidf)
  inertias.append(km.inertia_)
```

In [0]:
```
end_time=datetime.now(india_tz)
print(end_time)
```

```
2019-03-14 17:48:18.400271+05:30
```

In [0]:
```
#saving the inertias
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidf_inertias.pkl','wb') as f:
  pickle.dump(inertias,f)
```

In [0]:
```
plt.plot(n_cluster_list,inertias)
plt.xlabel('No of clusters (k)')
plt.ylabel('Inertia')
plt.title('No of clusters vs Inertia')
plt.show()
```

In [0]:
```
best_n_clusters=3
```

In [0]:
```
km=KMeans(n_clusters=best_n_clusters)
km.fit(X_tfidf)
```

Out[0]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidf_trained_model.pkl','wb') as
f:
  pickle.dump(km,f)
```

In [0]:
```
cluster_centers=km.cluster_centers_
labels=km.labels_

with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidf_cluster_centers.pkl','wb') a
s f:
  pickle.dump(cluster_centers,f)
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidf_labels.pkl','wb') as f:
  pickle.dump(labels,f)
```

In [0]:
```
cluster_predictions=km.predict(X_tfidf)

with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidf_cluster_predictions.pkl','w
b') as f:
  pickle.dump(cluster_predictions,f)
```

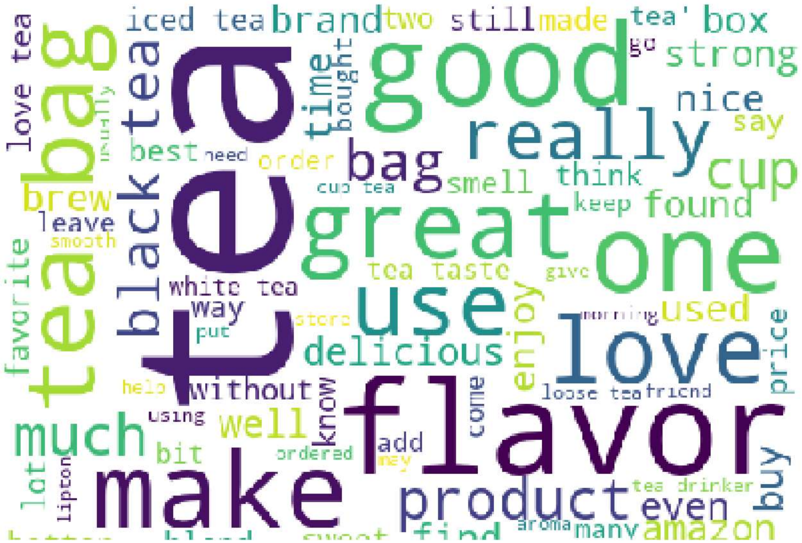
In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidf_cluster_predictions.pkl','r
b') as f:
  cluster_predictions=pickle.load(f)
```

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

In [24]:
```
generate_wordclouds(X,cluster_predictions,best_n_clusters)
```


food
dog
treat
one
cat
product
love
great
toy
good
time
even
well
give
eat
chew
found
flavor
little
really
cat food
make
much
know
think
dog food


one
taste
product
good
great
tried
coffee
well
used
made
really
even
little
make
love
flavor
use
delicious
time
found
cookie


tea
teabag
good
great
one
really
flavor
make
use
love
black tea
product
much
delicious
iced tea
brand
strong
nice
cup

Observations

1) **Cluster 1 (1st row 1st column)** contains words like dog,cat,food. So this cluster belongs to **animal food**. Seeing words like (good, great,love), we may say that customers like these products

2) **Cluster 2 (1st row 2nd column)** contains words like (chocoloate, cookie, coffee). So this cluster may contains reviews on **confectionery** items. Seeing words like (best, great, good), we may say that customers loved the products in this category.

3) **Cluster 3 (2nd row 1st column)** contains reviews related to **tea products** (green tea, tea bag, black tea)

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

In [0]:
```
# Please write all the code with proper documentation\
# List of tokenized sentences
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [0]:
```
#W2V model on train data
list_of_sentances=list_of_sentance[:50000]
X_w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50,workers=4)
```

In [0]:
```
#vectorizing X_train using Avg W2V
X_w2v_words = list(X_w2v_model.wv.vocab)
X_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentances): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use g
oogle'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 X_w2v_words:
            vec = X_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_sent_vectors.append(sent_vec)
```

```
100%|██████████| 50000/50000 [01:52<00:00, 444.37it/s]
```

In [0]:
```
start_time=datetime.now(india_tz)
print(start_time)
```

```
2019-03-14 23:30:47.891395+05:30
```

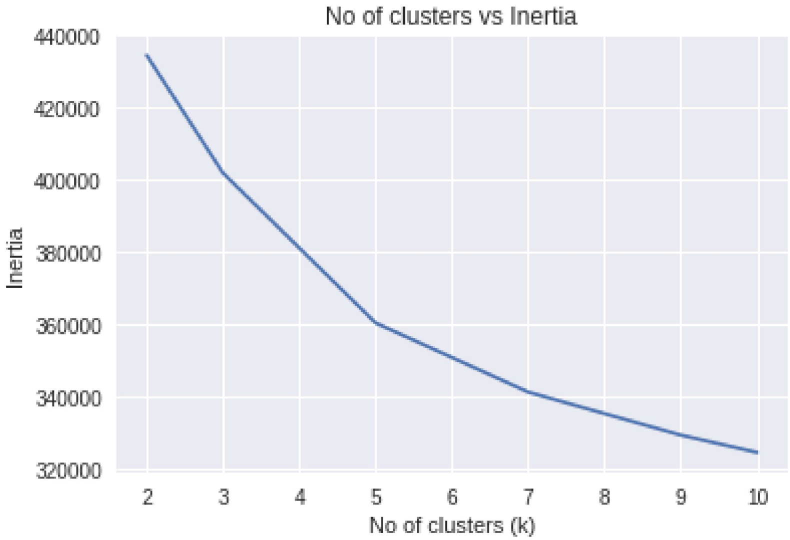
In [0]:
```
n_cluster_list=[2,3,5,7,9,10]
inertias=[]

for k in n_cluster_list:
  km=KMeans(n_clusters=k)
  km.fit(X_sent_vectors)
  inertias.append(km.inertia_)
```

In [0]:
```
end_time=datetime.now(india_tz)
print(end_time)
```

```
2019-03-14 23:31:26.423145+05:30
```

In [0]:
```
#saving the inertias
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_avgw2v_inertias.pkl','wb') as f:
  pickle.dump(inertias,f)
```

In [0]:
```
plt.plot(n_cluster_list,inertias)
plt.xlabel('No of clusters (k)')
plt.ylabel('Inertia')
plt.title('No of clusters vs Inertia')
plt.show()
```



In [0]:
```
best_n_clusters=5
```

In [0]:
```
km=KMeans(n_clusters=best_n_clusters)
km.fit(X_sent_vectors)
```

Out[0]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_avgw2v_trained_model.pkl','wb') as
f:
  pickle.dump(km,f)
```

In [0]:
```
cluster_centers=km.cluster_centers_
labels=km.labels_

with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_avgw2v_cluster_centers.pkl','wb')
as f:
  pickle.dump(cluster_centers,f)
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_avgw2v_labels.pkl','wb') as f:
  pickle.dump(labels,f)
```

In [0]:
```
cluster_predictions=km.predict(X_sent_vectors)

with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_avgw2v_cluster_predictions.pkl','w
b') as f:
  pickle.dump(cluster_predictions,f)
```

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

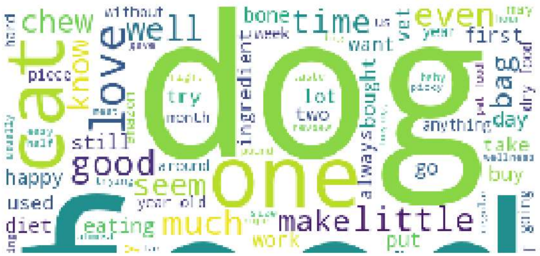
In [29]:
```
generate_wordclouds(X,cluster_predictions,best_n_clusters)
```

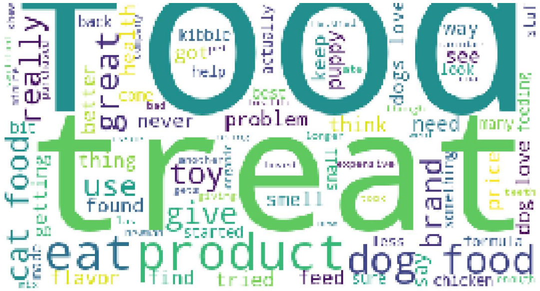

taste
coffee
flavor
drink
tea
good
make
green tea
one
use
tried
love
cup
strong
blend
time
great
really
product
first
best
tea bag
still
find
found
bag
much
buy
nice
delicious
well
water
brand
better
small
day
price
way
amazon
made
decaf
bitter
excellent
black tea
brew
sweet
lot
enjoy
aroma
order
try
say
want
keep
know
little
wonderful
iced tea
earl grey
drinking
bought
without
favorite
work
right
even
used
add
morning
leave


use
make
one
product
flavor
taste
good
great
used
add
made
need
eat
mix
time
well
love
best
rice
even
food
put
little
oil
water
first
think
sauce
noodle
recipe
tried
try
stuff
much
found
soup
using
delicious
brand
popcorn
really
pasta
cook
better
still
lot
alway
buy
come
bit
olive oil
hot
way
never
know
recipe


flavor
good
taste
one
love
product
really
make
chocolate
cookie
snack
great
sweet
chip
bar
delicious
eating
nut
calorie
much
think
sugar
cereal
nice
find
say
try
tasty
little
drink
bit
tried
know
well
use
buy
lot
want
best
found
eat
bag
better
thing
texture
made
even
still


product
amazon
one
bag
box
good
order
love
buy
great
even
price
time
taste
found
ordered
grocery store
use
really
flavor
bought
item
store
got
brand
cookie
little
make
package
pack
received
tea
much
shipping
purchase
tried
coffee
fresh
best
well
know
purchased
buying
company
used
still
year
first
problem
say
never
lot


dog
food
one
cat
love
chew
well
good
time
even
first
know
bag
still
seem
much
make
little
bone
want
try
lot
two
ingredient
always
bought
day
anything
go
take
buy
happy
used
diet
eating
work
put



Observations

1) **Cluster 1 (1st row 1st column)** contains words related to **beverages** tea and coffee. The words like (good, great, love) show that the most of the customers like the coffee and tea products
2) **Cluster 2 (1st row 2nd column)** contains words like (Noodles, Sauce, Soup, popcorn)
3) **Cluster 3 (2nd row 1st column)** contains words like (chocolate, cookie, coffee,sugar and sweet,chip, snack). So this cluster may contain reviews related to **confectionery**
4) **Cluster 4 (2nd row 2nd column)** contains words like (box, bag, order,package)
. So this cluster may contain reviews about the packing and shipping of products by Amazon
5) **Cluster 5 (3rd row 1st column)** contains words like dog,cat,food. So this cluster belongs to **animal food**. The words like (good,love,well,great) suggest that this cluster most of the customers are satisfied with these products

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

In [0]:
```
# Please write all the code with proper documentation
X=preprocessed_reviews[:50000]
```

In [0]:
```
model=TfidfVectorizer()
X_tfidfw2v=model.fit_transform(X)
```

In [0]:
```
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:
```
#vectorizing X_train using tfidf w2v
X_list_of_sents=[]
for sent in X:
  X_list_of_sents.append(sent.split())

#build w2v model for X_train
X_w2v_model=Word2Vec(X_list_of_sents,min_count=5,size=50,workers=4)
words=list(X_w2v_model.wv.vocab)

# 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

X_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_list_of_sents): # 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 words and word in tfidf_feat:
            vec = X_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
    X_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 50000/50000 [17:55<00:00, 46.51it/s]
```

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/X_tfidf_sent_vectors.pkl','wb') as f:
  pickle.dump(X_tfidf_sent_vectors,f)
```

In [0]:
```
start_time=datetime.now(india_tz)
print(start_time)
```

```
2019-03-14 23:50:09.605845+05:30
```

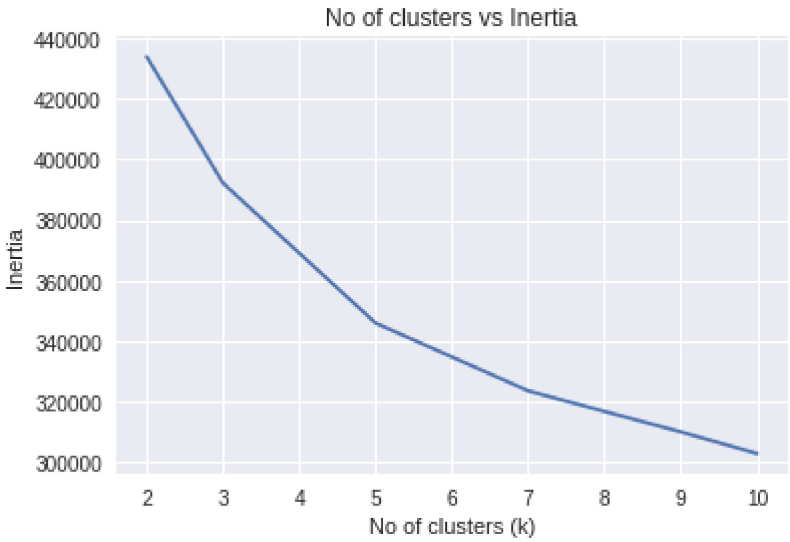
In [0]:
```
n_cluster_list=[2,3,5,7,9,10]
inertias=[]

for k in n_cluster_list:
  km=KMeans(n_clusters=k)
  km.fit(X_tfidf_sent_vectors)
  inertias.append(km.inertia_)
```

In [0]:
```
end_time=datetime.now(india_tz)
print(end_time)
```

```
2019-03-14 23:50:43.189038+05:30
```

In [0]:
```
#saving the inertias
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidfw2v_inertias.pkl','wb') as f:
  pickle.dump(inertias,f)
```

In [0]:
```
plt.plot(n_cluster_list,inertias)
plt.xlabel('No of clusters (k)')
plt.ylabel('Inertia')
plt.title('No of clusters vs Inertia')
plt.show()
```



In [0]:
```
best_n_clusters=5
```

In [0]:
```
km=KMeans(n_clusters=best_n_clusters)
km.fit(X_tfidf_sent_vectors)
```

Out[0]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

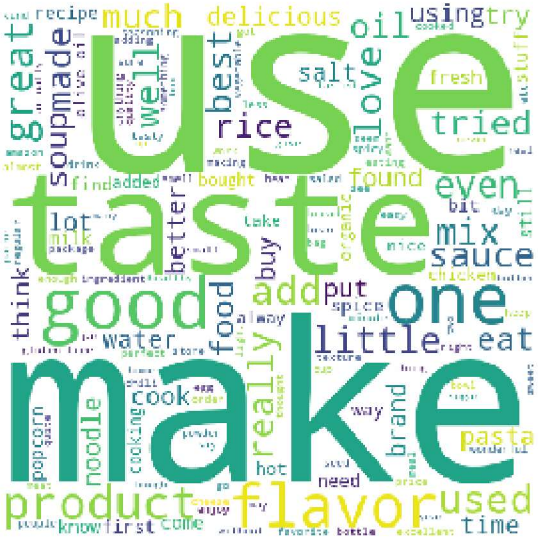
In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidfw2v_trained_model.pkl','wb')
as f:
  pickle.dump(km,f)
```

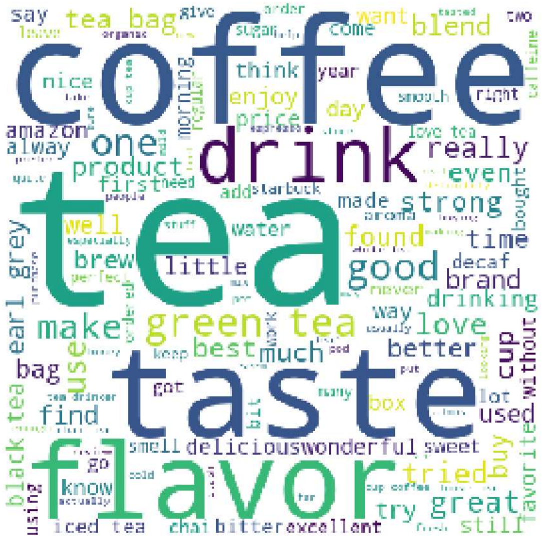
In [0]:
```
cluster_centers=km.cluster_centers_
labels=km.labels_

with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidfw2v_cluster_centers.pkl','wb'
) as f:
  pickle.dump(cluster_centers,f)
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidfw2v_labels.pkl','wb') as f:
  pickle.dump(labels,f)
```

In [0]:
```
cluster_predictions=km.predict(X_tfidf_sent_vectors)

with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/kMeans_tfidfw2v_cluster_predictions.pkl',
'wb') as f:
  pickle.dump(cluster_predictions,f)
```

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

In [0]:
```
generate_wordclouds(X,cluster_predictions,best_n_clusters)
```


use
taste
make
good
one
flavor
product
great
soupmade
best
love
rice
tried
oil
much
delicious
using
try
salt
fresh
found
even
mix
sauce
add
little
eat
really
food
noodle
popcorn
pasta
used
time
water
cook
better
lot
think
brand
buy
put
well
recipe


coffee
tea
drink
taste
flavor
tea bag
blend
green tea
good
really
strong
one
product
make
earl grey
black tea
love
better
find
brew
little
time
found
decaf
brand
drinking
delicious
wonderful
sweet
tried
great
cup
iced tea
bitter
excellent
nice
enjoy
price
morning
amazon
well
water
best
much


food
dog
treat
one
cat
product
dog food
cat food
eat
good
great
toy
chew
little
love
really
much
time
tried
give
well
seem
bag
even
make
brand
still
day
know


product
good
one
taste
make
bag
flavor
buy
amazon
time
box
love
order
great
little
price
use
tea
found
even
bought
coffee
best
really
store
package
grocery store
ordered
item
brand
know
think
got
well
still
much
first
tried
better


good
great
product
make
cookie
taste
chip
think
tried
best
much
sugar
delicious
cereal
snack
even

Observations
1) **Cluster 1 (1st row 1st column)** contains words noodles,soup,popcorn and sauce
2) **Cluster 2 (1st row 2nd column)** contains words related to **beverages** tea and coffee. The words like (good, great, love) show that the most of the customers like the coffee and tea products
3) **Cluster 3 (2nd row 1st column)** contains words like dog,cat,food. So this cluster belongs to **animal food**. The words like (good,love,well,great) suggest that this cluster most of the customers are satisfied with these products
4) **Cluster 4 (2nd row 2nd column)** contains words like (bag,box,tea,coffee). This cluster may contain reviews about taste and packing of tea and coffee related products
5) **Cluster 5 (3rd row 1st column)** contains words like (chocolate, cookie, coffee,sugar and sweet,chip, snack). So this cluster may contain reviews related to **confectionery**

[5.2] Agglomerative Clustering

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

In [0]:
```
# Please write all the code with proper documentation\
# list of tokenized sentences
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [0]:
```
#W2V model on train data
list_of_sentances=list_of_sentance[:5000]
X_w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50,workers=4)
```

In [0]:
```
#vectorizing X_train using Avg W2V
X_w2v_words = list(X_w2v_model.wv.vocab)
X_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentances): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use g
oogle'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 X_w2v_words:
            vec = X_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_sent_vectors.append(sent_vec)
```

```
100%|██████████| 5000/5000 [00:09<00:00, 530.28it/s]
```

In [0]:
```
start_time=datetime.now(india_tz)
print(start_time)
```

```
2019-03-16 09:30:05.474330+05:30
```

In [0]:
```
aggclustering=AgglomerativeClustering(n_clusters=2)
cluster_predictions=aggclustering.fit_predict(X_sent_vectors)
```

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/aggclustering_predictions_for_2_clusters.
pkl','wb') as f:
  pickle.dump(cluster_predictions,f)
```

In [0]:
```
aggclustering=AgglomerativeClustering(n_clusters=5)
cluster_predictions=aggclustering.fit_predict(X_sent_vectors)
```

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/aggclustering_predictions_for_5_clusters.
pkl','wb') as f:
  pickle.dump(cluster_predictions,f)
```

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

In [0]:
```
#wordcloud for 2 clusters
generate_wordclouds(list_of_sentances,cluster_predictions,2)
```

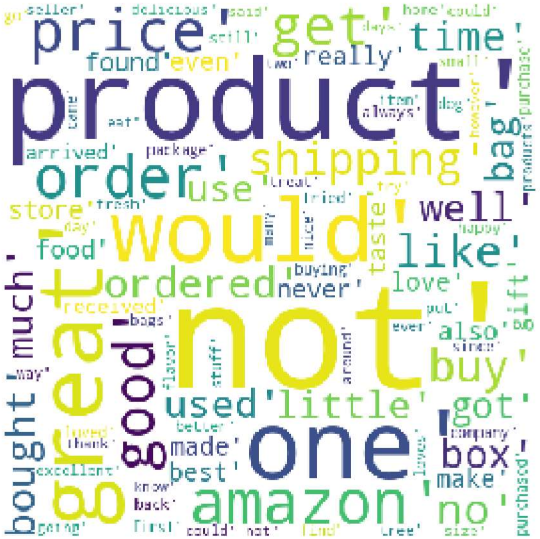
Observations

1) **Cluster 1 (1st row 1st column)** contains words noodles,soup,popcorn,chocolate,cookie and sauce,tea,coffee. This cluster may containe reviews related to both beverages and confectionery categories

. SInce 'not' word has more weightage, we may say that few customers are not satisfied with one of the category(beverages or confectionery)

2) **Cluster 2 (1st row 2nd column)** contains words like dog,cat,food. So this cluster may contain reviews related to animal food. Since 'not' word has more weightage, most of the customers are not satisfied with the products in this category

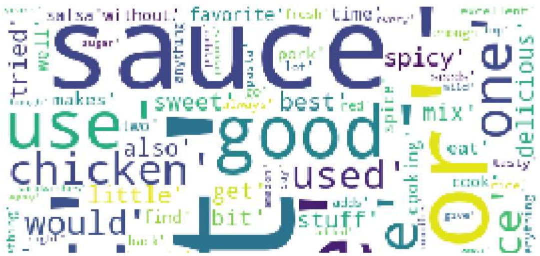
In [0]:
```
#wordcloud for 5 clusters
generate_wordclouds(list_of_sentances,cluster_predictions,5)
```

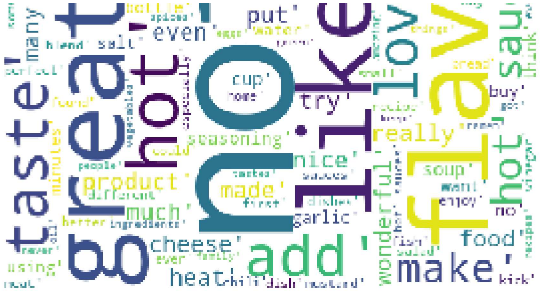

product'
not'
would'
great'
one'
price'
get'
time'
order'
shipping'
amazon'
good'
ordered'
like'
well'
bought'
much'
buy'
box'
used'
little'


food'
not'
dog'
dogs'
like'
treats'
treat'
one'
great'
good'
cat'
would'
little'
also'
get'
product'
healthy'
canidae'


taste'
like'
good'
not'
product'
chocolate'
flavor'
great'
one'
love'
best'
would'
use'
also'
really'
tea'
sugar'
much'
oil'
tried'


not'
coffee'
tea'
like'
taste'
drink'
flavor'
one'
great'
love'
good'
tried'
would'
time'
teas'
cup'
also'
really'
well'
get'


sauce'
good'
use'
chicken'
one'
delicious'
spicy'
would'
used'
sweet'
best'
mix'
salsa'
favorite'
tried'



Observations

1) **Cluster 1 (1st row 1st column)** contains words like shipping, order,product,amazon,box,bag suggest that this cluster may contain reviews related to the amazon packing and shipping of products. Seeing the weightage of 'not' we may say that a considerable amount of customers aren't satisifed with the amazon packing and shipping service
2) **Cluster 2 (1st row 2nd column)** contains words like dog,cat,food. So this cluster belongs to **animal food**. The words like (good,love,well,great) suggest that this cluster most of the customers are satisfied with these products
3) **Cluster 3 (2nd row 1st column)** contains sugar, tea,oil. These suggest that this cluster may contain reviews on **daily essential home products**.Seeing the weightage of 'not' we may say that a considerable amount of customers aren't satisifed with the products in this category
4) **Cluster 4 (2nd row 2nd column)** contains words like (tea,coffee,drink). This cluster may contain reviews about **beverages**.Seeing the weightage of 'not' we may say that a considerable amount of customers aren't satisifed with the products in this category
5) **Cluster 5 (3rd row 1st column)** contains words like sauce,cheese,chicken. Most customers found these good and few found them spicy

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

In [0]:
```
#considering only 5k points
X=preprocessed_reviews[:5000]
```

In [0]:
```
model=TfidfVectorizer()
X_tfidfw2v=model.fit_transform(X)
```

In [0]:
```
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:
```
#vectorizing X_train using tfidf w2v
X_list_of_sents=[]
for sent in X:
  X_list_of_sents.append(sent.split())

#build w2v model for X_train
X_w2v_model=Word2Vec(X_list_of_sents,min_count=5,size=50,workers=4)
words=list(X_w2v_model.wv.vocab)

# 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

X_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_list_of_sents): # 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 words and word in tfidf_feat:
            vec = X_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
    X_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 5000/5000 [00:41<00:00, 120.07it/s]
```

In [0]:
```
start_time=datetime.now(india_tz)
print(start_time)
```

```
2019-03-16 09:40:10.094013+05:30
```

In [0]:
```
aggclustering=AgglomerativeClustering(n_clusters=2)
cluster_predictions=aggclustering.fit_predict(X_tfidf_sent_vectors)
```

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/tfidf_aggclustering_predictions_for_2_clusters.pkl','wb') as f:
  pickle.dump(cluster_predictions,f)
```

In [0]:
```
aggclustering=AgglomerativeClustering(n_clusters=5)
cluster_predictions=aggclustering.fit_predict(X_sent_vectors)
```

In [0]:
```
with open('/content/drive/My Drive/Applied AI assignments/k-means clustering/tfidf_aggclustering_predictions_for_5_clusters.pkl','wb') as f:
  pickle.dump(cluster_predictions,f)
```

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

In [0]:
```
#wordcloud for 2 clusters
generate_wordclouds(X,cluster_predictions,2)
```

Observations
1) **Cluster 1 (1st row 1st column)** contains words noodles,soup,popcorn,chocolate,cookie and sauce,tea,coffee. This cluster may containe reviews related to beverages, groceries and confectionery categories

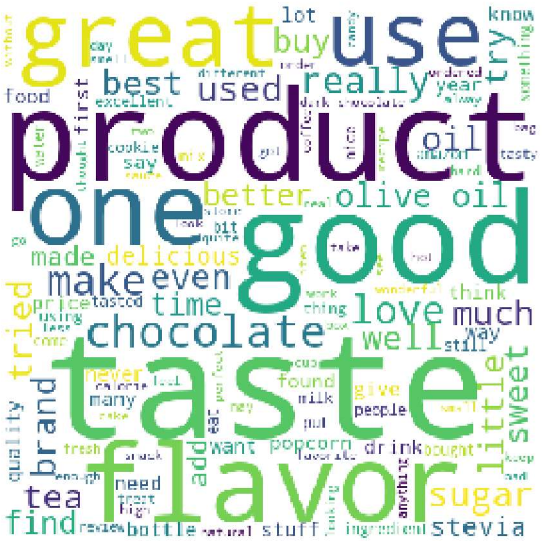
.

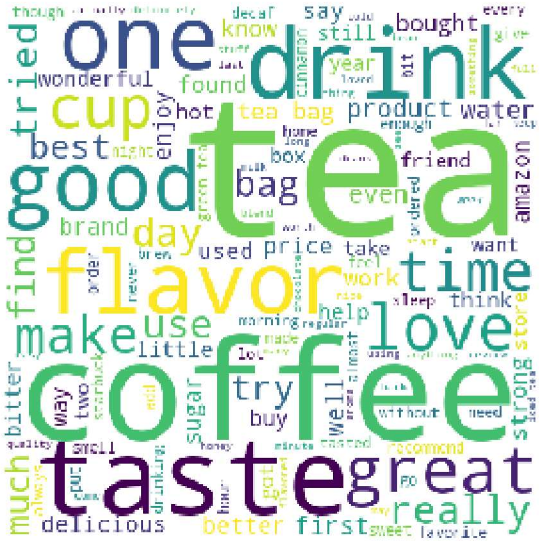
2) **Cluster 2 (1st row 2nd column)** contains words like dog,cat,food. So this cluster may contain reviews related to animal food.

In [0]:
```
#wordcloud for 5 clusters
generate_wordclouds(X,cluster_predictions,5)
```


product
great
one
order
good
love
amazon
buy
price
bag
time
shipping
ordered
gift
little
store
find
item
never
bought
made
arrived
much
make
package
found
received
taste
really
dog
day
box
got
use
used
small
well
best
year
treat
food
try
two


food
dog
one
treat
cat
love
product
know
well
toy
really
chew
great
little
seem
problem
day
eat
bag
smell
puppy
give
make
much
found
use
time
canidae
tried
brand
month
two
even
still
vet
healthy
good dog
first
size


product
taste
good
great
use
one
flavor
chocolate
olive oil
better
oil
best
used
really
buy
make
even
love
much
well
time
made
delicious
tried
brand
sweet
little
sugar
tea
find
stevia
try
know
lot


tea
coffee
taste
one
drink
good
cup
flavor
great
time
love
make
use
really
product
water
tea bag
best
bag
tried
find
day
brand
wonderful
found
bought
even
try
buy
well
strong
amazon
price
used
bitter
delicious
better
first
much


sauce
great
hot
make
one
delicious
food
try
spicy
stuff
tried
seasoning
favorite
wonderful
cooking
onion
mix
garlic
many

Observations
1) **Cluster 1 (1st row 1st column)** contains words like shipping, order,product,amazon,box,bag suggest that this cluster may contain reviews related to the amazon packing and shipping of products.
2) **Cluster 2 (1st row 2nd column)** contains words like dog,cat,food. So this cluster belongs to **animal food**. The words like (good,love,well,great) suggest that this cluster most of the customers are satisfied with these products
3) **Cluster 3 (2nd row 1st column)** contains sugar, tea,oil. These suggest that this cluster may contain reviews on **daily essential home products**.
4) **Cluster 4 (2nd row 2nd column)** contains words like (tea,coffee,drink). This cluster may contain reviews about **beverages**.
5) **Cluster 5 (3rd row 1st column)** contains words like sauce,cheese,chicken. Most customers found these good and few found them spicy

[5.3] DBSCAN Clustering

[5.3.1] Applying DBSCAN on AVG W2V, SET 3

In [0]:
```
X=preprocessed_reviews[:5000]
```

In [0]:
```
# Please write all the code with proper documentation\
# list of tokenized sentences
i=0
list_of_sentance=[]
for sentance in X:
    list_of_sentance.append(sentance.split())
```

In [0]:
```
#W2V model on train data
#list_of_sentances=list_of_sentance[:5000]
X_w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50,workers=4)
```

In [59]:
```
#vectorizing X_train using Avg W2V
X_w2v_words = list(X_w2v_model.wv.vocab)
X_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 300 if you use g
oogle'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 X_w2v_words:
            vec = X_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_sent_vectors.append(sent_vec)
```

```
100%|██████████| 5000/5000 [00:10<00:00, 471.40it/s]
```

In [0]:
```
#Rule of thumb minPts=2*dimension

minPts=2*50
```

In [0]:
```
from sklearn.neighbors import NearestNeighbors

neigh = NearestNeighbors(n_neighbors=minPts)

neigh.fit(X_sent_vectors)
```

Out[0]:
```
NearestNeighbors(algorithm='auto', leaf_size=30, metric='minkowski',
         metric_params=None, n_jobs=None, n_neighbors=100, p=2, radius=1.0)
```

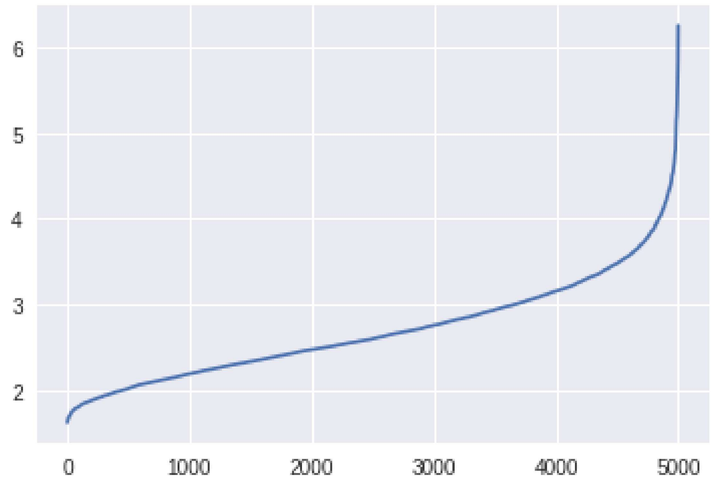
In [0]:
```
dist,ind=neigh.kneighbors(X_sent_vectors,n_neighbors=minPts,return_distance=True)
```

In [0]:
```
dist[:,99]
```

Out[0]:
```
array([2.53478147, 3.34438678, 1.98357611, ..., 2.19759227, 2.53953999,
       2.76274615])
```

In [0]:
```
distances=list(dist[:,99])
```

In [0]:
```
d_sorted=sorted(distances)
plt.plot(list(range(1,len(d_sorted)+1)),d_sorted)
plt.show()
```



In [0]:
```
eps=3.5
```

In [0]:
```
cluster_labels=DBSCAN(eps=3.5).fit(X_sent_vectors).labels_
```

In [0]:
```
n_clusters=np.unique(cluster_labels).shape[0]
```

In [70]:
```
n_clusters
```

Out[70]: 1

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

In [71]:
```
generate_wordclouds(X,cluster_labels,n_clusters)
```

Observations
This cluster contains reviews on all foods. The words like 'good','great' have more weightage which suggest that most customers liked these products

[5.3.3] Applying DBSCAN on TFIDF W2V, SET 4

In [0]:
```
#considering only 5k points
X=preprocessed_reviews[:5000]
```

In [0]:
```
model=TfidfVectorizer()
X_tfidfw2v=model.fit_transform(X)
```

In [0]:
```
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [75]:
```
#vectorizing X_train using tfidf w2v
X_list_of_sents=[]
for sent in X:
  X_list_of_sents.append(sent.split())

#build w2v model for X_train
X_w2v_model=Word2Vec(X_list_of_sents,min_count=5,size=50,workers=4)
words=list(X_w2v_model.wv.vocab)

# 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

X_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_list_of_sents): # 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 words and word in tfidf_feat:
            vec = X_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
    X_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 5000/5000 [00:43<00:00, 115.20it/s]
```

In [0]:
```
#Rule of thumb minPts=2*dimension

minPts=2*50
```

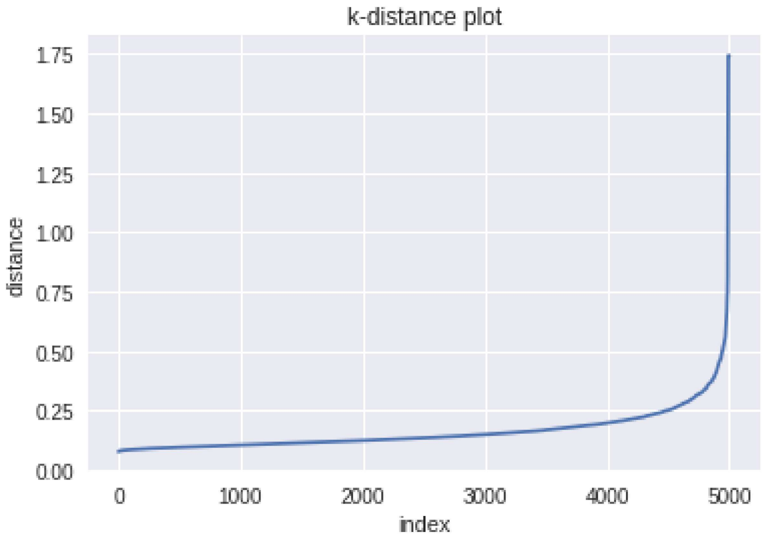
In [0]:
```
from sklearn.neighbors import NearestNeighbors

neigh = NearestNeighbors(n_neighbors=minPts)

neigh.fit(X_tfidf_sent_vectors)
```

Out[0]:
```
NearestNeighbors(algorithm='auto', leaf_size=30, metric='minkowski',
         metric_params=None, n_jobs=None, n_neighbors=100, p=2, radius=1.0)
```

In [0]:
```
#get the distance and index of given no of neighbors
dist,ind=neigh.kneighbors(X_tfidf_sent_vectors,n_neighbors=minPts,return_distance=True)
```

In [0]:
```
#taking the ditance of required neighbors (here 100th neighbors)
distances=list(dist[:,99])
```

In [0]:
```
#sorting the distances in ascending order and plotting against the indices (1 to len(distances))
d_sorted=sorted(distances)
plt.plot(list(range(1,len(d_sorted)+1)),d_sorted)
plt.xlabel('index')
plt.ylabel('distance')
plt.title('k-distance plot')
plt.show()
```



In [0]:
```
eps=0.4
```

In [0]:
```
cluster_labels=DBSCAN(eps=0.4).fit(X_tfidf_sent_vectors).labels_
```

In [0]:
```
n_clusters=np.unique(cluster_labels).shape[0]
```

In [86]:
```
n_clusters
```

Out[86]: 3

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

In [0]:
```
c1=[]
c2=[]
c3=[]

idx=0
for l in cluster_labels:
  if l==-1:
    c1.append(X[idx])
  if l==0:
    c2.append(X[idx])
  if l==1:
    c3.append(X[idx])
  idx+=1
```

In [106]:

```
wordcloud1=WordCloud(width = 500, height = 500,background_color ='white',min_font_size = 10).generate(str(c1))
plt.figure(figsize=(10,8))
plt.subplot(2,2,1)
# plot the WordCloud image
plt.imshow(wordcloud1)
plt.axis("off")
plt.tight_layout(pad = 0)

wordcloud2=WordCloud(width = 500, height = 500,background_color ='white',min_font_size = 10).generate(str(c2))
plt.subplot(2,2,2)
# plot the WordCloud image
plt.imshow(wordcloud2)
plt.axis("off")
plt.tight_layout(pad = 0)

#'c3' contains all empty strings
#wordcloud3=WordCloud(width = 500, height = 500,background_color ='white',min_font_size = 10).generate(str(c3))
#plt.subplot(2,2,3)
# plot the WordCloud image
#plt.imshow(wordcloud3)
#plt.axis("off")
#plt.tight_layout(pad = 0)
```

Observations

1) **Cluster 1** contains words related to sauce products. The weightage of 'recommended' word shows that most customers recommended sauce products to other customers
2) **Cluster 2** contains reviews related to all other foods like beverages, confectionaries, animal food </br>

[6] Conclusions

In [0]:
```
# Please compare all your models using Prettytable library.
# You can have 3 tables, one each for kmeans, agllomerative and dbscan
```

In [111]:
```
from prettytable import PrettyTable

print('\nk-means clustering\n')
x = PrettyTable()
x.field_names = ["Vectorization", "Best no of clusters"]
x.add_row(["BoW", 5])
x.add_row(["Tfidf", 3])
x.add_row(["AvgW2V", 5])
x.add_row(["TfidfW2v", 5])
print(x)

print('################################')

print('\nAgglomerative clustering\n')
print('Agglomerative clustering resulted in similar clusters for both AvgW2v and TfidfW2V vectorization')


print('\n\n################################')
print('\nDBSCAN\n')
x = PrettyTable()
x.field_names = ["Vectorization", "No of clusters"]
x.add_row(["AvgW2V", 1])
x.add_row(["TfidfW2v", 2])
print(x)
print('\nWith AVGW2V, DBSCAN clustering resulted in only one cluster which is bad')
```

k-means clustering

Vectorization	Best no of clusters
BoW	5
Tfidf	3
AvgW2V	5
TfidfW2v	5

################################

Agglomerative clustering

Agglomerative clustering resulted in similar clusters for both AvgW2v and TfidfW2V vectorization

################################

DBSCAN

Vectorization	No of clusters
AvgW2V	1
TfidfW2v	2

With AVGW2V, DBSCAN clustering resulted in only one cluster which is bad