Amazon Fine Food Reviews Analysis

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

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

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

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os
```

In [2]:

```python
# 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 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 50000""", 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 (50000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2

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

In [5]:

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

Out[5]:

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

In [6]:
```
display['COUNT(*)'].sum()
```
Out[6]: 393063

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
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 [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

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

Out[9]:
```
(46072, 10)
```

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]:
```
92.144
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
```

```
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

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

In [12]:

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

In [13]:

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

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

```
(46071, 10)
```

Out[13]:

```
1    38479
0     7592
```

```
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [14]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
```

```
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp
or cobbler. home made is better, but a heck of a lot more work. this is
great to have on hand for last minute dessert needs where you really wa
nt to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==================================================
For those of you wanting a high-quality, yet affordable green tea, you
should definitely give this one a try. Let me first start by saying tha
t everyone is looking for something different for their ideal tea, and
I will attempt to briefly highlight what makes this tea attractive to a
wide range of tea drinkers (whether you are a beginner or long-time tea
enthusiast).  I have gone through over 12 boxes of this tea myself, and
highly recommend it for the following reasons:<br /><br />-Quality:  Fi
rst, this tea offers a smooth quality without any harsh or bitter after
tones, which often turns people off from many green teas.  I've found m
y ideal brewing time to be between 3-5 minutes, giving you a light but
flavorful cup of tea.  However, if you get distracted or forget about y
our tea and leave it brewing for 20+ minutes like I sometimes do, the q
uality of this tea is such that you still get a smooth but deeper flavo
r without the bad after taste.  The leaves themselves are whole leaves
(not powdered stems, branches, etc commonly found in other brands), and
the high-quality nylon bags also include chunks of tropical fruit and o
ther discernible ingredients.  This isn't your standard cheap paper bag
with a mix of unknown ingredients that have been ground down to a fine
```

```
powder, leaving you to wonder what it is you are actually drinking.<br
/><br />-Taste:  This tea offers notes of real pineapple and other hint
s of tropical fruits, yet isn't sweet or artificially flavored.  You ha
ve the foundation of a high-quality young hyson green tea for those tru
e "tea flavor" lovers, yet the subtle hints of fruit make this a truly
unique tea that I believe most will enjoy.  If you want it sweet, you c
an add sugar, splenda, etc but this really is not necessary as this tea
offers an inherent warmth of flavor through it's ingredients.<br /><br
/>-Price:  This tea offers an excellent product at an exceptional price
(especially when purchased at the prices Amazon offers).  Compared to o
ther brands which I believe to be of similar quality (Mighty Leaf, Rish
i, Two Leaves, etc.), Revolution offers a superior product at an outsta
nding price.  I have been purchasing this through Amazon for less per b
ox than I would be paying at my local grocery store for Lipton, etc.<br
/><br />Overall, this is a wonderful tea that is comparable, and even b
etter than, other teas that are priced much higher.  It offers a well-b
alanced cup of green tea that I believe many will enjoy.  In terms of t
aste, quality, and price, I would argue you won't find a better combina
tion that that offered by Revolution's Tropical Green Tea.
==================================================
```

In [15]:

```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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [16]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how
-to-remove-all-tags-from-an-element
```

```python
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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp
or cobbler. home made is better, but a heck of a lot more work. this is
great to have on hand for last minute dessert needs where you really wa
nt to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==================================================
For those of you wanting a high-quality, yet affordable green tea, you
should definitely give this one a try. Let me first start by saying tha
t everyone is looking for something different for their ideal tea, and
```

```
I will attempt to briefly highlight what makes this tea attractive to a
wide range of tea drinkers (whether you are a beginner or long-time tea
enthusiast).  I have gone through over 12 boxes of this tea myself, and
highly recommend it for the following reasons:-Quality:  First, this te
a offers a smooth quality without any harsh or bitter after tones, whic
h often turns people off from many green teas.  I've found my ideal bre
wing time to be between 3-5 minutes, giving you a light but flavorful c
up of tea.  However, if you get distracted or forget about your tea and
leave it brewing for 20+ minutes like I sometimes do, the quality of th
is tea is such that you still get a smooth but deeper flavor without th
e bad after taste.  The leaves themselves are whole leaves (not powdere
d stems, branches, etc commonly found in other brands), and the high-qu
ality nylon bags also include chunks of tropical fruit and other discer
nible ingredients.  This isn't your standard cheap paper bag with a mix
of unknown ingredients that have been ground down to a fine powder, lea
ving you to wonder what it is you are actually drinking.-Taste:  This t
ea offers notes of real pineapple and other hints of tropical fruits, y
et isn't sweet or artificially flavored.  You have the foundation of a
high-quality young hyson green tea for those true "tea flavor" lovers,
yet the subtle hints of fruit make this a truly unique tea that I belie
ve most will enjoy.  If you want it sweet, you can add sugar, splenda,
etc but this really is not necessary as this tea offers an inherent war
mth of flavor through it's ingredients.-Price:  This tea offers an exce
llent product at an exceptional price (especially when purchased at the
prices Amazon offers).  Compared to other brands which I believe to be
of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), Revolution o
ffers a superior product at an outstanding price.  I have been purchasi
ng this through Amazon for less per box than I would be paying at my lo
cal grocery store for Lipton, etc.Overall, this is a wonderful tea that
is comparable, and even better than, other teas that are priced much hi
gher.  It offers a well-balanced cup of green tea that I believe many w
ill enjoy.  In terms of taste, quality, and price, I would argue you wo
n't find a better combination that that offered by Revolution's Tropica
l Green Tea.
```

In [17]:

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

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

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

In [18]:

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

```
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==================================================
```

In [19]:

```python
#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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [20]:

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

```
print(sent_1500)
```

```
Great flavor low in calories high in nutrients high in protein Usually
protein powders are high priced and high in calories this one is a grea
t bargain and tastes great I highly recommend for the lady gym rats pro
bably not macho enough for guys since it is soy based
```

In [21]:

```
# 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 [22]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
() not in stopwords)
    if len(sentance)==0:
        continue
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 46071/46071 [01:50<00:00, 417.99it/s]
```

In [23]:
```
preprocessed_reviews[717]
```

Out[23]:
```
'favorite starbucks roast dont go local starbucks purchase price go don
t pay shipping handeling stuff plus grind'
```

[3.2] Preprocessing Review Summary

In [24]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

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

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

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaa
aaaaaaaaaaaa', 'aaaaaaahhhhhh', 'aaaaaawwwwwwwwww', 'aaaaah', 'aaaand']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (45949, 39364)
the number of unique words  39364
```

[4.2] Bi-Grams and n-Grams.

In [26]:

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

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

# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
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_s
hape())
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  (45949, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [27]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_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', 'a
ble buy', 'able chew', 'able drink', 'able eat', 'able enjoy', 'able fe
ed', 'able figure', 'able find']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (45949, 27311)
the number of unique words including both unigrams and bigrams  27311
```

[4.4] Word2Vec

```
In [28]: # 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 [29]: # 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 val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# 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 ")
```

```
C:\Users\MANISH\Anaconda3\lib\site-packages\gensim\models\base_any2vec.py:743: UserWarning: C extension not loaded, training will be slow. Install a C compiler and reinstall gensim for fast training.
  "C extension not loaded, training will be slow. "
```

```
[('awesome', 0.8203691244125366), ('fantastic', 0.8191199898719788), ('terrific', 0.8103197813034058), ('good', 0.7949531078338623), ('excellent', 0.7681958675384521), ('wonderful', 0.762433648109436), ('amazing', 0.744584858417511), ('perfect', 0.7357608675956726), ('decent', 0.7111800312995911), ('nice', 0.6555903553962708)]
==================================================
[('nastiest', 0.7258411645889282), ('best', 0.7238538265228271), ('greatest', 0.7027143239974976), ('experienced', 0.6725594997406006), ('ive', 0.670496940612793), ('disgusting', 0.6523109674453735), ('awful', 0.639106035232544), ('closest', 0.6327093839645386), ('terrible', 0.6066190004348755), ('horrible', 0.6002564430236816)]
```

In [30]:

```
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  12798
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buying', 'anymore', 'hard', 'find', 'products', 'made', 'usa', 'one', 'isnt', 'bad', 'good', 'take', 'chances', 'till', 'know', 'going', 'imports', 'love', 'saw', 'pet', 'store', 'tag', 'attached', 'regarding', 'satisfied', 'safe', 'available', 'victor', 'traps', 'unreal', 'course', 'total', 'fly', 'pretty', 'stinky', 'right', 'nearby', 'used', 'bait', 'seasons', 'ca', 'not', 'beat', 'great']
```

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

[4.4.1.1] Avg W2v

In [31]:

```
# 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 google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 45949/45949 [09:48<00:00, 78.09it/s]
```

```
45949
50
```

[4.4.1.2] TFIDF weighted W2v

In [32]:

```
# 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 v
```

```
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [33]:

```
# 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(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/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
```

```
100%|██████████| 45949/45949 [1:41:52<00:00,  7.52it/s]
```

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

1. **Apply K-means Clustering on these feature sets:**

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

2. **Apply Agglomerative Clustering on these feature sets:**

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

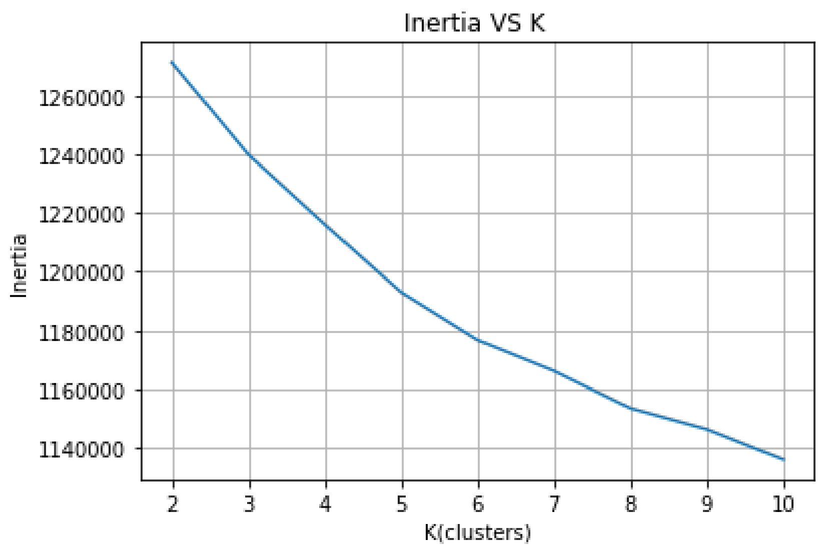
3. **Apply DBSCAN Clustering on these feature sets:**

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

[5.1] K-Means Clustering

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

In [34]:
```
# Please write all the code with proper documentation
from sklearn.cluster import KMeans
count_vect = CountVectorizer(min_df = 1000)
X_train=count_vect.fit_transform(preprocessed_reviews)
k=[2,3,4,5,6,7,8,9,10]
inertia=[]
for i in k:
    model=KMeans(n_clusters=i, n_jobs=-1)
    model.fit(X_train)
    inertia.append(model.inertia_)
#finding best k using elbow method
plt.plot(k, inertia)
plt.xlabel('K(clusters)')
plt.ylabel('Inertia')
plt.title('Inertia VS K ')
plt.grid()
plt.show()
```



In [35]:
```python
#we can see that at k=5 there is a point of inflection
k=5
model=KMeans(n_clusters=5, n_jobs=-1)
model.fit(X_train)
```

Out[35]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=5, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [36]:
```python
cluster1,cluster2,cluster3,cluster4,cluster5=[],[],[],[],[]
for i in range(model.labels_.shape[0]):
    if model.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    elif model.labels_[i] == 1:
        cluster2.append(preprocessed_reviews[i])
    elif model.labels_[i] == 2:
        cluster3.append(preprocessed_reviews[i])
    elif model.labels_[i] == 3:
        cluster4.append(preprocessed_reviews[i])
    else:
        cluster5.append(preprocessed_reviews[i])
```

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

In [37]:
```python
#for cluster 1
data=''
for i in cluster1:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [38]:
```
#for cluster 2
data=''
for i in cluster2:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

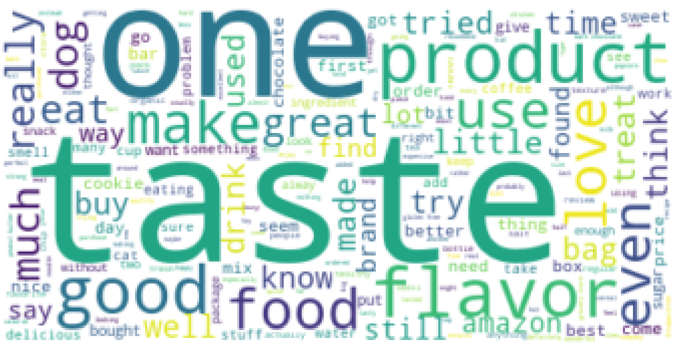
In [39]:
```
#for cluster 3
```

```
data=''
for i in cluster3:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [40]:
```
#for cluster 4
data=''
for i in cluster4:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [41]:

```
#for cluster 5
data=''
for i in cluster5:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



Top terms per clusters

In [42]:
```
#https://stackoverflow.com/questions/47452119/kmean-clustering-top-term
s-in-cluster
print("Top 5 terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = count_vect.get_feature_names()
for i in range(k):
    print("Cluster %d:" % (i+1), end='')
    for ind in order_centroids[i, :5]:
        print(' %s' % terms[ind], end='')
        print()
    print('-'*50)
```

```
Top 5 terms per cluster:
Cluster 1: coffee
 not
 cup
 like
 taste
--------------------------------------------------
Cluster 2: not
 great
 good
 like
 love
--------------------------------------------------
Cluster 3: not
 like
 food
 would
 one
--------------------------------------------------
Cluster 4: tea
 not
 like
 green
 good
--------------------------------------------------
Cluster 5: not
```

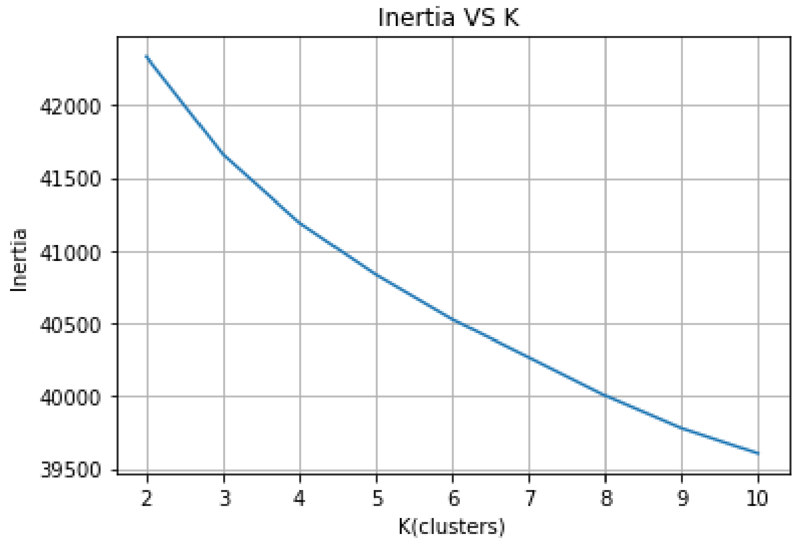
```
cluster 3: not

 like
 taste
 would
 good
--------------------------------------------------
```

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

In [43]:
```
# Please write all the code with proper documentation
from sklearn.cluster import KMeans
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=1000)
X_train=tf_idf_vect.fit_transform(preprocessed_reviews)
X_train.shape
```

Out[43]:
```
(45949, 299)
```

In [44]:
```
k=[2,3,4,5,6,7,8,9,10]
inertia=[]
for i in k:
    model=KMeans(n_clusters=i, n_jobs=-1)
    model.fit(X_train)
    inertia.append(model.inertia_)
#finding best k using elbow method
plt.plot(k, inertia)
plt.xlabel('K(clusters)')
plt.ylabel('Inertia')
plt.title('Inertia VS K ')
plt.grid()
plt.show()
```



In [45]:
```
#we can see that at k=3 there is a point of inflection
model=KMeans(n_clusters=3, n_jobs=-1)
model.fit(X_train)
```

Out[45]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [46]:
```
cluster1,cluster2,cluster3=[],[],[]
for i in range(model.labels_.shape[0]):
    if model.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    elif model.labels_[i] == 1:
        cluster2.append(preprocessed_reviews[i])
    else:
        cluster3.append(preprocessed_reviews[i])
```

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

In [47]:
```python
#for cluster 1
data=''
for i in cluster1:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [48]:
```python
#for cluster 2
data=''
for i in cluster2:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [49]:
```
#for cluster 3
data=''
for i in cluster3:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Top features from each clusters

In [50]:
```
#https://stackoverflow.com/questions/47452119/kmean-clustering-top-term
s-in-cluster
print("Top 5 terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = tf_idf_vect.get_feature_names()
for i in range(3):
    print("Cluster %d:" % (i+1), end='')
    for ind in order_centroids[i, :5]:
        print(' %s' % terms[ind], end='')
        print()
    print('-'*50)
```

```
Top 5 terms per cluster:
Cluster 1: coffee
 cup
 not
 like
 flavor
--------------------------------------------------
Cluster 2: not
 like
 great
 good
 product
--------------------------------------------------
Cluster 3: tea
 not
 green
 like
 drink
--------------------------------------------------
```

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

In [51]:
```
list_of_sentance_train=[]
```

```python
for sentance in preprocessed_reviews:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    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
    sent_vectors_train.append(sent_vec)
print(len(sent_vectors_train))
print(len(sent_vectors_train[0]))
```

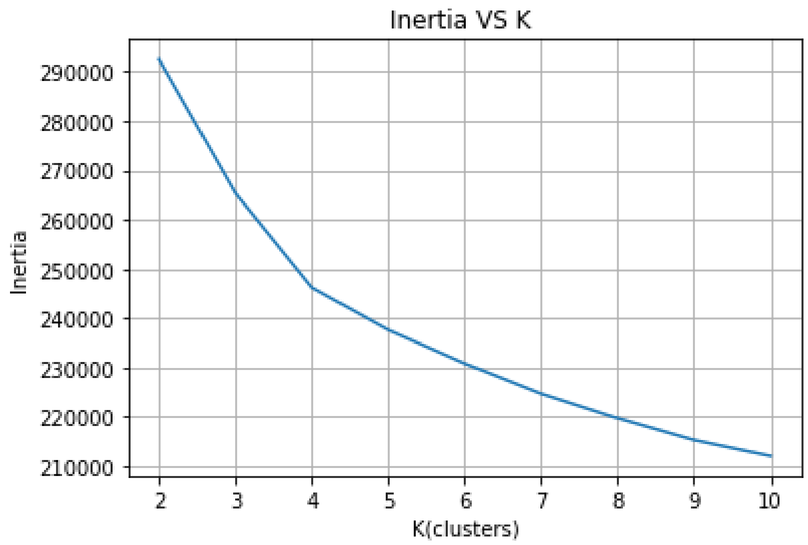
```
C:\Users\MANISH\Anaconda3\lib\site-packages\gensim\models\base_any2vec.
py:743: UserWarning: C extension not loaded, training will be slow. Ins
tall a C compiler and reinstall gensim for fast training.
  "C extension not loaded, training will be slow. "
100%|██████████| 45949/45949 [09:23<00:00, 81.50it/s]
```

```
45949
50
```

In [52]:

```python
k=[2,3,4,5,6,7,8,9,10]
X_train=sent_vectors_train
inertia=[]
for i in k:
    model=KMeans(n_clusters=i, n_jobs=-1)
    model.fit(X_train)
    inertia.append(model.inertia_)
#finding best k using elbow method
plt.plot(k, inertia)
plt.xlabel('K(clusters)')
plt.ylabel('Inertia')
```

```
plt.title('Inertia VS K ')
plt.grid()
plt.show()
```



In [53]:
```
#we can see that at k=4 there is a point of inflection
model=KMeans(n_clusters=4, n_jobs=-1)
model.fit(X_train)
```

Out[53]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [54]:
```
cluster1,cluster2,cluster3,cluster4=[],[],[],[]
for i in range(model.labels_.shape[0]):
    if model.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    elif model.labels_[i] == 1:
        cluster2.append(preprocessed_reviews[i])
    elif model.labels_[i] == 2:
        cluster3.append(preprocessed_reviews[i])
    else:
```

```
        cluster4.append(preprocessed_reviews[i])
```

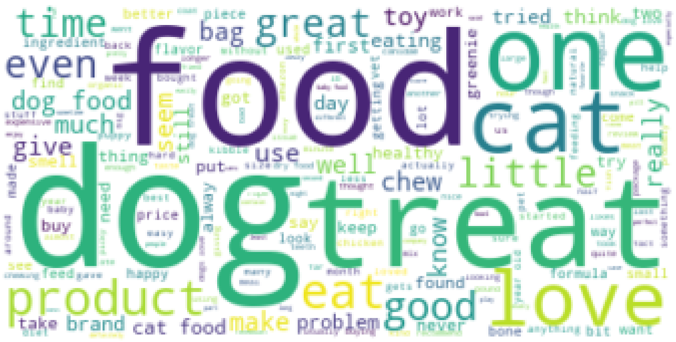
[5.1.6] Wordclouds of clusters obtained after applying k-means on AVG W2V SET 3

In [55]:
```
#for cluster 1
data=''
for i in cluster1:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

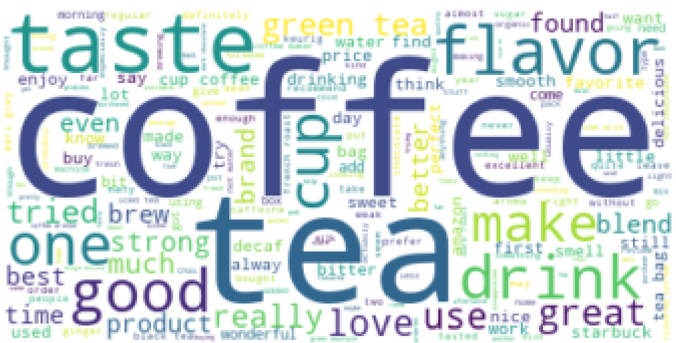
In [56]:
```
#for cluster 2
data=''
for i in cluster2:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)
```

```
# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```


```
In [57]: #for cluster 3
data=''
for i in cluster3:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```




```
In [58]: #for cluster 4
         data=''
         for i in cluster4:
             data+=str(i)
         from wordcloud import WordCloud
         wordcloud = WordCloud(background_color="white").generate(data)

         # Display the generated image:
         plt.imshow(wordcloud, interpolation='bilinear')
         plt.axis("off")
         plt.show()
```



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

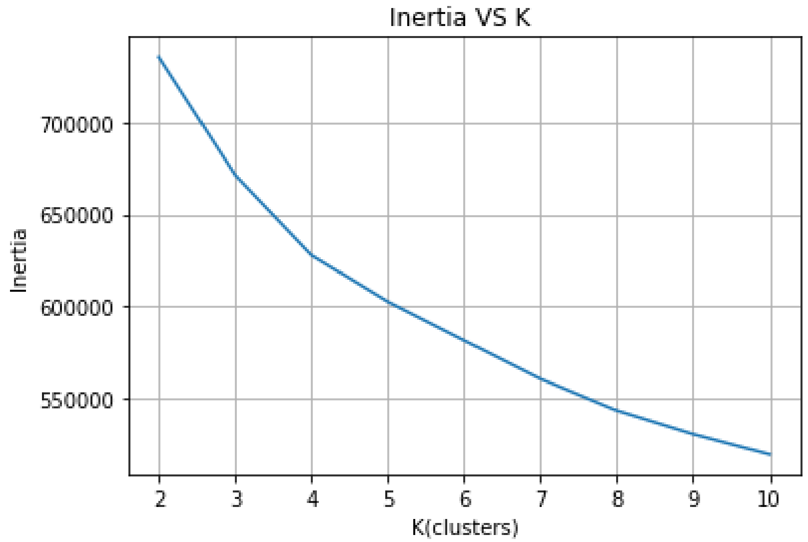
In [59]:
```
list_of_sentance_train=[]
for sentance in preprocessed_reviews:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10, max_features
=500)

tf_idf_matrix=tf_idf_vect.fit_transform(preprocessed_reviews)


tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect
.idf_)))

#for train data

tfidf_sent_vectors_train = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_train.append(sent_vec)
    row += 1
```

```
C:\Users\MANISH\Anaconda3\lib\site-packages\gensim\models\base_any2vec.
py:743: UserWarning: C extension not loaded, training will be slow. Ins
tall a C compiler and reinstall gensim for fast training.
  "C extension not loaded, training will be slow. "
```

```
C extension not loaded, training will be slow.
100%|██████████| 45949/45949 [11:50<00:00, 64.69it/s]
```

In [60]:

```python
k=[2,3,4,5,6,7,8,9,10]
X_train=tfidf_sent_vectors_train
inertia=[]
for i in k:
    model=KMeans(n_clusters=i, n_jobs=-1)
    model.fit(X_train)
    inertia.append(model.inertia_)
#finding best k using elbow method
plt.plot(k, inertia)
plt.xlabel('K(clusters)')
plt.ylabel('Inertia')
plt.title('Inertia VS K ')
plt.grid()
plt.show()
```



In [61]:

```python
#we can see that at k=4 there is a point of inflection
model=KMeans(n_clusters=4, n_jobs=-1)
model.fit(X_train)
```

```
Out[61]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
             n_clusters=4, n_init=10, n_jobs=-1, precompute_distances='auto',
             random_state=None, tol=0.0001, verbose=0)
```

In [62]:
```
cluster1,cluster2,cluster3,cluster4=[],[],[],[]
for i in range(model.labels_.shape[0]):
    if model.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    elif model.labels_[i] == 1:
        cluster2.append(preprocessed_reviews[i])
    elif model.labels_[i] == 2:
        cluster3.append(preprocessed_reviews[i])
    else:
        cluster4.append(preprocessed_reviews[i])
```

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

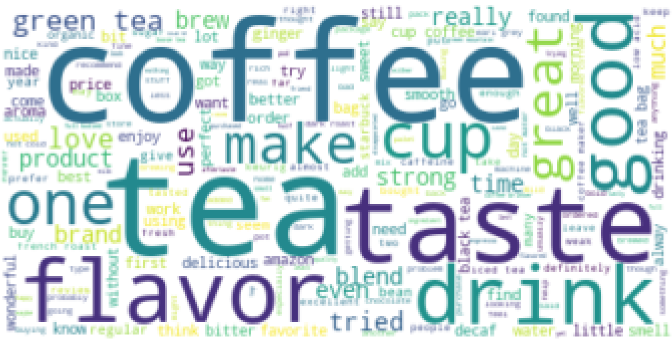
In [63]:
```
#for cluster 1
data=''
for i in cluster1:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [64]:
```
#for cluster 2
data=''
for i in cluster2:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [65]:
```
#for cluster 3
```

```python
data=''
for i in cluster3:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```


```python
In [66]: #for cluster 4
data=''
for i in cluster4:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



[5.2] Agglomerative Clustering

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

In [67]:
```
preprocessed_reviews=preprocessed_reviews[:5000]
list_of_sentance_train=[]
for sentance in preprocessed_reviews:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    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
    sent_vectors_train.append(sent_vec)
```

```
print(len(sent_vectors_train))
print(len(sent_vectors_train[0]))
```

```
C:\Users\MANISH\Anaconda3\lib\site-packages\gensim\models\base_any2vec.
py:743: UserWarning: C extension not loaded, training will be slow. Ins
tall a C compiler and reinstall gensim for fast training.
  "C extension not loaded, training will be slow. "
100%|██████████| 5000/5000 [00:31<00:00, 160.59it/s]
```

```
5000
50
```

for k=2

In [68]:

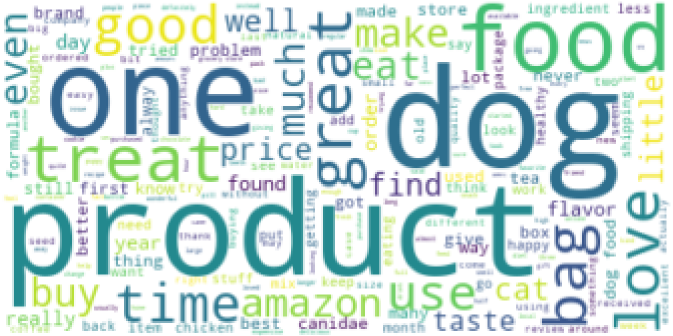
```
from sklearn.cluster import AgglomerativeClustering
X_train=sent_vectors_train
model=AgglomerativeClustering(n_clusters=2).fit(X_train)
```

In [69]:

```
cluster1,cluster2=[],[]
for i in range(model.labels_.shape[0]):
    if model.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    else:
        cluster2.append(preprocessed_reviews[i])
```

In [70]:

```
#for cluster 1
data=''
for i in cluster1:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



In [71]:

```
#for cluster 2
data=''
for i in cluster2:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

for k=5

In [72]:
```
X_train=sent_vectors_train
model=AgglomerativeClustering(n_clusters=5).fit(X_train)
```

In [73]:
```
cluster1,cluster2,cluster3,cluster4,cluster5=[],[],[],[],[]
for i in range(model.labels_.shape[0]):
    if model.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    elif model.labels_[i] == 1:
        cluster2.append(preprocessed_reviews[i])
    elif model.labels_[i] == 2:
        cluster3.append(preprocessed_reviews[i])
    elif model.labels_[i] == 3:
        cluster4.append(preprocessed_reviews[i])
    else :
        cluster5.append(preprocessed_reviews[i])
```

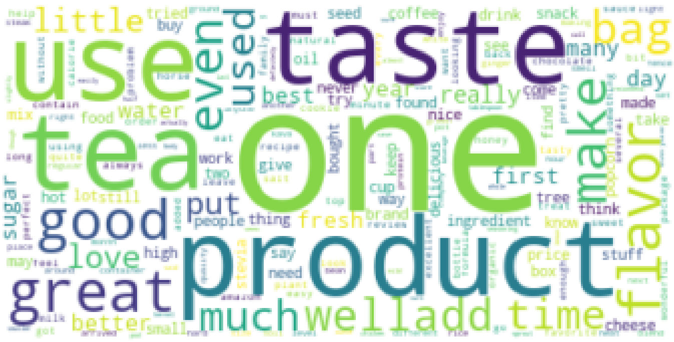
In [74]:
```
#for cluster 1
data=''
for i in cluster1:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [75]:
```
#for cluster 2
data=''
for i in cluster2:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [76]:
```
#for cluster 3
```

```python
data=''
for i in cluster3:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [77]:
```python
#for cluster 5
data=''
for i in cluster5:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



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

In [78]:

```
list_of_sentance_train=[]
for sentance in preprocessed_reviews:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10, max_features=500)

tf_idf_matrix=tf_idf_vect.fit_transform(preprocessed_reviews)


tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect.idf_)))

#for train data

tfidf_sent_vectors_train = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
```

```
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_train.append(sent_vec)
    row += 1
```

```
C:\Users\MANISH\Anaconda3\lib\site-packages\gensim\models\base_any2vec.
py:743: UserWarning: C extension not loaded, training will be slow. Ins
tall a C compiler and reinstall gensim for fast training.
  "C extension not loaded, training will be slow. "
100%|██████████| 5000/5000 [00:50<00:00, 98.42it/s]
```

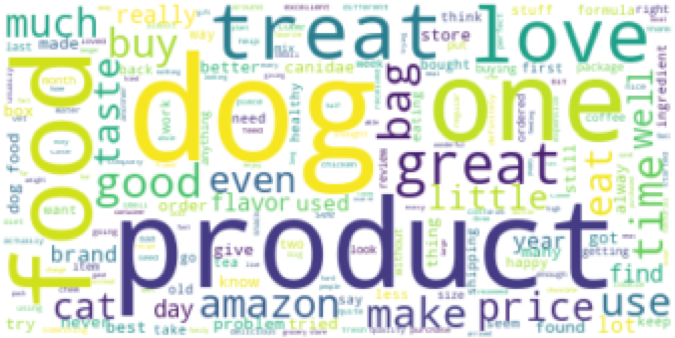
for k=2

In [79]:
```
from sklearn.cluster import AgglomerativeClustering
X_train=tfidf_sent_vectors_train
model=AgglomerativeClustering(n_clusters=2).fit(X_train)
```

In [80]:
```
cluster1,cluster2=[],[]
for i in range(model.labels_.shape[0]):
    if model.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    else:
        cluster2.append(preprocessed_reviews[i])
```

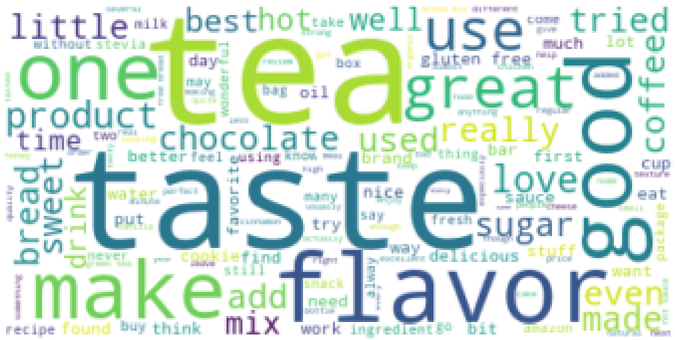
In [81]:
```
#for cluster 1
data=''
for i in cluster1:
    data+=str(i)
from wordcloud import WordCloud
```

```
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



In [82]:

```
#for cluster 2
data=''
for i in cluster2:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



for k=5

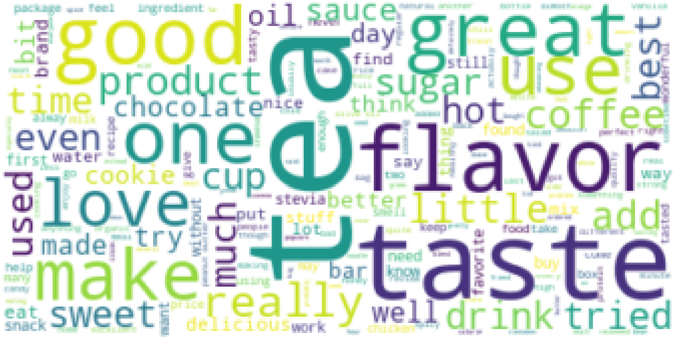
In [83]:
```
model=AgglomerativeClustering(n_clusters=5).fit(X_train)
cluster1,cluster2,cluster3,cluster4,cluster5=[],[],[],[],[]
for i in range(model.labels_.shape[0]):
    if model.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    elif model.labels_[i] == 1:
        cluster2.append(preprocessed_reviews[i])
    elif model.labels_[i] == 2:
        cluster3.append(preprocessed_reviews[i])
    elif model.labels_[i] == 3:
        cluster4.append(preprocessed_reviews[i])
    else :
        cluster5.append(preprocessed_reviews[i])
```

In [84]:
```
#for cluster 1
data=''
for i in cluster1:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
```

```
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [85]:

```
#for cluster 2
data=''
for i in cluster2:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

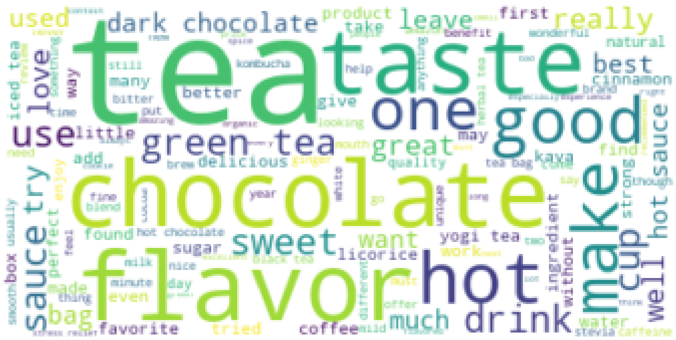


In [86]:
```
#for cluster 3
data=''
for i in cluster3:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [87]:
```
#for cluster 4
```

```
data=''
for i in cluster4:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [88]:
```
#for cluster 5
data=''
for i in cluster5:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



[5.3] DBSCAN Clustering

[5.3.1] Applying DBSCAN on AVG W2V, SET 3

In [89]:

```
preprocessed_reviews=preprocessed_reviews[:5000]
list_of_sentance_train=[]
for sentance in preprocessed_reviews:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
sent_vectors_train = [];
for sent in tqdm(list_of_sentance_train):
    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
    sent_vectors_train.append(sent_vec)
```

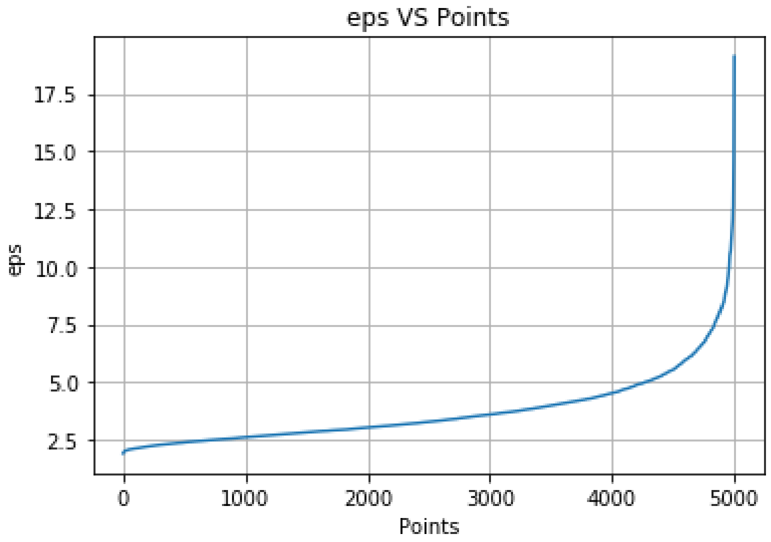
```
print(len(sent_vectors_train))
print(len(sent_vectors_train[0]))
```

```
C:\Users\MANISH\Anaconda3\lib\site-packages\gensim\models\base_any2vec.
py:743: UserWarning: C extension not loaded, training will be slow. Ins
tall a C compiler and reinstall gensim for fast training.
  "C extension not loaded, training will be slow. "
100%|██████████| 5000/5000 [00:33<00:00, 147.53it/s]
```

```
5000
50
```

In [99]:

```
min_points = 100
from sklearn.preprocessing import StandardScaler
data=StandardScaler().fit_transform(sent_vectors_train)
# Computing distances of nth-nearest neighbours
distance=[]
for x in data:
    value=np.sort(np.sum((data-x)**2,axis=1),axis=None)
    distance.append(value[min_points])
final_eps=np.sqrt(np.array(distance))

sorted_dist = np.sort(final_eps)
points = [point for point in range(len(sent_vectors_train))]

# Draw distances(d_i) VS points(x_i) plot
plt.plot(points, sorted_dist)
plt.xlabel('Points')
plt.ylabel('eps')
plt.title('eps VS Points')
plt.grid()
plt.show()
```



In [93]:
```
#we can see that point of inflexion is at eps=6
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=6, n_jobs=-1)
dbscan.fit(data)
print('No of clusters: ',len(set(dbscan.labels_)))
print('Cluster are ignoring (-1 for noise ): ',set(dbscan.labels_))
```

```
No of clusters:  2
Cluster are ignoring (-1 for noise ):  {0, -1}
```

In [94]:
```
#ignoring -1 as it is for noise
cluster1=[]
for i in range(dbscan.labels_.shape[0]):
    if dbscan.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
```

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

In [95]:
```python
#for cluster 1
data=''
for i in cluster1:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

[5.3.3] Applying DBSCAN on TFIDF W2V, SET 4

In [96]:
```python
list_of_sentance_train=[]
for sentance in preprocessed_reviews:
    list_of_sentance_train.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10, max_features
=500)

tf_idf_matrix=tf_idf_vect.fit_transform(preprocessed_reviews)
```

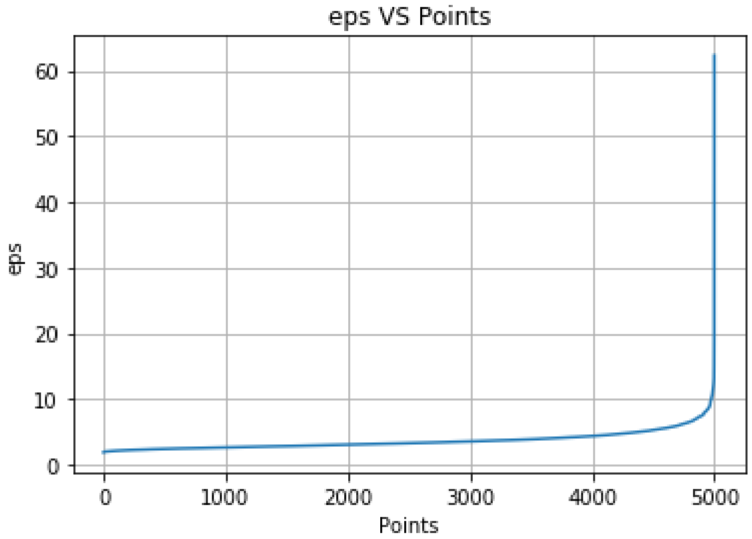
```
tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect
.idf_)))

#for train data

tfidf_sent_vectors_train = [];
row=0;
for sent in tqdm(list_of_sentance_train):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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_train.append(sent_vec)
    row += 1
```

```
C:\Users\MANISH\Anaconda3\lib\site-packages\gensim\models\base_any2vec.
py:743: UserWarning: C extension not loaded, training will be slow. Ins
tall a C compiler and reinstall gensim for fast training.
  "C extension not loaded, training will be slow. "
100%|██████████| 5000/5000 [00:54<00:00, 91.97it/s]
```

In [97]:

```
min_points = 100
from sklearn.preprocessing import StandardScaler
data=StandardScaler().fit_transform(tfidf_sent_vectors_train)
# Computing distances of nth-nearest neighbours
distance=[]
for x in data:
    value=np.sort(np.sum((data-x)**2,axis=1),axis=None)
    distance.append(value[min_points])
final_eps=np.sqrt(np.array(distance))
```

```
sorted_dist = np.sort(final_eps)
points = [point for point in range(len(tfidf_sent_vectors_train))]

# Draw distances(d_i) VS points(x_i) plot
plt.plot(points, sorted_dist)
plt.xlabel('Points')
plt.ylabel('eps')
plt.title('eps VS Points')
plt.grid()
plt.show()
```



In [98]:

```
#we can see that point of inflexion is at eps=9
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=9, n_jobs=-1)
dbscan.fit(data)
print('No of clusters: ',len(set(dbscan.labels_)))
print('Cluster are ignoring (-1 for noise ): ',set(dbscan.labels_))
```

```
No of clusters:  2
Cluster are ignoring (-1 for noise ):  {0, -1}
```

In [100]:

```
#ignoring -1 as it is for noise
```

```
cluster1=[]
for i in range(dbscan.labels_.shape[0]):
    if dbscan.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
```

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

In [101]:
```
#for cluster 1
data=''
for i in cluster1:
    data+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white").generate(data)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

[6] Conclusions

In [103]:

```
#prettytable for kmeans
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer","Best k"]
x.add_row(['BOW','5'])
x.add_row(['TFIDF','3'])
x.add_row(['AVG W2vec','4'])
x.add_row(['TFIDF W2vec','4'])
print(x)
```

```
+-------------+--------+
|  Vectorizer | Best k |
+-------------+--------+
|     BOW     |   5    |
|    TFIDF    |   3    |
|  AVG W2vec  |   4    |
| TFIDF W2vec |   4    |
+-------------+--------+
```

In [104]:

```
#prettytable for DBSCAN
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer","Optimal eps"]

x.add_row(['AVG W2vec','6'])
x.add_row(['TFIDF W2vec','9'])
print(x)
```

```
+-------------+-------------+
|  Vectorizer | Optimal eps |
+-------------+-------------+
|  AVG W2vec  |      6      |
| TFIDF W2vec |      9      |
+-------------+-------------+
```

Procedures and Observations

1) In K means clustering we took 50k datapoints, for Agglomerative and DBSCAN we took 5k

datapoints as these algorithms are very expensive in terms of run-time.

2) For K means clustering we applied k-means for different value of k and selected optimal k with the help of elbow method from graph between inertia vs k.

3) We sorted out top 5 features in each clusters of Bow and TFIDF vectorize.

4) For agglomerative clustering we took n_clusters=[2,3] and applied algorithm on it and plotted the word cloud for each clusters.

5) And at the end we applied DBSCAN on Avg-W2vec and TFIDF-W2vec, for optimal eps we first calculated the nth distance from each point, sorted them and plotted the curve between points and distances and the again we applied elbow method to figure out the best eps(At point of inflexion).