Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", co
n)
# 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 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 (50000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
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 Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia	1	1	1	1219017600	"Delight" says it all

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

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

Out[5]:

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

In [6]:

```
display['COUNT(*)'].sum()
```

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
```

```
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE

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='qui
cksort', na_position='last')
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=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	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because 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 be
tter, but a heck of a lot more work. this is great to have on hand for last minute dessert needs w
here you really want to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are hig
h priced and high in calories, this one is a great bargain and tastes great, I highly recommend fo
r the lady gym rats, 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 that 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 ove
r 12 boxes of this tea myself, and highly recommend it for the following reasons:<br /><br />-Qual
ity:  First, 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 my 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 ab
out your tea and leave it brewing for 20+ minutes like I sometimes do, the quality of this tea is
such that you still get a smooth but deeper flavor without the bad after taste.  The leaves themse
lves 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 other discernible ingredients.
This isn't your standard cheap paper bag with a mix of unknown ingredients that have been ground d
own to a fine powder, leaving you to wonder what it is you are actually drinking.<br /><br />-Tast
e:  This tea offers notes of real pineapple and other hints of tropical fruits, yet 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 believ
e most will enjoy.  If you want it sweet, you can add sugar, splenda, etc but this really is not n
ecessary as this tea offers an inherent warmth of flavor through it's ingredients.<br /><br />-Pri
ce:  This tea offers an excellent product at an exceptional price (especially when purchased at th
e prices Amazon offers).  Compared to other brands which I believe to be of similar quality
(Mighty Leaf, Rishi, Two Leaves, etc.), Revolution offers a superior product at an outstanding pri
ce.  I have been purchasing this through Amazon for less per box than I would be paying at my loca
l grocery store for Lipton, etc.<br /><br />Overall, this is a wonderful tea that is comparable, a
nd even better than, other teas that are priced much higher.  It offers a well-balanced cup of gre
en tea that I believe many will enjoy.  In terms of taste, quality, and price, I would argue you w
on't find a better combination that that offered by Revolution's Tropical Green Tea.
==================================================
```

In [15]:

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

```
sent_1500 = 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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [16]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because 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 be
tter, but a heck of a lot more work. this is great to have on hand for last minute dessert needs w
here you really want to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are hig
h priced and high in calories, this one is a great bargain and tastes great, I highly recommend fo
r the lady gym rats, 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 that 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 ove
r 12 boxes of this tea myself, and highly recommend it for the following reasons:-Quality:  First,
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 my 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 your tea and lea
ve it brewing for 20+ minutes like I sometimes do, the quality of this tea is such that you still
get a smooth but deeper flavor without the bad after taste.  The leaves themselves are whole leave
s (not powdered stems, branches, etc commonly found in other brands), and the high-quality nylon b
ags also include chunks of tropical fruit and other discernible ingredients.  This isn't your
standard cheap paper bag with a mix of unknown ingredients that have been ground down to a fine po
wder, leaving you to wonder what it is you are actually drinking.-Taste:  This tea offers notes of
real pineapple and other hints of tropical fruits, yet isn't sweet or artificially flavored.  You
have the foundation of a high-quality young hyson green tea for those true "tea flavor" lovers, ye
t the subtle hints of fruit make this a truly unique tea that I believe most will enjoy.  If you w
ant it sweet, you can add sugar, splenda, etc but this really is not necessary as this tea offers
an inherent warmth of flavor through it's ingredients.-Price:  This tea offers an excellent
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.), R
evolution offers a superior product at an outstanding price.  I have been purchasing this through
Amazon for less per box than I would be paying at my local grocery store for Lipton, etc.Overall,
this is a wonderful tea that is comparable, and even better than, other teas that are priced much
higher.  It offers a well-balanced 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 combination that that offered by R
evolution's Tropical Green Tea.
```

In [17]:

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

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

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

In [18]:

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

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

In [19]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

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

In [20]:

```
#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 pr
iced and high in calories this one is a great bargain and tastes great I highly recommend for the
lady gym rats probably 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', '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', "y
ou'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', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

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

```
100%|██████████| 46071/46071 [00:16<00:00, 2769.86it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'great flavor low calories high nutrients high protein usually protein powders high priced high ca
lories one great bargain tastes great highly recommend lady gym rats probably not macho enough guy
s since soy based'
```

[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(max_features = 2000,min_df = 10) #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  ['able', 'absolute', 'absolutely', 'according', 'acid', 'acidic', 'across', 'a
ctual', 'actually', 'add']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 2000)
the number of unique words  2000
```

[4.2] Bi-Grams and n-Grams.

In [26]:

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

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

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_s
hape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 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,max_features=5000)
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 buy', 'able find',
'able get', 'absolute', 'absolute favorite', 'absolutely', 'absolutely best', 'absolutely
delicious']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (46071, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[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 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=Tr
ue)
        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 ")
```

```
[('awesome', 0.8388543128967285), ('fantastic', 0.8100770711898804), ('good', 0.7924799919128418),
('terrific', 0.7708409428596497), ('amazing', 0.7702434062957764), ('excellent',
0.760534405708313), ('wonderful', 0.7364969253540039), ('perfect', 0.7134552001953125), ('decent',
0.6752809286117554), ('nice', 0.6618160009384155)]
==================================================
[('best', 0.7235621213912964), ('greatest', 0.7145978212356567), ('closest', 0.6845759153366089),
('awful', 0.6410499811172485), ('nastiest', 0.6340367197990417), ('eaten', 0.6287020444869995), ('
experienced', 0.6271936297416687), ('tastiest', 0.6190468072891235), ('disgusting',
0.6129512786865234), ('horrible', 0.6010608673095703)]
```

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%|██████████| 46071/46071 [01:30<00:00, 511.10it/s]
```

```
46071
50
```

[4.4.1.2] TFIDF weighted W2v

In [32]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer(min_df=10, max_features=5000)
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 [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 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%|██████████| 46071/46071 [03:40<00:00, 208.79it/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 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 [67]:

```
#import statements
from sklearn.cluster import KMeans
from wordcloud import WordCloud
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from prettytable import PrettyTable
```

In [76]:

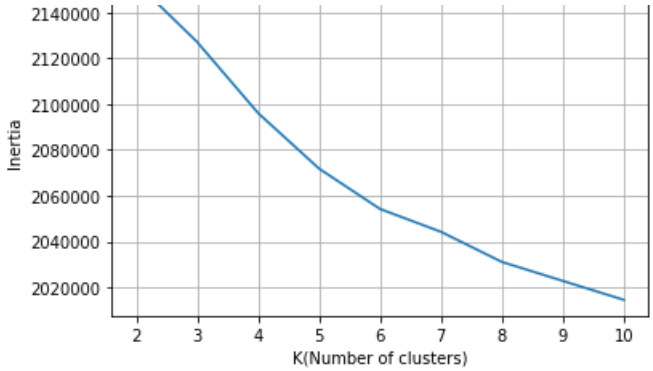
```
count_vect = CountVectorizer(max_features = 2000,min_df = 10)
X_tr=count_vect.fit_transform(preprocessed_reviews)
```

In [77]:

```
#Applying k-means to find best k
k_values=[2,3,4,5,6,7,8,9,10]
inertia=[]
for i in k_values:
    clf=KMeans(n_clusters=i, n_jobs=-1)
    clf.fit(X_tr)
    inertia.append(clf.inertia_)
```

In [78]:

```
#finding the best k using elbow method
plt.plot(k_values, inertia)
plt.xlabel('K(Number of clusters)')
plt.ylabel('Inertia')
plt.title('K VS Inertia')
plt.grid()
plt.show()
```



In [80]:

```
#We can observe that at k=6,we get the bending
kmeans_bow=6
clf=KMeans(n_clusters=kmeans_bow, n_jobs=-1)
clf.fit(X_tr)
```

Out[80]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=6, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [81]:

```
#some help from AAIC members
#Getting the reviews of different clusters
cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []
cluster5 = []
cluster6 = []

for i in range(clf.labels_.shape[0]):
    if clf.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    elif clf.labels_[i] == 1:
        cluster2.append(preprocessed_reviews[i])
    elif clf.labels_[i] == 2:
        cluster3.append(preprocessed_reviews[i])
    elif clf.labels_[i] == 3:
        cluster4.append(preprocessed_reviews[i])
    elif clf.labels_[i] == 4:
        cluster5.append(preprocessed_reviews[i])
    else:
        cluster6.append(preprocessed_reviews[i])
```

In [83]:

```
# Printing the number of reviews in each cluster
print("No. of reviews in Cluster-1 : ",len(cluster1))
print("\nNo. of reviews in Cluster-2 : ",len(cluster2))
print("\nNo. of reviews in Cluster-3 : ",len(cluster3))
print("\nNo. of reviews in Cluster-4 : ",len(cluster4))
print("\nNo. of reviews in Cluster-5 : ",len(cluster5))
print("\nNo. of reviews in Cluster-6 : ",len(cluster6))
```

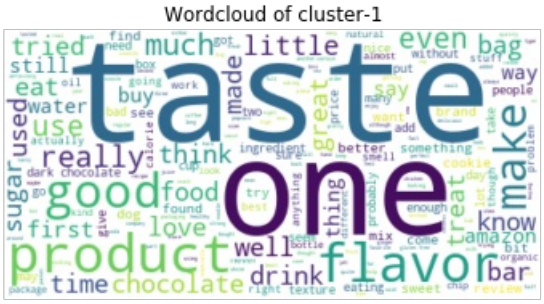
```
No. of reviews in Cluster-1 :  1741

No. of reviews in Cluster-2 :  10783

No. of reviews in Cluster-3 :  1173

No. of reviews in Cluster-4 :  1361

No. of reviews in Cluster-5 :  903

No. of reviews in Cluster-6 :  30110
```

No. of reviews in cluster-0 : 50110

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

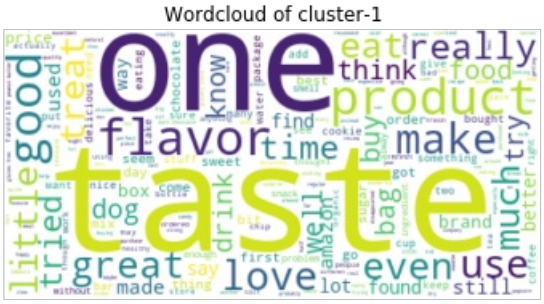
In [84]:

```
#some help from AAIC members
#for cluster 1
text=''
for word in cluster1:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```



In [85]:

```
#some help from AAIC members
#for cluster 2
text=''
for word in cluster2:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```



In [86]:

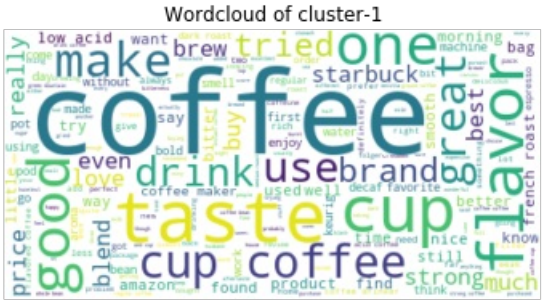
```
#some help from AAIC members
#for cluster 3
text=''
for word in cluster3:
    text+=str(word)
```

```
#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```

We can observe that the most frequent word is tea.So the above cluster is reviews of tea product

In [87]:

```
#some help from AAIC members
#for cluster 4
text=''
for word in cluster4:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```



We can observe that the most frequent word is coffee.So the above cluster is reviews of the coffee product

In [88]:

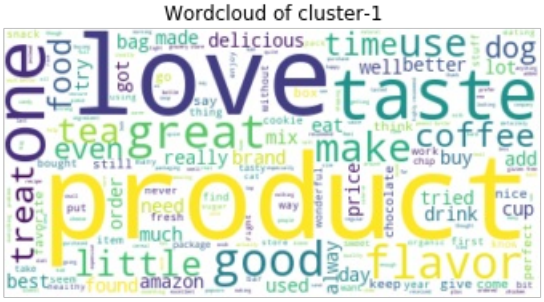
```
#some help from AAIC members
#for cluster 5
text=''
for word in cluster5:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
```

```
plt.title("Wordcloud of cluster-1")
plt.show()
```

Above cluster represents the reviews of cat/dog food product

In [89]:

```
#some help from AAIC members
#for cluster 6
text=''
for word in cluster6:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```



From the frequent words of above cluster we are not sure about reviews of a particular product but we are sure that the product is very good

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

In [90]:

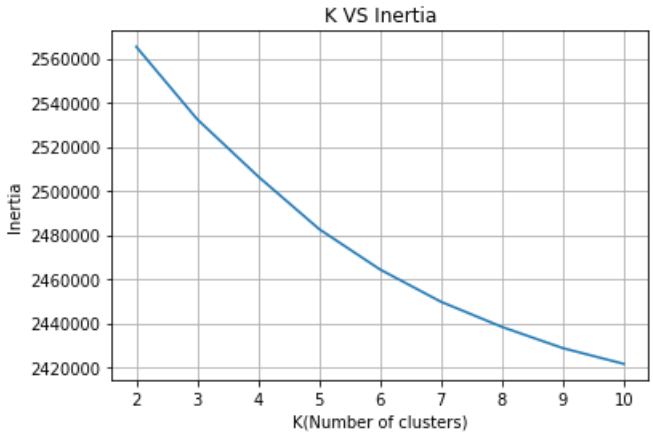
```
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
X_tr = count_vect.fit_transform(preprocessed_reviews)
```

In [91]:

```
#Applying k-means to find best k
k_values=[2,3,4,5,6,7,8,9,10]
inertia=[]
for i in k_values:
    clf=KMeans(n_clusters=i, n_jobs=-1)
    clf.fit(X_tr)
    inertia.append(clf.inertia_)
```

In [92]:

```
#finding the best k using elbow method
plt.plot(k_values, inertia)
plt.xlabel('K(Number of clusters)')
plt.ylabel('Inertia')
plt.title('K VS Inertia')
plt.grid()
plt.show()
```



In [93]:

```
#we can observe that at k=3,there is a bend
kmeans_tfidf=3
clf=KMeans(n_clusters=kmeans_tfidf, n_jobs=-1)
clf.fit(X_tr)
```

Out[93]:

```
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 [94]:

```
#Getting the reviews of different clusters
cluster1 = []
cluster2 = []
cluster3 = []

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

In [95]:

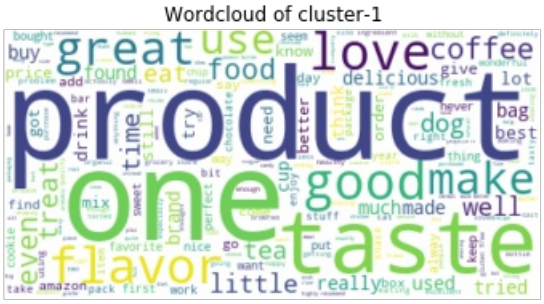
```
# Printing the number of reviews in each cluster
print("No. of reviews in Cluster-1 : ",len(cluster1))
print("\nNo. of reviews in Cluster-2 : ",len(cluster2))
print("\nNo. of reviews in Cluster-3 : ",len(cluster3))
```

```
No. of reviews in Cluster-1 :  37744

No. of reviews in Cluster-2 :  7120

No. of reviews in Cluster-3 :  1207
```

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

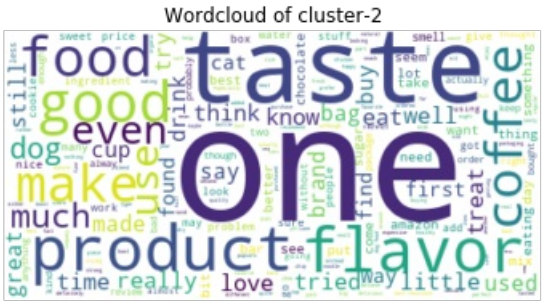
In [96]:

```
#some help from AAIC members
#for cluster 1
text=''
for word in cluster1:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```



In [97]:

```
#some help from AAIC members
#for cluster 2
text=''
for word in cluster2:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-2")
plt.show()
```

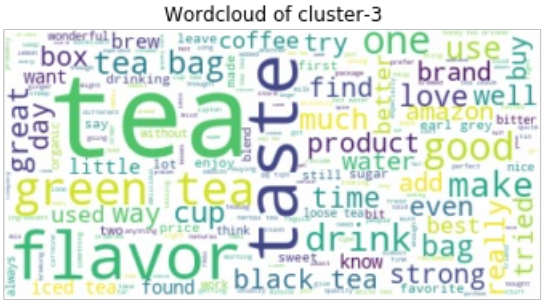


We can observe that above cluster represents taste of coffee

In [98]:

```
#some help from AAIC members
#for cluster 3
text=''
for word in cluster3:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)
```

```
# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-3")
plt.show()
```



We can observe that the most frequent word is tea.So the above cluster is reviews of tea product

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

In [118]:

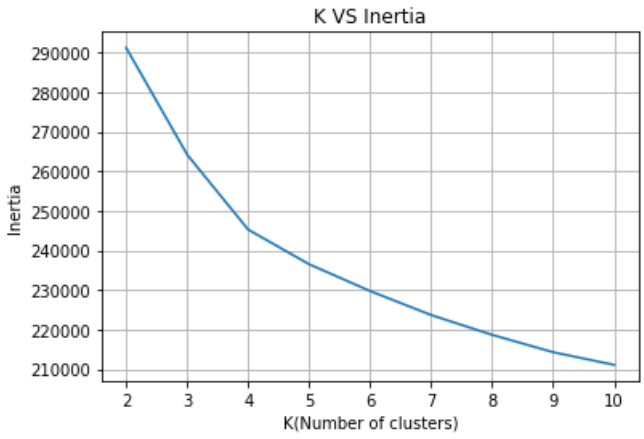
```
X_tr=sent_vectors
```

In [119]:

```
#Applying k-means to find best k
k_values=[2,3,4,5,6,7,8,9,10]
inertia=[]
for i in k_values:
    clf=KMeans(n_clusters=i, n_jobs=-1)
    clf.fit(X_tr)
    inertia.append(clf.inertia_)
```

In [120]:

```
#finding the best k using elbow method
plt.plot(k_values, inertia)
plt.xlabel('K(Number of clusters)')
plt.ylabel('Inertia')
plt.title('K VS Inertia')
plt.grid()
plt.show()
```



In [121]:

```
#we can observe that at k=4,there is bending
kmeans_avgw2v=4
```

```
clf=KMeans(n_clusters=kmeans_avgw2v, n_jobs=-1)
clf.fit(X_tr)
```

Out[121]:

```
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 [122]:

```
#Getting the reviews of different clusters
cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []

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

In [123]:

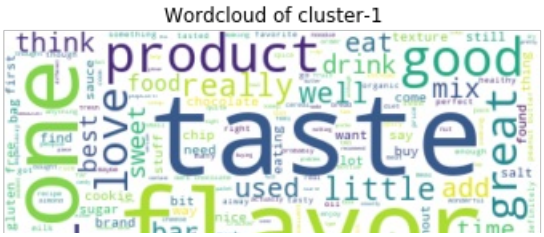
```
# Printing the number of reviews in each cluster
print("No. of reviews in Cluster-1 : ",len(cluster1))
print("\nNo. of reviews in Cluster-2 : ",len(cluster2))
print("\nNo. of reviews in Cluster-3 : ",len(cluster3))
print("\nNo. of reviews in Cluster-4 : ",len(cluster4))
```

```
No. of reviews in Cluster-1 :  18636

No. of reviews in Cluster-2 :  7913

No. of reviews in Cluster-3 :  12749

No. of reviews in Cluster-4 :  6773
```

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

In [124]:

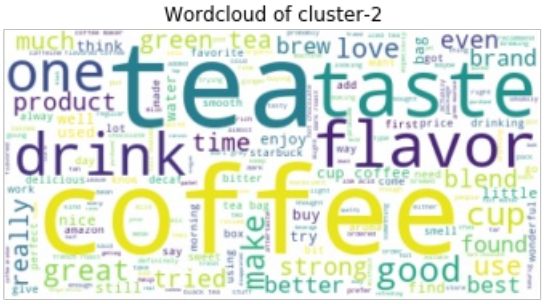
```
#some help from AAIC members
#for cluster 1
text=''
for word in cluster1:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```



We are not sure of product type but by observing the cluster we can know that the product is good

In [125]:

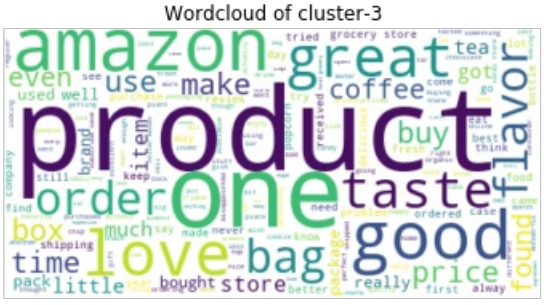
```
#some help from AAIC members
#for cluster 2
text=''
for word in cluster2:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-2")
plt.show()
```



We can observe that the most frequent word is coffee.So the above cluster is reviews of coffee product

In [126]:

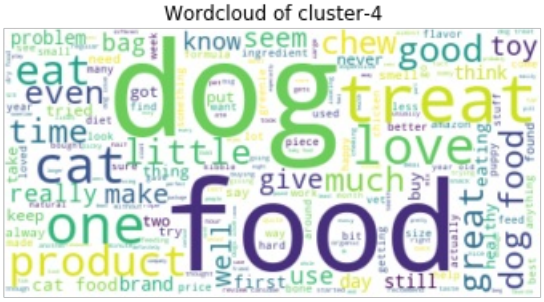
```
#some help from AAIC members
#for cluster 3
text=''
for word in cluster3:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-3")
plt.show()
```



We are not sure of product type but by observing the cluster we can know that the product is good

In [127]:

```
#some help from AAIC members
#for cluster 4
text=''
for word in cluster4:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-4")
plt.show()
```



We can observe that the most frequent word is dog and food.So the above cluster is reviews of dogs food product

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

In [108]:

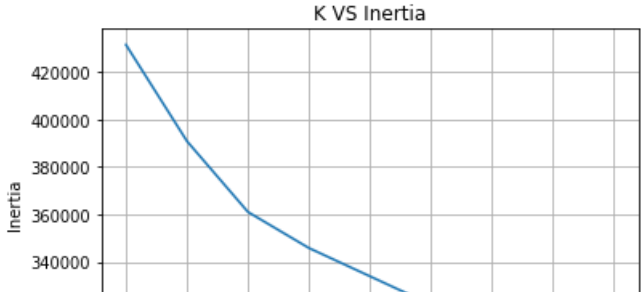
```
X_tr=tfidf_sent_vectors
```

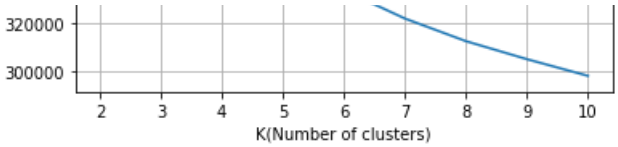
In [109]:

```
#Applying k-means to find best k
k_values=[2,3,4,5,6,7,8,9,10]
inertia=[]
for i in k_values:
    clf=KMeans(n_clusters=i, n_jobs=-1)
    clf.fit(X_tr)
    inertia.append(clf.inertia_)
```

In [110]:

```
#finding the best k using elbow method
plt.plot(k_values, inertia)
plt.xlabel('K(Number of clusters)')
plt.ylabel('Inertia')
plt.title('K VS Inertia')
plt.grid()
plt.show()
```





In [111]:

```
#we can observe that at k=4,there is bending
kmeans_tfidfw2v=4
clf=KMeans(n_clusters=kmeans_tfidfw2v, n_jobs=-1)
clf.fit(X_tr)
```

Out[111]:

```
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 [112]:

```
#Getting the reviews of different clusters
cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []

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

In [113]:

```
# Printing the number of reviews in each cluster
print("No. of reviews in Cluster-1 : ",len(cluster1))
print("\nNo. of reviews in Cluster-2 : ",len(cluster2))
print("\nNo. of reviews in Cluster-3 : ",len(cluster3))
print("\nNo. of reviews in Cluster-4 : ",len(cluster4))
```

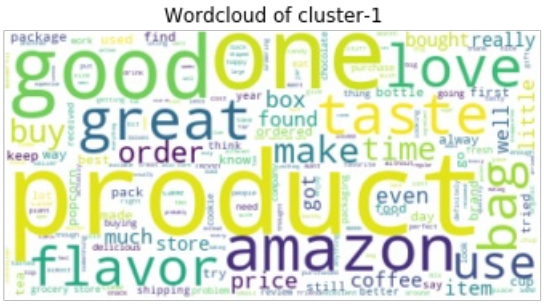
```
No. of reviews in Cluster-1 :  14449

No. of reviews in Cluster-2 :  18384

No. of reviews in Cluster-3 :  6259

No. of reviews in Cluster-4 :  6979
```

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

In [114]:

```
#some help from AAIC members
#for cluster 1
text=''
for word in cluster1:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
```

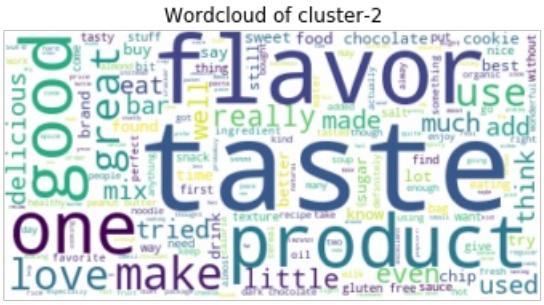
```
plt.show()
```



We are not sure of product type but by observing the cluster we can know that the product is good

In [115]:

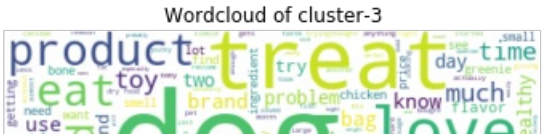
```
#some help from AAIC members
#for cluster 2
text=''
for word in cluster2:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-2")
plt.show()
```



We are not sure of product type but by observing the cluster we can know that the product is good

In [116]:

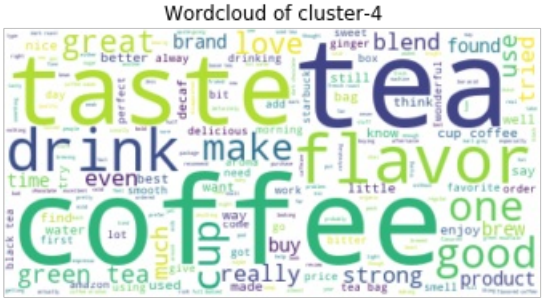
```
#some help from AAIC members
#for cluster 3
text=''
for word in cluster3:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-3")
plt.show()
```



We can observe that the most frequent word is dog and food.So the above cluster is reviews of food product related to dog

In [117]:

```
#some help from AAIC members
#for cluster 4
text=''
for word in cluster4:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-4")
plt.show()
```



We can observe that the most frequent word is coffee and tea.So the above cluster is reviews of coffee and tea product

[5.2] Agglomerative Clustering

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

In [71]:

```
#Taking due to memory constraints
preprocessed_reviews=preprocessed_reviews[:5000]
# average Word2Vec
# compute average word2vec for each review.
list_of_sentance_tr=[]
for sentance in preprocessed_reviews:
    list_of_sentance_tr.append(sentance.split())
X_tr = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_tr): # 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
    X_tr.append(sent_vec)
```

```
100%|██████████| 5000/5000 [00:08<00:00, 561.02it/s]
```

Number of clusters k=2

In [72]:

```
clf=AgglomerativeClustering(n_clusters=2)
clf.fit(X_tr)
```

Out[72]:

```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=2,
            pooling_func='deprecated')
```

In [73]:

```
#Getting the reviews of different clusters
cluster1=[]
cluster2=[]
for i in range(clf.labels_.shape[0]):
    if clf.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    else:
        cluster2.append(preprocessed_reviews[i])
```

In [74]:

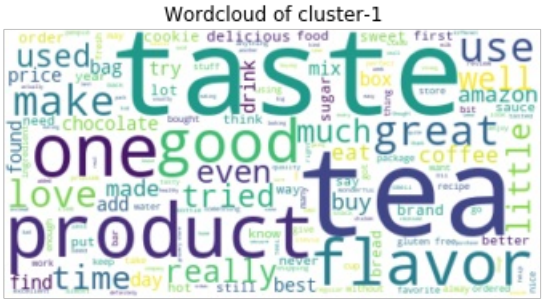
```
# Printing the number of reviews in each cluster
print("No. of reviews in Cluster-1 : ",len(cluster1))
print("\nNo. of reviews in Cluster-2 : ",len(cluster2))
```

```
No. of reviews in Cluster-1 :  4110

No. of reviews in Cluster-2 :  890
```

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

In [75]:

```
#some help from AAIC members
#for cluster 1
text=''
for word in cluster1:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```



We can observe that one of the most frequent word is tea.So the above cluster is reviews of tea product

In [76]:

```
#some help from AAIC members
#for cluster 2
text=''
for word in cluster2:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-2")
plt.show()
```

We can observe that the most frequent word is dog.So the above cluster is food reviews related to dog product

Number of clusters k=5

In [77]:

```
clf=AgglomerativeClustering(n_clusters=5)
clf.fit(X_tr)
```

Out[77]:

```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=5,
            pooling_func='deprecated')
```

In [78]:

```
#Getting the reviews of different clusters
cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []
cluster5 = []

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

In [79]:

```
# Printing the number of reviews in each cluster
print("No. of reviews in Cluster-1 : ",len(cluster1))
print("\nNo. of reviews in Cluster-2 : ",len(cluster2))
print("\nNo. of reviews in Cluster-3 : ",len(cluster3))
print("\nNo. of reviews in Cluster-4 : ",len(cluster4))
print("\nNo. of reviews in Cluster-5 : ",len(cluster5))
```

```
No. of reviews in Cluster-1 :  1755

No. of reviews in Cluster-2 :  1203

No. of reviews in Cluster-3 :  818

No. of reviews in Cluster-4 :  890

No. of reviews in Cluster-5 :  334
```

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

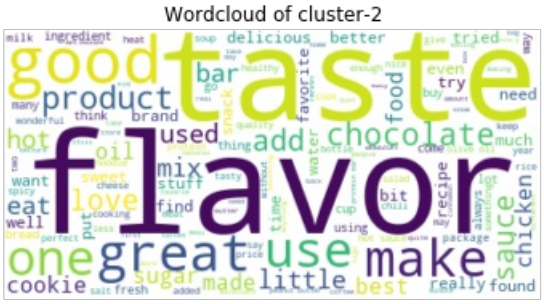
In [80]:

```
#some help from AAIC members
#for cluster 1
text=''
for word in cluster1:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```

We are not sure of product type but by observing the cluster we can know that the product is good

In [81]:

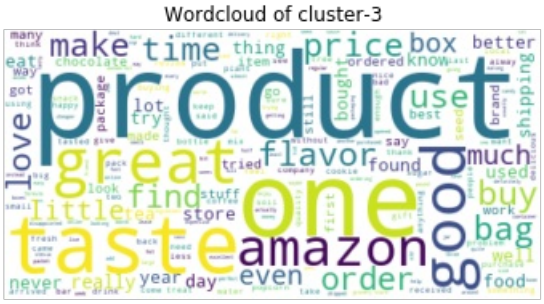
```
#some help from AAIC members
#for cluster 2
text=''
for word in cluster2:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-2")
plt.show()
```



We are not sure of product type but by observing the cluster we can know that the product is good

In [82]:

```
#some help from AAIC members
#for cluster 3
text=''
for word in cluster1:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-3")
plt.show()
```



In [83]:

```
#some help from AAIC members
#for cluster 4
text=''
for word in cluster4:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-4")
plt.show()
```

We can observe that one of the most frequent word is dog.So the above cluster is food reviews of food product

In [84]:

```
#some help from AAIC members
#for cluster 5
text=''
for word in cluster5:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-5")
plt.show()
```

We can observe that the above cluster is about bread product

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

In [85]:

```
# 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

list_of_sentance_tr=[]
for sentance in preprocessed_reviews:
    list_of_sentance_tr.append(sentance.split())
X_tr = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_tr): # 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
    X_tr.append(sent_vec)
    row += 1
```

```
100%|██████████| 5000/5000 [00:22<00:00, 217.71it/s]
```

Number of clusters k=2

In [86]:

```
clf=AgglomerativeClustering(n_clusters=2)
clf.fit(X_tr)
```

Out[86]:

```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=2,
            pooling_func='deprecated')
```

In [87]:

```
#Getting the reviews of different clusters
cluster1=[]
cluster2=[]
for i in range(clf.labels_.shape[0]):
    if clf.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
    else:
        cluster2.append(preprocessed_reviews[i])
```

In [88]:

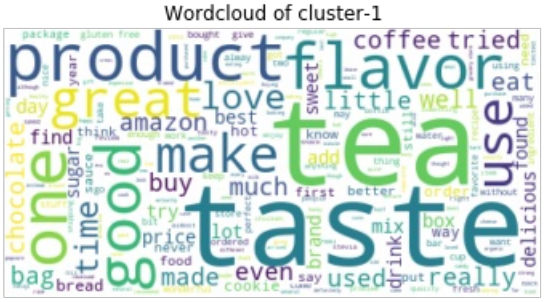
```
# Printing the number of reviews in each cluster
print("No. of reviews in Cluster-1 : ",len(cluster1))
print("\nNo. of reviews in Cluster-2 : ",len(cluster2))
```

```
No. of reviews in Cluster-1 :  4123

No. of reviews in Cluster-2 :  877
```

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

In [89]:

```
#some help from AAIC members
#for cluster 1
text=''
for word in cluster1:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```



We can observe that one of the most frequent word is tea.So the above cluster is reviews of tea product

In [90]:

```
#some help from AAIC members
#for cluster 2
text=''
for word in cluster2:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-2")
plt.show()
```

We can observe that one of the most frequent word is dog and food.So the above cluster is food reviews of dog related product

Number of clusters k=5

In [91]:

```
clf=AgglomerativeClustering(n_clusters=5)
clf.fit(X_tr)
```

Out[91]:

```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=5,
            pooling_func='deprecated')
```

In [92]:

```
#Getting the reviews of different clusters
cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []
cluster5 = []

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

In [93]:

```
# Printing the number of reviews in each cluster
```

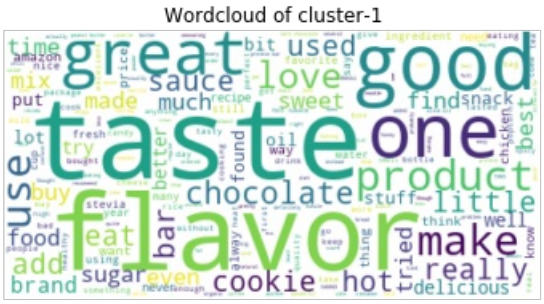
```
print("No. of reviews in Cluster-1 : ",len(cluster1))
print("\nNo. of reviews in Cluster-2 : ",len(cluster2))
print("\nNo. of reviews in Cluster-3 : ",len(cluster3))
print("\nNo. of reviews in Cluster-4 : ",len(cluster4))
print("\nNo. of reviews in Cluster-5 : ",len(cluster5))
```

```
No. of reviews in Cluster-1 :  1872

No. of reviews in Cluster-2 :  877

No. of reviews in Cluster-3 :  774

No. of reviews in Cluster-4 :  1175

No. of reviews in Cluster-5 :  302
```

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

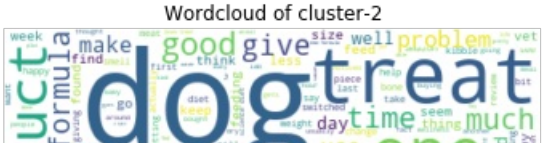
In [94]:

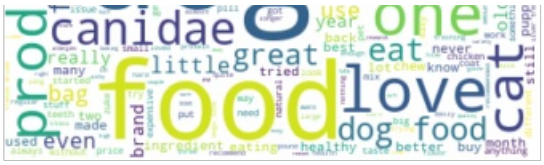
```
#some help from AAIC members
#for cluster 1
text=''
for word in cluster1:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-1")
plt.show()
```



In [95]:

```
#some help from AAIC members
#for cluster 2
text=''
for word in cluster2:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-2")
plt.show()
```

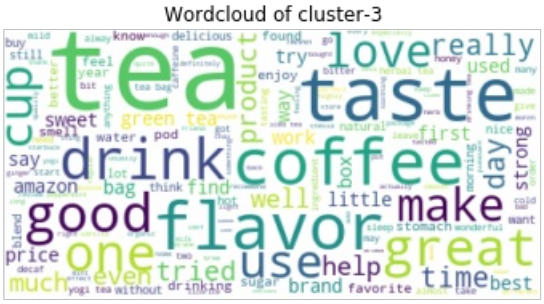




We can observe that one of the most frequent word is dog and food.So the above cluster is food reviews of dog related product

In [96]:

```
#some help from AAIC members
#for cluster 3
text=''
for word in cluster3:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-3")
plt.show()
```



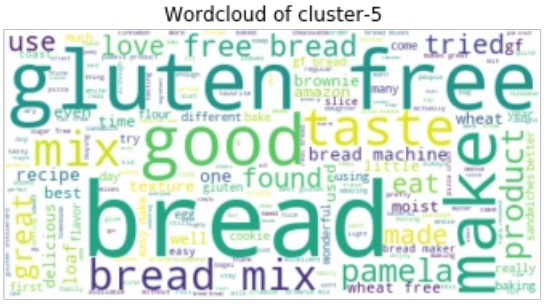
We can observe that the most frequent word is coffee and tea.So the above cluster is reviews of coffee and tea product

In [97]:

```
#some help from AAIC members
#for cluster 4
text=''
for word in cluster4:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-4")
plt.show()
```

In [98]:

```
#some help from AAIC members
#for cluster 5
text=''
for word in cluster5:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of cluster-5")
plt.show()
```



We can observe that one of the most frequent words is bread.So the above cluster is reviews of bread product

[5.3] DBSCAN Clustering

[5.3.1] Applying DBSCAN on AVG W2V, SET 3

In [41]:

```
#Taking due to memory constraints
preprocessed_reviews=preprocessed_reviews[:5000]
# average Word2Vec
# compute average word2vec for each review.
list_of_sentance_tr=[]
for sentance in preprocessed_reviews:
    list_of_sentance_tr.append(sentance.split())
X_tr = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_tr): # 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
    X_tr.append(sent_vec)
```

```
100%|██████████| 5000/5000 [00:08<00:00, 580.08it/s]
```

In [44]:

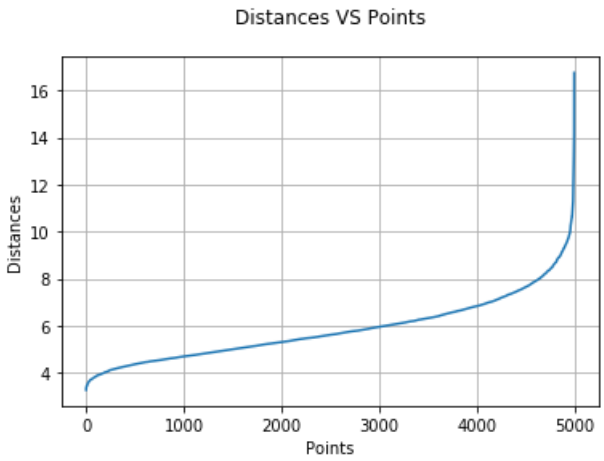
```
#Standardizing the data
std=StandardScaler(with_mean = False)
X_tr=std.fit_transform(X_tr)
```

In [47]:

```
#some help from AAIC members
min_pts=50
distance=[]

for data in X_tr:
    temp_dist=np.sort(np.sum((X_tr-data)**2,axis=1),axis=None)
    distance.append(temp_dist[min_pts])

sorted_distances=np.sort(distance)
points=[i for i in range(X_tr.shape[0])]
```

In [49]:

```
# Plotting elbow graph
sorted_distance=np.sqrt(sorted_distances)
plt.plot(points, sorted_distance)
plt.xlabel('Points')
plt.ylabel('Distances',)
plt.title('Distances VS Points\n')
plt.grid()
plt.show()
```



In [51]:

```
#we can observe that inflection is at point 9
clf = DBSCAN(eps=9, n_jobs=-1)
clf.fit(X_tr)
print("Total number of cluster with label -1 is: ", list(clf.labels_).count(-1))
print("total number of cluster with label 0 is: ", list(clf.labels_).count(0))
```

```
Total number of cluster with label -1 is:  10
total number of cluster with label 0 is:  4990
```

In [52]:

```
#We ignore cluster with label -1 as it is noise
cluster1=[]
for i in range(clf.labels_.shape[0]):
    if clf.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
```

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

In [55]:

```
#some help from AAIC members
#for cluster 1
text=''
for word in cluster1:
    text+=str(word)
```

```
#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Wordcloud of reviews")
plt.show()
```

We can observe that one of the most frequent words is tea but dog word is also present.So,this cluster words are mixed and cannot be judged

[5.3.3] Applying DBSCAN on TFIDF W2V, SET 4

In [60]:

```
# 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

list_of_sentance_tr=[]
for sentance in preprocessed_reviews:
    list_of_sentance_tr.append(sentance.split())
X_tr = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_tr): # 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
    X_tr.append(sent_vec)
    row += 1
```

```
100%|██████████| 5000/5000 [00:22<00:00, 222.63it/s]
```

In [61]:

```
#Standardizing the data
std=StandardScaler(with_mean = False)
X_tr=std.fit_transform(X_tr)
```

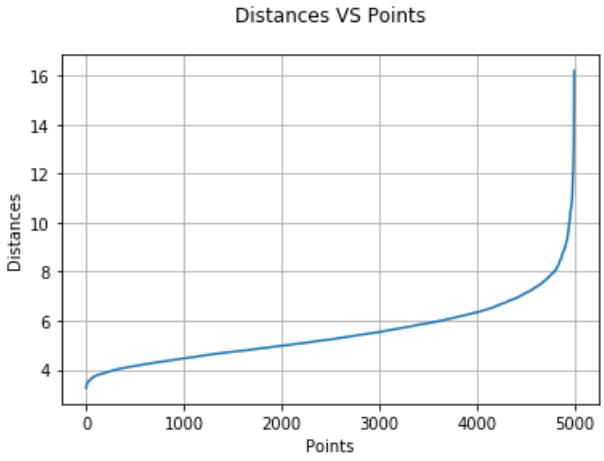
In [62]:

```
#some help from AAIC members
min_pts=50
distance=[]
```

```
for data in X_tr:
    temp_dist=np.sort(np.sum((X_tr-data)**2,axis=1),axis=None)
    distance.append(temp_dist[min_pts])

sorted_distances=np.sort(distance)
points=[i for i in range(X_tr.shape[0])]
```

In [63]:

```
# Plotting elbow graph
sorted_distance=np.sqrt(sorted_distances)
plt.plot(points, sorted_distance)
plt.xlabel('Points')
plt.ylabel('Distances',)
plt.title('Distances VS Points\n')
plt.grid()
plt.show()
```



In [64]:

```
#we can observe that inflection is at point 7
clf = DBSCAN(eps=7, n_jobs=-1)
clf.fit(X_tr)
print("Total number of cluster with label -1 is: ", list(clf.labels_).count(-1))
print("total number of cluster with label 0 is: ", list(clf.labels_).count(0))
```

```
Total number of cluster with label -1 is:  40
total number of cluster with label 0 is:  4960
```

In [65]:

```
#We ignore cluster with label -1 as it is noise
cluster1=[]
for i in range(clf.labels_.shape[0]):
    if clf.labels_[i] == 0:
        cluster1.append(preprocessed_reviews[i])
```

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

In [66]:

```
#some help from AAIC members
#for cluster 1
text=''
for word in cluster1:
    text+=str(word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# Plotting the word cloud
```

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

We can observe that one of the most frequent words is tea but dog word is also present.So,this cluster words are mixed and cannot be judged

[6] Conclusions

In [68]:

```
# prettytable for kmeans clustering
x = PrettyTable()
x.field_names = ["Vectorizer","Best k"]
x.add_row(['BOW','6'])
x.add_row(['TFIDF','3'])
x.add_row(['AVG-W2vec','4'])
x.add_row(['TFIDF-W2vec','4'])
print(x)
```

```
+-------------+--------+
|  Vectorizer | Best k |
+-------------+--------+
|     BOW     |   6    |
|    TFIDF    |   3    |
|  AVG-W2vec  |   4    |
| TFIDF-W2vec |   4    |
+-------------+--------+
```

In [70]:

```
#prettytable for DBSCAN clustering technique
from prettytable import PrettyTable
y = PrettyTable()
y.field_names = ["Vectorizer","Optimal-eps"]
y.add_row(['AVG W2vec','9'])
y.add_row(['TFIDF W2vec','7'])
print(y)
```

```
+-------------+-------------+
|  Vectorizer | Optimal-eps |
+-------------+-------------+
|  AVG W2vec  |      9      |
| TFIDF W2vec |      7      |
+-------------+-------------+
```