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 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
```

```
D:\AAnaconda\lib\site-packages\gensim\utils.py:1212: UserWarning: detected Windows; aliasing chunkize t
o chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

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""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", 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 (100000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995

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='quickso
rt', 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]:

```
(87775, 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]:

```
87.775
```

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	
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	12248
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128

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()
```

```
(87773, 10)
```

Out[13]:

```
1    73592
0    14181
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 har
d to find any chicken products made in the USA but they are out there, but this one isnt.  Its too bad
too because its a good product but I wont take any chances till they know what is going on with the chi
na imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to it.
Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy the candy
again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of the
fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are really sm
all in size. They are great for training. You can give your dog several of these without worrying about
him over eating. Amazon's price was much more reasonable than any other retailer. You can buy a 1 pound
bag on Amazon for almost the same price as a 6 ounce bag at other retailers. It's definitely worth it t
o buy a big bag if your dog eats them a lot.
==================================================
```

In [15]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

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

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-elem
ent
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 har
d to find any chicken products made in the USA but they are out there, but this one isnt.  Its too bad
too because its a good product but I wont take any chances till they know what is going on with the chi
na imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to it.
Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy the candy
again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of the
fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are really sm
all in size. They are great for training. You can give your dog several of these without worrying about
him over eating. Amazon's price was much more reasonable than any other retailer. You can buy a 1 pound
bag on Amazon for almost the same price as a 6 ounce bag at other retailers. It's definitely worth it t
o buy a big bag if your dog eats them a lot.
```

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)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

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 very har
d to find any chicken products made in the USA but they are out there, but this one isnt.  Its too bad
too because its a good product but I wont take any chances till they know what is going on with the chi
na 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)
```

```
was way to hot for my blood took a bite and did a jig lol
```

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', "you'r
e", "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', 't
heir',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'whil
e', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'bef
ore', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'a
gain', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each
', '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', "doesn
't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't",
'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't",
'weren', "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%|██████████| 87773/87773 [00:40<00:00, 2143.42it/s]
```

In [77]:

```
preprocessed_reviews[1500:1502]
final['Cleaned_Text']=preprocessed_reviews
```

[3.2] Preprocessing Review Summary

In [196]:

```
## 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', 'aaaaaaaaaaaaaa', 'aaaaaaahhhhhh',
'aaaaaaarrrrrggghhh', 'aaaaaawwwwwwwwww', 'aaaaah']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 54904)
the number of unique words  54904
```

[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/skl
earn.feature_extraction.text.CountVectorizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your choice
```

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [28]:

```
tf_idf_vect = TfidfVectorizer(min_df=5)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[5:20])
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])
#print(tf_idf_vect.get_feature_names())
```

```
some sample features(unique words in the corpus) ['abandon', 'abandoned', 'abc', 'abdomen', 'abdominal'
, 'abide', 'abilities', 'ability', 'abit', 'able', 'abnormal', 'abnormalities', 'abomination', 'abrasiv
e', 'abroad']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (87773, 16545)
the number of unique words including both unigrams and bigrams  16545
```

[4.4] Word2Vec

In [0]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [0]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
```

```
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own
w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.9946032166481018), ('e
xcellent', 0.9944332838058472), ('especially', 0.9941144585609436), ('baked', 0.9940600395202637), ('sa
lted', 0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.9936816692352295), ('health
y', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.9992750883102417), ('
de', 0.9992610216140747), ('miss', 0.9992451071739197), ('melitta', 0.999218761920929), ('choice', 0.99
92102384567261), ('american', 0.9991837739944458), ('beef', 0.9991780519485474), ('finish', 0.999156713
4857178)]
```

In [0]:

```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby', 'used'
, 'ca', 'not', 'beat', 'great', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call
', 'instead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows', 'beautif
ully', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'like', 'tv', 'computer', 'really', 'goo
d', 'idea', 'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'made']
```

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

[4.4.1.1] Avg W2v

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 3
00 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%|██████████████████████████████████████████████████| 4986/4986 [00:03<00:
00, 1330.47it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:

```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:

```python
# 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 corpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████| 4986/4986 [00:20<00
:00, 245.63it/s]
```

[5] Assignment 11: Truncated SVD

1. **Apply Truncated-SVD on only this feature set:**

   - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)

   - **Procedure:**
     - Take top 2000 or 3000 features from tf-idf vectorizers using idf_ score.
     - You need to calculate the co-occurrence matrix with the selected features (Note: X.X^T doesn't give the co-occurrence matrix, it returns the covariance matrix, check these bolgs blog-1, blog-2 for more information)
     - You should choose the n_components in truncated svd, with maximum explained variance. Please search on how to choose that and implement them. (hint: plot of cumulative explained variance ratio)
     - After you are done with the truncated svd, you can apply K-Means clustering and choose the best number of clusters based on elbow method.
     - Print out wordclouds for each cluster, similar to that in previous assignment.
     - You need to write a function that takes a word and returns the most similar words using cosine similarity between the vectors(vector: a row in the matrix after truncatedSVD)

Truncated-SVD

[5.1] Taking top features from TFIDF, SET 2

In [36]:

```python
#Sample data
sentences = ["abc def ijk pqr","pqr klm opq","lmn pqr xyz abc def pqr abc"]
tf_idf_vect = TfidfVectorizer()
```

```
response = tf_idf_vect.fit_transform(sentences)
idf_score = tf_idf_vect.idf_  # obtaining the idf score from TFIDFVECTORIZER
feature_names  = tf_idf_vect.get_feature_names()
idfscore_feat=[]
for i in range(len(idf_score)):
    idfscore_feat.append([idf_score[i],feature_names[i]])
idfscore_feat.sort()
idfscore_feat=idfscore_feat[:15]
#some top features in idfscore_feat list
for i in idfscore_feat:
    print(i)
```

```
[1.0, 'pqr']
[1.2876820724517808, 'abc']
[1.2876820724517808, 'def']
[1.6931471805599454, 'ijk']
[1.6931471805599454, 'klm']
[1.6931471805599454, 'lmn']
[1.6931471805599454, 'opq']
[1.6931471805599454, 'xyz']
```

In [52]:

```
#Actual data
tf_idf_vect = TfidfVectorizer(min_df=10)
response = tf_idf_vect.fit_transform(preprocessed_reviews)
idf_score = tf_idf_vect.idf_  # obtaining the idf score from TFIDFVECTORIZER
feature_names  = tf_idf_vect.get_feature_names()
idfscore_feat=[]
for i in range(len(idf_score)):
    idfscore_feat.append([idf_score[i],feature_names[i]])
idfscore_feat.sort()
idfscore_feat=idfscore_feat[:3000]
#some top features in idfscore_feat list
for i in idfscore_feat[:10]:
    print(i)
```

```
[1.605378462881722, 'not']
[2.198111282636449, 'like']
[2.3127138563535645, 'good']
[2.412215882632696, 'great']
[2.500358842384766, 'one']
[2.516371541325381, 'taste']
[2.591809893272974, 'would']
[2.6538627467056726, 'product']
[2.681394067801694, 'love']
[2.697064695098047, 'flavor']
```

[5.2] Calulation of Co-occurrence matrix

In [51]:

```
#sample CORPUS
from tqdm import tqdm
n_neighbor = 2
occ_matrix = np.zeros((3,3))
top_features = []
for i in range(3):
    top_features.append(idfscore_feat[i][1])
print(top_features)
for row in sentences:
    words_in_row = row.split()
    for index,word in enumerate(words_in_row):
        if word in top_features:
            for j in range(max(index-n_neighbor,0),min(index+n_neighbor,len(words_in_row)-1) + 1):
                if words_in_row[j] in top_features and words_in_row[j]!=word:
                    occ_matrix[top_features.index(word),top_features.index(words_in_row[j])] += 1
                else:
                    continue
        else:
            continue
```

```
print(occ_matrix)
```

```
['pqr', 'abc', 'def']
[[0. 3. 2.]
 [3. 0. 3.]
 [2. 3. 0.]]
```

#Co-occurence matrix for actual data #Converted to Raw cell to avoid print statement from tqdm import tqdm n_neighbor = 5 occ_matrix = np.zeros((3000,3000)) top_features = [] for i in range(3000): top_features.append(idfscore_feat[i][1]) #print(top_features) for row in preprocessed_reviews: words_in_row = row.split() for index,word in enumerate(words_in_row): if word in top_features: for j in range(max(index-n_neighbor,0),min(index+n_neighbor,len(words_in_row)-1) + 1): if words_in_row[j] in top_features and words_in_row[j]!=word: occ_matrix[top_features.index(word),top_features.index(words_in_row[j])] += 1 else: continue else: continue #print(occ_matrix)

[5.3] Finding optimal value for number of components (n) to be retained.

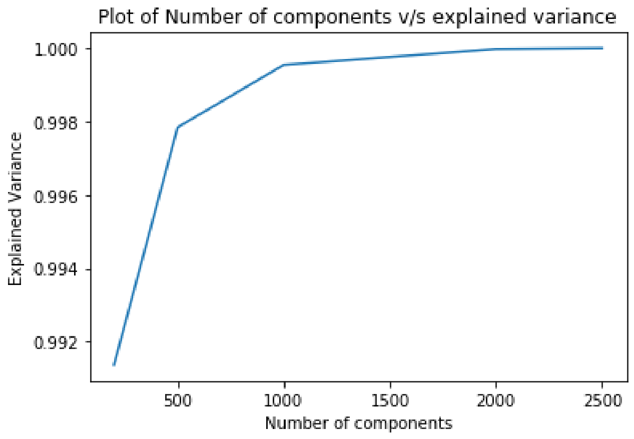
In [54]:

```
from sklearn.decomposition import TruncatedSVD
n_comp = [200,500,1000,2000,2500] # list containing different values of components
explained = [] # explained variance ratio for each component of Truncated SVD
for x in n_comp:
    svd = TruncatedSVD(n_components=x)
    svd.fit(occ_matrix)
    explained.append(svd.explained_variance_ratio_.sum())
    print("Number of components = %r and explained variance = %r"%(x,svd.explained_variance_ratio_.sum(
)))
plt.plot(n_comp, explained)
plt.xlabel('Number of components')
plt.ylabel("Explained Variance")
plt.title("Plot of Number of components v/s explained variance")
```

```
Number of components = 200 and explained variance = 0.9913611811134232
Number of components = 500 and explained variance = 0.9978390982997309
Number of components = 1000 and explained variance = 0.9995409547474605
Number of components = 2000 and explained variance = 0.9999742434582516
Number of components = 2500 and explained variance = 0.9999972741148713
```

Out[54]:

```
Text(0.5, 1.0, 'Plot of Number of components v/s explained variance')
```



In [56]:

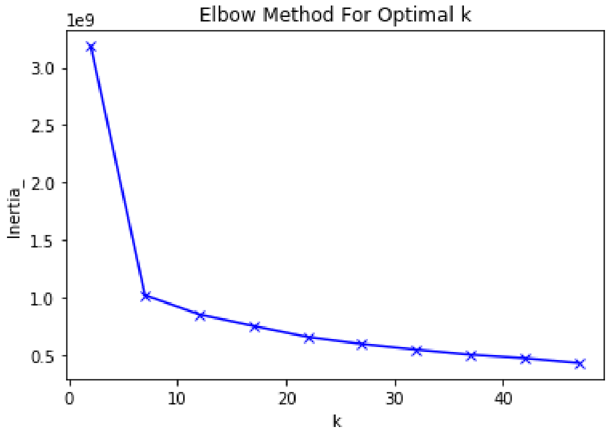
```
svd = TruncatedSVD(n_components=200)
tfidf_svd = svd.fit_transform(occ_matrix)
print(svd.explained_variance_ratio_.sum())
```

```
0.9913619415882681
```

[5.4] Applying k-means clustering

In [57]:

```
# Please write all the code with proper documentation
from sklearn.cluster import KMeans
K = range(2,50,5)
Sum_of_squared_distances = []
for k in K:
    km = KMeans(n_clusters=k,n_jobs=-1)
    km = km.fit(tfidf_svd)
    Sum_of_squared_distances.append(km.inertia_)
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Inertia_')
plt.title('Elbow Method For Optimal k')
plt.show()
```



In [70]:

```
Kmean_tfidf = KMeans(n_clusters=7, n_jobs=-1).fit(tfidf_svd)
#centers = Kmean_tfidf.cluster_centers_
label = Kmean_tfidf.labels_.tolist()
#final['kmeans_labels'] = label
```

In [80]:

```
cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []
cluster5 = []
cluster6 = []
cluster7 = []
X1 =final['Cleaned_Text'].values
for i in range(Kmean_tfidf.labels_.shape[0]):
    if Kmean_tfidf.labels_[i] == 0:
        cluster1.append(X1[i])
    elif Kmean_tfidf.labels_[i] == 1:
        cluster2.append(X1[i])
    elif Kmean_tfidf.labels_[i] == 2:
        cluster3.append(X1[i])
    elif Kmean_tfidf.labels_[i] == 3:
        cluster4.append(X1[i])
    elif Kmean_tfidf.labels_[i] == 4:
        cluster5.append(X1[i])
    elif Kmean_tfidf.labels_[i] == 5:
        cluster6.append(X1[i])
    else :
        cluster7.append(X1[i])

# Number of reviews in different clusters
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))
print("\nNo. of reviews in Cluster 7 : ",len(cluster7))
```

```
No. of reviews in Cluster 1 :  2485

No. of reviews in Cluster 2 :  10

No. of reviews in Cluster 3 :  1

No. of reviews in Cluster 4 :  355

No. of reviews in Cluster 5 :  29

No. of reviews in Cluster 6 :  119

No. of reviews in Cluster 7 :  1
```

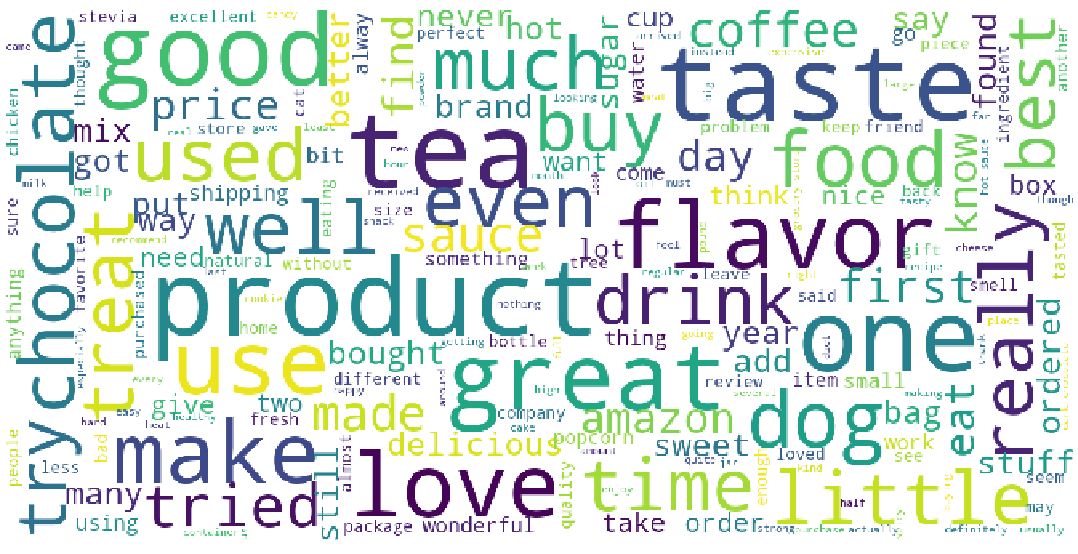
[5.5] Wordclouds of clusters obtained in the above section

In [83]:

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
stopwords = set(STOPWORDS)

def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(1, figsize=(12, 12))
    plt.axis('off')


    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()
```

In [84]:

```
show_wordcloud(cluster1,"Cluster 1")
```



Cluster 1

In [85]:

```
show_wordcloud(cluster2,"Cluster 2")
```

Cluster 2

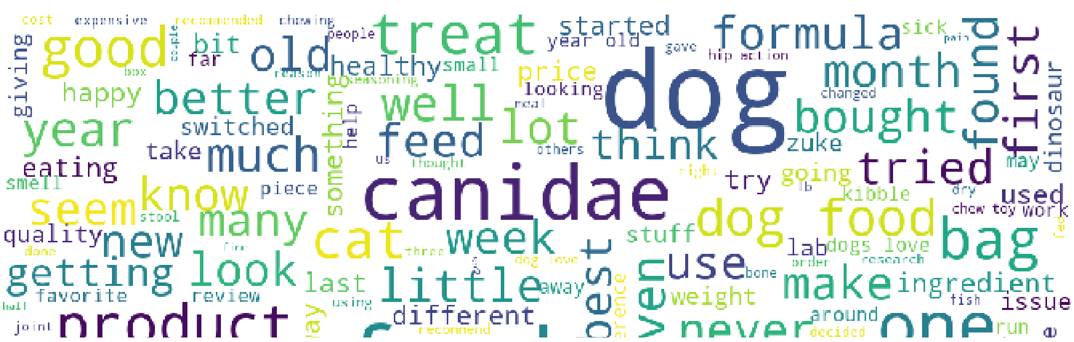
In [86]:

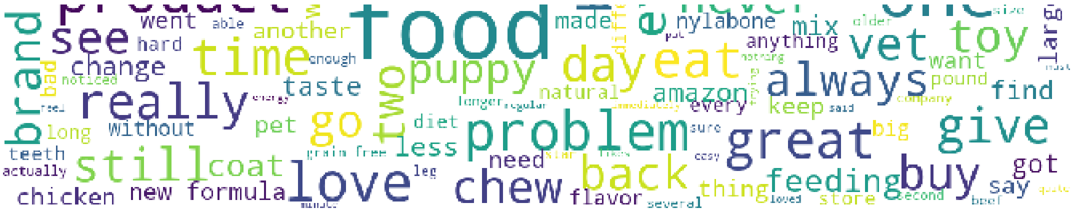
```
show_wordcloud(cluster3,"Cluster 3")
```

Cluster 3

In [87]:

```
show_wordcloud(cluster4,"Cluster 4")
```

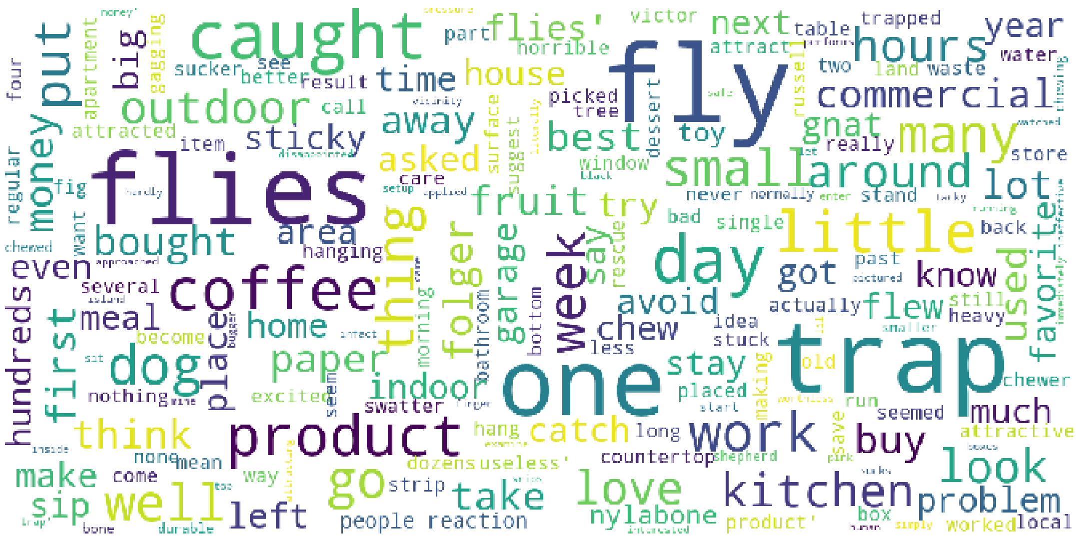




Cluster 4

In [88]:

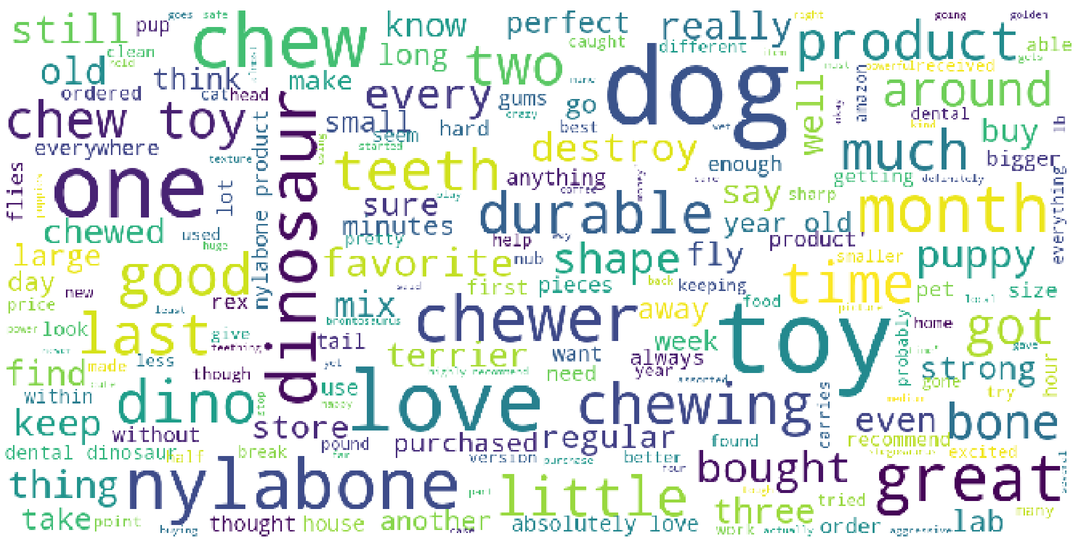
```
show_wordcloud(cluster5,"Cluster 5")
```



Cluster 5

In [89]:

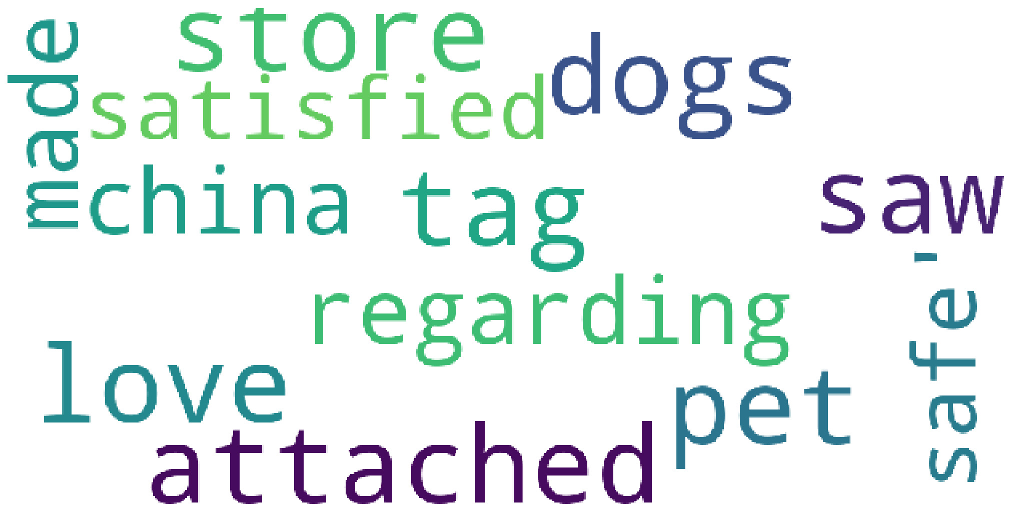
```
show_wordcloud(cluster6,"Cluster 6")
```



Cluster 6

In [90]:

```
show_wordcloud(cluster7,"Cluster 7")
```



Cluster 7

[5.6] Function that returns most similar words for a given word.

In [101]:

```
from sklearn.metrics.pairwise import cosine_similarity
def similar_word_10(word):
    similarity = cosine_similarity(occ_matrix)
    #print(similarity)
    word_vect = similarity[top_features.index(word)]
    #print(word_vect)
    print("Similar Words for word:",word)
    index = word_vect.argsort()[::-1][1:11]
    for j in range(len(index)):
        print("\n", top_features[index[j]], word_vect[j])
```

In [102]:

```
similar_word_10('brownie')
```

```
Similar Words for word: brownie

 cake 0.5739862456780668

 brownies 0.6299846059054564

 cookie 0.666984499815649

 muffin 0.638353860870048

 cookies 0.6659832079442504

 bread 0.6899542108170942

 pancake 0.6419733334644928

 trail 0.6353346993418154

 mixes 0.6577498892913883

 yummy 0.6866760698631598
```

In [107]:

```
similar_word_10('coffe')
```

```
Similar Words for word: coffe

 espresso 0.6403461649483765
```

```
 starbucks 0.6603336806562716

 coffee 0.7028575017888523

 cups 0.6876208413755817

 blend 0.7383715022648847

 pod 0.7762777544727628

 blends 0.7030163808487823

 coffees 0.7090099570654037

 flavored 0.727213059310215

 strong 0.7776976012130973
```

In [108]:

```
similar_word_10('dog')
```

```
Similar Words for word: dog

 cat 0.5381728468693134

 dry 0.6315539771854709

 baby 0.6478094399288632

 junk 0.6069484454060059

 cats 0.6934777843725866

 feed 0.6130100400149878

 eats 0.6804676791688363

 canned 0.6659228430326952

 allergies 0.6811696250955686

 pets 0.5977253171818074
```

[6] Conclusions

In [0]:

```
# Please write down few lines about what you observed from this assignment.
# Also please do mention the optimal values that you obtained for number of components & number of clus
ters.
1. TFIDF Vectorizer with min_df = 5 is applied on the preprocessed reviews and top 3000 features are se
lected using idf_ score.
2. After plotting a graph between n_components and explained variance ratio sum , the n_components is c
hosen as 200, for which the explained variance ratio is 0.99.
3. Truncated SVD is applied for the 3000 features and number of clusters is estimated as 7 from the elb
ow curve.
4. Similar words are found for a given word using cosine similarity of co-occurence matrix.
```