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 [0]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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

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

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

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

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

from tqdm import tqdm
import os
```

In [0]:
```
!pip install paramiko
```

```
Collecting paramiko
  Downloading https://files.pythonhosted.org/packages/cf/ae/94e70d49044
ccc234bfdba20114fa947d7ba6eb68a2e452d89b920e62227/paramiko-2.4.2-py2.py
3-none-any.whl (193kB)
     |████████████████████████████████| 194kB 2.9MB/s
Collecting bcrypt>=3.1.3 (from paramiko)
  Downloading https://files.pythonhosted.org/packages/d0/79/79a4d167a31
cc206117d9b396926615fa9c1fdbd52017bcced80937ac501/bcrypt-3.1.6-cp34-abi
3-manylinux1_x86_64.whl (55kB)
     |████████████████████████████████| 61kB 22.4MB/s
Collecting cryptography>=1.5 (from paramiko)
  Downloading https://files.pythonhosted.org/packages/5b/12/b0409a94dad
```

```
366d98a8eee2a77678c7a73aafd8c0e4b835abea634ea3896/cryptography-2.6.1-cp
34-abi3-manylinux1_x86_64.whl (2.3MB)
     |████████████████████████████████| 2.3MB 47.9MB/s
Collecting pynacl>=1.0.1 (from paramiko)
  Downloading https://files.pythonhosted.org/packages/27/15/2cd0a203f31
8c2240b42cd9dd13c931ddd61067809fee3479f44f086103e/PyNaCl-1.3.0-cp34-abi
3-manylinux1_x86_64.whl (759kB)
     |████████████████████████████████| 768kB 45.1MB/s
Requirement already satisfied: pyasn1>=0.1.7 in /usr/local/lib/python3.
6/dist-packages (from paramiko) (0.4.5)
Requirement already satisfied: cffi>=1.1 in /usr/local/lib/python3.6/di
st-packages (from bcrypt>=3.1.3->paramiko) (1.12.3)
Requirement already satisfied: six>=1.4.1 in /usr/local/lib/python3.6/d
ist-packages (from bcrypt>=3.1.3->paramiko) (1.12.0)
Collecting asn1crypto>=0.21.0 (from cryptography>=1.5->paramiko)
  Downloading https://files.pythonhosted.org/packages/ea/cd/35485615f45
f30a510576f1a56d1e0a7ad7bd8ab5ed7cdc600ef7cd06222/asn1crypto-0.24.0-py
2.py3-none-any.whl (101kB)
     |████████████████████████████████| 102kB 29.0MB/s
Requirement already satisfied: pycparser in /usr/local/lib/python3.6/di
st-packages (from cffi>=1.1->bcrypt>=3.1.3->paramiko) (2.19)
Installing collected packages: bcrypt, asn1crypto, cryptography, pynac
l, paramiko
Successfully installed asn1crypto-0.24.0 bcrypt-3.1.6 cryptography-2.6.
1 paramiko-2.4.2 pynacl-1.3.0
```

In [0]:

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

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 50
0000 data points
# you can change the number to any other number based on your computing
 power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Sco
re != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points
```

```python
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 LIMIT 5000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<3 a negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (5000, 10)
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomin
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

In [0]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

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

```
(80668, 7)
```

Out[0]:

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

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

Out[0]:

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

In [0]:
```
display['COUNT(*)'].sum()
```
Out[0]:
```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

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

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

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

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

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

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

Out[0]:
```
(4986, 10)
```

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

Out[0]:
```
99.72
```

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

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

display.head()
```

Out[0]:

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

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

In [0]:
```
#Before starting the next phase of preprocessing lets see the number of
 entries left
print(final.shape)

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

```
(4986, 10)
```

Out[0]:
```
1    4178
0     808
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [0]:

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

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

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

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

```
Why is this $[...] when the same product is available for $[...] here?<
br />http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<
br /><br />The Victor M380 and M502 traps are unreal, of course -- tota
l fly genocide. Pretty stinky, but only right nearby.
==================================================
```

```
I recently tried this flavor/brand and was surprised at how delicious t
hese chips are.  The best thing was that there were a lot of "brown" ch
ips in the bsg (my favorite), so I bought some more through amazon and
shared with family and friends.  I am a little disappointed that there
are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion fl
avor because they do not seem to be as salty, and the onion flavor is b
etter.  If you haven't eaten Kettle chips before, I recommend that you
try a bag before buying bulk.  They are thicker and crunchier than Lays
but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I'm sorry; but t
hese reviews do nobody any good beyond reminding us to look  before ord
ering.<br /><br />These are chocolate-oatmeal cookies.  If you don't li
ke that combination, don't order this type of cookie.  I find the combo
quite nice, really.  The oatmeal sort of "calms" the rich chocolate fla
vor and gives the cookie sort of a coconut-type consistency.  Now let's
also remember that tastes differ; so, I've given my opinion.<br /><br /
>Then, these are soft, chewy cookies -- as advertised.  They are not "c
rispy" cookies, or the blurb would say "crispy," rather than "chewy."
I happen to like raw cookie dough; however, I don't see where these tas
te like raw cookie dough.  Both are soft, however, so is this the confu
sion?  And, yes, they stick together.  Soft cookies tend to do that.  T
hey aren't individually wrapped, which would add to the cost.  Oh yeah,
chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if you
want something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you
want a cookie that's soft, chewy and tastes like a combination of choco
late and oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />Thi
s k cup is great coffee.  dcaf is very good as well
==================================================
```

In [0]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
```

```python
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<
br /> /><br />The Victor M380 and M502 traps are unreal, of course -- t
otal fly genocide. Pretty stinky, but only right nearby.
```

In [0]:

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

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

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

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

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

```
Why is this $[...] when the same product is available for $[...] here?
/>The Victor M380 and M502 traps are unreal, of course -- total fly gen
ocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious t
hese chips are.  The best thing was that there were a lot of "brown" ch
ips in the bsg (my favorite), so I bought some more through amazon and
shared with family and friends.  I am a little disappointed that there
```

```
are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion fl
avor because they do not seem to be as salty, and the onion flavor is b
etter.  If you haven't eaten Kettle chips before, I recommend that you
try a bag before buying bulk.  They are thicker and crunchier than Lays
but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I'm sorry; but t
hese reviews do nobody any good beyond reminding us to look  before ord
ering.These are chocolate-oatmeal cookies.  If you don't like that comb
ination, don't order this type of cookie.  I find the combo quite nice,
really.  The oatmeal sort of "calms" the rich chocolate flavor and give
s the cookie sort of a coconut-type consistency.  Now let's also rememb
er that tastes differ; so, I've given my opinion.Then, these are soft,
chewy cookies -- as advertised.  They are not "crispy" cookies, or the
blurb would say "crispy," rather than "chewy."  I happen to like raw co
okie dough; however, I don't see where these taste like raw cookie doug
h.  Both are soft, however, so is this the confusion?  And, yes, they s
tick together.  Soft cookies tend to do that.  They aren't individually
wrapped, which would add to the cost.  Oh yeah, chocolate chip cookies
tend to be somewhat sweet.So, if you want something hard and crisp, I s
uggest Nabiso's Ginger Snaps.  If you want a cookie that's soft, chewy
and tastes like a combination of chocolate and oatmeal, give these a tr
y.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cu
p is great coffee.  dcaf is very good as well
```

In [0]:

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

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

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
```

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

In [0]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I am sorry; but
these reviews do nobody any good beyond reminding us to look  before or
dering.<br /><br />These are chocolate-oatmeal cookies.  If you do not
like that combination, do not order this type of cookie.  I find the co
mbo quite nice, really.  The oatmeal sort of "calms" the rich chocolate
flavor and gives the cookie sort of a coconut-type consistency.  Now le
t is also remember that tastes differ; so, I have given my opinion.<br
/><br />Then, these are soft, chewy cookies -- as advertised.  They are
not "crispy" cookies, or the blurb would say "crispy," rather than "che
wy."  I happen to like raw cookie dough; however, I do not see where th
ese taste like raw cookie dough.  Both are soft, however, so is this th
e confusion?  And, yes, they stick together.  Soft cookies tend to do t
hat.  They are not individually wrapped, which would add to the cost.
Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />S
o, if you want something hard and crisp, I suggest Nabiso is Ginger Sna
ps.  If you want a cookie that is soft, chewy and tastes like a combina
tion of chocolate and oatmeal, give these a try.  I am here to place my
second order.
==================================================
```

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

```
Why is this $[...] when the same product is available for $[...] here?<
br /> /><br />The Victor  and  traps are unreal, of course -- total fly
genocide. Pretty stinky, but only right nearby.
```

In [0]:

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

```
Wow So far two two star reviews One obviously had no idea what they wer
e ordering the other wants crispy cookies Hey I am sorry but these revi
ews do nobody any good beyond reminding us to look before ordering br b
r These are chocolate oatmeal cookies If you do not like that combinati
on do not order this type of cookie I find the combo quite nice really
The oatmeal sort of calms the rich chocolate flavor and gives the cooki
e sort of a coconut type consistency Now let is also remember that tast
es differ so I have given my opinion br br Then these are soft chewy co
okies as advertised They are not crispy cookies or the blurb would say
crispy rather than chewy I happen to like raw cookie dough however I do
not see where these taste like raw cookie dough Both are soft however s
o is this the confusion And yes they stick together Soft cookies tend t
o do that They are not individually wrapped which would add to the cost
Oh yeah chocolate chip cookies tend to be somewhat sweet br br So if yo
u want something hard and crisp I suggest Nabiso is Ginger Snaps If you
want a cookie that is soft chewy and tastes like a combination of choco
late and oatmeal give these a try I am here to place my second order
```

In [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in
 the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
```

```
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [0]:

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

```
100%|██████████| 4986/4986 [00:01<00:00, 3137.37it/s]
```

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

Out[0]:
```
'wow far two two star reviews one obviously no idea ordering wants cris
py cookies hey sorry reviews nobody good beyond reminding us look order
ing chocolate oatmeal cookies not like combination not order type cooki
e find combo quite nice really oatmeal sort calms rich chocolate flavor
gives cookie sort coconut type consistency let also remember tastes dif
fer given opinion soft chewy cookies advertised not crispy cookies blur
b would say crispy rather chewy happen like raw cookie dough however no
t see taste like raw cookie dough soft however confusion yes stick toge
ther soft cookies tend not individually wrapped would add cost oh yeah
chocolate chip cookies tend somewhat sweet want something hard crisp su
ggest nabiso ginger snaps want cookie soft chewy tastes like combinatio
n chocolate oatmeal give try place second order'
```

[3.2] Preprocessing Review Summary

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

[4] Featurization

[4.1] BAG OF WORDS

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

final_counts = count_vect.transform(preprocessed_reviews)
```

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

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abb
ott', 'abby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [0]:

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

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

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [0]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape
())
print("the number of unique words including both unigrams and bigrams "
, final_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'a
ble find', 'able get', 'absolute', 'absolutely', 'absolutely deliciou
s', 'absolutely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

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

In [0]:
```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
```

```
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors
-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_trai
n_w2v = True, to train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wond
erful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('espec
ially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted',
0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.99
```

```
36816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('p
opcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.99
92451071739197), ('melitta', 0.999218761920929), ('choice', 0.999210238
4567261), ('american', 0.9991837739944458), ('beef', 0.999178051948547
4), ('finish', 0.9991567134857178)]
```

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

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'st
inky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'receiv
ed', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'ins
tead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use',
'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fu
n', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea',
'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'mad
e']
```

[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, yo
u 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/re
view
```

```
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████████████████████████████████████████████
██████████| 4986/4986 [00:03<00:00, 1330.47it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:

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

In [0]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
```

```
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████| 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)

In [2]:
```
# Load the Drive helper and mount
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

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

In [3]:
```
cd drive/My Drive
```

```
/content/drive/My Drive
```

In [4]:
```
con1 = sqlite3.connect('final.sqlite')

# Eliminating neutral reviews i.e. those reviews with Score = 3
final = pd.read_sql_query(" SELECT * FROM Reviews WHERE Score != 3 ", con1)

print(final.shape)
final.head()
```

```
(364171, 12)
```

Out[4]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
3	138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
4	138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [0]:
```
# Sort data based on time
final["Time"] = pd.to_datetime(final["Time"], unit = "s")
```

```
final = final.sort_values(by = "Time")
```

In [20]:
```
# Select first 50k data-points
final = final.iloc[:50000,:]
final.shape
```

Out[20]: (50000, 12)

In [21]:
```
# Data
X = final["CleanedText"]
X.shape
```

Out[21]: (50000,)

Truncated-SVD

[5.1] Taking top features from TFIDF, SET 2

In [22]:
```
tfidf = TfidfVectorizer(use_idf = True, max_df = 0.80)
feat = tfidf.fit_transform(X)
feat.shape
```

Out[22]: (50000, 27002)

In [24]:
```
# Standardization
from sklearn.preprocessing import StandardScaler
std = StandardScaler(with_mean = False)
std_data = std.fit_transform(feat)
std_data
```

Out[24]:
```
<50000x27002 sparse matrix of type '<class 'numpy.float64'>'
	with 1484064 stored elements in Compressed Sparse Row format>
```

In [26]:
```
# List of vocabulary
list(tfidf.vocabulary_.keys())[0:10]
```

Out[26]:
```
['witti',
 'littl',
 'book',
 'make',
 'son',
 'laugh',
 'loud',
 'recit',
 'car',
 'drive']
```

In [27]:
```
# List of vocabulary values
list(tfidf.vocabulary_.values())[0:10]
```

Out[27]:
```
[26391, 13687, 2616, 14164, 21891, 13279, 13867, 19392, 3528, 7090]
```

In [28]:
```
# Get feature names from tfidf
features = tfidf.get_feature_names()
# feature weights based on idf score
coef = tfidf.idf_
# Store features with their idf score in a dataframe
coeff_df = pd.DataFrame({'Features' : features, 'Idf_score' : coef})
coeff_df = coeff_df.sort_values("Idf_score", ascending = True)[:2000]
print("shape of selected features :", coeff_df.shape)
print("Top 5 features :\n\n",coeff_df[0:10])
```

```
shape of selected features : (2000, 2)
Top 5 features :

       Features  Idf_score
23388     tast   2.183929
13562     like   2.227603
9954      good   2.342489
10188    great   2.364709
13880     love   2.418011
8741    flavor   2.467872
16581      one   2.516879
18607  product   2.538627
```

```
24353      tri   2.588675
25249      use   2.598518
```

[5.2] Calulation of Co-occurrence matrix

In [29]:
```
# https://cs224d.stanford.edu/lecture_notes/notes1.pdf
# co-occurence matrix
co_occurence_matrix = np.zeros((len(coeff_df), len(coeff_df)))
print(co_occurence_matrix.shape)
df = pd.DataFrame(co_occurence_matrix, index = coeff_df["Features"], co
lumns = coeff_df["Features"])
df.shape
```

```
(2000, 2000)
```

Out[29]:
```
(2000, 2000)
```

In [41]:
```
# Calculate Co-Occurrence Matrix
# with windows size 4 in forward and backward pass
%time
window_size = 4
for sent in final["CleanedText"]:
    word = sent.split(" ")
    for i, d in enumerate(word):
        for j in range(max(i - window_size, 0), min(i + window_size, le
n(word))):
            if (word[i] != word[j]):
                try:
                    df.loc[word[i], word[j]] += 1
                    df.loc[word[j], word[i]] += 1
                except:
                    pass
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 4.77 µs
```

In [42]:
```
df.head()
```

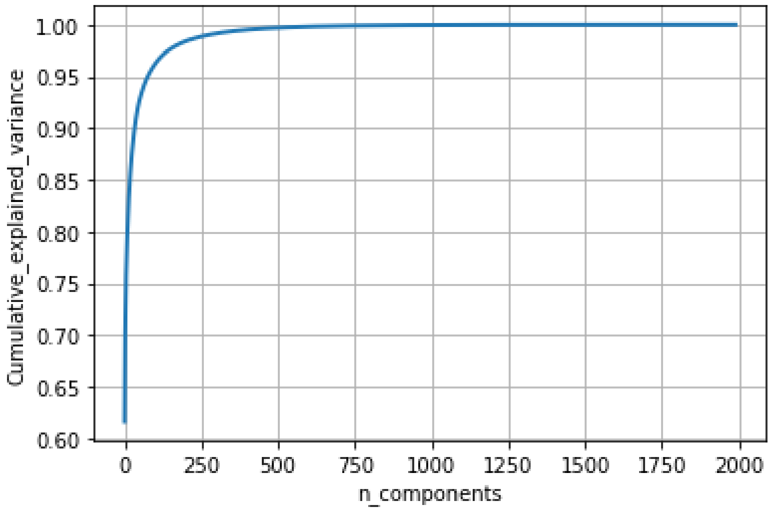
Out[42]:

Features	tast	like	good	great	love	flavor	one	product	tri	use	...	cor
Features												
tast	0.0	9065.0	6348.0	6140.0	2278.0	2846.0	2004.0	2100.0	1621.0	1525.0	...	
like	9065.0	0.0	2529.0	1606.0	1674.0	3862.0	2495.0	1980.0	2092.0	1574.0	...	
good	6348.0	2529.0	0.0	1586.0	1166.0	2768.0	1681.0	2361.0	1246.0	1211.0	...	
great	6140.0	1606.0	1586.0	0.0	1856.0	2938.0	1063.0	3603.0	985.0	1413.0	...	
love	2278.0	1674.0	1166.0	1856.0	0.0	2085.0	1409.0	1921.0	1419.0	1033.0	...	

5 rows × 2000 columns

In [43]:

```
# TrucatedSVD
from sklearn.decomposition import TruncatedSVD
ts = TruncatedSVD(n_components = 1990)
ts_data = ts.fit_transform(df)

percentage_var_explained = ts.explained_variance_ / np.sum(ts.explained
_variance_)

cum_var_explained = np.cumsum(percentage_var_explained)

# Plot the PCA spectrum
plt.figure(1, figsize=(6, 4))

plt.clf()
plt.plot(cum_var_explained, linewidth = 2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```

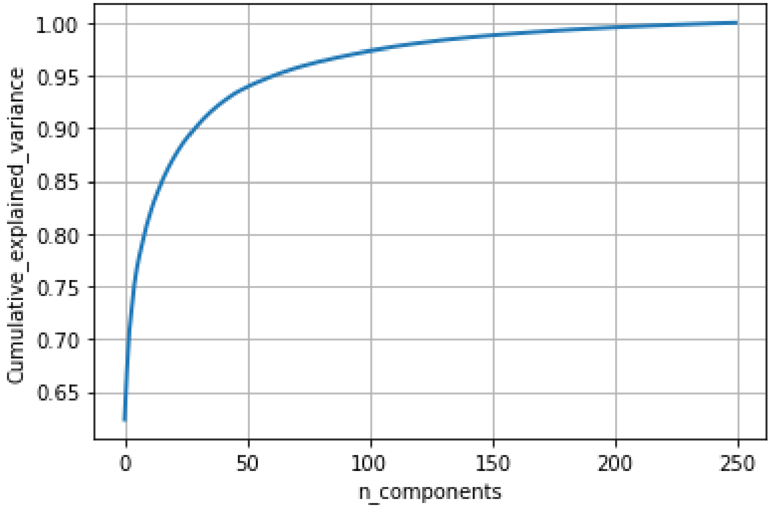


consider 250 bcz nearly above 97 % cumulative variance can be observed for that component value

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

In [44]:
```
# TrucatedSVD
from sklearn.decomposition import TruncatedSVD
ts = TruncatedSVD(n_components = 250)
ts_data = ts.fit_transform(df)

percentage_var_explained = ts.explained_variance_ / np.sum(ts.explained
_variance_)

cum_var_explained = np.cumsum(percentage_var_explained)

# Plot the PCA spectrum
plt.figure(1, figsize=(6, 4))

plt.clf()
```

```
plt.plot(cum_var_explained, linewidth = 2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```

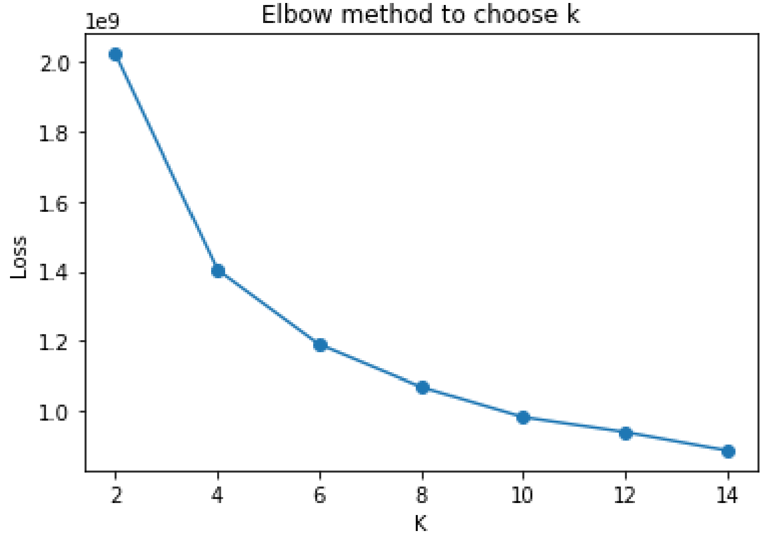


[5.4] Applying k-means clustering

In [0]:
```
# Elbow method to find K
def find_optimal_k(data):
    loss = []
    k = list(range(2, 15, 2))
    for noc in k:
        model = KMeans(n_clusters = noc)
        model.fit(data)
        loss.append(model.inertia_)
    plt.plot(k, loss, "-o")
    plt.title("Elbow method to choose k")
    plt.xlabel("K")
```

```
    plt.ylabel("Loss")
    plt.show()
```

In [47]:
```
# Find best k using elbow method
from sklearn.cluster import KMeans
find_optimal_k(ts_data)
```



consider k value 6

In [0]:
```
# After applying truncated svd store data into dataframe
df = pd.DataFrame(ts_data)
```

In [49]:
```
# Data shape
df.shape
```

Out[49]: (2000, 250)

In [50]:
```
# K-means clustering
clf = KMeans(n_clusters = 6)
clf.fit(ts_data)
```

Out[50]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=6, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [51]:
```
# Assign each data-points with its correspondincg label
df["Cluster_labels"] = clf.labels_
df["Words"] = coeff_df["Features"].values
df.head()
```

Out[51]:

	0	1	2	3	4	5	
0	14814.893210	8099.498023	-2917.585684	-1252.037454	-1497.280937	1556.765371	477.66470
1	13699.515772	-6214.106686	-2456.979315	-341.233815	-1190.436030	573.890385	1032.54712
2	10805.604304	-3798.003452	-801.003918	-13.879903	-1166.905145	932.558293	-176.27461
3	10018.773681	-4217.973761	-75.468188	151.195548	-1148.876862	977.452439	-1087.27687
4	7926.299703	-865.313997	284.179464	-1343.237594	-170.093426	-742.426350	693.33256

5 rows × 252 columns

[5.5] Wordclouds of clusters obtained in the above section

In [0]:
```
# Plotting word cloud
from wordcloud import WordCloud, STOPWORDS
def plot_word_cloud(txt):
    # store each word from review
    cloud = " ".join(word for word in txt)
    # Remove duplicate words
    stopwords = set(STOPWORDS)
    # call built-in method WordCloud for creating an object for drawing
 a word cloud
    wordcloud = WordCloud(width = 1000, height = 600, background_color
='white', stopwords = stopwords).generate(cloud)
    # plot the WordCloud image
```

```
    plt.figure(figsize = (10, 8))
    plt.imshow(wordcloud, interpolation = 'bilinear')
    plt.axis("off")
    plt.title("World cloud of top words")
    plt.tight_layout(pad = 0)
    plt.show()
```

In [55]:

```
# print word cloud of each cluster
for i in range(clf.n_clusters):
    l = list()
    # Groups each label
    label = df.groupby(["Cluster_labels"]).groups[i]
    # store each word from a particular label in a list and pass it wor
d cloud method
    for j in range(len(label)):
        l.append(df.loc[label[j]]["Words"])
    print("total number of word in cluster {} is {}".format(i, len(labe
l)))
    plot_word_cloud(l)
```

```
total number of word in cluster 0 is 285
```

World cloud of top words

total number of word in cluster 1 is 25

World cloud of top words

total number of word in cluster 2 is 5

World cloud of top words

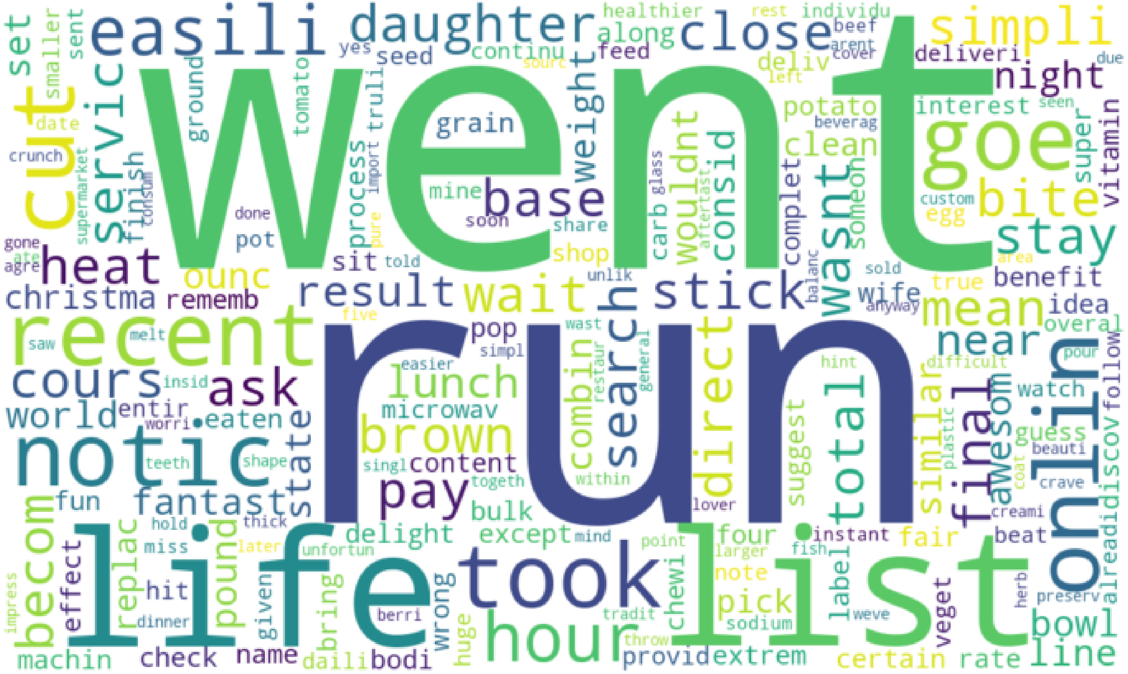
total number of word in cluster 3 is 94


```
total number of word in cluster 4 is 1
```

World cloud of top words


```
total number of word in cluster 5 is 1590
```

World cloud of top words



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

In [0]:
```
# Calculate cosine similarity
from sklearn.metrics import pairwise_distances
def cosine_similarity(word_index, total_results):
```

```
    # calculate pairwise distances from given word
    # The smaller the distance, the more similar the word
    dist = pairwise_distances(ts_data, ts_data[word_index:word_index +
1,:])
    # Store index of the distances
    indices = np.argsort(dist.flatten())[0:total_results]
    # Sort distances
    pdist = np.sort(dist.flatten())[0:total_results]
    # put indices at particular index of dataframe
    df_indices = list(df.index[indices])
    print("Most_Similar Words \t Distances")
    # Loop through indices and find match
    for i in range(len(indices)):
        if indices[i] == df.index[indices[i]]:
            print(df.Words.loc[indices[i]], "\t\t\t", dist[indices[i]])
```

In [60]:

```
# given index of a word
# find how similar words are from this index word
cosine_similarity(70, 10)
```

```
Most_Similar Words       Distances
purchas                          [0.]
bought                   [879.35894636]
alway                    [1250.0060528]
avail                    [1256.09770047]
got                      [1257.00504544]
```

In [62]:

```
cosine_similarity(25, 10)
```

```
Most_Similar Words       Distances
littl                    [0.]
well                     [2132.10146639]
even                     [2185.86008166]
also                     [2215.84175607]
way                      [2240.19490834]
think                    [2255.48280728]
nice                     [2278.00093295]
delici                   [2354.95233838]
```

```
lot                             [2383.42108125]
want                            [2389.42295875]
```

[6] Conclusions

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 clusters.

```
STEP-1:We have used 50k data-points and applied tfidf on top of
 it to vectorize text data and then selected top 2k features bas
ed on idf score.
STEP-2:We have calculated co-occurence matrix to store count of
 how often a features occur together in a context and then used
 truncatedsvd.
STEP-3:Used k-means clustering to group similar features togethe
r.
STEP-4:we calculated cosine similarity to get which words are mo
re similar to a given word.
STEP-5:optimal no.of components and clusters that i considered a
re 250,6
```