Apply Truncated SVD on Amazon Fine Food Reviews Dataset

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

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
```

```
C:\Users\HIMANSHU NEGI\Anaconda3\lib\site-packages\gensim\utils.py:1212: Use
rWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:

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

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

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rat
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[2]:

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

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	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 [5]:

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

Out[5]:

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

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	HelpfulnessDen
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
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 [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, k
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='firs
final.shape
```

Out[9]:

```
(4986, 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]:

```
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 [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	HelpfulnessDen
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

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()
```

```
(4986, 10)
```

Out[13]:

```
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 [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)
```

```
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 -- total fly genocide.
Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these
chips are.  The best thing was that there were a lot of "brown" chips in the
bsg (my favorite), so I bought some more through amazon and shared with fami
ly 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 t
hem better than the yogurt and green onion flavor because they do not seem t
o be as salty, and the onion flavor is better.  If you haven't eaten Kettle
chips before, I recommend that you try a bag before buying bulk.  They are t
hicker and crunchier than Lays but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they wer
e ordering; the other wants crispy cookies.  Hey, I'm sorry; but these revie
ws do nobody any good beyond reminding us to look  before ordering.<br /><br
/>These are chocolate-oatmeal cookies.  If you don't like that combination,
don't order this type of cookie.  I find the combo quite nice, really.  The
oatmeal sort of "calms" the rich chocolate flavor 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 "crispy" cookies, or the blurb would say "crisp
y," rather than "chewy."  I happen to like raw cookie dough; however, I do
n't see where these taste like raw cookie dough.  Both are soft, however, so
is this the confusion?  And, yes, they stick together.  Soft cookies tend to
do that.  They aren't individually wrapped, which would add to the cost.  Oh
yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if yo
u want something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you wa
nt a cookie that's soft, chewy and tastes like a combination of chocolate an
d oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k c
up is great coffee.  dcaf is very good as well
==================================================
```

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)
```

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

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-
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 genocide. Pret
ty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these
chips are.  The best thing was that there were a lot of "brown" chips in the
bsg (my favorite), so I bought some more through amazon and shared with fami
ly 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 t
hem better than the yogurt and green onion flavor because they do not seem t
o be as salty, and the onion flavor is better.  If you haven't eaten Kettle
chips before, I recommend that you try a bag before buying bulk.  They are t
hicker and crunchier than Lays but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they wer
e ordering; the other wants crispy cookies.  Hey, I'm sorry; but these revie
ws do nobody any good beyond reminding us to look  before ordering.These are
chocolate-oatmeal cookies.  If you don't like that combination, don't order
this type of cookie.  I find the combo quite nice, really.  The oatmeal sort
of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-
type consistency.  Now let's also remember that tastes differ; so, I've give
n 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 cookie dough; however, I don't see where these taste li
ke raw cookie dough.  Both are soft, however, so is this the confusion?  An
d, yes, they stick together.  Soft cookies tend to do that.  They aren't ind
ividually wrapped, which would add to the cost.  Oh yeah, chocolate chip coo
kies 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 t
astes like a combination of chocolate and oatmeal, give these a try.  I'm he
re to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup is
great coffee.  dcaf is very good as well
```

In [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)
```

```
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 combinatio
n, 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 cookie s
ort of a coconut-type consistency.  Now let is also remember that tastes dif
fer; so, I have given my opinion.<br /><br />Then, these are soft, chewy coo
kies -- as advertised.  They are not "crispy" cookies, or the blurb would sa
y "crispy," rather than "chewy."  I happen to like raw cookie dough; howeve
r, I do not see where these taste like raw cookie dough.  Both are soft, how
ever, so is this the confusion?  And, yes, they stick together.  Soft cookie
s tend to do that.  They are not individually wrapped, which would add to th
e cost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br
/>So, if you want something hard and crisp, I suggest Nabiso is Ginger Snap
s.  If you want a cookie that is soft, chewy and tastes like a combination o
f chocolate and oatmeal, give these a try.  I am here to place my second ord
er.
==================================================
```

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)
```

```
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 [20]:

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

```
Wow So far two two star reviews One obviously had no idea what they were ord
ering the other wants crispy cookies Hey I am sorry but these reviews do nob
ody any good beyond reminding us to look before ordering br br These are cho
colate oatmeal cookies If you do not like that combination 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 cookie sort of a coconut type consis
tency Now let 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 co
okies or the blurb would say crispy rather than chewy I happen to like raw c
ookie dough however I do not see where these taste like raw cookie dough Bot
h are soft however so is this the confusion And yes they stick together Soft
cookies tend to do that They are not individually wrapped which would add to
the cost Oh yeah chocolate chip cookies tend to be somewhat sweet br br So i
f you want something hard and crisp I suggest Nabiso is Ginger Snaps If you
want a cookie that is soft chewy and tastes like a combination of chocolate
and oatmeal give these a try I am here to place my second order
```

In [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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            '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%|████████████████████████████████████████████████████████████████████████
█████| 4986/4986 [00:02<00:00, 1711.32it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'wow far two two star reviews one obviously no idea ordering wants crispy co
okies hey sorry reviews nobody good beyond reminding us look ordering chocol
ate oatmeal cookies not like combination not order type cookie find combo qu
ite nice really oatmeal sort calms rich chocolate flavor gives cookie sort c
oconut type consistency let also remember tastes differ given opinion soft c
hewy cookies advertised not crispy cookies blurb would say crispy rather che
wy happen like raw cookie dough however not see taste like raw cookie dough
soft however confusion yes stick together soft cookies tend not individually
wrapped would add cost oh yeah chocolate chip cookies tend somewhat sweet wa
nt something hard crisp suggest nabiso ginger snaps want cookie soft chewy t
astes like combination chocolate oatmeal give try place second order'
```

Truncated-SVD

[5.1] Taking top features from TFIDF, SET 2

In [24]:

```
tf_idf_vect = TfidfVectorizer(max_features=2500)
tfidf_vec = tf_idf_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer :",type(tfidf_vec))
print("the shape of out text TFIDF vectorizer : ",tfidf_vec.get_shape())
print("the number of unique words :", tfidf_vec.get_shape()[1])

        # Top 'n' words
top_words = tf_idf_vect.get_feature_names()
        # tfidf frequencies of top 'n' words
freq = tf_idf_vect.idf_
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer :  (4986, 2500)
the number of unique words : 2500
```

[5.2] Calulation of Co-occurrence matrix

In [25]:

```
#def cooccurrenceMatrix(5,top_words):
corpus = dict()
        # List of all words in the corpus
doc = []
index = 0
for sent in preprocessed_reviews:
    for word in sent.split():
        doc.append(word)
        corpus.setdefault(word,[])
        corpus[word].append(index)
        index += 1

        # Co-occurrence matrix
matrix = []
        # rows in co-occurrence matrix
for row in top_words:
            # row in co-occurrence matrix
    temp = []
            # column in co-occurrence matrix
    for col in top_words :
        if( col != row):
                    # No. of times col word is in neighbourhood of row word
            count = 0
                    # Value of neighbourhood
            num = 5
                    # Indices of row word in the corpus
            positions = corpus[row]
            for i in positions:
                if i<(num-1):
                            # Checking for col word in neighbourhood of row
                    if col in doc[i:i+num]:
                        count +=1
                elif (i>=(num-1)) and (i<=(len(doc)-num)):
                            # Check col word in neighbour of row
                    if (col in doc[i-(num-1):i+1]) and (col in doc[i:i+num]):
                        count +=2
                            # Check col word in neighbour of row
                    elif (col in doc[i-(num-1):i+1]) or (col in doc[i:i+num]):
                        count +=1
                else :
                    if (col in doc[i-(num-1):i+1]):
                        count +=1


                    # appending the col count to row of co-occurrence matrix
            temp.append(count)
        else:
                    # Append 0 in the column if row and col words are equal
            temp.append(0)
            # appending the row in co-occurrence matrix
    matrix.append(temp)
```

In [26]:

```
co_occurrence_matrix=np.array(matrix)
```

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

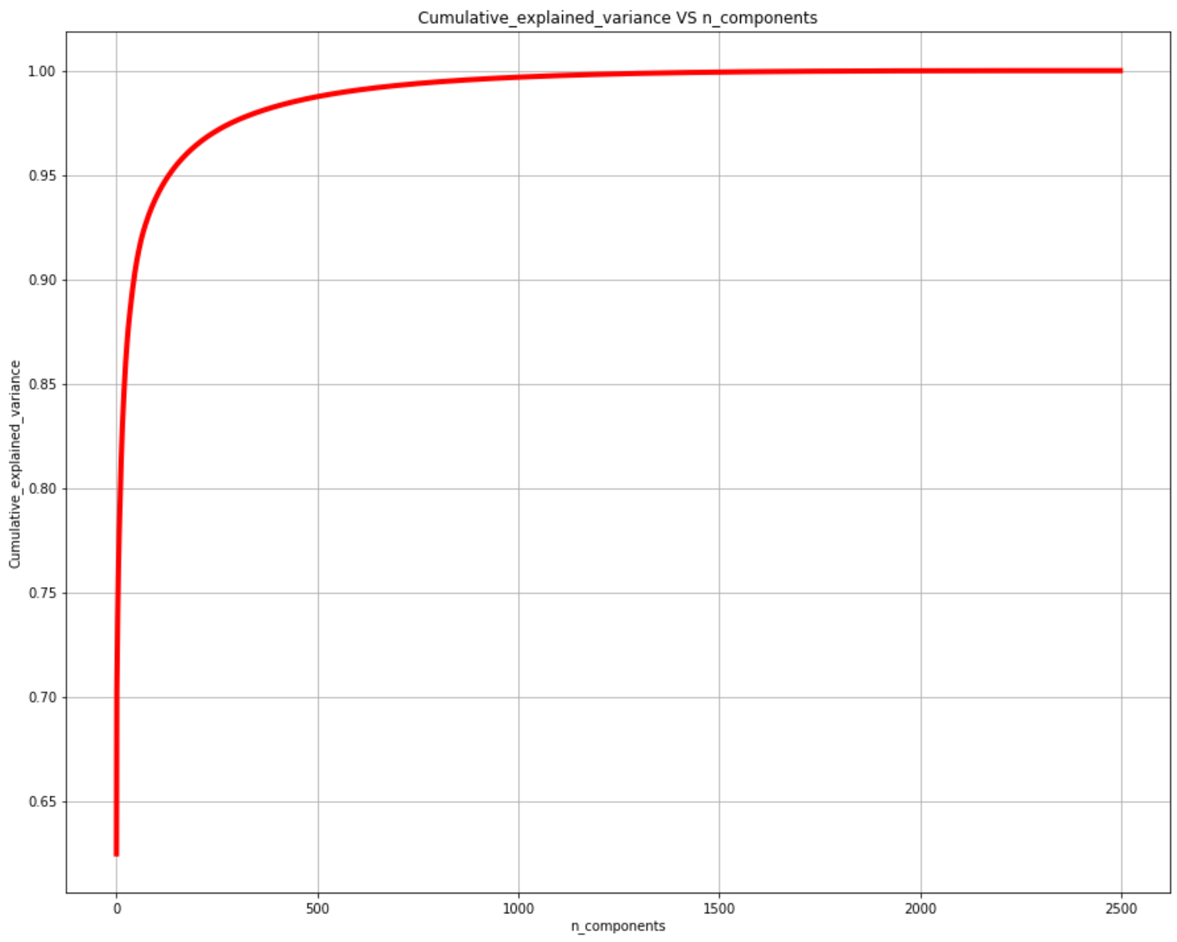
In [28]:

```
from sklearn.decomposition import TruncatedSVD

max_features = co_occurrence_matrix.shape[1]-1
svd = TruncatedSVD(n_components=max_features)
svd_data = svd.fit_transform(co_occurrence_matrix)
percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_variance_)
cum_var_explained = np.cumsum(percentage_var_explained)
```

In [29]:

```
# Plot the TrunvatedSVD spectrum
plt.figure(1, figsize=(15, 12))
plt.clf()
plt.plot(cum_var_explained, linewidth=4,color='red')
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.title("Cumulative_explained_variance VS n_components")
plt.show()
```



OBSERVATION :- From above plot we have observed that only 500 components can explain almost 99% of variance . So, it will be good to use only 500 components instead of total 2500 components .

In [30]:

```
# Computing word vectors with 500 components
svd_trunc = TruncatedSVD(n_components=500)
svd_transform = svd_trunc.fit_transform(co_occurrence_matrix)
word_vec_matrix = svd_transform
print("Shape of word-vector : ",word_vec_matrix.shape)
```

```
Shape of word-vector :  (2500, 500)
```

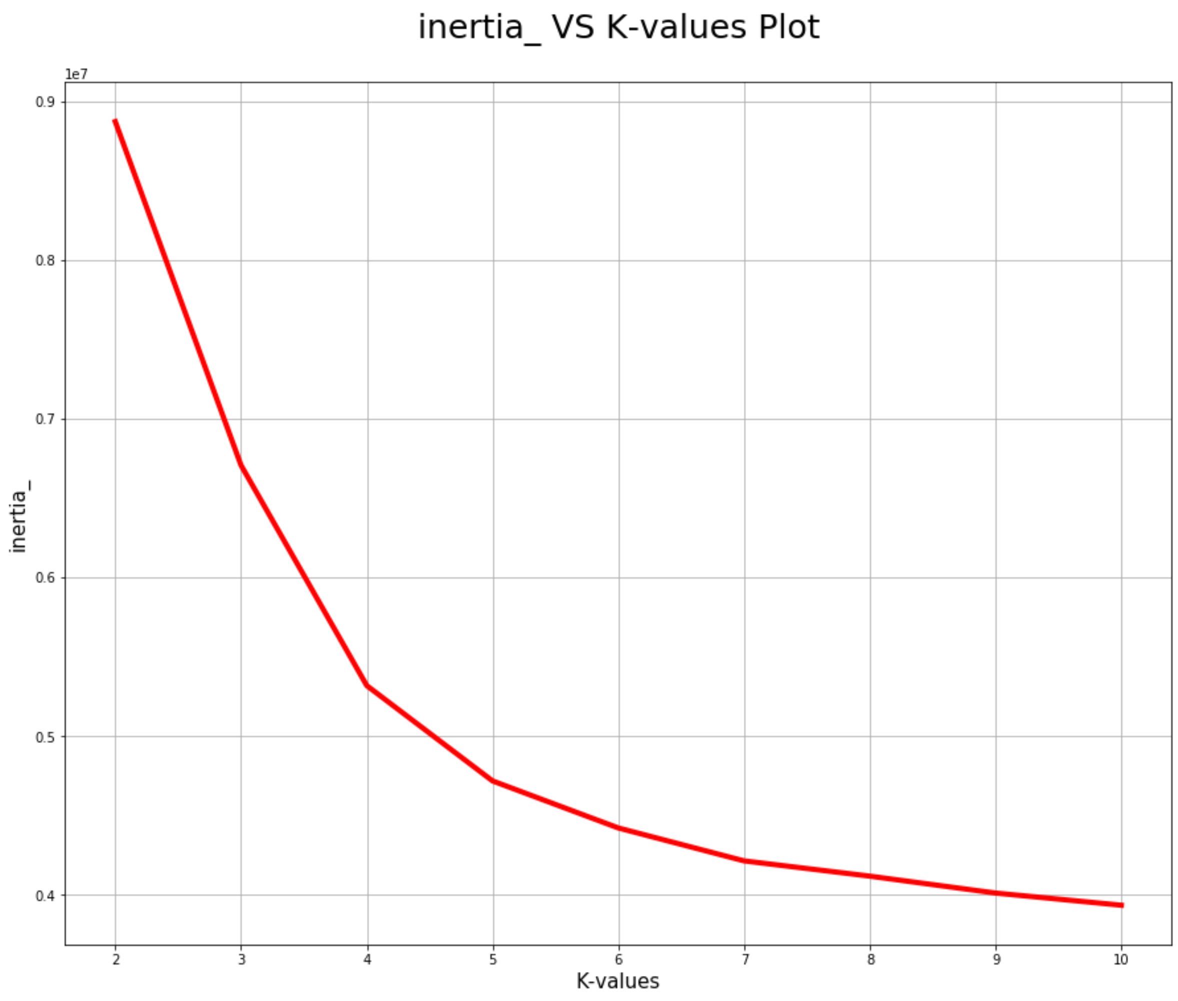
[5.4] Applying k-means clustering

In [33]:

```
from sklearn.cluster import KMeans
hyper = [2,3,4,5,6,7,8,9,10]
loss = []
for i in hyper:
    model = KMeans(n_clusters=i).fit(word_vec_matrix)
    loss.append(model.inertia_)
```

In [34]:

```
# Draw Loss VS K values plot
plt.figure(figsize=(15, 12))
plt.plot(hyper, loss,linewidth=4,color='red')
plt.xlabel('K-values',size=15)
plt.ylabel('inertia_',size=15)
plt.title('inertia_ VS K-values Plot\n',size=25)
plt.grid()
plt.show()
```



In [35]:

```

optimal_k = 4
# Variable that will be used in the conclusion
bow_means_k = optimal_k

# Implementing K-Means using optimal value of K
kmeans = KMeans(n_clusters=optimal_k, n_jobs=-1).fit(word_vec_matrix)
```

[5.5] Wordclouds of clusters obtained in the above section

In [40]:

```
reviews = preprocessed_reviews
# Getting all the reviews in different clusters
cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []



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

In [41]:

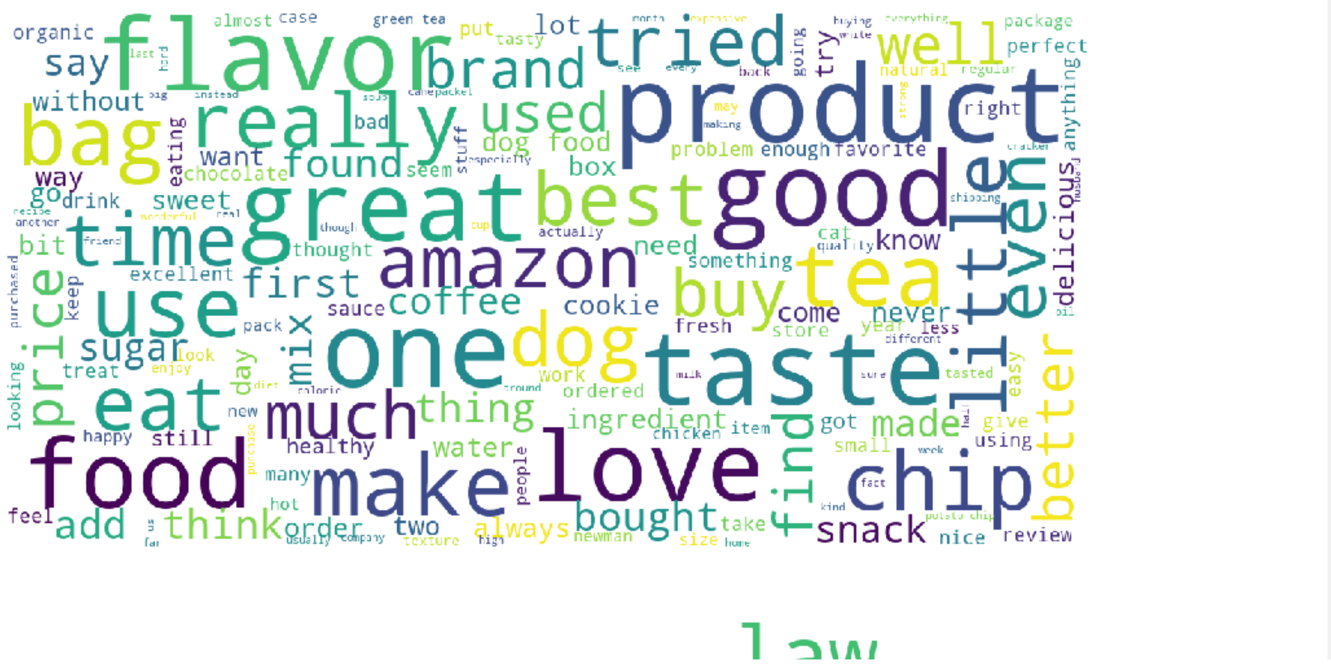
```
# Applying k-means with no_of_clusters = 50 on 'word_vec_matrix' and get all clusters
#code snippet from https://stackoverflow.com/questions/16645799/how-to-create-a-word-cloud-
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
stopwords = set(STOPWORDS)
def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        #max_words=20,
        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=(15, 20))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

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

In [42]:

```
show_wordcloud(cluster1)
show_wordcloud(cluster2)
show_wordcloud(cluster3)
show_wordcloud(cluster4)
```



OBSERVATION: cluster1,2,3,4

1: As we can see in above four clusters many words are related to each other in same clusters shows that reviews which are related are in same cluster.

2:There are some words in cluster which also are in other cluster, some of them may be outliers.

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

In [43]:

```
co_occurrence_matrix.shape
```

Out[43]:

```
(2500, 2500)
```

In [44]:

```
word_vec_matrix.shape
```

Out[44]:

```
(2500, 500)
```

In [47]:

```
word_vec_matrixT=word_vec_matrix.T
```

In [48]:

```
word_vec_matrixT.shape
```

Out[48]:

```
(500, 2500)
```

In [79]:

```
# Scikit Learn
from sklearn.feature_extraction.text import  TfidfVectorizer
df = pd.DataFrame(word_vec_matrixT)

# Compute Cosine Similarity
from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(df, df))
```

```
[[ 1.00000000e+00 -9.71445147e-17  2.01227923e-16 ... -6.50521303e-19
  -4.33680869e-18 -2.42861287e-17]
 [-9.71445147e-17  1.00000000e+00 -1.78676518e-16 ... -1.01915004e-17
  -3.77302356e-17  2.60208521e-18]
 [ 2.01227923e-16 -1.78676518e-16  1.00000000e+00 ...  2.21177243e-17
   4.77048956e-18 -1.59377719e-17]
 ...
 [-6.50521303e-19 -1.01915004e-17  2.21177243e-17 ...  1.00000000e+00
  -3.40439482e-16  1.35308431e-16]
 [-4.33680869e-18 -3.77302356e-17  4.77048956e-18 ... -3.40439482e-16
   1.00000000e+00  4.33680869e-17]
 [-2.42861287e-17  2.60208521e-18 -1.59377719e-17 ...  1.35308431e-16
   4.33680869e-17  1.00000000e+00]]
```

In [51]:

```
co_sim=cosine_similarity(df, df)
co_sim.shape
```

Out[51]:

```
(500, 500)
```

In [52]:

```
tf_idf_vect_500 = TfidfVectorizer(max_features=500)
tfidf_vec_500 = tf_idf_vect_500.fit_transform(preprocessed_reviews)
top_500_words = tf_idf_vect_500.get_feature_names()
```

In [55]:

```
top_500_words_array=np.asarray(top_500_words)
```

In [64]:

```
word='wish'
for i in range(500):
    if(top_500_words_array[i]==word):
        r=i
        print(r)
```

```
488
```

In [66]:

```
co_sim[488].shape
```

Out[66]:

```
(500,)
```

In [76]:

```
indices_for_most_similar_word=np.argsort(co_sim[488])[:50]
```

In [77]:

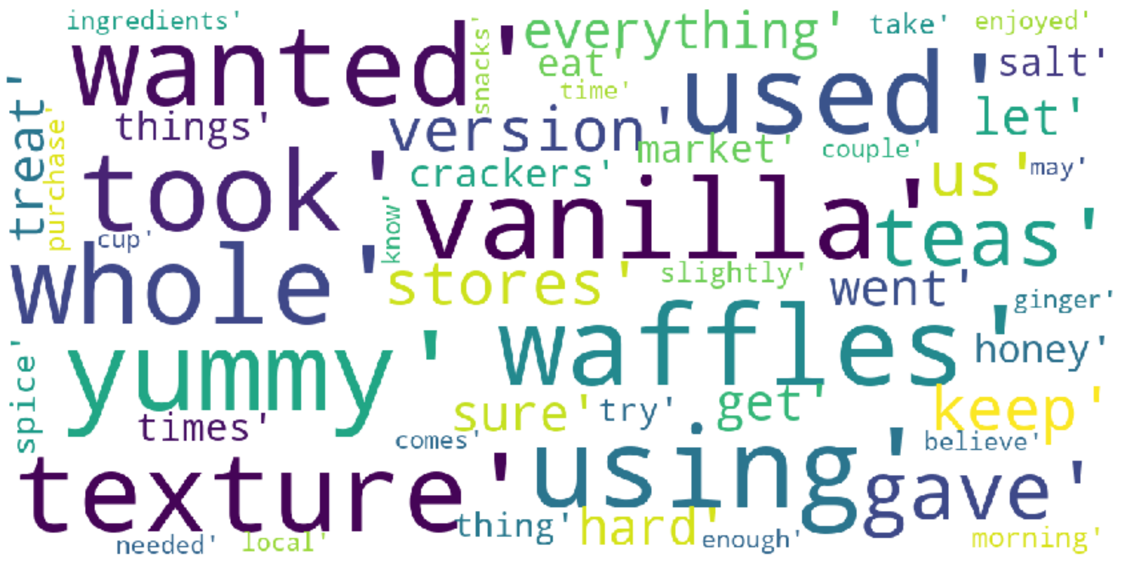
```
wordcloud=[]
for i in (indices_for_most_similar_word):
    word=top_500_words_array[i]
    wordcloud.append(word)
print(wordcloud)
```

```
['used', 'waffles', 'took', 'using', 'vanilla', 'yummy', 'wanted', 'textur
e', 'whole', 'gave', 'teas', 'version', 'us', 'stores', 'keep', 'everythin
g', 'hard', 'sure', 'let', 'treat', 'went', 'get', 'things', 'eat', 'honey',
'times', 'crackers', 'salt', 'market', 'thing', 'try', 'spice', 'slightly',
'morning', 'take', 'purchase', 'ingredients', 'ginger', 'comes', 'enough',
'couple', 'enjoyed', 'local', 'snacks', 'time', 'cup', 'believe', 'may', 'kn
ow', 'needed']
```

Wordcloud for words return via Function that returns most similar words

In [78]:

```
show_wordcloud(wordcloud)
```



[6] Conclusions

NOTE--->> We have taken 5000 POINTS and top 2500 features for this assignment.

Procedure Followed :---->>

STEP 1 :- Text Preprocessing.

STEP 2 :- Taking all text data and ignoring class variable.

STEP 3:- Taking top 2500 features from TFIDF vectorizer.

STEP 4:- Calculate Co-occurrence matrix on top 2500 features or words.

STEP 5:- Finding right number of components using cumulative_explained_variance VS n_components plot.

STEP 6:- Again compute TFIDF word vectors with 500 components.

STEP 7 :- Applying TruncatedSVD on this co-occurrence matrix with right number of components i.e 500 in order to find matrix of word-vectors .

STEP 8:- Applying the Elbow Method in order to find the right value of k for k-means clustering algorithm.

STEP 9:- Draw K VS inertia_ for k-means clustering algorithm for finding optimal value of k.

STEP 10:- Implementing k-means with optimal k.

STEP 10:- Draw Wordclouds for each clusters of k-means with optimal k .

STEP 11:- Write a function that returns most similar words for a given word.

STEP 12:- Draw Wordcloud for words return via Function that returns most similar words.

In [ ]:

```

```