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
```

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

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

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

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

from tqdm import tqdm
import os
```

In [2]:

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

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

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, 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 (525814, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia	1	1	1	1219017600	"Delight" says it all

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	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 ...	5

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

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

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

Out[7]:

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

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

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

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

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

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

In [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=False)
final.shape
```

Out[9]:

```
(364173, 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]:

```
69.25890143662969
```

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

In [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

In [12]:

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

In [13]:

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

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

```
(364171, 10)
```

Out[13]:

```
1    307061
0     57110
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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer
bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back
in 2005 for gosh sakes.  I admit that Amazon agreed to credit me for cost plus part of shipping, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured
out a way to fix that. I still like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this
product.<br /><br />Thick, delicious.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple
Flavor.  PERIOD.  No chemicals.  No garbage.<br /><br />Have numerous friends & family members
hooked on this stuff.  My husband & son, who do NOT like "sugar free" prefer this over major label
regular syrup.<br /><br />I use this as my SWEETENER in baking: cheesecakes, white brownies,
muffins, pumpkin pies, etc... Unbelievably delicious...<br /><br />Can you tell I like it? :)
==================================================
```

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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [16]:

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

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

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

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

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer
bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back
in 2005 for gosh sakes.  I admit that Amazon agreed to credit me for cost plus part of shipping, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured
out a way to fix that. I still like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this
product.Thick, delicious.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple Flavor.
PERIOD.  No chemicals.  No garbage.Have numerous friends & family members hooked on this stuff.  M
y husband & son, who do NOT like "sugar free" prefer this over major label regular syrup.I use thi
s as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc...
Unbelievably delicious...Can you tell I like it? :)
```

In [17]:

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

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

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

In [18]:

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

```
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
```

```
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today is Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70 is it was poisonous until they
figured out a way to fix that. I still like it but it could be better.
==================================================
```

In [19]:

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [20]:

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

```
Great ingredients although chicken should have been 1st rather than chicken broth the only thing I
do not think belongs in it is Canola oil Canola or rapeseed is not someting a dog would ever find
in nature and if it did find rapeseed in nature and eat it it would poison them Today is Food indu
stries have convinced the masses that Canola oil is a safe and even better oil than olive or virgi
n coconut facts though say otherwise Until the late 70 is it was poisonous until they figured out
a way to fix that I still like it but it could be better
```

In [21]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

```python
# 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%|██████████| 364171/364171 [02:35<00:00, 2346.35it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'great ingredients although chicken rather chicken broth thing not think belongs canola oil canola rapeseed not someting dog would ever find nature find rapeseed nature eat would poison today food industries convinced masses canola oil safe even better oil olive virgin coconut facts though say otherwise late poisonous figured way fix still like could better'
```

[3.2] Preprocessing Review Summary

In [24]:

```
## Summary preprocessing
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].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_summary.append(sentance.strip())
```

```
100%|██████████| 364171/364171 [01:42<00:00, 3560.76it/s]
```

In [23]:

```
final['Cleaned_text'] = preprocessed_reviews
```

In [24]:

```
### Sort data according to time series
final.sort_values('Time',inplace=True)
```

In [25]:

```
### Taking 100k samples
final_100k = final.sample(n=100000)
```

In [26]:

```
x = final_100k['Cleaned_text']
x.size
```

Out[26]:

```
100000
```

[4] Featurization

[4.1] BAG OF WORDS

In [29]:

```
#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', 'aaaaaa', 'aaaaaaaaaaa', 'aaaaaaaaaaaa',
'aaaaaaaaaaaaa', 'aaaaaaaaaaaaaa', 'aaaaaaaaaaaaaaa']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (364171, 116756)
the number of unique words  116756
```

[4.2] Bi-Grams and n-Grams.

In [30]:

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

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

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (364171, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [27]:

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

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

```
some sample features(unique words in the corpus) ['aa', 'aback', 'abandon', 'abandoned', 'abc', 'a
bdominal', 'abilities', 'ability', 'ability make', 'able']
==================================================
the type of count vectorizer  <class 'sklearn.feature_extraction.text.TfidfVectorizer'>
```

[4.4] Word2Vec

In [ ]:

```
# 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 [ ]:

```
# Using Google News Word2Vectors

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

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

```
[('terrific', 0.9073498845100403), ('fantastic', 0.8992555737495422), ('awesome',
0.8687705993652344), ('excellent', 0.8576902151107788), ('good', 0.8564110398292542),
('wonderful', 0.8142356872558594), ('perfect', 0.7783650159835815), ('amazing',
0.7489546537399292), ('nice', 0.7472279071807861), ('fabulous', 0.7337148785591125)]
==================================================
[('nastiest', 0.9013340473175049), ('greatest', 0.7858481407165527), ('disgusting',
0.7614876627922058), ('saltiest', 0.7270584106445312), ('horrible', 0.7270214557647705), ('best',
0.718712568283081), ('terrible', 0.7127232551574707), ('tastiest', 0.7062922716140747), ('nicest',
0.6990509033203125), ('vile', 0.6960209608078003)]
```

In [ ]:

```
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  33573
sample words  ['witty', 'little', 'book', 'makes', 'son', 'laugh', 'loud', 'recite', 'car',
'driving', 'along', 'always', 'sing', 'refrain', 'learned', 'whales', 'india', 'drooping',
'roses', 'love', 'new', 'words', 'introduces', 'silliness', 'classic', 'willing', 'bet', 'still',
'able', 'memory', 'college', 'grew', 'reading', 'sendak', 'books', 'watching', 'really', 'rosie',
```

```
'movie', 'incorporates', 'loves', 'however', 'miss', 'hard', 'cover', 'version', 'seem', 'kind', '
flimsy', 'takes']
```

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

[4.4.1.1] Avg W2v

In [ ]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 364171/364171 [28:41<00:00, 211.60it/s]
```

```
364171
50
```

[4.4.1.2] TFIDF weighted W2v

In [ ]:

```
# 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 [ ]:

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
  3%|▌          | 12387/364171 [35:07<10:27:59,  9.34it/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 [28]:

```
### Take top 2000 features from tf-idf vectorizers
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect = TfidfVectorizer(ngram_range = (1,1) , max_features = 2000)
tfidf_train = tfidf_vect.fit_transform (final_100k['Cleaned_text'])
```

In [31]:

```
top_2000 = tfidf_vect.get_feature_names()
```

[5.2] Calulation of Co-occurrence matrix

In [158]:

```
## co-occurence matrix
from tqdm import tqdm
n_neighbor = 5
occ_matrix_2000 = np.zeros((2000,2000))
for row in tqdm(final_100k['Cleaned_text'].values):
    words_in_row = row.split()
    for index,word in enumerate(words_in_row):
        if word in top_2000:
            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_2000:
                    occ_matrix_2000[top_2000.index(word),top_2000.index(words_in_row[j])] += 1
                else:
                    pass
        else:
            pass
```

```
100%|██████████| 100000/100000 [30:20<00:00, 54.92it/s]
```

In [ ]:

```
### check co-occurrence matrix code for the given example.
```

In [160]:

```
str = "abc def ijk pqr", "pqr klm opq", "lmn pqr xyz abc def pqr abc"
```

In [161]:

```
to_fea1 =  "abc", "pqr", "def"
```

In [162]:

```
from tqdm import tqdm
n_neighbor = 2
occ_matrix = np.zeros((3,3))
for row in tqdm(str):
    words_in_row1 = row.split()
    for index,word in enumerate(words_in_row1):
        if word in to_fea1:
            for j in range(max(index-n_neighbor,0),min(index+n_neighbor,len(words_in_row1)-1) + 1):
                if words_in_row1[j] in to_fea1:
                    occ_matrix[to_fea1.index(word),to_fea1.index(words_in_row1[j])] += 1
                else:
                    pass
        else:
            pass
```

```
100%|██████████| 3/3 [00:00<00:00, 4670.72it/s]
```

In [163]:

```
occ_matrix
```

Out[163]:

```
array([[3., 3., 3.],
       [3., 4., 2.],
       [3., 2., 2.]])
```

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

In [35]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import csr_matrix
import numpy as np
###
https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_of_components_in_ts
```

In [37]:

```
# Standardize the feature matrix
X = StandardScaler().fit_transform(occ_matrix_2000)

# Make sparse matrix
X_sparse = csr_matrix(occ_matrix_2000)
```

In [38]:

```
# Create and run an TSVD with one less than number of features
tsvd = TruncatedSVD(n_components=X_sparse.shape[1]-1)
X_tsvd = tsvd.fit(occ_matrix_2000)
```

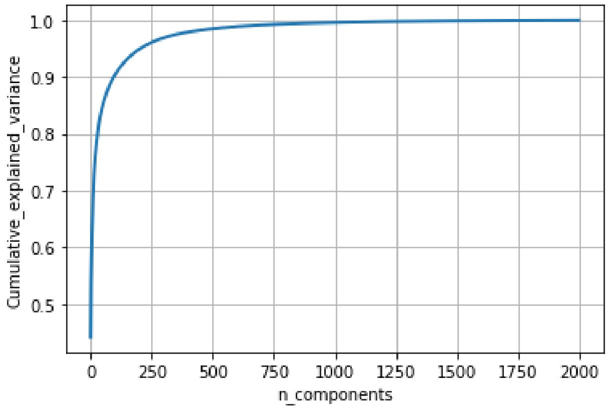
In [43]:

```
percentage_var_explained = X_tsvd.explained_variance_ / np.sum(X_tsvd.explained_variance_);
cum_var_explained = np.cumsum(percentage_var_explained)
```

In [44]:

```
plt.figure(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()
```



Observation: By observing plot, getting high cumulative explained variance at 150 n_components.

In [64]:

```
## Train model on optimal n_components
svd = TruncatedSVD(n_components = 150)
svd_2000 = svd.fit_transform(occ_matrix_2000)
```

[5.4] Applying k-means clustering

In [ ]:

```
#### elbow method
```

In [65]:

```
from sklearn.cluster import KMeans
num_clus = [x for x in range(3,11)]
num_clus
```

Out[65]:

```
[3, 4, 5, 6, 7, 8, 9, 10]
```

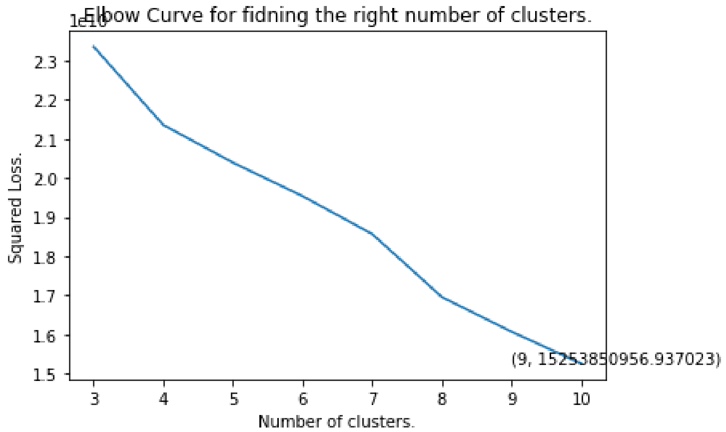
In [66]:

```
### Hyperparameter tunnning for k clusters
squared_errors = []
for cluster in num_clus:
    kmeans = KMeans(n_clusters = cluster, n_jobs = -1)
    kmeans = kmeans.fit(svd_2000)
    squared_errors.append(kmeans.inertia_)

optimal_clusters = np.argmin(squared_errors) + 2
plt.plot(num_clus, squared_errors)
plt.title("Elbow Curve for fidning the right number of clusters.")
plt.xlabel("Number of clusters.")
plt.ylabel("Squared Loss.")
xy = (optimal_clusters, min(squared_errors))
plt.annotate('(%s, %s)' % xy, xy = xy, textcoords='data')
plt.show()

print ("The optimal number of clusters obtained is - ", optimal_clusters)
```

```
print ("The optimal number of clusters obtained is - ", optimal_clusters)
print ("The loss for optimal cluster is - ", min(squared_errors))
```




```
The optimal number of clusters obtained is -  9
The loss for optimal cluster is -  15253850956.937023
```

In [77]:

```
optimal_k = KMeans(n_clusters = 9)
p = optimal_k.fit_predict(svd_2000)
```

In [78]:

```
list_of_sent = []
for i in final_100k['Cleaned_text'].values:
    sent = []
    for word in i.split():
        sent.append(word)
    list_of_sent.append(sent)
```

In [79]:

```
### append a same label cluster in index
index_0 = []
index_1 = []
index_2 = []
index_3 = []
index_4 = []
index_5 = []
index_6 = []
index_7 = []
index_8 = []

for i in range(len(p)):
    if p[i] == 0:
        index_0.append(i)
for i in range(len(p)):
    if p[i] == 1:
        index_1.append(i)
for i in range(len(p)):
    if p[i] == 2:
        index_2.append(i)
for i in range(len(p)):
    if p[i] == 3:
        index_3.append(i)
for i in range(len(p)):
    if p[i] == 4:
        index_4.append(i)
for i in range(len(p)):
    if p[i] == 5:
        index_5.append(i)
for i in range(len(p)):
    if p[i] == 6:
        index_6.append(i)
for i in range(len(p)):
    if p[i] == 7:
        index_7.append(i)
```

```
for i in range(len(p)):
    if p[i] == 8:
        index_8.append(i)
```

In [80]:

```
text_0 = []
text_1 = []
text_2 = []
text_3 = []
text_4 = []
text_5 = []
text_6 = []
text_7 = []
text_8 = []
for i in range(len(index_0)):
    text_0.append(list_of_sent[index_0[i]])
for i in range(len(index_1)):
    text_1.append(list_of_sent[index_1[i]])
for i in range(len(index_2)):
    text_2.append(list_of_sent[index_2[i]])
for i in range(len(index_3)):
    text_3.append(list_of_sent[index_3[i]])
for i in range(len(index_4)):
    text_4.append(list_of_sent[index_4[i]])
for i in range(len(index_5)):
    text_5.append(list_of_sent[index_5[i]])
for i in range(len(index_6)):
    text_6.append(list_of_sent[index_6[i]])
for i in range(len(index_7)):
    text_7.append(list_of_sent[index_7[i]])
for i in range(len(index_8)):
    text_8.append(list_of_sent[index_8[i]])
```

In [62]:

```
# Using the groups formed we will store all words which have been grouped into one cluster for plo
tting lateron.
sent = list()
for i in range(0,9):
    string = " "
    for x in wcdgroup.groups[i]:
        # Get the words which belong to cluster i and store in sentence i.
        string += wcd.loc[x, 'words']
        string += " "
    sent.append(string)

len(sent)
```

Out[62]:

9

[5.5] Wordclouds of clusters obtained in the above section

In [81]:

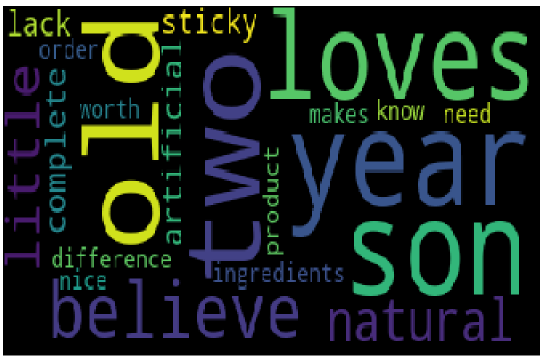
```
## Cleate a wordcloud of cluster 0
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_0)):
    for i in range(len(text_0[j])):
        t_b = t_b + text_0[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```

In [82]:

```python
## Cleate a wordcloud of cluster 1
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_1)):
    for i in range(len(text_1[j])):
        t_b = t_b + text_1[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```

In [83]:

```python
## Cleate a wordcloud of cluster 2
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_2)):
    for i in range(len(text_2[j])):
        t_b = t_b + text_2[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [84]:

```
## Cleate a wordcloud of cluster 3
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_3)):
    for i in range(len(text_3[j])):
        t_b = t_b + text_3[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```

In [85]:

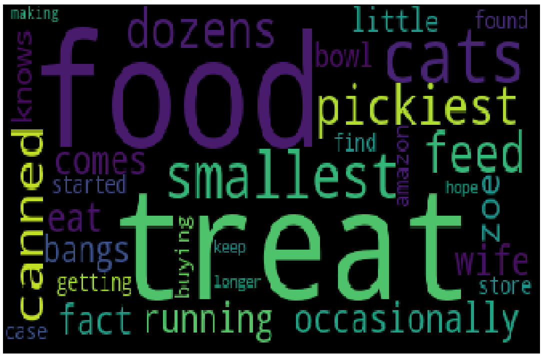
```
## Cleate a wordcloud of cluster 4
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_4)):
    for i in range(len(text_4[j])):
        t_b = t_b + text_4[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```

In [86]:

```
## Cleate a wordcloud of cluster 5
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_5)):
    for i in range(len(text_5[j])):
        t_b = t_b + text_5[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```

In [87]:

```
## Cleate a wordcloud of cluster 6
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_6)):
    for i in range(len(text_6[j])):
        t_b = t_b + text_6[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```

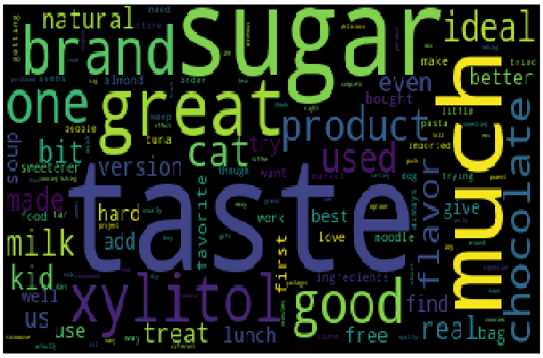


In [88]:

```
## Cleate a wordcloud of cluster 7
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_7)):
    for i in range(len(text_7[j])):
        t_b = t_b + text_7[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```

In [89]:

```
## Cleate a wordcloud of cluster 8
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_8)):
    for i in range(len(text_8[j])):
        t_b = t_b + text_8[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



Observation: By observing wordclouds ,some clusters contain only 1 word and some are grouped well.

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

In [115]:

```
from sklearn.metrics.pairwise import cosine_similarity
def cos_similarity(word):
    similarity = cosine_similarity(occ_matrix_2000)
    word_vect = similarity[top_2000.index(word)]
    print("Similar Word to",word)
    index = word_vect.argsort()[::-1][:5]
    for j in range(len(index)):
        print(top_2000[index[j]] ,":",word,"\n")
```

In [116]:

```
cos_similarity(top_2000[1])
```

```
Similar Word to absolute
absolute : absolute

favorite : absolute

best : absolute

ever : absolute

not : absolute
```

In [118]:

```
cos_similarity(top_2000[100])
```

```
Similar Word to bacon
bacon : bacon
```

```
like : bacon

flavor : bacon

not : bacon

fake : bacon
```

[6] Conclusions

In [ ]:

```
# 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.
```

1) By observing wordclouds ,clusters are grouped well. 2) The optimal number of clusters obtained is - 9 3) The optimal values that you obtained for number of components - 150