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
```

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

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

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

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

from tqdm import tqdm
import os
```

In [2]:

```python
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&response_type=code&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

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

In [3]:
```
# using SQLite Table to read data.
con = sqlite3.connect('/content/drive/My Drive/Colab Notebooks/databas
e.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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 LIMIT 50000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a 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 (50000, 10)
```

Out[3]:

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomin

	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 [5]:

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

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)

	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 [6]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[6]:

	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 [7]:
```
display['COUNT(*)'].sum()
```

Out[7]: 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 [8]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
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 [10]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time"
,"Text"}, keep='first', inplace=False)
final.shape
```

Out[10]:
```
(46072, 10)
```

In [11]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[11]:
```
92.144
```

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

In [12]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

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

In [14]:
```
#Before starting the next phase of preprocessing lets see the number of
 entries left
print(final.shape)

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

```
(46071, 10)
```

Out[14]:
```
1    38479
0     7592
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.

3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

In [15]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp
or cobbler. home made is better, but a heck of a lot more work. this is
great to have on hand for last minute dessert needs where you really wa
nt to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usua
```

lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==
For those of you wanting a high-quality, yet affordable green tea, you
should definitely give this one a try. Let me first start by saying tha
t everyone is looking for something different for their ideal tea, and
I will attempt to briefly highlight what makes this tea attractive to a
wide range of tea drinkers (whether you are a beginner or long-time tea
enthusiast). I have gone through over 12 boxes of this tea myself, and
highly recommend it for the following reasons:

-Quality: Fi
rst, this tea offers a smooth quality without any harsh or bitter after
tones, which often turns people off from many green teas. I've found m
y ideal brewing time to be between 3-5 minutes, giving you a light but
flavorful cup of tea. However, if you get distracted or forget about y
our tea and leave it brewing for 20+ minutes like I sometimes do, the q
uality of this tea is such that you still get a smooth but deeper flavo
r without the bad after taste. The leaves themselves are whole leaves
(not powdered stems, branches, etc commonly found in other brands), and
the high-quality nylon bags also include chunks of tropical fruit and o
ther discernible ingredients. This isn't your standard cheap paper bag
with a mix of unknown ingredients that have been ground down to a fine
powder, leaving you to wonder what it is you are actually drinking.

-Taste: This tea offers notes of real pineapple and other hint
s of tropical fruits, yet isn't sweet or artificially flavored. You ha
ve the foundation of a high-quality young hyson green tea for those tru
e "tea flavor" lovers, yet the subtle hints of fruit make this a truly
unique tea that I believe most will enjoy. If you want it sweet, you c
an add sugar, splenda, etc but this really is not necessary as this tea
offers an inherent warmth of flavor through it's ingredients.

-Price: This tea offers an excellent product at an exceptional price
(especially when purchased at the prices Amazon offers). Compared to o
ther brands which I believe to be of similar quality (Mighty Leaf, Rish
i, Two Leaves, etc.), Revolution offers a superior product at an outsta
nding price. I have been purchasing this through Amazon for less per b
ox than I would be paying at my local grocery store for Lipton, etc.

Overall, this is a wonderful tea that is comparable, and even b
etter than, other teas that are priced much higher. It offers a well-b
alanced cup of green tea that I believe many will enjoy. In terms of t

```
aste, quality, and price, I would argue you won't find a better combina
tion that that offered by Revolution's Tropical Green Tea.
==================================================
```

In [16]:

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

print(sent_0)
```

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

In [17]:

```
# 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')
```

```python
text = soup.get_text()
print(text)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp
or cobbler. home made is better, but a heck of a lot more work. this is
great to have on hand for last minute dessert needs where you really wa
nt to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==================================================
For those of you wanting a high-quality, yet affordable green tea, you
should definitely give this one a try. Let me first start by saying tha
t everyone is looking for something different for their ideal tea, and
I will attempt to briefly highlight what makes this tea attractive to a
wide range of tea drinkers (whether you are a beginner or long-time tea
enthusiast).  I have gone through over 12 boxes of this tea myself, and
highly recommend it for the following reasons:-Quality:  First, this te
a offers a smooth quality without any harsh or bitter after tones, whic
h often turns people off from many green teas.  I've found my ideal bre
wing time to be between 3-5 minutes, giving you a light but flavorful c
up of tea.  However, if you get distracted or forget about your tea and
leave it brewing for 20+ minutes like I sometimes do, the quality of th
is tea is such that you still get a smooth but deeper flavor without th
e bad after taste.  The leaves themselves are whole leaves (not powdere
d stems, branches, etc commonly found in other brands), and the high-qu
ality nylon bags also include chunks of tropical fruit and other discer
nible ingredients.  This isn't your standard cheap paper bag with a mix
of unknown ingredients that have been ground down to a fine powder, lea
ving you to wonder what it is you are actually drinking.-Taste:  This t
ea offers notes of real pineapple and other hints of tropical fruits, y
et isn't sweet or artificially flavored.  You have the foundation of a
```

```
high-quality young hyson green tea for those true "tea flavor" lovers,
yet the subtle hints of fruit make this a truly unique tea that I belie
ve most will enjoy.  If you want it sweet, you can add sugar, splenda,
etc but this really is not necessary as this tea offers an inherent war
mth of flavor through it's ingredients.-Price:  This tea offers an exce
llent product at an exceptional price (especially when purchased at the
prices Amazon offers).  Compared to other brands which I believe to be
of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), Revolution o
ffers a superior product at an outstanding price.  I have been purchasi
ng this through Amazon for less per box than I would be paying at my lo
cal grocery store for Lipton, etc.Overall, this is a wonderful tea that
is comparable, and even better than, other teas that are priced much hi
gher.  It offers a well-balanced cup of green tea that I believe many w
ill enjoy.  In terms of taste, quality, and price, I would argue you wo
n't find a better combination that that offered by Revolution's Tropica
l Green Tea.
```

In [0]:

```python
# 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 [19]:

```python
sent_1500 = decontracted(sent_1500)
print(sent_1500)
```

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

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

In [20]:
```
#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)
```

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

In [21]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Great flavor low in calories high in nutrients high in protein Usually
protein powders are high priced and high in calories this one is a grea
t bargain and tastes great I highly recommend for the lady gym rats pro
bably not macho enough for guys since it is soy based
```

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 [23]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
```

```
() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 46071/46071 [00:21<00:00, 2112.51it/s]
```

In [24]:
```
preprocessed_reviews[1500]
```

Out[24]:
```
'great flavor low calories high nutrients high protein usually protein powders high priced high calories one great bargain tastes great highly recommend lady gym rats probably not macho enough guys since soy based'
```

In [0]:
```
#No need to split the data in unsupervised learning
X=preprocessed_reviews
Y=final['Score']
```

[4] Featurization

[4.3] TF-IDF

In [26]:
```
#below code for converting to tfidf
tf_idf_vect = TfidfVectorizer(max_features=3000)
tf_idf_vect.fit(X)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

X_tf_idf = tf_idf_vect.transform(X)
print("the type of count vectorizer ",type(X_tf_idf))
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'absolute', 'absolutely', 'acceptable', 'according', 'acid', 'acidic', 'acidity', 'acids']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

In [27]:
```
print(len(X))
print(X_tf_idf.shape)
```

```
46071
(46071, 3000)
```

[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

https://github.com/zl376/label_it/blob/447c674a3c0bae9fd9286b70f41cea7360735623/label_it/mod

In [0]:
```
#Reference : https://stackoverflow.com/questions/25217510/how-to-see-top-n-entries-of-term-document-matrix-after-tfidf-in-scikit-learn

from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

indices = np.argsort(tf_idf_vect.idf_)[::-1]#sorting via scores
features = tf_idf_vect.get_feature_names()#getting all features
top_n = 3000#considering top 3k features
top_features = [(features[i],tf_idf_vect.idf_[i]) for i in indices[:top_n]]#taking top features

#creating dataframe and putting feature names with there idf scores
df=pd.DataFrame(top_features)
df.columns=['features','Score']
```

[5.2] Calulation of Co-occurrence matrix

In [0]:
```
#creating a matrix with all zero initializationa  and shape (3000,3000)
matrix=np.zeros((3000,3000))

#creating a data frame for co-occurance matrix with rows and columns as words
df_co_occurance_matrix=pd.DataFrame(matrix,index=features,columns=features)
```

In [30]:
```
#code for co-occurance matrix
for sentence in tqdm(X):
  words = sentence.split(" ")
  for i in range(len(words)):
    # Considering 5 neighbours
    #Word_vectors using Truncated SVD
```

```
    for j in range(1,6):
      if(i + j < len(words) and words[i] != words[j]):
        try:
          # If word i occurs in the proximity of word j add +1 to matri
x initialized with null values .
          df_co_occurance_matrix.loc[words[i], words[j]] += 1

          df_co_occurance_matrix.loc[words[j], words[i]] += 1
        except:
          pass
```

```
100%|██████████| 46071/46071 [1:14:12<00:00, 10.35it/s]
```

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

In [31]:
```
print(df_co_occurance_matrix.shape)
```

```
(3000, 3000)
```

In [0]:
```
from sklearn.decomposition import TruncatedSVD
tsvd = TruncatedSVD(n_components=df_co_occurance_matrix.shape[1]-1)
df_tsvd = tsvd.fit(df_co_occurance_matrix)
```

In [0]:
```
tsvd_var_ratios = tsvd.explained_variance_ratio_
```

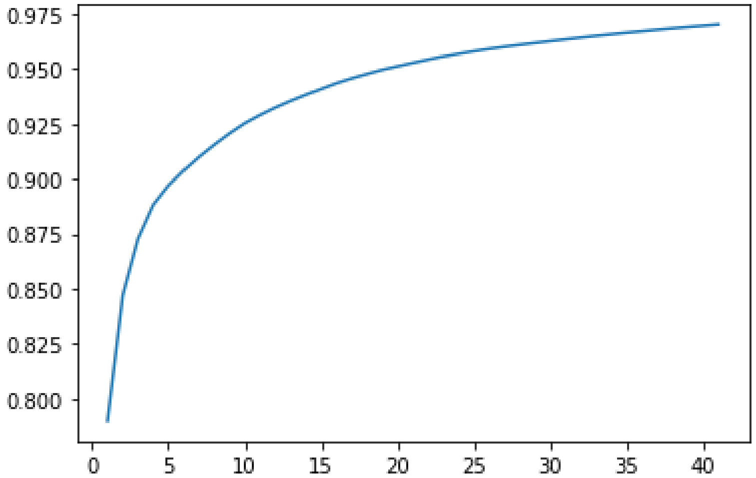
In [34]:
```
print(tsvd_var_ratios.shape)
```

```
(2999,)
```

In [0]:
```
#reference : https://chrisalbon.com/machine_learning/feature_engineerin
g/select_best_number_of_components_in_tsvd/
# Create a function
def select_n_components(var_ratio, goal_var: float) -> int:
  n_comp=[]
  tot_var=[]
```

```python
    # Set initial variance explained so far
    total_variance = 0.0

    # Set initial number of features
    n_components = 0

    # For the explained variance of each feature:
    for explained_variance in var_ratio:
      # Add the explained variance to the total
      total_variance += explained_variance

      # Add one to the number of components
      n_components += 1
      n_comp.append(n_components)
      tot_var.append(total_variance)

      # If we reach our goal level of explained variance
      if total_variance >= goal_var:
        plt.plot(n_comp,tot_var)
        plt.show()
        # End the loop
        break

    # Return the number of components
    return n_components
```

In [36]:
```python
## Here we have used a threshold of 97
select_n_components(tsvd_var_ratios, 0.97)
```



Out[36]: 41

In [0]:
```
#41 is the optimal number of points
```

In [0]:
```
def SVDTruncated(matrix):
  # Variance will store the value of explained variance for each value
 of k
  variance = []
  maxvar = -1
  svdret = 0
  svd = TruncatedSVD(n_components=41)
  svd.fit(matrix)
  U = svd.transform(matrix)
  VT = svd.components_

  arr = np.array([[0 for x in range(svd.singular_values_.shape[0])] for
 x in range(svd.singular_values_.shape[0])])
  for i in range(svd.singular_values_.shape[0]):
    for i in range(svd.singular_values_.shape[0]):
      arr[i, i] = svd.singular_values_[i]

  Sigma = arr
  return (U, Sigma, VT)
```

In [0]:
```
U, Sigma, VT = SVDTruncated(df_co_occurance_matrix)
```

In [40]:
```
print(U.shape, " ", Sigma.shape, " ", VT.shape)
```

```
(3000, 41)   (41, 41)   (41, 3000)
```

[5.4] Applying k-means clustering

In [41]:
```
# Standardizing the data with mean=0 and std.dev=1 of u
from sklearn.preprocessing import StandardScaler
standardized_data_tf_idf = StandardScaler().fit_transform(U)
print(standardized_data_tf_idf.shape)
```

```
(3000, 41)
```

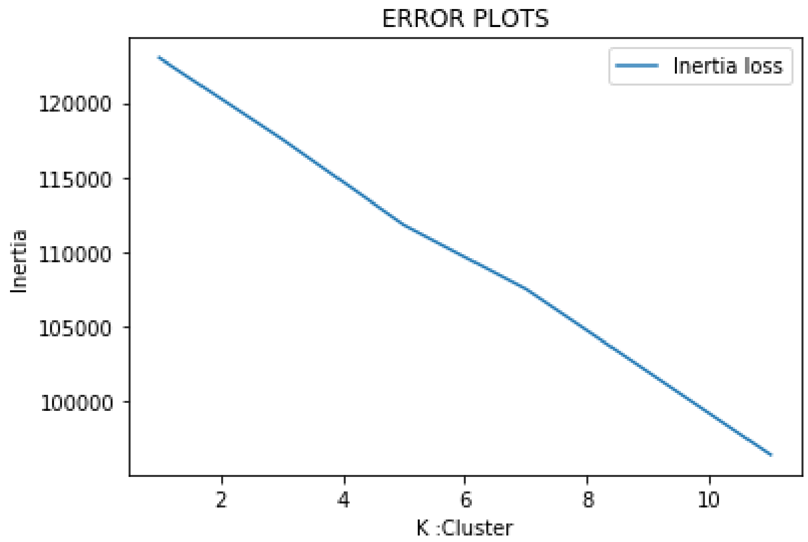
In [0]:
```
data_tf = np.array(standardized_data_tf_idf) # storing the values after
 standardization in a numpy array
```

In [0]:
```
from sklearn.cluster import KMeans
import numpy as np
##taking 3 to 10 cluster values in hyper parameter tuning
def Kmeans_Choosing_Best_K(vectorizer_data):
  inertia_value=[]
  k_values=[1,3,5,7,11]
  for i in tqdm(k_values):
    kmeans = KMeans(n_clusters=i, random_state=0).fit(data_tf)
    inertia_value.append(kmeans.inertia_)
  plt.plot(k_values, inertia_value, label='Inertia loss')
  plt.legend()
  plt.xlabel("K :Cluster")
  plt.ylabel("Inertia")
  plt.title("ERROR PLOTS")
  plt.show()
```

In [44]:
```
#finding optimal number of clusters
```

```
Kmeans_Choosing_Best_K(data_tf)
```

```
100%|██████████| 5/5 [00:00<00:00,  6.12it/s]
```



In [0]:
```
#by above observation ,optimal number of clusters are:
best_k=5
```

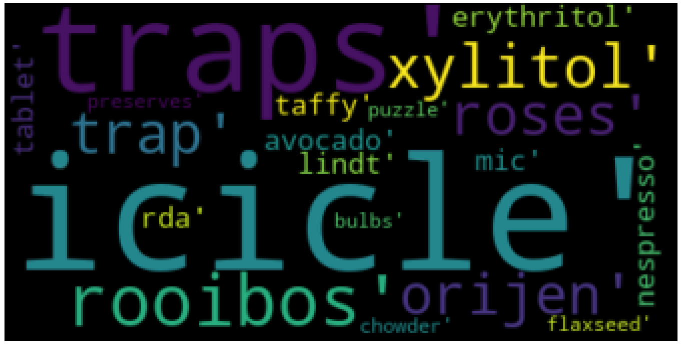
In [0]:
```
#fitting k means with optimal number of clusters
kmeans = KMeans(n_clusters=best_k, verbose=0, init='k-means++').fit(dat
a_tf)
```

[5.5] Wordclouds of clusters obtained in the above section

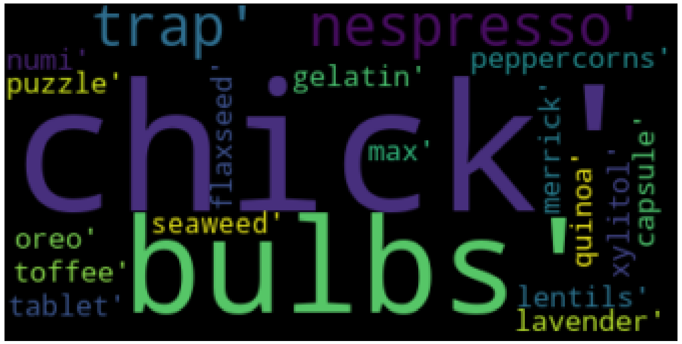
In [47]:
```
### Wordcloud for each cluster
from wordcloud import WordCloud, STOPWORDS
stopwords_t = set(STOPWORDS)
centroids = kmeans.cluster_centers_.argsort() # function for printing t
op 100 feature names with each cluster
terms =top_features
list1 = []
```

```
for i in range(best_k):
  print("Cluster %d:" % i, end='')
  for j in centroids[i, :20]:
    list1.append(terms[j])
  wc = WordCloud(background_color="black", max_words=len(str(list1)), s
topwords=stopwords_t)
  wc.generate(str(list1))
  print("Word Cloud for KMeans Cluster:", i)
  plt.imshow(wc, interpolation='bilinear')
  plt.axis("off")
  plt.show()
  list1.clear()
```

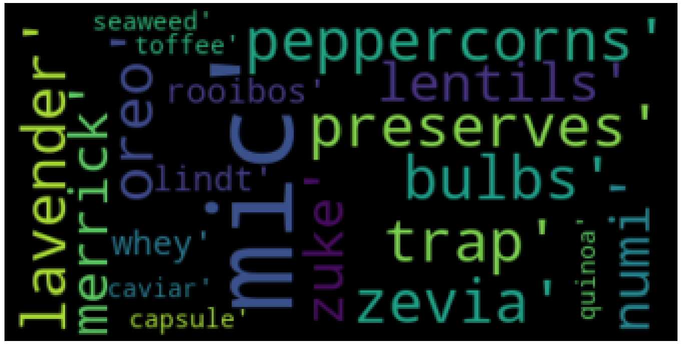
```
Cluster 0:Word Cloud for KMeans Cluster: 0
```




```
Cluster 1:Word Cloud for KMeans Cluster: 1
```

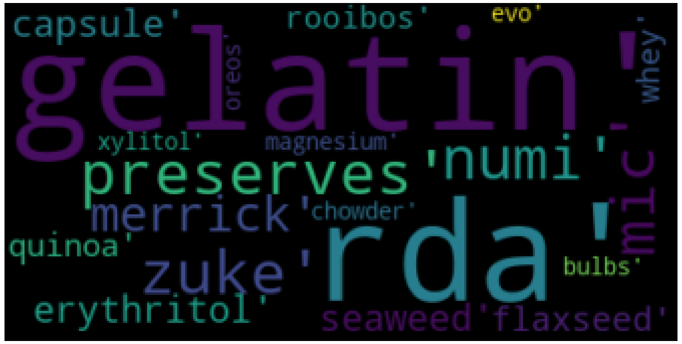



```
Cluster 2:Word Cloud for KMeans Cluster: 2
```



Cluster 3:Word Cloud for KMeans Cluster: 3

Cluster 4:Word Cloud for KMeans Cluster: 4



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

In [0]:
```
def Similarity_Matrix(Index_value):
  cos_sim=cosine_similarity(U[Index_value:Index_value+1],U)
  cos_sim=np.ravel(cos_sim)
  cos_sim_sorted_index=np.argsort(cos_sim)[::-1][0:4]
  print("You selected the word :",df_co_occurance_matrix.index[Index_va
lue])
  print('Most similar word near to ',df_co_occurance_matrix.index[Index
_value],'is ',df_co_occurance_matrix.index[cos_sim_sorted_index[1]])
  print('Cosine similarity(',df_co_occurance_matrix.index[Index_value],
',',df_co_occurance_matrix.index[cos_sim_sorted_index[1]],') =',cos_sim
[cos_sim_sorted_index[1]])
```

In [128]:
```
#You need to select the index ,the below function will print the select
ed word and most similar word to that word
#example 1:
Similarity_Matrix(0)
```

```
You selected the word : ability
Most similar word near to  ability is  everyday
Cosine similarity( ability , everyday ) = 0.9427291161121648
```

In [129]:
```
#example 3
Similarity_Matrix(2500)
```

```
You selected the word : staple
Most similar word near to  staple is  used
Cosine similarity( staple , used ) = 0.9669067144843613
```

[6] Conclusions

In [0]:
```
#clustor 0 : this cluster has words like traps ,toffe ,cheek orea
#cluster 1 :this has words like capsule ,puzzle etc
#cluster 2:this has words like bulbs ,tablets etc
#cluster 3 :this has words like origin etc
```

```
#cluster 4 : this has words like geletin ,levender etc

#words are complex and has no meaning in each cluster
#i dont know is the spelling mistake of the words or those words are at
 top and highlighting
```