#11_Amazon_Fine_Food_Reviews_Analysis_Truncated_SVD

May 29, 2019

1 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.

2 [1]. Reading Data

2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

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


        import sqlite3
        import pandas as pd
        import numpy as np
        import nltk
        import string
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.feature_extraction.text import TfidfTransformer
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.metrics.pairwise import cosine_similarity

        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
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import roc_auc_score
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.decomposition import TruncatedSVD
        from sklearn.cluster import KMeans
        from wordcloud import WordCloud, STOPWORDS

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

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

        from tqdm import tqdm
        import os
```

```
In [0]: 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=947318989803-6

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

```
In [0]: # using SQLite Table to read data.
        con = sqlite3.connect('drive/My Drive/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 point
        # 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 5
        # for tsne assignment you can take 5k data points

        filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 200

        # Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
        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 (200000, 10)
```

```
Out[0]:    Id  ...                                               Text
        0   1  ...  I have bought several of the Vitality canned d...
        1   2  ...  Product arrived labeled as Jumbo Salted Peanut...
        2   3  ...  This is a confection that has been around a fe...

        [3 rows x 10 columns]
```

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

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

(80668, 7)


Out[0]:                UserId  ... COUNT(*)
        0  #oc-R115TNMSPFT9I7  ...        2
        1  #oc-R11D9D7SHXIJB9  ...        3
        2  #oc-R11DNU2NBKQ23Z  ...        2
        3  #oc-R11O5J5ZVQE25C  ...        3
        4  #oc-R12KPBODL2B5ZD  ...        2

        [5 rows x 7 columns]

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

Out[0]:               UserId  ... COUNT(*)
        80638  AZY10LLTJ71NX  ...        5

        [1 rows x 7 columns]

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

Out[0]: 393063
```

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

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

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

Out[0]:        Id  ...                                                Text
        0   78445  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        1  138317  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        2  138277  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        3   73791  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        4  155049  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...

        [5 rows x 10 columns]
```

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=Fal
```

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

```
Out[0]: (160178, 10)
```

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

```
Out[0]: 80.089
```

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

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

        display.head()
```

```
Out[0]:       Id  ...                                               Text
        0  64422  ...  My son loves spaghetti so I didn't hesitate or...
        1  44737  ...  It was almost a 'love at first bite' - the per...

        [2 rows x 10 columns]
```

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

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

(160176, 10)


Out[0]: 1    134799
        0     25377
        Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [3.1]. Preprocessing Review Text

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

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

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

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

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

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

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

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

```
I remembered this book from my childhood and got it for my kids.  It's just as good as I rememb
==================================================
The qualitys not as good as the lamb and rice but it didn't seem to bother his stomach, you get
==================================================
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loan-word, and&quot;ko-&quo
==================================================
What can I say... If Douwe Egberts was good enough for my dutch grandmother, it's perfect for m
==================================================
```

```
In [0]: # 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)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as good as I rememb
```

```
In [0]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-
        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)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as good as I rememb
==================================================
The qualitys not as good as the lamb and rice but it didn't seem to bother his stomach, you get
==================================================
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loan-word, and"ko-"  is "ch
==================================================
What can I say... If Douwe Egberts was good enough for my dutch grandmother, it's perfect for m
```

```
In [0]: # https://stackoverflow.com/a/47091490/4084039
        import re

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

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

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

```
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loan-word, and&quot;ko-&quo
==================================================
```

```
In [0]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
        sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
        print(sent_0)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as good as I rememb
```

```
In [0]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
        sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
        print(sent_1500)
```

```
This is the Japanese version of breadcrumb pan bread a Portuguese loan word and quot ko quot is
```

```
In [0]: # 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',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "t
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", '
            'won', "won't", 'wouldn', "wouldn't"])
```

```
In [0]: # Combining all the above stundents
        from tqdm import tqdm
        preprocessed_reviews = []
        # tqdm is for printing the status bar
        for sentance in tqdm(final['Text'].values):
            sentance = re.sub(r"http\S+", "", sentance)
            sentance = BeautifulSoup(sentance, 'lxml').get_text()
            sentance = decontracted(sentance)
            sentance = re.sub("\S*\d\S*", "", sentance).strip()
            sentance = re.sub('[^A-Za-z]+', ' ', sentance)
            # https://gist.github.com/sebleier/554280
            sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwo
            preprocessed_reviews.append(sentance.strip())
```

```
100%|| 160176/160176 [01:15<00:00, 2133.20it/s]
```

```
In [0]: preprocessed_reviews[100000]
```

```
Out[0]: 'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [0]: #BoW
        count_vect = CountVectorizer() #in scikit-learn
        count_vect.fit(preprocessed_reviews)
        print("some feature names ", count_vect.get_feature_names()[:10])
        print('='*50)
        final_counts = count_vect.transform(preprocessed_reviews)
        print("the type of count vectorizer ",type(final_counts))
        print("the shape of out text BOW vectorizer ",final_counts.get_shape())
        print("the number of unique words ", final_counts.get_shape()[1])
```

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

5.2 [4.2] Bi-Grams and n-Grams.

```
In [0]: #bi-gram, tri-gram and n-gram

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

        # 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_
```

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

5.3 [4.3] TF-IDF

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

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

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

5.4 [4.4] Word2Vec

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

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

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


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

        is_your_ram_gt_16g=False
        want_to_use_google_w2v = False
        want_to_train_w2v = True

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

[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.99460321664810
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.999275088310
```

```
In [0]: w2v_words = list(w2v_model.wv.vocab)
```

```
        print("number of words that occured minimum 5 times ",len(w2v_words))
        print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

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

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


4986
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        tf_idf_matrix = model.fit_transform(preprocessed_reviews)
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # 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 li
        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
```

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

5.6 Truncated-SVD

5.6.1 [5.1] Taking top features from TFIDF, SET 2

```
In [0]: # Please write all the code with proper documentation
        X = preprocessed_reviews[:]
        y = final['Score'][:]

        tf_idf = TfidfVectorizer()
        tfidf_data = tf_idf.fit_transform(X)

        tfidf_feat = tf_idf.get_feature_names()
```

5.6.2 [5.2] Calulation of Co-occurrence matrix

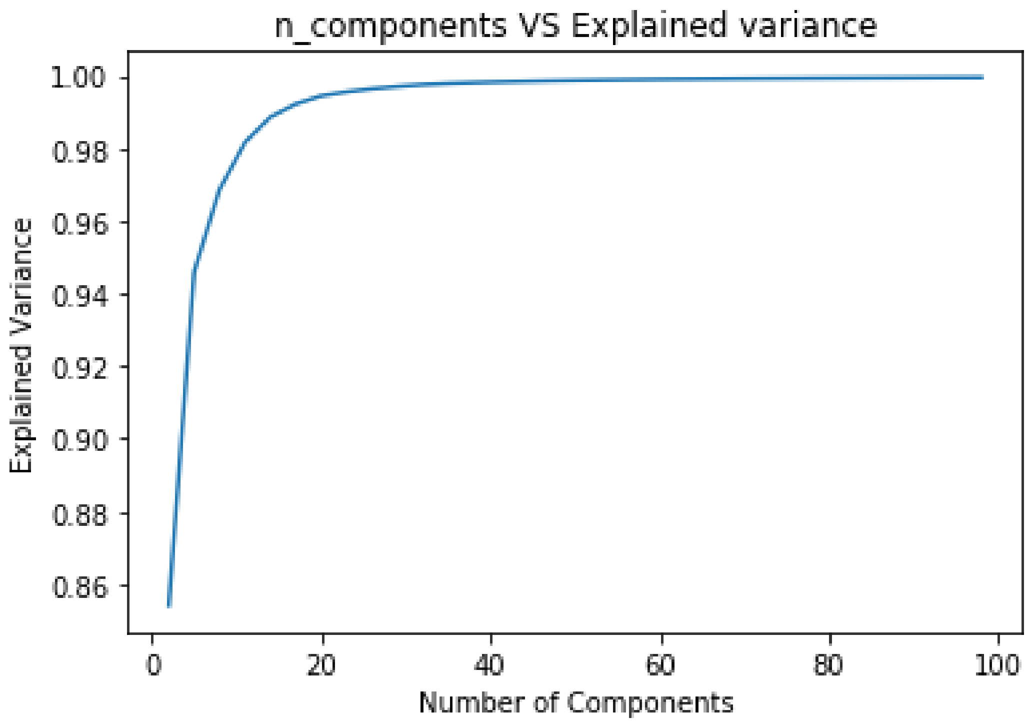
```
In [0]: # Please write all the code with proper documentation
        #Ref:https://datascience.stackexchange.com/questions/40038/how-to-implement-word-to-wo
        #Ref:# https://github.com/PushpendraSinghChauhan/Amazon-Fine-Food-Reviews/blob/master/

        def Co_Occurrence_Matrix(neighbour_num , list_words):

                # Storing all words with their indices in the dictionary
                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 list_words:
            # row in co-occurrence matrix
            temp = []
            # column in co-occurrence matrix
            for col in list_words :
                if( col != row):
                    # No. of times col word is in neighbourhood of row word
                    count = 0
                    # Value of neighbourhood
                    num = neighbour_num
                    # 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)
        # Return co-occurrence matrix
        return np.array(matrix)
```

```
In [0]: X_new = Co_Occurrence_Matrix(15, top_feat)
```

5.6.3 [5.3] Finding optimal value for number of components (n) to be retained.

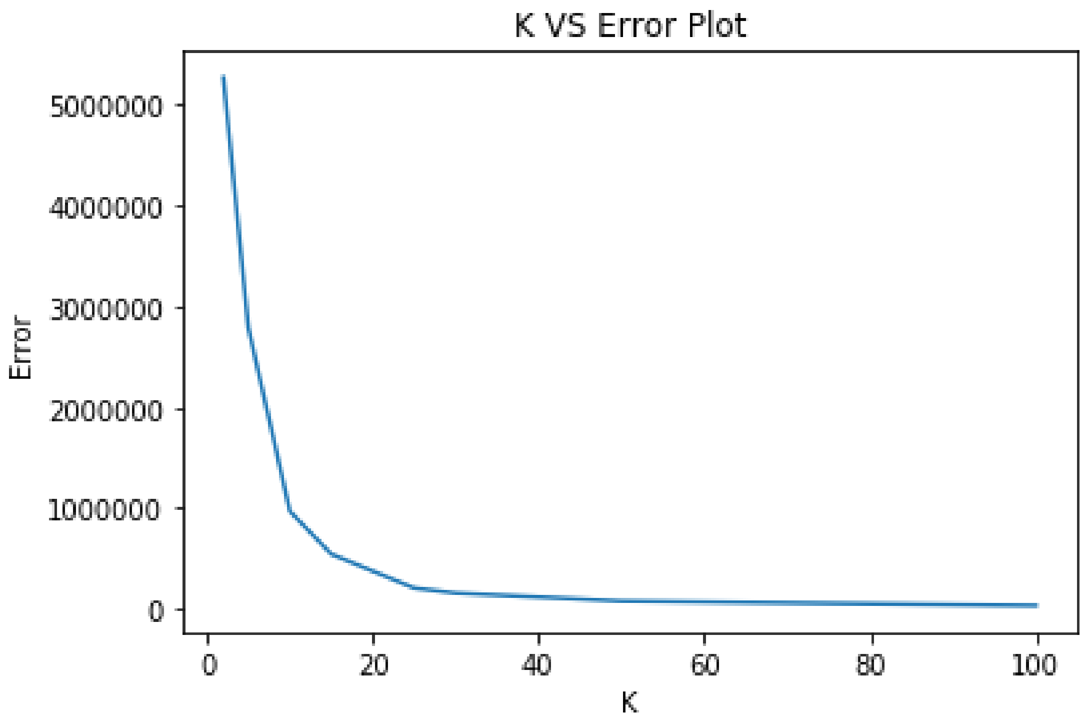
```
In [0]: # Please write all the code with proper documentation
        k = np.arange(2,100,3)

        variance =[]
        for i in k:
            svd = TruncatedSVD(n_components=i)
            svd.fit_transform(X_new)
            score = svd.explained_variance_ratio_.sum()
            variance.append(score)

        plt.plot(k, variance)
        plt.xlabel('Number of Components')
        plt.ylabel('Explained Variance')
        plt.title('n_components VS Explained variance')
        plt.show()
```



5.6.4 [5.4] Applying k-means clustering

```
In [0]: # Please write all the code with proper documentation

        errors = []
        k = [2, 5, 10, 15, 25, 30, 50, 100]
```

```
for i in k:
    kmeans = KMeans(n_clusters=i, random_state=0)
    kmeans.fit(X_new)
    errors.append(kmeans.inertia_)

plt.plot(k, errors)
plt.xlabel('K')
plt.ylabel('Error')
plt.title('K VS Error Plot')
plt.show()
```

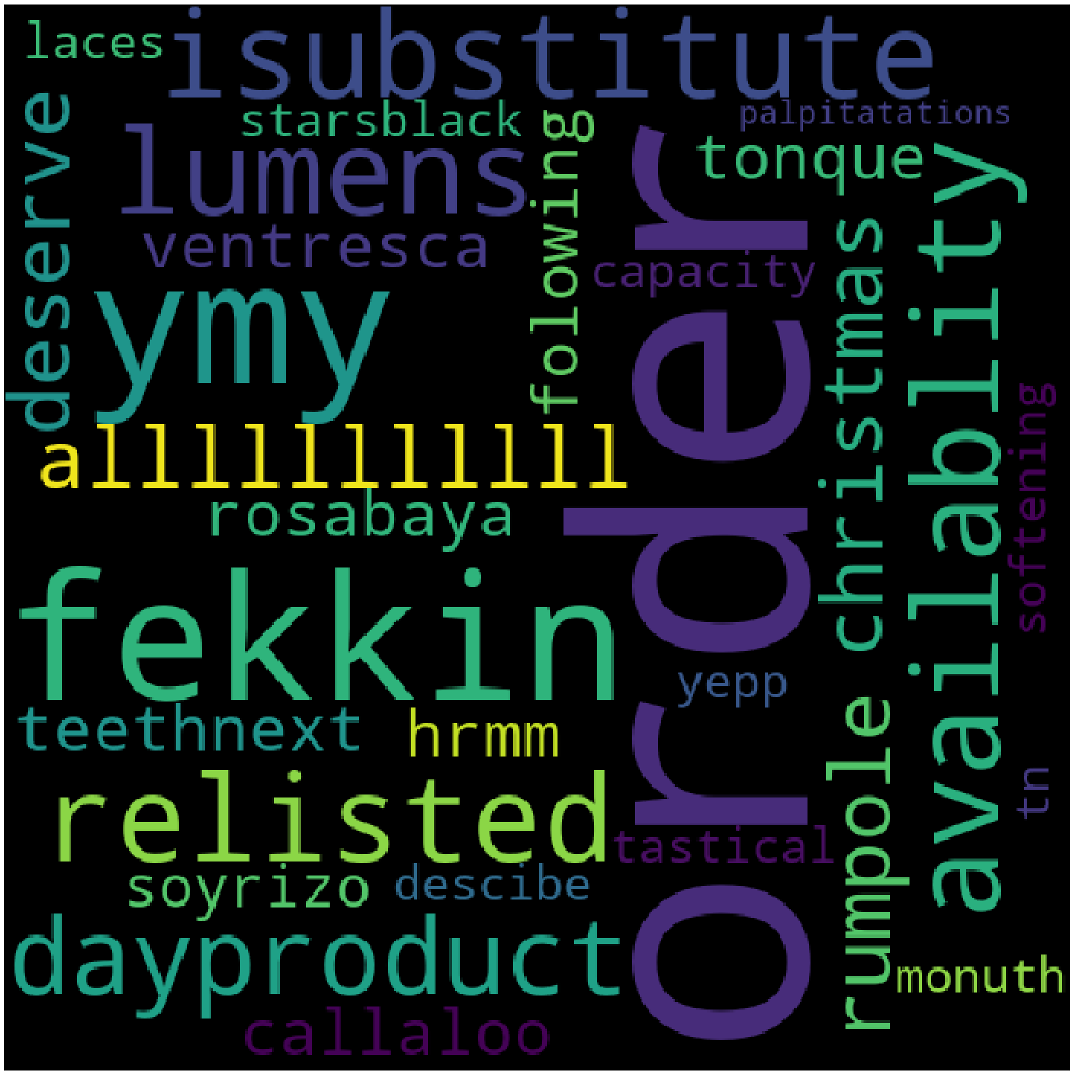


```
In [0]: svd = TruncatedSVD(n_components = 20)
        svd.fit(X_new)

        score = svd.explained_variance_ratio_
```

5.6.5 [5.5] Wordclouds of clusters obtained in the above section

```
In [0]: # Please write all the code with proper documentation

        indices = np.argsort(tf_idf.idf_[::-1])
        top_feat = [tfidf_feat[i] for i in indices[0:3000]]
        top_indices = indices[0:3000]
```

```
top_n = np.argsort(top_feat[::-1])

feature_importances = pd.DataFrame(top_n, index = top_feat, columns=['importance']).so
top = feature_importances.iloc[0:30]
comment_words = ' '
for val in top.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '
        stopwords = set(STOPWORDS)



wordcloud = WordCloud(width = 600, height = 600,
                background_color ='black',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

plt.figure(figsize = (10, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



5.6.6 [5.6] Function that returns most similar words for a given word.

```
In [0]: # Please write all the code with proper documentation
        def similarity(word):
            similarity = cosine_similarity(X_new)
            word_vect = similarity[top_feat.index(word)]
            index = word_vect.argsort()[::-1][1:5]
            for i in range(len(index)):
                print((i+1),top_feat[index[i]] ,"\n")
```

```
In [0]:  similarity('sugary')
```

```
1 extra
```

```
2 needs

3 craving

4 care
```

```
In [0]:  similarity('notlike')
```

```
1 maytag

2 slip

3 gibbles

4 farriage
```

6 [6] Conclusions

```
In [0]: # Please write down few lines about what you observed from this assignment.
        # Also please do mention the optimal values that you obtained for number of components
```

```
In [2]: from prettytable import PrettyTable

        x = PrettyTable()

        x.field_names = ["Algorithm","Best Hyperparameter"]

        x.add_row(["T-SVD",  20])
        x.add_row(["K-Means",  20])



        print(x)
```

```
+-----------+---------------------+
| Algorithm | Best Hyperparameter |
+-----------+---------------------+
|   T-SVD   |          20         |
|  K-Means  |          20         |
+-----------+---------------------+
```

- It can be obseverd that just 20 components preserve about 99.9% of the variance in the data.
- The co occurence matrix used is to find the correlation of one word with respect to the other in the dataset.