Amazon_Fine_Food_Reviews_Analysis_Truncated_SVD

July 25, 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.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

        # importing Cross validation libs
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import cross_val_score
        from sklearn import model_selection

        from matplotlib import pyplot
        import seaborn as sns

        import numpy as np
        from sklearn.cluster import KMeans
        from wordcloud import WordCloud
        from sklearn.decomposition import TruncatedSVD
        from sklearn.metrics.pairwise import cosine_similarity
        import pickle
```

```
In [53]: # Code to read csv file into colaboratory:
         !pip install -U -q PyDrive
         from pydrive.auth import GoogleAuth
         from pydrive.drive import GoogleDrive
         from google.colab import auth
         from oauth2client.client import GoogleCredentials

         # 1. Authenticate and create the PyDrive client.
         auth.authenticate_user()
         gauth = GoogleAuth()
         gauth.credentials = GoogleCredentials.get_application_default()
         drive = GoogleDrive(gauth)

         # load SQLite
         id = '1VDFn9tpwOOecnr5DiC7TQwg-GVoPJUEK'

         downloaded = drive.CreateFile({'id':id})
         downloaded.GetContentFile('database.sqlite')

         # df2 = pd.read_csv(io.BytesIO(uploaded['Filename.csv']))

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

         filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 10

         # Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative
         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 (100000, 10)
```

```
Out[53]:    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 [4]: print(display.shape)
        display.head()
```

```
(80668, 7)
```

```
Out[4]:                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 [5]: display[display['UserId']=='AZY10LLTJ71NX']
```

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

        [1 rows x 7 columns]
```

```
In [6]: display['COUNT(*)'].sum()
```

```
Out[6]: 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 [7]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
        ORDER BY ProductID
        """, con)
        display.head()
```

```
Out[7]:        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 [9]: #Deduplication of entries
        final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
        final.shape
```

```
Out[9]: (87775, 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]: 87.775
```

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  ...                                               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 [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()

(87773, 10)


Out[13]: 1    73592
         0    14181
         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 [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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid
==================================================
```

```
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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

```
In [16]: # 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afrai
```

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

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

```
In [19]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

```
In [20]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
         print(sent_1500)
```

```
was way to hot for my blood took a bite and did a jig lol
```

```
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]: #filtered out whole reviews

        from bs4 import BeautifulSoup
        # Combining all the above stundents
        from tqdm import tqdm
        # tqdm is for printing the status bar
        word_counter = []
        def filterised_text(text):
            preprocessed_text = []
            for sentance in tqdm(text):
                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 st
                count = len(sentance.split())
                word_counter.append(count)
                preprocessed_text.append(sentance.strip())
            return preprocessed_text
```

```
In [23]: preprocessed_reviews = filterised_text(final['Text'].values)
         final['preprocessed_reviews']  = preprocessed_reviews
         preprocessed_reviews[18]
```

```
100%|| 87773/87773 [00:39<00:00, 2224.95it/s]
```

```
Out[23]: 'could rate fly trap lower one star would think flies come miles away come laugh thin
```

4.2 [3.2] Preprocessing Review Summary

```
In [24]: preprocessed_summary = filterised_text(final['Summary'].values)
         final['preprocessed_summary']  = preprocessed_summary
         preprocessed_summary[18]
```

```
100%|| 87773/87773 [00:28<00:00, 3124.85it/s]
```

```
Out[24]: 'day zero flies'
```

5 [4] Featurization

5.1 [4.3] TF-IDF

```
In [27]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,1), min_df=10, use_idf=True)
         tf_idf_vect.fit(preprocessed_reviews)
         print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_name
         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) ['aa', 'aafco', 'aback', 'abandon', 'abandone
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (87773, 11524)
the number of unique words including both unigrams and bigrams  11524
```

6 [5] Assignment 11: Truncated SVD

```
In [0]: K = list(range(2, 13, 2))

        def finding_best_k(X_tr):

          train_ =[]

          for i in K:
              tr_kmeans = KMeans(n_clusters=i,verbose=1)
              tr_kmeans.fit(X_tr)

              train_inertia =  tr_kmeans.inertia_

              train_.append(train_inertia)
          return train_

        def plotErrorplot(train_):
          plt.plot(K, train_, label='Train AUC')
          plt.legend()
          plt.xlabel("K: hyperparameter")
          plt.ylabel("Inertia")
          plt.grid()
          plt.title("ERROR PLOTS")
          plt.show()


        # fetch result from dataframe, where matchlabel== expected label, finally convert to s
        def returnWordCloudString(df, matchlabel):
            return df['Features'][df.label == matchlabel].to_string()


        def plotWordCloud(text):

          #  https://www.geeksforgeeks.org/generating-word-cloud-python/

          wordcloud = WordCloud(width = 800, height = 800,
                          background_color ='black',
                          min_font_size = 10).generate(text)

          # plot the WordCloud image
          plt.figure(figsize = (10, 10), facecolor = None)
          plt.imshow(wordcloud)
          plt.axis("off")
          plt.tight_layout(pad = 0)

          plt.show()
```

6.1 Truncated-SVD

6.1.1 [5.1] Taking top features from TFIDF

```
In [29]: top_words_freq =tf_idf_vect.idf_
         print(top_words_freq.shape)

         top_word_count = 2000

         features = tf_idf_vect.get_feature_names()

         top_df = pd.DataFrame({'Features' : features, 'Idf_score' : top_words_freq})
         top_df = top_df.sort_values("Idf_score", ascending = True)[:top_word_count]
         print("shape of selected features :", top_df.shape)

         top2000 = top_df['Features'][:2000]
```

```
(11524,)
shape of selected features : (2000, 2)
```

6.1.2 [5.2] Calulation of Co-occurrence matrix

```
In [35]: # https://stackoverflow.com/questions/41661801/python-calculate-the-co-occurrence-mat
         # https://github.com/premvardhan/Amazon-fine-food-review-analysis/blob/master/Truncat
         # https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285
         # https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/


         co_occurence_matrix = np.zeros((len(top_df), len(top_df)))
         preprocess_review_df = pd.DataFrame(co_occurence_matrix, index = top_df["Features"], c

         window_size = 6
         for sent in tqdm(final["preprocessed_reviews"]):
             word = sent.split(" ")
             for i, d in enumerate(word):
                 for j in range(max(i - window_size, 0), min(i + window_size, len(word))):
                     if (word[i] != word[j]):
                             try:
                                 preprocess_review_df.loc[word[i], word[j]] += 1
                             except:
                                 pass
```

```
100%|| 87773/87773 [2:48:11<00:00,  8.70it/s]
```

```
In [0]: import pickle
        output = open('trunctSVD.pkl', 'wb')
        pickle.dump(preprocess_review_df,output)
        from pydrive.auth import GoogleAuth
        from pydrive.drive import GoogleDrive
        from google.colab import auth
        from oauth2client.client import GoogleCredentials

        # 1. Authenticate and create the PyDrive client.
        auth.authenticate_user()
        gauth = GoogleAuth()
        gauth.credentials = GoogleCredentials.get_application_default()
        drive = GoogleDrive(gauth)
        folder_id = '1WSEkuSi3HSyJ_4v0UH_JdyTZRQmeqHdj'
        # get the folder id where you want to save your file
        file = drive.CreateFile({'parents':[{u'id': folder_id}]})
        file.SetContentFile('trunctSVD.pkl')
        file.Upload()
```

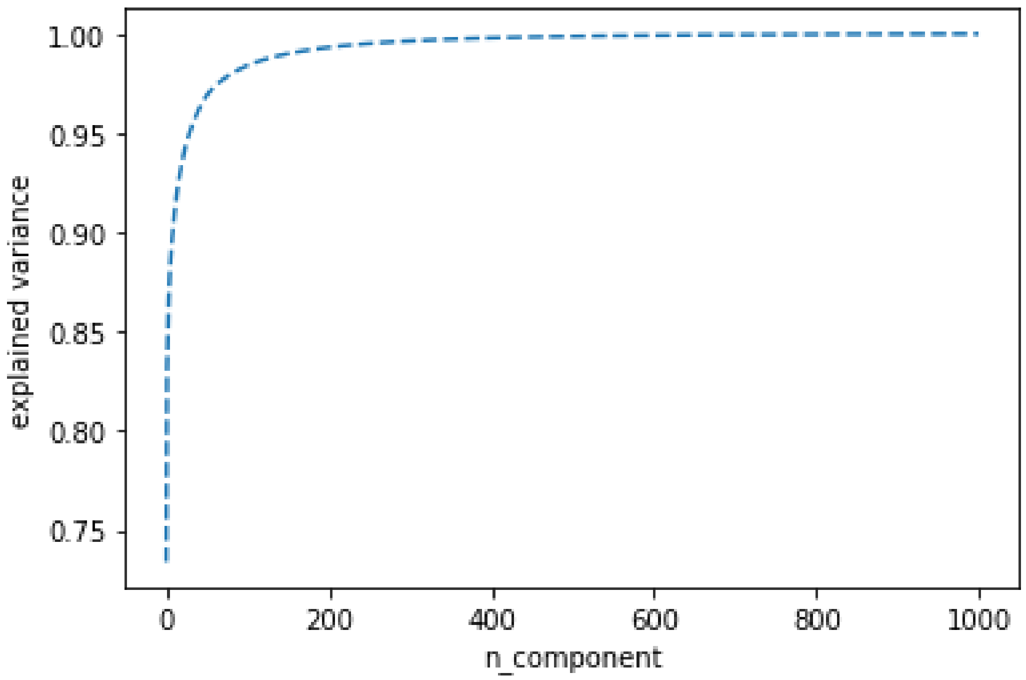
```
In [38]: preprocess_review_df.head()
```

```
Out[38]: Features      not     like    good   great  ...  email  allowed  kiwi  eaters
         Features                                  ...
         not           0.0  15880.0  9529.0  5361.0  ...   72.0     83.0  94.0    79.0
         like      15934.0      0.0  3532.0  2112.0  ...    7.0     14.0  54.0    29.0
         good       9549.0   3525.0     0.0  1932.0  ...   11.0     10.0  26.0    16.0
         great      5386.0   2109.0  1914.0     0.0  ...    5.0     11.0  11.0    13.0
         one        7012.0   2904.0  2134.0  1380.0  ...    9.0     18.0  16.0    18.0

         [5 rows x 2000 columns]
```

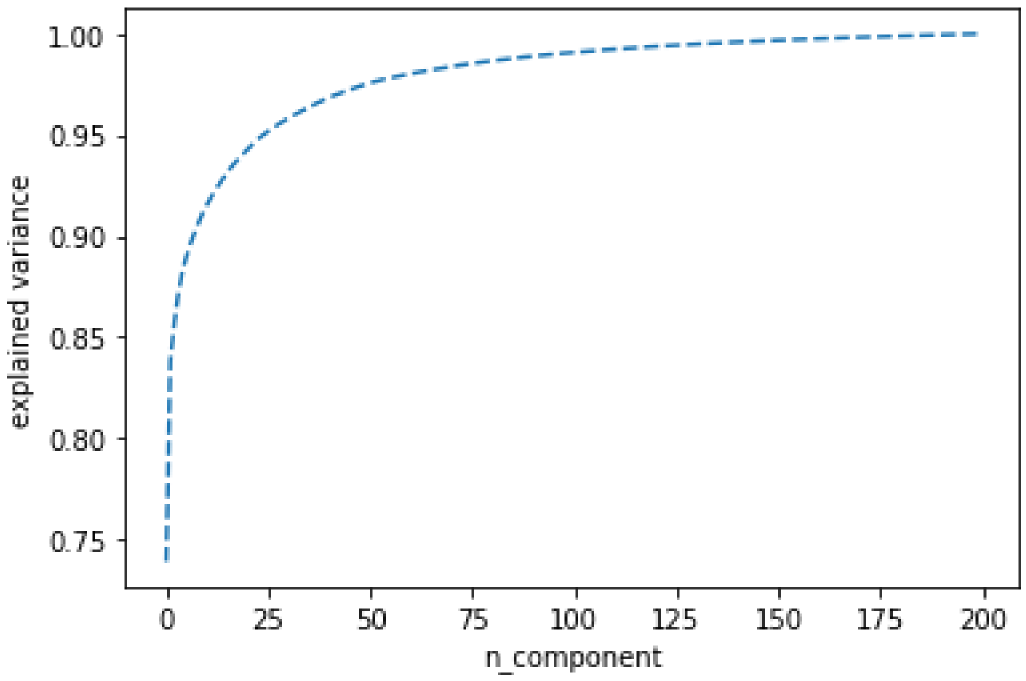
```
In [0]: top2000 = []

        for col in preprocess_review_df.columns:
            top2000.append(col)
```

6.1.3 [5.3] Finding optimal value for number of components (n) to be retained.

```
In [40]: # TrucatedSVD

         tsvd = TruncatedSVD(n_components = 1000)
         ts_data = tsvd.fit_transform(preprocess_review_df)

         var_explained = tsvd.explained_variance_ / np.sum(tsvd.explained_variance_)

         cum_var_explained = np.cumsum(var_explained)

         plt.plot(cum_var_explained,"--")
         plt.xlabel('n_component')
         plt.ylabel('explained variance')
         plt.show()
```



```
In [41]: # TrucatedSVD

         tsvd = TruncatedSVD(n_components = 200)
         ts_data = tsvd.fit_transform(preprocess_review_df)

         var_explained = tsvd.explained_variance_ / np.sum(tsvd.explained_variance_)

         cum_var_explained = np.cumsum(var_explained)

         plt.plot(cum_var_explained,"--")
         plt.xlabel('n_component')
         plt.ylabel('explained variance')
         plt.show()
```



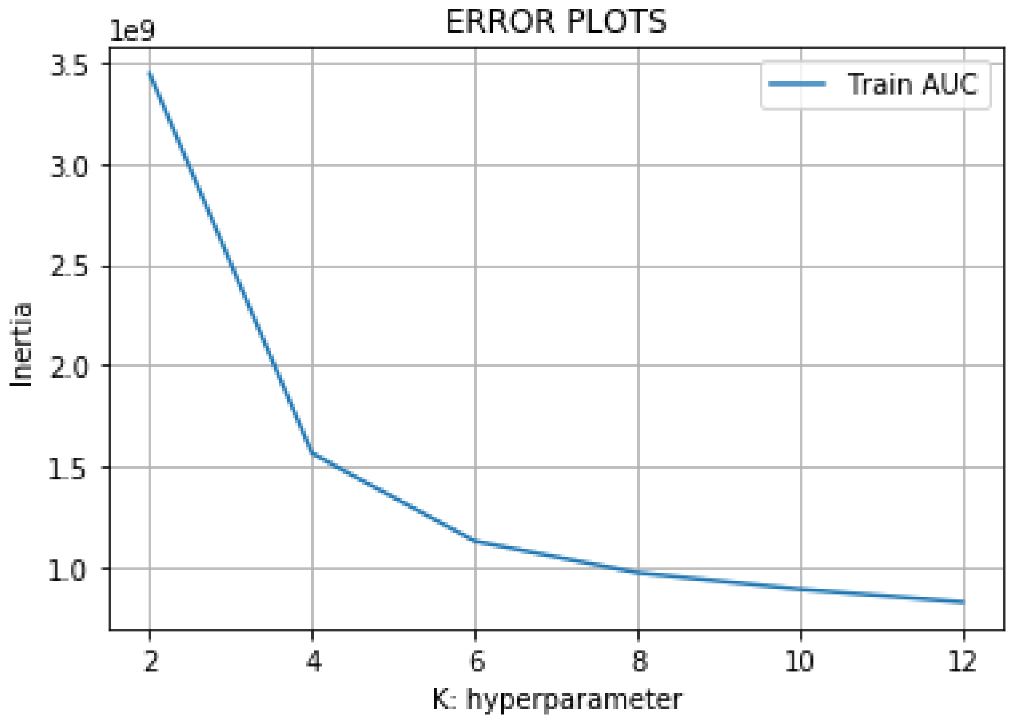
From above graph, it can observe that only 90 components can explain almost 99% of variance.

```
In [0]: # TrucatedSVD

        tsvd = TruncatedSVD(n_components = 90)
        ts_data = tsvd.fit_transform(preprocess_review_df)

        var_explained = tsvd.explained_variance_ / np.sum(tsvd.explained_variance_)

        cum_var_explained = np.cumsum(var_explained)
```

6.1.4 [5.4] Applying k-means clustering

```
In [0]: bow_train = finding_best_k(ts_data)
```

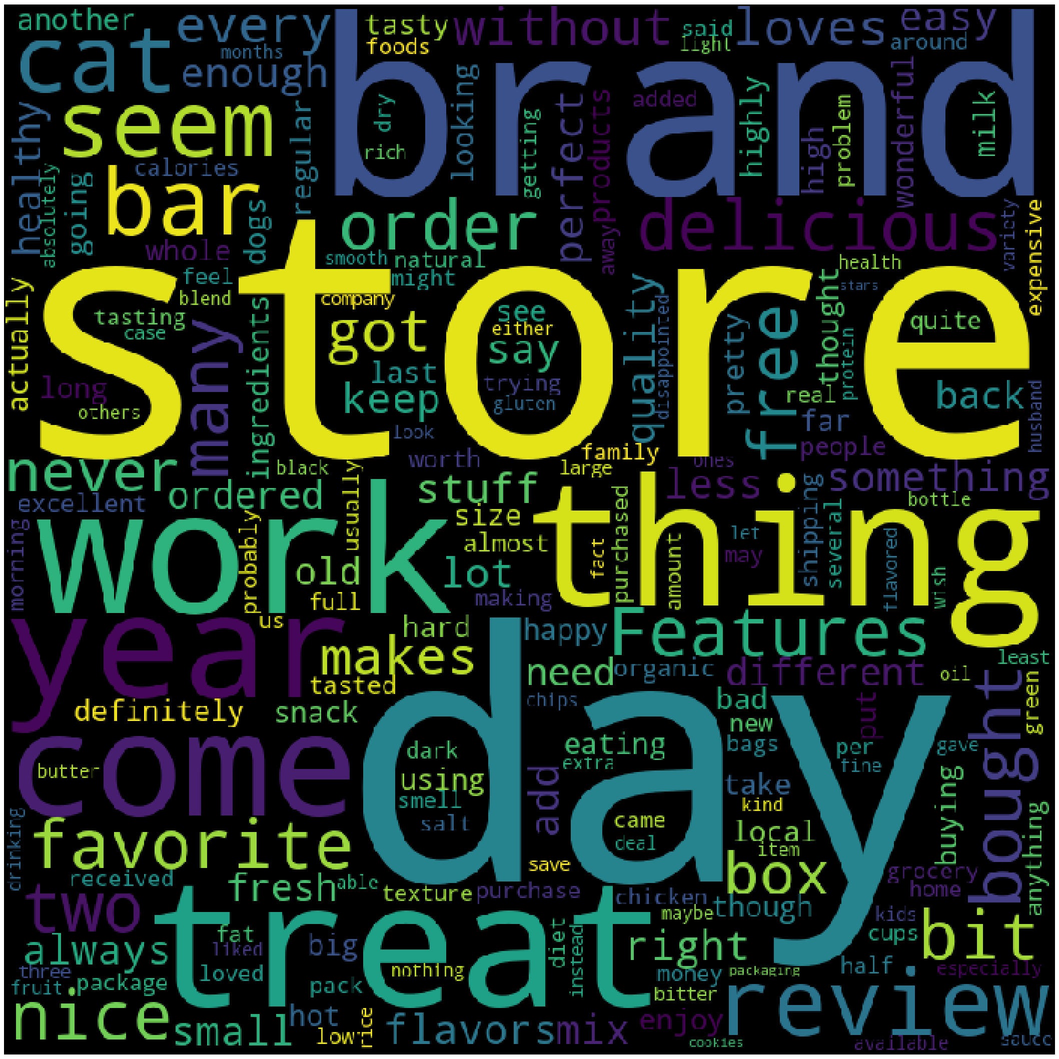
```
In [44]: plotErrorplot(bow_train)
```

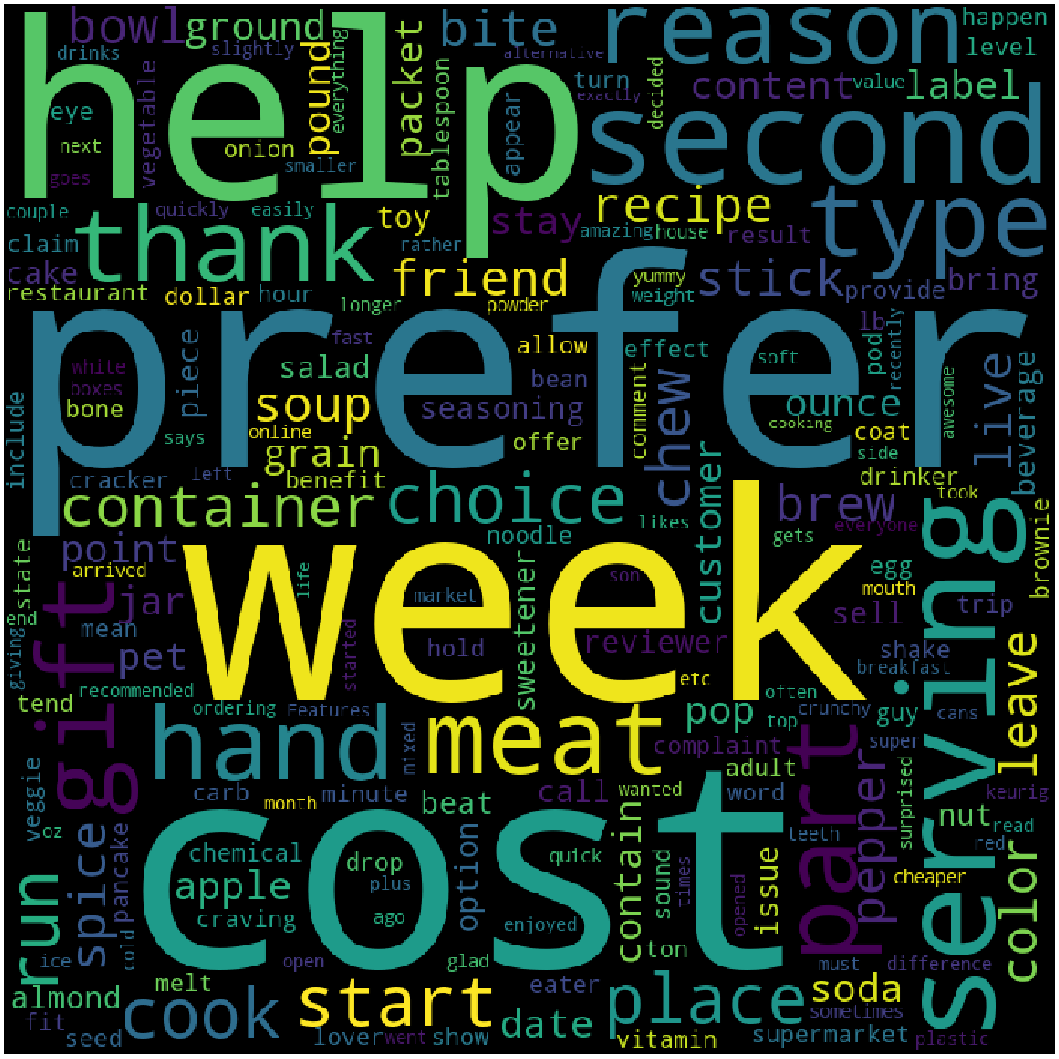


```
In [45]: kmeans = KMeans(n_clusters=6, n_jobs=-1).fit(ts_data)
         labels = kmeans.labels_

         no_of_clusters = set(labels.flatten())
         no_of_clusters

Out[45]: {0, 1, 2, 3, 4, 5}

In [0]: # create duplicate copy of original dataframe as final2
        preprocess_review_df2 = preprocess_review_df.copy()

        # adding this labels result to duplicate copy dataframe
        preprocess_review_df2['label']= labels

        preprocess_review_df2['Features'] = top_df['Features']

        # remove nan
        preprocess_review_df2 = preprocess_review_df2.replace(np.nan, '', regex=True)


        # fetch reviews, those label==0, also convert to string
        cluster_0 = returnWordCloudString(preprocess_review_df2,0)

        # fetch reviews, those label==1, also convert to string
        cluster_1 = returnWordCloudString(preprocess_review_df2,1)

        # fetch reviews, those label==2, also convert to string
        cluster_2 = returnWordCloudString(preprocess_review_df2,2)

        # fetch reviews, those label==3, also convert to string
        cluster_3 = returnWordCloudString(preprocess_review_df2,3)

        # fetch reviews, those label==4, also convert to string
        cluster_4 = returnWordCloudString(preprocess_review_df2,4)

        # fetch reviews, those label==5, also convert to string
        cluster_5 = returnWordCloudString(preprocess_review_df2,5)

        cluster_list = [cluster_0,cluster_1,cluster_2,cluster_3,cluster_4,cluster_5]
```

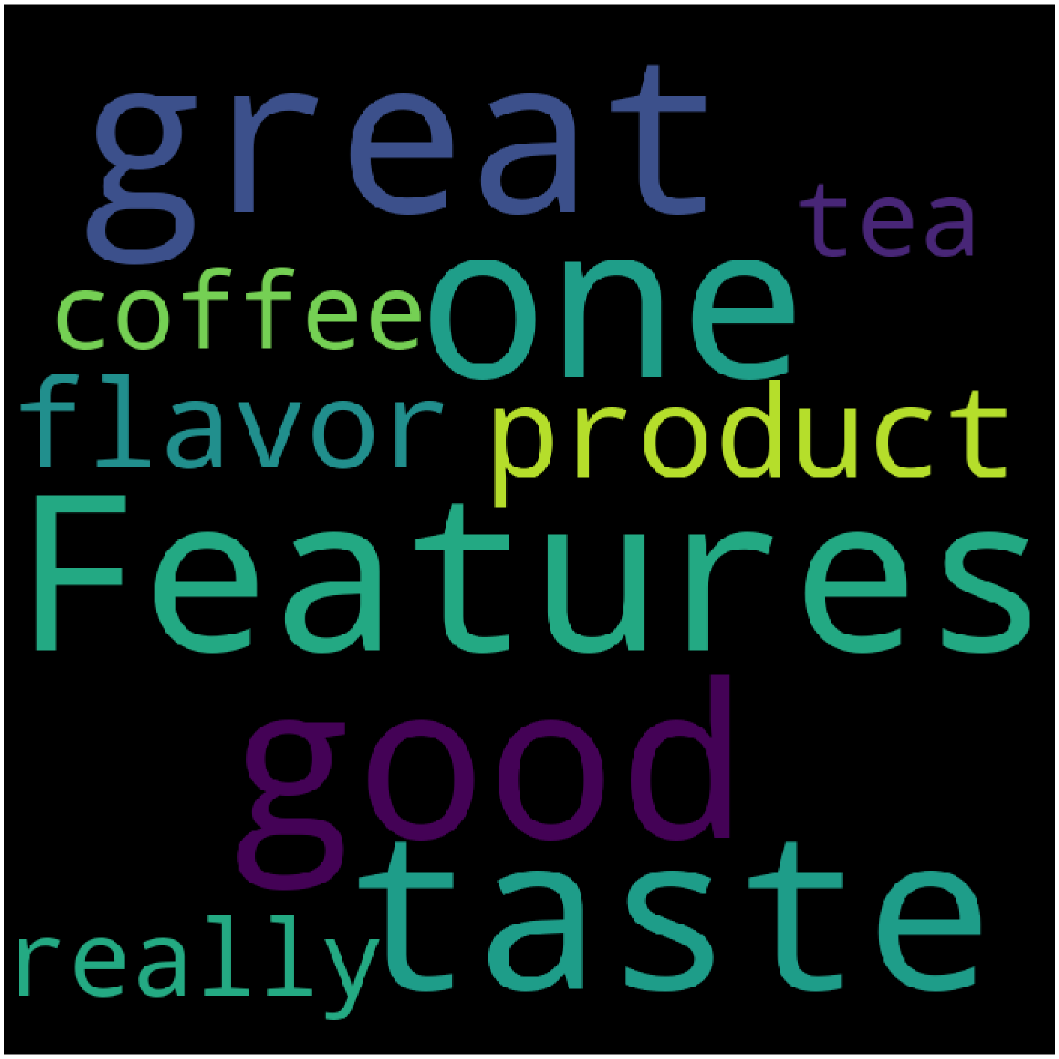
6.1.5 [5.5] Wordclouds of clusters obtained in the above section

```
In [47]: for i in cluster_list:
             plotWordCloud(i)
```

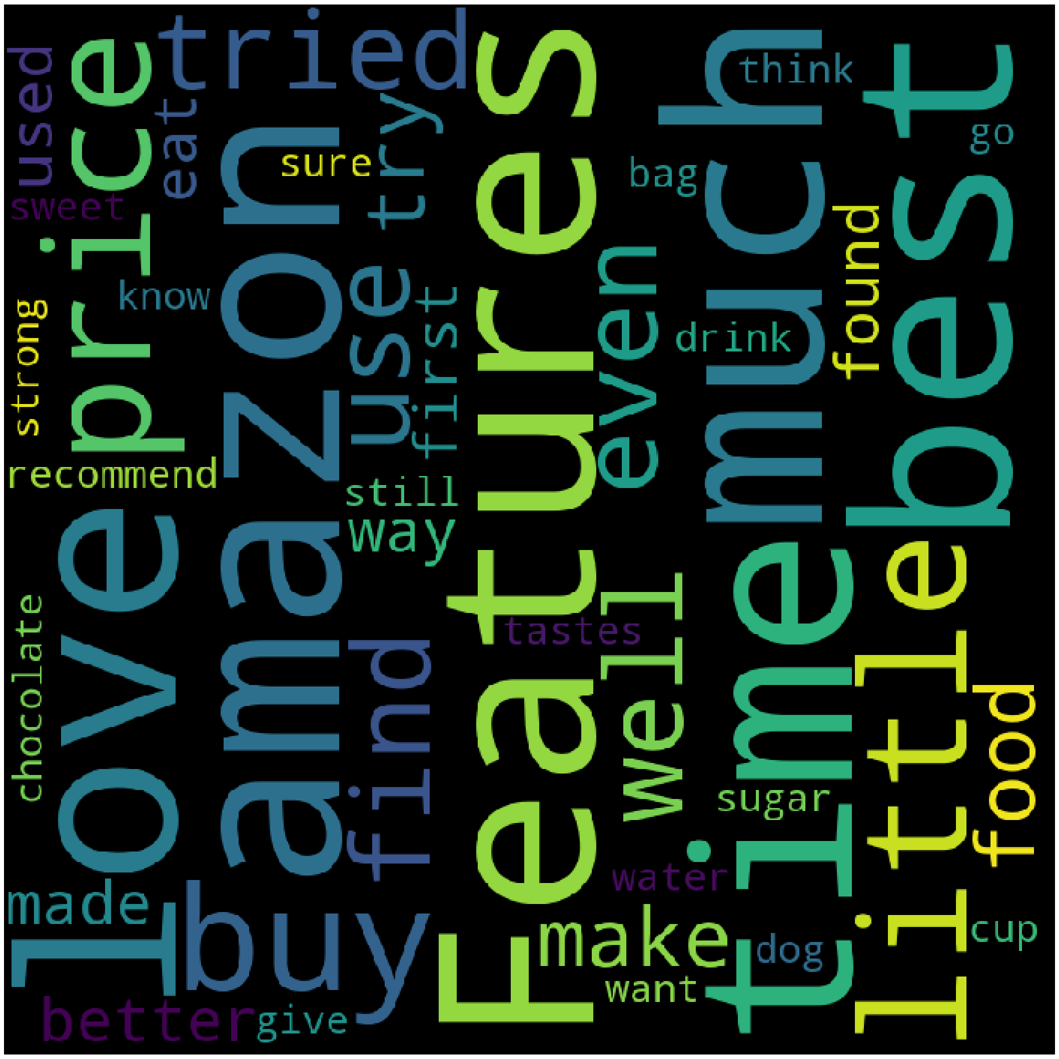

store
day
brand
treat
work
thing
year
come
review
delicious
favorite
seem
bar
cat
order
Features
many
never
got
free
makes
nice
two
bit
box
every
enough
without
loves
easy
another
tasty
foods
healthy
say
keep
quality
something
stuff
ordered
old
lot
less
different
bought
right
always
fresh
small
flavors
mix
definitely
snack
need
happy
ingredients
perfect
highly
high
problem
wonderful
milk
looking
regular
going
dogs
calories
whole
feel
blend
tasting
case
long
others
actually
excellent
morning
black
probably
usually
size
almost
hard
tasted
full
us
last
see
either
trying
gluten
look
worth
family
large
pretty
thought
protein
real
far
people
back
quite
expensive
health
stars
variety
husband
shipping
several
flavored
wish
purchased
amount
fact
may
left
bottle
least
green
oil
organic
chips
put
bad
new
bags
per
fine
gave
eating
extra
dark
using
smell
salt
add
take
kind
local
item
came
deal
save
buying
anything
grocery
home
chicken
though
texture
purchase
received
butter
drinking
able
fat
liked
big
three
fruit
package
loved
pack
hot
low
rice
nothing
diet
instead
maybe
money
bitter
enjoy
cups
kids
half
packaging
especially
cookies
available
sauce
months
light
said
around
dry
rich
getting
products
away
added
absolutely
smooth
natural
might
company
disappointed
ones
making



help
prefer
week
cost
serving
second
reason
thank
type
hand
meat
gift
part
container
choice
place
start
cook
run
bowl
ground
bite
recipe
friend
stick
label
content
soup
chew
customer
pepper
spice
brew
point
contain
date
soda
color
leave
live
sweetener
packet
pound
salad
grain
piece
ounce
seasoning
reviewer
pop
apple
chemical
option
issue
almond


great
tea
coffee
one
flavor
product
Features
good
really
taste

Features


love
amazon
Feature
time
much
best
little
price
tried
buy
find
well
even
use
try
make
eat
used
first
way
still
recommend
food
found
made
better
give
chocolate
strong
sweet
know
sure
drink
bag
think
go
tastes
sugar
water
dog
cup
want

6.1.6 [5.6] Function that returns most similar words for a given word.

```
In [0]: # https://stackoverflow.com/questions/18424228/cosine-similarity-between-2-number-list
        # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_si

        def findSimilaryWords(word):

          # find the index of word
          ind = top2000.index(word)

          # cosine similarity
          cos_similarity = cosine_similarity(ts_data)
          sim_vect_ = cos_similarity[ind]
          index = sim_vect_.argsort()[::-1][1:10]
          for j in range(len(index)):
              print((j+1),top2000[index[j]] ,"===",word,"\n")
```

```
In [49]: findSimilaryWords('calories')
```

```
1 fiber === calories

2 low === calories

3 sodium === calories

4 content === calories

5 protein === calories

6 carbs === calories

7 iron === calories

8 total === calories

9 reduced === calories
```

```
In [50]: findSimilaryWords('cheaper')
```

```
1 prices === cheaper

2 target === cheaper

3 walmart === cheaper

4 bulk === cheaper

5 sale === cheaper

6 costs === cheaper

7 selling === cheaper

8 sells === cheaper

9 cost === cheaper
```

```
In [51]: findSimilaryWords('good')
```

```
1 amazing === good

2 fine === good

3 however === good

4 awesome === good

5 course === good

6 ok === good

7 type === good

8 appreciate === good

9 totally === good
```

7 [6] Conclusions

7.0.1 Procedure

- We have consider '100k' datapoints for this experiement.
- Firstly took TFIDF weighted vector and find "top 2000" feature using idf_ (**The inverse document frequency vector**)
- Then build Co-Occurance matrix, which have 2000*2000 dimension.
- Then find optimal n_component value using 'elbow' method. we found **n_component = 90**
- Applied this Top feature to KMean algorithm and build cluster. While applying Kmean,firstly hyperparam tune and find number of cluster require again using 'elbow' method, we found **no_clusters = 6**
- plot word cloud on basis of clusters build by Kmean algorithm
- Using Cosine Similarity create function which will return most similar word from given corpus