11 Amazon Fine Food Reviews Analysis_Truncated SVD

May 23, 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 [1]: %matplotlib inline
        import warnings
        warnings.filterwarnings("ignore")


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

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

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

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

        from tqdm import tqdm
        import os
        from wordcloud import WordCloud, STOPWORDS
        from sklearn.metrics.pairwise import cosine_similarity
```

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

```
In [2]: # using SQLite Table to read data.
        con = sqlite3.connect('database.sqlite')
```

```
# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data 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 500
# for tsne assignment you can take 5k data points

#filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50

# 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 (500000, 10)
```

Out[2]:

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

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      1  1303862400
1                     0                       0      0  1346976000
2                     1                       1      1  1219017600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
```

In [3]:

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

In [4]:

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

```
(80668, 7)
```

```
Out[4]:                 UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

```
In [5]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[5]:               UserId   ProductId                       ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                              Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

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

```
3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                      2
4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                      2

   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
1                       2      5  1199577600
2                       2      5  1199577600
3                       2      5  1199577600
4                       2      5  1199577600

                            Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                               Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

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

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

```
In [8]: #Sorting data according to ProductId in ascending order
        sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

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

```
Out[9]: (348262, 10)
```

```
In [10]: #Checking to see how much % of data still remains
         (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]: 69.6524

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

In [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:
```
      Id   ProductId          UserId             ProfileName  \
0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
1  44737  B001EQ55RW  A2V0I904FH7ABY                     Ram

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     3                       1      5  1224892800
1                     3                       2      4  1212883200

                                    Summary  \
0           Bought This for My Son at College
1  Pure cocoa taste with crunchy almonds inside

                                               Text
0  My son loves spaghetti so I didn't hesitate or...
1  It was almost a 'love at first bite' - the per...
```

In [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:
```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

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

```
(348260, 10)
```

Out[13]:
```
1    293516
0     54744
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)
```

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and l
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Arabica.  My vote is defin
==================================================
This is a great product. It is very healthy for all of our dogs, and it is the first food that
==================================================
I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a
==================================================
```

In [15]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
```

```
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and l
```

```
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)
```

```
This book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and l
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Arabica.  My vote is defin
==================================================
This is a great product. It is very healthy for all of our dogs, and it is the first food that
==================================================
I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a
```

```
In [17]: # https://stackoverflow.com/a/47091490/4084039
         import re

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

             # general
             phrase = re.sub(r"n\'t", " not", phrase)
```

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

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

```
This is a great product. It is very healthy for all of our dogs, and it is the first food that
==================================================
```

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)
```

```
This book was purchased as a birthday gift for a  year old boy. He squealed with delight and hu
```

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

```
This is a great product It is very healthy for all of our dogs and it is the first food that t
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
            "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', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', '
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
            '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 [22]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwo
    preprocessed_reviews.append(sentance.strip())
```

```
100%|| 348260/348260 [03:15<00:00, 1777.94it/s]
```

In [23]:
```
preprocessed_reviews[1500]
```

Out[23]:
```
'great product healthy dogs first food love eat helped older dog lose weight year old
```

In [24]:
```
final['Text'] = preprocessed_reviews

#balancing the data
finalp = final[final.Score == 1].sample(50000,random_state =2)
finaln = final[final.Score == 0].sample(50000,random_state =2)
finalx = pd.concat([finalp,finaln],ignore_index=True)
finalx = finalx.sort_values('Time')
X = finalx.Text
```

5 [4] Featurization

5.1 [4.3] TF-IDF

In [25]:
```
tf_idf_vect = TfidfVectorizer(min_df=10)
tfidf = tf_idf_vect.fit_transform(X)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names
print('='*50)
print("the type of count vectorizer ",type(tfidf))
print("the shape of out text TFIDF vectorizer ",tfidf.get_shape())
print("the number of unique words including both unigrams and bigrams ", tfidf.get_sha
```

```
some sample features(unique words in the corpus) ['aa', 'aafco', 'aback', 'abandon', 'abandoned
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

```
the shape of out text TFIDF vectorizer  (100000, 12874)
the number of unique words including both unigrams and bigrams  12874
```

6 [5] Assignment 11: Truncated SVD

```
<li><strong>Apply Truncated-SVD on only this feature set:</strong>
    <ul>
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vector
<br>
<li><strong>Procedure:</strong>
    <ul>
<li>Take top 2000 or 3000 features from tf-idf vectorizers using idf_ score.</li>
<li>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)

```
        <li>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)

```
        <li>After you are done with the truncated svd, you can apply K-Means clustering and ch
```

the best number of clusters based on elbow method.

```
        <li> Print out wordclouds for each cluster, similar to that in previous assignment. </
        <li>You need to write a function that takes a word and returns the most similar words
```

cosine similarity between the vectors(vector: a row in the matrix after truncatedSVD)

```
    </ul>
</li>
<br>
```

6.1 Truncated-SVD

6.1.1 [5.1] Taking top features from TFIDF, SET 2

```
In [26]: indices = np.argsort(tf_idf_vect.idf_[::-1])
```

```
In [27]: features = tf_idf_vect.get_feature_names()
```

```
In [28]: top_feat = [features[i] for i in indices[:2500]]
```

6.1.2 [5.2] Calulation of Co-occurrence matrix

In [29]:
```
matrix = np.zeros((2500,2500))
```

In [30]:
```
for i in tqdm(X):
    each_sentence = i.split()
    for index,words in enumerate(each_sentence):
        if words in top_feat:
            for x in range(max(index-5,0),min(index+5,len(each_sentence)-1)+1):
                if each_sentence[x] in top_feat:
                    matrix[top_feat.index(words),top_feat.index(each_sentence[x])] +=
                else:
                    continue
        else:
            continue
```

```
100%|| 100000/100000 [10:04<00:00, 165.56it/s]
```

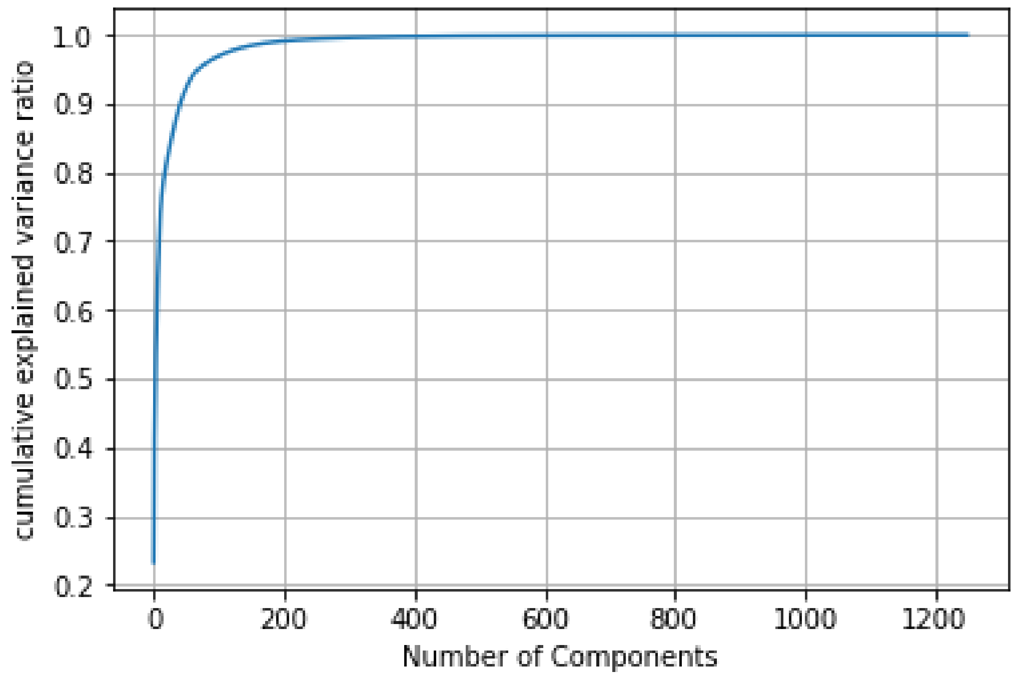
6.1.3 [5.3] Finding optimal value for number of components (n) to be retained.

In [540]:
```
from sklearn.decomposition import TruncatedSVD
model = TruncatedSVD(n_components=1250, n_iter=7, random_state=42)
svd = model.fit_transform(matrix)
```

In [541]:
```
evr = model.explained_variance_ratio_
```

In [542]:
```
cum_evr = np.cumsum(evr)
```

In [543]:
```
plt.plot(cum_evr)
plt.grid()
plt.xlabel('Number of Components')
plt.ylabel('cumulative explained variance ratio')
plt.show()
```



We observe that 100 components can explain 99% of the variance.

6.1.4 [5.4] Applying k-means clustering

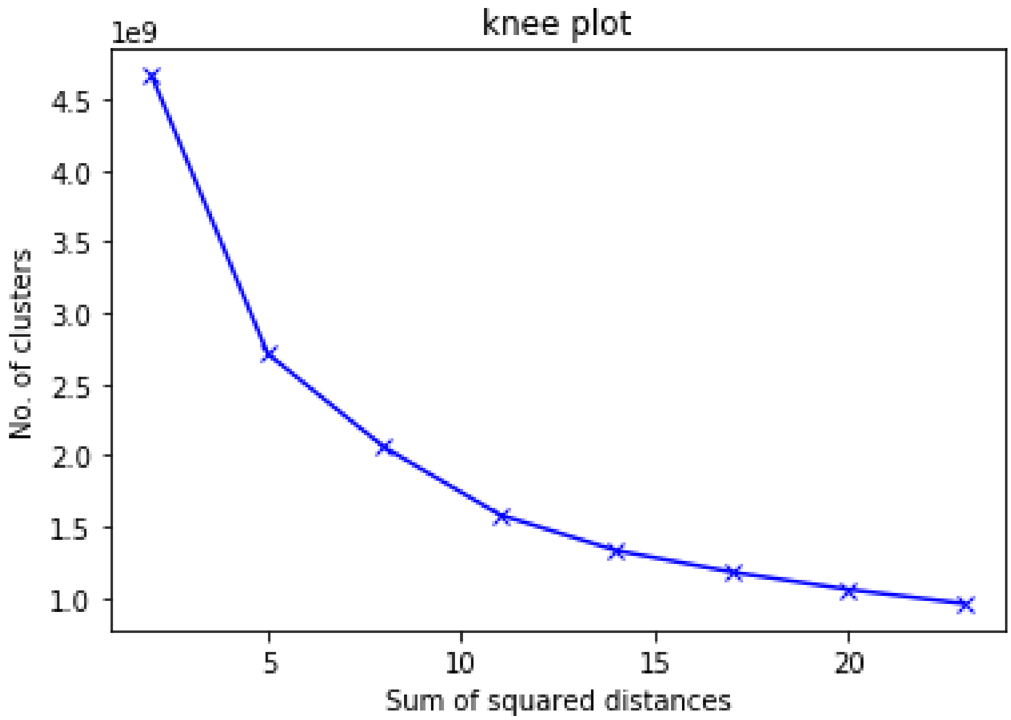
In [544]:
```
# Using 100 components as we got from S
model = TruncatedSVD(n_components=100, n_iter=7, random_state=42)
svd_final = model.fit_transform(matrix)
```

In [545]:
```
svd_final.shape
```

Out[545]:
```
(2500, 100)
```

In [546]:
```
#We define a function for hyperparameter tuning of KMeans
def kalgo(vector):
    ssd = []
    k = []
    for x in range(2,25,3):
        kmeans = KMeans(n_clusters=x, random_state=0, n_jobs=-1).fit(vector)
        ssd.append(kmeans.inertia_)
        k.append(x)

    plt.figure()
    plt.plot(k,ssd,'bx-')
    plt.title('knee plot')
    plt.xlabel("Sum of squared distances")
    plt.ylabel("No. of clusters")
    plt.show()
```

```
In [400]: from sklearn.cluster import KMeans
          kalgo(svd_final)
```



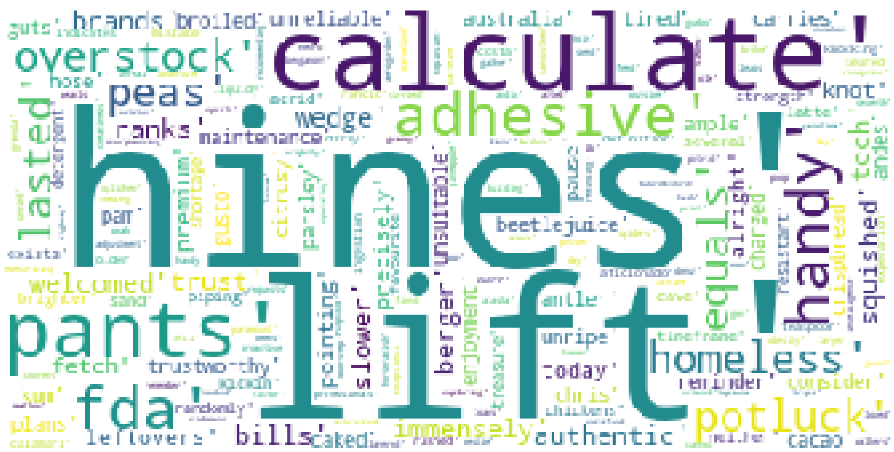
As we can see from knee plot, k=7 is optimal k.

```
In [401]: model = KMeans(n_clusters=7, random_state=0, n_jobs=-1).fit(svd_final)
          w = model.labels_

In [402]: w.shape

Out[402]: (2500,)
```

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

```
In [492]: #Create list of list in a varible name cluster of size max(w)+1. Append features to
          cluster = [[] for _ in range(max(w) + 1)]
          for i in range(0,2500):
              cluster[w[i]].append(top_feat[i])
          for i in range(0,max(w)+1):
              print("Wordcloud For cluster {}".format(i+1))
              wordcloud = WordCloud(background_color="white").generate(str(cluster[i]))
              plt.imshow(wordcloud)
              plt.axis("off")
              plt.show()
```

Wordcloud For cluster 1



Wordcloud For cluster 2

Wordcloud For cluster 3

Wordcloud For cluster 4

Wordcloud For cluster 5


```
Wordcloud For cluster 6
```


```
Wordcloud For cluster 7
```

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

```
In [549]: def cosine_sim(word):
              cmatrix = cosine_similarity(svd_final)
              similar_vect = cmatrix[top_feat.index(word)]
              print("Top 20 Words in order similar to '{}' are:".format(word))
              indices = similar_vect.argsort()[::-1][:20]
              #indices = np.argsort(similar_vect[::-1])[:20]
              for i in range(len(indices)):
                  print(top_feat[indices[i]])
```

```
In [550]: #using word from above wordcloud
          cosine_sim('love')
```

```
Top 20 Words in order similar to 'love' are:
love
em
chihuahuas
grandkids
pugs
bueno
doggies
pina
kiddos
snobs
delish
nom
wedge
hazlenut
flatbread
```

```
sincere
boylan
yorkshire
slathered
uh
```

```
In [539]: #using from top_feat
          cosine_sim(top_feat[1762])
```

```
Top 20 Words in order similar to 'product' are:
product
manner
page
marketed
disclaimer
meaningless
quoted
caliber
prompt
unreturnable
sketchy
specific
notification
cholestrol
marketing
consumers
claims
mechanical
development
advertise
```

7 [6] Conclusions

Optimal value for number of components : 100 Optimal value for number of clusters : 7
Conclusions:

```
1) Observed that SVD can manage with sparse matrix unlike PCA.
2) Using only the top components,we can cut down the cost of computation.
3) I have observed the cosine similarity with both the occurence matrix and svd_final matrix. I
It means SVD performs very well in getting the top components.
```