Document Clustering and Topic Modeling

In this project, we use unsupervised learning models to cluster unlabeled documents into different groups, visualize the results and identify their latent topics/structures. NLP data preprocessing: First step is to change text to numbers.

Contents

Part 0: Setup Google Drive Environment

```
!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

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)


file = drive.CreateFile({'id':'13gapHq7fada1q72UTGRqX36w8Mi60y3r'}) # id of file you w
file.GetContentFile('data.tsv')  # tab-separated
```

Part 1: Load Data

```
import numpy as np
import pandas as pd
import nltk #nlp常用package
# import gensim

from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt

nltk.download('punkt') #nlp常用数据：英文标点符号
nltk.download('stopwords') #没太多意义的词：noise
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
# Load data into dataframe
df = pd.read_csv('data.tsv', sep='\t', error_bad_lines=False) #error_bad_lines智能读取数
```

```
b'Skipping line 8704: expected 15 fields, saw 22\nSkipping line 16933: expected
b'Skipping line 85637: expected 15 fields, saw 22\n'
b'Skipping line 132136: expected 15 fields, saw 22\nSkipping line 158070: expecte
b'Skipping line 197000: expected 15 fields, saw 22\nSkipping line 197011: expecte
b'Skipping line 272057: expected 15 fields, saw 22\nSkipping line 293214: expecte
b'Skipping line 336028: expected 15 fields, saw 22\nSkipping line 344885: expecte
b'Skipping line 408773: expected 15 fields, saw 22\nSkipping line 434535: expecte
b'Skipping line 581593: expected 15 fields, saw 22\n'
b'Skipping line 652409: expected 15 fields, saw 22\n'
```

```
len(df.index)
```

```
960204
```

数据体量：96万

```
df.head()
```

	marketplace	customer_id	review_id	product_id	product_parent
0	US	3653882	R3O9SGZBVQBV76	B00FALQ1ZC	937001370
1	US	14661224	RKH8BNC3L5DLF	B00D3RGO20	484010722
2	US	27324930	R2HLE8WKZSU3NL	B00DKYC7TK	361166390

```
# Remove missing value
df.dropna(subset=['review_body'],inplace=True) #数据大，直接删掉缺失值


df.reset_index(inplace=True, drop=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 960056 entries, 0 to 960055
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   marketplace        960056 non-null  object
 1   customer_id        960056 non-null  int64 
 2   review_id          960056 non-null  object
 3   product_id         960056 non-null  object
 4   product_parent     960056 non-null  int64 
 5   product_title      960054 non-null  object
 6   product_category   960056 non-null  object
 7   star_rating        960056 non-null  int64 
 8   helpful_votes      960056 non-null  int64 
 9   total_votes        960056 non-null  int64 
 10  vine               960056 non-null  object
 11  verified_purchase  960056 non-null  object
 12  review_headline    960049 non-null  object
 13  review_body        960056 non-null  object
 14  review_date        960052 non-null  object
dtypes: int64(5), object(10)
memory usage: 109.9+ MB
```

```
# use the first 1000 data as our training data
```

```
data = df.loc[:999, 'review_body'].tolist()

data
```

```
['Absolutely love this watch! Get compliments almost every time I wear it. Dair
 'I love this watch it keeps time wonderfully.',
 'Scratches',
 'It works well on me. However, I found cheaper prices in other places after ma
 "Beautiful watch face.  The band looks nice all around.  The links do make tha
 'i love this watch for my purpose, about the people complaining should of done
 'for my wife and she loved it, looks great and a great price!',
 'I was about to buy this thinking it was a Swiss Army Infantry watch-- the des
 "Watch is perfect. Rugged with the metal &#34;Bull Bars&#34;. The red accents
 'Great quality and build.<br />The motors are really silent.<br />After fiddli
 "The watch was pretty much as it was described and how it looks. I really like
 'I bought this watch on 2013, the screen had a problem 10 months later. I sent
 "It is a cheap watch that looks cheap.  There isn't much else to say.",
 'Heavier than i though',
 'Had it for several weeks now and I love it - reliable, functional, wears easy
 'This one is different from the rest of my Invictas. I like the big watches bu
 "The watch is attractive and easy to read, except for the date. The little dia
 'said my wife..',
 'Nice watch, on time delivery from seller.',
 'Looks great and love to wear this watch.  Only negative thing is due to its b
 "I really like this watch. It has a great face that contrasts nicely with the
 'Works great but the watch a used it on was slim so I had to use a quarter to
 'Just what I needed for my Timex watch. Fits perfect on my wrist',
 "Absolutely LOVE this watch! Amazon prime saved me over $400 after tax from th
 'Its too large. When you order check the size of the case.',
 "This was a watch I bought during a quick sale thing, so it was cheaper than r
 'It works well with nice simple look.',
 'Thought that I read somewhere that you could swim with it. It does not howeve
 'Perfect Condition, Arrived on Time,Works & Looks Great',
 "This watch is a very beautiful time piece. This also could be an alternative
 "This is a beautiful watch. I love the look and feel of it, and I get complime
 'very good',
 'I was very apprehensive about buying a 28 dollar watch online, but it looked
 '[[VIDEOID:cc599be25462207f860e42621171d7e1]]This watch has an understated cle
 'Awesome watch for the price',
 'Love this watch, I just received it yesterday it looks really nice on my  wri
 'Nice face, easy to read. The band is a little too small.',
 'crap',
 'This is now my everyday watch.  Easy to read with lots of features.  Light  a
 'amazing product keeps everything safe and secure organized great quality for
 "Watch looks amazing and the features are second to none, but....the watch is
 'Metal color was changed after the first use , which indicates a very poor qua
 'So far so good. I have only had this watch a few weeks & so far it is still w
 'Grand Kids loved this',
 'Have worn it constantly, love the light at night',
 'Wife loves it and nice looking watch,',
 'Great product . Love the continous innovation in this field and given the pri
 'A very beautul and well-made pilots watch. Highly detailed dial,as well. The
 "Comfortable, looks great, very lightweight.The band is a little on the short
 'excelent product',
 'This press is very well constructed and sturdy. The different dies are easily
 "Love the look but the leather band is so stiff it doesn't conform well to my
```

```
 'Beyond my expectation..excellent product..good quality, well built, nicely do
 'It was bad it didnt fit my 22mm retro casio',
 'Watch kicks ass. Upon putting on this watch you will grow a full beard and be
 "It's a good value, and a good functional watch strap.  It's super wide though
 'Loved the watch; it just did not work supposedly self-winding; tried all poss
```

Part 2: Tokenizing and Stemming

Tokenize:分词-每个单词都是一个feature，所有的feature组成一个dictionary。一个data set有一个dict，每个data point有自己的matrix。

Stemming：保留词根，比如说去除时态和负数。（一般不使用）

Load stopwords and stemmer function from NLTK library. Stop words are words like "a", "the", or "in" which don't convey significant meaning. Stemming is the process of breaking a word down into its root.

```
# Use nltk's English stopwords.
stopwords = nltk.corpus.stopwords.words('english') #stopwords.append("n't")
#添加无意义的词
stopwords.append("'s")
stopwords.append("'m")
stopwords.append("br") #html <br>
stopwords.append("watch") #data is already watch, so we don't need to take care of wat

print ("We use " + str(len(stopwords)) + " stop-words in total.")
print (stopwords[:10])
```

```
We use 183 stop-words in total.
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

Use our defined functions to analyze (i.e. tokenize, stem) our reviews.

https://www.nltk.org/_modules/nltk/stem/snowball.html

```
from nltk.stem.snowball import SnowballStemmer
# from nltk.stem import WordNetLemmatizer

stemmer = SnowballStemmer("english")

# tokenization and stemming
def tokenization_and_stemming(text):
    tokens = []
```

```
    # exclude stop words and tokenize the document, generate a list of string
    for word in nltk.word_tokenize(text):
        if word.lower() not in stopwords:
            tokens.append(word.lower())

    filtered_tokens = []

    # filter out any tokens not containing letters (e.g., numeric tokens, raw punctuat
    for token in tokens:
        if token.isalpha():
            filtered_tokens.append(token)

    # stemming
    stems = [stemmer.stem(t) for t in filtered_tokens]
    return stems


tokenization_and_stemming(data[0])
```

```
['absolut',
 'love',
 'get',
 'compliment',
 'almost',
 'everi',
 'time',
 'wear',
 'dainti']
```

```
data[0]
```

```
'Absolutely love this watch! Get compliments almost every time I wear it. Daint
y. '
```

Part 3: TF-IDF

TF: Term Frequency，每个词出现的频率。

IDF: Inverse Document Frequency，这个词在本document里面出现的个数/这个词出现过的所有document的数量。

TF-IDF：TF*IDF

wiki:https://zh.wikipedia.org/wiki/Tf-idf

```
from sklearn.feature_extraction.text import TfidfVectorizer
# define vectorizer parameters
# TfidfVectorizer will help us to create tf-idf matrix
# max_df : maximum document frequency for the given word
```

```
# min_df : minimum document frequency for the given word
# max_features: maximum number of words
# use_idf: if not true, we only calculate tf
# stop_words : built-in stop words
# tokenizer: how to tokenize the document
# ngram_range: (min_value, max_value), eg. (1, 3) means the result will include 1-gram
tfidf_model = TfidfVectorizer(max_df=0.99, #set an upper bound:get rid of words occur
                              max_features=1000,
                                 min_df=0.01, stop_words='english',
                                 use_idf=True, tokenizer=tokenization_and_stemming, ng

tfidf_matrix = tfidf_model.fit_transform(data) #fit the vectorizer to synopses

print ("In total, there are " + str(tfidf_matrix.shape[0]) + \
      " reviews and " + str(tfidf_matrix.shape[1]) + " terms.")
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/feature_extraction/text.py:401: Us
  % sorted(inconsistent)
In total, there are 1000 reviews and 239 terms.
```

```
tfidf_matrix
```

```
<1000x239 sparse matrix of type '<class 'numpy.float64'>'
        with 6874 stored elements in Compressed Sparse Row format>
```

1000 data points, 239 features.

```
tfidf_matrix.toarray() #todense()
```

```
array([[0.       , 0.5125863, 0.       , ..., 0.       , 0.       ,
        0.       ],
       [0.       , 0.       , 0.       , ..., 0.       , 0.       ,
        0.       ],
       [0.       , 0.       , 0.       , ..., 0.       , 0.       ,
        0.       ],
       ...,
       [0.       , 0.       , 0.       , ..., 0.       , 0.       ,
        0.       ],
       [0.       , 0.       , 0.       , ..., 0.       , 0.       ,
        0.       ],
       [0.       , 0.       , 0.       , ..., 0.       , 0.       ,
        0.       ]])
```

matrix to array

```
tfidf_matrix.todense()
```

```
matrix([[0.       , 0.5125863, 0.       , ..., 0.       , 0.       ,
         0.       ],
```

```
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]])
```

array to matrix

```
print(type(tfidf_matrix.toarray()))
```

```
<class 'numpy.ndarray'>
```

```
print(type(tfidf_matrix.todense()))
```

```
<class 'numpy.matrix'>
```

Save the terms identified by TF-IDF.

```
# words
tf_selected_words = tfidf_model.get_feature_names()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWa
  warnings.warn(msg, category=FutureWarning)
```

```
# print out words
tf_selected_words
```

```
['abl',
 'absolut',
 'accur',
 'actual',
 'adjust',
 'alarm',
 'alreadi',
 'alway',
 'amaz',
 'amazon',
 'anoth',
 'arm',
 'arriv',
 'automat',
 'awesom',
 'bad',
```

```
        'band',
        'batteri',
        'beauti',
        'best',
        'better',
        'big',
        'bit',
        'black',
        'blue',
        'bought',
        'box',
        'bracelet',
        'brand',
        'break',
        'bright',
        'broke',
        'button',
        'buy',
        'ca',
        'came',
        'case',
        'casio',
        'chang',
        'cheap',
        'clasp',
        'classi',
        'clock',
        'color',
        'come',
        'comfort',
        'compliment',
        'cool',
        'cost',
        'crown',
        'crystal',
        'dark',
        'date',
        'daughter',
        'day',
        'deal',
        'definit',
        'deliveri',
```

Part 4: K-means clustering

kmeans停止条件：k点到组内每个点距离已经不再改变；或者自己设置运行次数（以免运行时间过长）。

找到best k的两种方法：Elbow method，Silhouette method。https://vitalflux.com/elbow-method-silhouette-score-which-better/

```
# k-means clustering
from sklearn.cluster import KMeans

num_clusters = 5

# number of clusters
km = KMeans(n_clusters=num_clusters, random_state=888)
km.fit(tfidf_matrix)

clusters = km.labels_.tolist()
```

Elbow method

```
from yellowbrick.cluster import KElbowVisualizer

visualizer = KElbowVisualizer(km, k=(2,10))
visualizer.fit(tfidf_matrix)        # Fit the data to the visualizer
visualizer.show()        # Finalize and render the figure
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
```

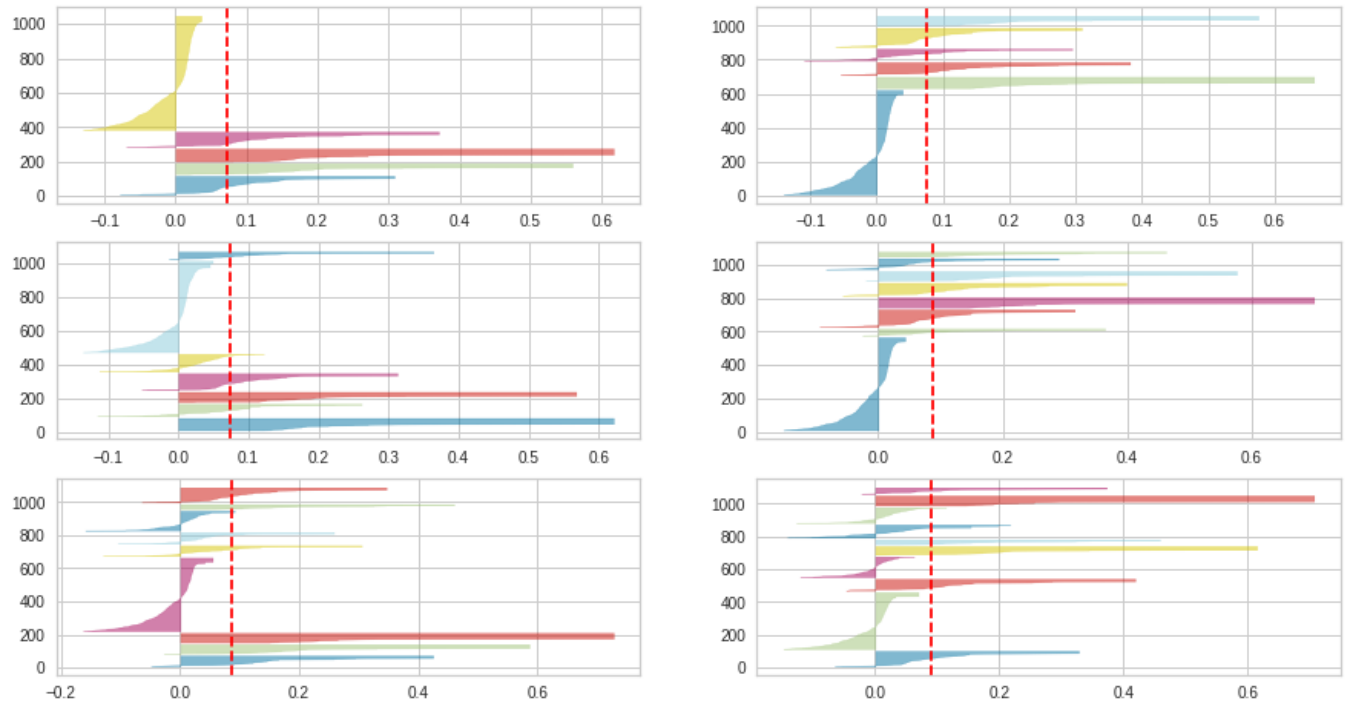
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:598: FutureWa
```

Elbow / SSE Plot: n_clusters = 8 represents the elbow you start seeing diminishing returns by increasing k

Silhouette method


```
import matplotlib.pyplot as plt
from yellowbrick.cluster import SilhouetteVisualizer

fig, ax = plt.subplots(3, 2, figsize=(15,8))
for i in [5, 6, 7, 8, 9, 10]:
    #Create KMeans instance for different number of clusters
    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=100, random_state=
    q, mod = divmod(i-3, 2)
    '''
    Create SilhouetteVisualizer instance with KMeans instance
    Fit the visualizer
    '''
    visualizer = SilhouetteVisualizer(km, colors='yellowbrick', ax=ax[q-1][mod])
    visualizer.fit(tfidf_matrix)
```



"The major difference between elbow and silhouette scores is that elbow only calculates the euclidean distance whereas silhouette takes into account variables such as variance, skewness, high-low differences, etc. The calculation simplicity of elbow makes it more suited than silhouette score for datasets with smaller size or time complexity.

Whether all the clusters' Silhouette plot falls beyond the average Silhouette score. If the silhouette plot for one of the clusters fall below the average Silhouette score, one can reject those numbers of clusters.If there are wider fluctuations, the number of cluster is sub-optimal. "

4.1. Analyze K-means Result

```
# create DataFrame films from all of the input files.
product = { 'review': df[:1000].review_body, 'cluster': clusters}
frame = pd.DataFrame(product, columns = ['review', 'cluster'])


frame.head(10)
```

	review	cluster
0	Absolutely love this watch! Get compliments al...	0
1	I love this watch it keeps time wonderfully.	0
2	Scratches	4
3	It works well on me. However, I found cheaper ...	4
4	Beautiful watch face. The band looks nice all...	4
5	i love this watch for my purpose, about the pe	0

```
print ("Number of reviews included in each cluster:")
frame['cluster'].value_counts().to_frame()
```

```
Number of reviews included in each cluster:
```

	cluster
4	664
0	109
3	88
2	76
1	63

```
km.cluster_centers_

# 239数的list -> cluster 0的中心点的tf-idf值
#-> assumption: 中心点的值可以代表这个cluster
#-> tf-idf值越大，对应的词越能代表这个document
#-> 选出了tf-idf最大的6个值对应的词来代表这个cluster：weight，最重要的词
```

```
array([[0.        , 0.03348306, 0.        , ..., 0.00942615, 0.01489936,
        0.        ],
       [0.00434466, 0.0059089 , 0.        , ..., 0.00962176, 0.01257854,
        0.00882311],
       [0.00454066, 0.        , 0.        , ..., 0.        , 0.        ,
        0.01669195],
       ...,
       [0.00726978, 0.        , 0.00899976, ..., 0.00221798, 0.04572863,
        0.01935265],
       [0.        , 0.        , 0.        , ..., 0.        , 0.01057221,
        0.        ],
       [0.        , 0.03717066, 0.        , ..., 0.        , 0.0318602 ,
        0.        ]])
```

```
km.cluster_centers_.shape
```

```
(10, 239)
```

```
print ("<Document clustering result by K-means>")

#km.cluster_centers_ denotes the importances of each items in centroid.
#We need to sort it in decreasing-order and get the top k items.
order_centroids = km.cluster_centers_.argsort()[:, ::-1]

Cluster_keywords_summary = {}
for i in range(num_clusters):
    print ("Cluster " + str(i) + " words:", end='')
    Cluster_keywords_summary[i] = []
    for ind in order_centroids[i, :6]: #replace 6 with n words per cluster
        Cluster_keywords_summary[i].append(tf_selected_words[ind])
        print (tf_selected_words[ind] + ",", end='')
    print ()

    cluster_reviews = frame[frame.cluster==i].review.tolist()
    print ("Cluster " + str(i) + " reviews (" + str(len(cluster_reviews)) + " reviews)
    print (", ".join(cluster_reviews))
    print ()
```

```
<Document clustering result by K-means>
Cluster 0 words:love,wife,look,beauti,husband,bought,
Cluster 0 reviews (109 reviews):
Absolutely love this watch! Get compliments almost every time I wear it. Dainty.

Cluster 1 words:look,beauti,realli,expect,awesom,want,
Cluster 1 reviews (63 reviews):
Nice watch, on time delivery from seller., It works well with nice simple look.,

Cluster 2 words:great,look,price,product,comfort,love,
Cluster 2 reviews (76 reviews):
very good, It's a good value, and a good functional watch strap.  It's super wid

Cluster 3 words:time,batteri,wear,watch,read,need,
Cluster 3 reviews (88 reviews):
for my wife and she loved it, looks great and a great price!, Watch is perfect. I

Cluster 4 words:nice,price,realli,good,look,love,
Cluster 4 reviews (664 reviews):
Scratches, It works well on me. However, I found cheaper prices in other places a
```

Part 5: Topic Modeling - Latent Dirichlet Allocation

相对于Kmeans，LDA每个分类数量差不多，Kmeans有些cluster可能特别大。

```
# Use LDA for clustering
```

```
from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components=5)


# document topic matrix for tfidf_matrix_lda
lda_output = lda.fit_transform(tfidf_matrix)
print(lda_output.shape)
print(lda_output)
```

```
(1000, 5)
[[0.75820546 0.06018249 0.06034838 0.06168863 0.05957505]
 [0.08646718 0.08531654 0.65810642 0.08702322 0.08308665]
 [0.2        0.2        0.2        0.2        0.2       ]
 ...
 [0.10000024 0.10000032 0.10018301 0.59980288 0.10001354]
 [0.0830288  0.36585255 0.08381583 0.08287598 0.38442684]
 [0.71503251 0.06900385 0.07932771 0.06727334 0.06936258]]
```

```
# topics and words matrix
topic_word = lda.components_
print(topic_word.shape)
print(topic_word) #correlation
```

```
(5, 239)
[[0.20832368 7.6899306  0.20779393 ... 0.21097836 5.72418958 3.67494357]
 [2.00416652 0.20154194 1.62931338 ... 4.13355229 0.20273598 0.20383886]
 [2.47970967 0.20117534 1.30559852 ... 0.28134704 9.71174658 0.76554501]
 [0.20045719 0.20136354 0.2021861  ... 1.38073671 0.20572316 3.24148361]
 [0.20086498 0.20014084 0.2087603  ... 0.92807624 0.21214166 4.71197538]]
```

```
# column names
topic_names = ["Topic" + str(i) for i in range(lda.n_components)]

# index names
doc_names = ["Doc" + str(i) for i in range(len(data))]

df_document_topic = pd.DataFrame(np.round(lda_output, 2), columns=topic_names, index=c

# get dominant topic for each document
topic = np.argmax(df_document_topic.values, axis=1)
df_document_topic['topic'] = topic

df_document_topic.head(10)
```

	Topic0	Topic1	Topic2	Topic3	Topic4	topic
Doc0	0.76	0.06	0.06	0.06	0.06	0
Doc1	0.09	0.09	0.66	0.09	0.08	2
Doc2	0.20	0.20	0.20	0.20	0.20	0
Doc3	0.75	0.06	0.06	0.07	0.06	0
Doc4	0.48	0.04	0.40	0.04	0.04	0
Doc5	0.08	0.08	0.69	0.08	0.08	2

```
df_document_topic['topic'].value_counts().to_frame()
```

	topic
2	327
0	204
3	192
1	144
4	133

```
# topic word matrix
print(lda.components_)
# topic-word matrix
df_topic_words = pd.DataFrame(lda.components_)

# column and index
df_topic_words.columns = tfidf_model.get_feature_names()
df_topic_words.index = topic_names

df_topic_words.head()
```

```
[[0.20832368 7.6899306  0.20779393 ... 0.21097836 5.72418958 3.67494357]
 [2.00416652 0.20154194 1.62931338 ... 4.13355229 0.20273598 0.20383886]
 [2.47970967 0.20117534 1.30559852 ... 0.28134704 9.71174658 0.76554501]
 [0.20045719 0.20136354 0.2021861  ... 1.38073671 0.20572316 3.24148361]
 [0.20086498 0.20014084 0.2087603  ... 0.92807624 0.21214166 4.71197538]]
```

```
# print top n keywords for each topic
def print_topic_words(tfidf_model, lda_model, n_words):
    words = np.array(tfidf_model.get_feature_names())
    topic_words = []
    # for each topic, we have words weight
    for topic_words_weights in lda_model.components_:
        top_words = topic_words_weights.argsort()[::-1][:n_words]
        topic_words.append(words.take(top_words))
    return topic_words

topic_keywords = print_topic_words(tfidf_model=tfidf_model, lda_model=lda, n_words=15)

df_topic_words = pd.DataFrame(topic_keywords)
df_topic_words.columns = ['Word '+str(i) for i in range(df_topic_words.shape[1])]
df_topic_words.index = ['Topic '+str(i) for i in range(df_topic_words.shape[0])]
df_topic_words
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWa
  warnings.warn(msg, category=FutureWarning)
```

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9	Word
Topic 0	excel	love	wife	time	band	face	absolut	heavi	want	hand	disapp
Topic 1	perfect	beauti	exact	band	fine	simpl	work	wear	happi	tri	func
Topic 2	nice	love	look	realli	color	pretti	great	awesom	pictur	big	
Topic											

Advanced:Hierachical clustering.https://www.analyticsvidhya.com/blog/2019/05/beginners-guide-hierarchical-clustering/

Part 6: Hierarchical Clustering

```
from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(linkage='ward',n_clusters=4).fit(tfidf_matrix.toarray())


hie_clusters = hc.labels_.tolist()
```

```
print(hie_clusters)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

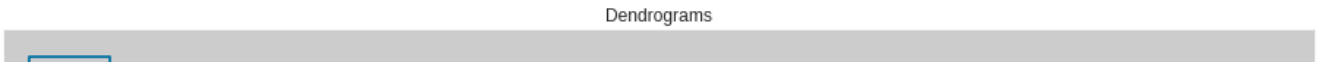
```
product = { 'review': df[:1000].review_body, 'cluster': hie_clusters}
frame = pd.DataFrame(product, columns = ['review', 'cluster'])
frame['cluster'].value_counts().to_frame()
```

	cluster
0	916
3	38
2	24
1	22

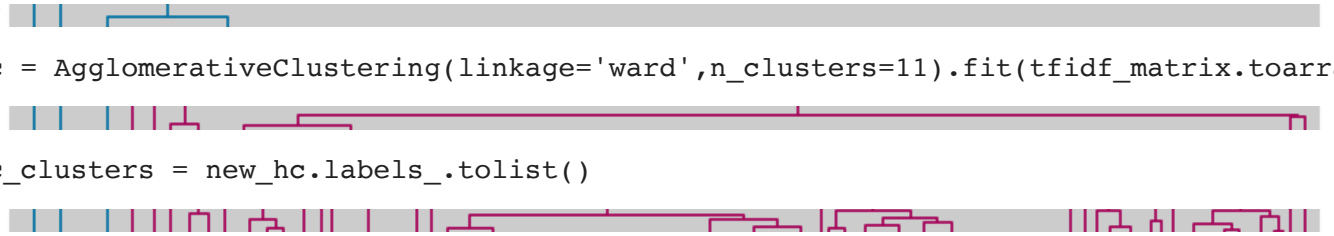
most data points are clustered into one group(about 8-900), from cluster 2-8

```
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(16, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(tfidf_matrix.toarray(), method='ward'))
plt.axhline(y=4, color='r', linestyle='--')
```

```
<matplotlib.lines.Line2D at 0x7fd69e6dd4d0>
```



From the dendrogram, we can clearly see that most data have distance less than 5. Threshold 4 gives 11 clusters. When distance is greater than 7, data could be separate into 2 groups.



```
new_hc = AgglomerativeClustering(linkage='ward',n_clusters=11).fit(tfidf_matrix.toarra
```

```
new_hc_clusters = new_hc.labels_.tolist()
```

```
product = { 'review': df[:1000].review_body, 'cluster': new_hc_clusters}
frame = pd.DataFrame(product, columns = ['review', 'cluster'])
frame['cluster'].value_counts().to_frame()
```

	cluster
0	756
1	63
7	38
4	28
5	24
10	22
3	22
2	16
6	12
8	10
9	9

Extension(Other clustering methods):

DBSCAN:密度聚类适用于聚类形状不规则的情况

Spectral Clustering:https://towardsdatascience.com/spectral-clustering-aba2640c0d5b

Mean Shift: https://www.geeksforgeeks.org/ml-mean-shift-clustering/