Document Clustering and Topic Modeling

In this project, I use unsupervised learning models to cluster unlabeled documents into different groups, visualize the results and identify their latent topics/structures.

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':'192JMR7SIqoa14vrs7Z9BXO3iK89pimJL'}) # replace the id w
file.GetContentFile('data.tsv')
```

Part 1: Load Data

```
import numpy as np
import pandas as pd
import nltk
import gensim
```

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

nltk.download('punkt')
nltk.download('stopwords')
```

```
[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)
```

```
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'
```

```
df.head()
```

marketplace	customer_id	review_id	product_id	product_parent	produc

Invicta

```
df.columns
```

```
Index(['marketplace', 'customer_id', 'review_id', 'product_id',
       'product_parent', 'product_title', 'product_category', 'star_rating',
       'helpful_votes', 'total_votes', 'vine', 'verified_purchase',
       'review_headline', 'review_body', 'review_date'],
      dtype='object')
```

Au

```
# Remove missing value
df.dropna(subset=['review_body'],inplace=True)
```

Stee

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

Citiz

```
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()
```

```
df.loc[:999, 'review_body']
```

```
0      Absolutely love this watch! Get compliments al...
1           I love this watch it keeps time wonderfully.
2                                              Scratches
3      It works well on me. However, I found cheaper ...
```

```
4      Beautiful watch face.  The band looks nice all...
                                   ...
995    I'm late getting to the party, but after disco...
996                              Wear it all the time!
997                                         very good.
998    Watch is exactly as it is shown in the picture...
999    Really large on the arm but that's what I want...
Name: review_body, Length: 1000, dtype: object
```

```
type(data)
```

```
list
```

```
data[:10]
```

```
['Absolutely love this watch! Get compliments almost every time I wear it. Dainty
 'I love this watch it keeps time wonderfully.',
 'Scratches',
 'It works well on me. However, I found cheaper prices in other places after mak
 "Beautiful watch face.  The band looks nice all around.  The links do make that
 '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 desc
 "Watch is perfect. Rugged with the metal &#34;Bull Bars&#34;. The red accents a
 'Great quality and build.<br />The motors are really silent.<br />After fiddlin
```

Part 2: Tokenizing and Stemming

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("'s")
stopwords.append("'m")
stopwords.append("n't")
stopwords.append("br")

print ("We use " + str(len(stopwords)) + " stop-words from nltk library.")
print (stopwords[:10])
```

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

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

```
from nltk.stem.snowball import SnowballStemmer

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',
 'watch',
 'get',
 'compliment',
 'almost',
 'everi',
 'time',
 'wear',
 'dainti']
```

```
data[0]
```

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

Part 3: TF-IDF

TF: Term Frequency

IDF: Inverse Document Frequency

example: (1,2) dictionary :[Arthur, da, Jason, huang, arthur da, da jason, jason da, da da, da huang]

document1: "Arthur da Jason"

document 2: "Jason da da huang"

document1: tf-idf [1, 0.5, 0.5, 0]; document2: tf-idf [0, 1, 0.5, 1]

2-gram:

document 1: Arthur da, da Jason; document 2: Jason da, da da, da huang bigram

3-gram:

document 1: Athur da Jason; document 2: Jason da da, da da huang

[Arhur, da, Jason...]

```
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, 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/python3.6/dist-packages/sklearn/feature_extraction/text.py:385: Us
  'stop_words.' % sorted(inconsistent))
In total, there are 1000 reviews and 239 terms.
```

```
type(tfidf_matrix)
```

```
scipy.sparse.csr.csr_matrix
```

```
# check the parameters
tfidf_model.get_params()
```

```
{'analyzer': 'word',
 'binary': False,
 'decode_error': 'strict',
 'dtype': numpy.float64,
 'encoding': 'utf-8',
 'input': 'content',
 'lowercase': True,
```

```
 'max_df': 0.99,
 'max_features': 1000,
 'min_df': 0.01,
 'ngram_range': (1, 1),
 'norm': 'l2',
 'preprocessor': None,
 'smooth_idf': True,
 'stop_words': 'english',
 'strip_accents': None,
 'sublinear_tf': False,
 'token_pattern': '(?u)\\b\\w\\w+\\b',
 'tokenizer': <function __main__.tokenization_and_stemming>,
 'use_idf': True,
 'vocabulary': None}
```

Save the terms identified by TF-IDF.

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

```
# print out words
tf_selected_words
```

```
 'run',
 'said',
 'say',
 'screw',
 'second',
 'seiko',
 'seller',
 'send',
 'sent',
 'set',
 'sever',
 'ship',
 'short',
 'simpl',
 'sinc',
 'size',
 'small',
 'smaller',
 'solid',
 'someth',
 'somewhat',
 'son',
 'star',
 'start',
 'stop',
 'strap',
 'sturdi',
 'style',
 'stylish',
 'super',
 'sure',
```

```
 'surpris',
 'swim',
 'tell',
 'thank',
 'thing',
 'think',
 'thought',
 'time',
 'timex',
 'tini',
 'tri',
 'turn',
 'use',
 'valu',
 'want',
 'watch',
 'water',
 'way',
 'wear',
 'week',
 'weight',
 'went',
 'wife',
 'wind',
 'wish',
 'work',
 'worn',
 'worth',
```

Part 4: K-means clustering

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

num_clusters = 5

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

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

```
km.labels_
```

```
array([0, 0, 0, 3, 3, 0, 2, 3, 2, 2, 3, 3, 3, 0, 0, 3, 3, 0, 1, 3, 3, 2,
       0, 0, 0, 3, 1, 0, 2, 3, 3, 4, 3, 3, 0, 0, 3, 0, 0, 0, 3, 3, 3, 0,
       0, 1, 0, 3, 3, 0, 3, 0, 1, 0, 3, 4, 3, 3, 0, 3, 3, 0, 3, 2, 0, 3,
       3, 3, 0, 0, 0, 2, 0, 3, 3, 4, 3, 4, 0, 0, 0, 0, 3, 3, 3, 0, 2, 0,
       0, 0, 3, 3, 2, 4, 4, 4, 2, 2, 0, 0, 3, 0, 0, 3, 0, 0, 0, 0, 4, 2,
       3, 0, 0, 4, 3, 3, 0, 3, 3, 4, 2, 3, 3, 0, 0, 3, 0, 3, 3, 2, 0, 3,
       0, 1, 0, 0, 3, 0, 3, 3, 3, 3, 3, 1, 2, 3, 3, 0, 3, 4, 3, 3, 0, 2,
       4, 3, 3, 0, 0, 3, 3, 2, 3, 4, 0, 0, 1, 1, 4, 4, 0, 4, 0, 0, 0, 0,
```

```
       3, 0, 3, 1, 0, 1, 3, 1, 0, 4, 3, 2, 0, 0, 2, 3, 2, 3, 4, 3, 3, 0,
       3, 2, 3, 0, 4, 0, 4, 1, 3, 0, 0, 3, 0, 1, 4, 1, 0, 3, 0, 3, 0, 0,
       3, 0, 3, 2, 3, 1, 3, 1, 3, 4, 3, 2, 3, 0, 1, 4, 0, 2, 3, 1, 3, 0,
       2, 0, 3, 0, 2, 1, 0, 3, 2, 1, 3, 3, 3, 3, 0, 3, 0, 0, 2, 3, 0, 0,
       1, 3, 3, 0, 2, 3, 3, 3, 3, 4, 0, 3, 3, 3, 0, 3, 0, 0, 3, 0, 3, 3,
       0, 3, 2, 3, 4, 2, 0, 3, 3, 1, 3, 3, 0, 2, 3, 3, 0, 2, 2, 0, 3, 3,
       4, 3, 3, 0, 3, 2, 3, 0, 0, 0, 3, 1, 0, 2, 3, 3, 0, 3, 3, 3, 3, 0,
       0, 4, 3, 2, 3, 0, 0, 3, 0, 3, 0, 0, 3, 2, 3, 3, 0, 3, 4, 3, 0, 3,
       0, 1, 3, 2, 3, 1, 3, 0, 1, 0, 3, 0, 3, 0, 0, 3, 1, 0, 3, 1, 0, 0,
       2, 0, 0, 0, 3, 0, 0, 3, 2, 3, 3, 3, 2, 3, 0, 0, 3, 3, 2, 3, 0, 0,
       0, 3, 3, 0, 3, 0, 0, 1, 3, 3, 1, 3, 0, 0, 3, 1, 3, 0, 3, 0, 0, 3,
       0, 3, 0, 4, 2, 3, 3, 3, 0, 3, 0, 3, 4, 3, 0, 0, 0, 0, 0, 0, 3, 0,
       0, 0, 0, 3, 0, 3, 4, 3, 3, 3, 4, 3, 1, 2, 0, 3, 3, 1, 0, 0, 3, 3,
       0, 3, 3, 3, 0, 3, 3, 0, 2, 0, 0, 0, 2, 3, 3, 3, 3, 1, 0, 0, 1, 3,
       0, 2, 0, 3, 0, 3, 0, 0, 0, 3, 2, 3, 2, 0, 3, 0, 4, 0, 1, 0, 3, 3,
       1, 4, 3, 3, 3, 0, 0, 3, 0, 0, 3, 2, 3, 0, 2, 2, 4, 0, 2, 0, 4, 3,
       2, 2, 0, 0, 3, 0, 3, 1, 0, 4, 0, 0, 3, 0, 2, 4, 0, 0, 3, 3, 0, 0,
       4, 0, 2, 2, 3, 0, 4, 0, 0, 3, 0, 3, 3, 3, 0, 0, 0, 3, 0, 3, 0, 1,
       3, 3, 3, 4, 3, 4, 1, 0, 3, 3, 0, 3, 3, 0, 3, 3, 0, 0, 3, 3, 3, 1,
       0, 0, 3, 0, 3, 2, 1, 2, 2, 0, 3, 3, 2, 3, 0, 3, 3, 3, 3, 3, 0, 0,
       0, 3, 4, 0, 0, 2, 0, 3, 3, 3, 1, 0, 0, 3, 0, 0, 3, 0, 2, 3, 0, 4,
       2, 4, 1, 3, 3, 0, 0, 1, 3, 0, 3, 0, 4, 3, 3, 3, 0, 3, 2, 4, 1, 3,
       0, 2, 2, 3, 1, 2, 1, 3, 3, 4, 3, 3, 3, 3, 0, 2, 3, 3, 3, 3, 3, 0,
       3, 3, 3, 1, 0, 4, 0, 4, 0, 3, 0, 4, 4, 3, 0, 3, 2, 3, 0, 0, 1, 0,
       0, 4, 3, 0, 0, 3, 2, 1, 3, 0, 0, 3, 1, 3, 3, 3, 0, 0, 3, 0, 0, 0,
       0, 3, 0, 0, 0, 3, 3, 1, 0, 3, 3, 3, 3, 0, 4, 3, 2, 3, 0, 3, 4, 3,
       3, 0, 3, 0, 0, 3, 3, 3, 4, 2, 2, 0, 2, 1, 0, 1, 3, 3, 3, 0, 0, 0,
       3, 4, 2, 3, 3, 0, 3, 3, 4, 2, 3, 2, 4, 0, 1, 3, 3, 0, 0, 2, 0, 0,
       0, 1, 0, 3, 2, 3, 3, 3, 3, 3, 0, 1, 0, 3, 3, 2, 0, 0, 3, 0, 3, 0,
       3, 2, 0, 3, 2, 4, 3, 1, 3, 3, 0, 3, 3, 0, 3, 3, 0, 0, 2, 2, 0, 3,
       3, 0, 0, 0, 3, 3, 3, 3, 3, 3, 4, 0, 3, 3, 3, 1, 0, 0, 3, 3, 0, 3,
       0, 3, 2, 3, 0, 0, 3, 3, 3, 3, 3, 3, 1, 0, 4, 4, 3, 3, 3, 0, 1, 0,
       0, 0, 3, 3, 0, 2, 1, 0, 3, 3, 2, 0, 3, 1, 3, 0, 0, 0, 3, 3, 0, 0,
       0, 1, 3, 4, 1, 1, 4, 3, 3, 3, 0, 0, 0, 4, 3, 3, 2, 1, 3, 3, 3, 0,
       0, 0, 3, 3, 0, 2, 2, 0, 0, 0, 2, 3, 0, 3, 3, 3, 4, 0, 3, 2, 2, 2,
       3, 0, 0, 3, 3, 3, 3, 3, 0, 0, 0, 0, 4, 1, 0, 3, 3, 0, 0, 2, 4, 2,
       2, 3, 3, 1, 3, 0, 3, 0, 3, 4, 0, 3, 0, 0, 0, 0, 4, 1, 0, 2, 3, 1,
       0, 0, 1, 3, 3, 3, 3, 4, 3, 0], dtype=int32)
```

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	0
3	It works well on me. However, I found cheaper ...	3
4	Beautiful watch face. The band looks nice all...	3
5	i love this watch for my purpose, about the pe...	0
6	for my wife and she loved it, looks great and ...	2

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

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

	cluster
3	404
0	350
2	101
1	73
4	72

```
frame['cluster']
```

```
0      0
1      0
2      0
3      3
4      3
      ..
995    3
996    3
997    4
998    3
999    0
Name: cluster, Length: 1000, dtype: int64
```

```
type(frame['cluster'])
```

```
pandas.core.series.Series
```

```
frame['cluster'].value_counts()
```

```
3    404
```

```
0    350
2    101
1     73
4     72
Name: cluster, dtype: int64
```

```
frame['cluster'].value_counts().to_frame()
```

	cluster
3	404
0	350
2	101
1	73
4	72

```
km.cluster_centers_

#241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
```

```
array([[0.00153768, 0.01933179, 0.        , ..., 0.00665645, 0.01596364,
        0.00268526],
       [0.        , 0.        , 0.        , ..., 0.        , 0.00652   ,
        0.        ],
       [0.00317517, 0.        , 0.        , ..., 0.00218233, 0.00368264,
        0.02248446],
       [0.00779165, 0.00154328, 0.00620645, ..., 0.00825386, 0.01916797,
        0.02012109],
       [0.        , 0.        , 0.        , ..., 0.        , 0.009062  ,
        0.        ]])
```

```
km.cluster_centers_.shape
```

```
(5, 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,beauti,watch,perfect,excel,like,
Cluster 0 reviews (350 reviews):
Absolutely love this watch! Get compliments almost every time I wear it. Dainty.

Cluster 1 words:nice,watch,price,look,simpl,realli,
Cluster 1 reviews (73 reviews):
Nice watch, on time delivery from seller., It works well with nice simple look.,

Cluster 2 words:great,watch,look,price,work,product,
Cluster 2 reviews (101 reviews):
for my wife and she loved it, looks great and a great price!, Watch is perfect. I

Cluster 3 words:watch,look,work,band,time,like,
Cluster 3 reviews (404 reviews):
It works well on me. However, I found cheaper prices in other places after makin

Cluster 4 words:good,product,seller,qualiti,price,big,
Cluster 4 reviews (72 reviews):
very good, It's a good value, and a good functional watch strap.  It's super wid
```

Part 5: Topic Modeling - Latent Dirichlet Allocation

```
# Use LDA for clustering
from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components=5)


from sklearn.feature_extraction.text import CountVectorizer
# LDA requires integer values
tfidf_model_lda = CountVectorizer(max_df=0.99, max_features=1000,
                                 min_df=0.01, stop_words='english',
                                 tokenizer=tokenization_and_stemming, ngram_range=(1,1

tfidf_matrix_lda = tfidf_model_lda.fit_transform(data) #fit the vectorizer to synopses

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

```
/usr/local/lib/python3.6/dist-packages/sklearn/feature_extraction/text.py:385: Us
```

```
  'stop_words.' % sorted(inconsistent))
In total, there are 1000 reviews and 239 terms.
```

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

```
(1000, 5)
[[0.43076132 0.49293953 0.02538472 0.02557876 0.02533567]
 [0.05147778 0.79485271 0.05046552 0.05218773 0.05101625]
 [0.2        0.2        0.2        0.2        0.2       ]
 ...
 [0.10000371 0.10019786 0.10017996 0.59923703 0.10038144]
 [0.53161031 0.0506994  0.31615559 0.05036461 0.05117009]
 [0.04031559 0.04028078 0.04039556 0.04018036 0.83882771]]
```

```
tfidf_matrix_lda.todense()
```

```
matrix([[0, 1, 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]])
```

```
lda.fit_transform(tfidf_matrix)
```

```
array([[0.05708095, 0.05881444, 0.76786403, 0.05814958, 0.058091  ],
       [0.07482673, 0.0784016 , 0.68908051, 0.08019457, 0.07749659],
       [0.2       , 0.2       , 0.2       , 0.2       , 0.2       ],
       ...,
       [0.10046172, 0.59925424, 0.10000027, 0.10000012, 0.10028365],
       [0.40122769, 0.07736187, 0.0774291 , 0.0776961 , 0.36628525],
       [0.06723474, 0.72992587, 0.06741638, 0.06805077, 0.06737225]])
```

```
tfidf_matrix.todense()
```

```
matrix([[0.        , 0.50552558, 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.        ]])
```

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

```
(5, 239)
[[0.20007458 0.2027109  0.2002773  ... 0.20123204 1.10525864 1.15351911]
 [0.2005357  0.2000741  0.20075595 ... 0.20097766 3.75509708 0.20182843]
 [0.20087772 7.58454701 0.20203673 ... 4.54498507 0.20102121 4.47594762]
 [4.20322356 0.20219621 2.70360591 ... 0.20772332 9.85519276 6.29848521]
 [0.20199209 0.20008486 0.2007308  ... 1.72981711 0.91493206 0.20991275]]
```

```
# 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=d

# 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	0.43	0.49	0.03	0.03	0.03	1
Doc1	0.05	0.79	0.05	0.05	0.05	1
Doc2	0.20	0.20	0.20	0.20	0.20	0
Doc3	0.03	0.03	0.03	0.03	0.88	4
Doc4	0.59	0.08	0.01	0.10	0.22	0
Doc5	0.51	0.37	0.04	0.04	0.04	0
Doc6	0.03	0.34	0.03	0.57	0.03	3
Doc7	0.62	0.03	0.03	0.03	0.29	0
Doc8	0.62	0.18	0.16	0.01	0.01	0
Doc9	0.90	0.03	0.03	0.03	0.03	0

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

	topic
0	281
4	216
3	191

```
# 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_lda.get_feature_names()
df_topic_words.index = topic_names

df_topic_words.head()
```

```
[[0.20007458 0.2027109  0.2002773  ... 0.20123204 1.10525864 1.15351911]
 [0.2005357  0.2000741  0.20075595 ... 0.20097766 3.75509708 0.20182843]
 [0.20087772 7.58454701 0.20203673 ... 4.54498507 0.20102121 4.47594762]
 [4.20322356 0.20219621 2.70360591 ... 0.20772332 9.85519276 6.29848521]
 [0.20199209 0.20008486 0.2007308  ... 1.72981711 0.91493206 0.20991275]]
```

	abl	absolut	accur	actual	adjust	alarm	alreadi	alway	
Topic0	0.200075	0.202711	0.200277	0.716977	0.207607	0.200118	0.749592	1.432611	1.32
Topic1	0.200536	0.200074	0.200756	0.201030	2.118276	2.121281	0.201960	0.200508	0.20
Topic2	0.200878	7.584547	0.202037	0.200873	2.872874	1.385377	0.201425	1.762263	0.20
Topic3	4.203224	0.202196	2.703606	4.497093	0.228688	0.206945	0.206170	0.213460	8.70
Topic4	0.201992	0.200085	0.200731	0.201693	0.813094	1.982238	4.365398	1.228218	0.20

5 rows × 239 columns

```
# 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, lda_model=lda, n_words

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
```

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9	Word 10
Topic 0	awesom	qualiti	watch	look	fast	deliveri	broke	ship	love	week	bette
Topic 1	good	work	beauti	watch	big	realli	look	bad	time	far	style
Topic 2	love	great	watch	look	wife	price	beauti	batteri	absolut	bought	comfor

```
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
```

```
km
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

```
km.cluster_centers_
```

```
array([[0.00153768, 0.01933179, 0.        , ..., 0.00665645, 0.01596364,
        0.00268526],
       [0.        , 0.        , 0.        , ..., 0.        , 0.00652   ,
        0.        ],
       [0.00317517, 0.        , 0.        , ..., 0.00218233, 0.00368264,
        0.02248446],
       [0.00779165, 0.00154328, 0.00620645, ..., 0.00825386, 0.01916797,
        0.02012109],
       [0.        , 0.        , 0.        , ..., 0.        , 0.009062  ,
        0.        ]])
```