# 6. Cluster analysis & topic models

Sogang University Department of Business Administration
Professor Myung Suk Kim

# Cluster analysis topic model

- Classification

- Classify similar words or documents into clusters using the relative distance of

- Establish document - word ( or word - document ) matrix ( frequency matrix , TF-IDF, etc. )

- Similarity measurement ( Euclidean , Cosine , Jacquard , etc. )

- Classification by applying

- topic model

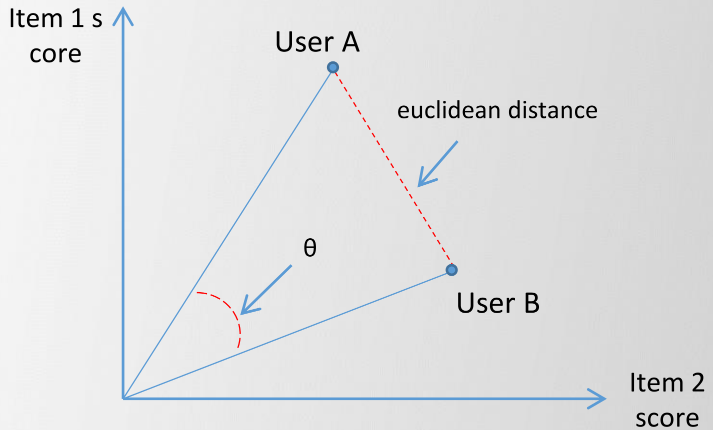
- Classify documents based on topics by using the probability of occurrence of latent topics in documents

# Similarity measure

- Euclidean similarity

  - Measure the relative distance difference between two document vectors

$$\sqrt{\sum_{f=1}^{p}(x_{if}-x_{jf})^2}$$




- cosine similarity

  - Measure

- The more similar two vectors are, the closer the cosine value is to 1, the less similar the closer to
- For example, even if the number of occurrences of words in the two documents differs greatly if the ratio is the same, the similarity is high.

$$x \cdot y = \|x\| \, \|y\| \cos\theta$$

$$\cos\theta = \frac{x \cdot y}{\|x\| \, \|y\|} = \frac{\sum_{i=1}^{n} x_i \times y_j}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

# Cosine similarity

**1. Cosine Similarity**

```
# https://wikidocs.net/24603
from numpy import dot
from numpy.linalg import norm
import numpy as np
def cos_sim (A, B):
return dot(A, B)/(norm(A)*norm(B))

#d1 = ' I like apples '
#d2 = ' I like mango '
#d3 = ' I like mango I like mango '
# Word vectors : [ me , apple , mango , like ]

doc1= np.array ([1,1,0,1]) # vector of words from document 1
doc2= np.array ([1,0,1,1]) # vector of words from document 2
doc3= np.array ([2,0,2,2]) # vector of words from document 3

print( cos_sim (doc1, doc2)) # cosine similarity between document 1 and document 2
print( cos_sim (doc1, doc3)) # cosine similarity between document 1 and document 3
print( cos_sim (doc2, doc3)) # cosine similarity between document 2 and document 3

import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

matx = np.vstack ([doc1, doc2, doc3])
cosine_similarity ( matx )
```

# Euclidean similarity

**2. Euclidean similarity**

```
import numpy as np
def dist ( x,y ):
return np.sqrt ( np.sum ((xy)**2))

#d1 = ' I like apples '
#d2 = ' I like mango '
#d3 = ' I like mango I like mango '
# Word vectors : [ me , apple , mango , like ]
#new = ' I like apples .

doc1= np.array ([1,1,0,1]) # vector of words from document 1
doc2= np.array ([1,0,1,1]) # vector of words from document 2
doc3= np.array ([2,0,2,2]) # vector of words from document 3
docQ = np.array ((1,1,0,1))

print( dist (doc1,docQ))
print( dist (doc2,docQ))
print( dist (doc3,docQ))
```

# Jacquard similarity

**3. Jacquard Similarity**

```
# Two words appearing in both documents: apple and banana .
doc1 = "apple banana everyone like likey watch card holder"
doc2 = "apple banana coupon passport love you"

# Perform tokenization .
tokenized_doc1 = doc1.split()
tokenized_doc2 = doc2.split()

# Output the tokenization result
print(tokenized_doc1)
print(tokenized_doc2)

# union of document 1 and document 2
union = set(tokenized_doc1).union(set(tokenized_doc2))
print(union)

# intersection of document 1 and document 2
intersection = set(tokenized_doc1).intersection(set(tokenized_doc2))
print(intersection)

# Jacquard similarity = number of intersections divided by number of unions
print( len (intersection)/ len (union))
```

# Similarity Between Too Many Documents :
# A Non-Hierarchical Clustering Method

❖ K- means clustering

- used in non-hierarchical clustering

- Hierarchical clustering becomes less useful as

❖ algorithm

Step 1 ) Dividing n entities into K predefined clusters and calculating the centroid of each cluster.

Use the cluster's mean as the center of the cluster

Step 2 ) Calculate the distance between each object and the center of the cluster. Calculate

Step 3 ) Classify each object into the cluster with the shortest distance

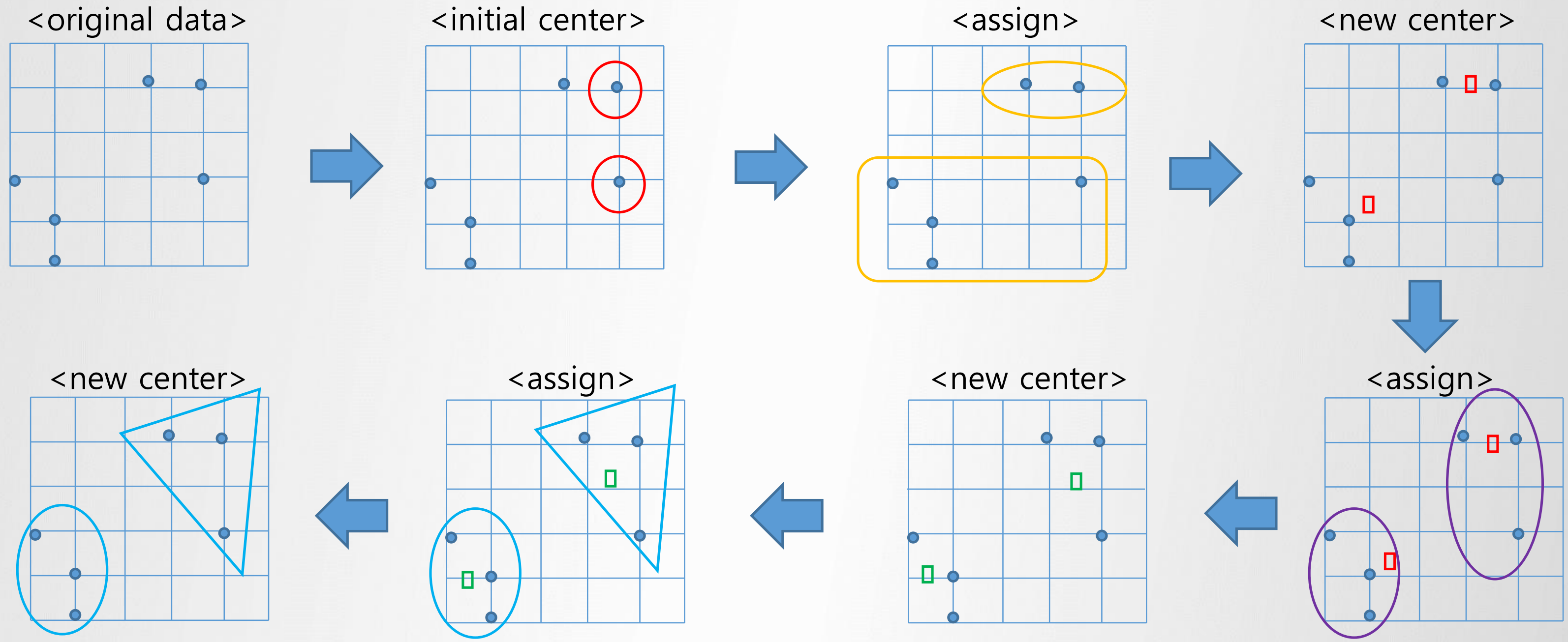
Step 4 ) Repeat the above steps until there are no more clusters

Limitation ) The number of clusters must be known in advance.

Much influenced by the initial classification. ( Different results possible for each analyst )

# K-means clustering

- K-means clustering procedures



# Non-hierarchical clustering method

❖ Algorithm application

Step 1 ) Divide the 5 objects into 2 predefined clusters and calculate the centroid of each cluster.

Randomly classify entities 1 and 2 as cluster A, and entities 3, 4, and 5 as cluster B.

The mean of cluster A is (6 (=(4+8)/2), 4 (=(4+4)/2)) and the mean of cluster B is (21, 8) .

Step 2 ) Calculate the distance between each object and the center of the cluster.

Step 3 ) Classify each object into the cluster with the shortest distance.

Step 4 ) Repeat the above steps until there are no more clusters.

Limitation ) The number of clusters must be known in advance.

Much influenced by the initial classification. ( Different results possible for each analyst )

| | X1 | X2 |
|---|---|---|
| One | 4 | 4 |
| 2 | 8 | 4 |
| 3 | 15 | 8 |
| 4 | 24 | 4 |
| 5 | 24 | 12 |

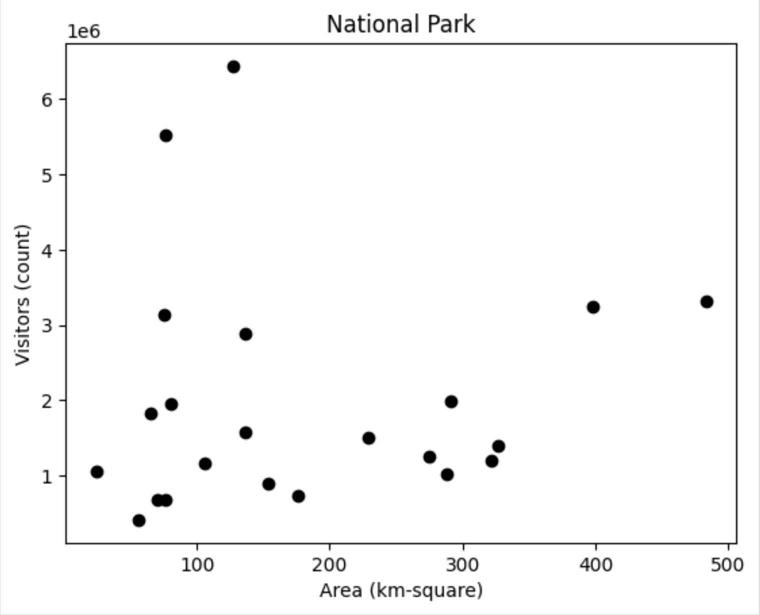
# K-Means clustering

**4. K-Means Clustering**

```
# https://blog.daum.net/geoscience/1515
# https://lucy-the-marketer.kr/en/growth/clustering-python-student-data-analysis/
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.font_manager as fm
import numpy as np
import random as rd
from google.colab import drive
drive.mount ('/content/ gdrive ')

# Specify the file name and save path
file_name = "/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/NLPRK_STA.csv"

# National Park Basic Statistics ( Source : KOSIS National Statistics Portal )
df = pd.read_csv ( file_name , encoding='cp949')
df.head ()
len ( df )

# id: national park , variable 1: land area , variable 2: number of visitors => Group id using variable 1 and variable 2
X = df.iloc [:, [1, 2]].values # shape=(22, 2)
n= X.shape [0] # number of sets (n=22)
m= X.shape [1] # number of features (m=2)
name = df.iloc [:,0].values
```

# K-Means clustering

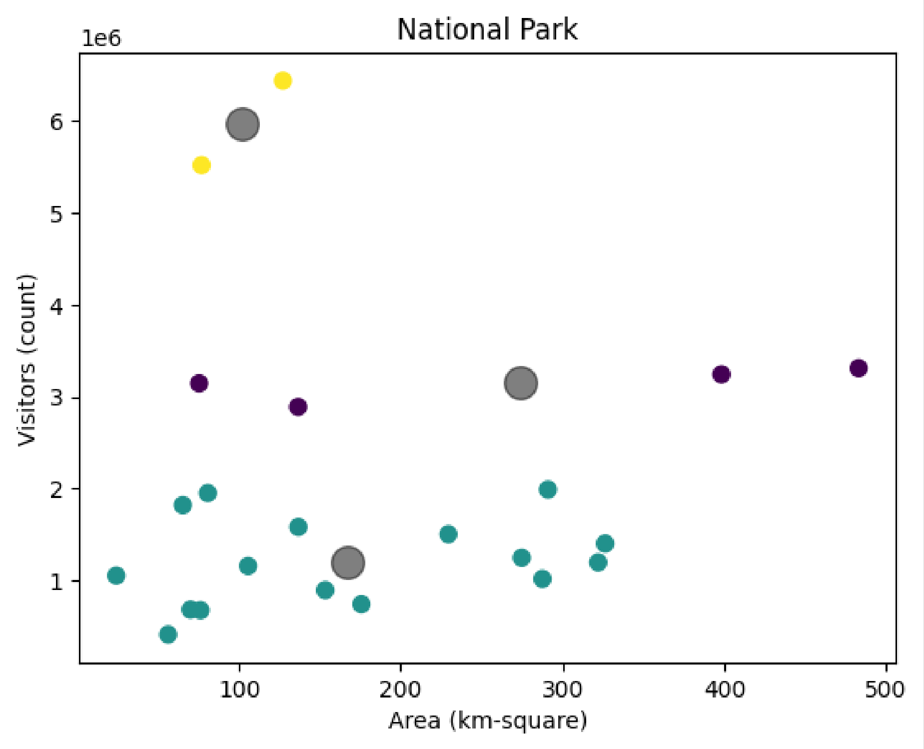
```
# Graph : x -axis : land area , y -axis : number of visitors
plt.scatter (X[:,0],X[:,1],c=' black',label =' national park ')
plt.xlabel ('Area (km-square)')
plt.ylabel ('Visitors(count)')
# plt.legend ()
plt.title ('National Parks')
plt.show ()
```




```
######### K-Means clustering : generate k=3 clusters ( similarity is Fixed to Euclidean distance )
# https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
from sklearn.cluster import KMeans
kmeans = KMeans ( n_clusters =3)
# kmeans.fit (X) # X: ( land area , number of visitors )
y_kmeans = kmeans.fit_predict (X)
# y_kmeans = kmeans.predict (X) # 22 park names {0, 1, 2} separated into 3 groups
list(zip(name, y_kmeans ))
df2 = pd.DataFrame (zip(name, y_kmeans ), columns = [' park name ', ' group (k-mean)'])
df2.sort_values(by=[' group (k-mean)'], axis=0, ascending = False) # Sort descending
#https://rfriend.tistory.com/281
```

# K-Means clustering

```
### Group classification visualization
plt.scatter (X[:, 0], X[:, 1], c= y_kmeans , s=50, cmap = ' viridis ')
centers = kmeans.cluster_centers_
plt.xlabel ('Area (km-square)') # land area
plt.ylabel ('Visitors (count)') # Number of visitors
plt.title ('National Park')
plt.scatter (centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```



# Similarity Analysis Between Multiple Documents: Clustering Method

## ❖ hierarchical clustering

Starting from the assumption that all individuals form an independent cluster, similar individuals are grouped together.

After forming clusters , clustering is performed among clusters, and finally all individuals form one cluster .

1) Single link clustering : single

2) Full link clustering : complete

3) Average link clustering : average

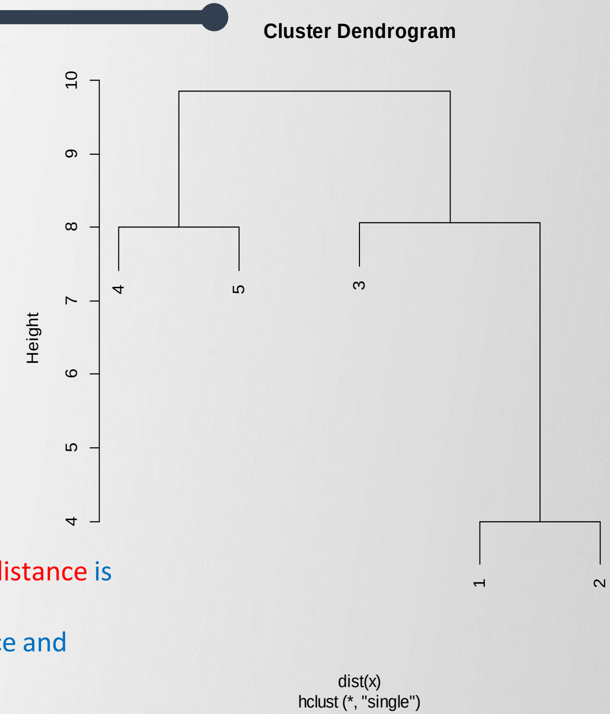
4) Ward link method : ward

# (continued)

- Output

Randomly Generated Material (DTM)

| | X1 | X2 |
|---|---|---|
| One | 4 | 4 |
| 2 | 8 | 4 |
| 3 | 15 | 8 |
| 4 | 24 | 4 |
| 5 | 24 | 12 |




❖ single link method

First of all , (1,2) of Since the distance is the shortest, it is grouped into clusters , and the shortest distance is calculated by comparing the distance between this and the others .
In this case , the shortest distance is (4,5) , so after grouping it again, calculate the minimum distance and group the closest ones together .

| | (1,2) | 3 | 4 | 5 |
|---|---|---|---|---|
| (1,2) | | 8.1 | 16 | 17.9 |
| 3 | 8.1 | | 9.8 | 9.8 |
| 4 | 16 | 9.8 | | 8 |
| 5 | 17.9 | 9.8 | 8 | |

⇨

| | (1,2) | 3 | (4,5) |
|---|---|---|---|
| (1,2) | | 8.1 | 16 |
| 3 | 8.1 | | 9.8 |
| (4,5) | 16 | 9.8 | |

➢ 8.1=min(d13, d23)
16=min(d14, 24)
17.9=min(d15, d25)
8 = d45
9.8 = d34

16=min(d14,d15,d24,d25)
8.1=min(d13,d23)
9.8=min(d34,d35)

# (continued)

|  | X1 | X2 |
|---|---|---|
| One | 4 | 4 |
| 2 | 8 | 4 |
| 3 | 15 | 8 |
| 4 | 24 | 4 |
| 5 | 24 | 12 |

- Output

❖ Full link method

First of all , (1,2) of Since the distance is the shortest, it is grouped into clusters , and the longest distance is calculated by comparing the distance between this and the others .
In this case , the shortest Since the distance is (4,5), group them again, calculate the maximum distance, and group the closest ones together .

|  | (1,2) | 3 | 4 | 5 |
|---|---|---|---|---|
| (1,2) |  | 11.7 | 20 | 21.5 |
| 3 | 11.7 |  | 9.8 | 9.8 |
| 4 | 20 | 9.8 |  | 8 |
| 5 | 21.5 | 9.8 | 8 |  |

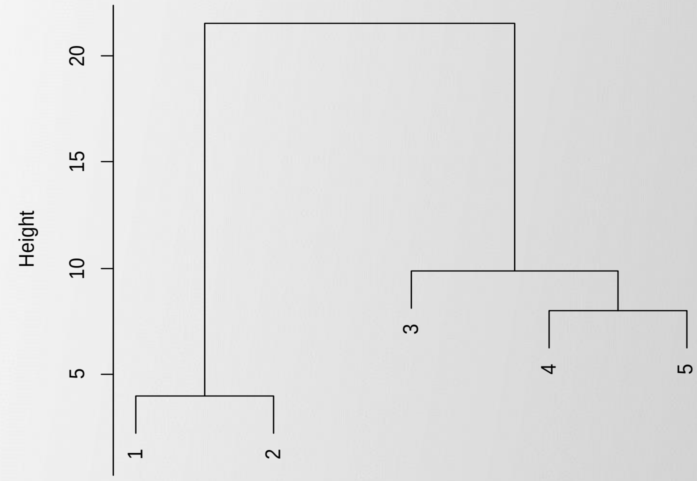
⇨

|  | (1,2) | 3 | (4,5) |
|---|---|---|---|
| (1,2) |  | 11.7 | 21.5 |
| 3 | 11.7 |  | 9.8 |
| (4,5) | 21.5 | 9.8 |  |

- 11.7=max(d13, d23)
  20=max(d14, d24)
  21.5=max(d15, d25)
  9.8=d35=d34
  8=d45

11.7=max(d13, d23)
21.5=max(d14, d15, d24, d25)
9.8=max(d34, d35)

Cluster Dendrogram




dist(x)
hclust (*, "complete")

# (continued)

- Output

❖ Average link method

First of all , (1,2) of Since the distance is the shortest, it is grouped into clusters, and the average distance is calculated by comparing the distance between this and the others .
In this case , the shortest The distance is (4,5) , so after grouping them again, calculating the average distance , group the ones with the closest distance .

| | X1 | X2 |
|---|---|---|
| One | 4 | 4 |
| 2 | 8 | 4 |
| 3 | 15 | 8 |
| 4 | 24 | 4 |
| 5 | 24 | 12 |

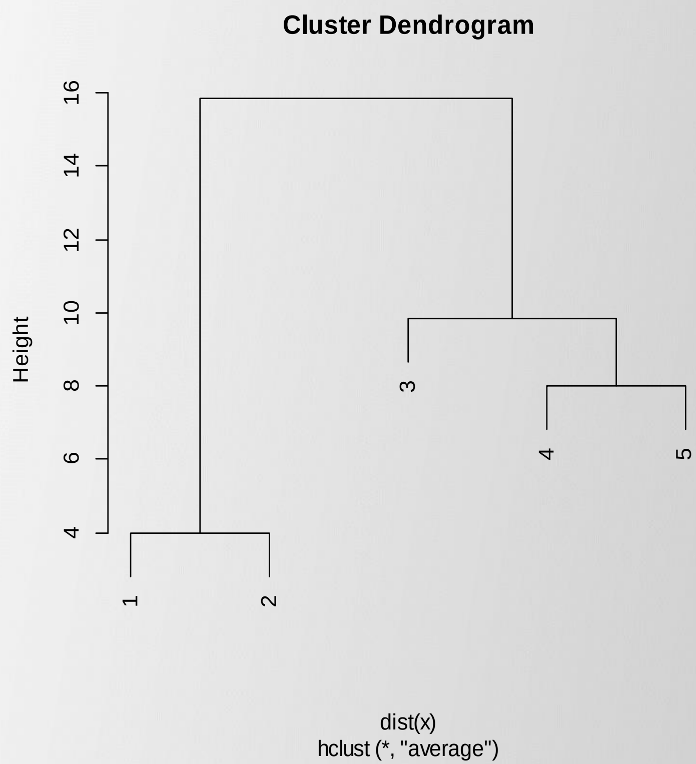
| | (1,2) | 3 | 4 | 5 |
|---|---|---|---|---|
| (1,2) | | 9.9 | 18 | 19.7 |
| 3 | 9.9 | | 9.8 | 9.8 |
| 4 | 18 | 9.8 | | 8 |
| 5 | 19.7 | 9.8 | 8 | |

⇨

| | (1,2) | 3 | (4,5) |
|---|---|---|---|
| (1,2) | | 9.9 | 18.9 |
| 3 | 9.9 | | 9.8 |
| (4,5) | 18.9 | 9.8 | |

- 9.9=(d13+d23)/2 9.9=(d13+d23)/2
  18=(d14+d24)/2
  19.7=(d15+d25)/2
  9.8=d35=d34
  8=d45

18.9=(d14+d15+d24+d25)/4
9.8=(d34+d35)/2



# (continued)

- Output

❖ Ward link method

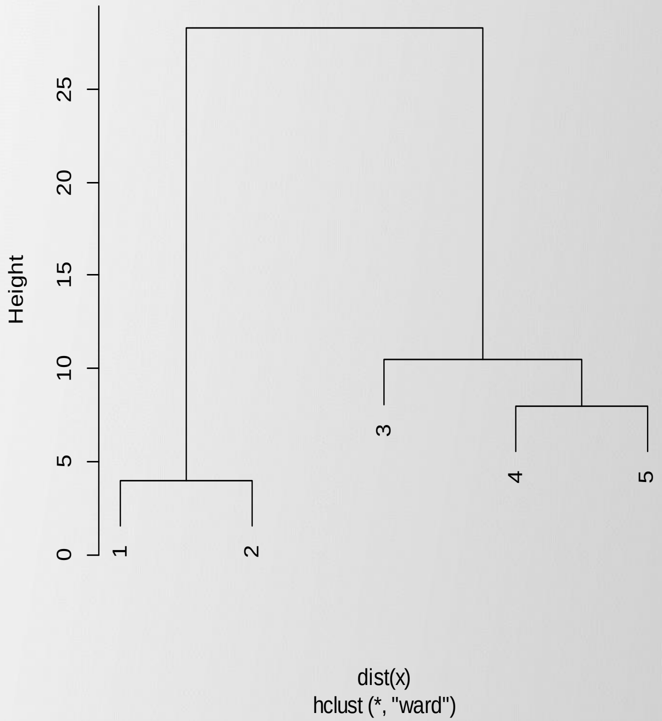
After grouping (1,2) with the smallest SSE as a cluster , calculate the SSE of this and others .
In this case , the smallest SSE is (4,5) , so after grouping it again, calculating the SSE ,
group the ones with the smallest SSE .

| | X1 | X2 |
|---|---|---|
| 1 | 4 | 4 |
| 2 | 8 | 4 |
| 3 | 15 | 8 |
| 4 | 24 | 4 |
| 5 | 24 | 12 |

➢ SSE(1,2)=(4-(4+8)/2)^2+(8-(4+8)/2)^2+(4-(4+4)/2)^2+( 4-(4+4)/2)^2=8
SSE(1,3)=(4-(4+15)/2)^2+(15-(4+15)/2)^2+(4-(4+8)/2)^2+( 8-(4+8)/2)^2=68.5
SSE(1,4)=200, SSE(1,5)=232
SSE(2,3)=32.5, SSE(2,4)=128, SSE(2,5)=160
SSE(3,4)=48.5, SSE(3,5)=48.5
SSE(4,5)=32

➢ SSE(1,2,3)=(4-(4+8+15)/3)^2+(8-(4+8+15)/3)^2+(15-(4+8+15)/3)^2
+(4-(4+4+8)/3)^2+(4-(4+4+8)/3)^2+(8-(4+4+8)/3)^2
SSE(1,2)(3,4)=?
SSE(1,2)(3,4,5)=?

Cluster Dendrogram



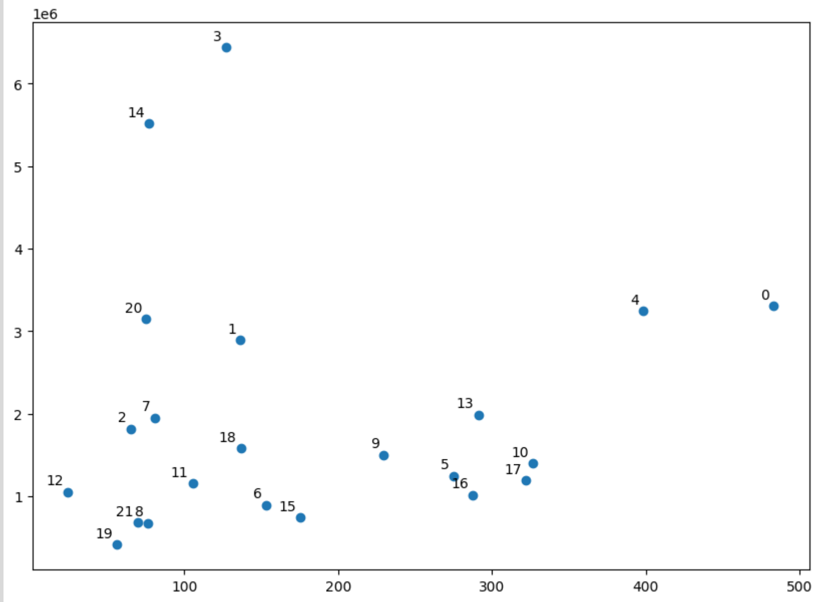
# hierarchical clustering

**5. Hierarchical clustering**

```
# https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.font_manager as fm
import numpy as np
import random as rd
from google.colab import drive
drive.mount ('/content/ gdrive ')

# Specify the file name and save path
file_name = "/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/NLPRK_STA.csv"

# National Park Basic Statistics ( Source : KOSIS National Statistics Portal )
df = pd.read_csv ( file_name , encoding='cp949')
df.head ()
len ( df )
name = df.iloc [:,0].values

# id: national park , variable 1: land area , variable 2: number of visitors => Group id using variable 1 and variable 2
X = df.iloc [:, [1, 2]].values # shape=(22, 2)
n = len (X)

# Park name label
index = list(range(n))
list(zip(index, name))
```

# hierarchical clustering

```
### graph
import matplotlib.pyplot as plt
labels = index
plt.figure ( figsize =(10, 7))
plt.subplots_adjust (bottom=0.1)
plt.scatter (X[:,0],X[:,1], label='True Position')

for label, x, y in zip(labels, X[:, 0], X[:, 1]):
    plt.annotate (
        label
        xy = ( x , y ) , xytext = ( -3 , 3 )
        textcoords = 'offset points', and='right', and ='bottom')
plt.show ()


###### scipy snowflake dendrogram 그리기
!pip install scipy
import scipy.cluster.hierarchy as shc

plt.figure ( figsize =(10,7))
plt.title ("Customer Dendograms ")
dend = shc.dendrogram ( shc.linkage (X, method='ward'))
dend
```



# hierarchical clustering

```
### Grouping : similarity measures ( cosine , euclidean ,...) and clustering Specifies the hierarchical method (ward, etc. )
# https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html
from sklearn.cluster import AgglomerativeClustering

# linkage = ['ward', 'complete', 'average', 'single']
# affinity = [euclidean, l1, l2, manhattan, cosine, precomputed]
# linkage is "ward", only "euclidean" is accepted
# (1) Euclidean 이용
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
h_cluster = cluster.fit_predict (X)
list(zip(name, h_cluster ))
df2 = pd.DataFrame (zip(name, h_cluster ), columns = [' park name ', ' group (h)'])
df2.sort_values(by=[' group (h)'], axis=0, ascending = False) # Sort descending

### visualization
plt.figure ( figsize = (10, 7))
plt.scatter(X[:,0], X[:,1], c=cluster.labels_, cmap='rainbow')

# (2) Cosine 이용
cluster = AgglomerativeClustering(n_clusters=3, affinity='cosine', linkage='complete')
h_cluster = cluster.fit_predict(X)
list(zip(name, h_cluster))
df2 = pd.DataFrame(zip(name, h_cluster), columns = ['공원명', '그룹(h)'])
df2.sort_values(by=[' group (h)'], axis=0, ascending = False) # Sort descending

### visualization
plt.figure ( figsize = (10, 7))
plt.scatter (X[:,0], X[:,1], c= cluster.labels_ , cmap ='rainbow')
```

# Cluster analysis application

**1. Similarity analysis ( using movie title and synopsis : using TF-IDF and cosine similarity ): Used as a recommendation system**

```
# https://wikidocs.net/24603
# Download movies_metadata.csv from https://www.kaggle.com/rounakbanik/the-movies-dataset

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from google.colab import drive
drive.mount('/content/gdrive')

# Specify file path
file_name = "/content/gdrive/My Drive/Colab Notebooks/Textmining/download/movies_metadata.csv"
data = pd.read_csv ( file_name , engine="python")
data. head (3)

# Utilize only 2000 samples
data = data.head (2000)

# Use only
data['overview'][0]
Check if column is null when creating TF-IDF
data['overview']. isnull ().sum()

# If there is a null value in the overview, remove the null value
data['overview'] = data['overview']. fillna ('') # fillna (""): remove null values
data['overview']. isnull ().sum() # Check Null
```

# Cluster analysis application

Create TF-IDF

```
tfidf = TfidfVectorizer ( stop_words = ' english ')
# tf-idf for overview Perform
tfidf_matrix = tfidf.fit_transform (data['overview'])
print( tfidf_matrix.shape ) # rows ( number of movies ) X columns ( number of words )
```

# Create movie index

```
indices = pd.Series ( data.index , index=data['title']). drop_duplicates ()
print( indices. head ()) # Create a table with titles and indices of movies, printing only the first 5
```

# Enter the title of a movie and return the index

```
idx = indices['Father of the Bride Part II']
print( idx )
data['title']
```

# Cluster analysis application

```
##### Using cosine similarity , a function that finds 10 movies with
from sklearn.metrics.pairwise import cosine_similarity
cosine_sim = cosine_similarity ( tfidf_matrix , tfidf_matrix )

def get_recommendations (title, cosine_sim = cosine_sim ):
    # Get the corresponding index from the title of the selected movie . You can now do calculations with selected movies .
    idx = indices[title]

    # For every movie, find the similarity with that movie .
    sim_scores = list(enumerate( cosine_sim [ idx ]))

    # Sort movies according to similarity .
    sim_scores = sorted( sim_scores , key=lambda x: x[1], reverse=True)

    # Get the 10 most similar movies .
    sim_scores = sim_scores [0:10] # starting from 0, self included

    # Get the index of the 10 most similar movies .
    movie_indices = [ i [0] for i in sim_scores ]

    # Returns the titles of the 10 most similar movies .
return data['title']. iloc [ movie_indices ]

###### Find movies with similar overview to Toy Story
get_recommendations ( ' Toy Story ')
```

# Cluster analysis application

**2. Similarity analysis ( using movie title and synopsis : using TF-IDF , K-means , and hierarchical clustering technique ): Used as a recommendation system**

```
# http://brandonrose.org/clustering
!pip install mpld3
import numpy as np
import pandas as pd
import nltk
import re
import os
import codecs
from sklearn import feature_extraction
import mpld3
from google.colab import drive
drive.mount('/content/gdrive')

# 파일명 및 저장 경로 지정하기
file_name = "/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/movies_metadata.csv"
data = pd.read_csv ( file_name , engine="python")
data. head (3)

# Utilize only 20 million samples
data = data.head (2000)

# Use only the movie title (title) and synopsis (overview) among several variables
Check if column is null when creating TF-IDF
data['overview']. isnull ().sum()
```

# Cluster analysis application

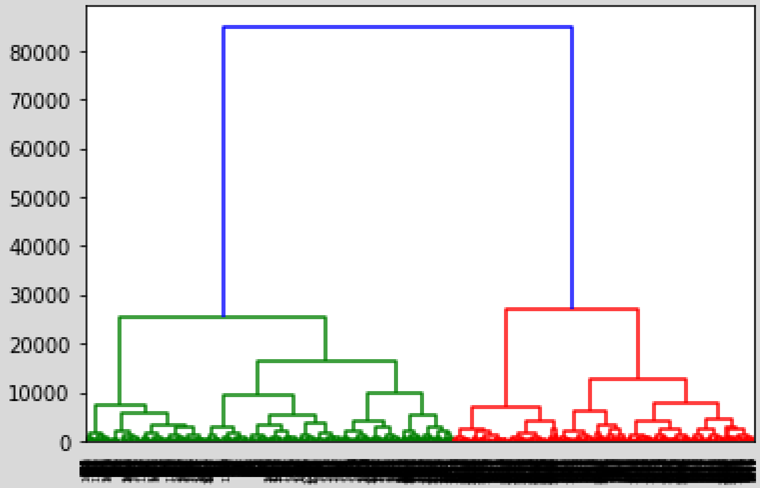
```
# If there is a null value in the overview, remove the null value
data['overview'] = data['overview']. fillna ('') # fillna (""): remove null values
data['overview']. isnull ().sum() # Check Null

titles = data['original_title']
synopses =data['overview']
len(titles)
len(synopses)

print(titles[0])
synopses[0][:200] #first 200 characters in first synopses


########### Tf-idf and document similarity ################################
from sklearn.feature_extraction.text import TfidfVectorizer
# TF-IDF
tfidf_vectorizer = TfidfVectorizer ( stop_words = ' english ')
# tf-idf for overview Perform
tfidf_matrix = tfidf_vectorizer.fit_transform (synopses)
print( tfidf_matrix.shape ) # rows ( number of movies ) X columns ( number of words )
terms = tfidf_vectorizer.get_feature_names_out ()
```

# Cluster analysis application

```
########### K-means clustering
from sklearn.cluster import KMeans
num_clusters = 5
km = KMeans(n_clusters=num_clusters)
%time km.fit(tfidf_matrix)   # %time: 계산 시간을 알려줌
clusters = km.labels_.tolist()

films = { 'title': titles, 'synopsis': synopses, 'cluster': clusters }
frame = pd.DataFrame(films)
frame['cluster'].value_counts()   #number of films per cluster (clusters from 0 to 4)


########### Hierarchical document clustering
# Sklearn library
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
result1 = cluster.fit_predict(tfidf_matrix.toarray())

films1 = { 'title': titles, 'cluster': result1 }
frame1 = pd.DataFrame (films1)
frame1['cluster']. value_counts ()
```

# Cluster analysis application

```
###### scipy dendrogram using library drawing
!pip install scipy
import scipy.cluster.hierarchy as shc
import matplotlib.pyplot as plt

###### calculate distance using cosine
from sklearn.metrics.pairwise import cosine_similarity
dist = 1 - cosine_similarity(tfidf_matrix)

#https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html
linkage_matrix = shc.ward(dist)
plt.figure(figsize=(10, 7))
dend = shc.dendrogram(shc.linkage(linkage_matrix, method='ward'))
dend
```



# Topic model

- Topic model: usually based on a matrix (DTM) of documents and words that might be latent in a document. Statistical text processing technique for estimating the probability of occurrence of hypothesized topics
- Meaning of Topic: It can be arbitrary, so it needs to be interpreted with care.
- Number of topics (k): You can use the ldatuning package, but after setting the range of optimal k values. It would be appropriate to make a selection based on the researcher's theoretical rational reasoning. Functions for selecting the number of topics: coherence, perplexity
- Starting with LDA (Latent Dirichlet Allocation), derivatives of CTM (Correlated Topic Model),

  STM (Structural Topic Model) and BTM (Biterm Topic Model), etc.
- LDA, CTM, STM, and BTM assume that word appearance order or part-of-speech information do not affect the extraction of topics from documents.
- LDA, CTM , STM Approach :

  Decompose into [ document x word ]=[ document x topic ]x[ topic x word ] , and logistic normal distribution for topic extraction use , of topic distribution Independent ( LDA ) or Mutual associate having case ( CTM, STM , etc. ) assumed.

  - STM describes or predicts the possibility of occurrence of topics using metadata (attributes of variables).
  - BTM uses data frame (document ID and word name variables), not document x word matrix. Useful when individual documents in the corpus have very few words (e.g. Twitter)

# Topic model application

**1. Scikit Run LDA Practice**

**(A) TfidfVectorizer** # https://wikidocs.net/30708

Use news data with

```
# 1) Understanding News Article Title Data
# You can download English data, which is a collection of news article titles published for about 15 years, from the link below .
# https://www.kaggle.com/therohk/million-headlines
import pandas as pd
import urllib.request
import nltk
nltk.download (' punkt ')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from google.colab import drive
drive.mount('/content/drive’)

data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Textmining/download/abcnews-date-text.csv', error_bad_lines=False)
print(' Number of news titles :', len (data))

# print only the top 5 samples
print( data. head (5))

# has two columns, publish_data and headline_text
# Indicates the date of the news and the title of the news article, respectively
```

# Topic model application

```
# Of these, headline_text fever Since it is the title of a news article, save only this part separately
text = data[[' headline_text ']]
text. head (5)

#2) Text preprocessing
# Uses three preprocessing techniques : stopword removal , lemma extraction , and short word removal.
text[' headline_text '] = text. apply (lambda row: nltk. word_tokenize (row[' headline_text ']), axis=1)
# Check word tokenization results by outputting only the top 5 samples
print( text. head (5))

# remove stop words
stop_words = stopwords.words (' english ')
text[' headline_text '] = text[' headline_text ']. apply(lambda x: [word for word in x if word not in ( stop_words )])
print( text. head (5)) # Check to remove

# Change 3rd person singular expressions to 1st person by extracting headwords , and change past present tense verbs to present tense
text[' headline_text '] = text[' headline_text ']. apply(lambda x: [ WordNetLemmatizer (). lemmatize(word, pos='v') for word in x])
print( text. head (5))

# Remove words of length 3 or less
tokenized_doc = text[' headline_text '].apply(lambda x: [word for word in x if len (word) > 3])
print( tokenized_doc [:5])
```

# Topic model application

```
# 3) Create a TF-IDF matrix
# reverse tokenization ( revert tokenization operation )
detokenized_doc = []
for i in range( len (text)):
t = ' '.join( tokenized_doc [ i ])
    detokenized_doc.append (t)

# Save back to text[' headline_text ']
text[' headline_text '] = detokenized_doc

# output 5 samples
text[' headline_text '][:5]
```

```
0 decide community broadcast license
1 fire witness must be aware of
defamation
2 call infrastructure protection summit
3 staff aust strike rise
4 strike affect australian travelers
```

```
# of Scikit Learn Creating TF-IDF matrices using TfidfVectorizer
# Simply limit to 1,000 words
# Preserve the top 1,000 words
vectorizer = TfidfVectorizer ( stop_words = ' english ', max_features = 1000)
X = vectorizer.fit_transform (text[' headline_text '])

Check size of TF-IDF matrix : (1244184, 1000)
print(' Size of TF-IDF matrix :', X.shape )
```

# Topic model application

```
#4) Topic Modeling
lda_model = LatentDirichletAllocation ( n_components = 10 , learning_method ='online', random_state
=777,max_iter=1)
lda_top = lda_model.fit_transform (X)
print( lda_model. components _)
print( lda_model.components_.shape ) # (10, 1000) Number
print('#shape of lda_top:', lda_top.shape)
print('#Sample of lda_top:', lda_top[0])
sum(lda_top[0])
a = lda_top[0].tolist()
print(a.index(max(a)))  # Which topic (0~9) of document[0]?

# word set . 1,000 words stored .
terms = vectorizer.get_feature_names_out()

def get_topics(components, feature_names, n=5):
    for idx, topic in enumerate(components):
        print("Topic %d:" % (idx+1), [(feature_names[i], topic[i].round(2)) for i in topic.argsort()[:-n - 1:-1]])

get_topics(lda_model.components_,terms)
Topic 1: [('australia', 20556.0), ('sydney', 11219.29), ('melbourne', 8765.73), ('kill', 6646.06), ('court', 6004.12)]
Topic 2: [('coronavirus', 41719.62), ('covid', 28960.68), ('government', 9793.89), ('change', 7576.98), ('home', 7457.74)]
Topic 3: [('south', 7102.98), ('death', 6825.39), ('speak', 5402.31), ('care', 4521.48), ('interview', 4058.71)]
Topic 4: [('donald', 8536.49), ('restrictions', 6456.4), ('world', 6320.61), ('state', 6087.5), ('water', 4219.67)]
Topic 5: [('vaccine', 8040.87), ('open', 6915.17), ('coast', 5990.55), ('warn', 5472.84), ('morrison', 5247.93)]
Topic 6: [('trump', 14878.13), ('charge', 7717.96), ('health', 6836.95), ('murder', 6663.45), ('house', 6624.13)]
Topic 7: [('australian', 13885.88), ('queensland', 13373.53), ('record', 9037.88), ('test', 7713.9), ('help', 5922.05)]
Topic 8: [('case', 13146.83), ('police', 11143.1), ('live', 7528.03), ('border', 6855.54), ('tasmania', 5664.64)]
Topic 9: [(' victoria ', 11777.5), ('school', 6009.78), ('attack', 5503.39), ('national', 4672.53), ('concern', 4112.28)]
Topic 10: [('election', 8942.32), ('news', 8568.78), ('china', 8452.56), ('people', 6645.62), ('make', 6482.1)]
```

# Topic model application

**(B) CountVectorizer**
# The Complete Guide to Python Text Mining ( Wikibooks )

(1) data preparation

```
import nltk
nltk.download('punkt')
nltk.download('webtext')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')

from sklearn.datasets import fetch_20newsgroups

categories = ['alt.atheism', 'talk.religion.misc', 'comp.graphics', 'sci.space',
              'comp.sys.ibm.pc.hardware', 'sci.crypt']

#학습 데이터셋을 가져옴
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)

print('#Train set size:', len(newsgroups_train.data))
print('#Selected categories:', newsgroups_train.target_names)

# CountVectorizer 이용
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(token_pattern="[₩w']{3,}", stop_words='english',
                     max_features=2000, min_df=5, max_df=0.5)
review_cv = cv.fit_transform(newsgroups_train.data)
```

# Topic model application

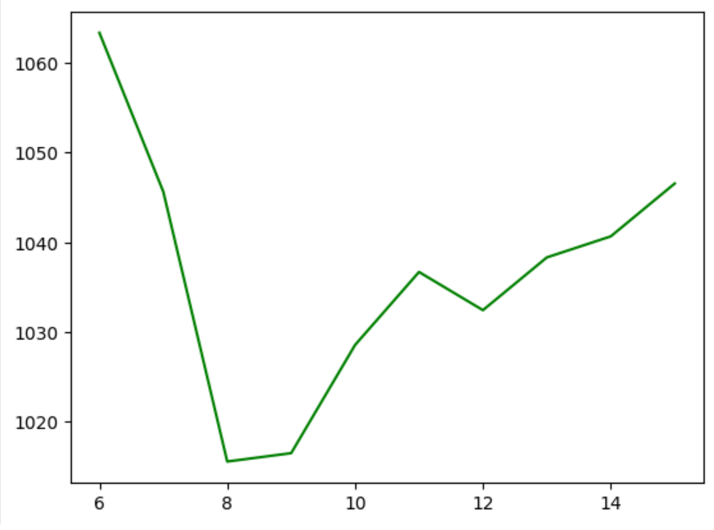
(2) LDA

```
from sklearn.decomposition import LatentDirichletAllocation
import numpy as np
np.set_printoptions(precision=3)

lda = LatentDirichletAllocation(n_components = 10, #number of topic
                                max_iter=5,
                                topic_word_prior=0.1, doc_topic_prior=1.0,
                                learning_method='online',
                                n_jobs= -1,
                                random_state=0)

review_topics = lda.fit_transform(review_cv)
print('#shape of review_topics:', review_topics.shape)
print('#Sample of review_topics:', review_topics[0])

gross_topic_weights = np.mean(review_topics, axis=0)
print('#Sum of topic weights of documents:', gross_topic_weights)
print('#shape of topic word distribution:', lda.components_.shape)

def print_top_words(model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print("Topic #%d: " % topic_idx, end='')
        print(", ".join([feature_names[i] for i in topic.argsort()[:-n_top_words - 1:-1]]))
        #print(", ".join([ feature_names [ i ]+'('+str(topic[ i ])+')' for i in topic.argsort ()[:- n_top_words - 1:-1]]) )
# In the above slicing , -1 at the end means reverse order , from the beginning to n_top_words in reverse order
    print()
print_top_words ( lda,cv.get_feature_names_out (), 10)
```

# Topic model application

(3) Number of topic selection

```
import matplotlib.pyplot as plt
%matplotlib inline

def show_perplexity(cv, start=1, end=30, max_iter=5, topic_word_prior= 0.1,
                    doc_topic_prior=1.0):
    iter_num = []
    per_value = []

    for i in range(start, end + 1):
        lda = LatentDirichletAllocation(n_components = i, max_iter=max_iter,
                                        topic_word_prior= topic_word_prior,
                                        doc_topic_prior=doc_topic_prior,
                                        learning_method='batch', n_jobs= -1,
                                        random_state=7)
        lda.fit(cv)
        iter_num.append(i)
        pv = lda.perplexity(cv)
        per_value.append(pv)
        print(f'n_components: {i}, perplexity: {pv:0.3f}')

    plt.plot(iter_num, per_value, 'g-')
    plt.show()
return start + per_value. index (min( per_value ))
show_perplexity ( review_cv , start=6, end=15)
```



# Topic model application

```
lda = LatentDirichletAllocation ( n_components = 8, # specify the number of topics to extract
                       max_iter = 20,
                              topic_word_prior = 0.1,
                              doc_topic_prior =1.0,
                              learning_method='batch',
                              n_jobs= -1,
                              random_state=7)

review_topics = lda.fit_transform(review_cv)
print_top_words(lda, cv.get_feature_names_out(), 10)
```

```
Topic #0: image, graphics, mail, available, file, ftp, data, files, software, information
Topic #1: nasa, gov, posting, space, university, host, nntp, ___, center, distribution
Topic #2: com, keith, article, morality, think, posting, nntp, caltech, don't, host
Topic #3: com, article, jesus, know, just, posting, host, nntp, don't, i'm
Topic #4: people, god, does, don't, think, say, believe, just, way, like
Topic #5: drive, scsi, card, com, disk, thanks, ide, controller, bus, hard
Topic #6: space, access, article, launch, just, year, like, digex, moon, com
Topic #7: key, encryption, clipper, chip, com, government, keys, use, security, public
```

# Topic model application

**2. gensim LDA practice**

```
import pandas as pd
import urllib.request
import nltk
nltk.download (' point ')
nltk.download (' stopwords ')
nltk.download ('wordnet')
nltk.download ('omw-1.4')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation

from google.colab import drive
drive.mount('/content/drive')

data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Textmining/download/abcnews-date-text.csv',
error_bad_lines=False)
print('number of news title:',len(data))

# Of these, headline_text heat . In other words , since it is the title of a news article, save only this part separately.
text = data[[' headline_text ']]
```

# Topic model application

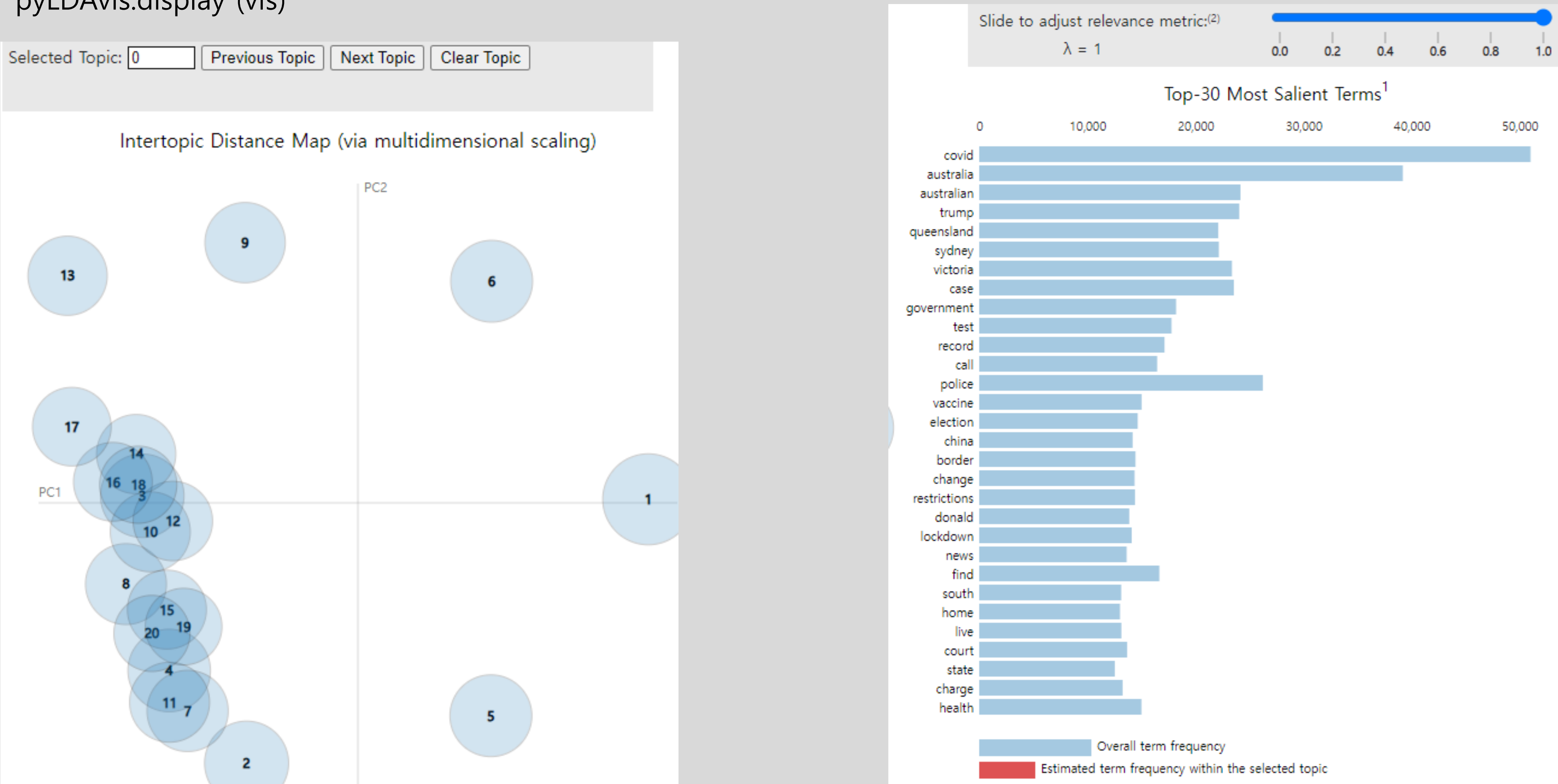
```
####### uses three preprocessing techniques : stopword removal , lemma extraction , and short word removal
####################
text['headline_text'] = text.apply(lambda row: nltk.word_tokenize(row['headline_text']), axis=1)
stop_words = stopwords.words('english')
text['headline_text'] = text['headline_text'].apply(lambda x: [word for word in x if word not in (stop_words)])

# Change 3rd person singular expressions to 1st person by extracting headwords , and change past present tense verbs to present tense
text[' headline_text '] = text[' headline_text ']. apply(lambda x: [ WordNetLemmatizer (). lemmatize(word, pos='v') for word in x])

# remove words of length 3 or less
tokenized_doc = text[' headline_text '].apply(lambda x: [word for word in x if len (word) > 3])
####### Up to this point, it is the same as the previous method with preprocessing
############################### ############



# 1) Encoding integers and creating word sets
tokenized_doc [:5]

# log the frequency of each word in each news , while encoding each word as an integer
# Convert each word to form ( word_id , word_frequency )
# word_frequency means the frequency of the word in the news
# transform the original document using gensim 's corpora.Dictionary ( )
# Save words to corpus using doc2bow
# Perform integer encoding on all news , print the second news
```

## Topic model application

```
from gensim import corpora
dictionary = corpora.Dictionary ( tokenized_doc )
corpus = [dictionary. doc2bow (text) for text in tokenized_doc ]
print(corpus[1]) # Print the second news from the executed result . The index of the first document is 0
# (4, 1) means that the word assigned an integer encoding of 4 appeared once in the second news

#4 was before integer encoding
print(dictionary[4]) # aware

# check the length of the dictionary
len (dictionary) #92311


# 2) Train the LDA model
import gensim
NUM_TOPICS = 20 # 20 topics , k=20
ldamodel = gensim.models.ldamodel. LdaModel (corpus, num_topics = NUM_TOPICS, id2word=dictionary, passes=15)
topics = ldamodel.print_topics ( num_words =4)
for topics in topics:
print(topic)

# The number in front of each word is the contribution of the word to that topic.
# Since the topic number at the beginning starts from 0 , a total of 20 topics are assigned numbers from 0 to 19 .
# passes refers to the number of operations of the algorithm , so that the value of the topic determined by the algorithm can
converge properly
# You can set a sufficient number of times . Execute a total of 15 times # num_words =4 outputs a total of 4 words
# If you want to print 10 words, run the code below
print( ldamodel. print_topics ())
```

# Topic model application

```
# 3) Visualize LDA
! pip install pyLDAvis
import pyLDAvis.gensim_models
pyLDAvis.enable_notebook ()
vis = pyLDAvis.gensim_models.prepare ( ldamodel , corpus, dictionary)
pyLDAvis.display (vis)
```



# Topic model application

```
# 4) View Topic Distribution by Document
for i , topic_list in enumerate( ldamodel [corpus]):
if i == 5:
break
print( i ,' th document's percentage of topics ', topic_list )
```

```
The topic ratio of the 0th document is [(9, 0.21), (12, 0.41), (13, 0.21)]
The topic ratio of the first document is [ ( 2, 0.34166667), (5, 0.175), (12, 0.175),
(18, 0.175)]
The topic ratio for the 2nd document is [ ( 5, 0.61), (19, 0.21)]
The topic ratio for the 3rd document is [ ( 0, 0.41), (4, 0.21), (5, 0.21)]
The topic ratio for the 4th document is [(0, 0.21), (4, 0.41), (10, 0.21)]
```

# Topic model application

```
# output in dataframe format
def make_topictable_per_doc ( ldamodel , corpus):
    topic_table = pd.DataFrame ()

    # Take out the document number, which means the number of the document, and the weight of the topic of the docume
nt, line by line .
for i , topic_list in enumerate( ldamodel [corpus]):
doc = topic_list [0] if ldamodel.per_word_topics else topic_list
doc = sorted(doc, key=lambda x: (x[1]), reverse=True)
# For each document, sort the topics in order of importance .
# EX) Document 0 before sorting : ( Topic 2 , 48.5%), ( Topic 8 , 25%), ( Topic 10 , 5%), ( Topic 12 , 21.5%),
# Ex) Document 0 after sorting : ( Topic 2 , 48.5%), ( Topic 8 , 25%), ( Topic 12 , 21.5%), ( Topic 10 , 5%)
# Arranged in order of 48 > 25 > 21 > 5 .

# Do the following for each document
      for j, ( topic_num , prop_topic ) in enumerate(doc): # Store the number of topics and their respective weights .
if j == 0: # Sort, so the first topic is the most important topic
            topic_table = topic_table.append ( pd.Series ([int( topic_num ), round(prop_topic,4), topic_list ]), ignore_index
=True)
               # Stores the most important topic , the weight of the most important topic , and the weight of all topics .
else:
break
return( topic_table )


topictable = make_topictable_per_doc ( ldamodel , corpus)
topictable = topictable.reset_index () # Create another index column to use as a column for the document number .
topictable.columns = [' Article Number ', ' Most Topic ', ' Most Topic Weight ', ' Most Weighted Each Topic ']
topictable [10]
```

# Topic model application

**3. Similarity Analysis ( Using Movie Title and Synopsis : Using Topic Model (LDA) )**

```
# http://brandonrose.org/clustering
!pip install mpld3
import numpy as np
import pandas as pd
import nltk
import re
import os
import codecs
from sklearn import feature_extraction
import mpld3
from google.colab import drive
drive.mount ('/content/ gdrive ')

# Specify the file name and save path
file_name = "/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/movies_metadata.csv"
data = pd.read_csv ( file_name , engine="python")
data. head (3)

Check if column is null when creating TF-IDF
data['overview']. isnull ().sum()

# If there is a null value in the overview, remove the null value
data['overview'] = data['overview']. fillna ('') # fillna (""): remove null values
data['overview']. isnull ().sum() # Check Null
```

# Topic model application

```
# Utilize only 2000 samples
data = data.head (2000)
titles = data[' original_title ']
synopses =data['overview']


# 전처리용 함수
import string
def strip_proppers(text):
    # first tokenize by sentence, then by word to ensure that punctuation is caught as it's own token
    tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent) if word.islower()]
    return "".join([" "+i if not i.startswith("'") and i not in string.punctuation else i for i in tokens]).strip()

def tokenize_and_stem(text):
    # first tokenize by sentence, then by word to ensure that punctuation is caught as it's own token
    tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
    filtered_tokens = []
    # filter out any tokens not containing letters (e.g., numeric tokens, raw punctuation)
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    stems = [stemmer.stem(t) for t in filtered_tokens]
    return stems
```

# Topic model application

```
from gensim import corpora, models, similarities
import nltk
nltk.download('punkt')
nltk.download('stopwords')
stopwords.words('english')[0:10]
stopwords = nltk.corpus.stopwords.words('english')

#remove proper names
%time preprocess0 = [strip_proppers(doc) for doc in synopses]  # synopses는 data['overview']: 영화 줄거리

#remove apostrophe s
preprocess1 = [doc1.replace("'s", "") for doc1 in preprocess0]

#remove special a word
preprocess = [doc2.replace("n₩'t", "") for doc2 in preprocess1]

#tokenize
%time tokenized_text = [tokenize_and_stem(text) for text in preprocess]

#remove stop words
%time texts = [[word for word in text if word not in stopwords] for text in tokenized_text]

print(texts)
print(texts[0])
len(texts)
len(texts[0])
```

# Topic model application

```
#create a Gensim dictionary from the texts
dictionary = corpora.Dictionary(texts)
list(dictionary)

#remove extreme words (similar to the min/max df step used when creating the tf-idf matrix)
dictionary.filter_extremes(no_below=1, no_above=0.8)

#convert the dictionary to a bag of words corpus for reference
corpus = [dictionary.doc2bow(text) for text in texts]
len(corpus)
# 각 문서별 (단어, 빈도수) 의 list
print(corpus[10])

########### Topic model: specify number of topics
%time lda = models.LdaModel(corpus, num_topics=5, id2word=dictionary, update_every=5, chunksize=10000, passes=
100)
# Each topic has a set of words that defines it
lda.show_topics()

topics = lda.print_topics(num_words=10)
for topic in topics:
    print(topic)

#토픽 5개: 0~4
#(0, '0.020*"hi" + 0.007*"get" + 0.005*"find" + 0.005*"take" + 0.005*"team" + 0.005*"young" + 0.005*"help" + 0.004*"love" + 0.004*"stori" + 0.004*"tri"')
#(1, '0.046*"hi" + 0.009*"ha" + 0.008*"film" + 0.007*"friend" + 0.007*"life" + 0.007*"famili" + 0.007*"young" + 0.006*"becom" + 0.006*"thi" + 0.005*"find"')
#(2, '0.033*"hi" + 0.010*"ha" + 0.007*"find" + 0.006*"live" + 0.006*"year" + 0.006*"life" + 0.005*"one" + 0.005*"wife" + 0.005*"two" + 0.005*"friend"')
#(3, '0.025*"hi" + 0.007*"life" + 0.006*"ha" + 0.005*"love" + 0.005*"live" + 0.005*"find" + 0.005*"make" + 0.005*"young" + 0.005*"thi" + 0.005*"get"')
#(4, '0.019*"hi" + 0.009*"ha" + 0.008*"man" + 0.007*"one" + 0.007*"father" + 0.007*"live" + 0.006*"get" + 0.005*" find" + 0.005*"young" + 0.005*"love"')
```

# Topic model application

```
#perflexibility: Search optimal number of topics
# Find the interval with the lowest value and select the optimal number of topics
# However , if the y-axis is negative, it is not appropriate to use
import matplotlib.pyplot as plt
perplexity_values = []
for i in range(2, 10):
  ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics = i, id2word=dictionary)
  perplexity_values.append(ldamodel.log_perplexity(corpus))

x = range(2, 10)
plt.plot(x, perplexity_values)
plt.xlabel("number of topics")
plt.ylabel("perplexity score")
plt.show()
```

# Topic model application

```
# convert the topics into the top 20 words in each topic
import numpy as np
topics_matrix = lda.show_topics (formatted=False, num_words =20 )
topics_matrix = np.array ( topics_matrix )
topics_matrix.shape

topics_matrix [0,1] # top 20 words of first topic and probabilities
topics_matrix [0,1][0] # top 1 word in first topic and probabilities

# Create a matrix of the top 20 words per topic and their occurrence probability
topic_words = topics_matrix [:,1]
for i in topic_words:
    print([str(word) for word in i])
    print()
```

```
["('hi', 0.046179127)", "('ha', 0.009918294)", "('get', 0.0073311045)", "('life', 0.007301379)", "('find', 0.00672676)", "('wife', 0.005817176)", "('friend', 0.00537953)", "('help', 0.005026635)", "('tri', 0.0045415345)", "('work', 0.004470129)", "('live', 0.004425876)", "('one', 0.004354662)", "('famili', 0.0042638136)", "('young', 0.0040400517)", "('becom', 0.0038308685)", "('time', 0.0037564558)", "('make', 0.003526484)", "('new', 0.0034016042)", "('daughter', 0.003382129)", "('back', 0.0033613625)"]

["('hi', 0.014568295)", "('love', 0.0074597355)", "('becom', 0.0063061235)", "('film', 0.004373097)", "('one', 0.0043533356)", "('stori', 0.004091147)", "('girl', 0.0038401587)", "('ha', 0.0036824096)", "('father', 0.0036058913)", "('get', 0.0035501595)", "('women', 0.0034420814)", "('young', 0.0033873245)", "('life', 0.0033141999)", "('rescu', 0.0032391849)", "('thi', 0.0031734419)", "('discov', 0.0030876377)", "('wa', 0.0030828917)", "('meet', 0.0028788808)", "('take', 0.0027476214)", "('dure', 0.0027429552)"]

["('hi', 0.025910392)", "('young', 0.009041517)", "('friend', 0.008844582)", "('film', 0.007952838)", "('life', 0.0071295965)", "('ha', 0.007045285)", "('find', 0.0065567307)", "('man', 0.0065036267)", "('love', 0.005872036)", "('stori', 0.00582041)", "('thi', 0.005565319)", "('live', 0.0055244393)", "('becom', 0.005373759)", "('two', 0.0047534313)", "('year', 0.004387853)", "('one', 0.0042414432)", "('school', 0.0040359534)", "('woman', 0.0039027827)", "('girl', 0.0037916915)", "('meet', 0.0037799515)"]

["('hi', 0.02344223)", "('famili', 0.010095864)", "('take', 0.006818073)", "('ha', 0.00673385)", "('man', 0.00591352)", "('one', 0.0057992907)", "('find', 0.005746425)", "('young', 0.005686137)", "('father', 0.005323221)", "('son', 0.0053060073)", "('meet', 0.004850732)", "('year', 0.0046174056)", "('murder', 0.0045389216)", "('live', 0.004533878)", "('stori', 0.003979337)", "('two', 0.003812425)", "('woman', 0.0035914525)", "('boy', 0.0035735436)", "('life', 0.0035660593)", "('film', 0.0033936054)"]

["('hi', 0.020823201)", "('ha', 0.008688775)", "('love', 0.005928242)", "('thi', 0.005519404)", "('must', 0.0053986446)", "('find', 0.004739682)", "('killer', 0.0044990582)", "('turn', 0.0036266388)", "('onli', 0.003571175)", "('new', 0.0035600008)", "('serial', 0.0035537658)", "('work', 0.003545533)", "('tri', 0.0034741971)", "('fall', 0.0034309288)", "('get', 0.0030795054)", "('stori', 0.0029939236)", "('murder', 0.002893566)", "('play', 0.002705948)", "('power', 0.002694542)", "('live', 0.0026894265)"]
```

# Topic model application

```
# per document View Topic ( show only m )
m = 20
for i , topic_list in enumerate( lda [corpus]):
if i == m:
break
print( i ,' th document's percentage of topics ', topic_list )

The topic ratio of the #0th document is [(1, 0.9658418)]
The topic ratio of #1 document is [(3, 0.36365235), (4, 0.61931485)]
The topic ratio of the #2nd document is [(0, 0.9728442)]
# The topic ratio of the 3rd document is [(0, 0.961303)]
# 4th document's topic ratio is [(0, 0.6734644), (1, 0.29876304)]
# The topic ratio of the 5th document is [(3, 0.9717258)]
The topic proportions of #6th document are [(0, 0.010816155), (1, 0.01078664), (2, 0.9569362), (3, 0.010722903), (4, 0.010738126)]
The topic ratio of # 7th document is [(1, 0.60304856), (4, 0.37574196)]
```

# Topic model application

```
def make_topictable_per_doc ( ldamodel , corpus):
    topic_table = pd.DataFrame ()

    # Take out the document number, which means the number of the document, and the weight of the topic of the docume
nt, line by line .
for i , topic_list in enumerate( ldamodel [corpus]):
doc = topic_list [0] if ldamodel.per_word_topics else topic_list
doc = sorted(doc, key=lambda x: (x[1]), reverse=True)
        # For each document, sort the topics in order of importance .
# EX) Document 0 before sorting : ( Topic 2 , 48.5%), ( Topic 8 , 25%), ( Topic 10 , 5%), ( Topic 12 , 21.5%),
# Ex) Document 0 after sorting : ( Topic 2 , 48.5%), ( Topic 8 , 25%), ( Topic 12 , 21.5%), ( Topic 10 , 5%)
# Arranged in order of 48 > 25 > 21 > 5 .

# Do the following for each document
      for j, ( topic_num , prop_topic ) in enumerate(doc): # Store the number of topics and their respective weights .
if j == 0: # Sort, so the first topic is the most important topic
            topic_table = topic_table.append ( pd.Series ([ int ( topic_num ), round(prop_topic,4), topic_list ]),
ignore_index =True)
                # Stores the most important topic , the weight of the most important topic , and the weight of all topics .
else:
break
return( topic_table )

topictable = make_topictable_per_doc ( lda , corpus)
```

# Topic model application

```
topictable = topictable.reset_index () # Create another index column to use as a column for the document number .
topictable.columns = [' Article Number ', ' Most Topic ', ' Most Topic Weight ', ' Most Weighted Each Topic ']
topictable [:10]

# document number Most important topic The highest percentage of topics Weight of each topic
# 0 1.0 0.9658 [(1, 0.96584684)]
# 1 4.0 0.6193 [(3, 0.36365235), (4, 0.6193149)]
# 2 0.0 0.9728 [(0, 0.97284466)]
# 3 0.0 0.9613 [(0, 0.96131015)]
#4 0.0 0.6735 [(0, 0.67349595), (1, 0.2987315)]
#5 3.0 0.9717 [(3, 0.9717281)]
# 6 2.0 0.9569 [(0, 0.010814987), (1, 0.010786607), (2, 0.956...
# 7 1.0 0.6030 [(1, 0.603038), (4, 0.3757525)]
#8 2.0 0.9778 [(2, 0.9778024)]
# 9 1.0 0.9373 [(0, 0.015760956), (1, 0.9372948), (2, 0.01566...
```

# Topic model application

**4. Analysis of Corona-related topics in Naver News**

```
from google.colab import drive
drive.mount ('/content/ gdrive ')
import json
file_name = '/content/gdrive/My Drive/Colab Notebooks/Textmining/download/코로나_naver_news.json'
with open(file_name, "r", encoding = "utf-8") as f:
  data = json.load(f)
print(data)

# 데이터프레임 만들기
title = []
description = []
for i in data:
  title.append(i['title'])
  description.append ( i ['description'])
title
description
df = pd.DataFrame ({'title': title, 'description': description})
df

# Preprocessing ( remove all non-Korean characters )
import re
df ['title'] = df ['title'].apply(lambda x : re.sub (r'[^ □ - □ | 가 -劵]+', " ", x))
df ['description'] = df ['description'].apply(lambda x: re.sub (r'[^ a - heh | a - heh ] +', " ", x))
df.head ()
```

# Topic model application

```
!pip install konlpy
import conlpy
from konlpy.tag import Oct
oct = oct ()

des = df ['description']

des_noun_tk = []
for j in des:
  des_noun_tk.append ( okt.nouns (j)) #

des_noun_tk2 = []
for k in des_noun_tk :
item = [ i for i in k if len ( i )>1] # only extract token length greater than 1
  des_noun_tk2.append(item)

print(des_noun_tk2)
print(des_noun_tk2[0])
len (des_noun_tk2)
```

# Topic model application

```
# Building the LDA model
#https://lovit.github.io/ nlp /2018/09/27/ pyldavis_lda /
!pip install gensim
import gensim
import gensim.corpora as corpora

dictionary = corpora.Dictionary (des_noun_tk2)
print(dictionary)
len (dictionary)

# List of ( words , frequency ) for each document
corpus = [dictionary.doc2bow(word) for word in des_noun_tk2]
print(corpus)
print(corpus[0])

# Specify the number of topics
k=5
from gensim.models import LdaModel
lda_model = LdaModel ( corpus , id2word = dictionary , num_topics = k )
lda_model.show_topics ()

# topic for each document (only m amount of them)
m = 20
for i , topic_list in enumerate( lda [corpus]):
if i ==m:
break
print( i ,' th document's percentage of topics ', topic_list )
```

# Topic model application

```
def make_topictable_per_doc ( ldamodel , corpus):
    topic_table = pd.DataFrame ()

    # Take out the document number, which means the number of the document, and the weight of the topic of the docume
nt, line by line .
for i , topic_list in enumerate( ldamodel [corpus]):
doc = topic_list [0] if ldamodel.per_word_topics else topic_list
doc = sorted(doc, key=lambda x: (x[1]), reverse=True)
        # For each document, sort the topics in order of importance .
# EX) Document 0 before sorting : ( Topic 2 , 48.5%), ( Topic 8 , 25%), ( Topic 10 , 5%), ( Topic 12 , 21.5%),
# Ex) Document 0 after sorting : ( Topic 2 , 48.5%), ( Topic 8 , 25%), ( Topic 12 , 21.5%), ( Topic 10 , 5%)
# Arranged in order of 48 > 25 > 21 > 5 .

# Do the following for each document
      for j, ( topic_num , prop_topic ) in enumerate(doc): # Store the number of topics and their respective weights .
if j == 0: # Sort, so the first topic is the most important topic
            topic_table = topic_table.append ( pd.Series ([ int ( topic_num ), round(prop_topic,4), topic_list ]),
ignore_index =True)
                # Stores the most important topic , the weight of the most important topic , and the weight of all topics .
else:
break
return( topic_table )

topictable = make_topictable_per_doc ( lda_model , corpus)
topictable = topictable.reset_index () # Create another index column to use as a column for the document number .
topictable.columns = [' Article Number ', ' Most Topic ', ' Most Topic Weight ', ' Most Weighted Each Topic ']
topictable [:10]
```