문서 유사도 & Clustering

성균관대학교
정윤경

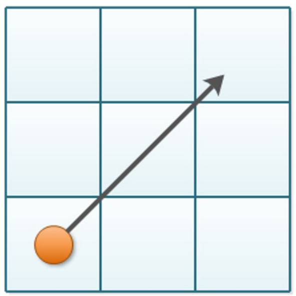
Outline

- 문서 유사도
- Text clustering

문서 유사도

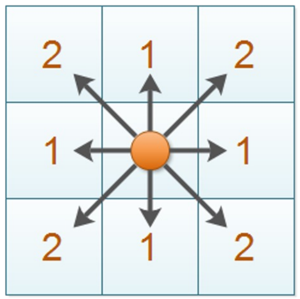
Euclidean distance vs. Manhattan distance

Euclidean Distance

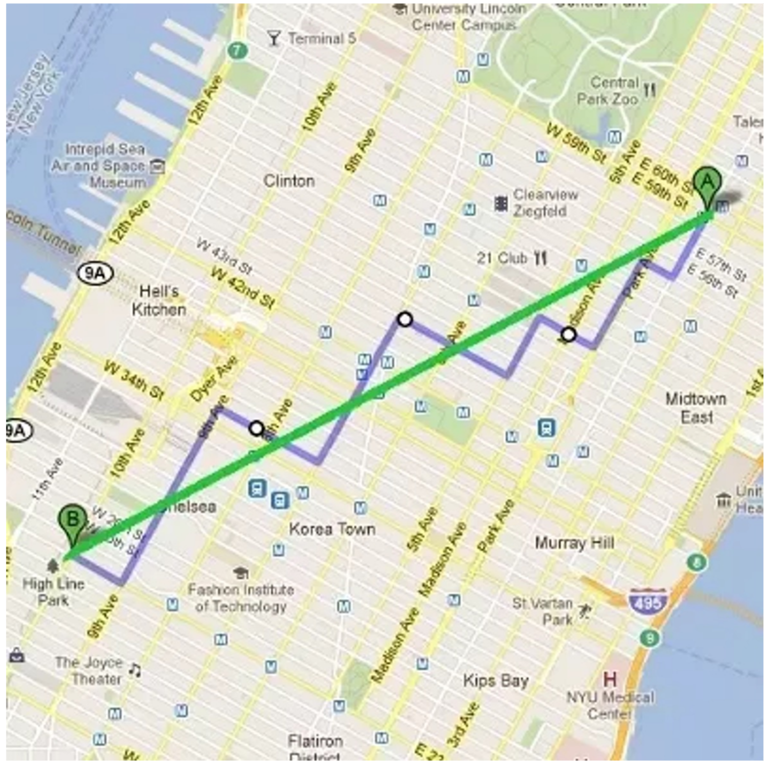


$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Manhattan Distance



$$|x_1 - x_2| + |y_1 - y_2|$$



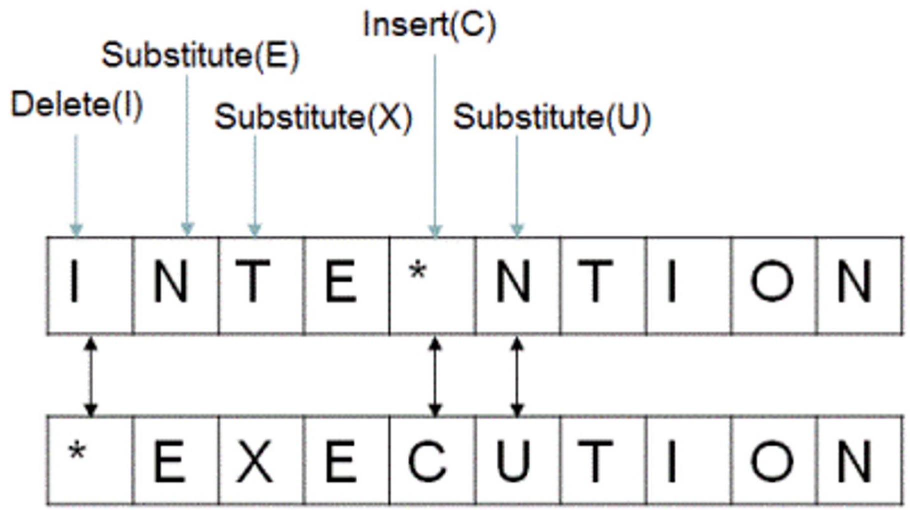
Hamming distance

Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different

- "karolin" and "kathrin" is 3
- "karolin" and "kerstin" is 3
- **1011101** and **1001001** is 2
- **2173896** and **2233796** is 3

Levenshtein edit distance

- **edit distance**: minimum number of operations required to transform one string into the other
- Operations are adding/removing or substituting letters from the strings
- Uses dynamic programming algorithm to compute the edit distance

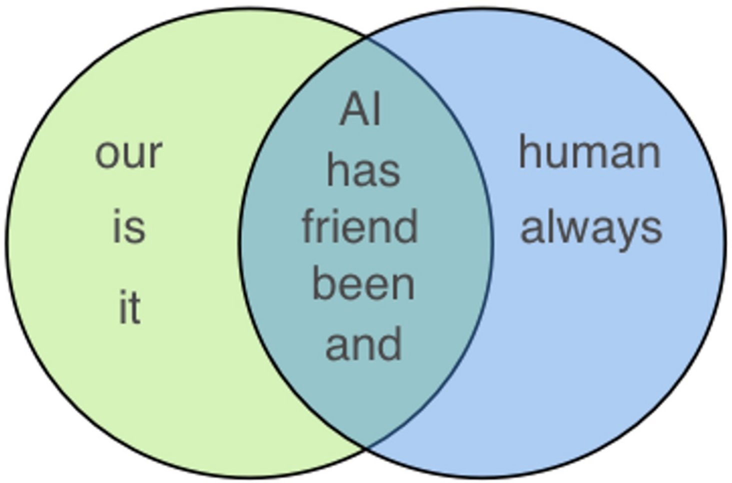


Jaccard Index

- Number of letters match
  - “Dynamo” and “yDnamo” as being identical
  - “Dynamo” and “Dyno” is a better match than “Dinomo”, because “Dyno” is only four letters long and shares more letters in common.

Jaccard Index	Dynamo	dynamo	yDnamo	Dyno	Dymamo	Dinomo
Dynamo	1.00	0.71	1.00	0.67	0.83	0.57
dynamo	0.71	1.00	0.71	0.43	0.57	0.38
yDnamo	1.00	0.71	1.00	0.67	0.83	0.57
Dyno	0.67	0.43	0.67	1.00	0.50	0.50
Dymamo	0.83	0.57	0.83	0.50	1.00	0.43
Dinomo	0.57	0.38	0.57	0.50	0.43	1.00

Jaccard Index




```
def jaccard(str1, str2):

    a = set(str1.lower().split())

    b = set(str2.lower().split())

    c = a.intersection(b)

    return float(len(c)) / (len(a) + len(b) - len(c))
```

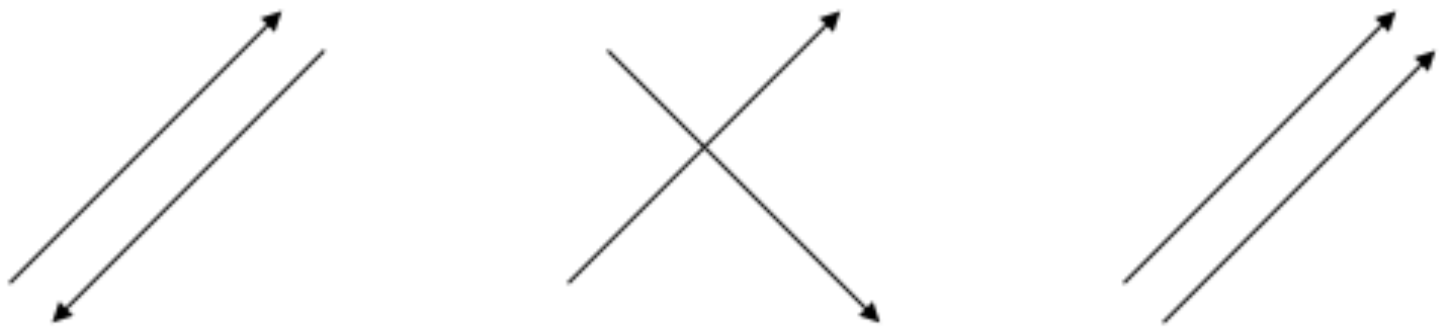
https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50

nltk.metrics.distance module

```
>>> from nltk.metrics import *
>>> edit_distance("rain", "shine") # Levenshtein edit-distance
3
>>> s1 = set([1,2,3,4])
>>> s2 = set([3,4,5])
>>> print(jaccard_distance(s1, s2))
0.6
>>> print(jaccard_distance(set('rain'), set('shine')))
0.7142857142857143
```

Cosine similarity

$$similarity = cos(\Theta) = \frac{A \cdot B}{||A||\ ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

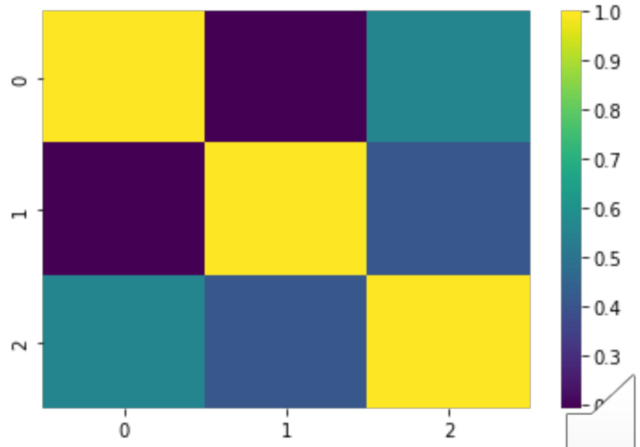


코사인 유사도 : -1　　코사인 유사도 : 0　　코사인 유사도 : 1

```
from konlpy.tag import Okt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
x_data = np.array(['영희가 사랑하는 강아지 백구를 산책시키고 있다.',
      '철수가 사랑하는 소 누렁이를 운동시키고 있다.',
      '영희와 철수는 소와 강아지를 산책 및 운동시키고 있다.'])
twitter = Okt()
for i, document in enumerate(x_data):
   nouns = twitter.nouns(document)
   x_data[i] = ' '.join(nouns)
print(x_data)
vect = TfidfVectorizer()
x_data = vect.fit_transform(x_data)
cosine_similarity_matrix = cosine_similarity(x_data,  x_data)
print(cosine_similarity_matrix)
sns.heatmap(cosine_similarity_matrix.toarray(), cmap='viridis')
plt.show()
```

```
['영희 사랑 강아지 백구 산책', '철수 사랑 소
누렁이 운동', '영희 철수 소 강아지 산책 및 운동']
```

```
[[1.         0.19212486 0.56053185]
 [0.19212486 1.         0.4113055 ]
 [0.56053185 0.4113055  1.        ]]
```



출처: https://wikidocs.net/76349

유사도를 이용한 추천 시스템 구현하기

- 코드 출처: https://wikidocs.net/24603
- 데이터셋 출처 : https://www.kaggle.com/rounakbanik/the-movies-dataset
- movies_metadata.csv: 총 24개의 열을 가진 45,466개의 샘플로 구성된 영화 정보 데이터
- Adult, belongs_to_collection, budget, genres, homepage, id, imdb_id, original_language, original_title, **overview**, popularity, poster_path, production_companies, production_countries, release_date, revenue, runtime, spoken_languages, status, tagline, **title**, video, vote_average, vote_count

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

data = pd.read_csv('movies_metadata.csv', low_memory=False)
data = data.head(20000)
data['overview'] = data['overview'].fillna(’’) # 결측값을 빈 값으로 대체

tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(data['overview'])
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
#영화의 타이틀을 key, 영화의 인덱스를 value로 하는 딕셔너리
title_to_index = dict(zip(data['title'], data.index))

def get_recommendations(title, cosine_sim=cosine_sim):
    #  선택한 영화 제목으로 인덱스 검색
    idx = title_to_index[title]
    # 해당 영화와 모든 영화와의 유사도
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11]
    # 가장 유사한 10개의 영화의 인덱스 리턴
    movie_indices = [idx[0] for idx in sim_scores]
    # 가장 유사한 10개의 영화의 제목을 리턴
    return data['title'].iloc[movie_indices]

get_recommendations('The Dark Knight Rises')
```

```
12481 The Dark Knight
150 Batman Forever
1328 Batman Returns
15511 Batman: Under the Red Hood
585 Batman
9230 Batman Beyond: Return of the Joker
18035 Batman: Year One
19792 Batman: The Dark Knight Returns, Part 1
3095 Batman: Mask of the Phantasm
10122 Batman Begins
```

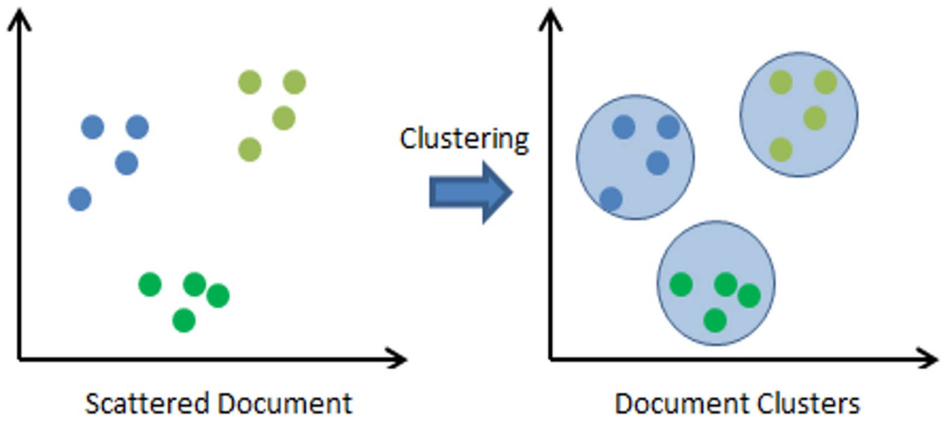
출처: https://wikidocs.net/76349

Text Clustering

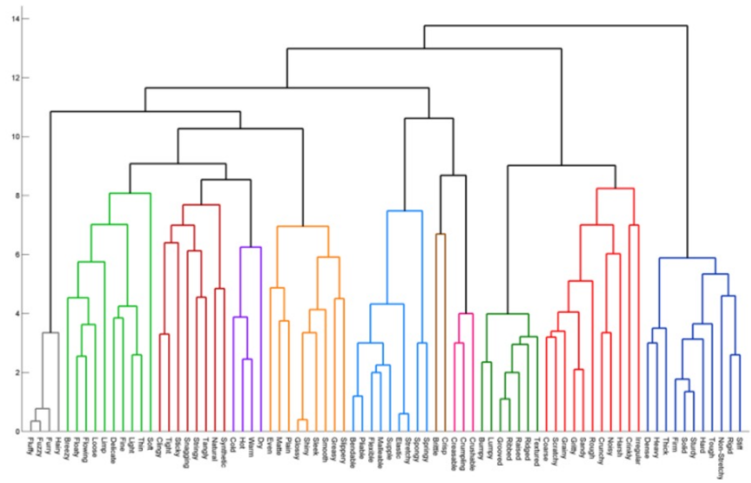
Text Clustering

Feature Vectors => Similarity, Distance between documents

- K-means algorithm



- Hierarchical Clustering – Dendrogram



Clustering evaluation with ground truth labels

- **Homogeneity:** quantifies how much clusters contain only members of a single class
- **Completeness**: quantifies how much members of a given class are assigned to the same clusters
- **V-measure:** harmonic mean of completeness and homogeneity
- **Rand-Index:** measures how frequently pairs of data points are grouped consistently
- **Adjusted Rand-Index**, a chance-adjusted Rand-Index such that a value close to 0.0 for random labeling independently of the number of clusters and samples and exactly 1.0 when the clusterings are identical

출처: https://scikit-learn.org/stable/modules/clustering.html#clustering-evaluation

Clustering evaluation without ground truth labels

- **Silhouette Coefficient** estimates how well samples are clustered with other samples that are similar to each other. Silhouette score is calculated for each sample of different clusters.
  - Mean distance (*a*) between the observation and all other data points in the same cluster. This distance can also be called as **mean intra-cluster distance**.
  - Mean distance (*b*) between the observation and all other data points of the next nearest cluster. This distance can also be called as **mean nearest-cluster distance**.

$$S = \frac{(b-a)}{max(a,b)}$$

The value of the score varies from -1 to 1.

1: the cluster is dense and well-separated than other clusters

0: overlapping clusters with samples very close to the decision boundary of the neighbouring clusters

a negative score [-1, 0] indicates that the samples might have got assigned to the wrong clusters.

Newsgroup clustering example

```
import numpy as np
from sklearn.datasets import fetch_20newsgroups

categories = [ "alt.atheism", "talk.religion.misc", "comp.graphics", "sci.space"]

dataset = fetch_20newsgroups(
    remove=("headers", "footers", "quotes"),
    subset="all",
    categories=categories,
    shuffle=True,
    random_state=42)

labels = dataset.target
unique_labels, category_sizes = np.unique(labels, return_counts=True)
true_k = unique_labels.shape[0]

print(f"{len(dataset.data)} documents - {true_k} categories")
```

- 데이터 로딩
- 4개의 카테고리, 3387 문서
- 출처: https://scikit-learn.org/stable/auto_examples/text/plot_document_clustering.html

```
3387 documents - 4 categories
```

```
from collections import defaultdict
from sklearn import metrics

def fit_and_evaluate(km, X, n_runs=5):
    scores = defaultdict(list)
    for seed in range(n_runs):
        km.set_params(random_state=seed)
        km.fit(X)
        scores["Homogeneity"].append(metrics.homogeneity_score(labels, km.labels_))
        scores["Completeness"].append(metrics.completeness_score(labels, km.labels_))
        scores["V-measure"].append(metrics.v_measure_score(labels, km.labels_))
        scores["Adjusted Rand-Index"].append(metrics.adjusted_rand_score(labels, km.labels_))
        scores["Silhouette Coefficient"].append(metrics.silhouette_score(X, km.labels_, sample_size=2000))

    for score_name, score_values in scores.items():
        mean_score, std_score = np.mean(score_values), np.std(score_values)
        print(f"{score_name}: {mean_score:.3f} ± {std_score:.3f}")

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_df=0.5, min_df=5, stop_words="english",)
X_tfidf = vectorizer.fit_transform(dataset.data)
kmeans = KMeans(n_clusters=true_k, max_iter=100, n_init=5)
fit_and_evaluate(kmeans, X_tfidf)
```

```
clustering done in 0.19 ± 0.05s
Homogeneity: 0.347 ± 0.009
Completeness: 0.397 ± 0.006
V-measure: 0.370 ± 0.007
Adjusted Rand-Index: 0.197 ± 0.014
Silhouette Coefficient: 0.007 ± 0.00
```

Summary

- Similarity between two words/sentences is associated with their distance
- Distance metrics
- Document recommendation
- Text clustering using K-means algorithm