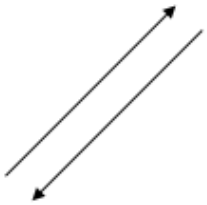
문서 유사도 (Document Similarity)

문서의 유사도의 성능은 **각 문서의 단어들을 어떤 방법으로 수치화하여 표현했는지(DTM, Word2Vec 등)**, **문서간의 단어들의 차이를 어떤 방법으로 계산했는지(유클리드, 코사인 유사도)**에 달려있음

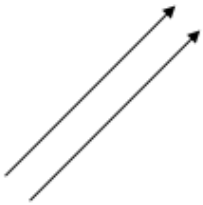
코사인 유사도 (Cosine Similarity)

- 코사인 유사도란 : **두 벡터 간의 코사인 각도를 이용하여 구할 수 있는 두 벡터의 유사도**를 의미.
- 코사인 유사도는 -1 ~ 1의 값을 가짐
  - 코사인 유사도의 값이 -1 : 두벡터의 방향이 $180°$로 반대의 방향을 가지는 경우
  - 코사인 유사도의 값이 0 : 두 벡터의 방향이 $90°$의 각을 이루는 경우
  - 코사인 유사도의 값이 1 : 두 벡터의 방향이 완전히 동일한 경우



코사인 유사도 : -1

코사인 유사도 : 0



코사인 유사도 : 1

- 코사인 유사도 식

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

  - A, B : 각각의 벡터를 의미함

- example
  문서1 : 저는 사과 좋아요
  문서2 : 저는 바나나 좋아요
  문서3 : 저는 바나나 좋아요 저는 바나나 좋아요

  문서 단어 행렬

-	바나나	사과	저는	좋아요
문서1	0	1	1	1
문서2	1	0	1	1
문서3	2	0	2	2

Numpy를 이용한 코사인 유사도 실습

```
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))
```

```
doc1=np.array([0,1,1,1])
doc2=np.array([1,0,1,1])
doc3=np.array([2,0,2,2])
```

```
print(cos_sim(doc1, doc2)) #문서1과 문서2의 코사인 유사도
print(cos_sim(doc1, doc3)) #문서1과 문서3의 코사인 유사도
print(cos_sim(doc2, doc3)) #문서2과 문서3의 코사인 유사도
```

```
0.67
0.67
1.00
```

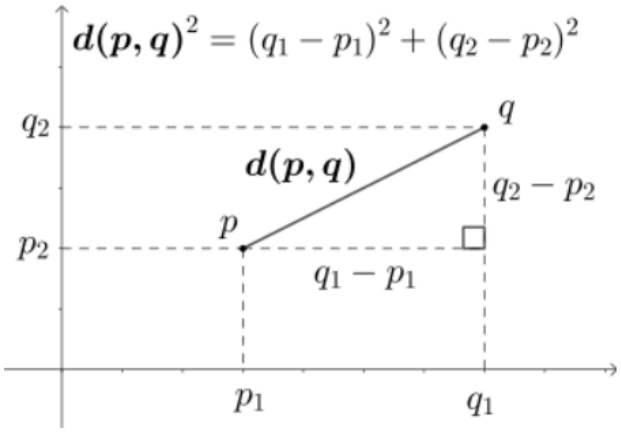
코사인 유사도는 두 벡터의 각도, 즉 두 벡터의 방향의 차이를 이용하여 유사함을 수치화한 것으로 문서2와 문서3의 코사인 유사도는 1이 나옴. 벡터의 크기는 중요하지 않음.

유클리드 거리 (Euclidean distance)

- Euclidean distance란 : **두 점 사이의 거리를 계산**(함으로서 유사도를 구함)
- Euclidean distance 식 :
  다차원 공간에서 두개의 점 p와 q가 각각 $p = (p_1, p_2, p_3, \ldots, p_n)$과 $q = (q_1, q_2, q_3, \ldots, q_n)$의 좌표를 가질 때, 두 점 사이의 거리 계산

$$\sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

  2차원 공간에서 두개의 점 p와 q 사이의 직선 거리를 구하는 예제



- Euclidean distance의 단점 : https://my-romance.github.io/ir/2020/03/01/12-The-Vector-Space-Model.html post 확인
- example (4차원 공간에서의 유클리드 거리 구하기):

문서 단어 행렬

-	바나나	사과	저는	좋아요
문서1	2	3	0	1
문서2	1	2	3	1
문서3	2	1	2	2

-	바나나	사과	저는	좋아요
문서Q	1	1	0	1

Euclidean distance 실습

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

```
doc1 = np.array((2,3,0,1))
doc2 = np.array((1,2,3,1))
doc3 = np.array((2,1,2,2))
docQ = np.array((1,1,0,1))
```

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

```
2.23606797749979
3.1622776601683795
2.449489742783178
```

Euclidean distance의 값이 작을수록 유사도가 큰 것. 즉, 문서 Q와 가장 유사한 것은 문서 1.

자카드 유사도 (Jaccard similarity)

- 자카드 유사도란 : **합집합에서 교집합의 비율을 구한다**면 두 집합 A와 B의 유사도를 구할 수 있음
- 자카드 유사도는 0~1의 값을 가짐
  - 자카드 유사도의 값 0 : 두 집합의 공통원소가 없는 경우
  - 자카드 유사도의 값 1 : 두 집합이 동일한 경우
- 자카드 유사도 수식 :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cup B|}{|A| + |B| - |A \cup B|}$$

- example ($J(doc1, doc2)$)

```
# 다음과 같은 두 개의 문서가 있습니다.
# 두 문서 모두에서 등장한 단어는 apple과 banana 2개.
doc1 = "apple banana everyone like likey watch card holder"
doc2 = "apple banana coupon passport love you"

# 토큰화를 수행합니다.
tokenized_doc1 = doc1.split()
tokenized_doc2 = doc2.split()

# 토큰화 결과 출력
print(tokenized_doc1)
print(tokenized_doc2)
```

```
['apple', 'banana', 'everyone', 'like', 'likey', 'watch', 'card', 'holder']
['apple', 'banana', 'coupon', 'passport', 'love', 'you']
```

```
union = set(tokenized_doc1).union(set(tokenized_doc2)) # 합집합 반환
print(union)
```

```
{'card', 'holder', 'passport', 'banana', 'apple', 'love', 'you', 'likey',
'coupon', 'like', 'watch', 'everyone'}
```

```
intersection = set(tokenized_doc1).intersection(set(tokenized_doc2)) # 교집합
반환
print(intersection)
```

```
{'banana', 'apple'}
```

```
print(len(intersection)/len(union)) # 2를 12로 나눔. 자카드 유사도이자, 두 문서
의 총 단어 집합에서 두 문서에서 공통적으로 등장한 단어의 비율
```

```
0.16666666666666666
```

출처

- https://wikidocs.net/24603
- https://wikidocs.net/24654