LC029 정보검색

2022 10/13

Chapter 6 : Scoring, Term Weighting, and the Vector Space Model

1~6주까지 배운 것

Problems with Boolean Search 불린 모델의 문제점

- Boolean queries often result in too many results.
  - This is particularly true of large document collections.
  - Most users don't want to wade through too many results.
- Boolean Search
  - Documents either match or don't.
  - Ranking the results is impossible. 검색 결과의 우선순위가 없음
  - It takes skill to write a query that produces a manageable number of results.

Basis of Ranked Retrieval : Scoring

검색 결과에 점수 매기기

- With a ranked list of documents it does not matter how large the retrieved set is.
  - We wish to return the documents in the order of usefulness.
- How can we rank the documents in the collection with respect to a query?
  - Assign a score, say in [0, 1], to each document. 점수는 보통 0~1 사이 실수
  - This score measures how well document and query match.

Query-Document Matching Scores

- We need a way of assigning a score to **(*query, document*)** pair.
- Let’s start with a one-term query.
  - If the query term does not occur in the document, the score should be 0.
  - The more frequent the query term in the document, the higher the score. 질의어가 자주 나올수록 높은 점수 주자
- We will look at a number of alternatives for this.

불린 모델에서 점수를 주는 방법

Scoring under Boolean Search Model

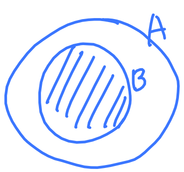
1. Jaccard Coefficient: Scoring
2. Weighted Zone Scoring

Jaccard Coefficient

자카드 계수

Jaccard Coefficient: Scoring

- Measurement of the overlap of two sets *A* and *B.*
  - Jaccard(A, B) = $|A \cap B| / |A \cup B|$
- Always assigns a number between *0* and *1*.
  - Jaccard(A, A) = 1
  - Jaccard(A, B) = 0 if $A \cap B = \emptyset$
- A and B don't have to be the same size.
- Used to measure the Query-Document Match Score



→ 자카드 계수 1보다 작음

Jaccard Coefficient: Scoring Example

- Example
  - Query: ides of march 3
  - Document 1: caesar died in march 4
  - Document 2: the long march 3

score(q, $d1$) = 1/(3 + 4 – 1) = 1/6 = 0.167

score(q, $d2$) = 1/(3 + 3 – 1) = 1/5 = 0.200

Jaccard Coefficient: Issues 자카드 계수의 문제

- Jaccard Coefficient doesn’t consider the following information: 아래 정보들을 반영하지 못함
  - **Term Frequency** → 이 문서 내에서 아주 중요한 단어다
    - Frequent terms in a document are more important than rare terms.
  - **Document Frequency** → 어디에나 나와서 별로 안 중요한 단어다
    - Rare terms in a collection are more informative than frequent terms.

Parametric and Zone Indexes

Parametric and Zone Indexes

- Most documents have additional structures. 문서 본문 이외 추가 정보
  - Digital documents encode, in machine-readable form, certain metadata, such as authors, title, abstract, date, etc.

zone
↓
검색 조건에
추가될 수 있음

Bibliographic Search

Search category	Value
Author	*Example:* Widom, J *or* Garcia-Molina
Title	Also a part of the title possible
Date of publication	*Example:* 1997 *or* <1997 *or* >1997 limits the search to the documents appeared in, before and after 1997 respectively
Language	Language the document was written in English
Project	ANY
Type	ANY
Subject group	ANY
Sorted by	Date of publication
	Start bibliographic search

Find document via ID

Parametric and Zone Indexes

- Query:
  Find documents written by ***William Shakespeare*** in ***1601***, containing the phrase ***alas poor Yorick***.
- We need:
  - **Inverted index** to find documents containing the phrase.
  - **Parametric index** to find documents published in 1601.
  - **Zone index** to find documents written by W. Shakespeare.

❖ **Parametric indexes** are required for the fields which assume a value from a finite set. 특정 범위로 제한된 값만 들어가는 인덱스

❖ **Zones** can contain arbitrary free text.

Parametric and Zone Indexes

- Basic Zone Index

william.abstract → ***11*** → ***121*** → ***144*** → ***172***

william.title → ***2*** → ***4*** → ***8*** → ***16***

william.author → ***2*** → ***3*** → ***5*** → ***8***

- Encoded Zone Index
  - Reduce the size of dictionary 사이즈↓ 대신 검색시간↑

william → ***2.author.title*** → ***3.author*** → ***4.title***

Weighted Zone Scoring

- WZS(***q***, ***d***) = $\sum_{i=1}^{l} g_i s_i$

  g_i : 가중치

  S_i : score

  모든 가중치의 합은 보통 1이 되게 맞춤

  where weights $g_1, g_2, \ldots, g_l \in [0,1]$ such that $\sum_{i=1}^{l} g_i = 1$

  and s_i is the **Boolean score** (1 for a match between query ***q*** and i-th zone of the document ***d***, otherwise 0)
- Example
  - Document has three zones: *author*, *title* and *body*.
  - Weights for each zone: g_1 = 0.2, g_2 = 0.3, g_3 = 0.5
  - Query: ***shakespeare***
  - WZS = 0.2 X 0 + 0.3 X 1 + 0.5 X 1 = 0.8
    when *title* and *body* zone include the term ***shakespeare***.

Learning Weights 좀더 객관적인 가중치 만들기

- Machine Learning 기계학습
  - We need training examples of the form (***q***, ***d***, ***answer***), where ***answer*** is either ***relevant*** or ***nonrelevant***.
  - The weights $\boldsymbol{g_i}$ are learned from these examples, in order that the learned score approximate the relevance judgments in the training examples.

    → optimization problem

Learning Weights

- Assume that each document has ***title*** and ***body*** zone.
- Given an example $\Phi_j = (d_j, q_j, \text{answer})$

  1 if relevant
  0 if nonrelevant

  - Weighted Zone Score is computed from:

    $$score(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1-g)s_B(d_j, q_j)$$

  - The error of the score is defined as:

오차

$$\varepsilon(g, \Phi_j) = (\text{ answer} - score(d_j, q_j))^2$$

  - The total error of a set of training examples is given by:

    $$\sum_j \varepsilon(g, \Phi_j)$$

- The problem of learning weights is reduced to a problem picking the value of *g* that minimizes the total error.

Learning Weights

- Training Examples

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Non-relevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Non-relevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3191	driver	1	0	Non-relevant

$$\sum_j \varepsilon(g, \Phi_j) = (1 - (0.5 \times 1 + 0.5 \times 1))^2 + (0 - (0.5 \times 0 + 0.5 \times 1))^2 + (1 - (0.5 \times 0 + 0.5 \times 1))^2 + (0 - (0.5 \times 0 + 0.5 \times 0))^2 + (1 - (0.5 \times 1 + 0.5 \times 1))^2 + (1 - (0.5 \times 0 + 0.5 \times 1))^2 + (0 - (0.5 \times 1 + 0.5 \times 0))^2 = 1.0$$

Learning Weights

- Training Examples

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Non-relevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Non-relevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3191	driver	1	0	Non-relevant

$$\sum_j \varepsilon(g, \Phi_j) = (1 - (0.6 \times 1 + 0.4 \times 1))^2 + (0 - (0.6 \times 0 + 0.4 \times 1))^2 + (1 - (0.6 \times 0 + 0.4 \times 1))^2 + (0 - (0.6 \times 0 + 0.4 \times 0))^2 + (1 - (0.6 \times 1 + 0.4 \times 1))^2 + (1 - (0.6 \times 0 + 0.4 \times 1))^2 + (0 - (0.6 \times 1 + 0.4 \times 0))^2 = 1.24$$

Learning Weights

- Training Examples

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Non-relevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Non-relevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3191	driver	1	0	Non-relevant

$$\sum_j \varepsilon(g, \Phi_j) = (1 - (0.3 \text{ X } 1 + 0.7 \text{ X } 1))^2 + (0 - (0.3 \text{ X } 0 + 0.7 \text{ X } 1))^2 + (1 - (0.3 \text{ X } 0 + 0.7 \text{ X } 1))^2 + (0 - (0.3 \text{ X } 0 + 0.7 \text{ X } 0))^2 + (1 - (0.3 \text{ X } 1 + 0.7 \text{ X } 1))^2 + (1 - (0.3 \text{ X } 0 + 0.7 \text{ X } 1))^2 + (0 - (0.3 \text{ X } 1 + 0.7 \text{ X } 0))^2 = 0.76$$

Vector Space Model

1. Term Frequency
2. Document Frequency
3. *tf-idf* Weight

→ Cosine Similarity

Term Frequency

Binary Term-Document Incidence Matrix

- Consider the **presence or absence** of a term in a doc:

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Dictionary of size V

Term-Document Count Matrix

- Consider the **number of occurrences** of a term in a doc:

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Each document is a vector in $\mathbb{N}^{|V|}$.

Dictionary of size V

Bag of Words Model

- Vector representation doesn't consider the ordering of words in a document. 벡터는 단어의 등장순서를 반영할 수 없다
  - ***John is quicker than Mary*** and ***Mary is quicker than John*** are represented with the same vectors.
- This is called the **bag of words model**. 그래도 rank를 사용하기 위해 감수한다.
  - In a sense, this is a step back: 불린 모델에 비하면 일 보 후퇴한 부분
    The **positional index** was able to distinguish these two documents.
  - For now, we will use the bag of words model.

Term Frequency *tf*

- The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- We will use term frequency when computing query-document match scores.
- But, ***raw term frequency*** is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with one occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase *proportionally* with term frequency.
  - We use log $tf_{t,d}$ instead of $tf_{t,d}$ to dampen the effect of tf.

Document Frequency

Document Frequency *df*

- Terms rare in the collection are more informative than frequent terms.
  - Recall stop words
- Consider a term in the query that is rare in the collection
  - e.g. *arachnocentric*
- A document containing this term is very likely to be relevant to the query ***arachnocentric***.
  - We want a high weight for rare terms like *arachnocentric*.

Document Frequency *df*

- Consider query terms that are frequent in the collection, *e.g. high, increase, line, …*
  - A document containing such terms is more likely to be relevant than a document that doesn't, but they are not a strong indicator of relevance.
  - For frequent terms 너무 흔한 단어는 양수 가중치를 주되 다른 가치있는 단어보다 낮게 주고싶다.
    - we want positive weights
    - but lower weights than for rare terms
- Use ***document frequency*** to capture this in the score.
  - 모든 문서 수
  - *df* ($\leq N$) is the number of documents that contain the term.
  - *df* is a measure of the informativeness of the term.

Inverse Document Frequency *idf*

- Let df_t be the document frequency of term *t.*
- Inverse document frequency of term *t* is defined:

$$idf_t = \log N/df_t$$

where *N* is the total number of documents in a collection.

  - We use log N/df_t instead of N/df_t to dampen the effect of *idf*.

Inverse Document Frequency *idf*

- Suppose N = 1,000,000.

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

There is one *idf* value for each term t in a collection.

tf-idf Weight

Term Frequency Weight ?

- Relevance does not increase *proportionally* with *tf*.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Each document is a vector in $\mathbb{N}^{|V|}$.

Dictionary of size V

Log-Frequency Weight

- The log frequency **weight of term** *t* in *d* is

$$w_{t,d} = \begin{cases} 1 + \log tf_{t,d} & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Example

  $tf_{t,d} = 0 \rightarrow w_{t,d} = 0.0$
  $tf_{t,d} = 1 \rightarrow w_{t,d} = 1.0$
  $tf_{t,d} = 2 \rightarrow w_{t,d} = 1.3$
  $tf_{t,d} = 10 \rightarrow w_{t,d} = 2.0$
  $tf_{t,d} = 100 \rightarrow w_{t,d} = 3.0$

Log-Frequency Weight

- Log-frequency weight dampens the effect of *tf*.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	3.19	2.86	0	0	0	1
Brutus	1.6	3.19	0	1	0	0
Caesar	3.37	3.36	0	1.3	1	1
Calpurnia	0	2	0	0	0	0
Cleopatra	2.76	0	0	0	0	0
mercy	1.3	0	1.48	1.7	1.7	1
worser	1.3	0	1	1	1	0

Each document is a vector in $\mathbb{N}^{|V|}$.

Dictionary of size V

tf-idf Weight

- The *tf-idf* weight of a term is the product of *tf* and *idf*.

$$w_{t,d} = \underbrace{(1+\log tf_{t,d})}_{tf} \times \underbrace{\log N/df_t}_{idf} \quad \textit{if } tf_{t,d} > 0 \textit{ and } df_t > 0.$$

- Best known weight scheme in information retrieval.
  - Increases with the number of occurrences within a document.
  - Increases with the rarity of the term in the collection.

tf-idf Weight

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	0.96	0.86	0	0	0	0.3
Brutus	0.48	0.96	0	0.3	0	0
Caesar	0.27	0.27	0	0.1	0.08	0.08
Calpurnia	0	1.56	0	0	0	0
Cleopatra	2.15	0	0	0	0	0
mercy	0.1	0	0.12	0.14	0.14	0.08
worser	0.23	0	0.18	0.18	0.18	0

Each document is now represented by
a real-valued vector of *tf-idf* weights $\in R^{|V|}$.

Dictionary of size V

Vector Space Model

- Vector Representation for both Document and Query
  - Vector Representation of Documents

Antony	Brutus	Caeser	Carpurnia	Cleopatra	Worser

< 1 , 1 , 1 , 0 , 1 , 1 >
< 157 , 4 , 232 , 0 , 57 , 2 >
< 0.96 , 0.48 , 0.27 , 0 , 2.15 , 0.37 >

  - Vector Representation of Query

query : Caeser Cleopatra

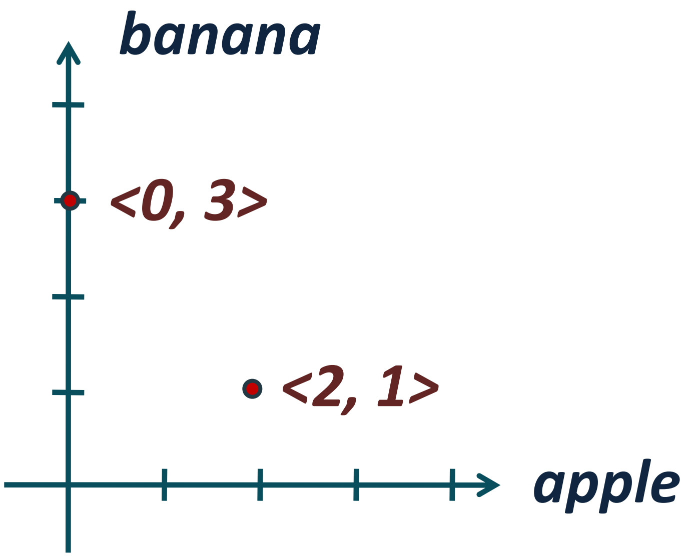
Antony	Brutus	Caeser	Carpurnia	Cleopatra	Worser

< 0 , 0 , 1 , 0 , 1 , 0 >

Vector Space Model

Documents as Vectors

- We have a $|V|$-dimensional vector space.
- Terms are axes of the space.
- Documents are points or vectors in this space.



Documents as Vectors

- Very high-dimensional
  - Hundreds of millions of dimensions when you apply this to a web search engine.

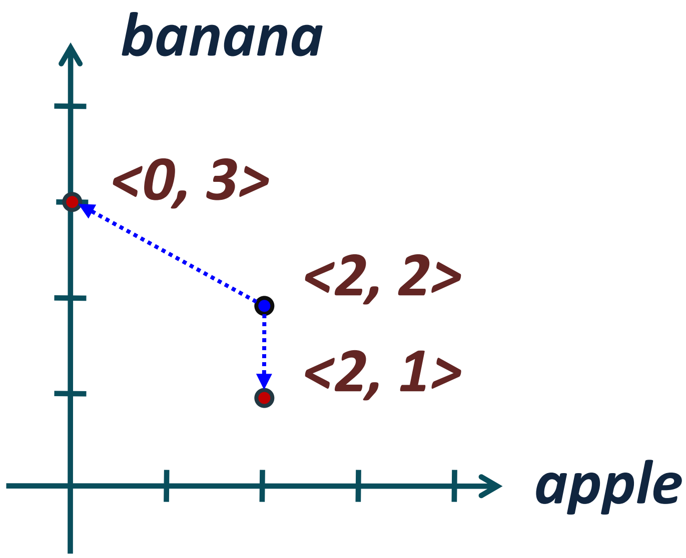
0이 많다

- This is a very sparse vector.
  - Most entries of the vector are zero.

Antony	***Brutus***	***Caeser***	***Carpurnia***	***Cleopatra***	***Worser***
< 5.25 ,	***1.21 ,***	***8.59 ,***	***0 ,***	***2.85 ,***	***1.37 >***

Queries as Vectors

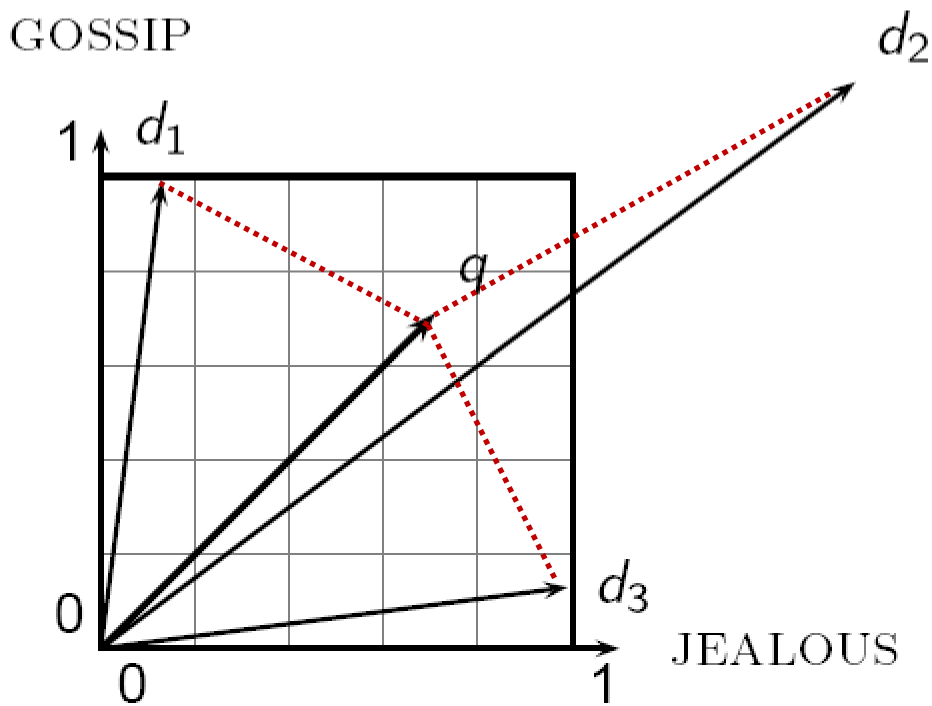
- Represent queries as vectors in the space.
- Rank documents according to their **similarity** to the query.
- Similarity could be given by inverse of Euclidean distance.



Similarity Measure

Euclidean distance is a bad idea

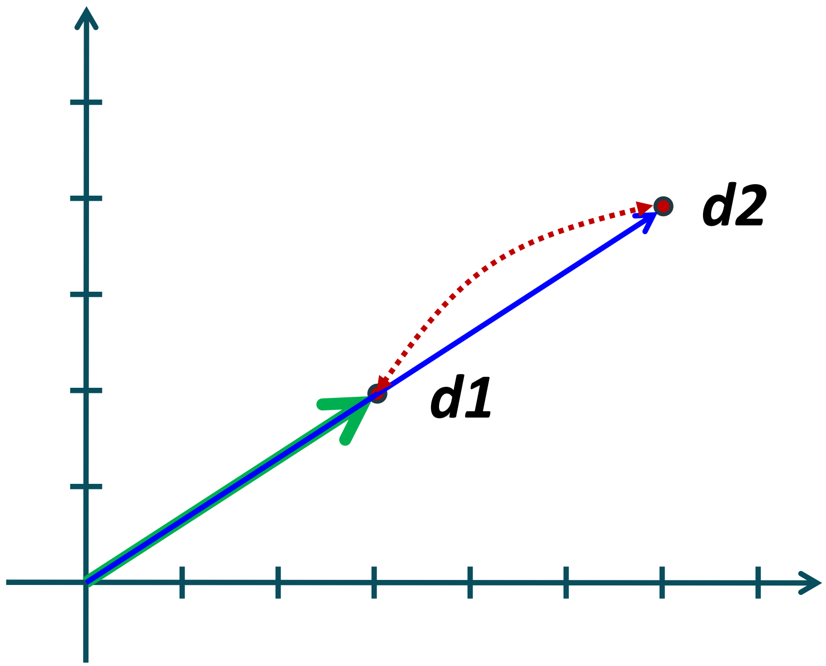
- The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Angle is used instead of distance

- Take a document ***d*** and append it to itself to make a new document ***d'***.
- Semantically ***d*** and ***d'*** have the same content.
- The Euclidean distance between the two documents could be quite large.
- The angle between ***d*** and ***d'*** is 0, corresponding to maximal similarity.
- Conclusion:
  Rank documents according to the angle with query.

Angle is used instead of distance



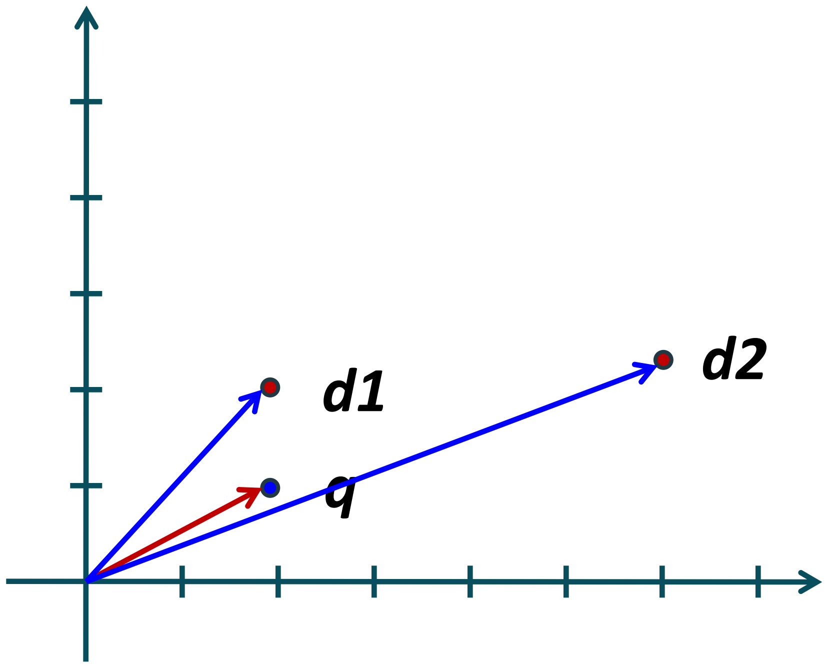
Take a document ***d1*** and append it to itself to make a new document ***d2***.

d1 = <3, 2>

d2 = <6, 4>

Angle is used instead of distance

- Which document is similar to a query ***q***?



From Angles to Cosines

- The following two notions are equivalent.
  - Rank documents in **increasing** order of the angle between query and document.
  - Rank documents in **decreasing** order of ***cos*(*q*, *d*)**.
- Thus, we will use **cosine** to measure the similarity of query and document.
  - **cosine** is a monotonically decreasing function for the interval [0º, 90º].
    단조 감소

Cosine Similarity : *cos*(*q*, *d*)

Dot Product 내적

길이가 항상 1

Unit Vectors

단위벡터

Dot Product

내적 계산을 식으로 정리한 것

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\cdot\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

벡터의 길이

벡터의 길이로 모든성분을 나눔

q_i is the *tf-idf* weight of term *i* in the query.

d_i is the *tf-idf* weight of term *i* in the document.

$\cos(\vec{q}, \vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$,
or equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

벡터 내적 : 같은 자리의 성분끼리 곱하고 모두 더함

ex) $\langle 1, 0, 2, 4 \rangle$
$\langle 4, 1, 0, 5 \rangle$ → $4+0+0+20=24$

Length Normalization

- A vector is normalized by dividing each of its components by its length.
  - The resulting vector is a unit vector of same direction.
  - Example
    Given vector <3, 4, 5>
    Length of the vector = $\sqrt{3^2 + 4^2 + 5^2} = \sqrt{50} = 7.07$
    Unit vector = <3/7.07, 4/7.07, 5/7.07> = <0.42, 0.57, 0.71>

Length Normalization

- Effect on the two documents ***d*** and ***d'*** (***d*** appended to itself).
  - They have identical vectors after length normalization.

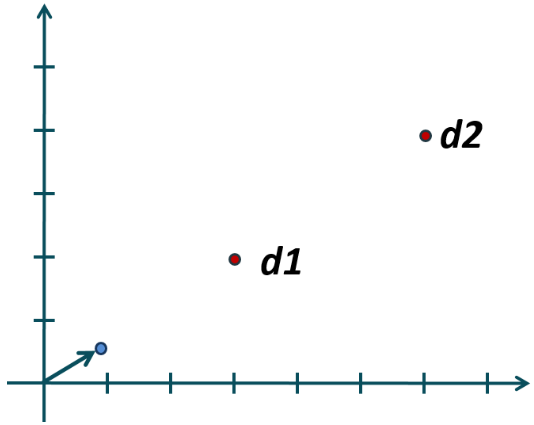
$$
\begin{aligned}
d &= \; < 3,4,5 > \\
|d| &= \sqrt{3^2+4^2+5^2} = 5\sqrt{2} \\
\frac{d}{|d|} &= \; < \frac{3}{5\sqrt{2}}, \frac{4}{5\sqrt{2}}, \frac{5}{5\sqrt{2}} >
\end{aligned}
$$

$$
\begin{aligned}
d' &= \; < 6,8,10 > \\
|d'| &= \sqrt{6^2+8^2+10^2} = 10\sqrt{2} \\
\frac{d'}{|d'|} &= \; < \frac{6}{10\sqrt{2}}, \frac{8}{10\sqrt{2}}, \frac{10}{10\sqrt{2}} > \\
&= \; < \frac{3}{5\sqrt{2}}, \frac{4}{5\sqrt{2}}, \frac{5}{5\sqrt{2}} >
\end{aligned}
$$

Cosine Similarity : Example

다 근사값이니 계산기 계산과 달라도 그런갑다 해라

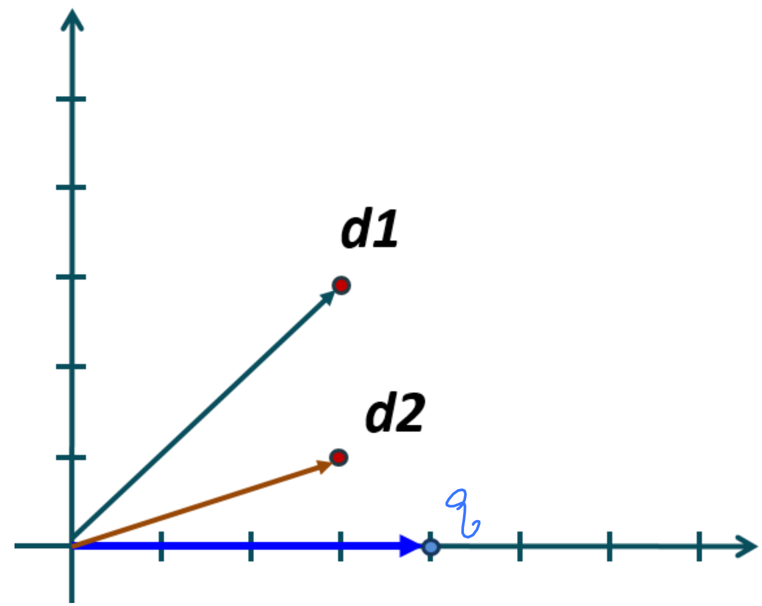
- d1 = <3, 2> d1 / |d1| = <0.83, 0.55>
  d2 = <6, 4> d2 / |d2| = <0.83, 0.55>



- $d1 \cdot d2 = 3 \times 6 + 2 \times 4 = 26$
- $|d1| = \sqrt{3^2 + 2^2} = 3.6$
  $|d2| = \sqrt{6^2 + 4^2} = 7.2$
- $\dfrac{d1 \cdot d2}{|d1| \times |d2|} = 26 / (3.6 \times 7.2) = 26 / 26 = 1.0$
- $\dfrac{d1}{|d1|} \times \dfrac{d2}{|d2|} = 0.83 \times 0.83 + 0.55 \times 0.55 = 0.7 + 0.3 = 1.0$

Cosine Similarity : Example

- q = <4, 0>, d1 = <3, 3>, d2 = <3, 1>
- cos(q, d1) = (4x3+0x3) /($\sqrt{4^2 + 0^2}$ x $\sqrt{3^2 + 3^2}$)
  = 12 / (4 x 3$\sqrt{2}$) = 1/$\sqrt{2}$ = cos(45)
- cos(q, d2) = (4x3+0x1) /($\sqrt{4^2 + 0^2}$ x $\sqrt{3^2 + 1^2}$)
  = 12 / (4 x $\sqrt{10}$) = 3/$\sqrt{10}$



Cosine Similarity : Example

- How similar are the novels?
  - SaS: *Sense and Sensibility*
  - PaP: *Pride and Prejudice*
  - WH: *Wuthering Heights*?

코사인 유사도는
문서와 문서 사이의
유사도 검사도 가능

허점 : 단어의 순서를 고려하지 못함
↓
같은 단어 다른 의미 고려 불가
구성

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Cosine Similarity : Example

Term frequencies

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

Log Frequency Weight : $w_{t,d} = 1 + \log tf_{t,d}$

$w_{affection,SaS} = 1 + \log 115 = 1 + 2.06 = 3.06$

$w_{jealous,SaS} = 1 + \log 10 = 1 + 1 = 2.0$

Cosine Similarity : Example

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

Nomalization : $\vec{d} \,/\, |\vec{d}|$

$\vec{d}$ = <3.06, 2.0, 1.3, 0.0>, $|\vec{d}|$ = 3.88

d / |d| = <0.789, 0.515, 0.335, 0.0>

Cosine Similarity : Example

After normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) $\approx 0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0$

≈ 0.94 ← 두 문서가 많이 유사하다

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\cdot\frac{\vec{d}}{|\vec{d}|}$$