Ranking and Topic Modeling

Jaegul Choo (주재걸)

고려대학교 컴퓨터학과

https://sites.google.com/site/jaegulchoo/

jchoo@korea.ac.kr

Overview

- Indexing documents and document retrieval
  - Term-document matrix

- Two approaches
  - Rank-independent evaluation (binary evaluation)
  - Rank-dependent evaluation

Term-Document Matrix

- Text is basically a sequence of words.
- Bag-of-words approach: Ignores the sequence info
  - Document 1 = “John likes movies. Mary likes too.”
  - Document 2 = “John also likes football.”
- Approaches considering sequence info
  - Hidden Markov models
  - Conditional random field
  - Recurrent neural network (RNN) / long short-term memory (LSTM)

Vocabulary	Doc 1	Doc 2
John	1	1
likes	2	1
movies	0	0
also	1	1
football	0	1
Mary	1	0
too	1	0

How to Search Documents?

1 if play contains word, 0 otherwise

Term-document incidence matrix

	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

For boolean query “Brutus AND Caesar AND NOT Calpurnia"

- 110100 AND 110111 AND 101111 = 100100

Issues of term-document incidence matrix

Building such an incidence matrix is infeasible for large collections.

For example,

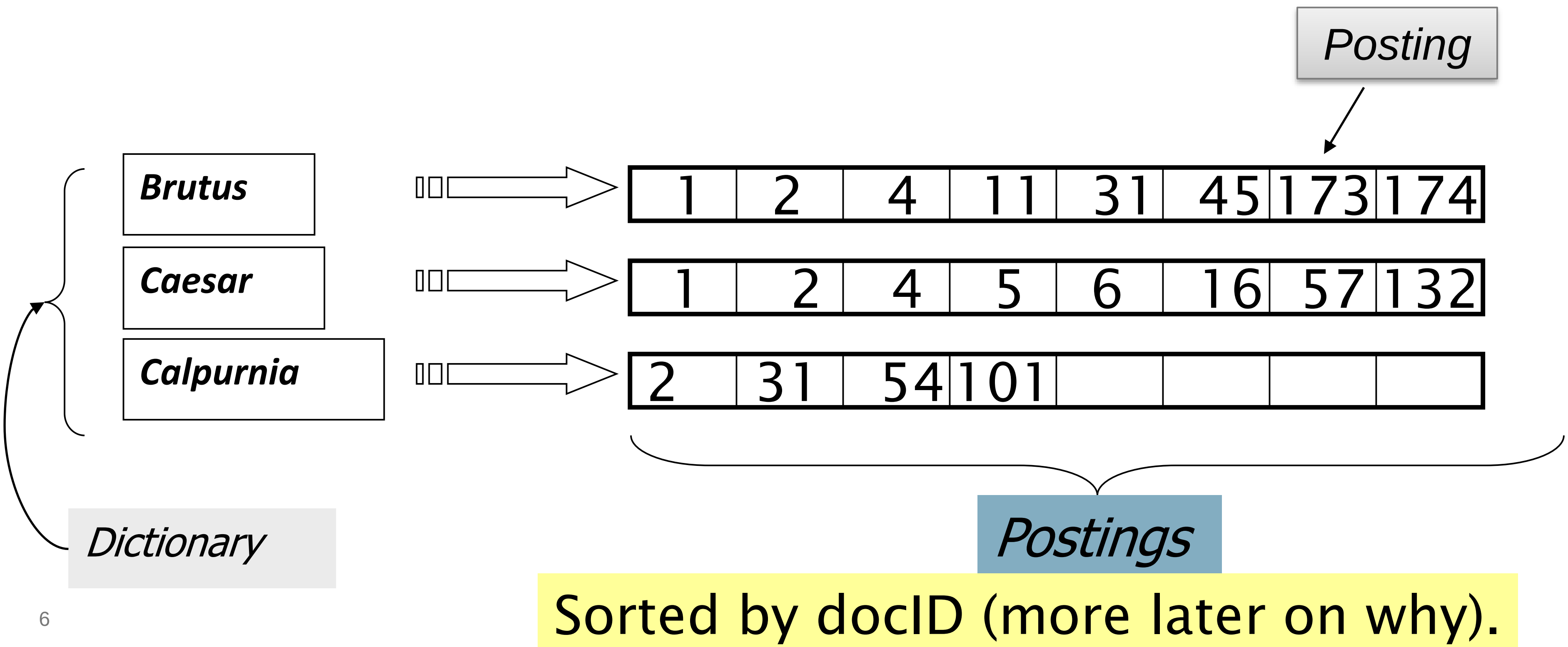
- Consider 1 million documents, each with about 1,000 words.
- Assume 6 bytes/word, storing all documents consumes 6GB space.
- Assume 500K distinct terms
- 500K x 1M matrix has half-a-trillion 0's and 1's.
- There should be roughly one billion 1's only. The incidence matrix is extremely sparse.

Is there a better solution?

Inverted Index

Inverted Index

- An inverted index consists of a dictionary and postings.
- For each term *T* in the dictionary, we store a list of documents (document IDs) containing *T.*



Indexer steps: Token sequence

- Sequence of (Modified token, Document ID) pairs.

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Indexer steps: Sort

- Sort by terms
  - And then docID

Core indexing step

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

Why frequency?
Will discuss later.

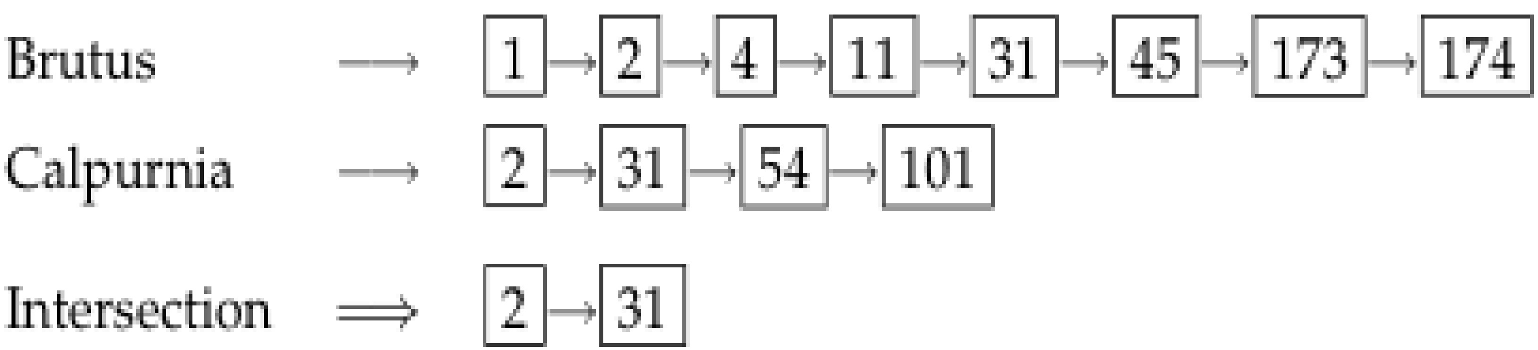
Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Processing Boolean queries

What are the documents contains “Brutus AND Calpurnia"?

- Locate Brutus in the Dictionary
- Retrieve its postings
- Locate Calpurnia in the Dictionary
- Retrieve its postings
- Intersect the two postings lists



- If the list lengths are *x* and *y*, the merge takes O(*x*+*y*) operations.

Important: postings sorted by docID.

Intersecting two postings lists (a “merge” algorithm)

```
INTERSECT(p1, p2)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then ADD(answer, docID(p1))
           p1 ← next(p1)
           p2 ← next(p2)
      else if docID(p1) < docID(p2)
              then p1 ← next(p1)
              else p2 ← next(p2)
return answer
```

Phrase queries

- We want to be able to answer queries such as ***“stanford university”*** – as a phrase
- Thus the sentence *“I went to a university at Stanford”* is not a match.
  - The concept of phrase queries has proven easily understood by users; one of the few “advanced search” ideas that works
  - Many more queries are *implicit phrase queries*
- For this, it no longer suffices to store only

  <*term* : *docs*> entries

A first attempt: Biword indexes

- Index every consecutive pair of terms in the text as a phrase
- For example the text "Friends, Romans, Countrymen" would generate the biwords
  - ***friends romans***
  - ***romans countrymen***
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

A first attempt: Biword indexes

- For multi-word phrase query such as "A B C D", we may use Boolean query "A B" AND "B C" AND "C D“
  - False positive: "A B B C C D" is a match
- larger index size (single-word + Bi-word dictionary)

Solution 2: Positional indexes

- In the postings, store, for each ***term*** the position(s) in which tokens of it appear:

<***term**,* number of docs containing ***term***;

doc1: position1, position2 … ;

doc2: position1, position2 … ;

etc.>

Positional index example

<***be***: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>

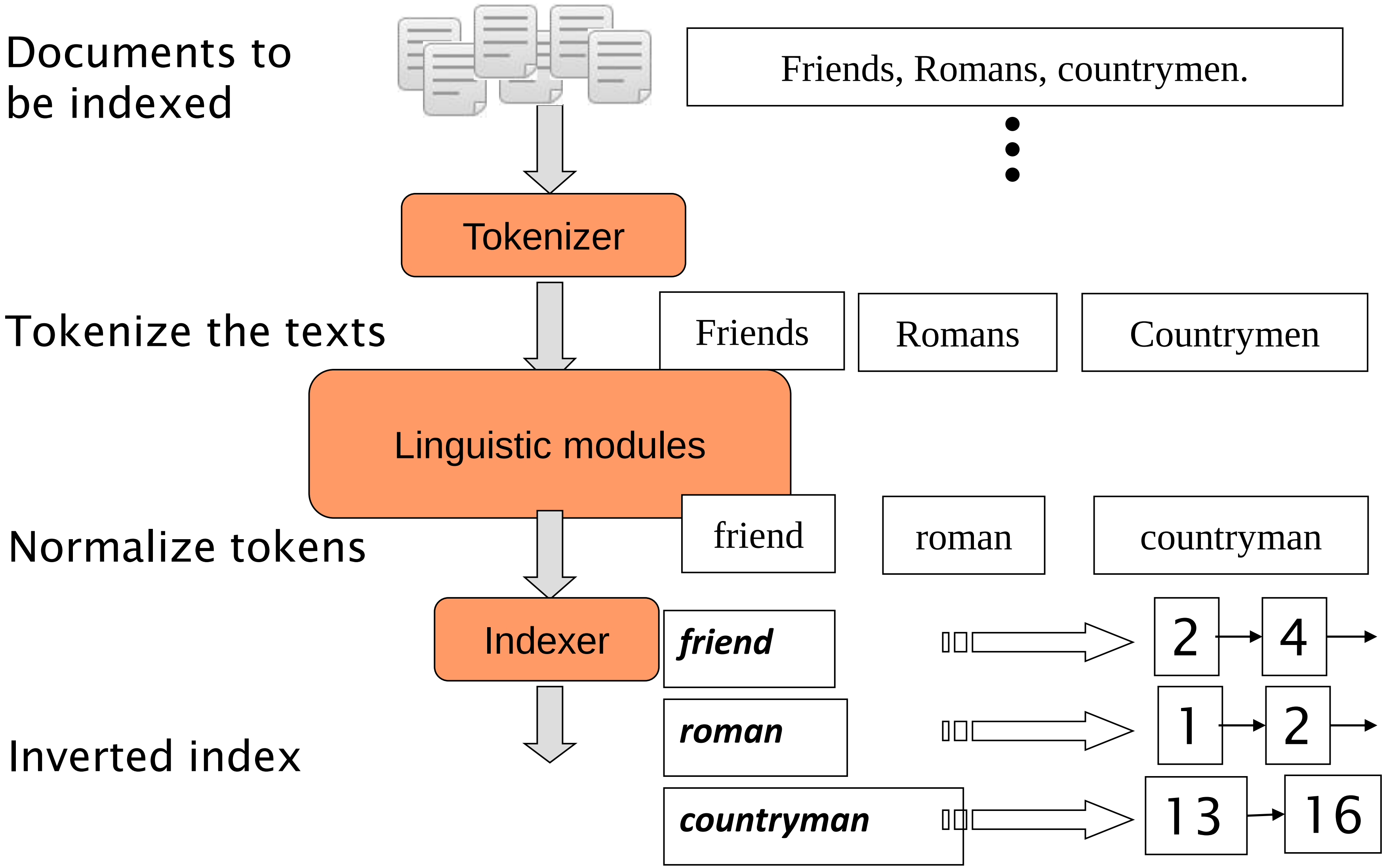
Which of docs 1,2,4,5 could contain "***to be or not to be***"?

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

Processing a phrase query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with "***to be or not to be***".
  - ***to****:*
    - *2*:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
  - ***be****:*
    - *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; ...
- Same general method for proximity searches

Inverted Index Construction



Tokenization

- Cut character sequence into word tokens
  - Complications:
    - "Hewlett-Packard" -> "Hewlett" and "Packard" as two separate tokens?
    - “State-of-the-art”
- Numbers often are not indexed as text
  - Dates: 2/12/91, Mar.12, 1991, 55 B.C.
  - My PGP key is 324a3df234cb23e
  - IP addresses: 100.2.86.144
  - Tracking numbers: 1Z9999W99845399981

Normalization

- We usually need terms in indexed texts as well as in queries in the same form
  - We want to match U.S.A. and USA.

- Difficulties in handling different languages
  - Accents: résumé vs. resume
  - Most important criterion:
    - How are your users likely to write their queries for these words?
  - Even in languages where the accents are common, users often do not type them, or the input device is not convenient
  - – E.g., in German: Tuebingen vs. Tübingen should be the same
- Difficulties in handling numbers
  - Dates: July 30 vs. 7/30

Normalization: Case Folding

- Often best to lower case everything, since users tend to use lowercase regardless of the correct capitalization
- However, many proper nouns are derived from common nouns
  - General Motors, Associated Press

Stop Words

- Stop-words: the most common words in the dictionary
  - They have little semantic content: the, a, and, to
  - They take a lot of space

- Now less use of stop words:
  - Good compression techniques
  - Good query optimization techniques mean one pays little at query time for including stop words
  - They are necessary in some settings:
    - Phrase queries: "King of Prussia“
    - (Song) titles : "Let it be", "To be or not to be“
    - Relational queries: "flights to London" vs. "flights from London"

Stemming

- Words in different forms may have similar meaning
  - Grammatical reasons: organize, organizes, organizing
  - Derivationally-related: democracy, democratic, democratization
- Stem the words to have one form
  - Reduce terms to their "roots" before indexing
  - "Stemming" suggest crude suffix chopping
    - language dependent
    - e.g., automate(s), automatic, automation -> automat

Porter's Stemmer

- The most common algorithm for stemming English

Typical rules

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

- Weight of word sensitive rules
- *(m>1) EMENT* →
  - *replacement* → *replac*
  - *cement* → *cement*

Examples of Porter Stemmer

False positives	*False negatives*
organization/organ	european/europe
generalization/generic	cylinder/cylindrical
numerical/numerous	matrices/matrix
policy/police	urgency/urgent
university/universe	create/creation
addition/additive	analysis/analyses
negligible/negligent	useful/usefully
execute/executive	noise/noisy
past/paste	decompose/decomposition
ignore/ignorant	sparse/sparsity
special/specialized	resolve/resolution
head/heading	triangle/triangular

- False positives: Incorrectly produced pairs of words that have the same stem. (잘못하여 다른 단어를 같게)
- False negatives: Pairs of words with the same stems, but are incorrectly stemmed into different stems. (잘못하여 같은 단어를 다르게)

Other stemmers

- Other stemmers exist:
  - Lovins stemmer
    - http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
  - Paice/Husk stemmer
  - Snowball

Summary So Far

- Term-document incidence matrix
- Inverted index
- “Merge” algorithm for processing Boolean queries
- Phrase queries
  - Biword indexing
  - Positional indexing
- Inverted index construction
  - Tokenization
  - Normalization
  - Stop words
  - Stemming

Problem of Boolean Queries

- So far, our queries have all been Boolean.
- What is the problem of this model?

Ranked Retrieval Models

- Ranking of the documents is important.
  - We wish to return in order the documents most likely to be useful/relevant to the searcher.

- Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering of the retrieved documents in the collection for a query

- Main idea: Let’s put a higher rank on those documents containing more query terms.

  -> Bag-of-words model or term-document matrix

Recall: (Binary) Term-Document Incidence Matrix

	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 $\in \{0,1\}^{|V|}$

Now, Term-Document (COUNT) Matrix

- Consider the number of occurrences of a term in a document
  - Each document is a count vector in *V*-dimensional space: a column vector shown below, where *V* is the dictionary size.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
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

Another Example of Term-Document Matrix

D_1 Tropical Freshwater Aquarium Fish.

D_2 Tropical Fish, Aquarium Care, Tank Setup.

D_3 Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.

D_4 The Tropical Tank Homepage - Tropical Fish and Aquariums.

Terms	Documents			
	D_1	D_2	D_3	D_4
aquarium	1	1	1	1
bowl	0	0	1	0
care	0	1	0	0
fish	1	1	2	1
freshwater	1	0	0	0
goldfish	0	0	1	0
homepage	0	0	0	1
keep	0	0	1	0
setup	0	1	0	0
tank	0	1	0	1
tropical	1	1	1	2

Bag-of-Words Model

The (column) vector representation of each document in a term-document matrix is called a bag-of-words model.

- *John is quicker than Mary* and *Mary is quicker than John* have the same vector representations.
- In a sense, this is a step back: The positional index was able to distinguish these two documents, but the bag-of-words model does not.

	Antony and Cleopatra	**Julius Caesar**	**The Tempest**	**Hamlet**	**Othello**	**Macbeth**
Antony	**157**	**73**	**0**	**0**	**0**	**0**
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**

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 want to use tf when computing query-document match scores.
  - We put a higher rankings on documents containing a larger tf of query words.

- However, the raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Log-Frequency Weighting on tf

- The log frequency weight $w_{t,d}$ of term *t* in *d* is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
  - score $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$
- The score is 0 if none of the query terms is present in the document.

Another Issue of Term Frequency

- Log-frequency weighting handles the issue of the frequency not being proportional to the relevance within a single term
- How about the relevance between different terms?
- e.g., Which of these tells you more about a document?
  - Suppose our query is “handsome person”
  - 10 occurrences of “handsome”?
  - 10 occurrences of “person”?

- Would like to attenuate the weights of common terms
  - But what is "common"?

Document Frequency

- Rare terms are more informative than frequent terms
  - Recall stop words
- Let's put more weights on rare terms.
  - e.g., *"handsome"* is rarer than *"person".*
- df_t (the document frequency of the term *t*): the number of documents that contain *t*
  - df_t is an inverse measure of the informativeness of *t*
  - $\mathrm{df}_t \leq N$ (total number of documents)

Inverse Document Frequency (idf)

- idf_t (the inverse document frequency of the term t):
  - Measure of informativeness of a term: its rarity across the whole corpus (많은 문서가 그 색인어를 가지고 있을수록, 그 색인어는 문서를 구분하는 변별력이 떨어진다.)
  - A simple definition could be $\mathrm{idf}_t = 1 / \mathrm{df}_t$

- However, the most commonly used definition is:
  - $\mathrm{idf}_t = \log_{10} (N / \mathrm{df}_t)$ where N is the total number of documents.
  - Why logarithm?
    - idf_t values will have more differences for small df_t values compared to large df_t values. However, such differences will become small as df_t gets bigger.
    - 적은 df_t 값의 차이가 idf_t 값에서 확실하게 차이가 나도록, 그러나 df_t 값이 어느 정도 커지면 적은 df_t 값의 차이는 별로 idf_t 값이 차이가 나지 않도록.

Effect of idf on Ranking

- Does idf have an effect on ranking for one-term queries?
  - e.g., the query “iPhone”
- idf has no effect on ranking one term queries
  - idf affects the ranking of documents for queries with at least two terms
  - For the query “handsome person”, idf weighting makes occurrences of handsome count for much more in the final document ranking than occurrences of person.

tf-idf Weighting

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

$$\mathrm{w}_{t,d} = \log(1+\mathrm{tf}_{t,d}) \times \log_{10}(N/\mathrm{df}_t)$$

- Best known weighting scheme in information retrieval
  - Note: the "-" in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Score for a Document given a Query

$$\mathrm{Score}(q,d) = \sum_{t \in q \cap d} \mathrm{tf-idf}_{t,d}$$

- There are many variants
  - How “tf” is computed (with/without logs)
  - Whether the terms in the query are also weighted
  - …

Binary → Count → Weight Matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

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

Documents as Vectors

- So we have a |V|-dimensional vector space
- Terms are axes of the space
- Documents are points or vectors in this space
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero.

Queries as Vectors

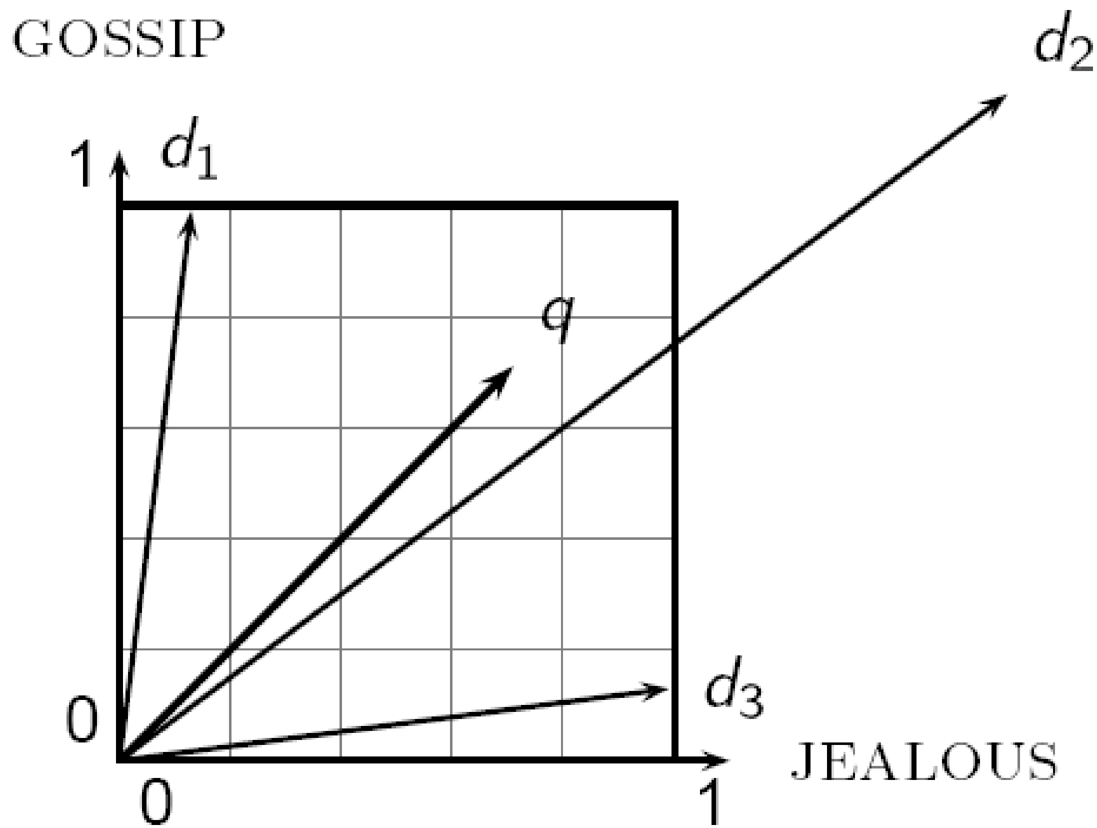
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the "you're-either-in-or-out" Boolean model.
- Instead: rank more relevant documents higher than less relevant documents

Formalizing Vector Space Proximity

- First approach: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .
- . . . because Euclidean distance is large for vectors of different lengths.

Why Distance is a Bad Idea

The Euclidean distance between *q* and d_2 is large even though the distribution of terms in the query *q* and the distribution of terms in the document d_2 are very similar.



Use Angle instead of Distance

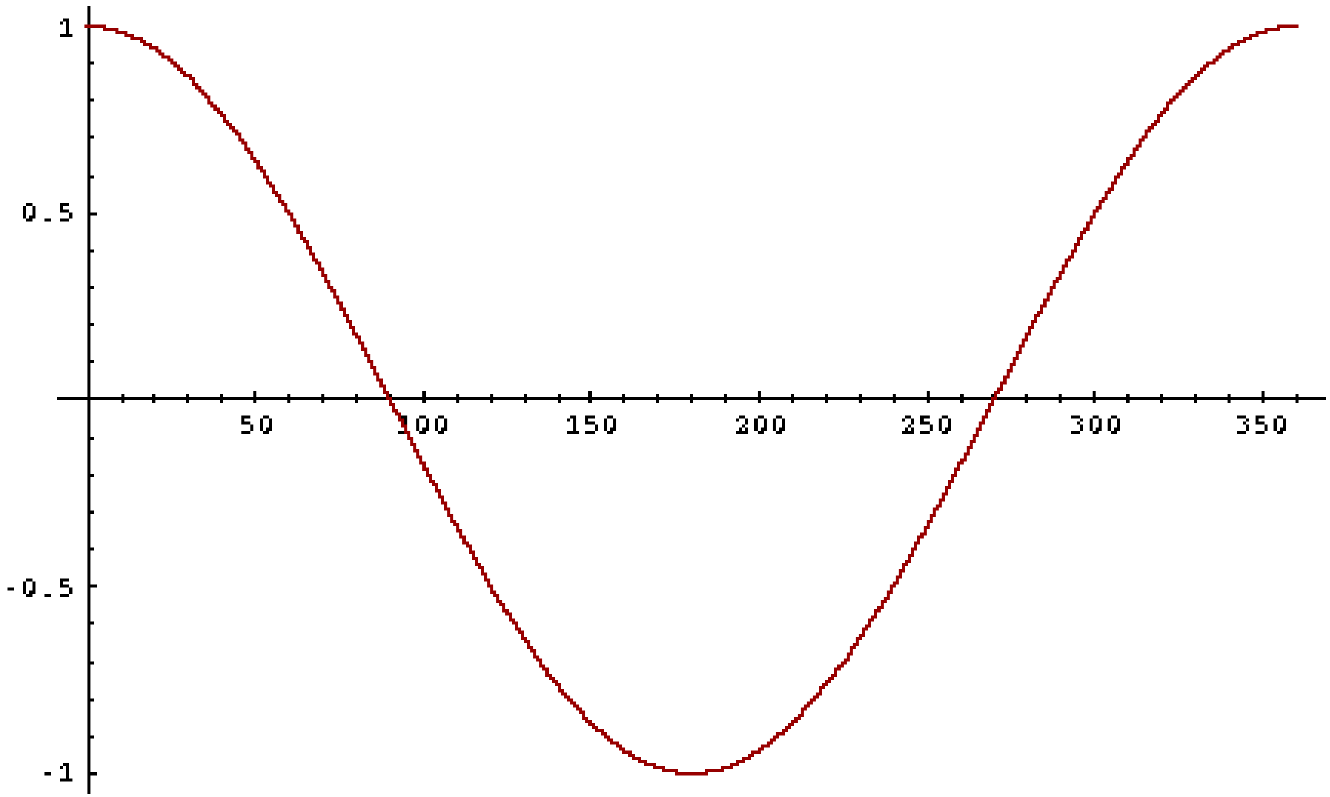
- Thought experiment: take a document *d* and append it to itself. Call this document *d*′.
- “Semantically” *d* and *d*′ have the same content
- The Euclidean distance between the two documents can be quite large.
- The angle between the two documents is 0, corresponding to maximal similarity.

- Key idea: Rank documents according to angle with query.

From Angles to Cosines

- The following two notions are equivalent.
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order of cosine(query, document)
- Cosine is a monotonically decreasing function in the interval $[0^o, 180^o]$

From Angles to Cosines



- But how – *and why* – should we be computing cosines?

Length Normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\left\|\vec{x}\right\|_2 = \sqrt{\sum_i x_i^2}$$

- Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)
- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

Cosine (query, document)

Dot product

Unit vectors

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

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 $\vec{d}$.

Cosine for Length-Normalized Vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

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

for q, d length-normalized.

Cosine Similarity among 3 Documents

How similar are the novels

SaS: *Sense and Sensibility*

PaP: *Pride and Prejudice*, and

WH: *Wuthering Heights*?

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

Term frequencies (counts)

Note: To simplify this example, we don’t do idf weighting.

Cosine Similarity among 3 Documents

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 length-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) ≈
$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0$
≈ 0.94

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

Computing Cosine Scores

CosineScore(q)

float Scores$[N] = 0$
float Length$[N]$
for each query term t
do calculate $w_{t,q}$ and fetch postings list for t
    for each pair$(d, \mathrm{tf}_{t,d})$ in postings list
    do $Scores[d] += w_{t,d} \times w_{t,q}$
Read the array *Length*
for each d
do $Scores[d] = Scores[d] / Length[d]$
return Top K components of *Scores*[]

Summary – Vector Space Ranking

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity score for the query vector and each document vector
- Rank documents with respect to the query by score
- Return the top K (e.g., $K = 10$) to the user

Evaluation

- How can we evaluate search results?
- Two approaches
  - Rank-independent evaluation (binary evaluation)
  - Rank-dependent evaluation

Measures for a Search Engine

- How fast does it index?
  - Number of documents / hour
- How fast does it search?
  - Latency as a function of index size
- How frequent is the index refreshed
  - Recency issues → Twitter
- Expressiveness of query language
  - Ability to express complex information needs
  - Speed on complex queries

Evaluating an IR System

- Note: the **information need** is translated into a **query**
- Relevance is assessed relative to the **information need** *not* the **query**
  - e.g., Information need: *I'm looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.*
  - Query: ***wine red white heart attack effective***
  - Evaluate whether the document addresses the information need, not whether it has these words
- Relevance measurement requires three elements:
  - A benchmark document collection
  - A benchmark suite of queries
  - A binary assessment of either *Relevance* or *Irrelevance* for ach query-doc pair

Standard Relevance Benchmarks

- TREC (Text REtrieval Conference)
  - A large IR testbed by National Institute of Standards and Technology (NIST)
- “Retrieval tasks” specified
- Human experts mark, for each query and for each doc, <u>Relevant</u> or <u>Non-relevant</u>
  - Or at least for subset of docs that some system returned for that query

Binary Evaluation: Precision and Recall

- Documents labeled as relevant or not relevant with respect to a query; For any fixed result set (retrieved documents) of labeled documents:
- **Precision**: fraction of retrieved docs that are relevant

  $= P(\text{relevant}|\text{retrieved})$
- **Recall**: fraction of relevant docs that are retrieved

  $= P(\text{retrieved}|\text{relevant})$

Confusion Matrix

	Relevant	Nonrelevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- True positive (tp): Retrieved (predicted as relevant) and indeed relevant
- True negative (tn): Not Retrieved (predicted as nonrelevant) and indeed nonrelevant
- False positive (fp): Retrieved (predicted as relevant), but in fact nonrelevant
- False negative (fn): Not Retrieved (predicted as nonrelevant), but in fact relevant

Precision

	Relevant	Nonrelevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- Precision P = tp / (tp + fp)

Recall

	Relevant	Nonrelevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- Precision P = tp / (tp + fp)

- Recall R = tp / (tp + fn)

Accuracy

	Relevant	Nonrelevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- Precision P = tp / (tp + fp)
- Recall R = tp / (tp + fn)
- Accuracy = (tp + tn) / (tp + fp + fn + tn)

Confusion Matrix: Example

	Relevant	Nonrelevant
Retrieved	81	252
Not Retrieved	23	9644

- What is Precision?
- What is Recall?
- What is Accuracy?

Should we instead use the accuracy measure for evaluation?

- Given a query an engine classifies each doc as "Relevant" or "Irrelevant".
- The **accuracy** of an engine: the fraction of these classifications that are correct.
- **Accuracy** is a commonly used evaluation measure in machine learning classification work.
- Why is this not a very useful evaluation measure in IR?

Why not just use accuracy?

- How to build a 99.9999% accurate search engine on a low budget….

snoogle.com

Search for:

0 matching results found.

	Relevant	Non-relevant
Retrieved	81	252
Not Retrieved	23	9644

	Relevant	Non-relevant
Retrieved	0	0
Not Retrieved	104	9896

- Accuracy is improved!

Precision-Recall Tradeoff

- You can increase recall by returning more docs.
- Recall is a non-decreasing function of the number of docs retrieved.
- A system that returns all docs has 100% recall!
- The converse is also true (usually): It’s easy to get high precision for very low recall when a small number of docs are retrieved.

A Combined Measure: F_β

- Combined measure that assesses this tradeoff is F_β measure (weighted harmonic mean):

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

  - where $\beta^2 = (1 - \alpha) / \alpha$
- People usually use balanced F_1 measure:
  - i.e., with $\beta = 1$ or $\alpha = ½$
- F_β measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision.
- Harmonic mean is a conservative average.

F_β : Example

	Relevant	Nonrelevant
Retrieved	18	2
Not Retrieved	82	1,000,000,000

- Precision = 18 / (18+2) = 0.9
- Recall = 18 / (18+82) = 0.18
- F_1 = 2*(Precision) / (Precision+Recall)

  = 2*0.9*0.18 / (0.9 + 0.18) = 0.3
- F_1 is a lot lower than the mean (0.9 + 0.18) / 2 = 0.54
- Number of true negatives is not factored in:
  - The F_1 result will be the same for 1000 true negatives (instead of 1,000,000,000).

F_β : Why Harmonic Mean?

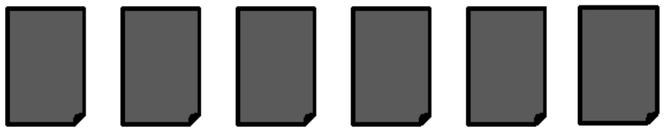
- The simple (arithmetic) mean is 50% for “snoogle”, which is too high.
- Main idea: Punish really bad performance on one of precision/recall.

Evaluating Ranked Results

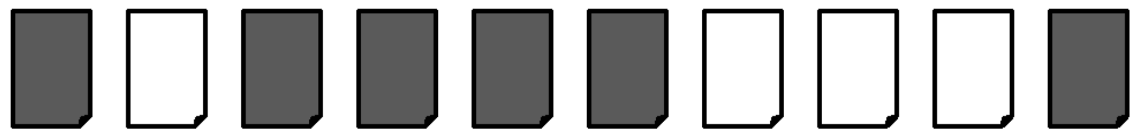
- Search engine returns ranked list of documents
  - Take first document, interpret as unordered set of size 1, compute unordered evaluation measures for this set.
  - Take top 2 documents, interpret as unordered set of size 2, compute unordered evaluation measures for this set.
  - Take top 3 documents, interpret as unordered set of size 3, compute unordered evaluation measure for this set.
  - ...
  - Plot individual measures → precision-recall curve

- Or : Combine individual measures into one.

Evaluating Ranked Results: Example

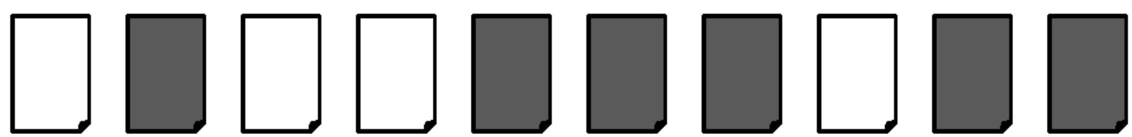
 = the relevant documents

Ranking #1



Recall	0.17	0.17	0.33	0.5	0.67	0.83	0.83	0.83	0.83	1.0
Precision	1.0	0.5	0.67	0.75	0.8	0.83	0.71	0.63	0.56	0.6

Ranking #2



Recall	0.0	0.17	0.17	0.17	0.33	0.5	0.67	0.67	0.83	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.5	0.57	0.5	0.56	0.6

Top 10 documents of two different rankings together with the recall and precision values calculated at every rank position for a query with 6 relevant documents

Evaluating Ranked Results: Example

At rank position 10, the two rankings have same effectiveness

- Recall 1.0, precision 0.6

At higher rank positions, the first ranking is better

- e.g., at rank position 4, the first ranking has 0.5 recall and 0.75 precision, the second ranking has 0.17 recall and 0.25 precision

Summarizing Ranking Evaluation

Method 1

- Calculating recall and precision at fixed rank positions

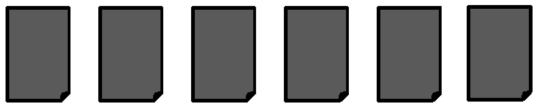
Method 2

- Calculating precision at standard recall levels, from 0.0 to 1.0 in increments of 0.1. Each ranking then is represented using 11 numbers.
  - requires *interpolation* (refers to any technique for calculating a new point between two existing data points, 보간법)
  - Generate recall-precision graph

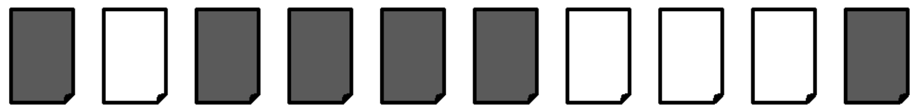
Method 3

- Averaging the precision values from the rank positions where a relevant document was retrieved
  - Average precision (평균 정확률)

Average Precision: Example

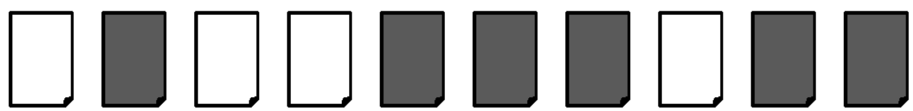
 = the relevant documents

Ranking #1



Recall	0.17	0.17	0.33	0.5	0.67	0.83	0.83	0.83	0.83	1.0
Precision	1.0	0.5	0.67	0.75	0.8	0.83	0.71	0.63	0.56	0.6

Ranking #2



Recall	0.0	0.17	0.17	0.17	0.33	0.5	0.67	0.67	0.83	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.5	0.57	0.5	0.56	0.6

$$\text{Ranking \#1: } (1.0 + 0.67 + 0.75 + 0.8 + 0.83 + 0.6)/6 = 0.78$$

$$\text{Ranking \#2: } (0.5 + 0.4 + 0.5 + 0.57 + 0.56 + 0.6)/6 = 0.52$$

Average Precision

- It is single number based on the ranking of all the relevant documents (적합한 문서들의 순위에 근거한).
- The value depends heavily on the highly ranked relevant documents.
- It is an appropriate measure for evaluating the task of finding as many relevant documents as possible while reflecting the intuition that top-ranked documents are the most important.
- We need a method of calculating the average effectiveness for the entire set of queries $\rightarrow$ MAP.

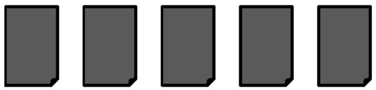
Mean Average Precision (MAP)

Mean Average Precision (MAP)

- summarize rankings from multiple queries by averaging average precision (평균 정확률을 모든 질의에 대하여 평균을 구함)
- most commonly used measure in research papers
- assumes user is interested in finding many relevant documents for each query
- requires many relevance judgments in text collection

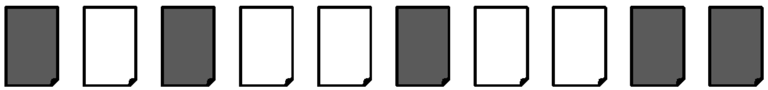
- Precision-Recall graph is also useful summaries.
  - Although MAP is useful, sometimes too much information is lost.

Mean Average Precision (MAP): Example

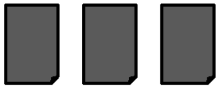


= relevant documents for query 1

Ranking #1

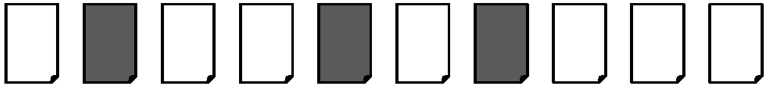


Recall	0.2	0.2	0.4	0.4	0.4	0.6	0.6	0.6	0.8	1.0
Precision	1.0	0.5	0.67	0.5	0.4	0.5	0.43	0.38	0.44	0.5



= relevant documents for query 2

Ranking #2



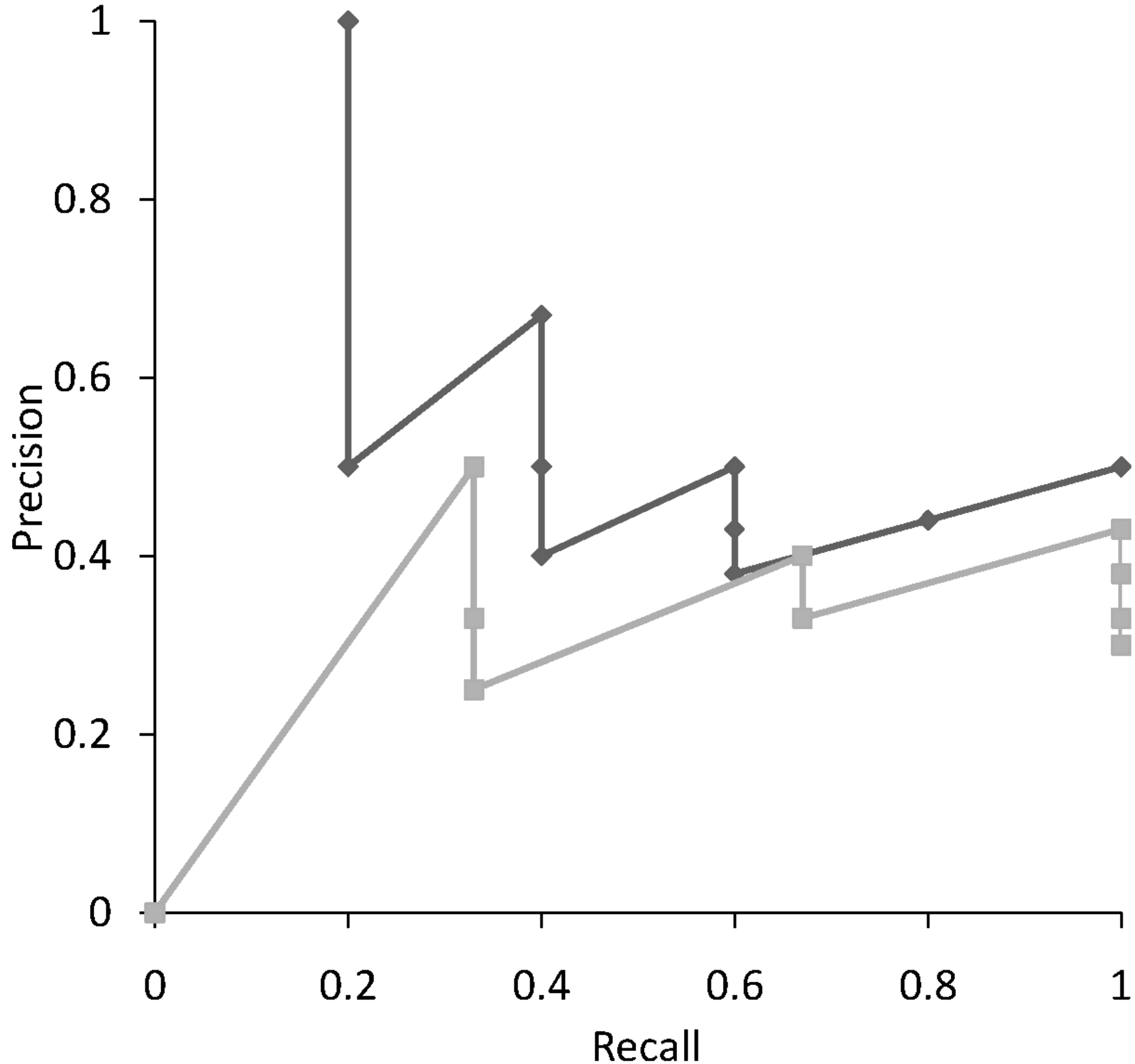
Recall	0.0	0.33	0.33	0.33	0.67	0.67	1.0	1.0	1.0	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.33	0.43	0.38	0.33	0.3

$$average\ precision\ query\ 1 = (1.0 + 0.67 + 0.5 + 0.44 + 0.5)/5 = 0.62$$

$$average\ precision\ query\ 2 = (0.5 + 0.4 + 0.43)/3 = 0.44$$

$$mean\ average\ precision = (0.62 + 0.44)/2 = 0.53$$

Precision Recall Curve



Interpolation

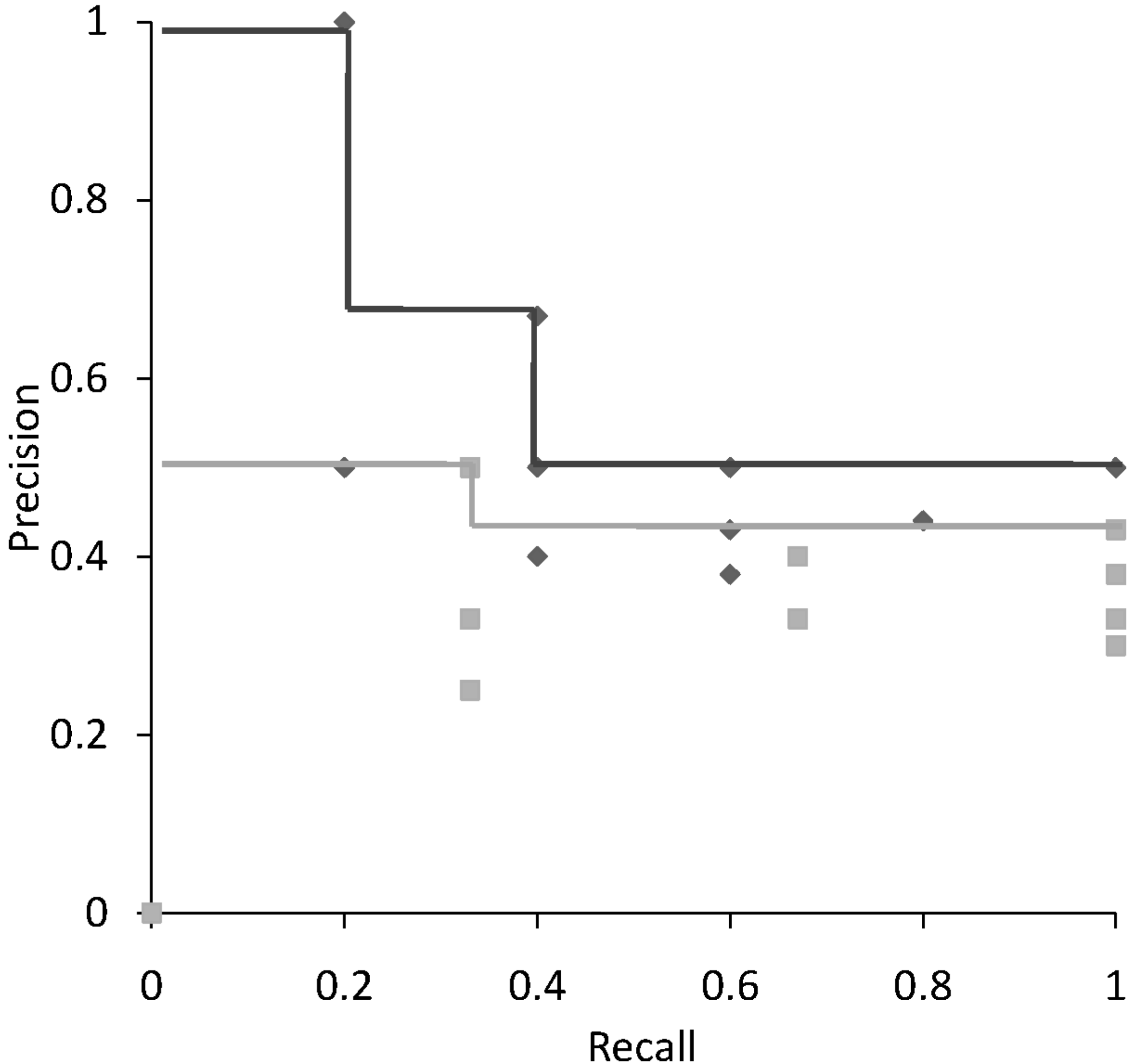
Why we need interpolation:

- We want to calculate precision at standard recall levels, from 0.0 to 1.0 in increments of 0.1.
- However, we do not have the exact value for these recall levels.

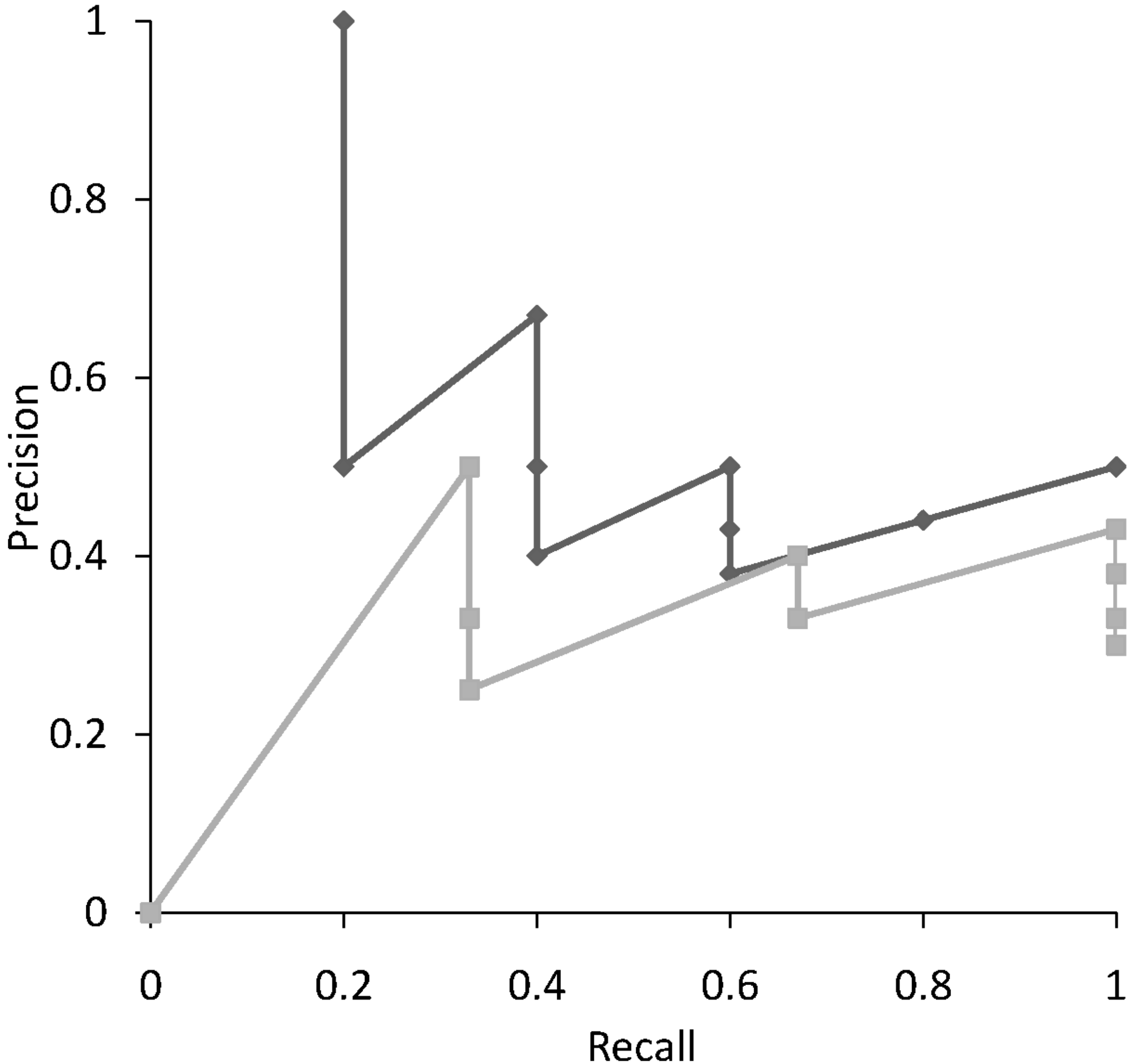
How we do interpolation:

- Defines precision at any recall level as the *maximum precision observed in any recall-precision point at a higher recall level* (재현율 수준에 따른 정확도를 정의하는데 만일 더 높은 재현율 수준에서의 관찰된 정확도가 현재의 정확도보다 높다면 더 높은 정확도를 현재의 정확도로 정의한다)
  - produces a step function
  - defines precision at recall 0.0

Interpolation: Example



Precision Recall Curve



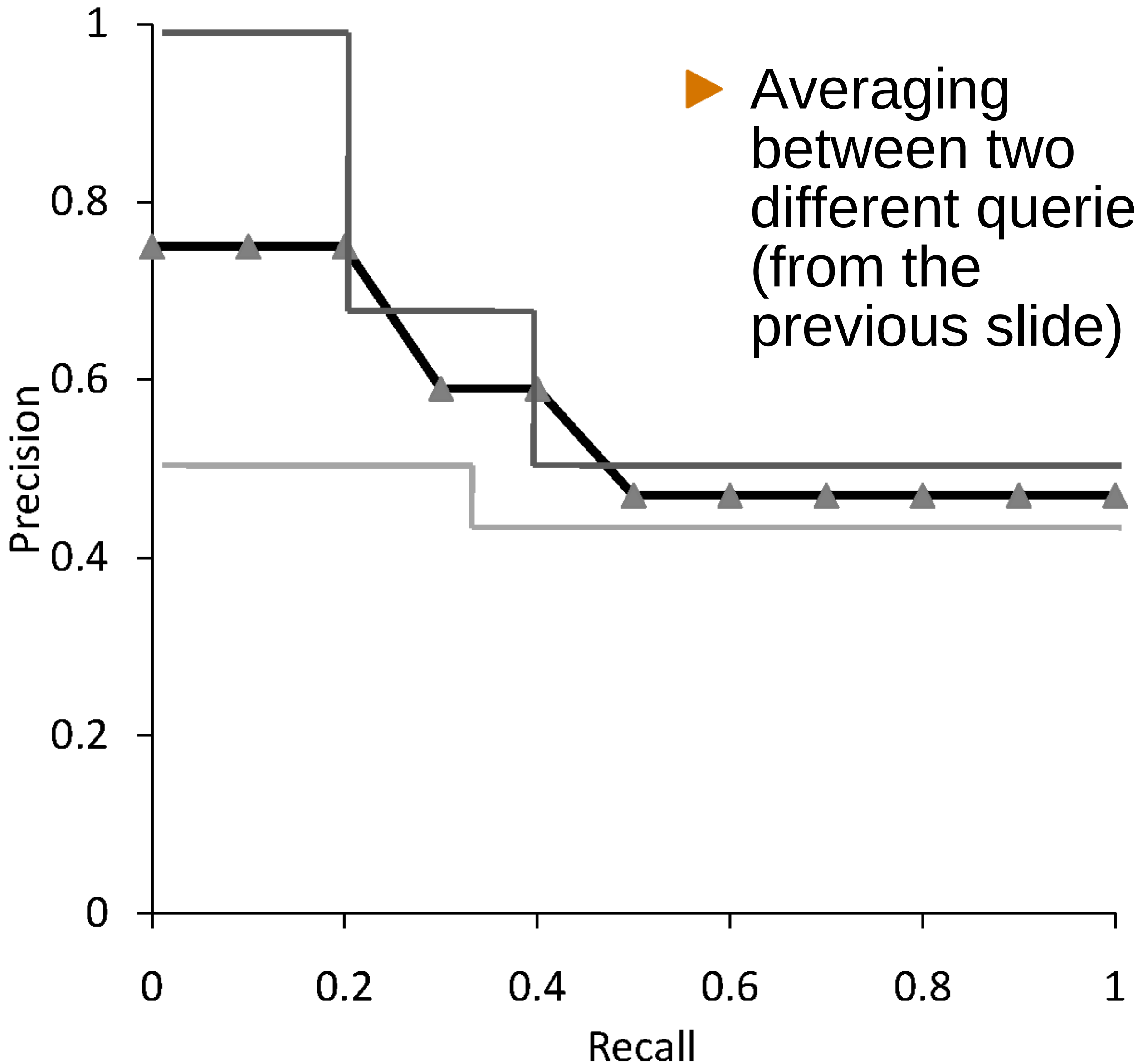
Average Precision at Standard Recall Levels

Averaging between two different queries

Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Ranking 1	1.0	1.0	1.0	0.67	0.67	0.5	0.5	0.5	0.5	0.5	0.5
Ranking 2	0.5	0.5	0.5	0.5	0.43	0.43	0.43	0.43	0.43	0.43	0.43
Average	0.75	0.75	0.75	0.59	0.47	0.47	0.47	0.47	0.47	0.47	0.47

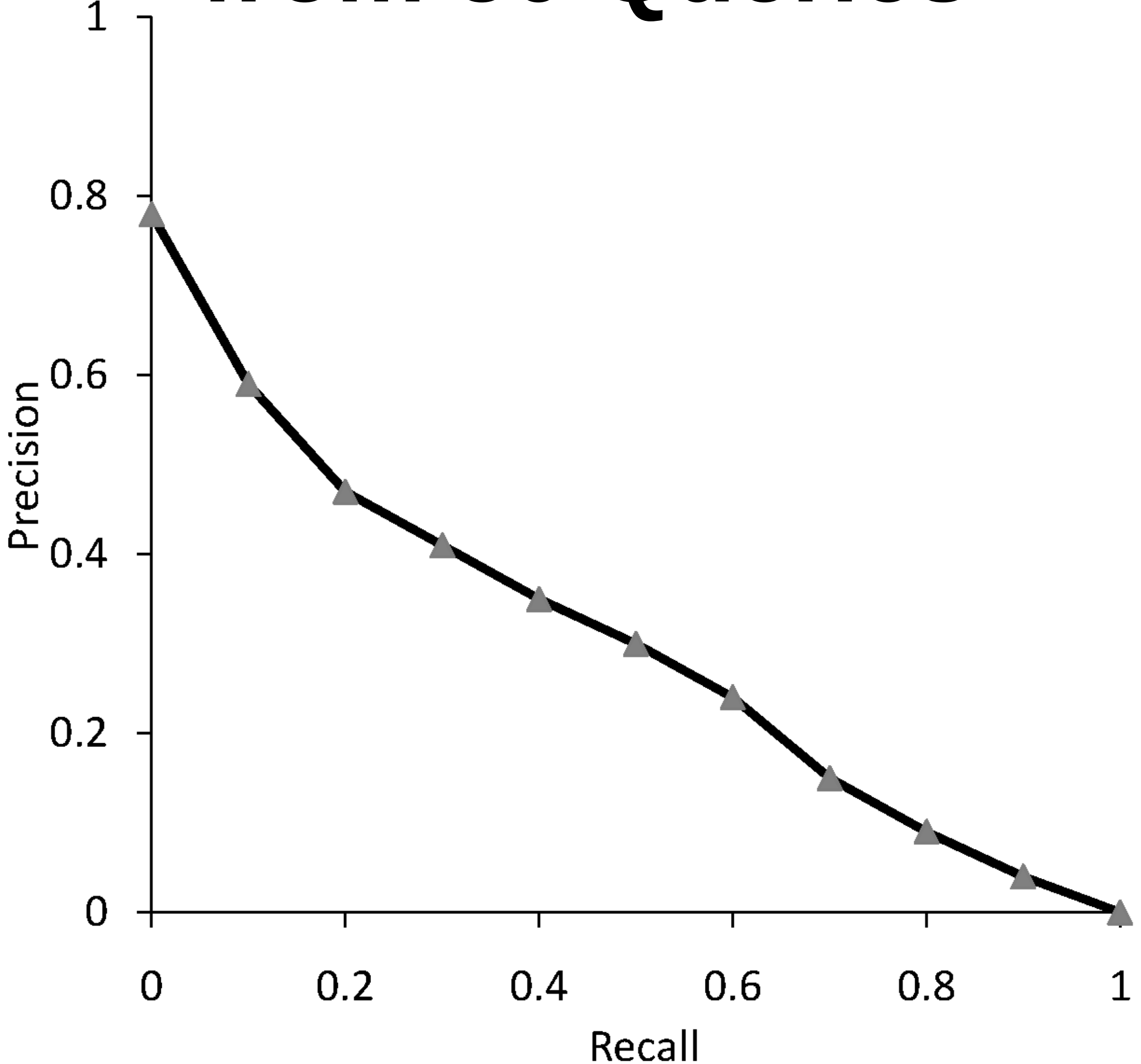
- Precision-recall curve plotted by simply joining the average precision points at the standard recall levels.

Average Precision-Recall Curve



- Averaging between two different queries (from the previous slide)

Average Precision-Recall Curve from 50 Queries



Focusing on Top Documents

- Users tend to look at only the top part of the ranked result list to find relevant documents.
- Some search tasks have only one relevant document (첫번째 검색결과만)
  - Navigational search, e.g., querying ‘youtube’ for finding a particular website
  - Question answering
- Recall is not appropriate.
  - Instead, we need to measure how well the search engine does at retrieving relevant documents at very high ranks.

Focusing on Top Documents: Methods

Precision at Rank R (R번째 순위에서의 정확도)

- R is typically set as 5, 10, or 20.
- Easy to compute, average, and understand
- Not sensitive to rank positions less than R

Reciprocal Rank (역순위)

- Reciprocal (역수) of the rank at which the first relevant document is retrieved.
  - e.g., 1st rank → 1/1, 3rd rank → 1/3.
- *Mean Reciprocal Rank (MRR)* (평균역순위) is the average of the reciprocal ranks over a set of queries.
- Very sensitive to the rank position of the first relevant document.

Discounted Cumulative Gain (DCG)

- Popular measure for evaluating web search and related tasks.

Two assumptions:

- Highly relevant documents are more useful than marginally relevant document (매우 적합한 문서가 조금 적합한 문서보다 유용하다.)
- The lower the ranked position of a relevant document is, the less useful it is for the user, since it is less likely to be examined.

Discounted Cumulative Gain (DCG)

Basic ingredients:

- Uses *graded relevance* (등급이 있는 적합성) as a measure of the usefulness, or *gain,* from examining a document.
  - e.g., Relevance level = {Perfect, Excellent, Good, Fair, Bad}
- Each ordinal label is associated with a relevance value rel
  - e.g., Perfect → 20, Excellent → 10, Good → 5, Fair → 1, Bad → 0
- A set of discount factors: $c_1 > c_2 > \cdots > c_k > 0$ where k is the rank value (e.g., 1st rank, 2nd rank, …).
  - Typical discount factor: c_1 =1 and $c_k = 1/\log_2(k)$ for $k > 2$.
  - With the log base of 2, the discount at rank 4 is 1/2, and at rank 8, it is 1/3.

Discounted Cumulative Gain (DCG)

DCG is defined as the total gain accumulated at a particular rank p:

- $DCG_p = \sum_{k=1}^{p} c_k \times rel_k$
- Where k is the rank value, rel_k is the relevance value of the document retrieved at rank k, and c_k is the discount factor at rank k.

For example, if $c_1 = 1$ and $c_k = 1/\log_2 k$ for $k > 2$,

- $DCG_p = rel_1 + \sum_{k=2}^{p} \frac{rel_k}{\log_2 k}$

DCG Example

10 ranked documents judged on 0-3 relevance scale:

- 3 , 2 , 3 , 0 , 0 , 1 , 2 , 2 , 3 , 0

Discounted gain (where $c_1 = 1$ and $c_k = 1/\log_2 k$ for $k > 2$)

- $c_k \times rel_k$: $\frac{3}{1}, \frac{2}{1}, \frac{3}{1.59}, \frac{0}{2}, \frac{0}{2.32}, \frac{1}{2.59}, \frac{2}{2.81}, \frac{2}{3}, \frac{3}{3.17}, \frac{0}{3.32}$

  = 3, 2, 1.89, 0, 0, 0.39, 0.71, 0.67, 0.95, 0

Discounted Cumulative Gain (DCG)

- 3, 5 (=3+2), 6.89 (=3+2+1.89), 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61

DCG numbers are averaged across a set of queries at specific rank values:

- e.g., DCG at rank 5 is 6.89 and at rank 10 is 9.61.

Normalized DCG (NDCG)

DCG values are often normalized by comparing the DCG at each rank with the DCG value for the perfect ranking.

Perfect scenario:

- Ranking: 3 , 3 , 3 , 2 , 2 , 2 , 1 , 0 , 0 , 0
- Discounted gain ($c_1 = 1$ and $c_k = 1/\log_2 k$ for $k > 2$)
  - 3 , 3 , 1.89 , 1 , 0.86 , 0.77 , 0.35 , 0 , 0 , 0
- Ideal DCG values
  - 3 , 6 , 7.89 , 8.89 , 9.75 , 10.52 , 10.88 , 10.88 , 10.88 , 10.88

Actual scenario:

- Actual DCG values
  - 3 , 5 , 6.89 , 6.89 , 6.89 , 7.28 , 7.99, 8.66 , 9.61 , 9.61
- Normalized DCG (NDCG) values (divide actual by ideal):
  - 1 (=3/3), 0.83 (=5/6), 0.87 (=6.89/7.89), 0.76 , 0.71 , 0.69 , 0.73 , 0.8 , 0.88 , 0.88
  - NDCG $\leq$ 1 at any rank position. (that is why we call it normalized DCG.)

Critique of Pure Relevance

- Relevance vs. Marginal Relevance
  - Marginal relevance: what is additional, new information after looking at the previously ranked documents?
  - A document can be redundant even if it is highly relevant.
  - There could be duplicates, or the same information from different sources.
  - Marginal relevance is a better measure of utility for the user.

- Other factors: diversity, novelty, recency, ...

Can We Avoid Human Judgment?

- Human judgments makes experimental work hard
  - Especially on a large scale
- In some very specific settings, can use proxies
- Implicit relevance judgments for web search: clicks

A/B Testing

- Purpose: Test a single innovation
- Prerequisite: You have a large search engine up and running.
- Have most users use old system.
- Divert a small proportion of traffic (e.g., 1%) to the new system that includes the innovation.
- Evaluate with an “automatic” measure like clickthrough on first result.
- Now we can directly see if the innovation does improve user happiness.
- Probably the evaluation methodology that large search engines trust most.
- In principle, less powerful than doing a multivariate regression analysis, but easier to understand.
- Microsoft example: http://www.exp-platform.com/Pages/KDD2015KeynoteExPKohavi.aspx

Topic Modeling

What is a Topic Modeling?

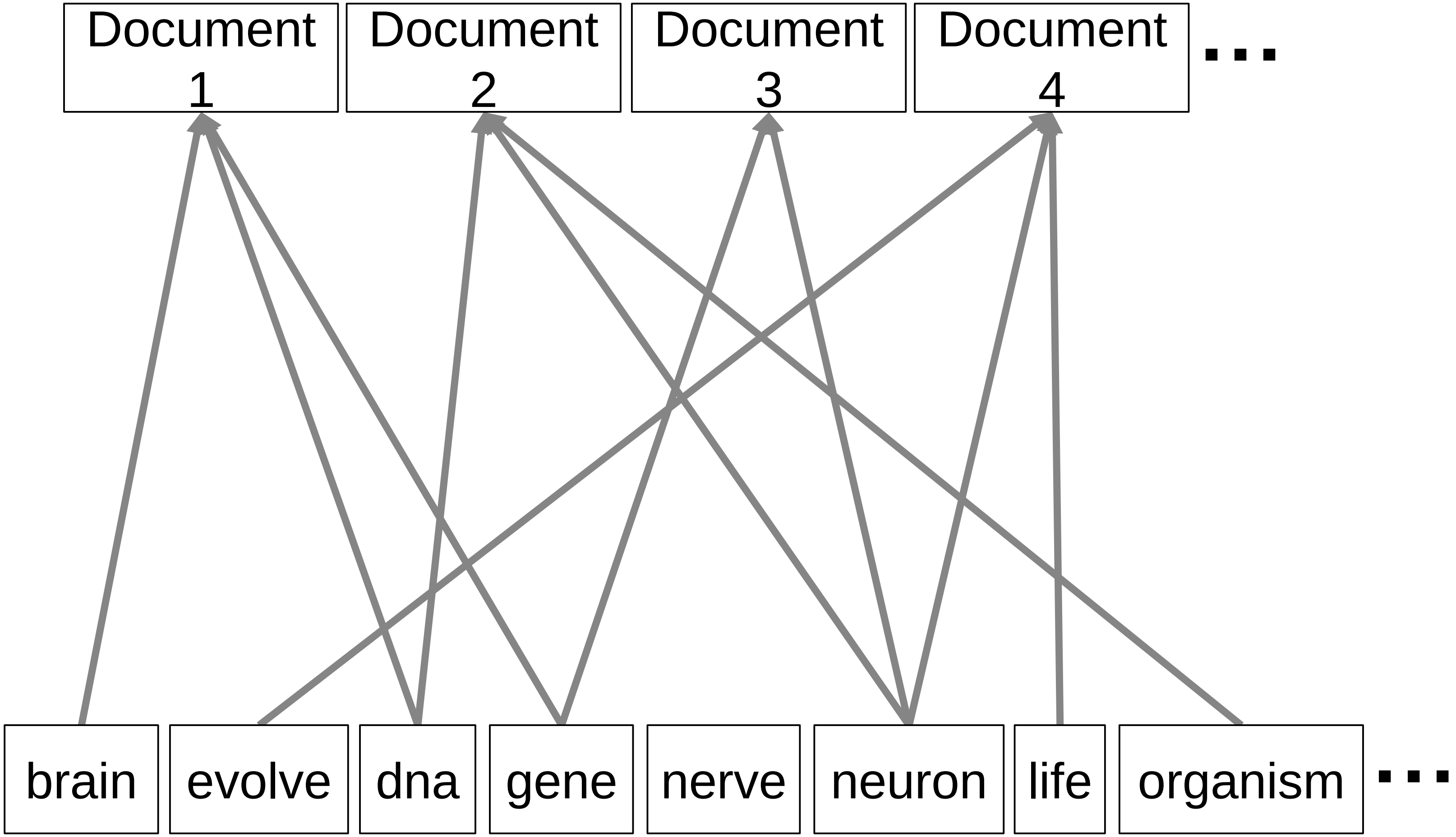
Recall:

- A topic is a probability distribution or keywords.
  - A different keyword has a different probability.
- Alternatively, a topic is a weighted combination of keywords.
  - A different keyword has a different importance score (or simply a different weight).

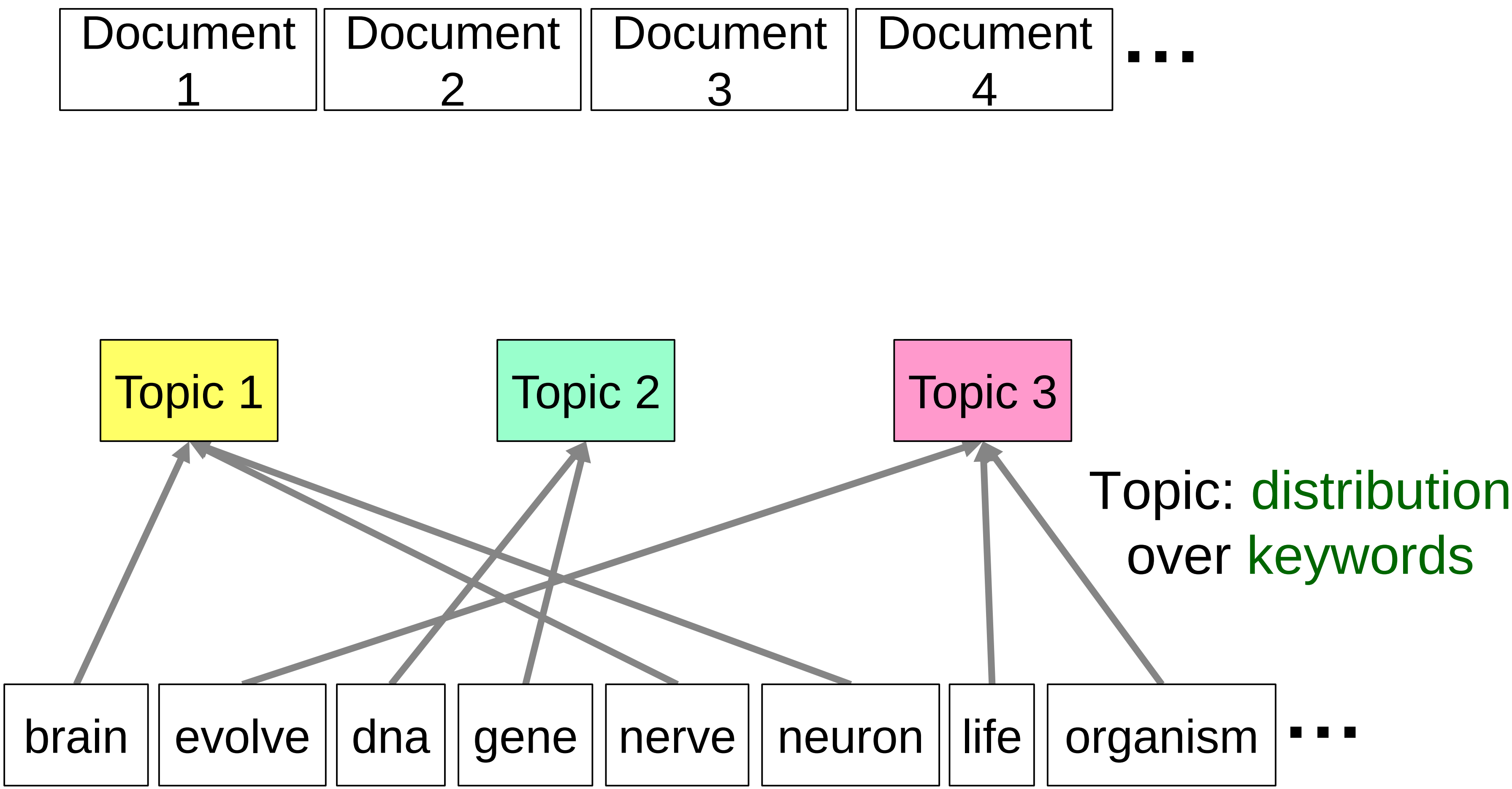
What is topic modeling?

- Topic modeling is a technique that gives a set of topics out of a document corpus.
- Additionally, topic modeling represents a document as a probability distribution over topics.
- More generally, topic modeling represents a document as a weighted combination of topics.

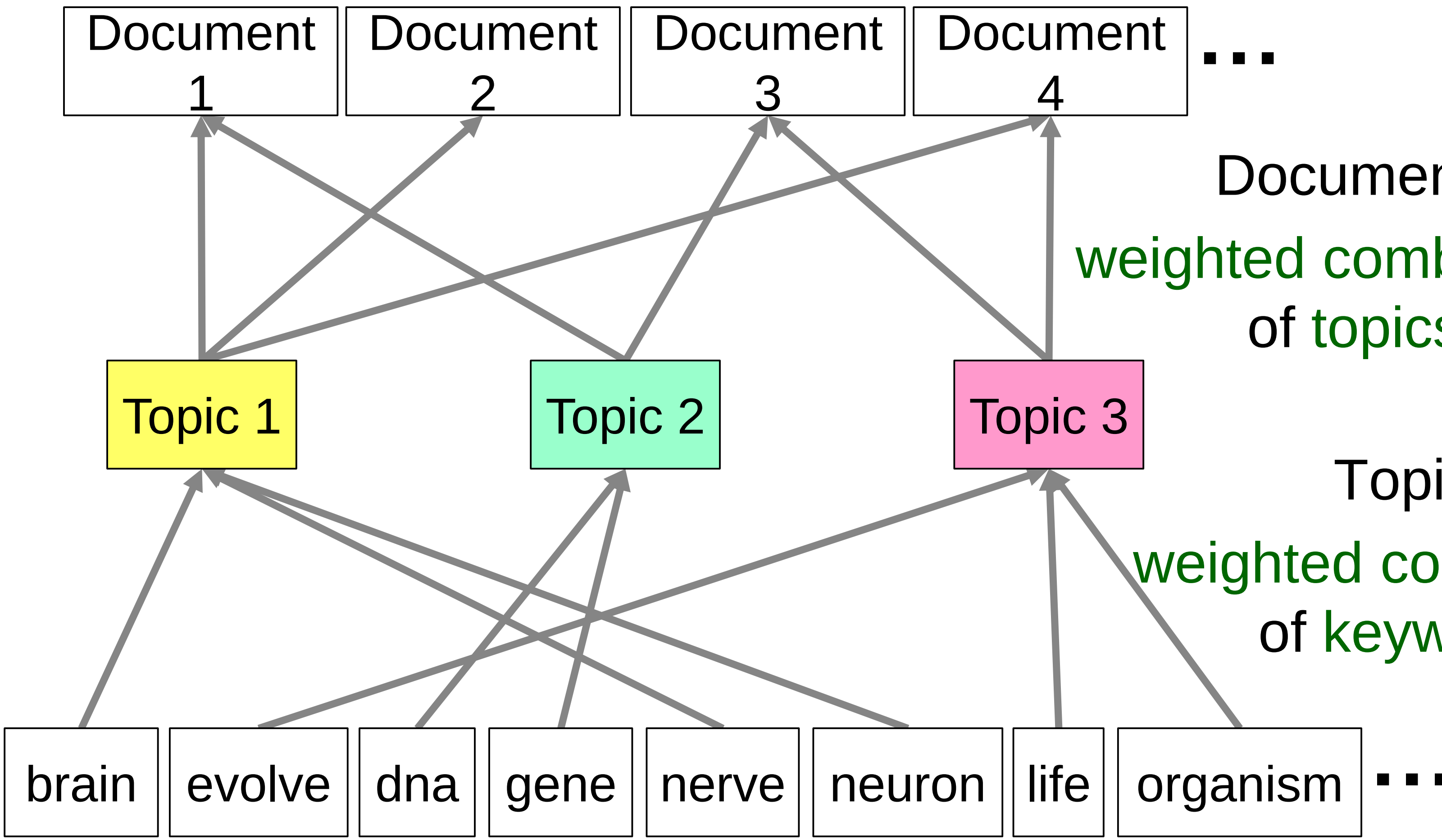
Topic Modeling:
Overview



Topic Modeling:
Overview



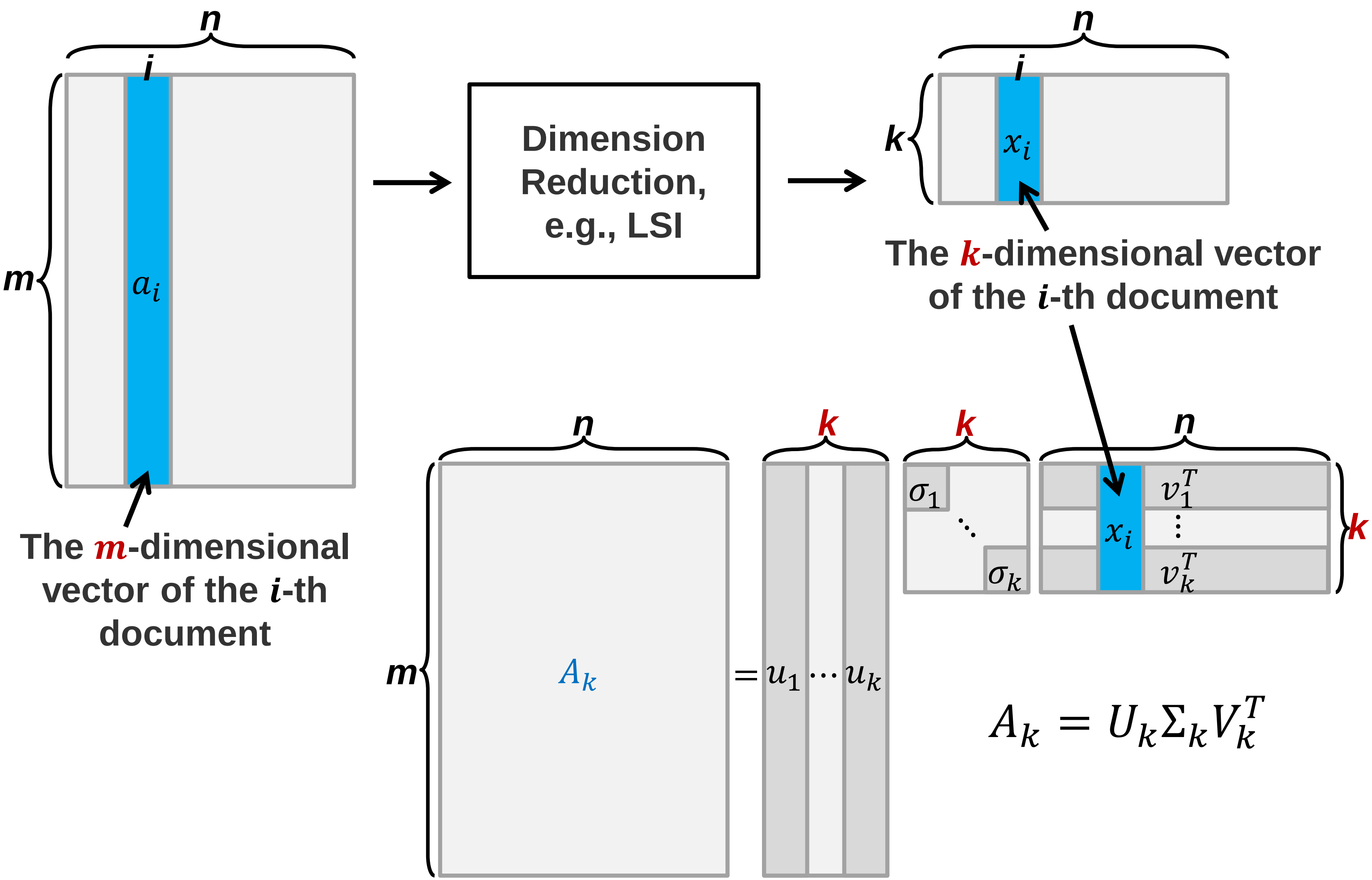
Topic Modeling:
Overview



Document:
weighted combination of topics

Topic:
weighted combination of keywords

Latent Semantic Indexing (LSI)



LSI Can Handle Polysemy and Synonymy

- Polysemy: Words often have a multitude of meanings and different types of usage (more severe in very heterogeneous collections).
- The vector space model is unable to discriminate between different meanings of the same word.

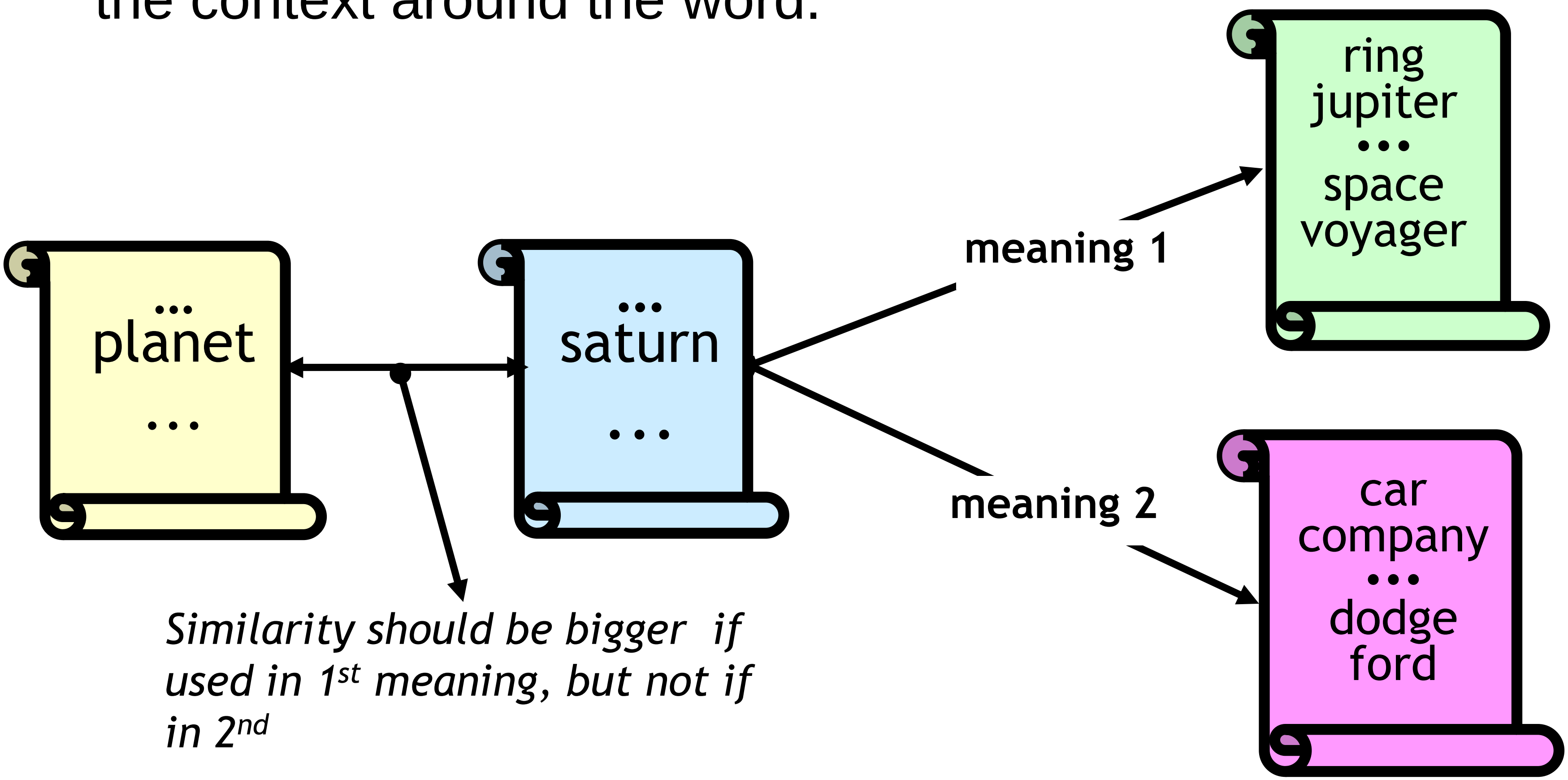
$$\mathrm{sim}_{\mathrm{true}}(d, q) < \cos(\angle(\vec{d}, \vec{q}))$$

- Synonymy: Different terms may have an identical or a similar meaning (weaker: words indicating the same topic).
- No associations between words are made in the vector space representation.

$$\mathrm{sim}_{\mathrm{true}}(d, q) > \cos(\angle(\vec{d}, \vec{q}))$$

Polysemy and Context

- Document similarity on single word level: polysemy and the context around the word.



Latent Semantic Indexing

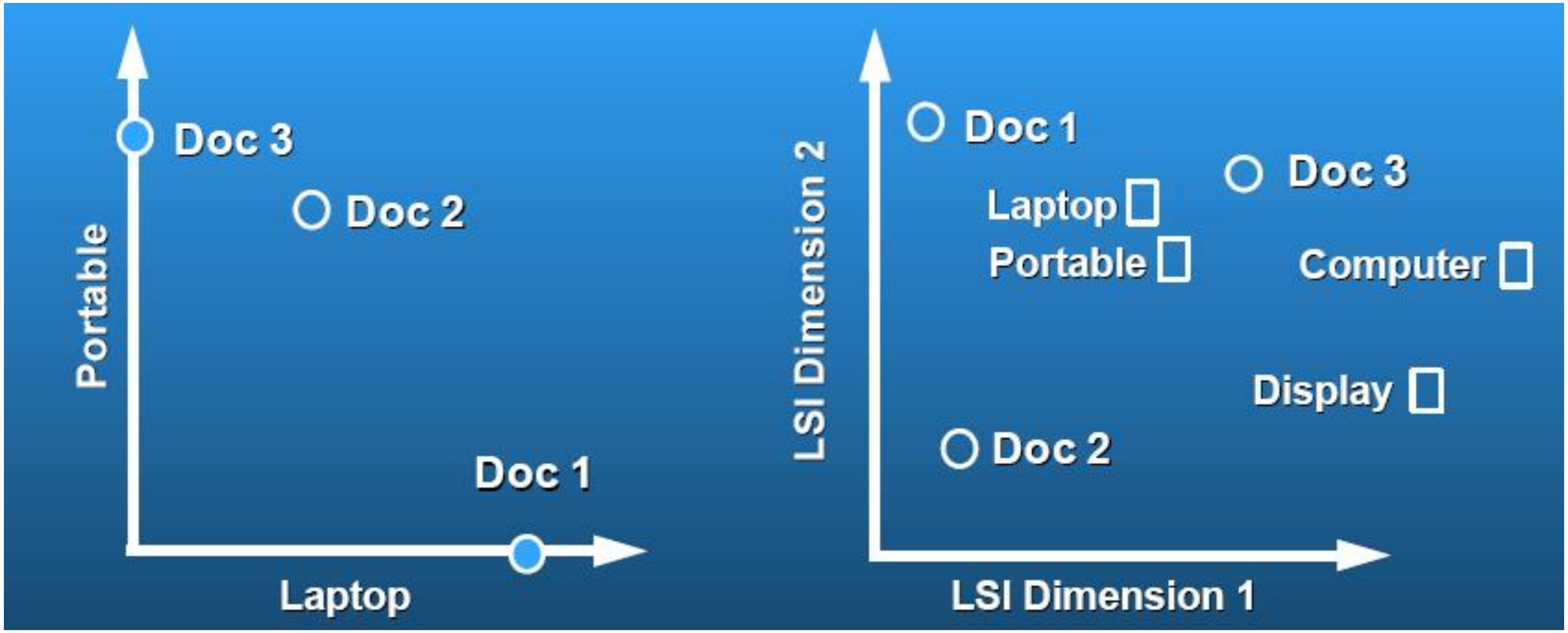
- Perform a **low-rank approximation** of a **term-document matrix** (typical rank **100–300**)
- General idea
  - Map documents (and terms) to a **low-dimensional** representation.
  - Design a mapping such that the low-dimensional space reflects **semantic associations** (latent semantic space).
  - Compute document similarity based on the **inner product** in this **latent semantic space**.

Latent Semantic Indexing: Goals

- LSI takes documents that are semantically similar (= talk about the same topics), but are not similar in the vector space (because they use different words) and re-represents them in a reduced vector space in which they have higher similarity.

- Similar terms map to similar location in low-dimensional space.
- Noise reduction by dimension reduction

Latent Semantic Indexing: Illustrating Example



courtesy of Susan Dumais

Latent Semantic Indexing: SVD in Term-Document Matrix

- A text corpus can be represented by the term-document matrix
- $A \in R^{m \times n}$, m: vocabulary size, n: number of documents
- Approximation based on Singular Value Decomposition (SVD) of A.
  - $A \simeq U_k \Sigma_k V_k^T$
  - rows of V_k are k-dimensional representations of documents.
  - documents are projected into a “latent" semantic space.
  - co-occurring terms(or documents) are projected into same dimension
- In latent semantic space, two documents may have high cosine similarity even if they do not share any terms.
- Dimensions of the latent space correspond to the axes of the greatest variation.

Mapping Documents to a Low-Dimensional Space

- Each row and column of *A* gets mapped into the *k*-dimensional LSI space, by the SVD.
- *Claim* – this is not only the mapping with the best (Frobenius error) approximation to *A*, but in fact *improves* retrieval.

- Since $V_k = A_k^T U_k \Sigma_k^{-1}$
  we should transform query *q* to q_k as follows

$$q_k = q^T U_k \Sigma_k^{-1}$$

  - A query is NOT a sparse vector.

How LSI Addresses Synonymy and Semantic Relatedness

- The dimensionality reduction forces us to omit “details”.
- We have to map different words (= different dimensions of the full space) to the same dimension in the reduced space.
- The “cost” of mapping synonyms to the same dimension is much less than the cost of collapsing unrelated words.
- SVD selects the “least costly” mapping (see below).
- Thus, it will map synonyms to the same dimension.
- But, it will avoid doing that for unrelated words.

Empirical Evidence

- Experiments on TREC 1/2/3 – Dumais
- Lanczos SVD code (available on netlib) due to Berry used in these expts
  - Running times of ~ one day on tens of thousands of docs [still an obstacle to use]
- Dimensions – various values 250-350 reported. Reducing *k* improves recall.
  - (Under 200 reported unsatisfactory)
- Generally expect recall to improve – what about precision?

Empirical Evidence

- Precision at or above median TREC precision
  - Top scorer on almost 20% of TREC topics
- Slightly better on average than straight vector spaces
- Effect of dimensionality:

Dimensions	Precision
250	0.367
300	0.371
346	0.374

Example of $C = U\Sigma V^T$: The matrix C

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

- This is a standard term-document matrix. Actually, we use a non-weighted matrix here to simplify the example.

Example of $C = U\Sigma V^T$: The matrix U

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

- This is an orthonormal matrix:
  - (i) Row vectors have unit length.
  - (ii) Any two distinct row vectors are orthogonal to each other.
- Think of the dimensions (columns) as "semantic" dimensions that capture distinct topics like politics, sports, economics. Each number u_{ij} in the matrix indicates how strongly related term i is to the topic represented by semantic dimension j .

Example of $C = U\Sigma V^T$: The matrix Σ

Σ	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	1.28	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00
5	0.00	0.00	0.00	0.00	0.39

- This is a square, diagonal matrix.
- The diagonal consists of the singular values of C. The magnitude of the singular value measures the importance of the corresponding semantic dimension. We make use of this by omitting unimportant dimensions.

Example of $C = U\Sigma V^T$: The matrix V^T

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

- Again, this is an orthonormal matrix
  - Column vectors have unit length.
  - Any two distinct column vectors are orthogonal to each other.
- These are again the semantic dimensions from the term matrix U that capture distinct topics like politics, sports, economics. Each number v_{ij} in the matrix indicates how strongly related document i is to the topic represented by semantic dimension j .

Example of $C = U\Sigma V^T$: All four matrices

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

$=$

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

$\times$

Σ	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	1.28	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00
5	0.00	0.00	0.00	0.00	0.39

$\times$

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

Reducing the Dimensionality to 2

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

$=$

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

$\times$

Σ_2	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00

$\times$

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

Original Matrix C vs. Reduced $C_2 = U\Sigma_2 V^T$

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

We can view C_2 as a two-dimensional representation of the matrix.

We have performed a dimensionality reduction to two dimensions.

Why is the Reduced Matrix "better"?

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

Similarity of d_2 and d_3 in the original space: 0.

Similarity of d_2 und d_2 in the reduced space:
0.52 * 0.28 + 0.36 * 0.16 + 0.72 * 0.36 + 0.12 * 0.20 + - 0.39 * - 0.08 ≈ 0.52

Why is the Reduced Matrix “better”?

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

“boat” and “ship” are semantically similar.

The “reduced” similarity measure reflects this.

What property of the SVD reduction is responsible for improved similarity?

LSI has Many Other Applications

- In many settings in pattern recognition and retrieval, we have a feature-object matrix.
  - For text, the terms are features and the docs are objects.
  - Could be opinions and users …
  - This matrix may be redundant in dimensionality.
  - Can work with low-rank approximation.
  - If entries are missing (e.g., users' opinions), can recover if dimensionality is low.
- Powerful general analytical technique.
  - Close, principled analog to clustering methods.

Difference between LSI and Topic Modeling

- LSI can be viewed as a topic modeling method

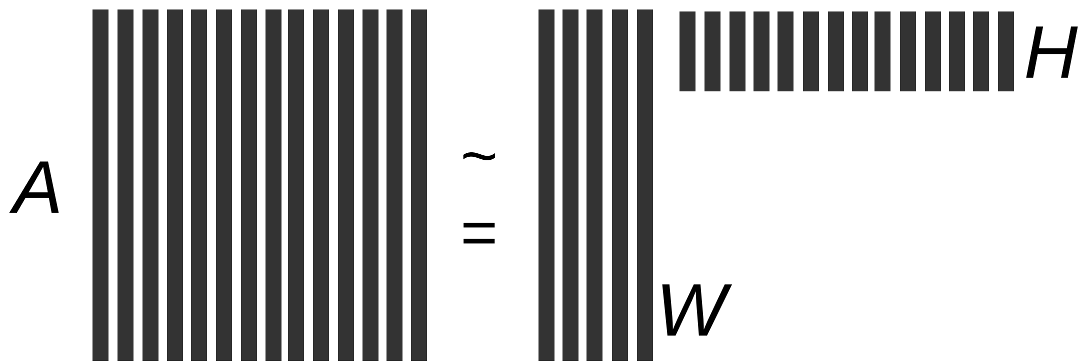
However…

- The main difference is that
  - LSA allows both positive and negative values.
  - Topic modeling allows only positive (or zero) values.

- Two different classes of topic modeling methods
  - Matrix factorization methods: nonnegative matrix factorization
  - Probabilistic methods: probabilistic LSI (pLSI), latent Dirichlet allocation (LDA)

Nonnegative Matrix Factorization (NMF)

Low-rank approximation via matrix factorization



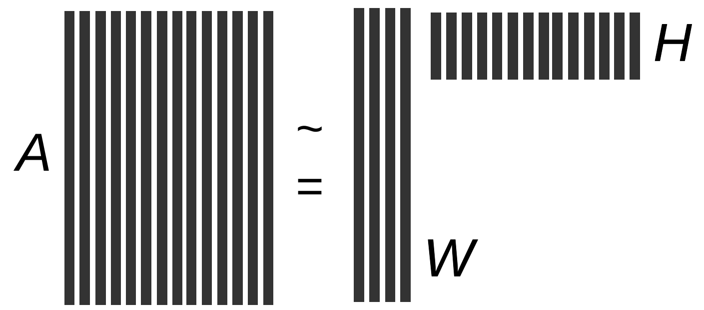
$\Rightarrow \min \| A - WH \|_F$

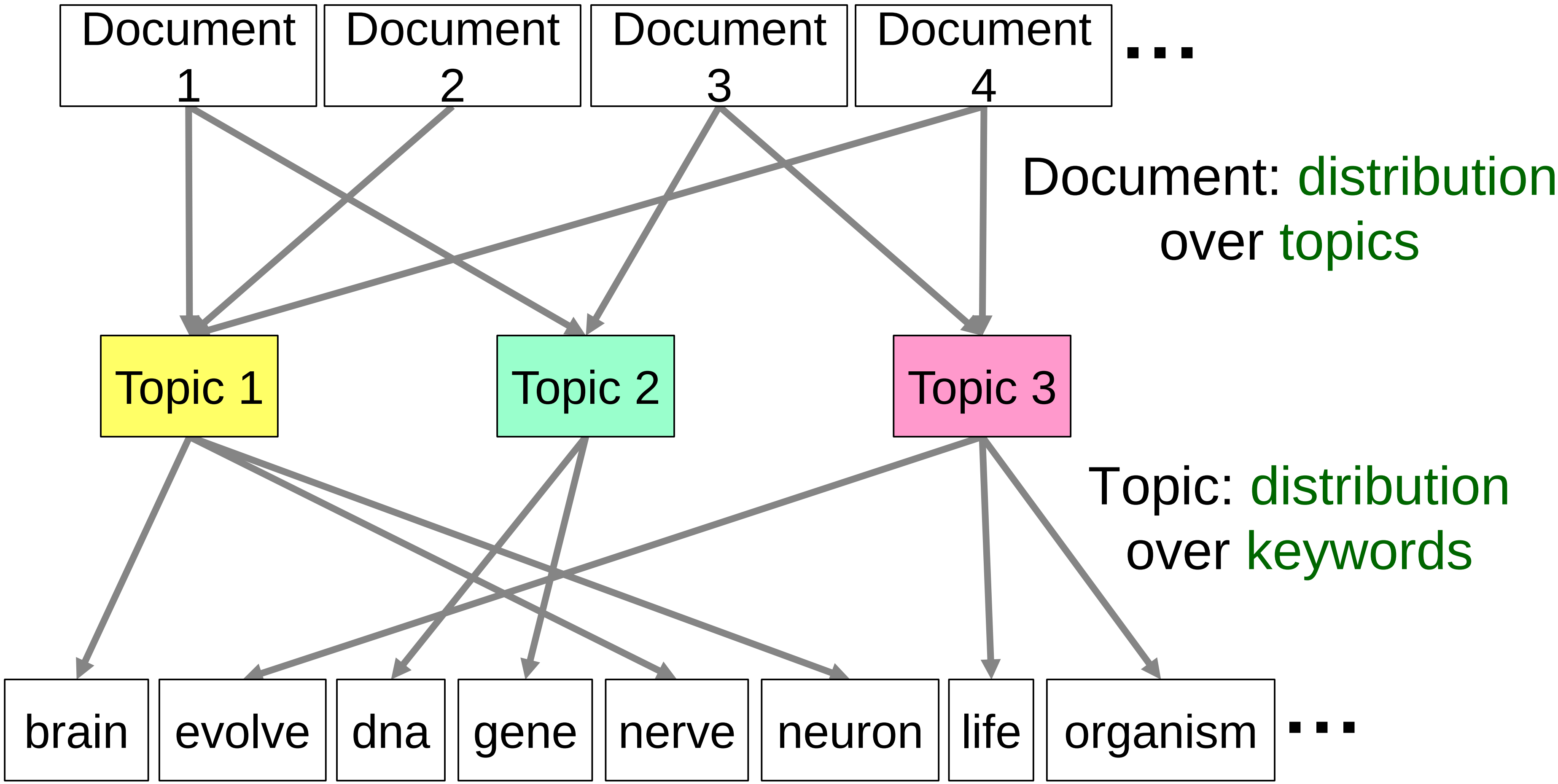
$W>=0, H>=0$

Why nonnegativity constraints?

- Better interpretation (vs. better approximation, e.g., SVD)

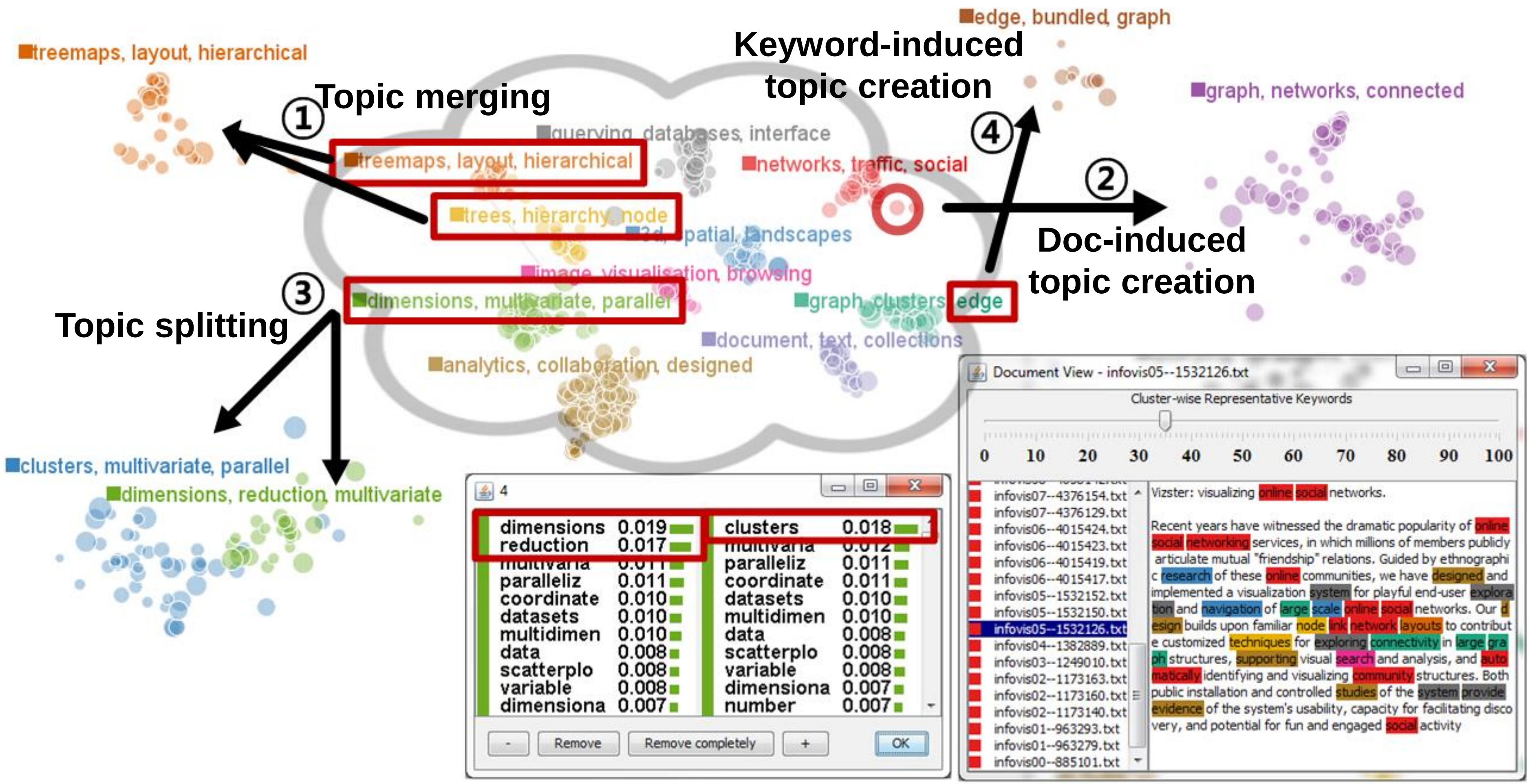
NMF as Topic Modeling



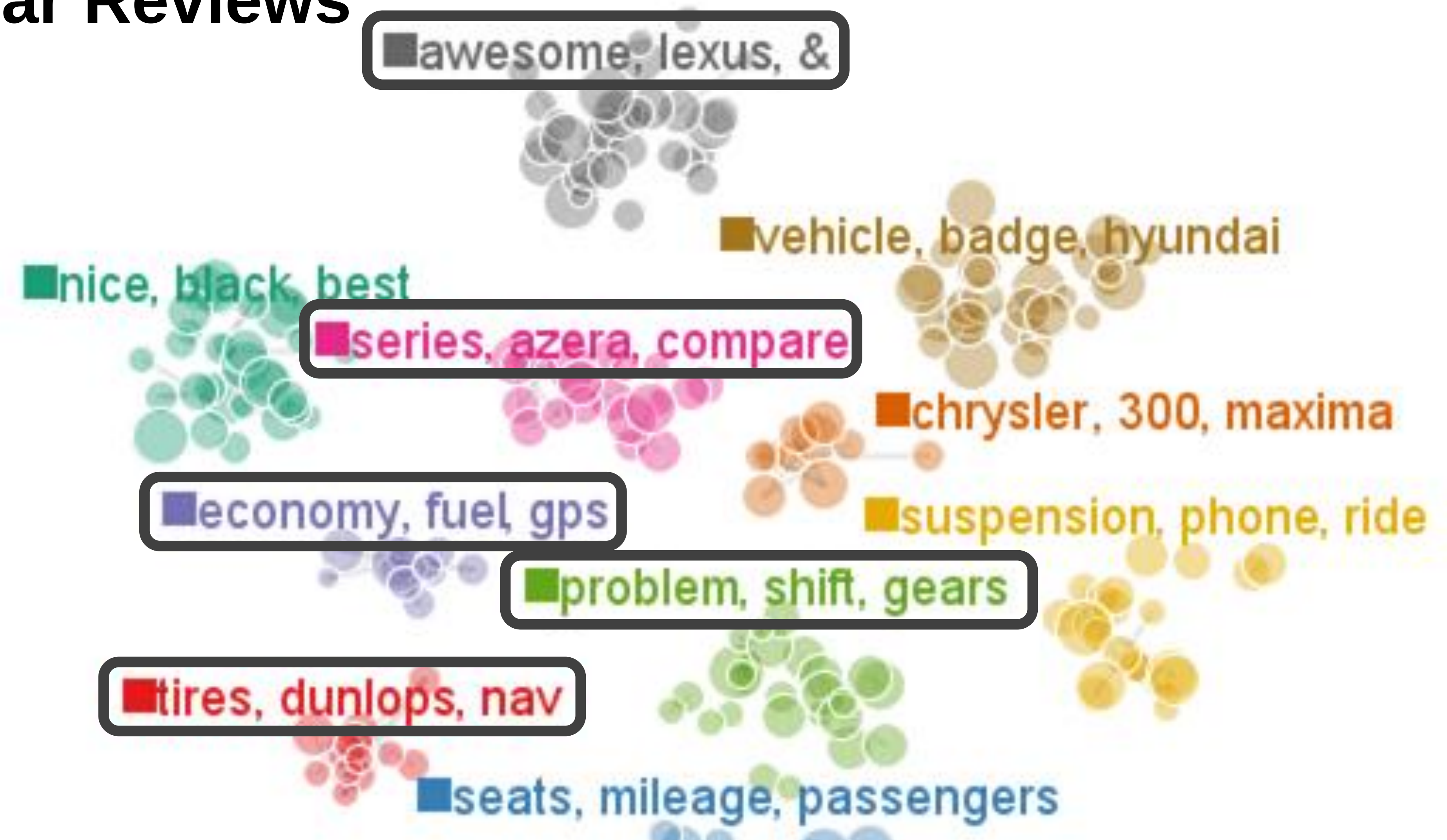


UTOPIAN: User-Driven Topic Modeling Based on Interactive NMF

[Choo et al., TVCG 2013]



Visualization Example:
Car Reviews



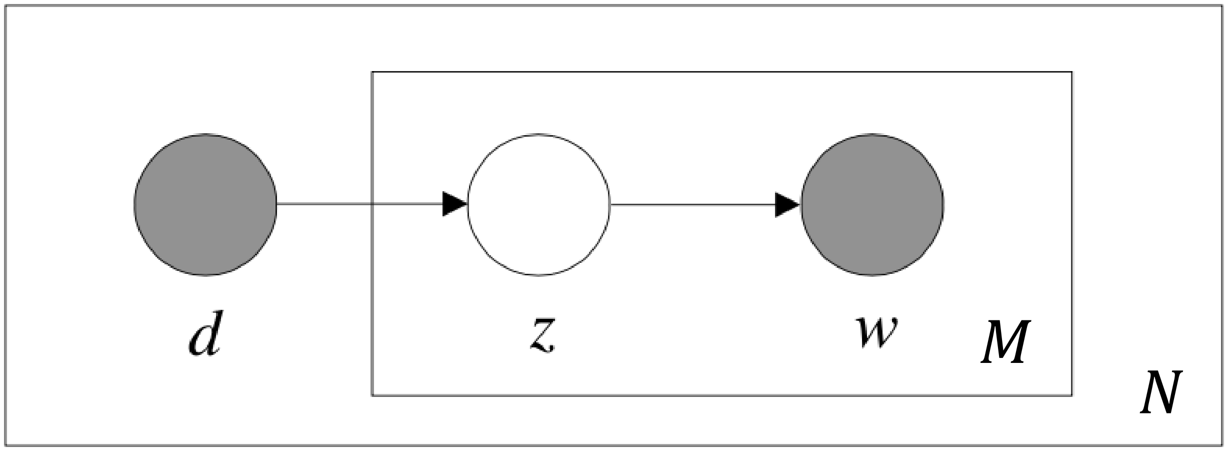
Topic summaries are NOT perfect.

➡ UTOPIAN allows user interactions for improving them.

Now, Let's Turn to Probabilistic Topic Modeling

- A topic is a probability distribution over the vocabulary.
- A document is generated from a topic.
  - That is, each word of a document is generated from a topic.
- Previously, we assumed that all the words of a single document is generated by a single topic.
  - e.g., computing the similarities between a topic and a document.
- Now, topic modeling assumes a different word can be generated from a different topic.

Probabilistic Latent Semantic Indexing (pLSI)



d: a document
z: a topic
w: a word
M: the number of words in a document d
N: the number of documents

- This shows the probabilistic generative (sampling) process.
  - Grey nodes: observed variables
  - White nodes: unobserved variables
  - Plates (rectangles): repetitive sampling process

Probabilistic Latent Semantic Indexing (pLSI)

- Joint probability for (w, d)

$$p(w,d) = \sum_{k=1}^{K} p(z_k) p(w|z_k) p(d|z_k)$$

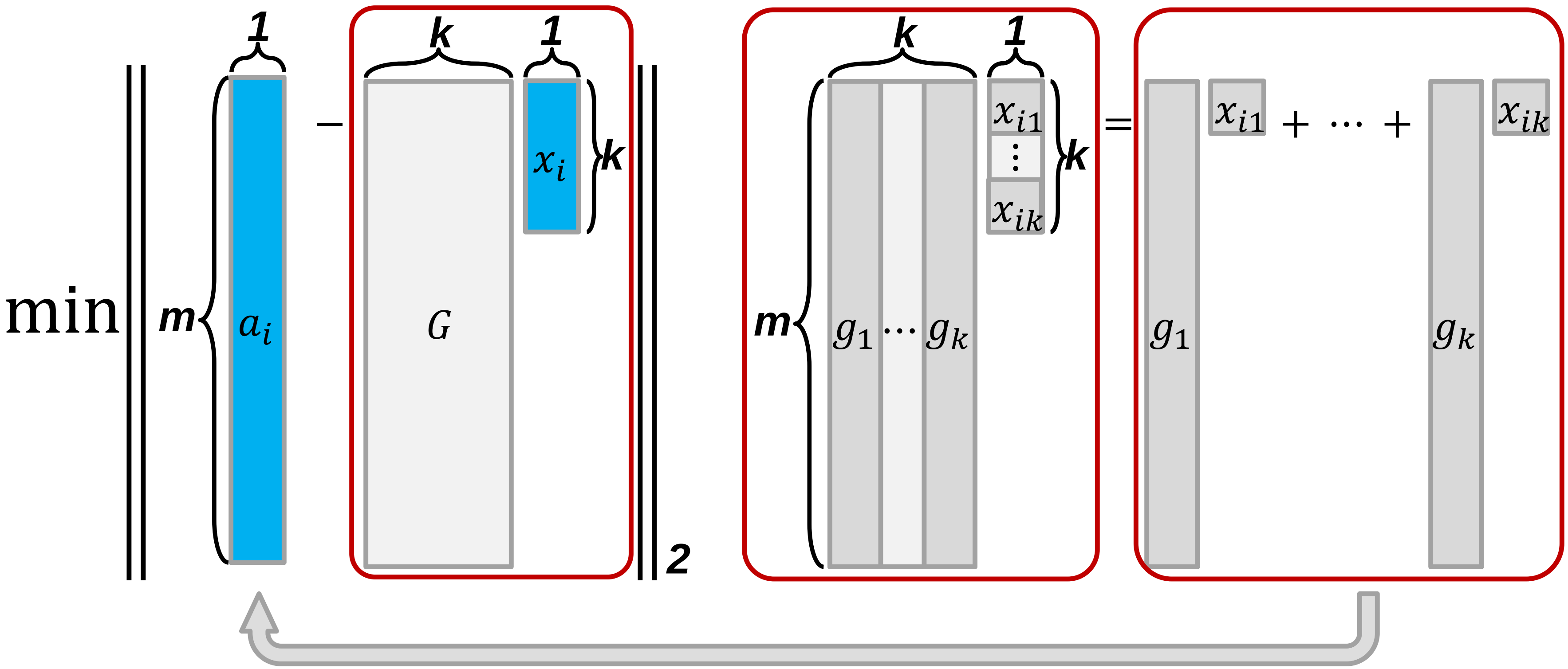
  - where K is the (pre-determined) number of topics.
  - Here, we assume that $p(w|d,z) = p(w|z)$, which assumes that generating a word is only dependent on its topic, but not its document.
- Or equivalently,

$$p(w|d) = \sum_{k=1}^{K} p(w|z_k) p(z_k|d)$$

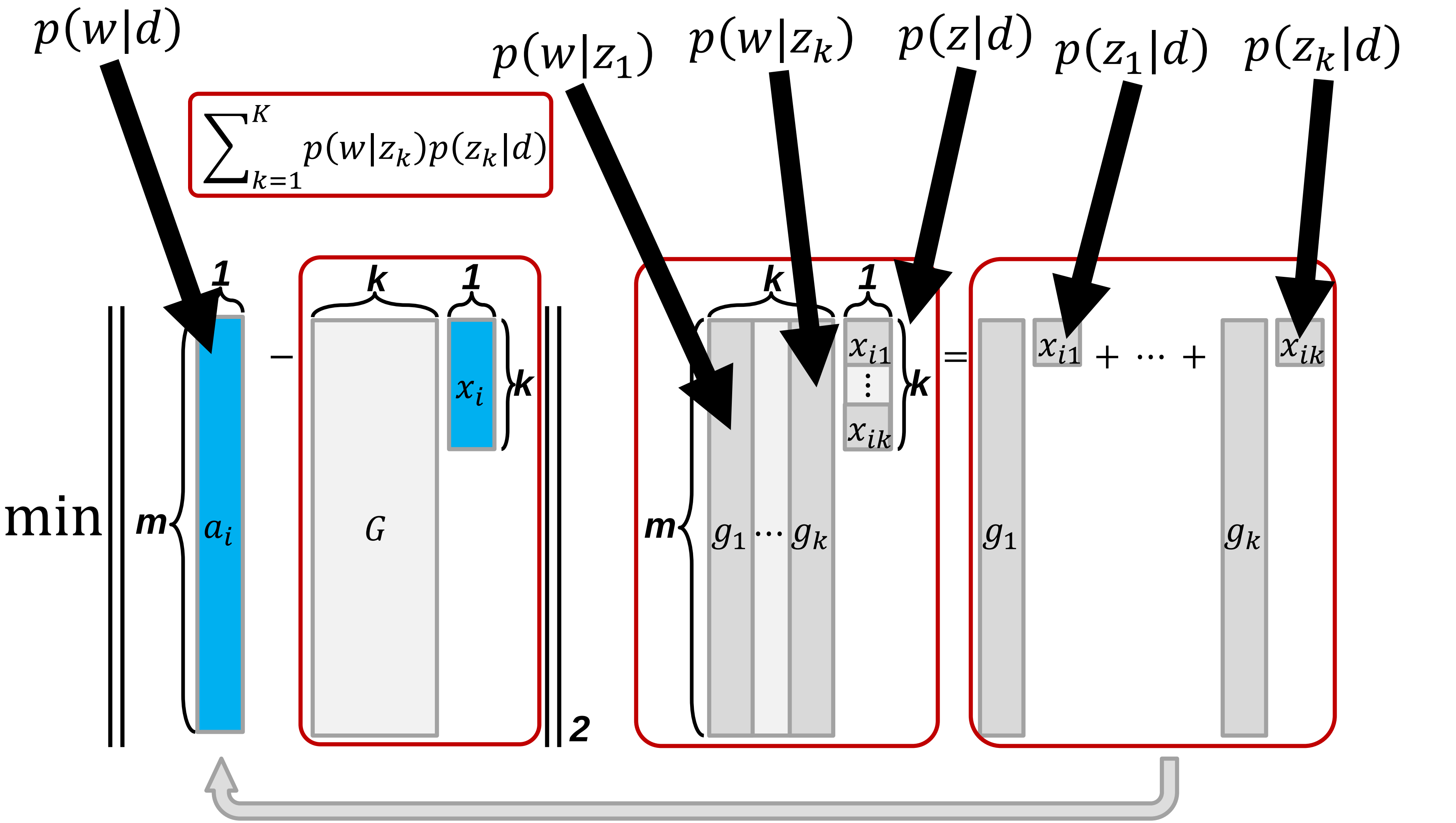
- A document is a mixture of topics. That is, the distribution of a document $p(z|d)$ is defined over topics.
- The distribution of a topic $p(w|z)$ is defined over words.

Recall: Intuition behind SVD

- A bag-of-words document vector is approximated as a linear combination of basis vectors or the columns of G.



In pLSI, …



We should make them as close as possible.

Maximizing Likelihood

- Let's write the log-likelihood function:

$$L = \sum_{i=1}^{N} \sum_{j=1}^{M} n(d_i, w_j) \boxed{\log p(d_i, w_j)}$$

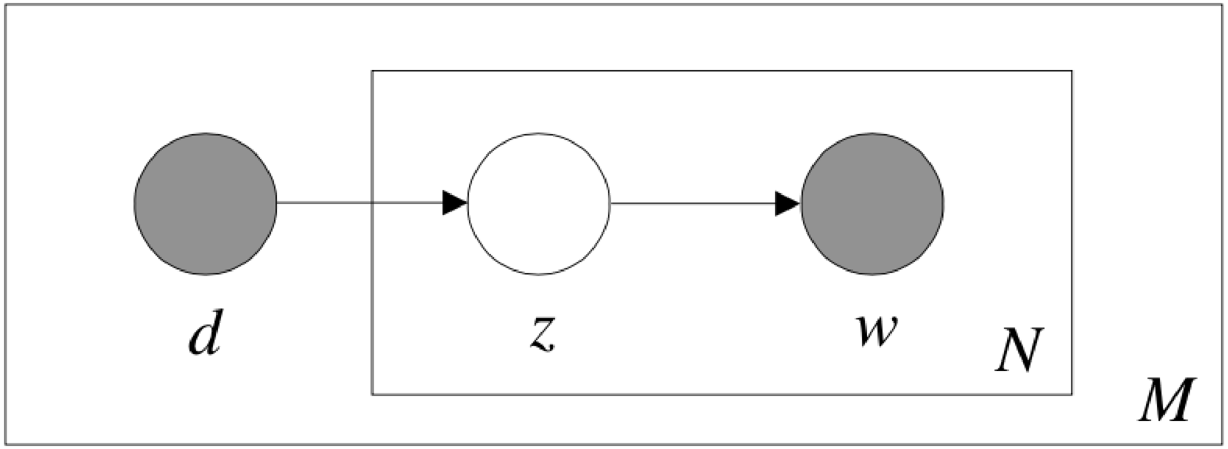
  - $n(d_i, w_j)$: the frequency of keyword w_j in document d_i.
- What we want to obtain is the parameters β and θ from maximizing the above function:
  - β: $p(w|z_k)$ for all $k = 1, \dots, K$.
  - θ: $p(z|d_i)$ for all $i = 1, \dots, N$
- By rewriting the log-likelihood function, we obtain

$$\sum_{i=1}^{N} \sum_{j=1}^{M} n(d_i, w_j) [\log p(d_i) + \log p(w_j|d_i)]$$

$$= \sum_{i=1}^{N} n(d_i) \left[\log p(d_i) + \sum_{j=1}^{M} \frac{n(d_i, w_j)}{n(d_i)} \log \boxed{\left(\sum_{k=1}^{K} p(w_j|z_k) p(z_k|d_i) \right)} \right]$$

  - $n(d_i)$: the number of keywords in document d_i

EM Algorithm



d: a document
z: a topic
w: a word
M: the number of words in a document d
N: the number of documents

Problem when maximizing the likelihood function:

- We have unobserved random variable such as z.

Two steps of EM (Expectation-Maximization) algorithm

- E-step: Estimating these unobserved random variables.
  - Filling unknown portions of data
- M-step: Maximizing the likelihood based on all the observed information.
  - It is the same as a typical maximum likelihood process.
- EM algorithm iterates between E- and M- steps until convergence.

EM Algorithm in pLSI

Again, log-likelihood is

$$\sum_{i=1}^{N} n(d_i)\left[\log p(d_i) + \sum_{j=1}^{M} \frac{n(d_i, w_j)}{n(d_i)} \log\left(\sum_{k=1}^{K} p(w_j|z_k)p(z_k|d_i)\right)\right]$$

1. E-step

- $$p(z_k|d_i, w_j) = \frac{p(z_k, w_j|d_i)}{\sum_{l=1}^{K} p(z_k, w_j|d_i)} = \frac{p(w_j|z_k)p(z_k|d_i)}{\sum_{l=1}^{K} p(w_j|z_l)p(z_l|d_i)}$$

2. M-step

- $$p(w_j|z_k) = \frac{p(w_j, z_k)}{p(z_k)} = \frac{p(w_j, z_k)}{\sum_{m=1}^{M} p(w_m, z_k)} = \frac{\sum_{i=1}^{N} n(d_i, w_j)p(z_k|d_i, w_j)}{\sum_{m=1}^{M}\sum_{i=1}^{N} n(d_i, w_m)p(z_k|d_i, w_m)}$$

- $$p(z_k|d_i) = \frac{p(z_k, d_i)}{p(d_i)} = \frac{p(z_k, d_i)}{\sum_{l=1}^{K} p(z_l, d_i)} = \frac{\sum_{j=1}^{M} n(d_i, w_j)p(z_k|d_i, w_j)}{n(d_i)}$$

Latent Dirichlet Allocation

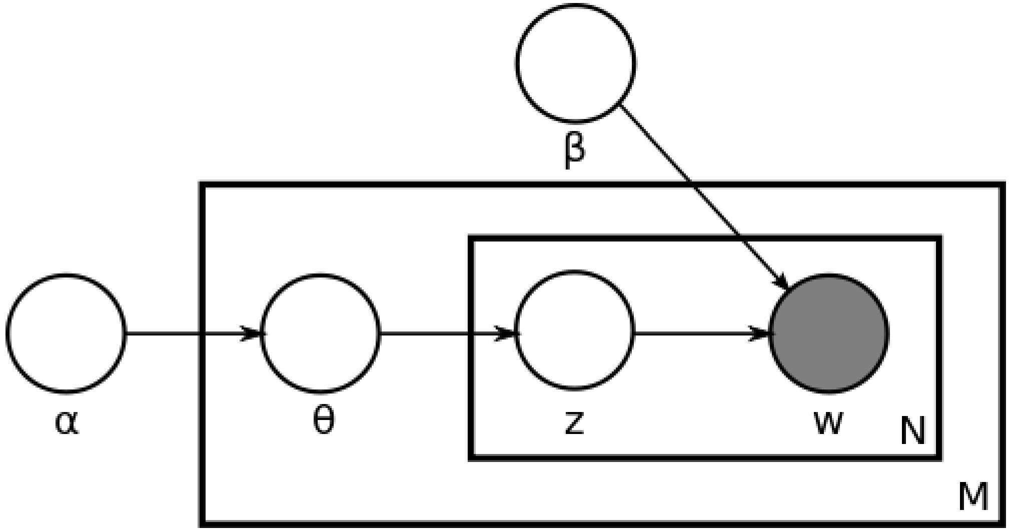
Recall:

- Another probabilistic topic modeling method

Key difference from pLSI:

- The distribution of a document over topics is not just determined independently per document.
- Instead, we impose a common prior probability distribution and sample this topic-wise distribution of a document from the prior.
  - The Dirichlet distribution is used as the prior probability distribution.

Generative Process of LDA



- For a document, its distribution over topics θ is sampled from $Dirichlet(\alpha)$.
- Then, for each word in the document
  - A topic z is sampled from $Multinomial(\theta)$.
  - A keyword w is sampled from $Multinomial\left(\beta_{z_k}\right)$, i.e., $p(w|z_k)$.
- EM Algorithm:
  - E-step: estimating θ and z
  - M-step: obtaining the parameter β by maximizing the likelihood

LDA on Reuters Data

Recall:

- 100-topic LDA on 16,000 documents
- Some standard stopwords are removed.
- Top keywords for some $p(w|z)$.

"Arts"	"Budgets"	"Children"	"Education"
new	million	children	school
film	tax	women	students
show	program	people	schools
music	budget	child	education
movie	billion	years	teachers
play	federal	families	high
musical	year	work	public

Low-Rank Approximation Viewpoint

- So far, we learned LSI, NMF, pLSI, and LDA, all of which can be viewed as a low-rank approximation of the original term-document matrix.
- So many other methods can be viewed in the same way.
- A notable recent method is called word2vec based on neural network, which considers sequence information of keywords.