2019 EE448, Big Data Mining, Lecture 8

# Search Engines

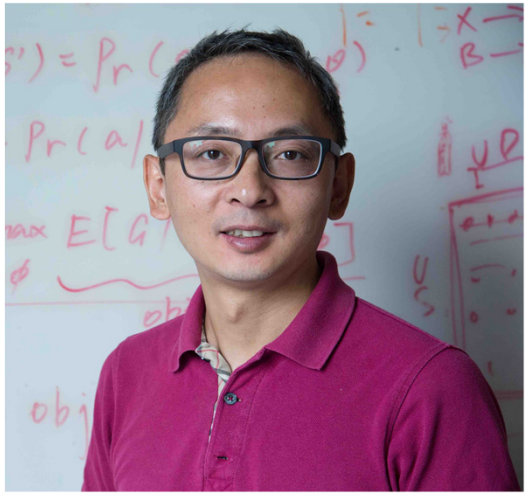
Weinan Zhang

Shanghai Jiao Tong University

http://wnzhang.net

http://wnzhang.net/teaching/ee448/index.html

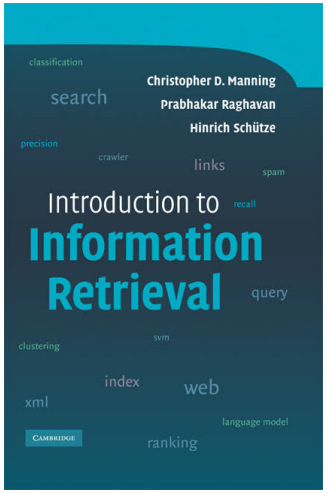
# Acknowledgement and References



- Dr. Jun Wang is the Chair Professor of Data Science and Founding Director of MSc Web Science and Big Data Analytics, Dept. of Computer Science, University College London (UCL)
- Most of slides in this lecture is based on Jun's Information Retrieval and Data Mining (IRDM) course at UCL

- Referred text book:

  Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. Introduction to Information Retrieval. Cambridge University Press. ISBN: 0521865719. 2008.



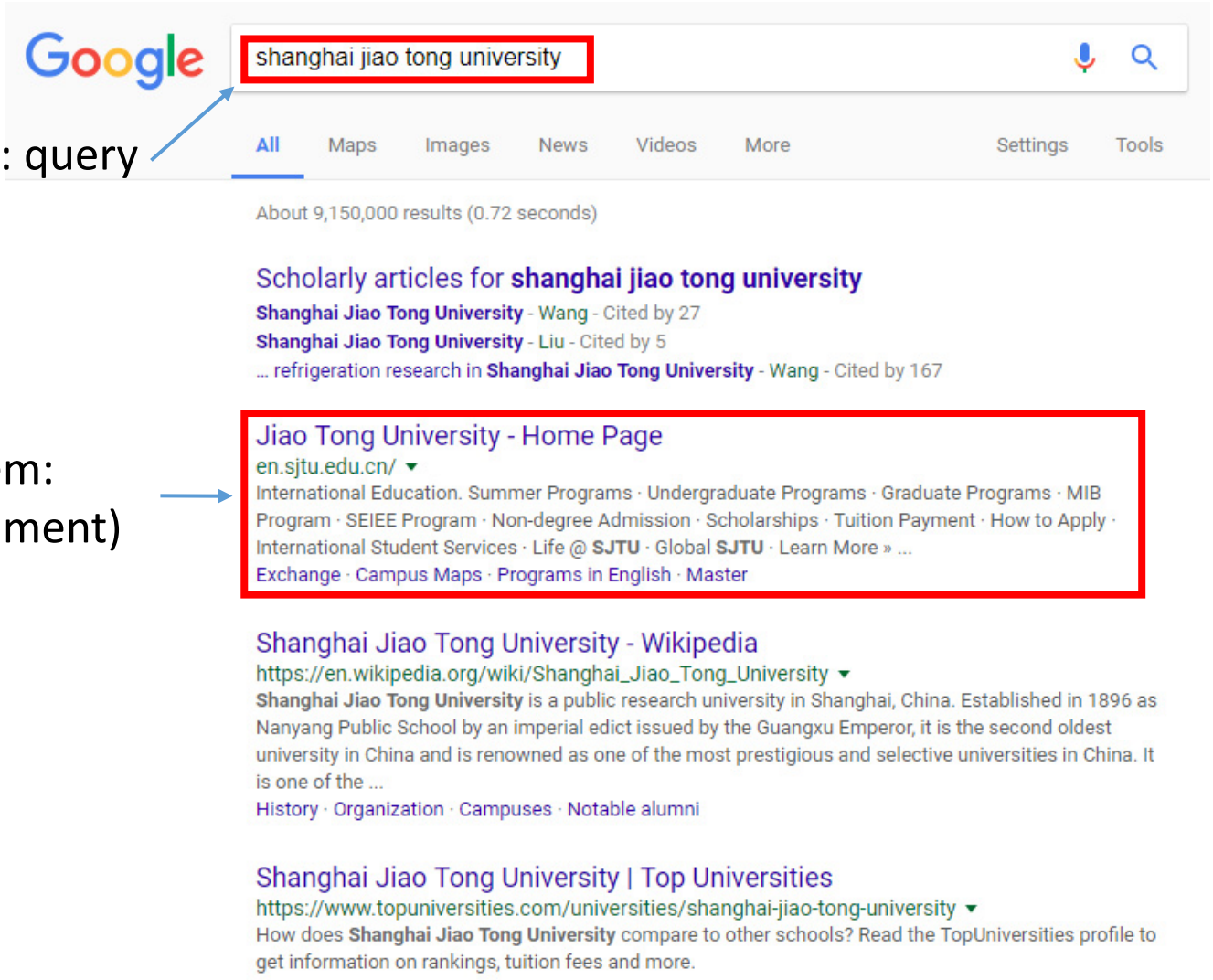
# Information Retrieval

- Information retrieval (IR) is the activity of obtaining information items relevant to an information need from a collection of information items.

# Web Search is the Typical Scenario of IR

Information need: query

Information item:
Webpage (or document)



# Other IR Scenarios

- Library Book Retrieval System
  - Information need: a book title, or an author name etc.
  - Information item: the book to seek for
- Recommender Systems
  - Information need: a user in a certain context (without query)
  - Information item: a move (music, product etc.) she would likes
- Search Advertising
  - Information need: a user with query keywords
  - Information item: a text ad she would click

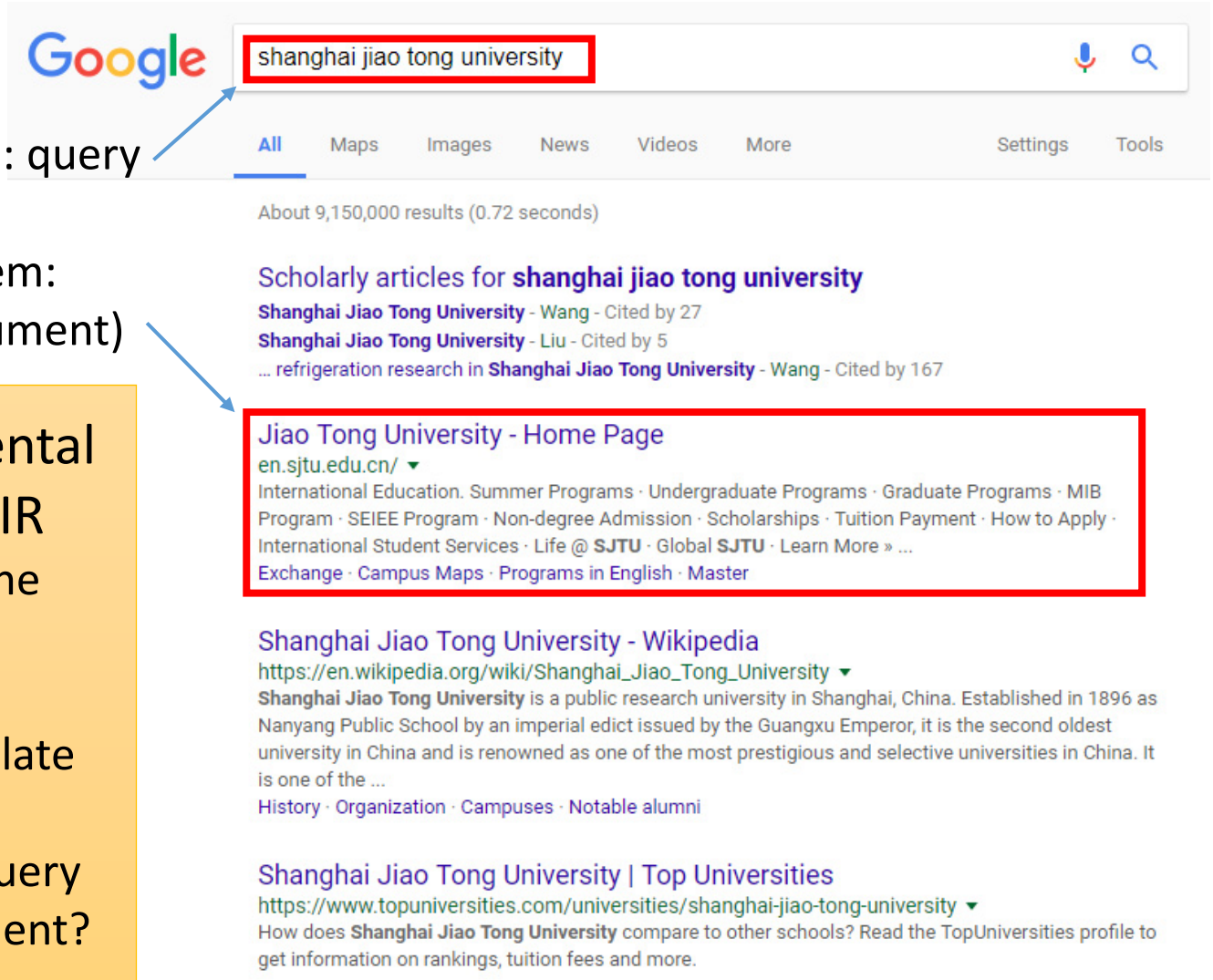
# Prof. Stephen Robertson


- Emeritus professor of University College London and City University London
- The pioneer of information retrieval
- The proposer of
  - Probabilistic Ranking Principle (1977)
  - BM25 (1980s)
  - Worked in Chengdu National Library in 1976!

# We Focus on Web Search Engines

Information need: query

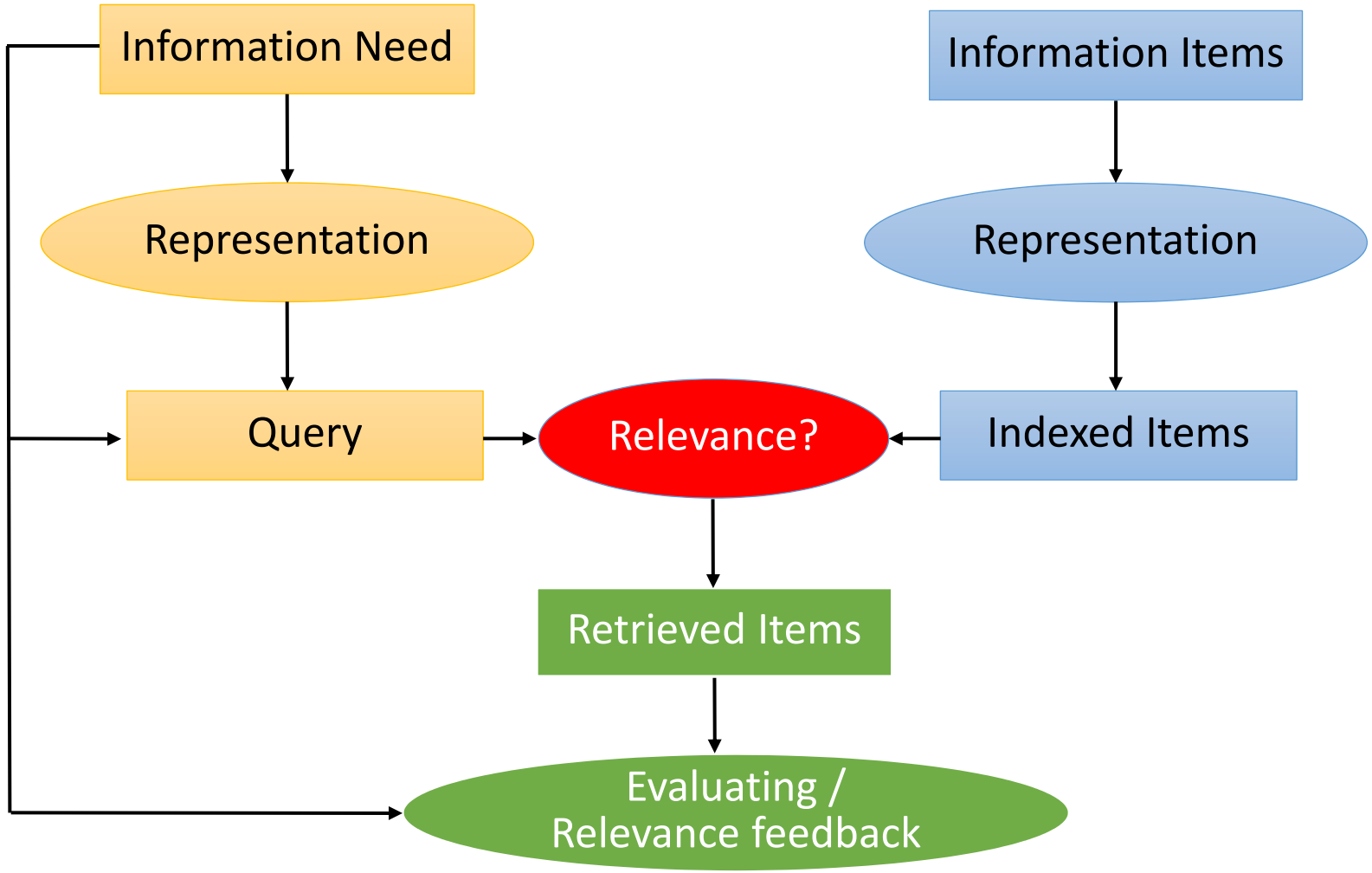
Information item: Webpage (or document)



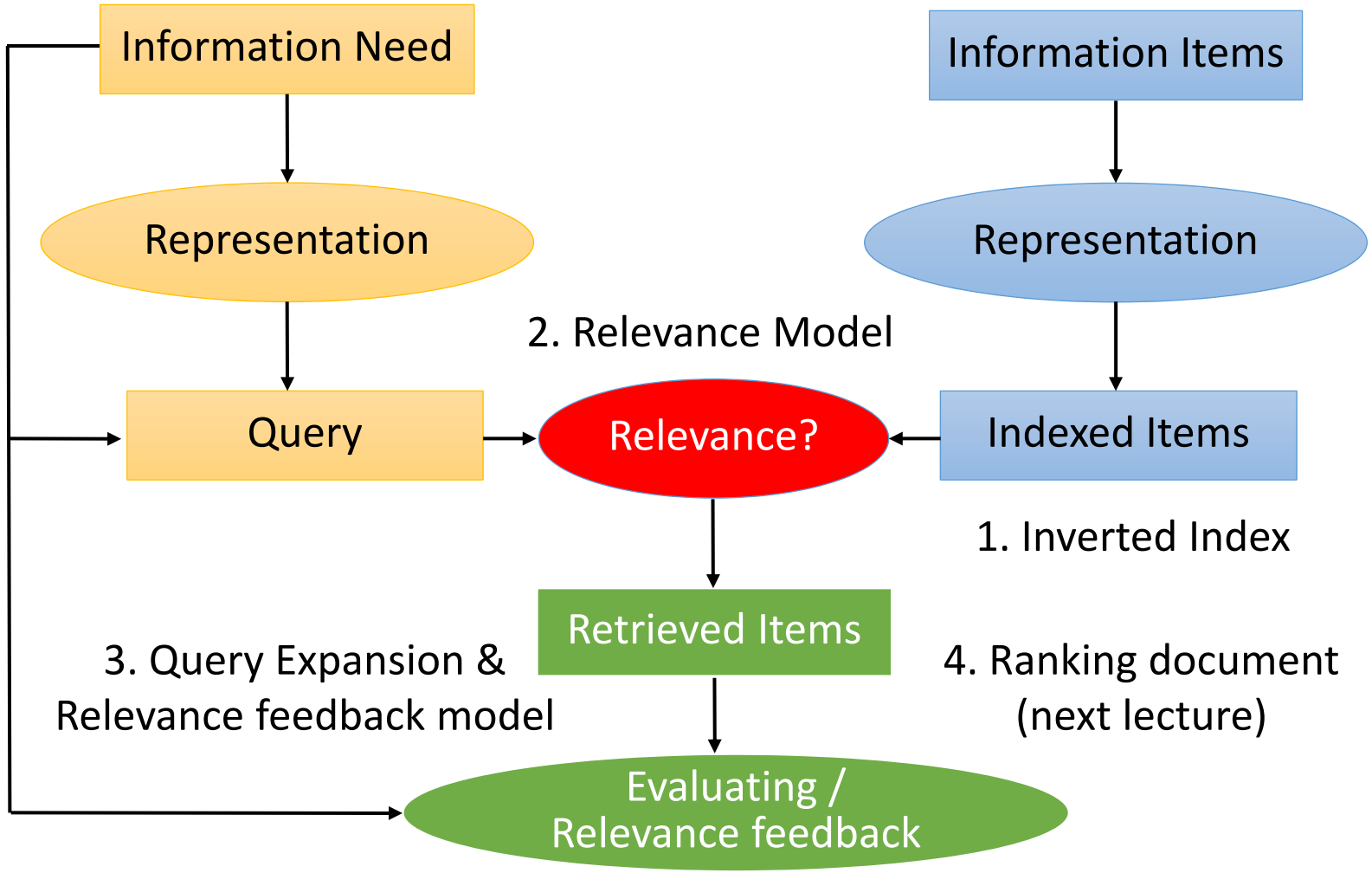

## Two fundamental problems for IR

- How to get the candidate documents?
- How to calculate relevance between a query and a document?

# Overview Diagram of Information Retrieval



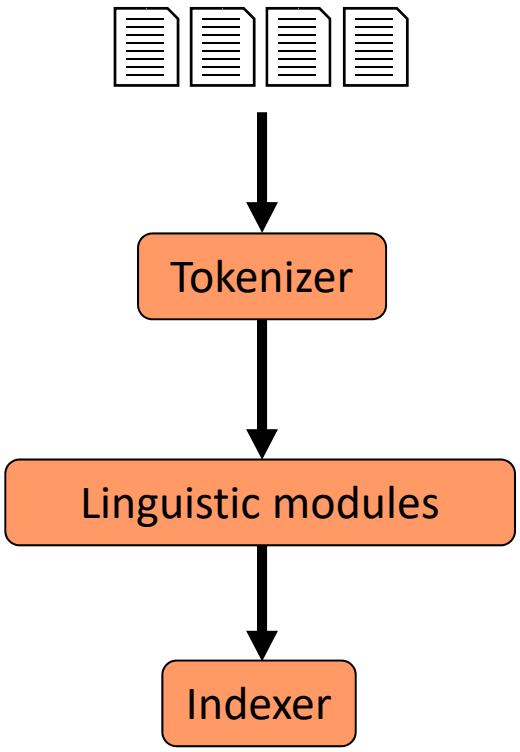
# Overview Diagram of Information Retrieval



# Content of This Lecture

- Inverted Index for Search Engine
- Relevance Models
- Query Expansion and Relevance Feedback

# Overall Indexing Pipeline




Documents to be indexed.

"Friends, Romans, countrymen, lend me your ears."

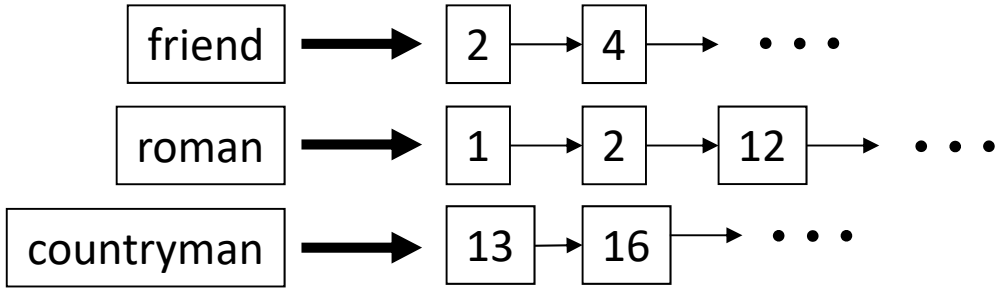
Token stream.

Friends | Romans | Countrymen

Modified tokens.

friend | roman | countryman

Inverted index.



# Tokenization

- Tokenization is the task of chopping a character sequence into the smallest units, called tokens
- It seems very easy: - Chop on whilespace and ignore punctuation characters
  - Input: *Friends, Romans, countrymen. So let . . .*
  - Output: *Friends Romans countrymen So let . . .*
- But, there are many tricky cases
  - Example *O'Neill → neill, oneill, or O neill*
  - How about *aren't , co-education, the While House*
- Need to do the exact same tokenization of document and query terms
  - Guarantee that a sequence of characters in a text will match the same sequence typed in a query
- Tokenization of other languages
  - E.g., Chinese (word segmentation)

# Normalization with Linguistic Models

- Normalize terms in indexed text and query terms into the same form
- Words can appear in different forms
- Need some way to recognize common concept
  - Examples:

  how to match ***U.S.A*** and ***USA*** → remove punctuation

  ***walking vs. walks*** → stemming

  ***Retrieval vs. retrieval*** → case folding

# Normalization: Case Folding

- Reduce all letters to lower case
  - ***Retrieval → retrieval***
  - ***ETHICS → ethics***
  - ***MIT → mit***
- Possible exceptions: capitalized words in mid-sentence
- It is often best to lowercase everything since users will use lowercase regardless of correct capitalization

# Normalization: Stemming

- Stemming is a technique to reduce morphological variants of search terms
- Stem: portion of a word which is left after the removal of its affixes
  - *walk ← walked, walker, walking, walks*
  - *be ← am, are, is*
  - *cut ← cutting*
  - *destroy ← destruction*
- Significantly reduce the number of the index terms
- Increase recall while harming precision

# Porter Algorithm for Stemming

- One of the most common stemming algorithms in English
  - Conventions plus five phases of reductions
  - Phases are applied sequentially
  - Each phase consists of a set of commands
- A few rules in phase 1 (apply sequentially)

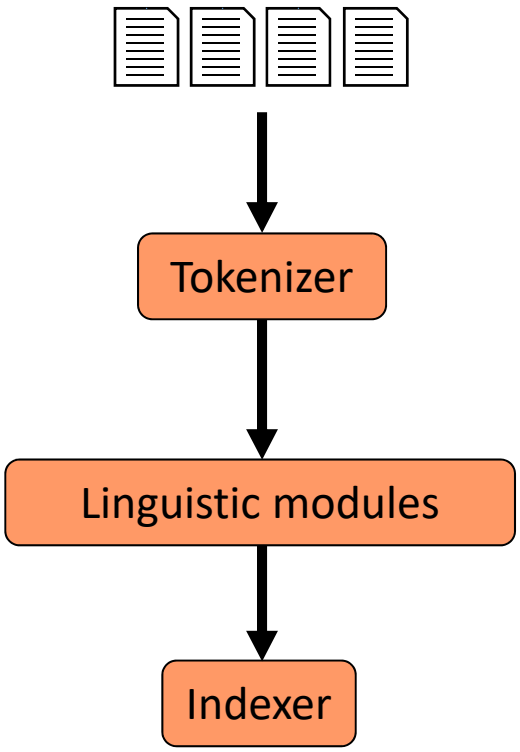
| **Rule** | **Example** |
|---|---|
| SSES → SS | caresses → caress |
| IES → | Ponies → poni |
| SS→SS | caress → caress |
| S→ | cats→ cat |

https://tartarus.org/martin/PorterStemmer/

# Normalization: Stop Words

- Drop some extremely common words from the vocabulary because they are of little value in helping selecting documents
  - examples*: "the", "a", "by", "will" ...*
- Take the most frequent terms (by *collection frequency*) to construct the stop word list
  - e.g., remove word that appears in more than 5% of documents
- Perhaps remove numbers and dates. However, these might be very useful
- Produce a considerable reduction of the index terms. Results: smaller index files and faster search
- Most web search engines index stop words

# Overall Indexing Pipeline




Documents to be indexed.

"Friends, Romans, countrymen, lend me your ears."

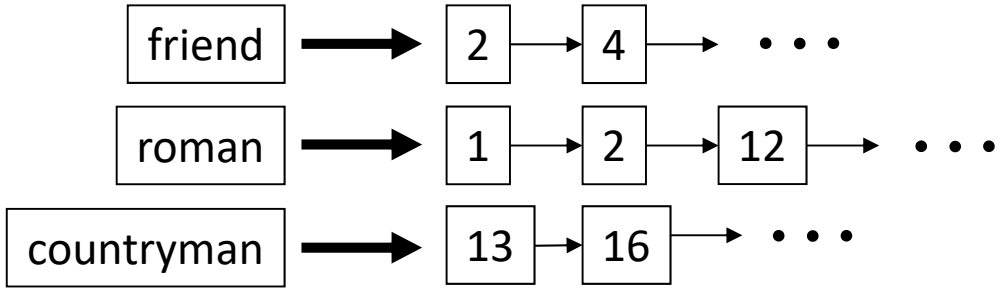
Token stream.

Friends | Romans | Countrymen

Modified tokens.

friend | roman | countryman

Inverted index.



# Indexing the Documents

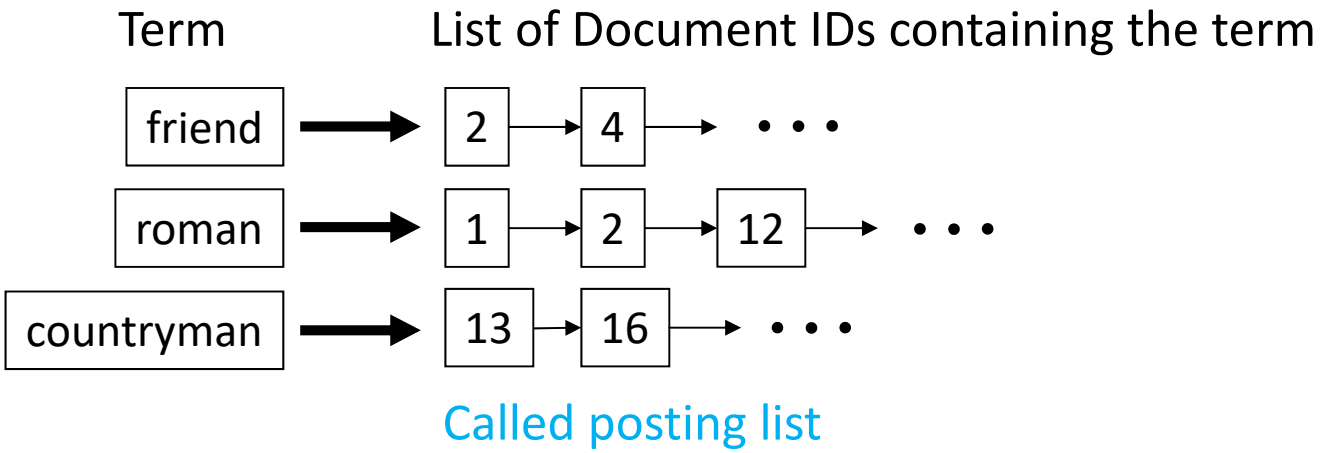
- Key Problem: given a query, how to obtain the candidates from the massive number of documents
- Solution: indexing the documents for IR
- The difficulties in IR
  - Indexing “titles”, “abstract”, etc. only does not support content-based retrieval; document contents are, in most case, unstructured.
  - Cannot predict the terms that people will use in queries - every word in a document is a potential search term
- A solution: index all terms in the documents
  - Full text indexing

# Data Access

- Scan the entire document collection
  - Typically used in early retrieval systems
  - Still popular today, e.g., grep command in Linux - “slow”; need real-time process
  - Practical for “small” collections
- Index (query) terms for direct access
  - An index associates each of the keys (normally terms) with one or more documents
  - “Fast”; practical for “large” collection
- Hybrid approaches - Use small index and then scan a subset of collection

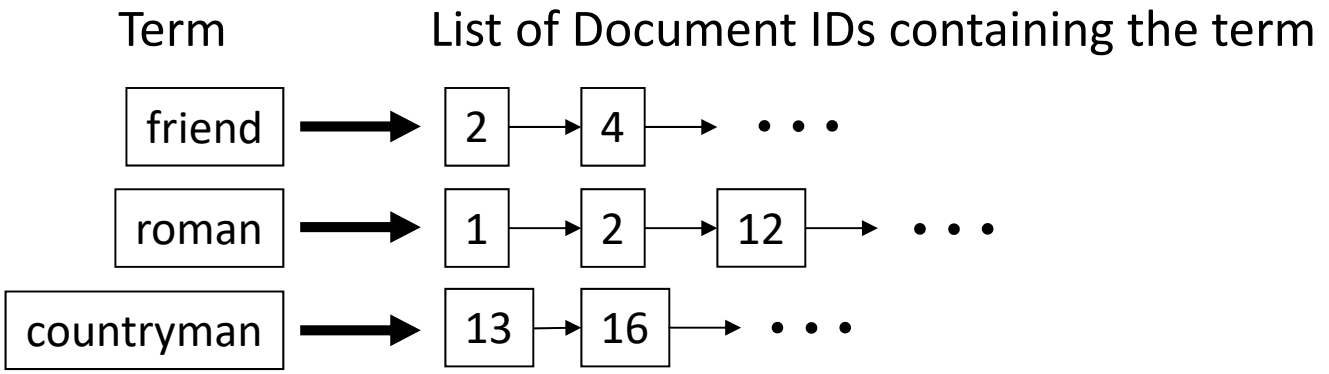
# Inverted Index

- Inverted index is the most common indexing technique
- Collection organized by terms (words). One record per term, listing locations (doc. IDs) where term occurs. May have more information.
- During retrieval, traverse lists for each query term



# Inverted Index

- Different terms have vastly different sizes of posting lists
  - E.g. on Google, ‘information’ has 2,990M documents, while ‘bayesian’ has 17M
- We need variable-size postings lists
  - On disk, a continuous run of postings is normal and best
  - In memory, can use linked lists or variable length arrays
    - Some tradeoffs in size/ease of insertion




Called posting list, sorted by IDs (why?)

# Steps of Building Inverted Index

Document 1

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

Document 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 |

- Step 1: extract the sequence of (modified term, document ID) pairs.

# Steps of Building Inverted Index

Document 1

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

Document 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 |

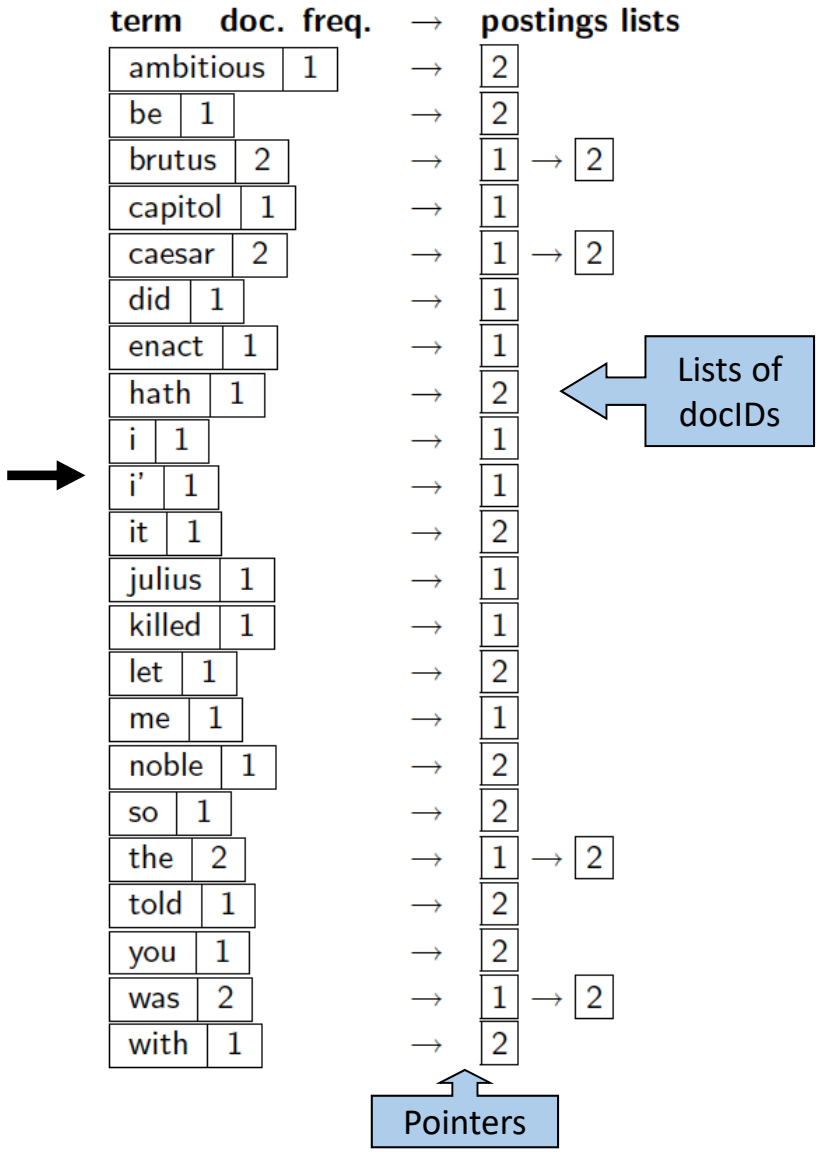
| Term | docID |
|---|---|
| ambitious | 2 |
| be | 2 |
| brutus | 1 |
| brutus | 2 |
| capitol | 1 |
| caesar | 1 |
| caesar | 2 |
| caesar | 2 |
| did | 1 |
| enact | 1 |
| hath | 1 |
| I | 1 |
| I | 1 |
| i' | 1 |
| it | 2 |
| julius | 1 |
| killed | 1 |
| killed | 1 |
| let | 2 |
| me | 1 |
| noble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

- Step 1: extract the sequence of (modified term, document ID) pairs.
- Step 2: sort by terms and then docID
  - Core indexing step

# Steps of Building Inverted Index

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

| 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 |

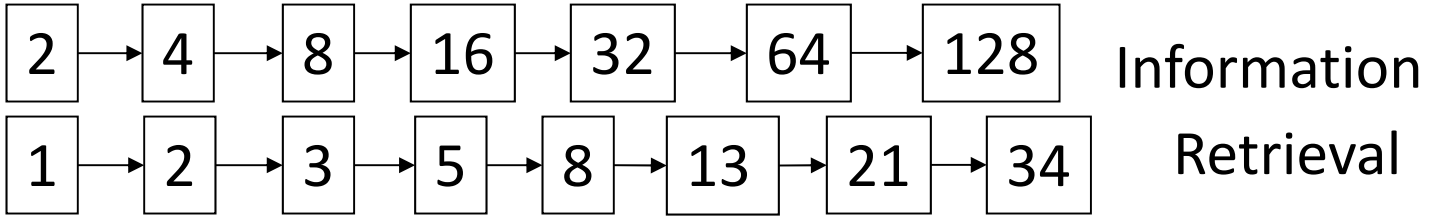


# Query Processing: AND

- Consider processing the query:

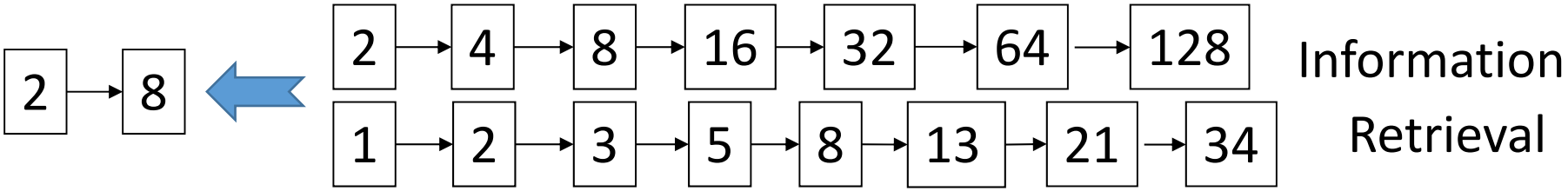
  'Information' AND 'Retrieval'

  - Locate 'Information' in the dictionary;
    - Retrieve its postings.
  - Locate 'Retrieval' in the dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:



# Merging the Posting Lists

- Walk through the two postings simultaneously, in time linear in the total number of postings entries




- If list lengths are *x* and *y*, merge takes *O*(*x*+*y*) operations.
- Crucial: postings sorted by docID.

# Phrase Queries

- Want to be able to answer queries such as "Shanghai Jiao Tong University" – as a phrase
  - Note that it is different from search Shanghai AND Jiao AND Tong AND University (why?)
- Thus the sentence "I went to Xi'an Jiao Tong University from Shanghai" is not a match.
  - The concept of phrase queries has proven easily understood by users
  - Many more queries are *implicit phrase queries*
- For this purpose, it no longer suffices to store only <*term*: *docs*> entries

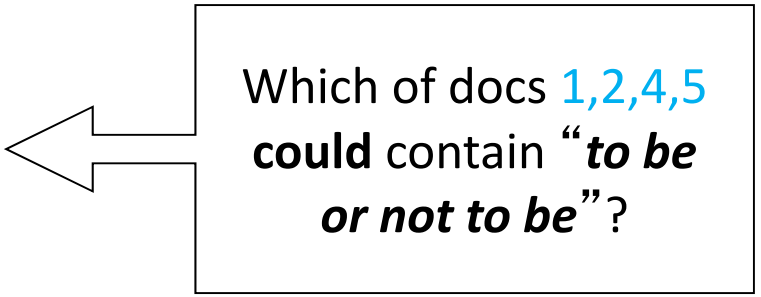
# Positional Indexes

- 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 … ;
…>

# Positional Index Example

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




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

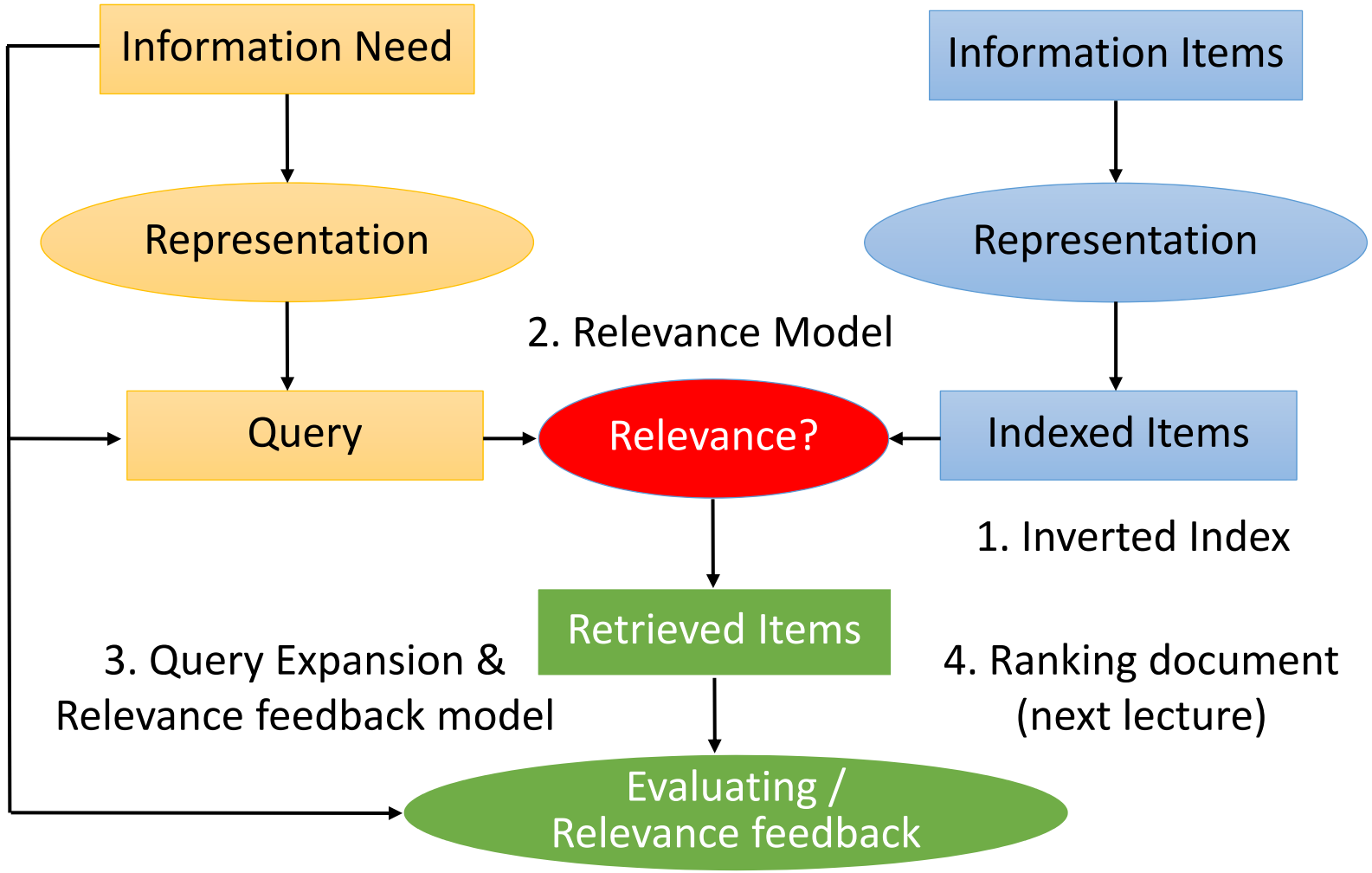
# Processing a Phrase Query

- 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; …
  - ***Or:***
    - 3:34,71; *4*:31,341,510; 8:31,420,551; …

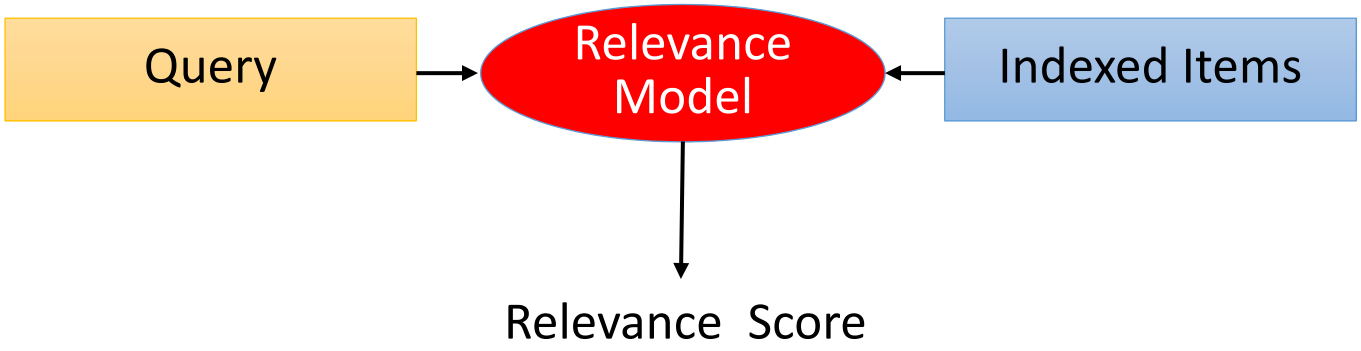
# Content of This Lecture

- Inverted Index for Search Engine
- Relevance Models
- Query Expansion and Relevance Feedback

# Overview Diagram of Information Retrieval



# Relevance Model




- Estimate the relevance between a query and a document
- Relevance is the "correspondence" between information needs (queries) and information items (documents, webpages, images etc.)
- But, the exact meaning of relevance depends on applications:
  - = usefulness
  - = aboutness
  - = interestingness
  - = ?
- Predicting relevance is the central goal of IR

# Representation of Information Need/Items

- We consider textual queries and documents
  - Boolean:
    - “(information AND retrieval) OR (machine AND learning)”
  - Free text: “movie matrix review”

- A bag-of-words representation
  - the item (query or document) is the “bag”
  - the bag contains word tokens
  - word order is ignored

REVIEW

# Bag-of-Words Representation for Text

- A sequence of words/tokens that represents semantic meanings of human

```
Text mining, also referred
to as text data mining,
roughly equivalent to text
analytics, is the process
of deriving high-quality
information from text.
```

Bag-of-Words Format:

```
{
    text: 4;
    mining: 2;
    also: 1;
    referred: 1;
    to: 2;
    as: 1;
    data: 1;
    roughly: 1;
    equivalent: 1;
    analytics: 1;
    is: 1;
    the: 1;
    process: 1;
    of: 1;
    deriving: 1;
    high-quality: 1;
    information: 1;
    from: 1;
}
```

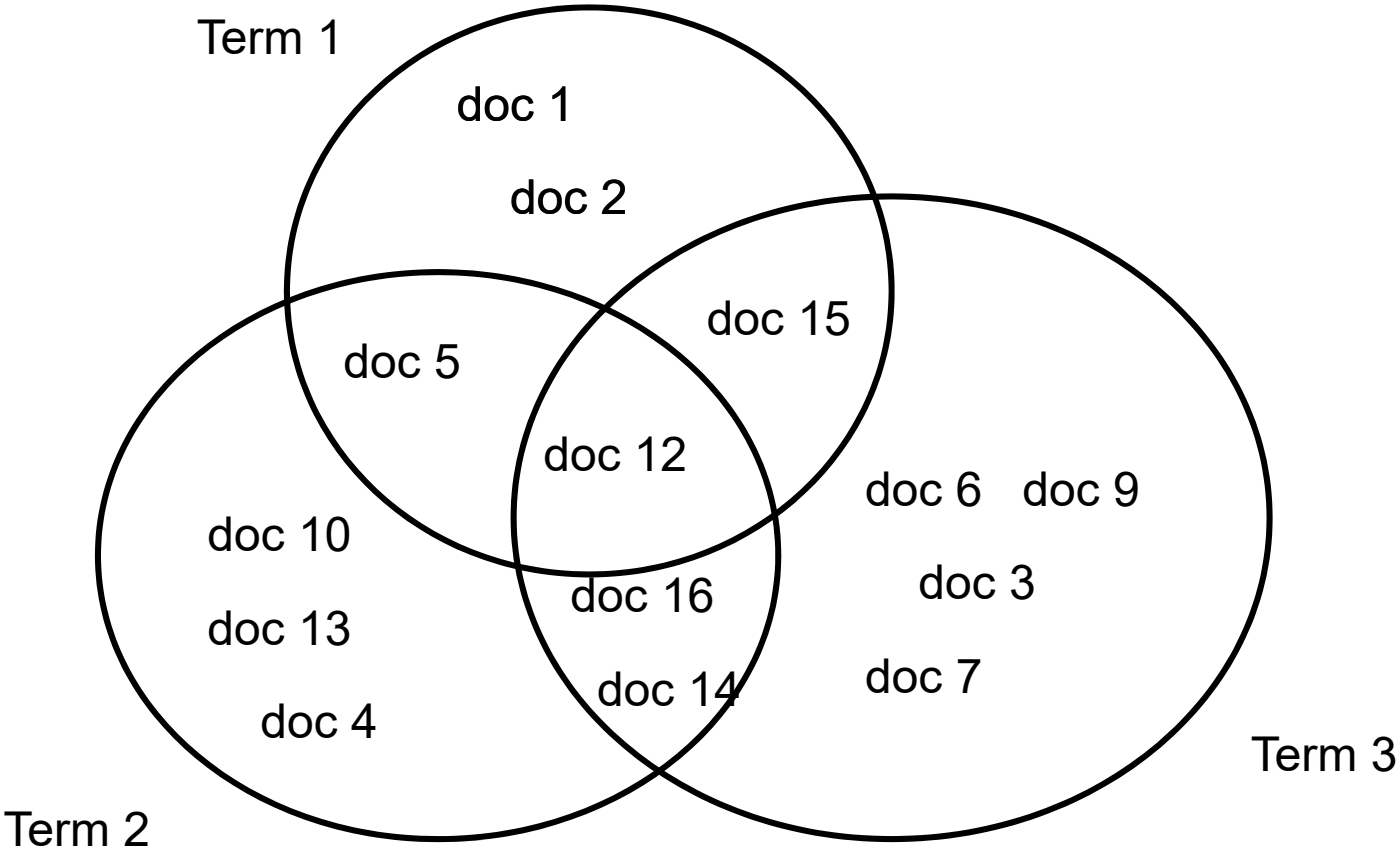
# Boolean Retrieval

- The simplest Exact Match model
  - Retrieve documents iff they satisfy a Boolean expression
  - Query specifies precise relevance criteria
  - Documents returned in no particular order
- Document: A bag of words
- Query: A Boolean expression
- Operators:
  - Logical operators:  AND, OR, AND NOT
  - Proximity  operators:  number of intervening words between two query terms, etc.
  - String matching operators:  Wild-card

# Boolean Retrieval

- Boolean logic:

Query: term 1 AND term 2 AND NOT term 3
retrieve doc 5



# Boolean Retrieval: Summary

- Advantages
  - Works great if you know exactly what you want
  - Structured queries
  - Simple to program
  - Complete expressiveness
- Disadvantages
  - Artificial language – unintuitive, misunderstood
  - Either too precise or too loose (the size of the output)
  - Unordered output: have to examine all of the results

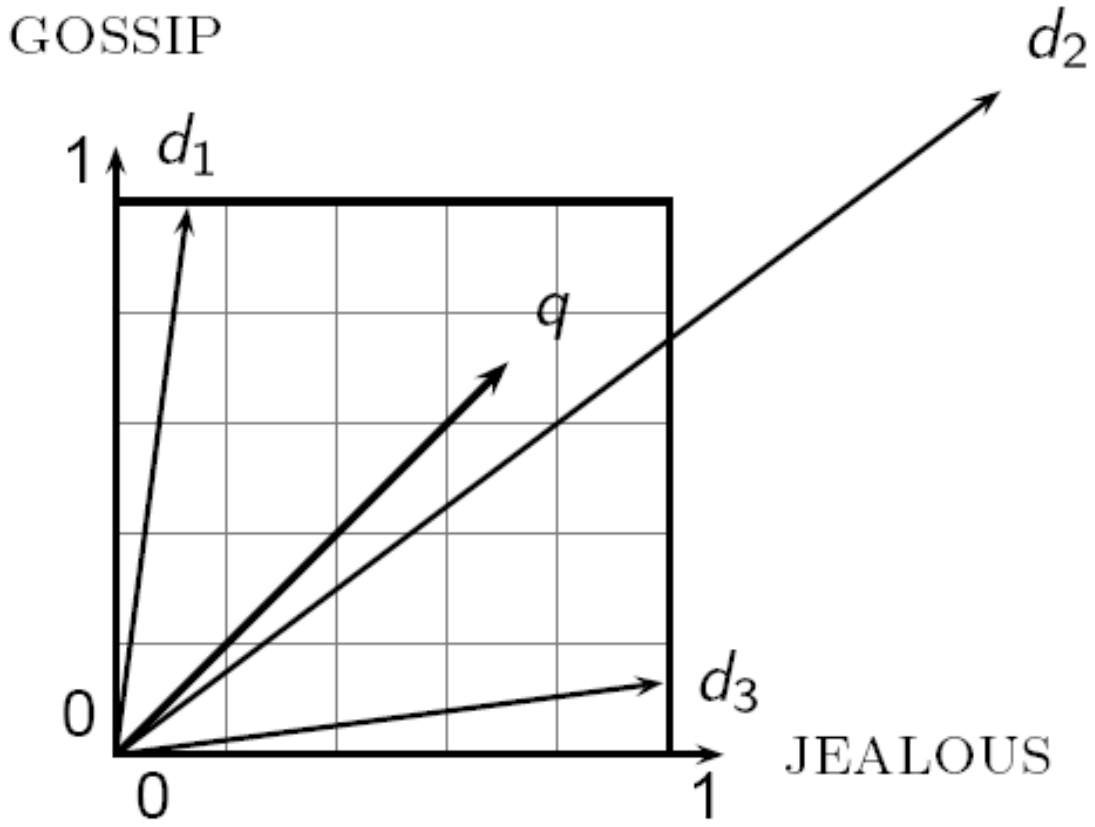
# Vector Space Model

- Regarding queries and documents as vectors
  - We have a $|V|$-dimensional vector space, where $|V|$ is the vocabulary size
  - Terms are axes of the space
  - Queries and 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 (as mentioned in inverted index part)

# Formalizing Vector Space Proximity

- We need to come up with a 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



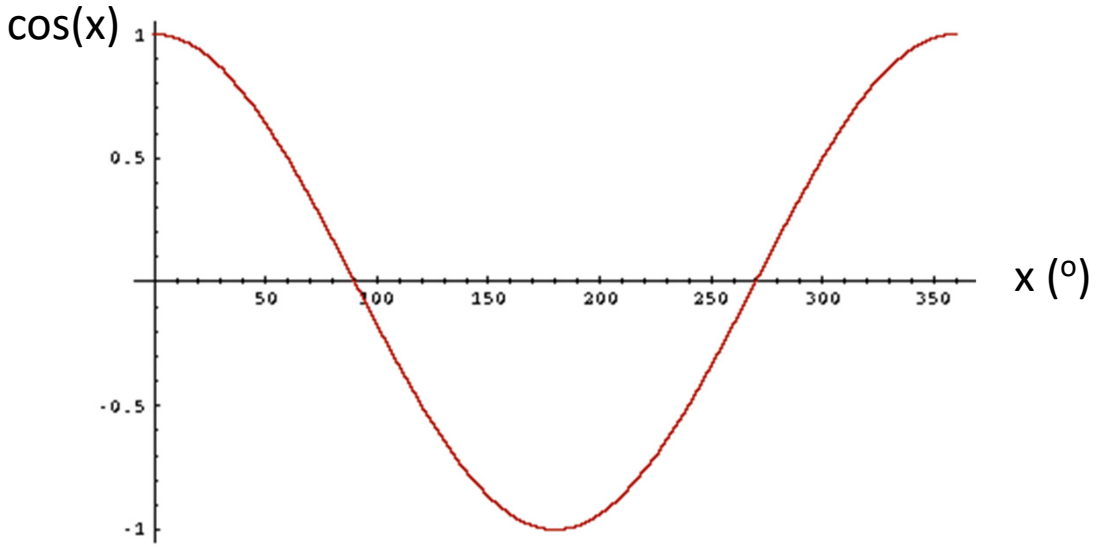

- The Euclidean distance between $q$ and $d_2$ is large even though the distribution of terms in the query $q$ and the distribution of terms in the document $d_2$ are very similar.

# 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.

# Cosine Similarity

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



# Cosine(query, document)

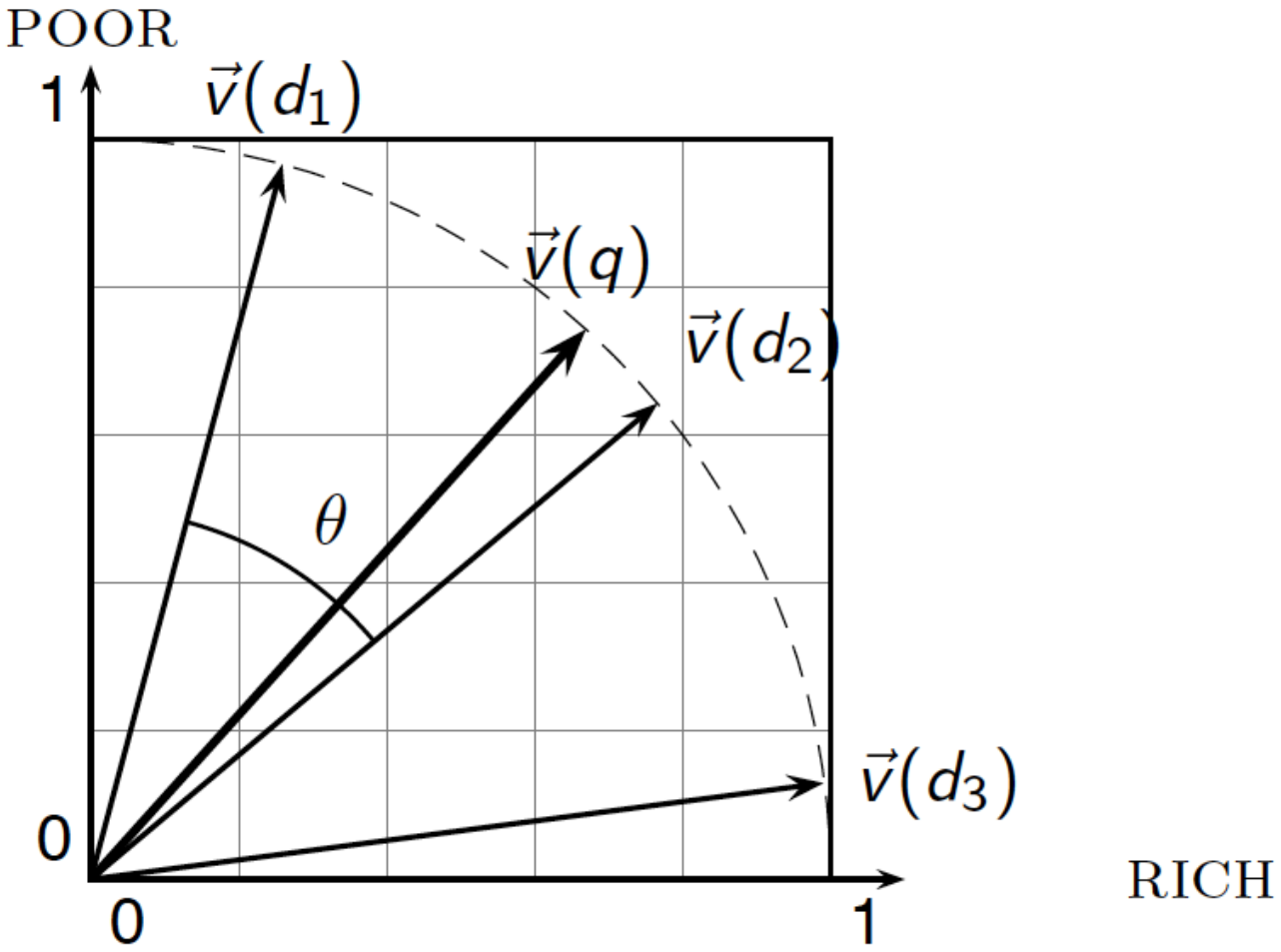
- $q_i$ is the weight of term *i* in the query
- $d_i$ is the weight of term *i* in the document

- cos(*q*,*d*) is the cosine similarity of *q* and *d* … or,
- equivalently, the cosine of the angle between *q* and *d*.

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

Unit Vectors

# Cosine Similarity Illustrated



# TF·IDF Term Weighting

- $q_i$ *and* $d_i$ are can be beyond just binary values nor term frequency values
- TF·IDF term weighting
  - $TF_{i,d}$ : term frequency of term *i* in the document
  - $IDF_i$ : inverse document frequency of term *i* in the document set

$$IDF_i = \log_{10} \frac{N}{n_i} \qquad TFIDF_{i,d} = TF_{i,d} \log_{10} \frac{N}{n_i}$$

$$\text{score}(q,d) = \sum_{i \in q \cap d} TFIDF_{i,d}$$

- TF·IDF term weighting has many variants
  - TF: $1+\log_{10}(\text{TF})$, bool etc.
  - IDF: $\log_{10}[(N-n_i+0.5)/(n_i+0.5)]$

# Okapi BM25 Term Weighting

- Consider document length in words $|d|$
- BM (Best Match) 25 Term weighting
  - $TF_{i,d}$ : term frequency of term *i* in the document
  - $IDF_i$ : inverse document frequency of term *i* in the document set
  - $\bar{d}$ : average document word length in the document set
  - $k_1$ and *b*: constant parameters

$$BM25_{i,d} = \frac{TF_{i,d} \cdot (k_1 + 1)}{TF_{i,d} + k_1 \cdot \left(1 - b + b \cdot |d|/\bar{d}\right)} \cdot IDF_i$$

$$\text{score}(q, d) = \sum_{i \in q \cap d} BM25_{i,d}$$

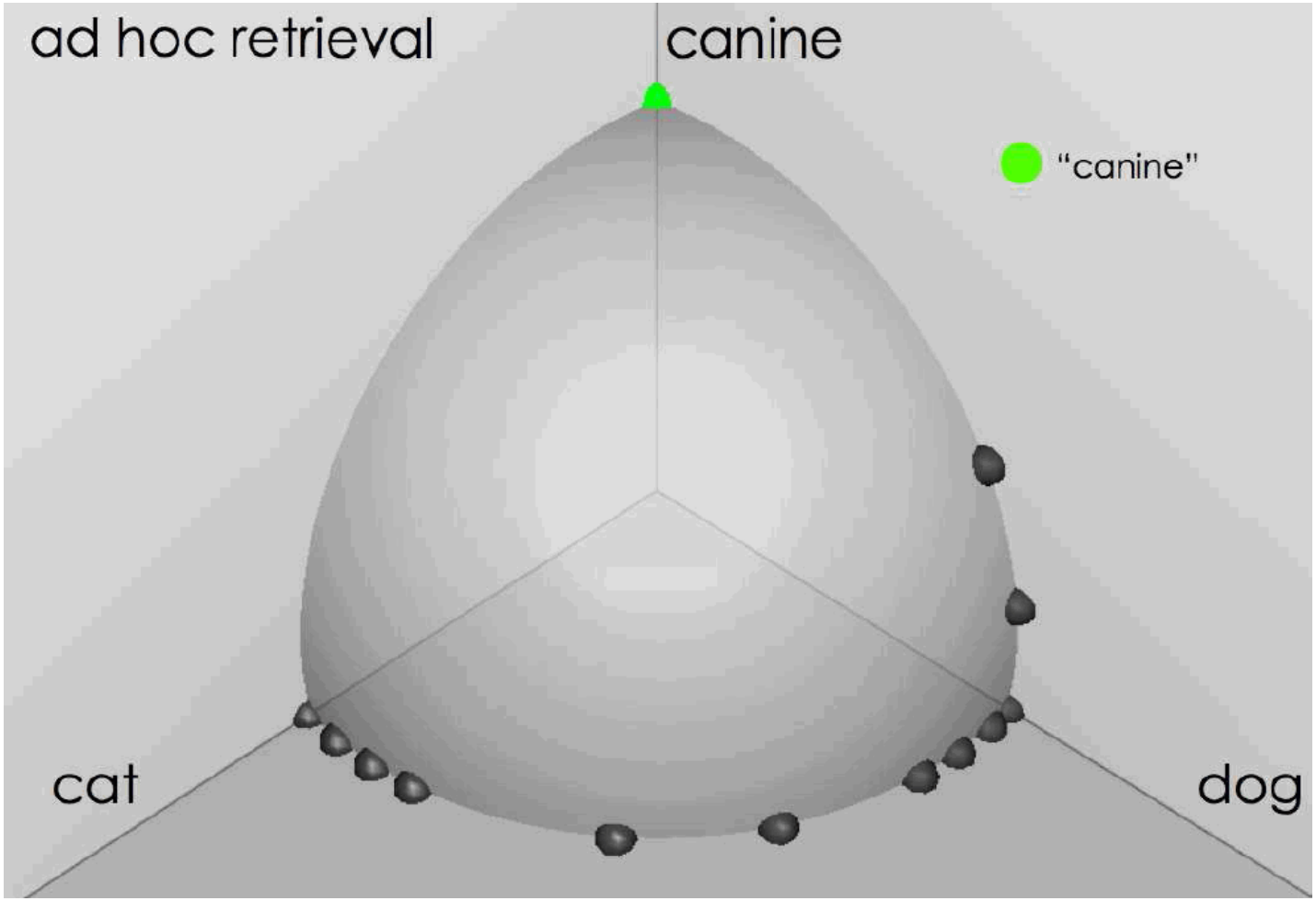
# Content of This Lecture

- Inverted Index for Search Engine
- Relevance Models
- Query Expansion and Relevance Feedback

# Relevance Feedback

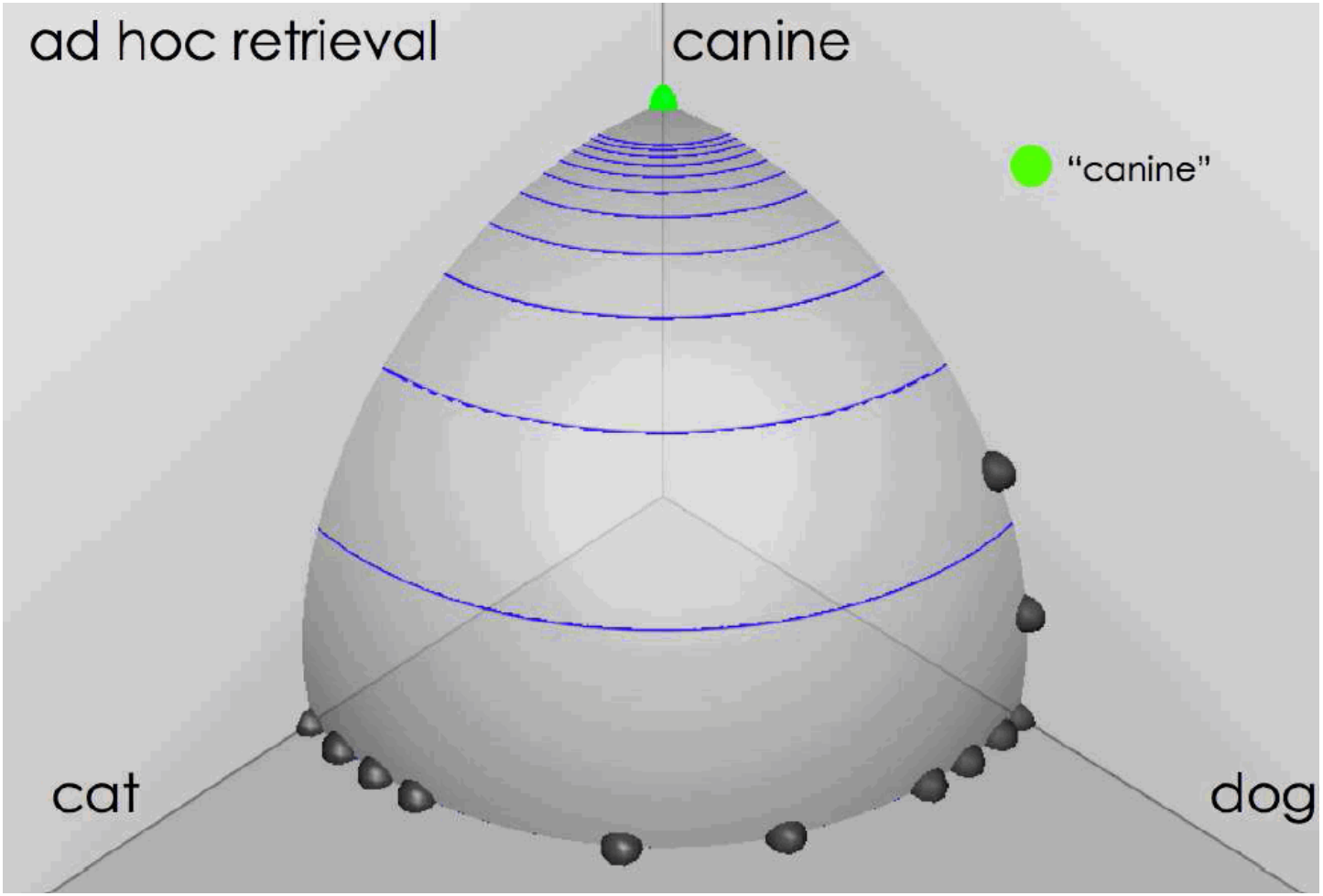
- Relevance feedback: user feedback on relevance of docs in initial set of results
  - User issues a (short, simple) query
  - The user marks some results as relevant or non-relevant.
  - The system computes a better representation of the information need based on feedback.
  - Relevance feedback can go through one or more iterations.
- Idea: it may be difficult to formulate a good query when you don't know the collection well, so iterate

# Ad hoc results for query *canine*



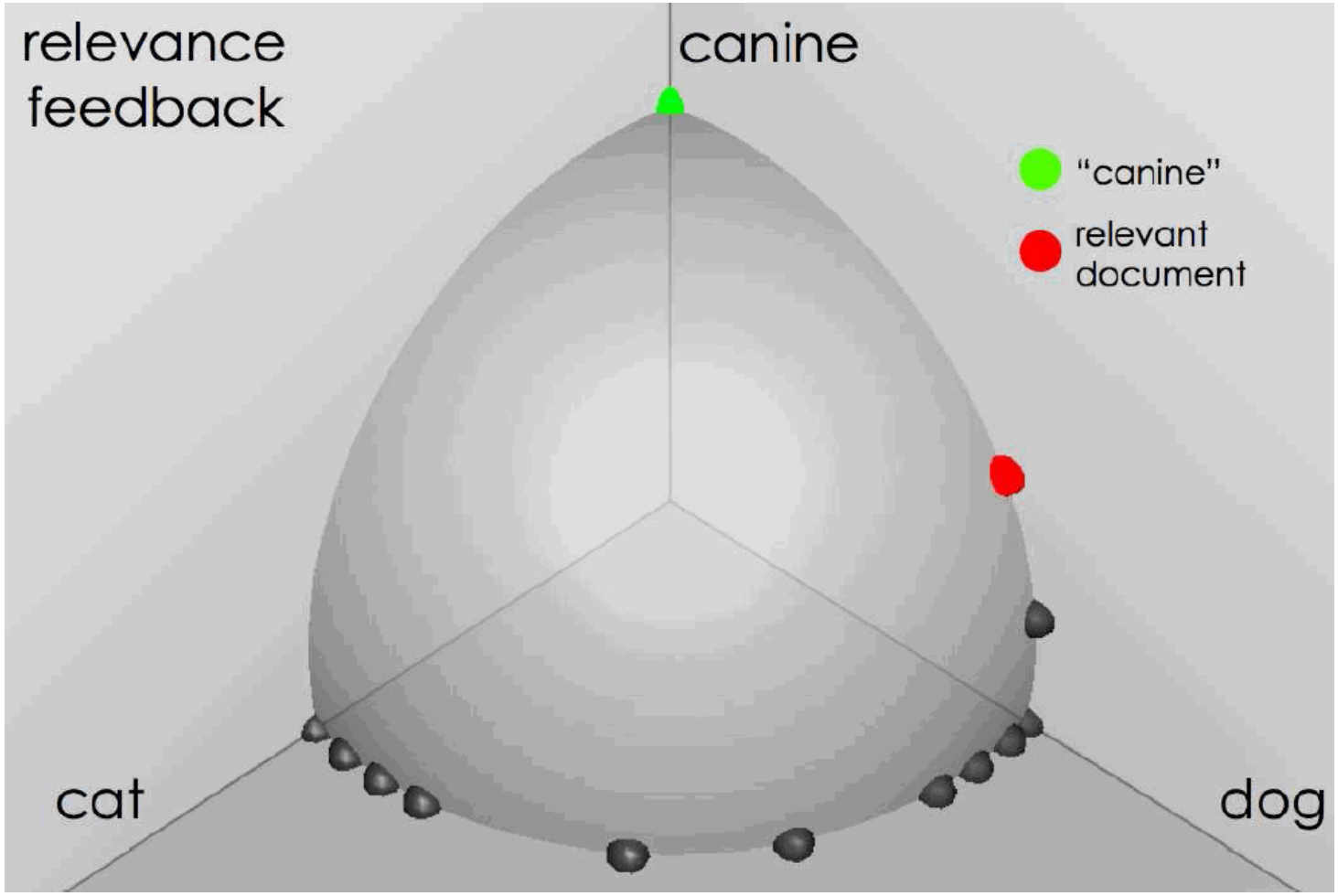

source: Fernando Diaz

# Ad hoc results for query *canine*



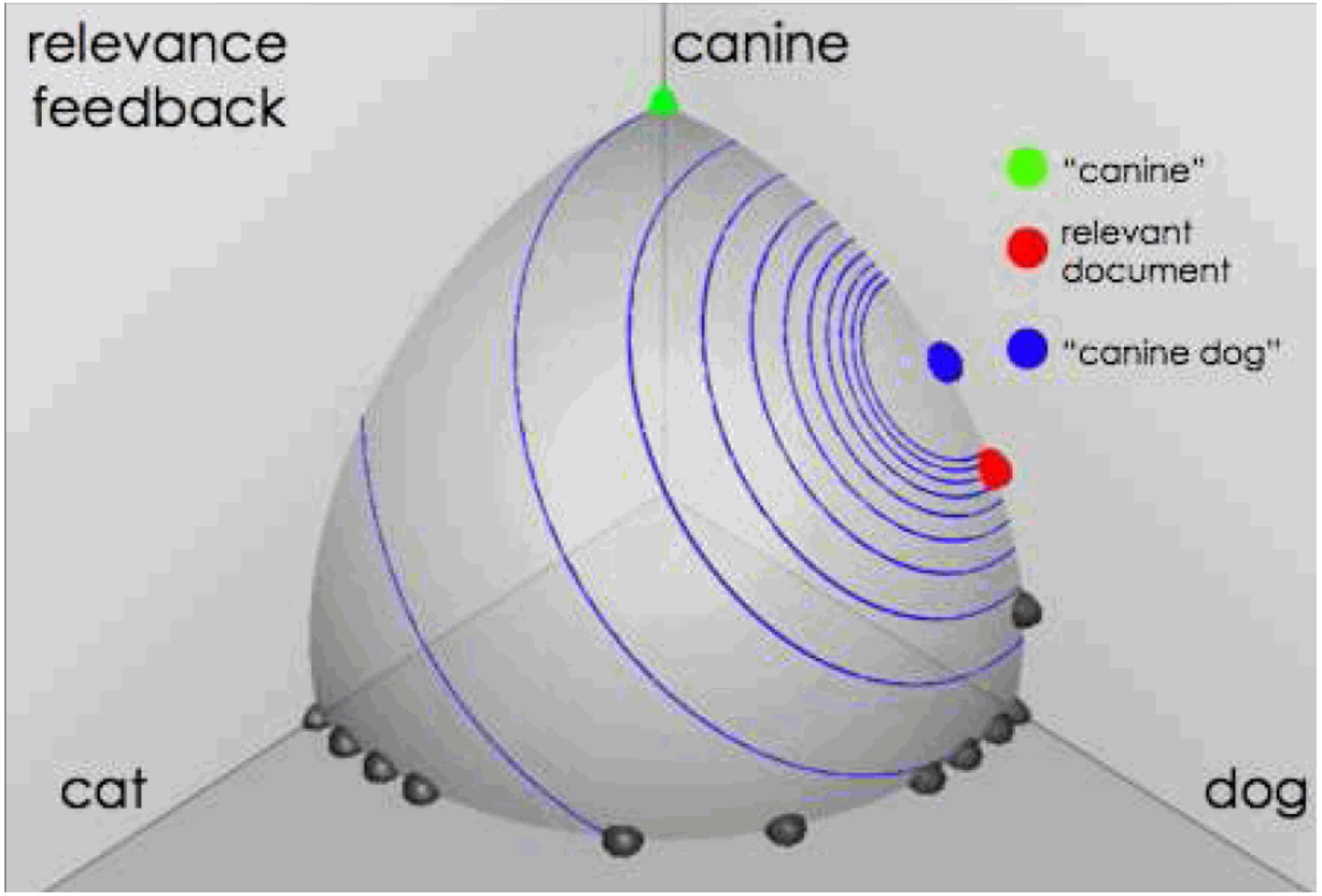

source: Fernando Diaz

# Ad hoc results for query *canine*




source: Fernando Diaz

# Ad hoc results for query *canine*




source: Fernando Diaz

# A Real (non-Image) Example

Initial query: [new space satellite applications]

Results for initial query:

| fb | rank | relevance | document |
|---|---|---|---|
| + | 1 | 0.539 | NASA Hasn't Scrapped Imaging Spectrometer |
| + | 2 | 0.533 | NASA Scratches Environment Gear From Satellite Plan |
| | 3 | 0.528 | Science Panel Backs NASA Satellite Plan, But Urges Launches of Smaller Probes |
| | 4 | 0.526 | A NASA Satellite Project Accomplishes Incredible Feat: Staying within Budget |
| | 5 | 0.525 | Scientist Who Exposed Global Warming Proposes Satellites for Climate Research |
| | 6 | 0.524 | Report Provides Support for the Critics Of Using Big Satellites to Study Climate |
| | 7 | 0.516 | Arianespace Receives Satellite Launch Pact From Telesat Canada |
| + | 8 | 0.509 | Telecommunications Tale of Two Companies |

User then marks relevant documents with "+".

# Query Expansion by Relevance Feedback

- Expanded query

| | | | |
|---|---|---|---|
| 2.074 | new | 15.106 | space |
| 30.816 | satellite | 5.660 | application |
| 5.991 | nasa | 5.196 | eos |
| 4.196 | launch | 3.972 | aster |
| 3.516 | instrument | 3.446 | arianespace |
| 3.004 | bundespost | 2.806 | ss |
| 2.790 | rocket | 2.053 | scientist |
| 2.003 | broadcast | 1.172 | earth |
| 0.836 | oil | 0.646 | measure |

Compared to the original query: [new space satellite applications]

# Results for Expanded Query

Initial query: [new space satellite applications]

Results for expanded query:

| fb | rank | relevance | document |
|---|---|---|---|
| * | 1 | 0.513 | NASA Scratches Environment Gear From Satellite Plan |
| * | 2 | 0.500 | NASA Hasn't Scrapped Imaging Spectrometer |
| | 3 | 0.493 | When the Pentagon Launches a Secret Satellite, Space Sleuths Do Some Spy Work of Their Own |
| | 4 | 0.493 | NASA Uses 'Warm' Superconductors For Fast Circuit |
| * | 5 | 0.492 | Telecommunications Tale of Two Companies |
| | 6 | 0.491 | Soviets May Adapt Parts of SS-20 Missile for Commercial Use |
| | 7 | 0.490 | Gaping Gap: Pentagon Lags in Race To Match the Soviets In Rocket Launchers |
| | 8 | 0.490 | Rescue of Satellite By Space Agency To Cost $90 Million |

Such "user feedback – query expansion – reranking" process can iterate multiple times

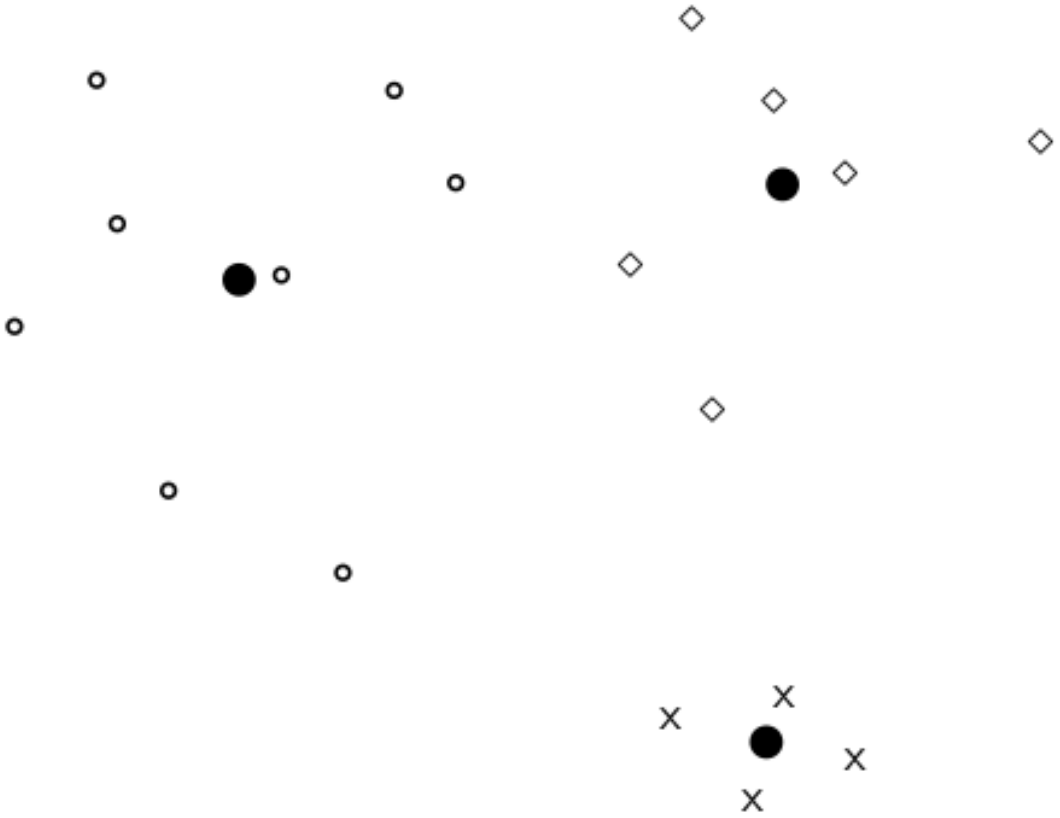
# Key Concept: Centroid

- The centroid is the center of mass of a set of points
- Suppose that we represent documents as points in a high-dimensional space using terms
- Definition: Centroid

$$\mu(C) = \frac{1}{|C|} \sum_{d \in C} d$$

where C is a set of documents.

# Centroid: Example



# Rocchio Algorithm

- The Rocchio algorithm uses the vector space to pick a relevance feedback query
- Rocchio seeks the query $q_{opt}$ that maximizes the similarity margin between the two clusters of docs

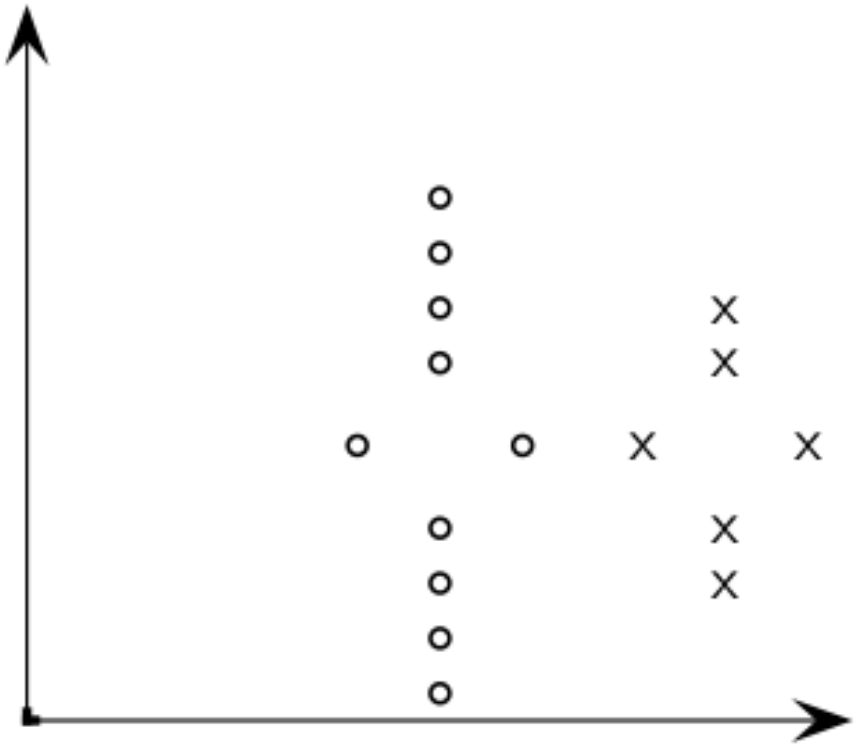
$$q_{\text{opt}} = \arg\max_{q} \left\{ \cos(q, \mu(C_r)) - \cos(q, \mu(C_n)) \right\}$$

- Implementation: try to separate docs marked relevant and non-relevant

$$q_{\text{opt}} = a \cdot q_0 + b \cdot \frac{1}{|C_r|} \sum_{d \in C_r} d - c \cdot \frac{1}{|C_n|} \sum_{d \in C_n} d$$

J. J. Rocchio, Relevance feedback in information retrieval In The SMART Retrieval System: Experiments in Automatic Document Processing (1971)

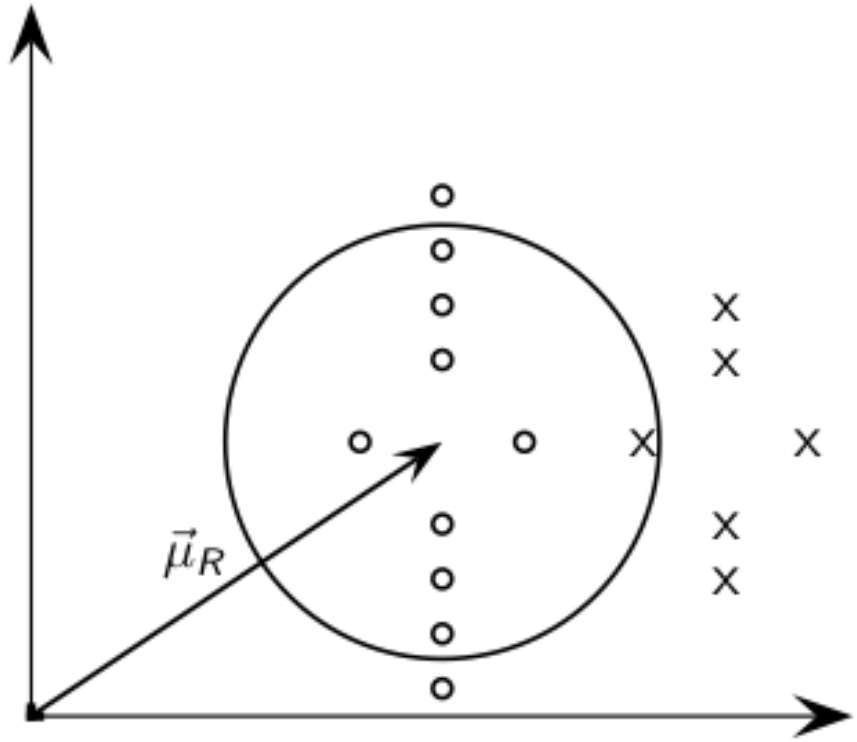
# Ricchio Example



x non-relevant documents

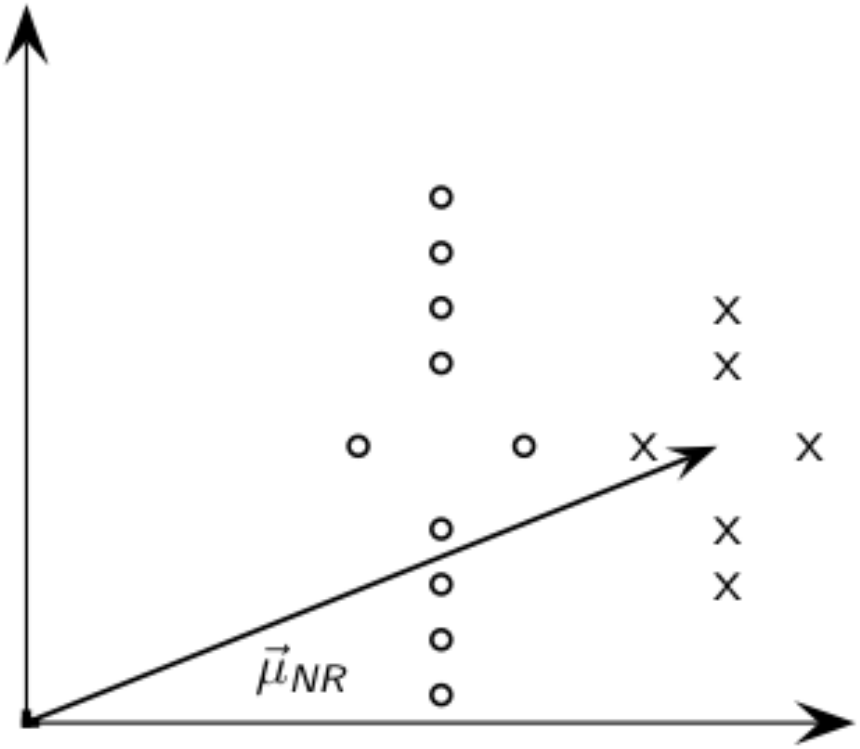
o relevant documents

# Ricchio Example

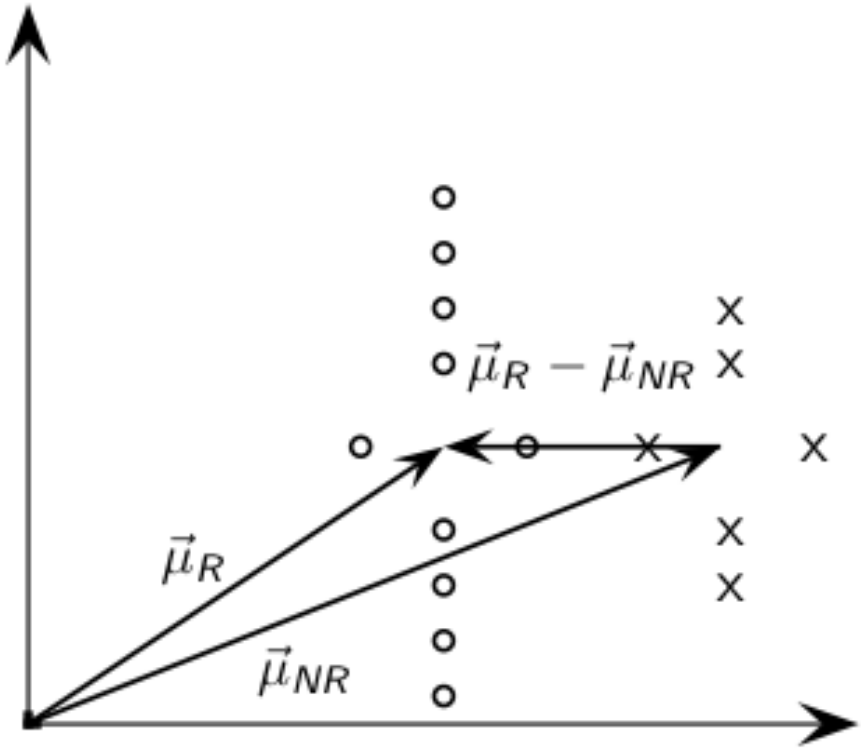



$\mu_R$ cannot separate relevant/non-relevant documents

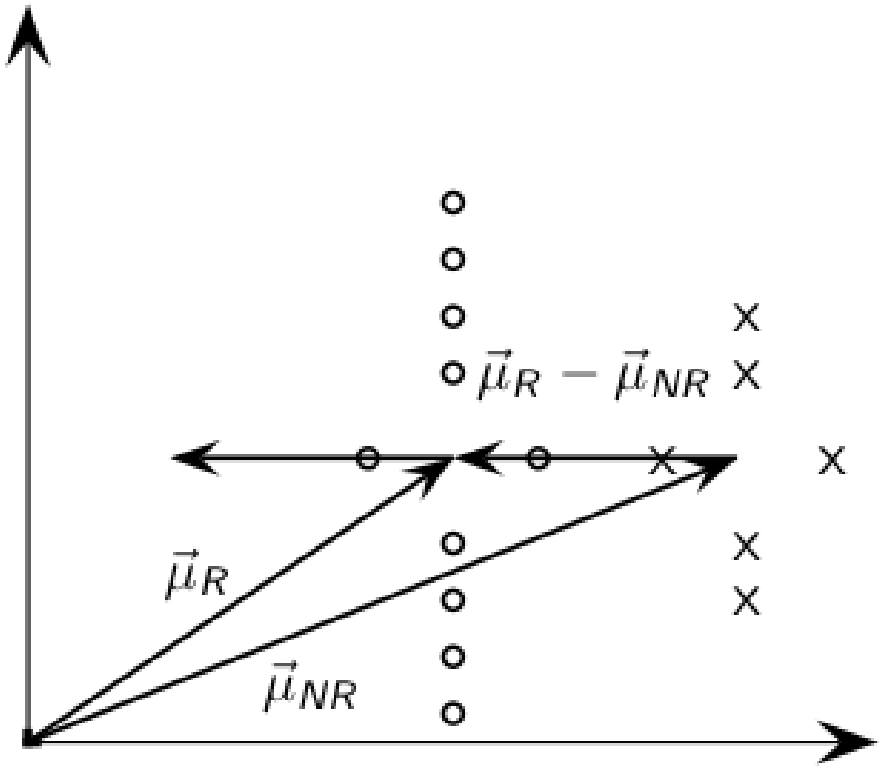
# Ricchio Example



# Ricchio Example

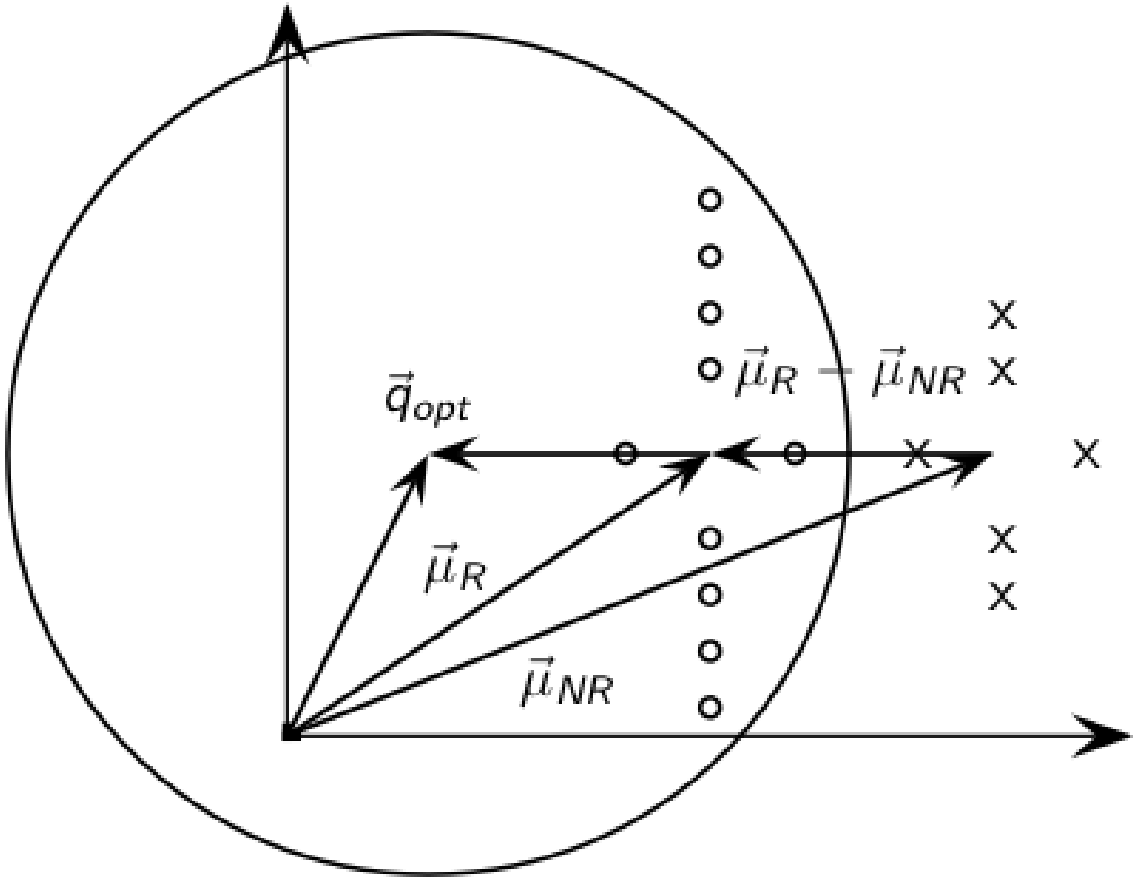


# Ricchio Example



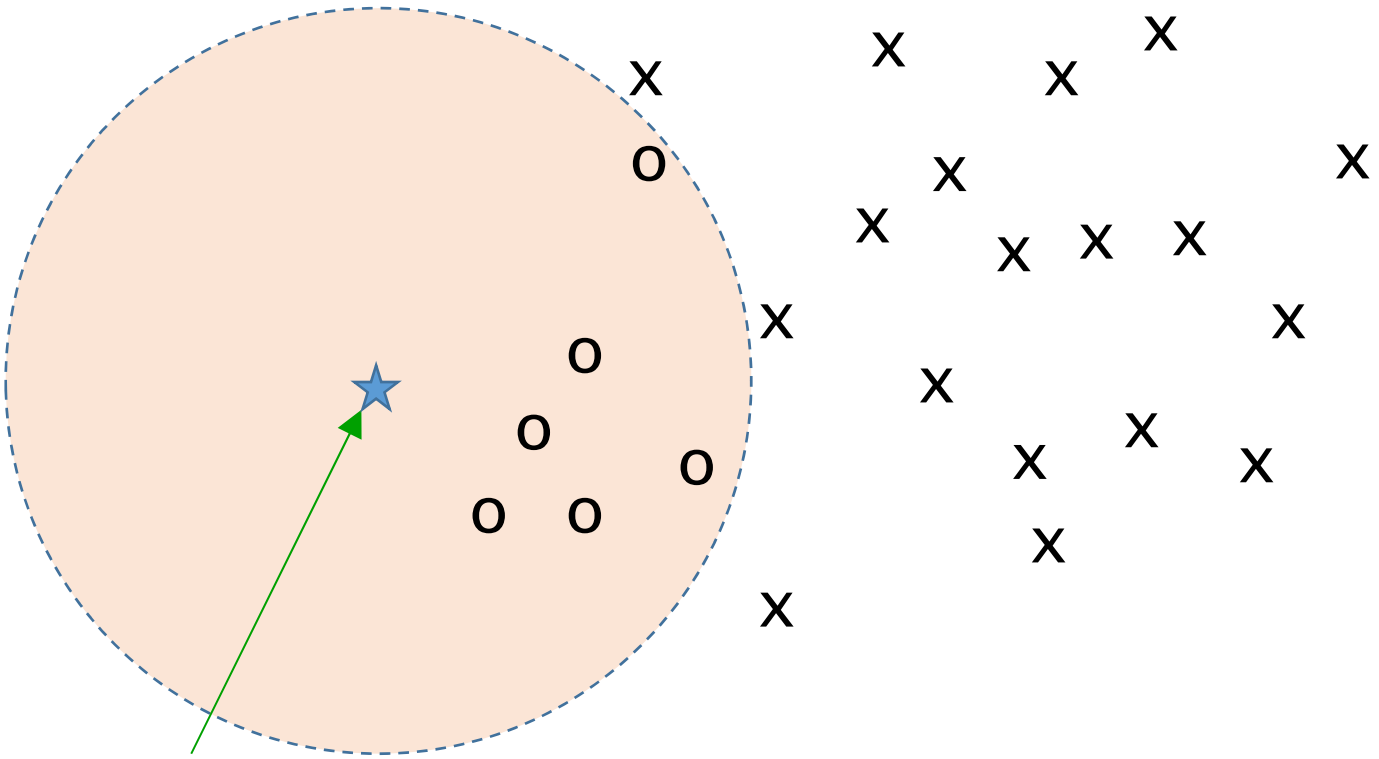

$$q_{\text{opt}} = \mu_R + \alpha(\mu_R - \mu_{NR})$$

# Ricchio Example




$q_{opt}$ could separate relevant / nonrelevant perfectly.

# The Theoretically Best Query




x non-relevant documents

o relevant documents

# Further on Relevance Feedback

- Probabilistic relevance feedback
  - There is a probability for each doc to be relevant to a query $P(r=1|q,d)$
    - Could be used to weight each document and search term
  - Robertson and Spärck-Jones (RSJ) Model
- Pseudo relevance feedback
  - There is no users' rating on the relevance of retrieved documents
  - Regarding the top-N retrieved documents as relevant ones to update the query

Prob. Ranking Principle: https://nlp.stanford.edu/IR-book/html/htmledition/the-probability-ranking-principle-1.html