Search and Rank
Lecture 6. Search and Rank

Jo Houghton

ECS Southampton

February 22, 2019

Search and Rank

Searching the web has become the default way to find information. However, there is so much information on the web, how can we find what we are actually looking for?

This is **information retrieval**

"the activity of obtaining information resources relative to an information need from a collection of information resources"

Search and Rank - Problem

It's all about the user.

'What is the population of Southampton?'
A simple question to answer.. (217000),

Search and Rank - Problem

It's all about the user.

'What is the population of Southampton?'
A simple question to answer.. (217000), assuming I mean the
Southampton near me, there are 6 in the world.

Information retrieval exercise

Search and Rank - Problem

It's all about the user.

'What is the population of Southampton?'
A simple question to answer.. (217000), assuming I mean the
Southampton near me, there are 6 in the world.

Information retrieval exercise
'Where is the best Chinese food?

Search and Rank - Problem

It's all about the user.

'What is the population of Southampton?'
A simple question to answer.. (217000), assuming I mean the
Southampton near me, there are 6 in the world.

Information retrieval exercise
'Where is the best Chinese food? I'm hungry!'

Search and Rank - Problem

It's all about the user.

'What is the population of Southampton?'
A simple question to answer.. (217000), assuming I mean the
Southampton near me, there are 6 in the world.

Information retrieval exercise
'Where is the best Chinese food? I'm hungry!'

If I am searching for the best Chinese food, I need somewhere near
me, that isn't too expensive, that will deliver to my house, and will
not take too long.

A good information retrieval system should be aware of the implicit
preferences I have.

Search and Rank - History

Information retrieval was originally aided by a catalogue.

For the ancient library of Alexandria, ca. 300BC, Greek poet and scholar Callimachus made the 'Pinakes' (tables)

This was a list of all the works sorted by genre, but only small fragments survive. The idea survived, and as soon as books could be printed, they had printed indexes (1460 CE)

In 1842, a Paris bookseller Jean Claude Brunet had developed a simple classification system for his books.

Most modern libraries use the Dewey Decimal system (1876) which introduced idea of location based on subject.

Search and Rank - History

With computers,

- ▶ 1960s, databases were indexed
- ▶ 1970s, larger boolean systems on the computer
- ▶ 1980s, expert systems, natural language processing
- ▶ 1990s, the **internet**.. ranking
- ▶ 2000 - now, much better search and ranking, big data, language modelling

Search and Rank - Text retrieval

Text retrieval:

- ▶ For a collection of text documents - text corpus
- ▶ User provides query - expresses information need
- ▶ Search engine returns relevant documents

This is **search technology** in industry

Search and Rank - Text retrieval

	Database	Text
Information	Well-defined structure and semantics	Unstructured, ambiguous semantics
Query	Well defined semantics, complete specification (eg SQL)	Ambiguous, incomplete specification
Answers	Matched records	Relevant documents

You cannot prove mathematically what the best ways to retrieve a text item is. It is *empirically defined*, noone knows what the user wants until the user finds it.

Search and Rank - Text retrieval

- ▶ Boolean Model: get every document that satisfies a Boolean Expression

Search and Rank - Text retrieval

- ▶ Boolean Model: get every document that satisfies a Boolean Expression result selection

Search and Rank - Text retrieval

- ▶ Boolean Model: get every document that satisfies a Boolean Expression result selection
- ▶ Vector Space Model: how similar is document to query vector?

Search and Rank - Text retrieval

- Boolean Model: get every document that satisfies a Boolean Expression result selection
- Vector Space Model: how similar is document to query vector? ranked results

Search and Rank - Text retrieval

- Boolean Model: get every document that satisfies a Boolean Expression result selection
- Vector Space Model: how similar is document to query vector? ranked results
- Probabilistic Model: what is the probability that the document is generated by the query?

Search and Rank - Text retrieval

▶ Boolean Model: get every document that satisfies a Boolean Expression result selection

▶ Vector Space Model: how similar is document to query vector? ranked results

▶ Probabilistic Model: what is the probability that the document is generated by the query? ranked results

Search and Rank - Text retrieval

- ▶ Boolean Model: get every document that satisfies a Boolean Expression *result selection*
- ▶ Vector Space Model: how similar is document to query vector? *ranked results*
- ▶ Probabilistic Model: what is the probability that the document is generated by the query? *ranked results*

Selecting results is hard: if query is *over-constrained*, you may get nothing. If query is *under-constrained*, you may get way too many unsorted results.

Ranking is preferred, allows prioritisation.

Search and Rank - Text retrieval

Robertson 1977:

Using Decision Theory, the optimal strategy for finding the most relevant document is:

To give a ranked list of documents in descending order of probability that a document is relevant to the query.
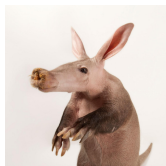
This assumes:

▶ utility of document is independent of utility of any other document

▶ user browses results sequentially

Do these assumptions hold?

Quick Recap - One Hot Encoding

We use a 'Bag of Words', where each word is a vector:

- ▶ a $\rightarrow [1, 0, 0, 0, 0, 0, 0, 0, \ldots, 0]$
- ▶ aa $\rightarrow [0, 1, 0, 0, 0, 0, 0, 0, \ldots, 0]$
- ▶ aardvark $\rightarrow [0, 0, 1, 0, 0, 0, 0, 0, \ldots, 0]$
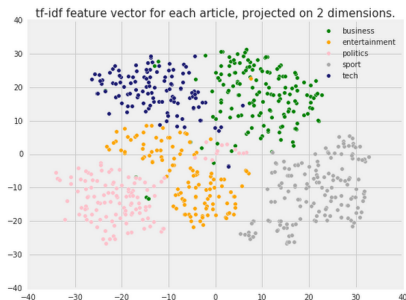- ▶ aardwolf $\rightarrow [0, 0, 0, 1, 0, 0, 0, 0, \ldots, 0]$

This is called *One Hot Encoding*

Search and Rank - Vector Space Model

Vector Space Model is quite simple.

- ▶ Each document is a vector
- ▶ Each query is a vector
- ▶ Assume that if close together in space they are similar
- ▶ Rank each document by similarity

tf-idf feature vector for each article, projected on 2 dimensions.

https://cloud.google.com/blog/products/gcp/problem-solving-with-ml-automatic-document-classification

Search and Rank - Vector Space Model

Each document is represented by a **term vector**
A **Term** is a basic concept, e.g. word or phrase
This gives an N-dimensional space for N **terms**

- Query Vector $q = (x_1, x_2, \ldots, x_N)$, $x_i \in \Re$ is query term weight
- Document Vector $d = (y_1, y_2, \ldots, y_N)$, $y_i \in \Re$ is document term weight

relevance($\mathbf{q}$, $\mathbf{d}$) $\propto$ similarity($\mathbf{q}$, $\mathbf{d}$) $= f(\mathbf{q}, \mathbf{d})$

Search and Rank - Vector Space Model

As seen before, each **term** is assumed to be orthogonal
We still don't know:

- ▶ Which basic concepts to select for the **terms**
- ▶ How to assign weights $(x_1, x_2, \ldots, x_N)$ and $(y_1, y_2, \ldots, y_N)$
- ▶ How to define the similarity measure $(f(\boldsymbol{q}, \boldsymbol{d}))$

Quick Recap - Documents

e.g. "The quick brown fox jumped over the lazy dog."
becomes:

Tokenisation
'The': 1, 'quick': 1, 'brown': 1, 'fox': 1, 'jumped': 1, 'over': 1,
'the': 1, 'lazy': 1, 'dog.': 1

Quick Recap - Documents

e.g. "The quick brown fox jumped over the lazy dog."
becomes:

Tokenisation
'The': 1, 'quick': 1, 'brown': 1, 'fox': 1, 'jumped': 1, 'over': 1,
'the': 1, 'lazy': 1, 'dog.': 1

After further tokenising and sorting, you could get:
'brown': 1, 'dog': 1, 'fox': 1, 'jumped': 1, 'lazy': 1,'over': 1,
'quick': 1, 'the': 2

Quick Recap - Documents

e.g. "The quick brown fox jumped over the lazy dog."
becomes:

Tokenisation
'The': 1, 'quick': 1, 'brown': 1, 'fox': 1, 'jumped': 1, 'over': 1,
'the': 1, 'lazy': 1, 'dog.': 1

After further tokenising and sorting, you could get:
'brown': 1, 'dog': 1, 'fox': 1, 'jumped': 1, 'lazy': 1,'over': 1,
'quick': 1, 'the': 2

Further stemming and removal of stop words could give:
'brown': 1, 'dog': 1, 'fox': 1, 'jump': 1, 'lazi': 1,'over': 1, 'quick':
1

Quick Recap - Documents

The bag of words vector for a document will be very sparse

The vocabulary or lexicon will be the *set* of all (processed) words across all documents known to the system.

For a document, the vector contains the number of occurrences of each **term**, like a histogram
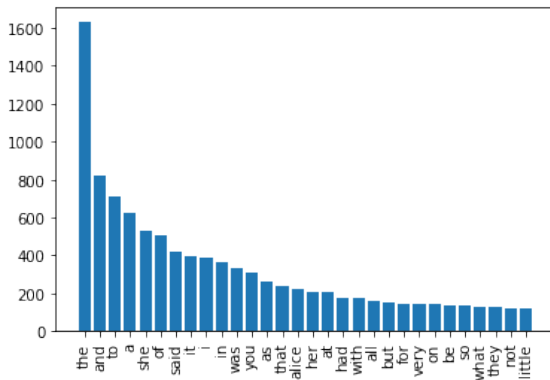
Vectors will have very high number of dimensions, but are very sparse
Alice ipynb demo

Search and Rank - Vector Space Model

Zipf's law

States that the frequency of a word is proportional to the inverse of its rank, i.e. the second most common word will be half the frequency of the first, the third will be a third the frequency of the first, e.t.c.

Search and Rank - Vector Space Model

We should remove the most common and the least common, as they hold little information, and may skew any vector to look similar.

We should also remove the very rare words, as they would make the document vector unnecessarily sparse without gaining much information.
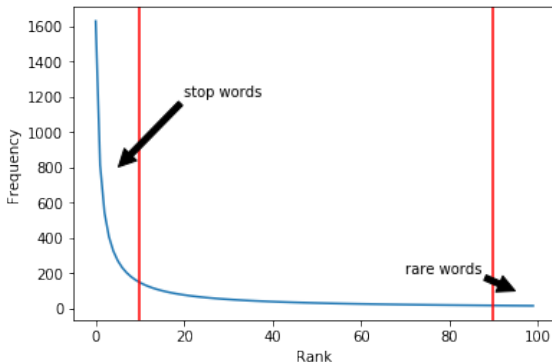
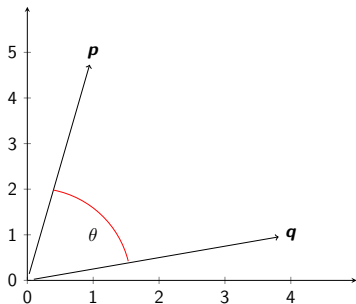# Search and Rank - Vector Space Model

How to search the Vector Space Model?
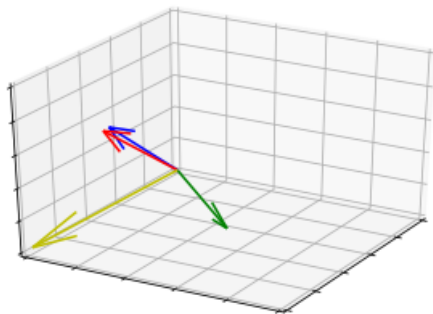
If my query vector is red, which of the three document vectors is it closest to?

Quick Recap - Cosine Similarity

▶ Cosine Similarity

Only measures direction, not magnitude of vector.

p and q are N-dim vectors,
$p = [p_1, p_2, ..., p_N]$,
$q = [q_1, q_2, ..., q_N]$

Cosine Similarity:

$$cos(\theta) = \frac{p \cdot q}{|p||q|}$$

$$= \frac{\sum_{i=1}^{N} p_i q_i}{\sqrt{\sum_{i=1}^{N} p_i^2} \sqrt{\sum_{i=1}^{N} q_i^2}}$$

$\sum p_i^2$ and q_i^2 can be precomputed and stored

Search and Rank - Vector Space Model

Inverted Index:
A mapping from content to location, e.g. in a set of documents.

Inverted index ipynb demo

alic	(0, 399), (1, 207), (2, 232)
said	(0, 462), (1, 11), (2, 0)
cooper	(0, 0), (1, 398), (2, 0)
spring	(0, 0), (1, 2), (2, 218)
not	(0, 145), (1, 21), (2, 5)
retriev	(0, 0), (1, 111), (2, 47)
littl	(0, 128), (1, 4), (2, 3)
one	(0, 105), (1, 21), (2, 5)

A posting is a pair formed by a document ID and the number of times the specific word appeared in that document.

Search and Rank - Vector Space Model

To efficiently compute the cosine similarity, look up the relevant postings list and accumulate similarities only for the documents in those lists.

alic	(0, 399), (1, 207), (2, 232)
said	(0, 462), (1, 11), (2, 0)
cooper	(0, 0), (1, 398), (2, 0)
spring	(0, 0), (1, 2), (2, 218)
not	(0, 145), (1, 21), (2, 5)
retriev	(0, 0), (1, 111), (2, 47)
littl	(0, 128), (1, 4), (2, 3)
one	(0, 105), (1, 21), (2, 5)

For example: "Alice Cooper"

Accumulation table:

Doc ID	Frequency
0	399
1	207
2	232

Search and Rank - Vector Space Model

To efficiently compute the cosine similarity, look up the relevant postings list and accumulate similarities only for the documents in those lists.

alic	(0, 399), (1, 207), (2, 232)
said	(0, 462), (1, 11), (2, 0)
cooper	(0, 0), (1, 398), (2, 0)
spring	(0, 0), (1, 2), (2, 218)
not	(0, 145), (1, 21), (2, 5)
retriev	(0, 0), (1, 111), (2, 47)
littl	(0, 128), (1, 4), (2, 3)
one	(0, 105), (1, 21), (2, 5)

For example: "Alice Cooper"

Accumulation table:

Doc ID	Frequency
0	399
1	$207 + 398$
2	232

cosine similarity ipynb demo

Search and Rank - Vector Space Model

Using frequency of a word in a document is not always a good idea

How can we weight these vectors better?

- ▶ Binary weights
  record only if a word is present or absent in the vector

Search and Rank - Vector Space Model

Using frequency of a word in a document is not always a good idea

How can we weight these vectors better?

- ▶ Binary weights
  record only if a word is present or absent in the vector
- ▶ Raw Frequency
  record frequency of occurrence of a term in the vector

Search and Rank - Vector Space Model

Using frequency of a word in a document is not always a good idea

How can we weight these vectors better?

- ▶ Binary weights
  record only if a word is present or absent in the vector
- ▶ Raw Frequency
  record frequency of occurrence of a term in the vector
- ▶ TF-IDF - Term Frequency - Inverse Document Frequency
  - ▶ Term Frequency - the raw frequency of a word in a document usually normalised by the number of words in the document
  - ▶ Inverse Document Frequency - 1/number of occurrences of word in all documents

Search and Rank - Vector Space Model

Using frequency of a word in a document is not always a good idea

How can we weight these vectors better?

▶ Binary weights
record only if a word is present or absent in the vector

▶ Raw Frequency
record frequency of occurrence of a term in the vector

▶ TF-IDF - Term Frequency - Inverse Document Frequency
  ▶ Term Frequency - the raw frequency of a word in a document usually normalised by the number of words in the document
  ▶ Inverse Document Frequency - 1/number of occurrences of word in all documents

A high weight in TF-IDF is reached by a high frequency in a given document, but low frequency in the whole collection of documents, this would filter out the more common terms.

Search and Rank - Vector Space Model

There are many variants of TF-IDF
For the TF (term frequency term:

- ▶ term frequency $\qquad \frac{f_{t,d}}{\sum_{t' \in d} f_{t,d}}$
- ▶ log normalisation $\qquad \log(1 + f_{t,d})$
- ▶ double normalisation K $\qquad K + (1 - K)\frac{f_{t,d}}{\max_{t' \in d} f'_{t,d}}$

For the inverse document frequency term:

- ▶ Inverse document frequency $\qquad \log \frac{N}{n_t}$
- ▶ Inverse document frequency smooth $\qquad \log \frac{N}{1 + n_t}$
- ▶ Inverse document frequency max $\qquad \log \frac{\max_{t' \in d} n_{t'}}{i + n_t}$
- ▶ Probabilistic inverse document frequency $\qquad \log \frac{N - n_t}{n_t}$
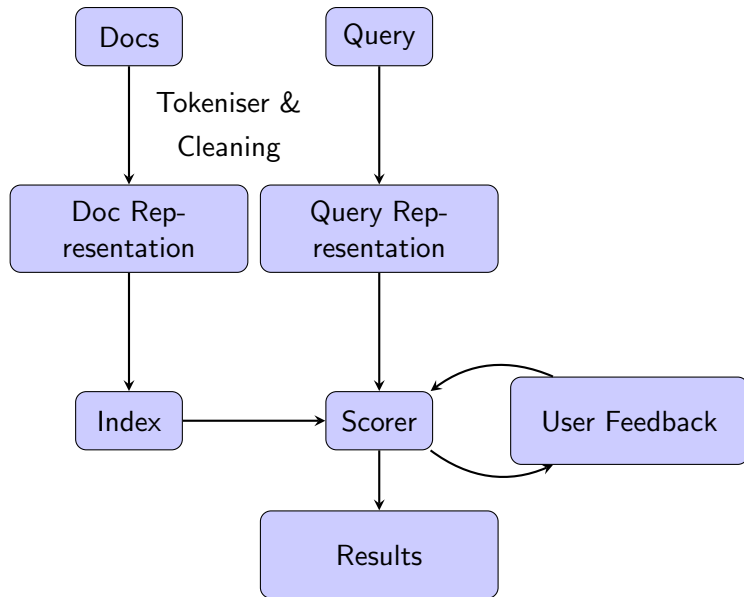
Search and Rank - Vector Space Model

Then TF-IDF is calculated as TF $\times$ IDF

TF-IDF ipynb demo

Some possible schemes for TF-IDF:

$$
\begin{array}{cc}
\textit{Document} & \textit{Query} \\[4pt]
f_{t,d} \log \dfrac{N}{n_t} & (K + K \dfrac{f_{t,q}}{\max_t f_{t,q}}) \log \dfrac{N}{n_t} \quad K = 0.5 \\[10pt]
1 + \log f_{t,d} & \log(1 + \dfrac{N}{n_t}) \\[10pt]
(1 + \log f_{t,d}) \log \dfrac{N}{n_t} & (1 + \log f_{t,q}) \log \dfrac{N}{n_t}
\end{array}
$$

Search and Rank - Retrieval System

Search and Rank - Retrieval System

Building the index is difficult with big data:
Can't just use memory

Can use sort based methods:

- ▶ collect local $<$term, doc, freq$>$ tuples in a run
- ▶ sort tuples within the run and write to disk
- ▶ merge runs on disk
- ▶ output inverted index

Search and Rank - Retrieval System

Go through a few
documents..
<"alic", 0, 399>
<"said", 0, 462>
<"not", 0, 145>
..
<"alic", 1, 207>
<"said", 1, 11>
<"cooper", 1, 398>
...
<"alic", 2, 232>
<"spring", 2, 218>
<"not", 2, 5>

Search and Rank - Retrieval System

Go through a few
documents..
<"alic", 0, 399>
<"said", 0, 462>
<"not", 0, 145>
..
<"alic", 1, 207>
<"said", 1, 11>
<"cooper", 1, 398>
...
<"alic", 2, 232>
<"spring", 2, 218>
<"not", 2, 5>

Sort by term
<"alic", 0, 399>
<"alic", 1, 207>
<"alic", 2, 232>
..
<"cooper", 1, 398>
<"said", 0, 462>
<"said", 1, 11>
...
<"not", 0, 145>
<"not", 2, 5>
<"spring", 2, 218>

Search and Rank - Retrieval System

Go through a few
documents..
<"alic", 0, 399>
<"said", 0, 462>
<"not", 0, 145>
..
<"alic", 1, 207>
<"said", 1, 11>
<"cooper", 1, 398>
...
<"alic", 2, 232>
<"spring", 2, 218>
<"not", 2, 5>

Sort by term
<"alic", 0, 399>
<"alic", 1, 207>
<"alic", 2, 232>
..
<"cooper", 1, 398>
<"said", 0, 462>
<"said", 1, 11>
...
<"not", 0, 145>
<"not", 2, 5>
<"spring", 2, 218>

This is one run.

Then run a merge
sort with other runs,
and build the inverted
index with the sorted
data

Search and Rank - Improving the system

Improving Information retrieval:
Can use Location weighting:

- ▶ More relevant if closer to start of document
- ▶ Exact phrase or proximity of terms in query

Requires term position of every instance in the document to be indexed
i.e. "alic" = [Doc0, 399, $< 3, 11, 29, ..>$]

Increases size of index dramatically, need to use compression

Search and Rank - Web Search

For web pages, we can use the number of links to a document as a way of scoring them

However this is prone to manipulation, as you can make lots of pages that all point to each other lots of times.

Page and Brin: PageRank

Search and Rank - Web Search

For web pages, we can use the number of links to a document as a way of scoring them

However this is prone to manipulation, as you can make lots of pages that all point to each other lots of times.

Page and Brin: PageRank Markus will cover this in May

In brief: Calculates the importance of a page from the importance of all the other pages that link to it and the number of links each of those other pages have.

Search and Rank - User Feedback

What user feedback do we get with a web search?

Use what they actually click on. Documents that are more clicked on could be more important.

Can also learn associations between queries and documents, and if this query is met again, use this to increase the rank of the document that was clicked on before for this query.

Search and Rank - Result Diversification

Is a ranked list always the right thing?

e.g. If I search for "University"
Do I want..

- ▶ Lots of links to Southampton University..?
  or
- ▶ Links to a range of universities

Search and Rank - Result Diversification

Is a ranked list always the right thing?

e.g. If I search for "University"
Do I want..

- ▶ Lots of links to Southampton University..?
  or
- ▶ Links to a range of universities

A diverse range of results has a better chance of finding what the search was actually looking for.

Search and Rank - Summary

Search engines are a key tool in Data Mining

Important Points:

- ▶ Feature extraction
- ▶ Scalable and efficient indexing and search

Information Retrieval Process:

- ▶ Encode Documents
  - ▶ stemming, lemmatization, stop word removal, TF-IDF..
  - ▶ Make feature vector (e.g. Bag of words)
- ▶ Make index (inverted index aids search)
- ▶ Encode query
- ▶ Search index using encoded query