Search Engines

Gertjan van Noord

September 19, 2024

This week

- Some notes on coding
- Chapter 6: scoring, ranked retrieval, tf-idf

How to recognize potential inefficiency?

```
for x in lista:
    if x in listb:

# if listb is a list, this is quadratic
# if listb is a dictionary, this is linear
```

How to recognize potential inefficiency?

```
for x in lista:
    for y in listb:

# if listb is a list, this is quadratic
# if listb is a dictionary, this is still quadratic
```

How to recognize potential inefficiency?

```
for x in lista:

# if lista is a list, you get the elements in order
# if lista is a dictionary, it depends ...
```

In some implementations, you get the elements of a dictionary in the order the elements were placed in the dictionary. In other implementations, you get the elements in some other order.

Your code should not depend on a particular order.

Sorted lists and sets, again

- sorted lists: intersection linear, membership logarithmic (cf. `bisect` module)
- hash (Python dictionaries/sets): intersection linear, membership linear (elements must be non-mutable)
- sorted lists are memory-efficient, hashes "waste" some memory
- footnote: check out "perfect hash" if you want both compact and efficient

Chapter 6: Scoring

- parametric indexing; zone indexing
- weighted zone indexing
- term frequency and weighting
- vector model, queries as vectors
- computing scores

NB: we skip sections 6.1.2, 6.1.3, 6.4

Meta-data of the document is important

- *fields*: author, title, keywords, date, . . .
- per field, a separate *parametric* index is built
- the user interface should accomodate querying of such fields in addition to text queries
- the system must combine the query parts for retrieval

Zone indexes

- Zones are similar to fields in parametric index, except that zones contain arbitrary free text
- Per zone, a separate index is built
- Zones could be author, title, abstract, keyword, body, ...
- This assumes that documents have a common outline

Zone index for zone queries

- Query: `author:Greene AND abstract:war AND body:London`
- Can be answered assuming we have several zone indexes
  - author-Greene $\longrightarrow$ [doc1,doc6,doc10]
  - abstract-war $\longrightarrow$ [doc3,doc4,doc6,doc10]
  - body-London $\longrightarrow$ [doc2,doc4,doc6]
- Business as usual: merge posting lists

Weighted zone scoring

- If a search term occurs in the title, the document is probably more relevant
- Certain $zones$ of a document are more important than others
- Assign a weight to each zone
- A matching document is assigned a weight
  - score of a zone: its weight if there is a hit, 0 otherwise
  - total score: sum zone scores

Weighted zone scoring

Query: `pie AND cream`

Two zones: title (weight:0.6), abstract (weight:0.4)

	title	abstract
doc1	`apple pie`	`pie cream`
doc2	`cream pie recipe`	`apple cream pie`
doc3	`apple pie`	`apple cream`

Weighted zone scoring

Query: `pie AND cream`

Two zones: title (weight:0.6), abstract (weight:0.4)

	title	abstract
doc1	`apple pie`	`pie cream`
doc2	`cream pie recipe`	`apple cream pie`
doc3	`apple pie`	`apple cream`

doc1: 0.6 * 0 + 0.4 * 1 = 0.4
doc2: 0.6 * 1 + 0.4 * 1 = 1.0
doc3: 0.6 * 0 + 0.4 * 0 = 0.0

Weighted zone scoring

Query: `pie AND cream`

Two zones: title (weight:0.6), abstract (weight:0.4)

	title	abstract
doc1	`apple pie`	`pie cream`
doc2	`cream pie recipe`	`apple cream pie`
doc3	`apple pie`	`apple cream`

doc1: 0.6 * 0 + 0.4 * 1 = 0.4
doc2: 0.6 * 1 + 0.4 * 1 = 1.0
doc3: 0.6 * 0 + 0.4 * 0 = 0.0

This is first example of *ranked* retrieval

Problems with Boolean model

- Boolean systems often give either too many or too few results
  - but useful for specific precise search in a homogeneous corpus
- Boolean operators are difficult for many users
- Users want to type some query words, no operators, and get best results
- We need methods to rank results
- Is there more than presence/absence? Frequency?
- Some terms may be more important than others?

Advanced Scores for Terms in Documents

Two key insights:

- A document in which a search term occurs more often is more relevant
- A search term which occurs in fewer documents is more important
- Combined into the very famous heuristic **tf-idf**

TF-IDF

- $\text{tf}_{t,d}$: term frequency of term t in doc d

Please note: this is just the number of occurrences of t in d. *So it is not the relative frequency*.

TF-IDF

- $\text{tf}_{t,d}$: term frequency of term t in doc d
- df_t: document frequency of term t (in how many documents does t occur)
- relative document frequency of term t is defined as df_t/N where N is the number of documents
- idf_t: inverse document frequency. It is defined as $\log(N/\text{df}_t)$
- If df is higher, idf is lower; a term that only occurs in few documents has a high idf
- Score for a term t in document d: $\text{tf}_{t,d} \times \text{idf}_t$.

TF-IDF

idf_t: inverse document frequency. It is defined as $\log(N/\text{df}_t)$

It does not matter much which log you use. Therefore, we use the default choice in Python.

```
import math

idf = math.log(number_of_docs / docfreq)
```

TF-IDF

- Score for a term t in document d: $\text{tf}_{t,d} \times \text{idf}_t$.
- Score for a query $q = t_1 \dots t_n$ is the sum of the tf-idf values for each search term t_i.

$$\text{score}(q, d) = \sum_{t_i} (\text{tf}_{t_i,d} \times \text{idf}_{t_i})$$

Example

TF	ape	child	food	of	panther
doc1	8	2	2	10	2
doc2	1	5	9	20	0
Query	1	0	1	1	0

Example

TF	ape	child	food	of	panther
doc1	8	2	2	10	2
doc2	1	5	9	20	0
Query	1	0	1	1	0
IDF	5	2	2	0.001	6

Example

TF	ape	child	food	of	panther
doc1	8	2	2	10	2
doc2	1	5	9	20	0
Query	1	0	1	1	0
IDF	5	2	2	0.001	6

score(food of ape,doc1):

score(food of ape,doc2):

Example

TF	ape	child	food	of	panther
doc1	8	2	2	10	2
doc2	1	5	9	20	0
Query	1	0	1	1	0
IDF	5	2	2	0.001	6

score(food of ape,doc1): 8*5 + 2*2 + 10*0.001 = 44.01
score(food of ape,doc2): 1*5 + 9*2 + 20*0.001 = 23.02

Vectors

TF	ape	child	food	of	panther
doc1	8	2	2	10	2
doc2	1	5	9	20	0
IDF	5	2	2	0.001	6

We can immediately build vectors with tf-idf weights.

Vectors

TF-IDF	ape	child	food	of	panther
doc1	40	4	4	0.01	12
doc2	5	10	18	0.02	0

Remaining problem: the frequencies do not take document length into account.

Vectors

TF-IDF	ape	child	food	of	panther
doc1	40	4	4	0.01	12
doc2	5	10	18	0.02	0

Remaining problem: the frequencies do not take document length into account.

Solution: **normalize vectors**

Normalize vectors

- Divide each cell in the vector by its length
- Length of a vector: square root of the sum of the squares of all the elements
- Length of vector for doc1: 42.1 (square root of (1600 + 16 + 16 + 0 + 144))
- Length of vector for doc2: 21.2 (square root of (25 + 100 + 0 +324))

Normalize vectors

- Divide each cell in the vector by its length
- Length of a vector: square root of the sum of the squares of all the elements
- Length doc1: 42.1; length doc2: 21.2

	ape	child	food	of	panther
doc1	0.95	0.09	0.09	0	0.3
doc2	0.24	0.48	0.86	0	0

Normalized vectors

	ape	child	food	of	panther
doc1	0.95	0.09	0.09	0	0.3
doc2	0.24	0.48	0.86	0	0
Query	1	0	1	1	0

Score for each document?

$$\sum_i \text{doc}_i \text{query}_i$$

Sum of products

score(query,doc1): 0.95*1 + 0.09*1 + 0*1 = 1.04
score(query,doc2): 0.24*1 + 0.86*1 + 0*1 = 1.10

Query can be weighted too

This also works if the terms in the query are weighted with tf-idf. Perhaps some terms occur twice in the query.

query: ape food of ape

	ape	child	food	of	panther
TF	2	0	1	1	0
IDF	5	2	2	0.001	6
TFIDF	10	0	2	0.001	0

Normalized vector for query too

Vector query is normalized too

TFIDF	10	0	2	0.001	0

$\longrightarrow$

Query	0.98	0	0.20	0.0	0

Normalized vectors

	ape	child	food	of	panther
doc1	0.95	0.09	0.09	0	0.3
doc2	0.24	0.48	0.86	0	0
Query	0.98	0	0.20	0.00	0

Score for each document?

$$\sum_i \text{doc}_i \text{query}_i$$

so, as before: sum of products

Normalized vectors

	ape	child	food	of	panther
doc1	0.95	0.09	0.09	0	0.3
doc2	0.24	0.48	0.86	0	0
Query	0.98	0	0.20	0.00	0

Score for each document?

$$\sum_i \text{doc}_i \text{query}_i$$

Sum of products.

score(query,doc1): 0.95*0.98 + 0.09*0.2 + 0*0.0 = 0.95
score(query,doc2): 0.24*0.98 + 0.86*0.2 + 0*0.0 = 0.41

Cosine similarity

Score for each document?

$$\sum_i \text{doc}_i \text{query}_i$$

Sum of products.

If doc and query are length-normalized vectors, then this is the *cosine similarity* of the two vectors!

Cosine similarity

Score for each document?

$$\sum_i \text{doc}_i \text{query}_i$$

Sum of products.

If doc and query are length-normalized vectors, then this is the *cosine similarity* of the two vectors!

It is a similarity measure ranging from 0 to 1 which indicates how similar two normalized vectors are.

(This is a useful notion in many other situations)

Retrieval: find a document that is most similar to the query

Each document is represented by its tf-idf vector

- Each document is a tf-idf vector
- As in week 1: the resulting matrix is way too big
- As in week 1: represent only the non-zero values!

Wrap up

- *ranked* retrieval
- **tf-idf**: prefer
  - documents in which search term occurs more often
  - documents in which infrequent search term occurs
- A document can be represented by a normalized vector
- A query can be represented by a normalized vector
- Best document: which document is most similar to the query?
- Cosine similarity

Excursion: word2vec

Deep learning models create vectors.

Most famous example: `word2vec`

You can try it out on the LWP machine:

```
$ /net/aps/haytabo/src/word2vec/distance /net/shared/vannoord/word2vec/Dutch-words/vectors.bin
Enter word or sentence (EXIT to break): boom
Word: boom  Position in vocabulary: 2329

                                              Word       Cosine distance
------------------------------------------------------------------------
                                             bomen 0.657305
                                       kastanjeboom 0.571457
                                          appelboom 0.569731
                                    verlichtingspaal 0.550243
                                  elektriciteitspaal 0.548399
                                          huisgevel 0.533083
                                            plataan 0.532200
                                           knotwilg 0.531878
```

Compute analogy

France is to Paris as Spain is to X? Paris - France + Spain is ?

```
$ /net/aps/haytabo/src/word2vec/word-analogy /net/shared/vannoord/word2vec/Dutch-words/vectors.bin
Enter three words (EXIT to break): Frankrijk Parijs Spanje

                                              Word              Distance
------------------------------------------------------------------------

                                            Madrid 0.695480
                                         Barcelona 0.627260
                                            Milaan 0.591913
                                            Malaga 0.568099
                                           Sevilla 0.558453

...
```

Other examples

Rotterdam	Feyenoord	Amsterdam	**Ajax**
Vlaanderen	Vlaams	Wallonie	**Waals**
België	Ardennen	Nederland	**Veluwe**
Nederland	Beatrix	Engeland	**Elizabeth**
België	Dewinter	Nederland	**Wilders**
koning	koningin	prins	**prinses**
Nederland	Suriname	België	**Congo**
Frankrijk	Seine	België	**Thames**

Assignment

Search Engine for Wikipedia with ranked output, using tf-idf

Take the Wikipedia search engine from the first and second assignment as starting point to implement `query_tfidf.py`, a variant which uses the same data format as before, and which will return the Wikipedia articles in ranked order, where the order is determined by the tf-idf score. Every query consists of multiple search terms, and your engine returns all Wikipedia articles in which at least one of the terms occurs. The tf-idf score is printed for each article, and the articles are returned in order. For our purposes, it is easier if the higher scoring articles are shown last.

Assignment

To make the output both readable and not too large, for this assignment you should print (only) the title of each Wikipedia article (remember that the first line of each Wikipedia article is the title).

```
$ python3 query_tfidf.py small.txt

Two search terms (q to quit): ijsvogel kwak

8.7949764316887 7
Grijskopijsvogel .

8.7949764316887 7
Nederlandse spelling van dieren- en plantennamen .

9.893588720356881
Reigers .

26.38492929506631
IJsvogel .
```

Please note that we use the simple tf-idf definition here, we are not yet normalizing scores and not yet working with vectors - this we leave for next week.

Proposed data-structure for this week's exercise

- dictionary: doc-id $\rightarrow$ text (as before)
- dictionary: term $\rightarrow$ doc-id $\rightarrow$ tf-idf

```
{ t1 : { doc1 : 12.1,
         doc2 : 8.4,
         doc6 : 134.0 },
  t2 : { doc1 : 3.1,
         doc3 : 6.7 }
}
```