Information Retrieval and Web Search

Lecture 06

Lecture by Dr. Inbal Budowski-Tal
Typeset by Steven Karas

2018-05-09
Last edited 20:43:20 2018-05-09

Disclaimer These lecture notes are based on the lecture for the course Information Retrieval and Web Search, taught by Dr. Inbal Budowski-Tal at IDC Herzliyah in the spring semester of 2017/2018. Sections may be based on the lecture slides written by Dr. Inbal Budowski-Tal.

Agenda

- Scoring
- Term Weighting
- Vector Space Model

1 Recap

1.1 Heaps' Law

Heaps' law states that the number of unique terms is $K \cdot N^{\beta}$, where both K and β are derived empirically[1]. For English collections, K is typically between 30 and 100, where β is around 0.5. Note that Heap's law is linear in the log-log space. As an example, in the RCV1 corpus, $K = 44$, and $\beta = 0.49$.

As a practical matter, named entities tend to constitute the growth of terms in a corpus over time, rather than new functional words.

1.2 Zipf's law

Zipf's law states that the frequency of a term is approximately the inverse of its frequency rank. In formal terms, it's an estimate on the frequency of a given term with frequency rank i: $cf_i = c \cdot i^k$, where the constant terms are derived empirically[2].

1.3 Dictionary Compression

From least efficient to most efficient:

- Fixed-length records to fit largest term
- Pointers from records into a text pool
- Chunking multiple terms into one record (and save on offsets). However, need to save lengths in text pool.
- Front coding - common prefix for block, save suffixes.

$$\texttt{8automata8...10automation} \Rightarrow \texttt{8automat*a1}\circ\texttt{e2}\circ\texttt{ic3}\circ\texttt{ion}$$

1.4 Postings compression

Encode gaps between consecutive documents.

Variable Length Encoding uses the MSB to indicate if it's the last block.

[1] by linear regression in the log-log space
[2] again, by linear regression in the log-log space

1.4.1 Gamma Codes

Gamma codes embed a prefix-free self-synchronizing encoding. To encode a number, encode the offset of a number, which is the number with the leading bit chopped off (e.g. `13 = 1101` $\Rightarrow$ `101`). The length of this offset is then added as a prefix to the encoded term as an unary prefix:

$$\texttt{13} \Rightarrow \texttt{1110101}$$

For practical purposes, there is typically a huge performance penalty to be paid for bit manipulation and queries that aren't aligned at a word boundary, so variable length encoding is better for most purposes.

2 Ranked Retrieval

So far, we've only answered the boolean question of if a document matches a query or not. It is often impractical to craft a boolean query that returns only a handful of documents. To order results, we assign a score to each document, typically in $[0, 1]$.

2.1 Jaccard Coefficient

A common measure of similarity between two sets:

$$\begin{aligned}
&\textsc{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (A \neq \emptyset \vee B \neq \emptyset) \\
&\textsc{Jaccard}(A, A) = 1 \\
&\textsc{Jaccard}(A, B) = 0 \text{ if } A \cap B = \emptyset
\end{aligned}$$

Always assigns a score in $[0, 1]$, and doesn't assume anything regarding the size of the sets. However, it doesn't consider term frequency, relative position, or rarity (rare terms are typically more important). Also, we need to control for document length; later we'll use the cosine similarity:

$$\textsc{Cosine}(A, B) = \frac{|A \cap B|}{\sqrt{|A \cup B|}}$$

3 Term Frequency

Recall that we can represent the postings as an incidence matrix. This matrix can either record the appearance, or the actual frequency. A column vector in this matrix represents the frequency with which terms appear in the document. This is called the bag of words model, and it's in some senses a step back from positional postings.

The term frequency $tf_{t,d}$ of a term t in document d is the number of times that term appears in the document. However, relevance does not increase linearly, so we use the log term frequency:

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d} & \text{if } tf_{t,d} > 0 \\ 0 & \text{else} \end{cases}$$

The relevance of a document for a query is just the sum of each log term frequency.

3.1 Inverse Document Frequency (tf-idf)

Rare terms are more relevant for a query than frequent terms. Conversely, common words shouldn't provide much relevance for scoring, as they appear too often (e.g. stop words). We can use the number of documents in the collection the term appears in (the document frequency) for this. Note that there is a difference between the collection frequency, which is the frequency of a term across the collection, whereas the document frequency is the number of documents that include the term. Design concerns can sometimes guide us towards choosing a different scoring method. We define the new scoring measure where N is the number of documents in the collection as:

$$idf_t = \log_{10} \frac{N}{df_t}$$

$$w_{t,d} = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t}$$

4 Vector Model

If we apply the term scoring scheme to the incidence matrix, we get an embedding of our collection into a vector space. The axes of this space are the terms. Queries are represented as vectors in this space as well. However, we need a good similarity measure, as Euclidean similarity is not very good, since it doesn't compare vectors of different magnitude very well. The cosine of the angle is a good measure:

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

If we normalize all the vectors as part of constructing the index, this maps each document onto the unit sphere. A fully worked example of this is given on slides 55-57.

There are many different distance measures and variants on weighting, which can be found on slide 59. The weighting scheme need not be identical between document weighting and query weighting. As a result, we use `ddd.qqq` notation to indicate which weighting schemes we used. E.g. `lnc.ltn` which implies both use logarithmic term frequency, the query uses idf document frequency, and documents are cosine normalized.

If we compute the cosine similarity between the query vector and each document vector, then we can rank documents by similarity, and return the first K as results.

5 Next Session

We will continue with the example of the weighted tf-idf query next time.

References

[1] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.