Chapter 6/7:
Ranked Retrieval

Reading:

- Chapter 6

All our focus has been on Boolean retrieval, but is that really enough?

Ranked retrieval: rather than display all documents matching a query (what's wrong with that?), order the documents by "rank" and display in decreasing order. To do so, we need to compute a "rank" for each document in the corpus relative to a given query.

Zone indexes

An easy improvement to inverted indexes is a **zone index** which captues *metadata* information for documents in the corpus. Record information like *title*, *author*, etc. in a separate inverted index. Then when doing queries, use that index in addition to the content index to rank documents.

Example 1. `william -> 2.author, 2.title -> 3.author -> 4.title -> 5.author`

Weighted zone scoring:

We can run queries against a zone index to give greater weight to documents containing the terms depending on the zone in which they are found. Given a query q and a document d, a **weighted zone score** assigns a value in the range $[0, 1]$ to the pair (q, d) by computing a **linear combination** of zone scores. Suppose we have l zones (including document content). Choose weights g_i for each zone such that $\sum_{i=1}^{l} g_i = 1$. then to compute a score for (q, d) we find

$$Score\,(q, d) = \sum_{i=1}^{l} g_i \cdot s_{\mathrm{i}}$$

where s_{i} gives a Boolean value of 0 or 1 indicating whether the document satisfies the query in zone i.

Example 2. Example 6.1: Consider the query *shakespeare* in a collection in which each document has three zones: author, title and body. The Boolean score function for a zone takes on the value 1 if the query term shakespeare is present in the zone, and zero otherwise. Weighted zone scoring in such a collection would require three weights g_1 , g_2 and g_3, respectively corresponding to the author, title and body zones. Suppose we set g 1 = 0.2, g 2 = 0.3 and g 3 = 0.5 (so that the three weights add up to 1); this corresponds to an application in which a match in the author zone is least important to the overall score, the title zone somewhat more, and the body contributes even more. Thus if the term shakespeare were to appear in the title and body zones but not the author zone of a document, the score of this document would be 0.8.

Term frequency and weighting

All our search engines so far have stopped at "does the document contain these terms". Hence the name Boolean retrieval. Ranked retrieval does not use Boolean operators; it instead treats a query as a "bag of words". Such a *free text query* expresses an information need as a freeform text without any query operators. Ranked scoring is then calculating and summing scores for each term in the query over every document in the corpus.

If we want to *rank* documents, the first logical step is this observation: a document that mentions a term more frequently is more relevant to a query and should receive a higher score.

$\mathrm{w}_{d,t}$:

To do this calculation, we assign a **weight** to each term in a document that reflects the frequency of the term in the document. We call this $\mathrm{w}_{d,t}$ – the weight of term t in document d, and would like this number to reflect

how frequently the term occurs in the document. The easiest value for $w_{d,t}$ is $tf_{t,d}$, the exact frequency of the term in the document. We can now do a ranked retrieval for a query q by letting $Score(q,d) = \sum_{t \in q} w_{d,t}$

Ranked retrieval, first attempt:

For each term t in a query q, retrieve the postings list for t. For each document d in the list, calculate $w_{d,t}$. Sum $w_{d,t}$ terms for all documents in at least one of the postings lists from q. Voila!

Example 3. Build an inverted index from the document collection in Ebert: Lecture 1. Use the index to rank documents for the query "hot soup".

Immediate observation: **are all words in a query equally important?**

$w_{q,t}$**:**

No, not all terms in a query have equal importance. "los angeles angels of anaheim" – "of" is not particularly important. Why? Because it occurs in many many documents. In fact, we note that the importance of a term is roughly **inversely proportional** to its frequency in the collection. So we want to weight $w_{d,t}$ values by $w_{q,t}$ values indicating the importance of the term within the query.

First attempt: a term's importance is inversely related to its collection frequency cf_t.

Word	cf	df
try	10422	8760
insurance	10440	3997

cf_tunfairly punishes words that are distinguishing/important but occur very very frequently in certain documents in the collection. Instead we like df_t as a means of indicating importance: high df_t means low importance, and low df_t means high importance. Very few documents contain `insurance` and we want those that do to be weighted more heavily than those that contain `try`.

To get the inverse relationship, we use **inverse document frequency**

$$idf_t = \log \frac{N}{df_t}$$

Now when ranking, we scale $w_{d,t}$ values by $w_{q,t}$values, where $w_{q,t} = idf_t$. This is called **tf-idf weighting**, as the final calculation after substitution becomes $Score(q,d) = \sum_{t \in q} tf_{t,d} \times idf_t$. How does this give us good rankings? This score values documents d that have high frequency for terms that are in few documents in the collection.

The Vector Space Model

This approach to ranking "feels" nice, and it passes a spot-check of correctness: the more frequent a term is in a document, the higher that document will score, but that score will be weighted by how "distinguishing" the term is inside the corpus. Still, it would be nice if we had a mathematical model to serve as the foundation of scoring. Fortunately we do.

The **vector space model** for information retrieval views documents in the corpus as a T-dimensional vector of weights, where T is the size of the vocabulary. Such a vector is denoted as $\vec{V}(d)$ for some document d. Each of the components of the vector gives a **weight** in the document for a corresponding term from the vocabulary – for now, assuming this is $tf_{t,d}$ but realize it ultimately does not matter. Then the set of all documents in the corpus is just a set of vectors in T-dimensional space.

EXAMPLE OF VECTORS IN 2-DIMENSIONAL SPACE

We can compare two documents for "similarity" by using vector math. LOOK AT THE PICTURE: WHICH TWO DOCUMENTS ARE "MOST SIMILAR", and WHY? Because the **angle** between them is minimal. Finding the angle between two vectors is easy; we call this "cosine similarity",

$$\cos\theta = \frac{\overrightarrow{V}(d_1) \cdot \overrightarrow{V}(d_2)}{\left|\overrightarrow{V}(d_1)\right| \times \left|\overrightarrow{V}(d_2)\right|}$$

where θ is the angle formed by $\overrightarrow{V}(d_1)$ and $\overrightarrow{V}(d_2)$. We know that $\cos 0 = 1$, so the closer the two vectors are in angle, the closer their cosine approaches 1.

This gives us the math foundation for ranked retrieval: if we treat a query as another vector in the T-dimensional vector space model, then the "relevance" score for an arbitrary document in the corpus to that query can be found as the cosine similarity between the document's vector $\overrightarrow{V}(d)$ and the query's vector $\overrightarrow{V}(q)$.

Final scoring formula... for now

$\mathrm{tf}_{t,d}$ has major bias towards longer documents, which many not necessarily be more relevant. Example: take document A and make document B by pasting A's contents twice. Is B more relevant than A? No, but it will score higher. We want to normalize scores based on lengths of documents and lengths of queries. Thus score becomes

$$Score(q,d) = \sum_{t \in q} \frac{tf_{t,d} \times idf_t}{L_d \times L_q}$$

where $L_d = \sqrt{tf_{t1,d}^2 + tf_{t2,d}^2 + \ldots + tf_{tn,d}^2}$ and likewise for L_q. We note that L_q is constant for each document for a given query and that relative score orderings can ignore the term.