Chapter 6: Scoring, term weighting, and the vector space model

- Boolean queries are good for users with very precise undertstanding of their needs, and the collection.
  - Often results in either too few or too many results.
- Alternative: Free-text queries and Rank-order the documents
  - Free-text queries: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language.
- Three ideas:
  1. Parametric and zone indexes
     - To index and retrieve documents
     - Simple means of scoring
  2. Weighting importance of a term in a document, using statistics of occurrence
  3. Viewing each document as *a vector of weights*
     - Vector space scoring: to compute a score between *a query* and *each document*

6.1 Parametric and zone indexes

- Digital documents often have *metadata*
- One parametric index for each field
  - Support querying *ranges* on ordered values: Structures like B-tree may be used for the field's dictionary
- Zones: Similar to fields, but the contents can be arbitrary free text
  - Document titles, abstracts, etc.
  - The dictionary for a zone index must structure whatever vocabulary stems from the text of that zone.
- We can directly encode the *zone* in which a term occurs in the *postings*, and reduce the dictionary size
  - Also allows efficient computation of *weighted zone scoring*

6.1.1 Weighted zone scoring

- Given a boolean query q and a document d, weighted zone scoring assigns to the pair (q, d) a score in the interval $[0, 1]$
  - By computing a *linear* combination of **zone scores**: each zone of the document contributes a Boolean value.
  - The Boolean score from a zone would be 1 if *all* the query terms occur in that zone.
  - $\sum_{i=1}^{l} g_i \cdot s_i$, where g_i are weights given for each zone, and s_i is the score from each zone
- Weighted zone scoring is also referred to as **ranked Boolean retrieval**.

6.1.2 Learning weights

- How do we set the weights??
- Used to be set by 'experts,' but nowadays we learn them from curated training examples
- *Machine-learned relevance*

6.1.3 The optimal weight g

- Differentiation of total error

6.2 Term frequency and weighting

- A document or zone that mentions a query term *more often* should be given higher scores.
- Free text query: Terms are given without any connecting search operators - we simply view them as a set of words
  - Then we could simply compute the total score by summing up over each term a match score between each query term and the document
- We need to assign *weights* to each term in the document
  - The simplest approach: Use *term frequency* - Weights to be equal to *the number of occurrences* of term t in the document d.
- **Bag of Words Model**: Having number of occurrences as weights is a *quantitative digest* of the document; ignores the exact ordering of the terms
  - Intuitive that two documents with similar bag of words represnetations would be similar *in content*.

6.2.1 Inverse document frequency

- Using plain term frequency could be problematic when certain terms have very little or no discriminating power in determining relevance

    - Simple Solution: *Scale down* the term weights of terms with high *collection* frequency (total number of occurrences within the entire collection)
- *Document frequency*: The number of *documents* in the collection that contain the term
    - Document frequency and collection frequency could behave quite differently
- **Inverse document frequency (idf)**: $\text{idf}_t = \log \frac{N}{\text{df}_t}$

6.2.2 Tf-idf weighting

- Produce a composite *weight* for each term in each document, using term frequency and idf
- $\text{tf-idf}_{t,d} = \text{tf}_{t,d} \cdot \text{idf}_t$, where t is a term and d is a document
    - Highest when t occurs many times within a *small* number of documents (thus lending high discriminating power to those documents)
    - Lower when the term occurs fewer times in a document, or occurs in many documents
    - Lowest when the term occurs in virtually *all* documents
- We can now consider a document to be a *vector*
    - with one component corresponding to each term in the dictionary
    - together with a tf-idf for each component
- *Overlap score measure*: Sum up the tf-idf of each term in d

6.3 The vector space model for scoring

- Basic ideas underlying vector space scoring

6.3.1 Dot products

- How do we quantify the similarity between two documents?
- Simple idea: Measure the magnitude of the vector difference between the two
    - Drawback: Difference could be big, just because one is much longer than the other, even though the *contents* are quite similar
        * The relative distribution of terms could be quite similar, even when the absolute frequencies of one may be far larger.
- **Cosine similarity**: $\frac{V(d1) \cdot V(d2)}{|V(d1)| \cdot |V(d2)|}$
    - The numerator is the *dot product*: The cosine of the angle Θ between the two vectors
    - The denominator is the product of their Euclidean lengths: length-normalization
- **Term-Document Matrix**: $M \times N$ matrix
    - M terms
    - N documents
- Terms should be stemmed before indexing

6.3.2 Queries as vectors

- We can view queries as vectors in the **same vector space** as the document collection
- The number of dimensions will equal the vocabulary size M.
- A document may have a high cosine score for a query *even if it does not contain all query terms.*
- Computing similarities in tens of thousands of dimensions could be expensive

6.3.3 Computing vector scores

- We seek the K documents of the collection with the highest vector space scores on the given query.
- Term-at-a-time scoring or accumulation: Need to be maintaining weight values of each term t for document d, which could be wasteful as they are floating point values
    - We could instead simply store $\frac{N}{\text{df}_t}$ at the head of postings for t and $\text{tf}_{t,d}$ for each postings entry
- Select the top K scores would require a priority queue structure, often using a heap
    - $2N$ comparisons to construct
    - each of K scores can be extracted from the heap at a cost of $O(\log N)$ comparisons
- Document-at-a-time: We might be able to traverse the postings lists of the various query terms *concurrently* - We would then compute the scores, one document at a time

6.4 Variant tf-idf functions

6.4.1 Sublinear tf scaling

- It is questionable whether 20 times the occurrence necessarily indicates *20 times the importance*
- Alternative: Use the logarithm of the term frequency
- $\mathrm{wf}_{t,d} = 1 + \log \mathrm{tf}_{t,d}$ if $\mathrm{tf}_{t,d} > 0$, 0 otherwise
- $\text{wf-idf}_{t,d}$

6.4.2 Maximum tf normalization

- Normalize the tf weights of all terms occuring in a document by *the maximum tf in that document.*
- Let $\mathrm{tf}_{\max}(d) = \max_{\tau \in d} \mathrm{tf}_{\tau,d}$, where τ range over all terms in d.
- $\mathrm{ntf}_{t,d} = a + (1-a) \cdot \frac{\mathrm{tf}_{t,d}}{\mathrm{tf}_{\max}(d)}$
  - a is a *smoothing term*; values between 0 and 1 and is generally set to 0.4. *Dampens* the contribution of the second term
    - We want to avoid a *large swing* in ntf from modest changes in $\mathrm{tf}_{t,d}$.
- We want to use this because we want to deal with the cases of higher term frequencies in longer documents: longer ones tend to *repeat the same words over and over again*
- This method could be unstable in the cases like the following:
  - when the list of stop words changes
  - A document may contain an outlier term with an unusually large number of occurrences
  - If the most frequent term appears roughly as often as many other terms, compared to having a more skewed distribution, that should be treated differently.

6.4.3 Document and query weighting schemes

- *SMART* notation
- `ddd.qqq`: `ddd` represents the term weighting of the document vector; `qqq` indicates the weighting for the query vector
  - the first letter: term frequency
  - the second: document frequency
  - the third: normalization
- Quite common to apply different normalization to `d` and `q`

6.4.4 Pivoted normalized document length

- Normalizing each document vector by the Euclidean length...
  - Masks some subtleties about *longer* documents
    - Higher tf values
    - More distinct terms
- The nature of longer documents
  1. Verbose documents that essentially *repeat the same content*: the length does not alter the relative weights of different terms
  2. Documents covering *multiple different topics*: Search terms probably match *small segments* of the document but not all of it
     - Relative weights of terms are quite different from a single short document that matches the query terms
     - Need normalization that is *independent of term and document frequencies*
- Resulting normalized documents to be not necessarily of unit length
- *Pivoted document length normalization*: when computing dot product score with a (unit) query vector, the score is *skewed* to account for the effect of document length on relevance.
- Suppose that we have a document collection with an ensemble of queries
  - and Boolean judgments of whether or not each d is relevant to each query q.
- Then we could calculate a *probability of relevance*: a *function* of *document length*, averaged over all queries in the ensemble.
  - (Imagine an upward-sloping curve here)
- Cosine normalization equation has a tendency to distort the true relevance, at the expense of longer documents.
  - **Pivot length** l_p: the point where distortion trend changes
- Want to adjust this to match more closely to the true relevance curve: rotate the cosine normalization curve *counter-clockwise* about p
  - Use normalization factor *larger* than the Euclidean length for *each* documents shorter than l_p
  - Use normalization factor *smaller* than the Euclidean length for *each* documents longer than l_p
- Simple implementation: $a \cdot |V(d)| + (1-a) \cdot \mathrm{piv}$, where piv is the cosine normalization value at which the two curves intersect.
  - Still linear in $|V(d)|$, but slope < 1