Information Retrieval Models

Two phases:

- **Indexing**: happens offline while the system is setup
- **Scoring documents** for a particular query: online, depends on the query

Phases of indexing:

- Tokenization
- Stemming/Stop wording
- Storing the information on file with special structure for fast access during query time

Document scoring phase. We have a function or model which computes a score between a query and each document. The score is the "system's opinion" if a particular document is relevant. In order to create meaningful functions we need to make models of what a document is, how people write documents, how to decide whether a document comes from the correct distribution of relevant documents, etc.

So, in query time the following steps take place:

- Phase 1. Process query in the same way documents were processed during indexing. For example: if stemming was applied during indexing, the query should be stemmed
- Phase 2. Retrieve all the documents which contain at least one of the query terms. (Also possible: retrieve the documents which contain all of the query terms, we recommend the first one for the project). All other documents will have a score of 0 and will not be shown in the retrieved list.
- Phase 3. For each of the collected documents in Phase 2 compute the function **Score(doc, query).** This function will be different for different retrieval models.
- Rank the documents according to their scores in descending order. The document with highest score is the best match to the query.

The scoring function for all the models we consider will look like this:

```
Function Score(document, query)
        Accumulator = 0.0;
        For each word in query do
                qtf = number of times word appears in the query; qtf stands for query term
                                                                frequency

                tf = number of times word appears in document; tf stands for term frequency

                word_weight = weight(tf, qtf, and other stuff)
                accumulator = accumulator + word_weight
        EndFor

        Return accumulator
EndFunction
```

The Okapi Model (Okapi is the name of an animal related to zebra, the system where this model was first implemented was called Okapi)

Here is the formula that Okapi uses.

tf	is the term's frequency in document
qtf	is the term's frequency in query
N	is the total number of documents in the collection
df	is the number of documents that contain the term
dl	is the document length (in bytes), and
$avdl$	is the average document length

Okapi weighting based document score:

$$\sum_{t \in Q,D} ln \frac{N - df + 0.5}{df + 0.5} \cdot \frac{(k_1 + 1)tf}{(k_1(1-b) + b\frac{dl}{avdl}) + tf} \cdot \frac{(k_3 + 1)qtf}{k_3 + qtf}$$

k_1 (between 1.0–2.0), b (usually 0.75), and k_3 (between 0–1000) are constants.

(Source: Modern Information Retrieval: a brief overview by Amit Singhal)

This formula has three components:

- IDF component. This is
  ln [(N – df + 0.5) / (df + 0.5)]

This component reflects the discriminative power of each word. Examine for cases df = 0 and df = N.

- TF component. This is
  (k1 + 1) tf / [k1 (1 – b) + b dl/avgdl) + tf]
  Two things should be stressed here:
  - This function is increasing in tf, but reaches an asymptotic limit from below. This means whether a term appears a 100 times or a 1000 times the function will weight it almost the same.
  - There is a correction for document weight. If a document is short, the tf for all its words is increased; if a document is long the tf for all its words is decreased. The count of each word is measured w.r.t. the document of average length in the collection.
- QTF component. If a word in the query appears more times than another it should be weighted higher. This component is
  [(k3 + 1) qtf] / [k3 + qtf]

Function OkapiScore(*document*, *query*)

accumulator = 0.0;

For each *word* in query **do**

qtf = number of times *word* appears in the query; qtf stands for query term frequency

tf = number of times *word* appears in *document;* tf stands for term frequency

df = number of documents in which *word* occurs

N = total number of documents in the collection

dl = document length, total number of words in document (not number of unique words)

avgdl = the average of all document lengths

Const k1 = 1
Const b = 0.75
Const k3 = 1
(You may need to experiment a bit with the above values)

$$\text{idf} = ln\frac{N - df\,(\text{word}) + 0.5}{df\,(\text{word}) + 0.5}$$

$$\text{okapi\_tf} = \frac{(k_1 + 1)\,tf\,(\text{word , doc})}{(k_1(1 - b) + b\frac{dl}{avdl}) + tf\,(\text{word , doc})}$$

$$\text{query\_weight} = \frac{(k_3 + 1)qtf}{k_3 + qtf}$$

$$\text{word\_weight} = \text{idf} * \text{okapi\_tf} * \text{query\_weight}$$

accumulator = accumulator + word_weight

EndFor

Return accumulator

EndFunction

The Language Model with Laplace Smoothing
Smoothing means: change the zero probabilities to non-zero.

Function *LM-Laplace-Score*(*document*, *query*)
    Accumulator = 0.0;
    For each *word* in query **do**
        qtf = number of times *word* appears in the query; qtf stands for query term frequency

        tf = number of times *word* appears in *document;* tf stands for term frequency

        doc_len = document length, number of words in the document

        V = vocabulary size, number of unque words in the collection
        word_weight = qtf * Log ((tf + 1) / (doc_len + V))

        accumulator = accumulator + word_weight
    EndFor

    Return accumulator
EndFunction

Language Model with Jelinek-Mercer Smoothing

Function LM-Jelinek-Mercer-Score(*document*, *query*)
Accumulator = 0.0;
For each *word* in query **do**
qtf = number of times *word* appears in the query; qtf stands for query term frequency

tf = number of times *word* appears in *document;* tf stands for term frequency

doc_len = document length, number of words in the document

V = vocabulary size, number of unque words in the collection

p_doc = tf / doc_length

ctf = collection term frequency, number of times term occurs in the collection
M = number of total words (not number of unique words) in the collection

p_collection = ctf / M

//note: you may need to experiment with lambda
Const lambda = 0.2

p_interpolated = lambda * p_doc + (1 – lambda) * p_collection

word_weight = qtf * Log (p_ interpolated)

accumulator = accumulator + word_weight
EndFor

Return accumulator
EndFunction

More information:
Chapters 11 and 12 from the book:
http://www-csli.stanford.edu/~schuetze/information-retrieval-book.html