Information Retrieval: Assignment 2 - Retrieval System

member1@student.ethz.ch
member2@student.ethz.ch
member3@student.ethz.ch

November 30, 2015

1 Retrieval System

In this project, we have build a system which retrieves a ranked list of documents relevant to a query. We have used the provided collection of documents and we have evaluated the system performance on 40 queries and relevant statements.

The system retrieves the ranked document lists for multiple queries simultaneously. The process flow consists of several steps:

1. extract queries and relevance judgements from the files provided.
2. scan the collection of documents in a streaming manner, using *TipsterCorpusIterator* and build up maps with term and collection frequencies for the query words, inverse document frequencies and document lengths. This will be used by the different models to evaluate the relevance of a document.
3. for every query, calculate the score of every document in the collection and keep the top 100 documents in a map. The score is calculated either using the term-based model or the language model depending on the value of the boolean parameter "languageModel".
4. print the resulting ranking to a text file
5. evaluate the system's performance in terms of P, R, F_1 and MAP for the training set.

Both the queries and the documents are tokenized by splitting on various standard characters, stop words are removed and then each word is stemmed using the *PorterStemmer*. In the case of query words which contain hyphens, we have decided to keep both the hyphenated word and the individual terms to increase the chance of finding the term in the collection as queries are usually very short.

2 Models

We have experimented with various models and here we describe the best performing ones with the optimal parameters based on the test queries.

2.1 Term-based model

Initially we implemented the standard TF-IDF model as described in the lecture slides. As the performance was not satisfactory, we experimented with *augmented TF-IDF* and the *OKAPI BM25*[1] model. Both of those improved the performance over the standard function, with the **OKAPI BM25** ranking function **??** performing best. This was used in our system.

$$score(Q, D) = \sum_{q \in Q} IDF(q) * \frac{tf(q, D) * (k + 1)}{k(1 - b + b\frac{|D|}{avgDL}) + tf(q, D)}, \tag{1}$$

where k and b are parameters of the model controlling the scaling of the term frequencies and the normalization of the term frequencies by the document length.

The inverse documented frequency is defined as:

$$IDF(q) = \log \frac{N - df(q) + 0.5}{df(q) + 0.5}, \tag{2}$$

N - documents in collection, $df(q)$ - #docs in which q occurs

The IDF has the property that if a word occurs in more than half of the documents, the log term becomes negative, thus a document is penalized for containing the query word. This should not happen as we have removed stop words from the documents and queries, but just in case we limit the IDF to 0 from the bottom but capping the document frequency to half of the number of documents. This method has been used in other similar projects.

We experimented with different values for the model parameters and found the optimal values to be 0.35 and 1 for b and k respectively.

2.2 Language model

For the language model, we used the standard probabilistic scheme, where $P(q|d)$ is the probability of the query q given the document d and documents are ranked by their probability to generate q according to the maximum likelihood principle. Independence of query terms is assumed given the relevant document, so that the likelihood of the query is the product of the likelihoods of the individual query words **??**.

$$P(q|d) = \prod_{w_i \in q} P(w_i|d) \tag{3}$$

We used the observed term frequencies in the collection to estimate the posterior probabilities of the query words given a document - $\hat{P}(w_i|d)$. However, as not all query words occur in all documents, for the likelihood $P(w_i|d)$ we experimented with different types of collection smoothing - Jelinek-Mercer, Dirichlet and two-stage smoothing which combines both [2]. We ended up using the two-stage smoothing as defined in equation **??** because it proved to be the most effective on the training data.

$$logP(w_i|d) = log\Big[(1 - \lambda)\frac{\hat{P}(w_i|d) + \mu * P(w_i)}{|d| + \mu} + \lambda P(w_i)\Big], \tag{4}$$

[1] S. Robertson, S. Walker, M. M. Beaulieu, M. Gatford, and A. Payne. Okapi at TREC-4. In NIST Special Publication 500-236: The Fourth Text REtrieval Conference (TREC-4), pages 73 96, Gaithersburg, MD, 1995.

[2] Zhai, Chengxiang, and John Lafferty. "A study of smoothing methods for language models applied to information retrieval." ACM Transactions on Information Systems (TOIS) 22.2 (2004): 179-214.

The prior $P(w_i) = \frac{cf(w_i)}{|C|}$ is the collection frequency of the word normalized by the size of the collection (in terms of words) and λ and μ are model parameters.

In terms of interpretation, λ roughly controls the noise in the query, thus the bigger it is, the less weight is given to the query terms. On the other hand, μ is the standard Dirichlet prior smoothing parameter and controls the collection smoothing to account for words not present in the document.

The optimal values were determined to be 0.1 for λ, which is expected given the short queries, and 1000 for μ.

3 Evaluation on training data

To evaluate our model, we have used the provided *qrels* statements. Any document not explicitly stated as relevant in the relevance judgements is assumed to be irrelevant, thus the scores represent worst-case behaviour in terms of accuracy. For each query we report the precision (P), recall (R), the harmonic mean F_1 and the AP, as well as the MAP for the whole system. All reported values are at rank 100 and the total number of relevant documents is assumed to be $min(TP + FN, 100)$ for the average precision calculations.