Project Report 2

Lidia Freitas, Gokula Krishnan Santhanam and Ankit Srivastava

December 12, 2016

1 General Strategy

1.1 Common Preprocessing Steps

1. **Stemming** - We used the Porter Stemmer algorithm to reduce words to their word stems. This helps in greatly reducing the vocabulary size and reducing all the different forms of a word into its stem form.
2. **Stopword removal** - We also used the list of stopwords from the TinyIR library to remove low information words.

1.2 Approach

We first built an inverted index consisting of pairs: $term- > list((documentName, termFrequency))$. For each query, we do the same preprocessing as for the original documents, then consult the inverted index and retrieve the documents that have the terms that belong to that query. After, we consider two models: language model and query term model.

In the language model we calculate the probability that a query was generated from a given document, which is the value used for ranking the documents.

In the query term model we rank the documents based on the frequency of the query terms in the document as well as in the whole corpus. The higher the frequency in the document and the lower the frequency of the terms of the query in the corpus, the higher the score of the document for the given query. In other words, we sum the tf-idf scores of the query terms for each document and use it to score and rank the documents.

1.3 Measuring Results

We used bounded recall to calculate average precision as the per-query metric for each of the two models. Then we used mean average precision as system-wide metric to measure the overall performance of both the models on the 40 training queries provided.

2 Term-based model

For implementing term based model, we calculate the following values:

$$tfidf(w;d) = log(1 + tf(w;d))[log(\frac{n}{df(w)})]$$

$$score(d,q) = \sum_{w \in q} tfidf(w;d)$$

The result of the $score(d,q)$ function gives a ranking to document d based on query q.

2.1 Performance

The MAP score of all validation queries was 0.129166 and the total time of training and testing was 403 s, which is roughly 7 min.

3 Language Model

We realized that in the corpus some words were concatenated, for example, *George Bush* was present as *George-Bush*. Hence, for each word in the query, we looked for keys that contained it. This introduced many false positives and was time consuming because it iterated through all keys of the inverted index for each word in query. Unfortunately it didn't improve the score, so we left it out. In the model we tried to model the likelihood of a document generating the given query, for that we used the following formulas:

$$P(w|d) = (1 - \lambda_d)\widehat{P}(w|d) + \lambda_d \frac{cf(w)}{\sum_v cf(v)}, \lambda_d \in [0,1]$$

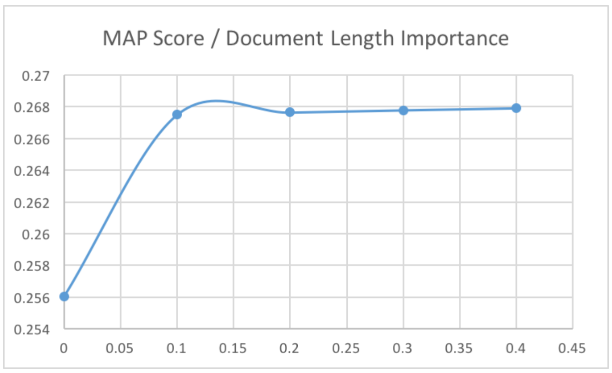
$$score(q,d) = \sum_{w \in q} logP(w|d)$$

The result of the $score(d,q)$ function gives a ranking to document d based on query q.

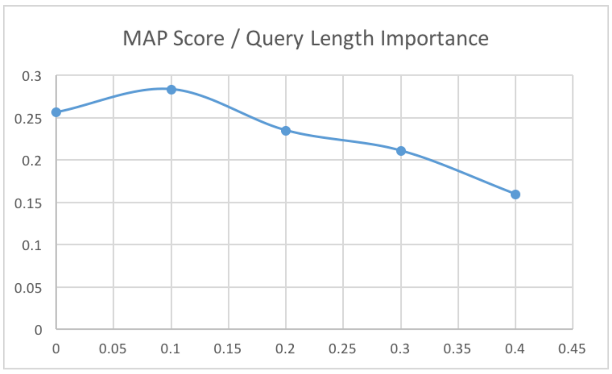
3.1 Tuning λ_d

We tried tunning the parameter λ_d in different ways:

1. **constant**
2. **weighting with document length**



3. **weighting with query length**



In the end weighted both the document length and the query length and tried to find the best combination of weights to get a good lambda. The higher the value of the weights in the plots, the lower the lambda.

3.2 Performance

The MAP score of all validation queries was 0.283147 and the total time of training and testing was roughly 689 seconds which is roughly 11 minutes.

4 Performance of Inverted Index

We observed that running the algorithm without the inverted index, traversing the document collection sequentially for each query, was really time consuming compared to the version with inverted index. We considered top 1000 words (after stopword removal and porter stemming) for each document, then the average time per query with inverted index was 0.0976 sec and without inverted index was 1.785 sec.

5 Conclusion

Performance-wise, we expected the language model to outperform the term model by a large margin and our results corroborate that. We also saw that inverted index was much faster than simply passing through the document collection sequentially for each query, which proves the importance of inverted index as a data structure in information retrieval systems.