Information Retrieval (Fall'16)
Assignment 2
Retrieval System

Andreas Georgiadis
15-926-710

Georgios Touloupas
16-932-550

December 14, 2016

1 Implementation Details

1.1 Data Preprocessing

In order for our models to be able to produce decent results, some text preprocessing steps are essential when reading the contents of the dataset. Firstly, words are stripped from all non alphanumeric symbols. Several words contained in the provided dataset are spelling mistakes since they consist of a concatenation of two words and therefore in the next preprocessing step we split the most common concatenated words using a list of spelling corrections for common mistakes contained in the `SpellChecker` object. Since we do not split the infrequent concatenated words we filter most of them out by removing words of over a certain length (`characterThreshold`), since we consider them the result of a concatenation and therefore a spelling mistake. Subsequently, known common non-informative words are removed using a stop words list. Finally, a stemming step using the Porter stemmer algorithm is applied on top of the remaining words.

1.2 Vocabulary Construction

To construct the vocabulary we scan the document collection for the first time. The scan is performed in batches to reduce the memory footprint. The vocabulary words are extracted and preprocessed using the steps described in Subsection 1.1 and their frequencies over the collection are calculated. Finally we prune the vocabulary by removing words with small collection frequency. This way we achieved to significantly reduce the memory footprint of our models as well as their training runtime, while not observing any notable performance degradation.

1.3 Inverted Index Construction

To construct the inverted index from words to document IDs we first build the forward index, scanning the document collection for the second time. The scan is performed in batches to reduce the memory footprint. For every document in the collection the words are extracted, preprocessed using the steps described in Subsection 1.1 and filtered with the pruned vocabulary and their document frequencies are calculated. Subsequently, the inverted index is constructed by using the forward index. To reduce the memory footprint of the index and the inverted index we use hashCodes to store words and document IDs as well as Short integers to store the term frequencies.

1.4 Term-based model

For the term-based model we use the maximum likelihood estimation model with smoothing described in the lecture slides.

$$\sum_{w \in q} \log P(w|d) = \sum_{w \in q \wedge w \in d} \log \left[1 + \frac{(1-\lambda_d)}{\lambda_d} \frac{\hat{P}(w|d))}{\hat{P}(w))} \right] + n \cdot \log \lambda_d$$

We implemented two variants of smoothing: Jelinek-Mercer smoothing (JM) and Bayes smoothing with Dirichlet Priors (DP). For JM smoothing $\lambda_d = \lambda$ is constant for all documents. For DP smoothing $\lambda_d = \frac{\mu}{|d|+\mu}$. As we can see DP smoothing applies smoothing proportional to the length of the document, while JM smoothing applies a fixed amount of smoothing, regardless of document length.

1.5 Language Model

For the language model, we utilize the latent topic model that was described in the lecture, that is the conditional probability of a token contained in the query is modeled by the quantity

$$P(w|d) = \sum_t P(w|t)P(t|d)$$

The aformentioned model is trained using the Csiszar-Tusnady Algorithm, that was presented in the lecture slides. We utilize the implementation provided in tinyIR, which we have slightly adapted, so that the document-topic probabilities $P(t|d)$ are calculated in parallel, using the parallel sequences framework of Scala.

In order to rank the candidate documents, we employ a two-stage ranking scheme. Initially, we rank the documents based on how many tokens of the query appear in the document and the documents which include the same number of tokens of the query are ranked based on the score $\sum_{w \in q} P(w|d)$. We observed a significant improvement in our results, compared to ranking the documents based only on the score $\sum_{w \in q} P(w|d)$.

2 Experiments

2.1 Varying the hyperparameters

The metrics used for this part of the report are bounded Recall, AP and MAP, since for most training queries the number of documents returned by our models (bounded by 100) is smaller than number of relevant documents.

2.1.1 Term-based model

The parameter exploration regarding the term-based model consists of varying smoothing variant as well as its hyperparameter. The values we try out for JM smoothing are `lambda = {0.1,0.3,0.5,0.7,0.9}`, while for DP smoothing we try out `mu = {10,100,1000,10000}`. Our findings are presented in Figures 1 and 2. The best MAP is achieved by the configuration with Bayes smoothing and $\mu = 100$. The metrics for this configuration are presented in the following table.

Metric	**Value**
Precision	0.3405
Recall	0.3531
F1	0.3450
MAP	0.2491

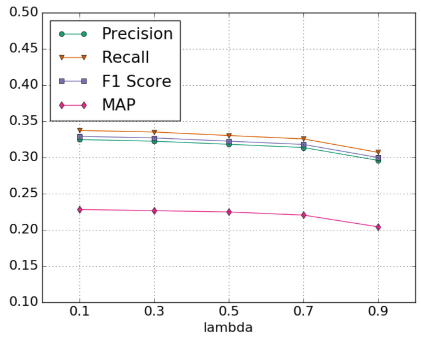

Figure 1: Term-based model with JM smoothing experiments

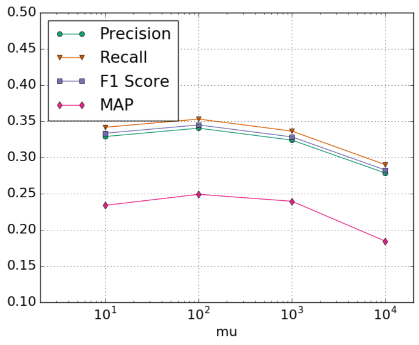

Figure 2: Term-based model with DP smoothing experiments

2.1.2 Language Model

The parameter exploration regarding the language model regarded the number of the latent topics, the vocabulary pruning threshold and the character pruning threshold. The values we try out are `numTopics = {100,200,300,400,500}`, `vocabularyThreshold = {5,10,15}` and `characterThreshold = {5,10,15}`. Throughout the parameter exploration experiments the default parameters were `vocabularyThreshold = 10`, `characterThreshold = 15`, `numTopics = 200`. Our findings are presented in Figure 3.

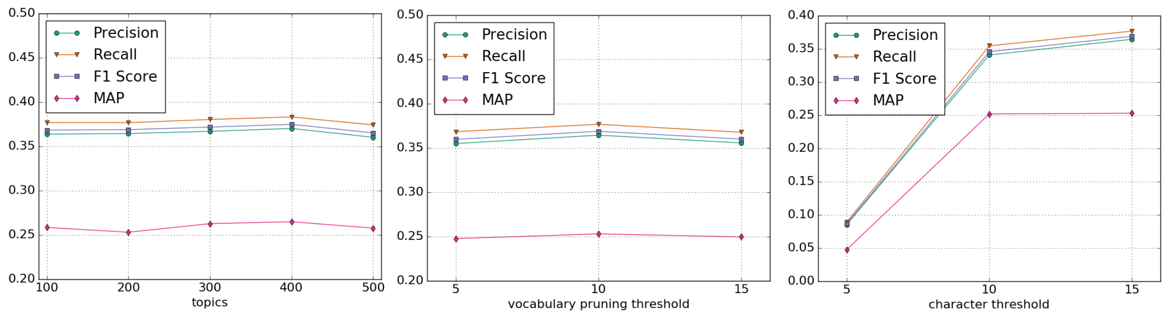

Figure 3: Language model parameter exploration

The best MAP is achieved by the configuration with `vocabularyThreshold = 10`, `characterThreshold = 15`, `numTopics = 400`. The metrics for this configuration are presented in the following table.

Metric	Value
Precision	0.3703
Recall	0.3832
F1	0.3749
MAP	0.2649