Project 2

Andrii Ihnatov
Taivo Pungas
Karim Labib

In this project, our goal was to implement two distinct models for scoring query-document pairs: the first one uses a term-based approach, and the other one is based on a generative language model.

Though it is really easier to use these two (tf-idf and generative language) models separately, in reality search engines use hundreds of different features and perform scoring and ranking of query – document pairs based on all of them. Therefore, in this project we decided to go with this approach of creating several different features augmented together, and then scoring each query-document pair based on them. We should also note that both tf-idf and language-based scores are included into the considered set of features.

In the following sections we will explain in detail feature generation and scoring processes, and provide the results obtained by the model on the training dataset.

Preliminaries:

First of all, stop words were removed from both the query and the document. So, they are used neither in a term-based nor in a language-based model.

Term-based features:

We created several different features for a term-based model. The first basic feature is just calculating a tf-idf score of a query-document pair.

Along with this feature, the following ones are used:

- The percentage of terms in common between the query and the title of the document. The intuition behind this is that if a document title contains words from the query, than the document is much more likely to be relevant. Before scoring, we used here Porter's stemmer on both query and the document title, since in this case it is cheap to compute it due their short lengths.
- Another similar feature is the calculation of tf-idf score for the first 20% of the document. Again, more relevant documents will most probably have words from the query in their top part.
- We also have a tf-idf score for 50% of the document with the same intuition as before.
- Normal tf-idf score for the whole document.
- We also added some extra basic scoring such as percentage of query terms found in the document and a term overlap which is equal to the number of occurrences of query terms in the document divided by the multiplication of document Euclidean length (in vector space representation) and the query length.

Language based model:

Several different ways of smoothing for generative language models are proposed in the literature. We implemented two of them. The first one is the *Jelinek-Mercer* method that was explained in the lecture. However, when reading a paper by Lafferty[1] we found that he was comparing in it the performance of different smoothing methods and there he mentioned that the results showed that "The Dirichlet prior method generally performs well, but tends to perform much better for concise title queries than for long verbose queries." This l inspired us to implement the Dirichlet prior method, which fits our setting perfectly since the query terms we have are normally short and concise.

The formulas for two smoothing methods implemented are as follows:

Method	$p_s(w \mid d)$	α_d	*Parameter*
Jelinek-Mercer	$(1-\lambda)\, p_{ml}(w \mid d) + \lambda\, p(w \mid C)$	λ	λ
Dirichlet	$\frac{c(w;d) + \mu\, p(w \mid C)}{\sum_w c(w;d) + \mu}$	$\frac{\mu}{\sum_w c(w;d) + \mu}$	μ

$$p(w \mid d) = \begin{cases} p_s(w \mid d) & \text{if word } w \text{ is seen} \\ \alpha_d\, p(w \mid C) & \text{otherwise} \end{cases}$$

We used $\mu = 1500$ since it gave the best MAP performance in our tests.

Classification step:

When choosing a classifier for our task, we have also referred to a real-world experience, which shows that boosting of trees over the features can be a good idea if absolute values of the features are less important than their relation between each other. So, after calculating all the previous features we constructed a feature vector for each query-document pair, and then passed it to a classifier, which in our case was Random Forest. To train it the provided *qrel* data was used.

Metrics calculation:

To assess an accuracy of our model, MAP score was used: first we calculated an average precision and then divided it by min(100, number of relevant documents for topic).

Results:

The result for the proposed model that uses all features mentioned before is equal to 24% MAP on the provided dataset;

```
----- Machine learning model ------
Mean precision:               0.262
Mean recall:                  0.109
Mean F1-score:                0.137
Mean average precision:       0.246
-----------------------------------
```

Besides this, we were also curious how does the language model perform separately on the same dataset. When running the Jelinek-Mercer smoothing computation, the mean average precision was equal to 12%.

When running using the Dirichlet smoothing, we found out that the mean average precision was equal to 17%, which affirmed the findings of the paper that Dirichlet smoothing gives better results for short and concise queries.

```
------ Language-based model -------
Mean precision:               0.321
Mean recall:                  0.133
Mean F1-score:                0.167
Mean average precision:       0.171
-----------------------------------
```

We also ran the algorithms without Porter's stemmer. Porter stemmer didn't affect much the machine learning model with all combined features. However it helped improve the results for the language model separately with an increase of MAP to 17% (compared to 14% without the stemmer). However, it is very expensive in terms of computation power. The results reported above are given **with** the Porter stemmer.

References:

[1] A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval/ John Lafferty & Chengxiang Zhai