Research Challenge

Daniel Carneiro Freire
daniel.carneiro.freire@gmail.com
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais, Brazil

ABSTRACT

In this research challenge we were tasked with coming up with different approaches for entity search, and submit the results of those approaches to a Kaggle competition.

1 INTRODUCTION

In this research challenge we were tasked with coming up with different approaches for entity search, and submit the results of those approaches to a Kaggle competition. In the following section we will briefly describe 5 of these approaches.

All the scores mentioned in this document were computed by Kaggle, and the metric used was NDCG@100. Also, all submissions made to Kaggle were named `out`, and the relevant submissions were discerned by their section titles written in the submission details of the submission. Also all submissions that required training were trained using the provided queries and qrels files.

2 INDEXING

For the experiments 3 relevant indexes were ultimately created. A sparce Lucene index using pyserini, a dense Faiss index using pyserini and a pyTerrier index. The pyserini's indexes were created using the default parameters, and ultimately was only successfully used for BM25. The pyTerrier index was the product of many iterations of adding more fields and parameters to the index to run more experiments. The final version has fields and metafields for text, title, list of keywords, concatenated keywords (space separated), and body, which is all fields together, also it has blocks enabled (Required for SDM).

3 BM25

This attempt consisted of using BM25 to rank the documents. To optimize the parameters a grid search was performed using `pt.GridSearch`, and on pyTerrier the optimal BM25 values were $c = 0.4, k_1 = 0.9, k_3 = 0.5$, with these parameters it scored ~ 0.36 on the test queries. However it was not the best attempt at using BM25, using pyserini's sparce Lucene index with the recommended parameters for MSMARCO of $k_1 = 0.82, b = 0.68$ scored 0.39578. This proved to be very hard to surpass.

This was the only experiment that will be described in this document that used pyserini, every other one used pyTerrier.

4 LAMBDAMART

LambdaMART was the first learning to rank approach attempted, and initially it provided very subpar results, the process of finding good features for the algorithm to use was a tedious one, based mostly on trial and error. The guiding idea for finding features was introducing variety, meaning that two features that act similarly (i.e. TF_IDF, BM25), would not be selected as individual features, instead the aim was introducing features that revealed something new about the queries.

After many iterations the best set of features was: BM25 over the text, BM25 over the title, BM25 over the concatenated keywords, coordinate match over the title, coordinate match over the text, and a pipeline of SDM(Sequential Dependence Model) to BM25.

This was the first experiment that surpassed BM25 (albeit by a very thin margin), scoring a whopping 0.39674. Being the first (of many attempts) to beat BM25 a lot of attention was given to this approach, and many feature combinations were tested, but no other combination of features was able to further improve the score.

5 BERT

The BERT approach was very straightforward, and pretty much only consisted of training the BERT transformer on the provided data. The only improvement that was made was the introduction of the "body" field to the index, which increased the score by almost 10%.

This submission scored 0.50443 and was the best attempt. Other experiments that were conducted with BERT were the usage of a sliding window passager to chunk each document in smaller parts, since BERT can't take too many terms as input (512), and use the max score over the passages as the relevant score. This performed much worse, scoring 0.43443. Another approach that was experimented on was using BERT as a feature for LambdaMART, which was also worse that just BERT by itself. We also tried to use the max, first, and mean score as features for learning to rank approaches (this part of the library was particularly buggy in my machine, and some of the code had to be rewritten for it to train the model, but even then it did not behave well with the LTR libraries).

This result was the best obtained throughout all experiments, in the next section we'll describe some experiments that did not even beat BM25, an lastly an effort to use BERT with LTR.

6 PRE-TRAINED RANKERS

A lot of pre-trained approaches were tested when searching for something that could beat my initial *BM25* attempt. This led me to try some pre-trained or partially pre-trained approaches. Namely all of the pyserini "reproduction guides" and EPIC (using an MS-MARCO checkpoint available at https://macavaney.us/epic.msmarco.tar.gz).

All of the reproduction guides when applied to our retrieval task performed very poorly (≤ 0.2), but probably would perform a lot better if they were trained with the provided qrels. The best result of this type of experiment was EPIC with a score of 0.32803, which underperformed BM25.

7 BERT + LTR

We executed a series of experiments on possible features using BERT, to be then fed to a learning to rank algorithm. The features we attempted to implement were: sliding windows to produce many BERT scores for each document; BERT trained only on the highly relevant qrels; BERT trained only on the lower relevance qrels; BERT trained only on titles, keywords and text. As mentioned before, sliding windows did not behave well with the LTR libraries, so we were left with the other features.

Two LTR algorithms were tested with those features, FastRank and LambdaRANK.

The idea of separating the qrels by relevance is that the fitting function for bert does not take into account how relevant a document is, only if it is or isn't relevant. However using these models as features did not perform better than bert alone, with the highest score being 0.48754 using FastRank.

The idea behind training an instance of BERT on each text field, was to: 1- mitigate the effects of the input size limitation of BERT; 2- The title and keyword fields are expected to have more relevant information, so maybe training models on those fields separately would reduce the noise when learning. This approach got a high score of 0.46183 using FastRank.

Another idea would be to train on both topics and relevance, totaling 6 different models, however the GPU used does not have enough memory for that. This approach could be further explored by testing different combinations of those features, and by implementing a sliding window that works properly.