Assignment 2

Silvan Robert Adrian
zlp432

***Abstract*—In this paper we compare 3 ranking methods and how they perform against each other. We accomplish this by using the `robust04` dataset with about 500'000 documents and run each of the methods on it (but only on a small subset of the data). To our surprize BM25 actually outperforms all with word embeddings extended methods, so adding more information to a query can even reduce the performance of the ranking.**

***Index Terms*—BM25, Word2vec, query extension, word embeddings**

I. Introduction

The problem of ad-hoc retrieval is to be able to find relevant documents according to a query. This query normally consists of a few words from which we can find documents which consists of those words or similar words depending on the approach. Okapi Bm25 is an example of such an algorithm to rank documents according to a query, BM25 already gets used in quite a few Softwares around the world (elasticsearch) and is therefore quite a stable algorithm.

This paper will reproduce 2 methods to improve the effectivness of BM25 by utilizing word embedings. As a dataset `robust04` (a trec dataset) is used which contains about 500'000 documents and 250 queries for a subset of the 500'000 documents.

The first method we are going to reproduce is the by extending the query (Saar Kuzi, Anna Shtok, Oren Kurland, 2016).

As a second method we have a look at `saliency-weigthed semantic network` (SWSN) from (om Kenter, Maarten de Rijke, 2015). The SWSN ranking function looks very similar to the BM25 but instead of using a simple bag-of-words approach we seek to find a better similarity measure by looking at the similarity of each word in each document to each word in the query.

Both of the methods will be compared to BM25 as a baseline.

So overall the difficulty in ad-hoc information retrieval is to actually find proper answers to what a user or a software is searching for in a collection of documents. So actually finding useful infromation and not totally useless answers.

II. Method

For all the following described methods we use the `anserini` library which offers us a out of the box BM25 on which we can work and improve on. Also for both methods with use Wored emeddings we used the Google News word2vec word embeddings (trained with 3'000'000 words).

BM25

BM25 ranking is calculated as follows, for a document D and a query $Q = q_1, ..., q_n$, with which we calculate the ranking score s:

$$s(D,Q) = \sum_{i \in Q} IDF(q_i) \cdot \frac{f(q_i, D)}{\cdot} K_1 + 1 \cdot \frac{f(q_i, D) + k_1}{1 - b + b \cdot \frac{|D|}{avgdl}} \tag{1}$$

$k1$ and b are free parameters which we can use to tune BM25 further that it fits better with our dataset. $f(q_i, D)$ is the term frequency of the query q_i in the document D. $|D|$ is the length of the document D and $avgdl$ is the average length of the documents of the whole collection. IDF (inverse document frequency), we calculate as follows:

$$IDF(q_1) = log\left(\frac{N - n(q_1) + 0.5}{n(q_i + 0.5}\right) \tag{2}$$

Here N is the total number of documents in the collection and $n(q_i)$ is the number of documents containing the term q_i.

Since we are using the `anserini` library this whole calculation is done for us and we don't need to do it ourselves.

BM25 Query extension

With this method we will use the same BM25 method as described above. The only thing that changes is the query itself, so we will add additional words which can be helpful for the improvement of finding relevant documents. For this we use two approaches, the centroid and the fusion method.

1) Centroid method: With the centroid method we take all the terms in the query Q into consideration by adding up all the query terms q_i and finding similar words to that new "centroid" vector.

$$\overrightarrow{q}_{cent} = \sum_{q_i \in Q} \overrightarrow{q}_{cent} \tag{3}$$

So with the calculated $\overrightarrow{q}_{cent}$ we now have to compute a relevance score for each term t in our word embedding to find the most similar words.

$$S_{Cent}(t; Q) = exp(cos(\overrightarrow{t}, \overrightarrow{q}_{Cent})). \tag{4}$$

so t here is the term from our word embedding, $\overrightarrow{t}$ is the vector representation of that term. Afterwords we calculate the cosine similarity between $\overrightarrow{t}$ and $\overrightarrow{q}$[1].

[1]In the implementation we use Gensim most_similar or similar functions to find nearby words without the use of exp

Additionally we can choose a number n which tells us how many similar words we want to add to the query, depending on the amount of additional words we might retrieve better results but this we will show in experiments. Those n terms we then add to the original query and search with that query.

2) Fusion method: With the fusion method we go through each query term and find n similar words and from them choose the v top terms. One approach is using the following method:

$$S_{CombMAX}(t;q) = max_{q_i \in q} p(t|q_i). \tag{5}$$

Here we get $n \cdot Q$ ranked terms but in case we get same term twice we will choose the one with the higher probability. Additionally to that we use softmax normalize for the similarities to get probabilities, as follows:

$$p(t|q_i) = \frac{exp(cos(\overrightarrow{t}, \overrightarrow{q}))}{\sum_{t' \in L_{q_i}} exp(cos(\overrightarrow{t}, \overrightarrow{q})} \tag{6}$$

As a last step as mentioned before we then take the top v terms and append it to the original query.

A. BM25 with word embeddings

By using embeddings we use the following method, which looks very similar to BM25 with the difference that we multiply by the semantic similarity. let f_{sts} be a function that returns a similarity score between 2 documents, s_l should be the longer text (mostly document) and s_s the smaller text (mostly query). The $avgsl$ is the average length of the longest text, then do:

$$f_{sts}(s_l, s_s) = \tag{7}$$

$$\sum_{w \in s_t} IDF(w) \cdot \frac{sem(w, s_s) \cdot (k_1 + 1)}{sem(w, s_s) + k_1 \cdot (1 - b + b \cdot \frac{|s_s|}{avgsl})} \tag{8}$$

w are the words which are contained in s_l, the parameters k_1 and b are free parameters again as in BM25 which can be tuned. IDF here also gets calculated as in BM25.

The function for semantic similarity is defined as: such:

$$sem(w, s) = max_{w' \in s}(w, w') \tag{9}$$

where we measure the semantic similarity between w and s_s

So we compare the two terms by their cosine similarity which tells us the semantic similarity, so $f_{sem}(w, w') = cos(\overrightarrow{w}, \overrightarrow{w'})$ After going through all documents we can sort according to the scores and we get the ranked documents.

Since we have very limited computational power, we decided to rank only a small amount first by BM25 and then rerank those by BM25 with word embeddings, with the implication that we might not get the documents we wished for. At reranking we process the documents and remove all unneeded texts, punctuation and so on. This we do for a better compatibility with the word embeddings. Additionally if it happens that a term w can't be found in the word embeddings we set the similarity to zero.

III. Experiments

For the experiments we use "Mean Average Precision" (MAP) for evaluation, so first a short explanation how MAP is calculated.

First we need to find the precision P of a single query which gives us:

$$P(k) = \frac{|\text{relevant documents}| \cap |\text{retrieved documents}|}{|\text{relevant documents}|} \tag{10}$$

Here k is the cut-off value (top k ranked documents for example). When we have the precision we can calculate the average precision as follows.

$$AveP = \frac{\sum_{k=1}^{n} P(k) \cdot rel(k)}{\text{number of relevant documents}} \tag{11}$$

And as a last step calculating the mean average precision by taking the mean over all queries q from set of queries Q:

$$MAP = \frac{\sum_{q \in Q} AveP(q)}{|Q|} \tag{12}$$

We will use map for cut-off values 5,10 and 20, as well as for all 250 queries.

A. Hyperparameter tunning

We split the queries we have into 80% training and 20% testing data. Afterwords we split up the 80% training data into 5 equal sized folds. Those folds we use then for parameter tunning on all methods and select the parameter with the best MAP and get the final MAP scores on the test data with the newly chosen parameters. Since the computational power is not available all following methods only have been running on a small set of the data.

B. Bm25 ranking

For the BM25 ranking we can tune the parameters $k1$ and b, we decided to use for $k1 = 0.5, 1, 1.5, 2.0$ and for $b = 0.25, 0.5, 0.75, 1$.

C. BM25 Query extension

1) Centroid: For the Centroid method we can only tune one single parameter and that is n (amount of similar words), for the experiments we decided to use for $n = 1, 2, 3, 4$.

2) Fusion: The Fusion method offers 2 paramters (n,v), we decided to run it with for $n = 1, 3$ and $v = 1, 3$.

D. BM25 Embeddings

For the Bm25 embeddings we also have $k1$ and a b, parameter (which are different from the BM25), for the inital ranking of BM25 the default values ($k1 = 1.5$ and $b = 0.75$) get used. For the reanking with the embeddings we then go through the folds and find the best parameter for $k1 = 1, 1.5, 2$ and $b = 0.5, 0.75, 1$.

IV. Results

After running the cross-validation and tunned the parameters, we ended up on the following MAP scores for the 3 different methods, C is the centroid method and F the fusion method.

MAP	**BM25**	**BM25+QE**	**BM25+WE**
All Topics	0.2326	C: 0.1832, F: 0.1903	0.1458
Cut off 5	0.0914	C: 0.0751, F: 0.0778	0.0414
Cut off 10	0.1358	C: 0.1089, F: 0.1100	0.0577
Cut off 20	0.1773	C: 0.1408, F: 0.1420	0.0897

Discussion

According to the results we get that BM25 actually still has the highest score, even though we tried to improve it with query expansion and word embeddings. BM25+WE would be interesting to run on all the data or actually all the methods would be great to run on the whole dataset. Also the initial ranking for BM25+WE should be removed and directly calculate the scores and ranks instead of reranking it from BM25 then we might also achieve a higher score. But overall in our experiements BM25 on it's own still shows the best results.

References

[1] Saar Kuzi, Anna Shtok, Oren Kurland, Query Expansion Using Word Embeddings

[2] Tom Kenter, Maarten de Rijke, Short Text Similarity with Word Embeddings

[3] Okapi BM25, https://en.wikipedia.org/wiki/Okapi_BM25

[4] Word2Vec, https://code.google.com/archive/p/word2vec/

[5] Evaluation measures (information retrieval), https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)#Mean_average_precision