Information Retrieval Assignment 2

Anon
Department of Computer Science - KU

TODO: write a real abstract.
I have implemented almost everything, save for cross validation and some minor left out equation for one of the methods. It took me a lot of time to get "anserini" working properly, so this has been my main factor for not writing a complete report. Also, clean up the code a bit and comment.

Introduction

The challenge of ad-hoc information retrieval is to match a query (typically of a very short length) against a large collection of documents, each with a length ranging from a few words to thousands of sentences (Bhaskar Mitra and Nick Craswell, 2018). Some information retrieval systems might utilize metadata attached to the documents. Search engines like Google and Yahoo might examine number of views, downloads, links referring to the document (or web page) or recently updated dates. But when no metadata are available, the information retrieval system has to rely on the raw text of the documents themselves. This is the task we focus on, ranking documents in a collection using only the raw text of said documents. The Okapi BM25 algorithm is already able to this job, and has been implemented efficiently in (Castorini, 2019).

The purpose of this paper is to reproduce two methods that can improve the effectiveness of the BM25 algorithm. Both of these methods utilizes word embeddings (vector representation of words in a high dimensional space). The dataset used is the *robust04* collection, which can be found on (TREC, 2004). It contains 500k documents with 250 querys, each manually labelled for a subset of the documents with relevance scores.
The first method implemented is the query expansion method from (Saar Kuzi, Anna Shtok, Oren Kurland, 2016), which simply adds terms to the query itself, before being given as input to the ranking function (in our case, the BM25). The added terms are selected by their similarity to the terms in the query, using a word embedding. These terms are then appended to the query itself with the intention of broadening the possibility of matching the query with relevant documents.
The second method integrates a word embedding with the ranking function itself. Specifically, we will examine the effectiveness of the *saliency-weigthed semantic network* (SWSN) from (Tom Kenter, Maarten de Rijke, 2015). The ranking function's mathematical equation may look like the BM25, but it is actually a wholly different equation. SWSN compares two texts just like BM25. However, BM25 uses a bag-of-words approach, meaning that BM25 mainly uses term and document frequencies to rank their documents. SWSN aims to inspect semantic similarities instead, which is done by examining the similarity of each word in each document of the collection, with each word in a given query. As such, SWSN seeks to establish a similarity measure between two text holistically. The word embeddings are used as a means to compute similarity scores for pairs of words.

Both of the above two methods will be evaluated against a baseline run of the BM25 algorithm.

Method

The query expansion method from (Saar Kuzi, Anna Shtok, Oren Kurland, 2016) trains a word embedding from the collection that is to be searched within, which are all of the 500k documents. On the other hand, the SWSN does not make such an assumption, any word embedding can be plugged in and utilized. Since we have limited computing power, we will not be training word embeddings ourselves. Instead, the GLoVe 300-dimensional pretrained word embedding will be used for both methods (which can be found at (Jeffrey Penning, 2004)).

For all of the three methods implemented (original BM25, query expansions, SWSN) the `anserini` library (Castorini, 2019) will do a lot of work for us. We will describe more precisely exactly how much work it does for us, in each of the following three method descriptions:

BM25

The BM25 algorithm is based on a bag-of-words representation of the collection vocabulary. That means it does not care about semantic similarity or ordering of the words (Wikipedia, 2019).
For a document D, a query $Q = \{q_1, q_2.., q_n\}$, BM25 returns

the ranking score s:

$$s(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{\text{avgdl}}\right)}$$

where $f(q_1, D)$ is the term frequency of q_1 in the document D, $|D|$ is the length of the document D, avgdl is the average length of the documents in the whole collection that D is drawn from. k_1 and b are free parameters, which we will tune in the next section for optimal ranking scores. The IDF component is computed by:

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

where N is the total number of documents in the collection and $n(q_i)$ is the number of documents containing the term q_i (Wikipedia, 2019).

All of this is implemented very efficiently in the `anserini` library, ready for convenient use.

Query expansions

This method is actually a collection of two methods, a *centroid-based* method and a *fusion-based* method. Both methods are meant to be run before the ranking function, and both methods aims to find similar words in a query, and then append these words to the query (hence, query expansion). The expanded query is then fed into the BM25 algorithm, and everything proceeds as explained for BM25 .

The centroid method. The first step is to build up a vector representation of the whole query, Q. This is done by adding the vector representations of the terms q_i, within Q:

$$\vec{q}_{cent} = \sum_{q_i \in Q} \vec{q_i}$$

The next step is to compute a relevancy score for each term t in our word embedding [1]. The score will be computed by:

$$S_{cent}(t,Q) = \exp\left(\cos(\vec{t},\vec{q}_{cent})\right)$$

where t is a term in our word embedding, $\vec{t}$ is the vector representation of t and $\cos(\vec{t},\vec{q})$ is a function that calculates the cosine similarity between $\vec{t}$ and $\vec{q}$.

Now we choose a number v, which represents how many words we wish to extend our query Q with. We compute the relevancy score for each term in our word embedding, and then extract the v highest ranked terms, and append these to our original query.

Fusion-based methods. In the fusion based methods, we start by choosing a number n. Then for each query term, we gather the top n most similar terms t, from the word embedding, in a list, L_{q_i}, by using the cosine similarity function denoted by $\cos(\vec{t},\vec{q})$. In (Saar Kuzi, Anna Shtok, Oren Kurland, 2016), the terms are softmax normalized to yield probabilities, since they use the method in conjunction with a language model. We will not be using these language models, but we do however still normalize the terms by the same formula:

$$p(t|q_i) = \frac{\exp\left(\cos(\vec{t},\vec{q})\right)}{\sum_{t' \in L_{q_i}} \exp\left(\cos(\vec{t},\vec{q})\right)}$$

From this equation, three methods were originally used in (Saar Kuzi, Anna Shtok, Oren Kurland, 2016) to obtain similarity scores in order to the expand the query. We have implemented two of them:

$$S_{combSUM} = \sum_{q_i \in q} p(t|q_i)$$

$$S_{combMAX} = \max_{q_i \in q} p(t|q_i)$$

The result are $n \cdot |Q|$ ranked terms. As with the centroid approach, we choose the top v terms, and append these to the original query.

TODO: implement softmax normalization

Saliency-weighted semantic network

SWSN tries to establish a similarity on a short-text level between two documents, and in our case will try to establish this similarity between our query and our documents. As mentioned earlier, SWSN might look like the BM25 algorithm, but actually is quite different.
Let f_{sts} be a function that returns a similarity score between two documents, let s_l be the longest of the two texts, let s_s be the shorter text and avsgl be the average length of the longest text, then:

$$f_{sts}(s_l, s_s) = \sum_{w \in s_l} IDF(w) \cdot \frac{sem(w,s_s)\cdot(k_1+1)}{sem(w,s_s)+\left(1-b+b\cdot\frac{|s_s|}{avgsl}\right)}$$

where w are the words contained in s_l, IDF is the inverse-document frequency identical to the one from BM25, k_1 and b are tunable parameters, also as for BM25.
The function $sem(w, s_s)$ is a measure for establishing the semantic similarity between w and s_s:

$$sem(w, s_s) = \max_{w' \in s_s} f_{sem}(w, w')$$

[1] As a note, (Saar Kuzi, Anna Shtok, Oren Kurland, 2016) uses all of the unique words from the whole collection, we will restrict ourselves to the terms in the pretrained GLoVe embedding.

where $f_{sem}(w, w') = \cos(\vec{w}, \vec{w'})$.
After iterating through all documents, we sort the scores in descending order and then the relevancy ranking is established.

Note, due to a limit in computational power, not all 500k documents are being examined. We make an initial run with the original (and very efficient) BM25 for ranking m documents, and then we rerank these m documents using SWSN. At the time of reranking, we preprocess the documents to lowercase every word in the document, for better compatability with our word embedding. Finally, if a term w found in a document does not appear in the word embedding, we assign the similarity of w with document s_s to zero.

Experiments

The evaluation measure used are the "Mean Average Precision". Let us explain this measure. First, we need to calculate the precision P of a single query, whcih gives us the fraction of relevant documents out of all the retrieved documents:

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

where k denotes the cut-off value, i.e. the k top ranked documents. Now the average precision AvP is computed, which means that we try to establish how well our ranking sequence order is aligned to the ground truth ordering:

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

Finally, we compute the mean average precision MAP, by taking the mean of AvP over all our queries q from our set of queries Q:

$$MAP = \frac{\sum_{q \in Q} AvP(q)}{|Q|}$$

We will evaluate this measure at cut-off values of 5, 10 and 20, as well as all of our 250 queries.

For hyperparameter tuning, we first split the queries randomly into a training set of size 80%, and set aside the remaining 20% for testing. Then we split the training set into 5 equally sized folds. For each fold, we use all of the queries in the fold, and store the MAP value of running the original BM25 in a list, for some pair of values of k_1 and b. The grid search values are $k_1 = \{1.2, 1.4, 1.6, 1.8, 2.0\}$ and $b = \{0.50, 0.75, 1.0, 1.25, 1.50\}$. Then we iterate through the lists obtained for each (k_1, b) pair, and compute the average MAP value from the cross-validation procedure within these lists. Afterwards, we select the (k_1, b) pair with the highest average MAP value, and compute the MAP value on the test set the we set aside in the beginning of the procedure.
Due to our limited computing power, we will reuse these optimal values on the query expansion method and SWSN for evaluating them.

TODO: implement cross-validation procedure

Results

Only results on a very small subset has been run at the moment. The query expansion method and SWSN does not seem to do very well with this implementation.

TODO run the experiment, get results, write tables.

Discuss

TODO: Maybe the GLoVe word embedding is not very usable for this data. A lot of query expansions are words that are *very* similar to the words. E.g. "International Organized Crime" becomes "International Organized Crime crimes criminal world" with the centroid based method.
Discuss the rest when results are in. By using BM25 to do initial ranking, we are already limiting ourselves in our ranking, especially if the initial ranking did not return very many relevant documents.

References

Bhaskar Mitra and Nick Craswell. (2018). An Introduction to Neural Information RetrievalâĂİ, Foundations and Trends. *Information Retrieval: Vol. 13, No. 1, pp 1âĂŞ126*, *13*(1), 1–126. doi: 10.1561/1500000061

Castorini. (2019). *Anserini: Open-source information retrieval toolkit built on lucene.* Retrieved from `https://github.com/castorini/anserini`

Jeffrey Penning, C. D. M., Richard Soscher. (2004). *Glove: Global vectors for word representation.* Retrieved from `https://nlp.stanford.edu/projects/glove/`

Saar Kuzi, Anna Shtok, Oren Kurland. (2016). Query Expansion Using Word Embeddings. *CIKM*, 1929–1932. doi: http://dx.doi.org/10.1145/2983323.2983876

Tom Kenter, Maarten de Rijke. (2015). Short Text Similarity with Word Embeddings. *CIKM*. doi: http://dx.doi.org/10.1145/2806416.2806475

TREC. (2004). *Trec 2004 robust track guidelines.* Retrieved from `https://trec.nist.gov/data/robust/04.guidelines.html`

Wikipedia. (2019). *Okapi bm25.* Retrieved from `https://en.wikipedia.org/wiki/Okapi_BM25`