Assignment 2 Report

Mario Giulianelli
University of Amsterdam
11567252

Akash Raj K N
University of Amsterdam
11617586

Florian Mohnert
University of Amsterdam
11770929

KEYWORDS

Lexical Retrieval, Latent Semantic Models, Learning to Rank

ACM Reference Format:

Mario Giulianelli, Akash Raj K N, and Florian Mohnert. 2018. Assignment 2 Report. https://github.com/akashrajkn/info-retrieval

1 INTRODUCTION

In this short report, we discuss our experiments with a diverse set of ranking algorithms. First, each method is presented along with its optimal hyperparameters, obtained using a validation set. Next, we scrutinize the performance of lexical retrieval methods, latent semantic models and learning to rank methods, each on a test set of query-document pairs, and considering four evaluation metrics: MAP@1000, NDCG@10, Recall@1000, and Precision@5. To better understand the differences in the methods, we examine a few hand-picked queries where particular methods score exceptionally good or bad in terms of NDCG@10. We finally make an inter-class comparison of the different approaches implemented for this homework.

2 EXPERIMENTS

2.1 Lexical retrieval methods

2.1.1 Models and hyperparameters. The following six lexical retrieval methods were compared: tf-idf, BM25, Jelinek-Mercer, Absolute Discounting, Dirichlet prior smoothing, and the Positional Language Model. Each of these comes with its own set of hyperparameters, which were tuned on a validation set of 30 queries. A listing of each of these methods is presented below along with the hyperparameters that yielded the best retrieval performance. Only tf-idf, for which term frequency is defined in terms of raw counts, is hyperparameter-free.

BM25:	$k_1 = 1.2,\ b = 0.75$
Jelinek-Mercer:	$\lambda = 0.1$
Dirichlet-prior:	$\mu = 500$
Absolute discounting:	$\delta = 0.9$
Positional LM:	$kernel$ = passage, $b = 1500$

We expect BM25 to be among the best lexical retrieval methods as it is repeatedly reported to be so in the related literature. Less confidence can be assigned to tf-idf, which is regarded as a rather elementary statistics because it does not include any kind of smoothing. As a consequence, we expect that the language models outperform tf-idf as they do discount probability mass from highly frequent terms. Finally, we presume that the positional language model will perform well on queries that consist of sequence of words that co-occur more often than would be expected by chance (i.e. collocations).

2.1.2 Comparison of all methods. To compare the lexical retrieval methods, we computed their mean performance on a test set of 120 queries and used a paired t-test to measure the differences between all possible pairs of methods. As each pair of methods is compared on 4 different measures, we ought to account for the multiple comparison problem. We used a Bonferroni correction and hence a t-statistic is significant when the p-value is below $0.05/4 = 0.0125$.

2.2 Latent semantic models

2.2.1 Models and Hyperparameters. Next to lexical features, we experimented with distributional semantics methods, Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA). These methods are expected to bridge the semantic gap between the wording of queries and the wording of the documents which satisfy the users information needs.

For latent semantic models to accommodate to the task of information retrieval, we need to be able to build a representation of a query q, a representation of a document d, and then calculate a similarity between q and d. For LSI, we construct a tf-idf model of the document collection and use it to build an LSI model of the corpus with a reduced dimensionality—i.e. number of topics—of 20. Queries are projected to the same 20-dimensional space. The similarity between query and document distributional representations is computed using cosine distance. As for LDA, model parameters are also estimated on a tf-idf representation of the document collection. With the learned model, topic distributions for queries can be inferred. These probability distributions cover 100 topics and they are compared using the Kullback-Leibler divergence.

Both LSI and LDA come with various hyperparameters to tune on the validation set. For LSI, we restricted our parameter search to the number of topics, the type of algorithm (one-pass merge algorithm or multi-pass stochastic algorithm), and the batch size—referring to the number of documents processed at a time. The batch size (or *chunk size*) is a tradeoff between speed and memory needs. For LDA, we optimized the number of topics and the *chunk size*. The parameter α of the per-document topic distributions, and β, the parameter of the per-topic word distribution, were automatically set by learning an asymmetric prior directly from the data.

, ,
https://github.com/akashrajkn/info-retrieval

Metric	**MAP@1000**	**NDCG@10**	**Recall@1000**	**Precision@5**
tf-idf vs. BM25	-0.0358 (p < 0.00)	-0.0787 (p < 0.00)	0.0001 (p = 0.3)	-0.0700 (p = 0.01)
tf-idf vs Jelinek-Mercer	-0.0316 (p < 0.00)	-0.0720 (p < 0.00)	0.0001 (p = 0.3)	-0.0800 (p < 0.00)
tf-idf vs. abs. discounting	-0.0225 (p < 0.00)	-0.0483 (p = 0.04)	-0.0001 (p = 0.3)	-0.0534 (p = 0.05)
tf-idf vs. Dirichlet	-0.0328 (p < 0.00)	-0.0758 (p < 0.00)	-0.0001 (p = 0.3)	-0.0666 (p = 0.02)
tf-idf vs. PLM	0.1077 (p < 0.00)	0.2127 (p < 0.00)	-0.0001 (p = 0.3)	0.2157 (p < 0.00)
BM25 vs. Jelinek-Mercer	0.0041 (p = 0.09)	0.0067 (p = 0.39)	0.0001 (p = 0.3)	-0.01 (p = 0.41)
BM25 vs. abs. discounting	0.0131 (p < 0.00)	0.0304 (p = 0.02)	-0.0001 (p = 0.3)	0.0166 (p = 0.31)
BM25 vs. Dirichlet	0.0028 (p < 0.00)	0.0030 (p = 0.73)	0.0001 (p = 0.3)	0.0033 (p = 0.81)
BM25 vs. PLM	0.1434 (p< 0.00)	0.2914 (p < 0.00)	0.0001 (p = 0.3)	0.2858 (p = 0.81)
Jelinek-Mercer vs. abs. discounting	0.0089 (p = 0.01)	0.0238 (p = 0.04)	-0.0001 (p = 0.3)	0.0132 (p < 0.00)
Jelinek-Mercer vs. Dirichlet	-0.0011 (p = 0.63)	-0.0037 (p = 0.65)	-0.0001 (p = 0.3)	0.0133 (p = 0.31)
Jelinek-Mercer vs. PLM	0.1393 (p < 0.00)	0.2848 (p < 0.00)	0.0001 (p = 0.3)	0.2958 (p < 0.00)
Abs. discounting vs. Dirichlet	-0.0102 (p < 0.00)	-0.0275 (p < 0.00)	0.0001 (p = 0.3)	-0.0132 (p = 0.39)
Abs. discounting vs. PLM	0.1303 (p < 0.00)	0.2610 (p < 0.00)	0.0001 (p = 0.3)	0.2692 (p < 0.00)
Dirichlet vs. PLM	0.1405 (p < 0.00)	0.2884 (p < 0.00)	-0.0001 (p = 0.3)	0.3832 (p < 0.00)

Table 1: Comparisons of lexical IR methods

LSM	**MAP@1000**	**NDCG@10**	**Recall@1000**	**Precision@5**
LSI vs. LDA	0.0260 (p = 0.01)	-0.0397 (p = 0.06)	0.0001 (p > 0.0125)	-0.0600 (p = 0.03)

Table 2: Comparison of Latent Semantic Models

2.2.2 *Comparisons.*

To compare the distributional semantic models on the test set, we first obtained an initial top-1000 ranking for each query using tf-idf, and then re-ranked the documents using LSI and LDA. The performance of the two methods on the 120 validation queries is reported on NDCG@10, Mean Average Precision (MAP@1000), Precision@5 and Recall@1000, and compared using the two-tailed paired t-test. As the semantic models are compared on 4 metrics, we again accounted for the multiple comparison problem using a Bonferroni correction. Thus the t-statistic is significant when the p-value is below $0.05/4 = 0.0125$.

2.3 From word to document and query embeddings

The following task was to compute word embeddings for the terms in the collection using the word2vec algorithm (Mikolov et al., 2013). We used the Skip-gram model with a context window of size 5 and obtained 300-dimensional word embeddings. For terms that occurred less than 5 times in the collection, no embeddings were computed. Next, we combined the word embeddings to obtain document and query representations. For queries we followed the naive approach of averaging the embeddings of all the words contained in the query. The motivation for this approach is that queries mostly consist of very few words in this collection. On the other hand, for document embeddings we used two different weighting schemes: (i) weighting each unique term in the bag-of-words representation of a document by tf-idf, or (ii) weighting each such term using BM25. We chose the log normalized version of term frequency and the standard inverse document frequency for each term. Finally, a document embedding is computed by taking the weighted sum of the unique word embeddings of each document. As word2vec is known to be less accurate when it is trained on relatively small corpora such as the one available for our experiments, we also applied the same weighting methods to pre-trained word vectors from Google released for this task.

2.4 Learning to rank

Besides lexical and distributional semantics methods, we also experimented with learning to rank algorithms. We implemented two approaches: pointwise and pairwise learning methods. In terms of accuracy, we expect the pairwise method to perform better as its cost function takes the nature of the ranking task into account. As we use the test set for training, the two approaches are compared on the provided validation set. In the following sections the pointwise and the pairwise algorithms are presented.

2.4.1 *Pointwise Learning to Rank.*

For pointwise learning to rank, we implemented a logistic regression model that simply learns a mapping f between a feature matrix X and binary output labels $\vec{y}$:

$$f(X) = \vec{y}$$

Each row of the feature matrix corresponds to a query-document level feature consisting of 8 dimensions, which correspond to the scores obtained with the following methods: tf-idf, BM25, Jelinek-Mercer, Dirichlet prior, absolute discounting, LSI, LDA and PLM. We constructed features in this fashion in order to make use of the

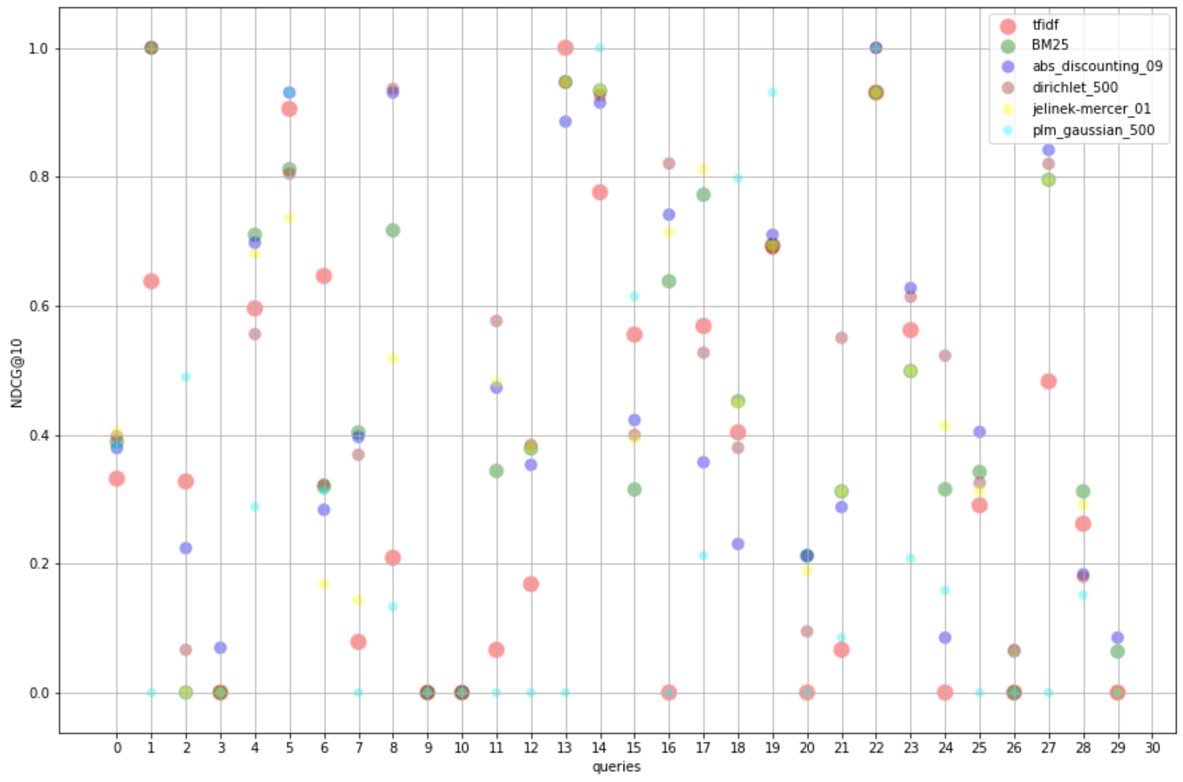

Figure 1: NDCG@10 on validation set

difference strengths of these scoring functions and hence to represent as many aspects of the query-document pair as possible. The logistic regression model was trained using 10-fold cross validation on the test set.

2.4.2 Pairwise Learning to Rank. Next, we applied the RankNet algorithm (Burges et al., 2005), a neural ranking algorithm in which the cost function minimizes the number of incorrect inversions of document ranking pairs. For our experiments, we used an open-source implementation of Ranknet [1]. The model was trained (i) on 8-dimensional features constructed as before, (i) on a concatenated version (600-dimensional) of the document and query embeddings, and (iii) on the addition of the two embeddings, resulting in a 300-dimensional vector. Here we deploy the pre-trained word2vec embeddings as we expect them to be more accurate due to the extremely large number of words over which they were estimated. We keep the model architecture constant across models, i.e. we use 2 hidden layers with 512 and 128 units respectively. RankNet implements a regression task and consequently the neural net includes a single output node.

2.5 Inter-class comparisons

As a final step, we performed an inter-class comparison of all the models we implemented. For this purpose, we devised an extra test set by sampling 30 queries out of the union of test and validation set.

3 RESULTS

3.1 Lexical retrieval methods

We report mean differences and p-values of all tests in Table 1. When compared on MAP@1000, NDCG@10 and Precision@5, tf-idf performs worse than all the other methods, except the positional language model. This could be due to the lack of smoothing in this method. BM25 on the other hand, is significantly better than the other methods on MAP@1000, except Jelinek- Mercer (compared to which it also has a higher score but not significantly so). Interestingly, for NDCG@10 and Precision@5, BM25 still scores the highest on nearly all comparisons but not significantly so (with the exception of the difference with PLM). Results for Jelinek-Mercer show that it is superior to absolute discounting, tf-idf and PLM, but not to Dirichlet and BM25.

A more general observation is that Recall@1000 is not a very informative measure in this case, because the number of documents that could possibly be retrieved for different queries in the test set is mostly below 1000. This means that recall will be 1 most of the times regardless of which method is chosen to retrieve documents. Also, MAP@1000 seems to be the most sensitive measure as it shows more significant differences compared to other measures. Especially, Precision@5 does not lead to significant differences as often,

[1] github.com/shiba24/learning2rank

which could be due to the fact that there is more variation in Precision for different queries. NDCG@10 displays the same pattern but shows significant differences slightly more often. An example is the comparison between BM25 and Dirichlet prior smoothing, where the difference in MAP@1000 is significant whereas NDCG@10 and Precision@5 do not confirm this trend.

Overall, it is difficult to make conclusive statements about the performance of the lexical methods on the test set. It is very query-dependent which method achieves the highest score on either of the evaluation metrics. Indeed, for this assignment, we were asked to examine queries where a particular method performs particularly well or rather poorly compared to the others. The following observations are based on a plot of NDCG@10 values on the validation set (30 queries), which can be found in Figure 1:

(i) **Query 18**: "leveraged buyouts". In this example, the positional language model wins. The words "leveraged" and "buyouts" occur next to each other very frequently (376 times) and for all bigrams where "buyouts" is the second term, "leveraged" is the first term 54% of the time.
(ii) **Query 6**: "protect u s farmers". In Query 6, tf-idf has the highest NDCG@10 score. A possible explanation is that most terms in this query have a high frequency in the validation set. Hence, the tf-idf model, which does not discount probability mass by smoothing as all the other techniques do, gives an advantage to highly frequent words. It is also worth pointing out that the PLM models the second highest scoring one, which can be again explained by the same pattern as in query 18 but this time for the tokens "u" and "s".
(iii) **Query 16**: "commercial overfishing creates food fish deficit". In this example the term "overfishing" occurs very infrequently and hence we think that that could be the reason that tf-idf has the lowest score, because it does not use any smoothing.
(iv) **Query 170**: "consequences implantation silicone gel breast devices". Here, it is quite interesting that PLM performs the worst. We are thinking that some subsets of this query occur often within the same passage and therefore documents are retrieved that only contain information relative to this subset.

3.2 Latent semantic models

The mean differences and the p-values of all significance tests made to compare LSI and LDA are reported in Table 2.

Retrieving documents using Latent Semantic Indexing yields a positive significant difference, with respect to Latent Dirichlet Allocation, in the Mean Average Precision (MAP@1000) score of the retrieved set of documents. When it comes to Normalized Discounted Cumulative Gain (NDCG@10) and Precision@5, LDA obtains higher scores, but not significantly. The values for recall (Recall@1000) are virtually equal for the two methods. As we observed above, recall is equal to 1 most of the times regardless of the retrieval method as the number of documents that could possibly be retrieved per query, in the test set, is often below 1000.

Overall, it seems difficult to draw partisan conclusions from the results of our significant tests. Indeed LSI is significantly better than LDA in terms of Mean Average Precision, but our previous tests have shown—as we discussed in Section 3.1—that MAP@1000 appears to be the most sensitive measure as it exhibits significant differences more often than the other measures. Moreover, on NDCG@10 and Precision@5, it is LDA which slightly outperforms the opponent.

	NDCG@10
LogReg ($x \in \mathbb{R}^8$)	**0.4091**
RankNet ($x \in \mathbb{R}^8$)	0.3209
RankNet ($x \in \mathbb{R}^{300}$)	0.1980
RankNet ($x \in \mathbb{R}^{600}$)	0.3001

Table 3: Learning to Rank

	NDCG@10
BM25	0.4951
LDA	0.3324
LogReg ($x \in \mathbb{R}^8$)	**0.5213**
RankNet ($x \in \mathbb{R}^{600}$)	0.2604

Table 4: Inter-class comparison

3.3 Learning to rank

In Table 3, we report NDCG@10 scores for the different learning to rank models. The simple logistic regression model with constructed features scores the highest on NDCG@10, followed by the RankNet, also with the same 8-dimensional features. Hence, it seems that models which use high-dimensional, dense embeddings seem not to provide as much discriminatory information about the relative relevance of documents given a query. The lower performance of RankNet could be attributed to the use of a relatively small dataset to train the complex neural network architecture and to the time limitations that kept us from performing extensive parameter optimization.

3.4 Inter-class comparison

Finally, we report NDCG@10 for all the models implemented in this assignment on a new sampled test set of 30 queries in Table 4. From these results, it is difficult to tell whether there is a silver bullet for information retrieval on this dataset. However, it is interesting that a simple and efficient method such as BM25 performs better than more complex models. The highest scoring model is the logistic regression model that uses our constructed 8-dimensional features. Juxtaposing features and combining them straightforwardly with a logistic regression model seems to capture the diverse information provided by the different features (see Section 3.1) and to constitute an advantage over the individual model scores used as features.

4 CONCLUSIONS AND OUTLOOK

In this report, we discussed our experiments with lexical retrieval methods, latent semantic models and learning to rank algorithms.

These approaches were evaluated on MAP@1000, NDCG@10, Recall@1000, and Precision@5.

For lexical retrieval and LSMs, the outcomes of intra-class comparisons did not provide us with clear winners. While BM25 seems to often outperform similar methods, there exist some query types where other approaches yield better retrieval scores. On the other hand, LSI and LDA obtain very similar results.

Inter-class comparisons have shown that combining all available features to train a logistic regression classifier yields the best retrieval results on our dataset. However, BM25 seems to be a very robust competitor. This result is remarkable as BM25 is a less complex and more efficient method than most of the ones it outperforms.

In future, we plan to experiment with reducing the dimensionality of our mixed feature vector including 8 different measures. We expect the 8 scores to be, to some extent, correlated. Using e.g. Principal Component Analysis, we would like to select the number of dimensions that retains the ideal amount of explained variance. Moreover, we will experiment with various ways of combining word embeddings into query and document embeddings: (i) merging word embeddings recursively based on binary syntactic parses (ii) using a recursive autoencoder to obtain sentence representations (Socher et al., 2011) for all the sentences in a document, and averaging over them to obtain a document embedding. These approaches are rather oriented towards document representations. For query representations, it is hard to combine word vectors according to a traditional kind of syntactic or semantic structure, as queries are subject to different writing practices.

REFERENCES

Burges, Chris, et al. "Learning to rank using gradient descent." Proceedings of the 22nd international conference on Machine learning. ACM, 2005.

Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

Socher, Richard, et al. "Dynamic pooling and unfolding recursive autoencoders for paraphrase detection." Advances in neural information processing systems. 2011.