Information Retrieval 1 – Project 2 – Accuracy of different retrieval methods

Juan Buhagiar – 11576014
University of Amsterdam
Amsterdam, The Netherlands
juan.buhagiar@student.uva.nl

Dimitrios Nikolopoulos – 23456789
University of Amsterdam
Amsterdam, The Netherlands
dimitris.nikolopoulos@student.uva.nl

Sarantos Tzortzis – 11863331
University of Amsterdam
Amsterdam, The Netherlands
sarantos_tzortzis@icloud.com

https://github.com/sarantinio/Information-Retrieval.git
January 27th, 2018

1 INTRODUCTION

In the present report we present the results of Homework 2 upon all required tasks.

All the experiments were conducted upon the standard TREC dataset AP88-89; a subset of the Associated Press collection from the years of 1988 and 1989. This dataset is accompanied with a set of 150 topic multi-termed queries used in all of our retrieval approaches. The given set includes a test set and a validation set, with each of which containing a relevance label for a set of topic query-document pair, in the TREC relevance file format.

2 EXPERIMENTS AND RESULTS

In this section, we present the conducted experiments for each of the four tasks. We give a brief description of the general setup for each task, accompanied with implementation details for every component of it. All of the following subsections include a description of the results with some visualizations and tables, wherever it was necessary.

Many auxiliary parameters and structures were already precomputed by given code. Such entities are: *the length of each document, the inverted index, the collection frequencies, the vocabulary of each document* etc. During the implementation, we made use of these parameters and structures whenever it was needed.

2.1 Task 1: Lexical IR methods

In this section, we describe the implementation of the Lexical Information Retrieval Methods. First, we give an overview of the general setup which includes how the ranking procedure is performed and how the ranked documents are evaluated. Then, details of each method are presented, such as the assumptions and restrictions that we posed and information about the hyperparameter optimization (wherever it was applicable). At the end of this section, a discussion upon the experiments and results is presented.

General Setup

Two core procedures describe the general setup of this task: 1) a procedure which run the retrieval process and 2) a procedure which run the evaluation of a ranking process.

The former one runs a given lexical retrieval method (e.g. TF-IDF, BM25, Language Model) upon all topic queries and writes the TREC-friendly results in a file. Names that characterize the retrieval method are given to the files. This method is responsible for calculating the scores for each topic query - document pair. The score of each topic query for a document is the sum of the scores of each topic query term upon the document. For that reason, any probabilistic model (i.e. language models) are transformed into a logarithmic scale. This also avoids any possible underflow in the calculations.

The evaluation procedure invokes the TREC evaluator through a script given an evaluation set and a run file. It returns all the results for NDCG@10, MAP@1000, Precision@5 and Recall@1000 evaluation metrics.

As for the hyperparameter optimization the default method we used is a simplified form of grid search. We tried all the given ranges upon the hyperparameters and chose the value that scores the best upon the validation set. The evaluation score we used was nDCG@10.

TF-IDF

The TF-IDF weight of a term upon a document is the product of its td weight and its idf weight. For each topic query and each document, we calculated the TF-IDF score by summing up all the individual scores of each term. No hyperparameter optimization was needed here.

BM25

Usually, the BM25 weighting scheme [2] contains 3 hyperparameters: k_1, b, k_3. Parameteter k_1 is set to 1.2 and the higher than 1 it is, the more it reduces the fast saturation of the term frequencies. Parameter b plays a role in the normalization of the document lengths. It controls the level of this normalization and for the purposes of this exercise it is set to 0.75. Parameter k_3 is applied when a topic query is very long, such as a paragraph, and it is used to scale the query length. Since the given queries contain only a few terms, we decided to drop this hyperparameter. Hence, the final retrieval scoring method for BM25 is characterized by the following function:

$$BM25 = \sum_{t \in q} \frac{(k_1 + 1) tf_{d,t}}{k_1((1 - b) + b(\frac{l_d}{l_{avg}})) + tf_{d,t}} idf(t)$$

Table 1: Jelinek-Mercer: nDCG@10 scores for different λ

Hyperparameter λ	nDCG@10 score
0.1	**0.3991**
0.5	0.3823
0.9	0.3676

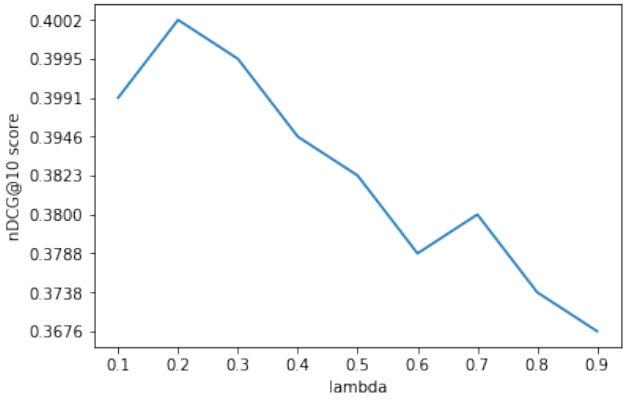

Figure 1: Jelinek Mercer scores upon validation set for different λ values.

Language models

a. Jelinek-Mercer Smoothing. Jelinek-Mercer is a Multinomial Smoothing method that estimates the score for a term w given a document d by using interpolation. Jelinek-Mercer probability of a given term and document is estimated as follows:

$$\hat{P_\lambda}(w \mid d) = \lambda \frac{tf(w;d)}{\mid d \mid} + (1-\lambda)\frac{tf(w;C)}{\mid C \mid}$$

We observe that the λ value either works mostly in favour of the relative frequence of a term in a document either in favour of the background probability over the whole collection of documents.

As mentioned in the homework requirements, we explored different values of λ within the range of {0.1, 0.5, 0.9}. Table 1 shows the nDCG@10 score upon the validation set. The rows with the bold number indicates the highest achieved score of 0.3991, which is for $\boldsymbol{\lambda = 0.1}$. From this we can directly conclude that by giving a 0.9 weight to the background probability, higher scores are achieved.

Figure 1 shows nDCG@10 scores for a wider range of λ parameters. We observe that the highest scores are achieved in values which are between 0.1 and 0.3.

b. Dirichlet Prior Smoothing. Dirichlet Prior Smoothing for a term given a document is estimated by the following equation:

$$p(w \mid \hat{\theta_d}) = \frac{tf(w;d) + \mu p(w \mid C)}{\mid d \mid + \mu}$$

We can observe that this is again an interpolation between the Maximum Likelihood Estimator and the background probability. Very high values of hyperparameter μ work in favour of the background probability. On the other hand very low values are in favour of the MLE.

Table 2: Dirichlet Prior Smoothing: nDCG@10 scores for different μ

Hyperparameter μ	nDCG@10 score
500	**0.4055**
1000	0.4002
1500	0.4026

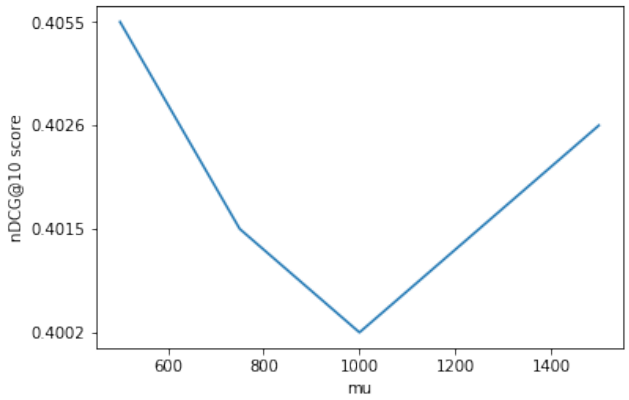

Figure 2: Dirichlet Prior scores upon validation set for different μ values.

Table 3: Absolute Discounting: nDCG@10 scores for different δ

Hyperparameter δ	nDCG@10 score
0.1	0.3614
0.5	0.3768
0.9	**0.3950**

As requested, we explored different values for μ within the range of {500, 1000, 1500}. Table 2 summarizes the results and shows that the highest score of 0.4055 was achieved for $\boldsymbol{\mu = 500}$.

Figure 2 shows the results for a denser range of μ values. The highest score is achieved at 500.

c. Absolute Discounting. Absolute Discounting score is estimated by the following equation:

$$p_\delta(w \mid \hat{\theta_d}) = \frac{max(tf(w;d) - \delta, 0)}{\mid d \mid} + \frac{\delta \mid d \mid_u}{\mid d \mid} p(w \mid C)$$

Hyperparameter δ lowers the ML probability by subtracting itself from the count of the words in a document. We explored the nDCG@10 score for δ ranging between the values of {0.1, 0.5, 0.9}, as asked in the homework. Table 3 summarizes the results upon the validation set. We observe that the highest score 0.3950 was received for $\boldsymbol{\delta = 0.9}$

Figure 3, we show once again a denser hyperparameter range. Highest score is observed at 0.8 and we notice that the there is an ascending trend with several local optima in between 0.1 and 0.9.

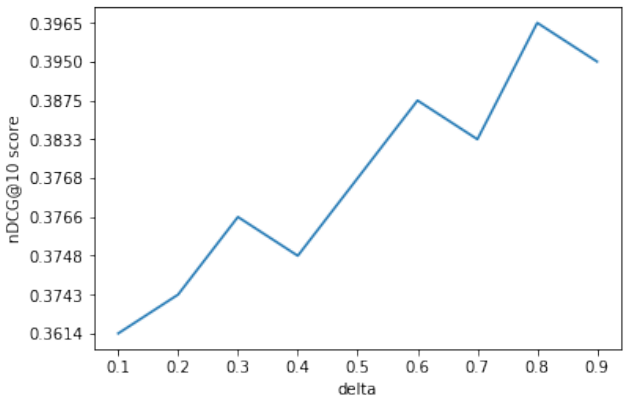

Figure 3: Absolute Discounting scores upon validation set for different δ values.

d. Positional Language Models. A Positional Language Model (PLM) is an estimated model based on propagated counts of words within a document through a proximity-based density function, which both captures proximity heuristics and achieves an effect of "soft" passage retrieval. For the implementation we followed the guidelines given in [4].

The implemented kernel functions are:

(1) Gaussian
(2) Triagnle
(3) Cosine
(4) Circle
(5) Passage

Hyperparameter σ is set to 50 by default. The language model probability of a query upon a certain position of a document is estimated through the Dirichlet Prior Smoothing:

$$p_\mu(w \mid D, i) = \frac{c'(w, i) + \mu(w \mid C)}{Z_i + \mu}$$

The scoring of a Query on a certain document position is estimated through KL-divergence.

$$S(Q, D, i) = -\sum_{w \in V} p(w \mid Q) \log \frac{p(w|Q)}{p_\mu(w \mid D, i)}$$

We estimate $p(w \mid Q)$ with the Maximum Likelihood Estimator $\frac{tf(q;Q)}{|Q|}$. Only the query terms contribute to the score, since other terms produce zero result to their contribution in the summation since the MLE of them is 0.

Finally, we receive a final score by using Best Position Strategy.

PLMs turnt out to be computationally expensive, therefore it was decided to make some optimizations, restrictions and assumptions:

- The most computationally complex part is the computation of the normalize lengths Z_i. Following the simplification posed by [4] and the code from the same authors [3] each of the first four kernels is calculated through their Cumulative Distribution Functions (CDF):
  (1) Gaussian: $\sqrt{2\pi\sigma^2}[\Phi(\frac{N-i}{\sigma}) - \Phi(\frac{1-i}{\sigma})]$
  (2) Triangle: $\Phi(\frac{N-i}{\sigma}) - \Phi(\frac{1-i}{\sigma})$
  (3) Cosine $\Phi(\frac{N-i}{\sigma}) - \Phi(\frac{1-i}{\sigma})$
  (4) Circle $\Phi(\frac{N-i}{\sigma}) - \Phi(\frac{1-i}{\sigma})$
  , where N is the document length, σ is the given hyperparameter set to 50 and i is a position in the document. $\Phi(\cdot)$ being the CDF of each kernel. For more details on how the CDFs are calculated one can look in the accompanied notebook, at the Cumulative Distribution Functions section.
- After simplifying the calculation of the normalized lengths Z_i we are creating a dictionary which precalculates these values for any possible position within a range of 1 till l, where l is the length of the largest document in our collection. This guarantees an immediate access to these values during the ranking. The dictionary is stored in a file named 'Z.pickle'.
- Since it is also computationally expensive to compute the propagated counts k(i,j) of each kernel during the retrieval, we decided to create all possible propagated count values beforehand and store them into a file 'kernel_tables.pickle'. This also guarantees $O(1)$ access.
- We create a dictionary which maps each word appearing in each document to a list of the positions that it occurs. This is a very important speed up, since the propagated counts need fast access to the positions of the query terms. This process dropped from 30 seconds per document to 3 seconds per document (tested with profilers).
- Our intention was to rank all the documents with the PLM model. However, the ranking of all the documents for each query and for each kernel needs a huge amount of time to be calculated. For that reason, we decided to re-rank the top k documents of the results of the best performing ranking method. TF-IDF scored the one of the highest values of NDCG@10, therefore we decided to re-rank the results of it using PLM. Due to time constraints, we have applied the hyperparameter optimization and the scoring upon the validation set by re-ranking only the top 100 ranked documents of TF-IDF. After the re-ranking we appended the rest of the TF-IDF ranked documents for each query to the results of PLMs.

After implementing each of the PLM components according to the aforementioned optimization assumptions, the optimization of the Dirichlet Prior hyperparameter μ and the choice of the best performing kernel should be performed. Our initial purpose was to first choose the best kernel by setting a stable value for μ, since the performance of each kernel function is not highly affected by the smoothing parameter. Then, after choosing the best kernel, we would like to tune the Dirichlet Prior upon it. However, we wanted to receive some representative results, therefore it was decided to perform a parallel optimization.

We optimized the hyperparameter μ upon the validation set for values ranging in {500, 1000, 1500}. Moreover, we applied those values to every possible kernel function. The re-ranking of the top-100 ranks of TF-IDF upon the validation set are summarized in Table 4.

The best results for each μ are highlighted. Among all kernel, it is clear that the **Triangle kernel** performs the best for any μ value. The highest score is achieved at $\boldsymbol{\mu = 1000}$ with a value of 0.3877.

Table 4: nDCG@10 scores upon the validation set for each kernel and for each chosen value of the Dirichlet hyperparameter μ.

	Gaussian	Triangle	Cosine	Circle	Passage
μ = 500	0.3713	**0.3838**	0.3734	0.3770	0.3773
μ = 1000	0.3726	**0.3877**	0.3754	0.3739	0.3725
μ = 1500	0.3759	**0.3817**	0.3690	0.3756	0.3703

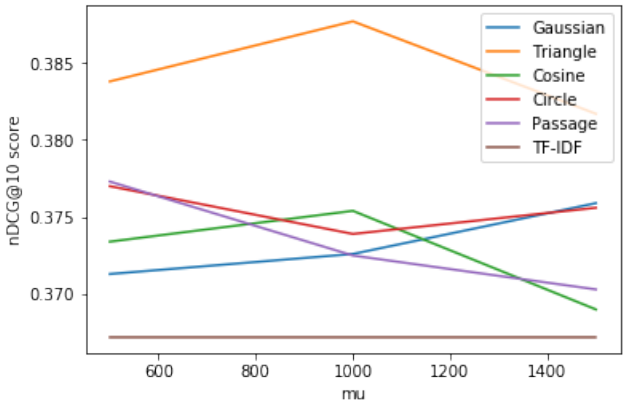

Figure 4: PLM scores upon validation set for different μ values and different kernel functions. The bottom straight line is the TF-IDF score.

This value is higher that the score of the TF-IDF ranking upon the validation set being at 0.3672.

Figure 4 depicts the results in a plot. Each line represents the scores of each kernel for different μ values, however the brown line is the score of TF-IDF which is stable to 0.3672. Clearly, all kernel functions outperformed TF-IDF.

Results on Lexical IR experiments

After optimizing the hyperparameters of each Lexical IR methods, we chose the best ones according to the description in the previous subsections. Table 5 sums up all the chosen parameter settings of each method.

After applying all the optimal settings, we receive the evaluation scores of our the best performing rankings upon the test set. These scores include the following: *nDCG@10, MAP@1000, Precision@5 and Recall@1000.* The obtained scores are summarized in Table 6. The bold numbers indicate the best scoring within each of the metrics.

We clearly observe that TF-IDF is achieving the best outcomes among all the other method, except the MAP@1000 and Recall@1000 scorings which get the highest value with BM25.

Between Jelinek-Mercer, Dirichlet Prior and Absolute Discounting, Dirichlet Prior Smoothing is the best performing Language Model, since it captures in a more compact trade-off the background probability and document related frequencies.

Table 5: Chosen hyperparameter settings for each model.

Lexical IR method	Parameter Setting
TF-IDF	–
BM25	• b = 0.75 • k_1 = 1.2
Jelinek-Mercer LM	• λ = 0.1
Dirichlet Prior LM	• μ = 500
Absolute Discounting LM	• δ = 0.9
PLM	• μ = 1000 • Kernel: Triangle

PLMs with Triangle kernel and μ = 1000 seem to perform worse than TF-IDF upon the test set. This hinters a possible overfitting upon the validation set. Moreover, we expect that the results will differ significantly if we apply a reranking of all top-1000 documents of TF-IDF.

After receiving the results, we performed statistical significance tests of the results using two-tailed paired Student t-tests. Two-tailed paired Student t-test is applied upon two populations with the following hypotheses:

- Null hypothesis H_0 : the means of the two populations are equal.
- Alternative hypothesis H_a: the means of the two populations are not equal.

For each possible pair of retrieval methods, we take their results which consist of 4 evaluation metrics. Four evaluation metrics means that four experiments were conducted, therefore we compare each metric upon the pair results. Having 4 evaluation metrics and 5 different retrieval methods, we conclude that we have to perform $(5!) \cdot 4 = 60$ statistical significance tests.

In order to tackle the multiple comparisons problem between the 4 conducted experiments, we apply the Sidak correction [8], which states that in each comparison we use:

$$\alpha_{\text{per comparison}} = 1 - (1 - \alpha)^{1/m}$$

,where m=4 (the number of experiments) and α=0.05. Thus,

$$\alpha_{\text{per comparison}} = 0.0127414551$$

Table 7 summarizes the outcomes of the significance test. It is a lower diagonal table whith each cell showing which of the tests were rejected. All means that all metrics rejected the null hypothesis whereas None means that there were no significant evidence to reject any of the test. If only some tests with specific evaluation metrics rejected the null hypothesis, we simply specify which ones were those.

Table 6: Scores for all the optimized Lexical IR methods.

	TF-IDF	BM25	Jelinek-Mercer	Dirichlet Prior	Absolute Discounting	PLM
nDCG@10	**0.4169**	0.4086	0.3489	0.4085	0.3860	0.3729
MAP@1000	0.2155	**0.2173**	0.1893	0.2104	0.2031	0.2037
Precision@5	**0.4317**	0.4133	0.3450	0.4200	0.3983	0.3867
Recall@1000	0.6510	**0.6524**	0.6203	0.6287	0.6262	0.6510

Table 7: Statistical significance tests of the results using two-tailed paired Student t-tests. Cells indicate which experiments rejected the null hypothesis of equal means.

TF-IDF	–					
BM25	None	–				
JM	All	All	–			
DP	Recall@1000	Recall@1000	nDCG@10, MAP@1000, Precision@5	–		
AD	MAP@1000, Recall@1000	MAP@100, Recall@1000	nDCG@10, MAP@1000, Precision@5	None	–	
PLM	nDCG@10, MAP@1000	nDCG@10, MAP@1000	MAP@1000, Recall@1000	nDCG@10	Recall@1000	–
	TF-IDF	**BM25**	**JM**	**DP**	**AD**	**PLM**

From those results we can observe that Jelinek-Mercer smoothing Language Model highly distinguishes itself from all the other methods except the PLM, as most of the tests were statistically significant. This is expected since Jelinek-Mercer performed badly in comparison to the others.

Even though TF-IDF performed better, the tests have shown that there was mostly no significant difference between its scores and the scores of other methods, except Jelinek-Mercer. However, we conclude that this is the best IR method in our experiments.

BM25 achieves a significantly better score at Recall@1000 in comparison to Jelinek-Mercer, Dirichlet Prior and Absolute Discounting. It also achieved the highest MAP@1000 score but is significantly different only from Jelinek-Mercer, Absolute Discounting and PLMs.

2.2 Task 2

In this task, we apply distributional semantics methods to score pairs of query-document according to their relevance. In general, semantic models try to represent the relevance of semantics between terms, while a lexical model, mostly inform us for the co-occurrences of the tokens in a pair. In this experiment, after we obtain the top 1000 ranking for each query using our previous lexical model the TF-IDF, and rerank them using our two latent semantics models, Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA). Both models return vectors in order to represent the embedding of the document and the query in each latent semantic space. The degree of relevance in a query-document pair is judged by their cosine similarity.

Both LSI and LDA were tuned with 64 number of topics. The evaluation of the two scoring methods was made again under the Trec Evaluator. On the result table 6 we can observe the scoring for the latent semantic models LSI and LDA and also the lexical model - TF-IDF(from task 1), under the *nDCG@10, MAP@1000, Precision@5 and Recall@1000.* From the results in Table 8, we can observe that the LSI outperforms the LDA.

Table 8: Scores for LSI, LDA, TF-IDF.

	TF-IDF	LSI	LDA
nDCG@10	0.4169	0.2102	0.1709
MAP@1000	0.2155	0.1091	0.0989
Precision@5	0.4317	0.2183	0.1867
Recall@1000	0.6510	0.6510	0.6510

As a last step we need to perform significance testing in the class of semantic matching methods. The testing is similar to the one in task 1. One can review the table of significances which is included in the notebook. These tests suggest that there is no significance difference between LSI and LDA. TF-IDF still greatly outperforms these two models and therefore the tests between them are insignificant.

All in all, both LSI and LDA scored relatively low values and lower than TF-IDF. This is due to the dense occurrences of the some words in documents that gave advantage to the lexical methods.

2.3 Task 3

In this task we joined together two concepts of Artificial Intelligence, Natural Language processing and Information Retrieval. To achieve this, we have utilized word-embedding to give a semantic element to both the query and the documents. Word embedding is a topic that has been researched on many occasion in the scientific world. Combining word-embedding with information retrieval is not as popular but has been investigated nonetheless.

The word embedding process used is amongst the most popular in current research, namely word2vec [5]. This is based on the Skip-gram model with some improvements that enable for a more expressive and efficient model.

To represent queries and documents using the word embedding we have chosen the method proposed by Nalisnick et al [6]. This method has shown good results when compared to BM25 and LSA.

	Precision@5	Recall@1000	ndcg@10	map cut@1000
300 Embedding	40.33	57.67	38.55	16.45
400 Embedding	41.17	58	39.71	17.23
500 Embedding	42.17	58.28	39.75	17.38

Table 9: Results for the word embedding model using different embedding sizes

	TFIDF & QL	BM25 & QL	WE & QL	TFIDF & BM25	WE & TFIDF	WE & BM25	WE, BM25 and TFIDF
Precession@5	0.02	0.02	0.018	0.021	0.018	0.021	0.023

Table 10: Results for the Logistic Regression model with average results on a 10-fold cross validation

Since, we also developed these methods it would be an interesting exercise to see if we also outperform these methods with the proposed method.

An overview of the process is as follows; we first learn the word embedding from the dictionary, then we embed all documents by using equation (2). Finally given a query and a document we give a score by using equation (1). This is repeated for every query pair in our dataset.

$$DESM(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{q_i^t D_x}{||q_i|| ||D_x||} \tag{1}$$

, where

$$D_x = \frac{1}{|D|} \sum_{d_j \in D} \frac{d_j}{||d_j||} \tag{2}$$

where D_x is the centroid of the normalized document word vectors serving as a single embedding for the whole document. q_i is the query embedding and d_i is the document embedding.

We have evaluated this method by scoring all query and document pairs as we have done in previous tasks and running the Trec evaluator. We have also tested 3 different embedding sizes, 300, 400 and 500 with 10 epochs. The results are tabulated in Table 9. The best performing method was the 500 embedding size with the top 5 Precision of 42.17. Unfortunately, we also wanted to test different epochs and were the tipping point may be but we found this beyond the scope of the task and it would be computationally too expensive. Nonetheless, the evaluation metrics show that this method obtains higher results than our previously developed methods.

2.4 Task 4

In this task we have investigated ways to use machine learning approaches to rank queries. Learn to Rank algorithms are a subset of information retrieval which intersects with machine learning. This topic has been investigated on many occasions [1], [7]. There are different types of learn to rank algorithms mainly pointwise, pairwise and listwise LTR.

In this task we only look at pointwise approaches particularly the logistic regression model. We initially extract all query document pairs and their relevance from the test set provided. We split this set into 10 different sets using 10-fold cross validation. Therefore, we train the model on 9-sets using different features extracted from these document query pairs with the target being the relevance label. Once the model is trained we predict the remaining set. This is repeated 10 times and the resulting evaluation measure is averaged to obtain the average performance on all the folds. 10-fold cross validation avoids overfitting the data.

The features which we have tested are TFIDF, BM25, Word Embedding (WE) and Query Length (QL). The results obtained are tabulated in Table 10. Unfortunately, the results obtained are not up to par with any of the methods listed above.

Our main concern is that the test dataset is very small especially when applying 10-fold cross validation. Another issue is choice of LTR algorithm, pointwise LTR algorithms have a history of poor performance when it comes to ranking mainly due to the loss function used. Some future work on this task would be to obtain a larger dataset and look at other pointwise approaches that would be able to generalize better. It would also be fruitful to take the time to compare this to pairwise and listwise approaches.

3 CONCLUSION

Observations on the results of the Lexical IR methods show that TF-IDF is the best scoring method, alongside with BM25. There is no significance in the results between these two methods. These two methods provide an efficient weighting of terms, however there is no (high) probabilistic aspect in their implementations. Some queries such us query 121: 'Death from Cancer' have really low term frequency in the documents that they appear, however the background probability of them in the whole collection could affect the scores positively. For instance, Jelinek-Mercer model manages to achieve a nDCG@10 score of 0.0734 whereas TF-IDF and BM25 both get a score of 0. Probabilistic models can achieve better scores in these kind of queries if they are properly tuned through hyperparameter optimization. This shows a trade-off between collection and document based weights. Future work in Lexical IR methods would be to use wider ranges of hyperparameter value during the tuning in order to explore if there would be better results in favour of Language Models.

The latent semantic models are very expensive in terms of memory and computation. They also require lot of data to be trained sufficiently. LSI scored a bit better than the LDA. Due to the lack of time for extra tuning though, both scored relatively low. It would be useful in future to train these models under even bigger corpus, and experiment with even more number of topics.

The word embedding method achieves very high results which are comparable to the best performing methods we showcased in this paper. This has been done with relatively low epochs of learning, we believe that if we have the time to train our model

on more epochs and maybe a slightly higher dataset we would be able to surpass all the methods proposed with ease. Finally, the LTR proposed has not obtained favorable results. Nonetheless, the general method of LTR has obtained very high results in literature. Therefore, we understand that the exploration of these methods could indeed lead to near state-of-the-art results.

REFERENCES

[1] Hsinchun Chen. 1995. Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms. *Journal of the Association for Information Science and Technology*, 46, 3, 194–216.

[2] K Sparck Jones, Steve Walker, and Stephen E. Robertson. 2000. A probabilistic model of information retrieval: development and comparative experiments: part 2. *Information processing & management*, 36, 6, 809–840.

[3] Yuanhua Lv. 2010. PLMRetEval. Retrieved Jan. 30, 2018 from http://sifaka.cs.uiuc.edu/~ylv2/pub/plm/PLMRetEval.cpp.

[4] Yuanhua Lv and ChengXiang Zhai. 2009. Positional language models for information retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 299–306.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*. C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, (Eds.) Curran Associates, Inc., 3111–3119. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[6] Eric Nalisnick, Bhaskar Mitra, Nick Craswell, and Rich Caruana. 2016. Improving document ranking with dual word embeddings. In *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 83–84.

[7] Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '15). ACM, Santiago, Chile, 373–382. ISBN: 978-1-4503-3621-5. DOI: 10.1145/2766462.2767738. http://doi.acm.org/10.1145/2766462.2767738.

[8] Zbyněk Šidák. 1967. Rectangular confidence regions for the means of multivariate normal distributions. *Journal of the American Statistical Association*, 62, 318, 626–633.