Information Retrieval homework 2

Michael Mo
University of Amsterdam
michael.mo@student.uva.nl

Changxin Miao
University of Amsterdam
mcx2576@gmail.com

ABSTRACT

In this paper, we investigate different information retrieval models. We start from the lexical models and semantic IR models. Then we dive more into depth to explore the word embedding and the learning to rank models. After we perform the experiments and record the corresponding results, we draw conclusions for each model and present possible explanations for the model performances.

ACM Reference Format:
Michael Mo and Changxin Miao. 2018. Information Retrieval homework 2. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this assignment, we are going to explore and compare different information retrieval models. For task 1, we implemented some lexical IR models such as TF-IDF, BM25 and some which are based on language models. Then we proceed to latent semantic models like LSI and LDA in task 2. In task 3 we begin by using a word2vec model to create word embeddings. Those word embeddings can be interpretated as having some semantic meaning attached to them, and in this task we experiment with how we can use that to design a ranking algorithm. For task 4 we trained a logistic regression neural networks with TF-IDF and BM25 scores as input features with the relevance as the output feature by using pointwise learning to rank algorithm.

For each of the four tasks we have constructed the models, performed some optimization of parameters on a validation set with grid search, and assessed its performance with the offline evaluation metrics: precision, recall, NDCG and MAP. In the following sections of the report we go through that process for each of those tasks and give the results. Together with some background information, we then discuss the results, potential problems or further improvements.

2 LEXICAL IR MODEL

A lexical IR model is able to gives scores to document-query pairs, with the goal of scoring documents higher if they are relevant. In order to do this, certain statistics (i.e. frequency of query term in document) are often used. We investigate a simple model like TF-IDF, and also look at BM25 and language models.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.1 Model introduction

2.1.1 TF-IDF. The TF-IDF model uses term frequency (tf) and inverted document frequency (df). The idea behind TF-IDF is that we score a document higher, whenever query terms appear more frequent in it, and whenever it is the case that those query terms do not appear in many different documents (the query terms have higher discriminating power). This can easily be seen by the formula for tfidf:

$$\text{tf-idf}(t;d) = log(1 + tf(t;d))[log\frac{n}{df(t)}]$$

To score a document-query pair, the tf-idf(t,d) are simply summed up for each term t of the query. Also note that longer documents have an advantage over shorter documents, since the term frequency is expected to be higher then.

2.1.2 BM25. To overcome the disadvantage of longer documents have in TF-IDF, BM25 improves the TF-IDF model by normalizing term frequency with document length. Sometimes a document with long length could be a concat of a same short multiple times. In contrast, long documents contains more information, which increases its chance to provide desired answers. BM25 model can be represented in a mathematical form as below:

$$\sum_{uniquet \in q} \frac{(k_1+1)tf_{d,t}}{k_1((1-b)+b\times(\frac{l_d}{l_{avg}}))+tf_{d,t}} \times \frac{(k_3+1)tf_{q,t}}{k3\times tf_{q,t}} \times idf(t)$$

2.1.3 Language models. Generally, language models simulate the probability distribution over word sequences. Every word follows the Bernoulli distribution for a certain position in the document. Multiple words competing for one position leads to the multinomial distribution of words in a document. Fundamentally, three scoring methods could be adopted to score the document/query pairs.They are document likelihood, query likelihood and KL-divergence. The document likelihood model tries to rank the document based on its probability to generate the objective query, query likelihood model tries to generate the document and rank the document candidates based on their similarity to current document. KL-divergence evaluation combines both approach and attempts to minimize the discrepancy between two models.
Language model could incur a potential problem, that is, if a term never occurs in the document, its probability will be zero . Thus, for all queries containing that word, the score will be zero. In order to avoid this extreme issue, we introduce some smoothing techniques to improve the current model.

- **Jelinek-Mercer smoothing (JM)** This technique directly incorporates document probabilistic model and background language model, by interpolating between the probability of

generating the word based on the document and the background language model. The formula is given by:

$$p_\lambda(w \mid d) = \lambda \frac{tf(w;d)}{\mid d \mid} + (1-\lambda)p(w \mid C) \tag{1}$$

Choosing a high λ thus means giving more priority to the document, and a low lambda means giving more priority to the background language model.

- **Absolute discounting smoothing** This method decreases the probability of seen words by subtracting a constant from their count. The math model is presented as below:

$$p_\lambda(w \mid d) = \frac{max(tf(w;d)-\delta, 0)}{\mid d \mid} + \delta \frac{|d|_u}{|d|} p(w \mid C) \tag{2}$$

- **Dirichlet prior smoothing (DP)** It includes maximum a posteriori(MAP) estimation together with lagrange multipliers, resulting a multinomial language model with Dirichlet prior smoothing.

$$p(w \mid \hat{\theta}_d) = \frac{tf(w;d) + \mu p(w \mid C)}{\mid d \mid + \mu} \tag{3}$$

$$= \frac{|d|}{|d|+\mu} \frac{tf(w;d)}{|d|} + \frac{\mu}{|d|+\mu} p(w \mid C) \tag{4}$$

In the rewritten way we can see that it is again an interpolation between the query likelihood based on the document and the background language model. The difference of it with with JM, is that the coefficients are not constant, but rather depend on the document length. We see that the longer the document is, the more priority/confidence is given to the query likelihood based on the document.

- **Positional language model (PLM)** This model differs from others in a sense that the probability of generating a word changes in different parts of a model. It also corporates with IR heuristics such as proximity and passage retrieval. The score function of the PLM could be calculated as below:

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

where $p(w \mid D, i)$ is the positional language model, described in [1]. After creating a language model for each position, we choose a model based on the score of its best matching position.

$$S(Q, D) = max_{i \in \{1, \dots N\}} S(Q, D, i)$$

In order to calculate the exact score, we have to calculate a score for each position i in the document. This is thus quite computational heavy.

2.2 Experiments and results

For each of the models, we began by optimizing the hyperparameters on the validation set based on NDCG@10, with the results visible in table 1. The full results can be seen in table 9 in the appendix. Most of them have similar performance. For JM we realize that a lower lambda seems to give a better NDCG score. This means that to estimate the query likelihood for our validation set, assigning more weights to the background language model gives a better performance.

For each of the models, we obtain the optimized model and check how these models perform in comparison to each other. In Figure 1 from the appendix we see their performance based on some evaluation metrics.

As mentioned earlier, because of the large amount of computations required, we limit the PLM to only score the top 10 documents according to the TF-IDF for each query. Therefore, the metrics *map@1000* and *recall@1000* for PLM are not reliable for this experiment. The language models and BM25 perform quite similar and the TF-IDF seems to give a lower level of performance than the other models. However, we note that these are average scores over all queries in the test set. The actual difference is query dependent. A possible explanation for that is the document length. For example, if we look at the performance of some models for query 96 = [computer,aided,medical,diagnosis] from the test set, we get the results shown in table 2.

Referring to the previous paragraph, documents with a longer length have the advantage in the ranking in tfidf, while in BM25 this is not the case. To demonstrate when this happens, consider a short document d1 precisely containing information about "computer aided medical diagnosis", and a very long document d2 which contains many sections about computers and medical equipments, with a tiny proportion actually mentioning comoputer aided medical diagnosis. Then TF-IDF will simply rank d2 higher because of the high frequency of the query words, while BM25 will penalize d1 because of the length. In table 3 we have the average document lengths of the top 5 documents for this query, and this indeed confirms our possible explanation.

Now we also saw that JM and DP were very similar, in the sense that both of them interpolate between whether the language model is based on the document or on the corpus. Since $\lambda = 0.1$, JM01 has more confidence in the corpus. The document length in the top 5 for DP500, is rather quite high, which means it did not only look at the language model based on the corpus. For this query it thus seems to be the case that it is better if the query likelihood is based more on the document model. Since the interpolation coefficients in dp are depending on the document length, this could be an possible explanation to the question why DP seems to be a bit better than JM.

Some plots concerning how the parameters affect a language models are visible in Figure 2 from the appendix.

Those results are based on the test set. We see that compared with optimizing on the validation set, a different value for the hyperparameter gives a better performance. Therefore, we conclude that the choice does not affect the average performance of the language model.

2.3 Statistical test result

We have 6 IR models, and to check whether the models are significantly different from each other, we thus perform 15 paired student t-tests. Each t-test is based on NDCG, since NDCG focuses more on the "head" of the returned ranked list. We chose NDCG@10 since our plm only gives ranking lists of 10 documents. The results are summarized in table 4.

As can be seen from the table, we only have two non-significant results within the 15 tests (only two p-values are bigger than 0.05).

Model	Hyperparameter
Jelinek mercer	$\lambda = 0.1$
Dirichlet prior	$\mu = 500$
Absolute discounting	$\delta = 0.9$
PLM	kernel=cosine, $\mu = 500$

Table 1: Optimized values on validation set for hyper-parameters

	tfidf	bm25	jm01	dp500
ndcg@5 for query 96	0.000	0.1461	0.000	0.6548

Table 2: ndcg@5 of four models

	TF-IDF	BM25	JM01	DP500
avg doc length	449.8	286.4	167.6	312.2

Table 3: Average document length of top 5 documents for the query

	TF-IDF	BM25	JM01	DP500	AD09	PLMKCO500
TF-IDF	-	1.006*10-8	2.604*10-4	1.131*10-8	8.556*10-7	0.3370
BM25	-	-	4.496*10-7	0.9908	0.01036	2.490*10-8
JM01	-	-	-	2.536*10-8	9.697*10-4	9.254*10-4
DP500	-	-	-	-	0.04538	3.181*10-8
AD09	-	-	-	-	-	2.644*10-6
PLMKCO500	-	-	-	-	-	-

Table 4: p-values from each t-test of a model pair

Hence, we could conclude that all models perform significantly different, with the exception of the model pairs (TF-IDF,PLMKCO500) and (BM25,DP500). The reason the t-test does not give a significant result for (TF-IDF,PLMKCO500) could be that the plm only scores the top10 results given by tfidf.

3 LATENT SEMANTIC MODELS (LSMS)

3.1 Model explanation

3.1.1 LSI. With the traditional IR model, we always confront with problems such as synonomy and polysemy. Therefore, establishing a model which could extract semantic meaning from the document might help with matching the query and document.
The model needs to different words to the same dimension in a reduced space. As a result, we will expect words with similar meaning are represented by vectors with similar value after SVD transformation.
At last, we calculate the cosine similarity between the document and query vectors generated from the LSI model.

3.1.2 LDA. LDA solves some potential problem of LSI from a probabilistic approach. Starting from PLSI, which assumes the documents generate the latent variable topic(z). The latent variable(z) also generates words with different probability. LDA improves this model in a sense that it defines the proportion parameter α and topic parameter β.

3.2 Results

At first we construct the LSI and LDA model with a fixed number of topics using *"gensim"*. Then we implement the TF-IDF function created from the first part. Based on the top 1000 document list from the first part, we create corresponding document vectors and compare them with query vectors with the function similarity matrix. It again outputs the ranking of top 1000 files with highest score.
Then we used four evaluation metrics from trec evaluation, namely *"precision@5"*, *"ndcg@10"*, *"map@1000"* and *"recall@1000"* to evaluate the outcome from the LSI and LDA model. For each model, we have to tune the hyper-parameter "Number of topics" . We used the same grid search strategy as in section 2 to find the optimal hyper-parameters, which are 20 topics for LSI and 30 topics for LDA. For each evaluation measures, we report the average score. The final results are summarized in the table below:

3.3 Statistical test

In the class of LSM models, we also perform the paired t-test on the trec evaluation results to check whether results of LSI and LDA are significantly different from each other. The statistical results could be summarized in the table below: As it is indicated from the above table, none of the p-value is smaller than 0.05, meaning that the results of LSI and LDA are not significant different from the other. Also the test results for *recall@1000* are not significant. The reason is simple: Since we preselected the top 1000 according to TF-IDF

model name	number of topics	evaluation model	results
LSI	20	precision@5	0.1067
		ndcg@10	0.1109
		map@1000	0.0748
		recall@1000	0.6498
LDA	5	precision@5	0.1433
		ndcg@10	0.1332
		map@1000	0.0821
		recall@1000	0.6498

Table 5: Hyper-parameters tuning for LSM models

evaluation model	average mean difference	p-value
precision@5	-1.2056	0.2292
ndcg@10	-0.7743	0.4396
map@1000	-0.6004	0.4396
recall@1000	0.0	1.0

Table 6: t-test results

for both LSI and LDA, the set of the 1000 documents they give is the same.

3.4 Conclusion

According to the above statistics, it is clear that the results for LSM models are not as accurate as lexical models such as TF-IDF and BM25. Not to mention the document list was already selected by TF-IDF before implementing LSM models.

LSM models might be effective in certain circumstances such as a high level of synonyms or polysemy. Otherwise it could not outperform the traditional term-matching IR models.

4 WORD EMBEDDINGS FOR RANKING

4.1 Model explanation

To create word embeddings (vectors) of the words in the vocabulary, we use a word2vec model. The word2vec model has been trained on the corpus for 5 iterations, and maps the words to a 300 dimensional vector. In the word2vec model, words with similar meaning are mapped to vectors lay close to the other in the vector space.

Then we perform the vector addition or subtraction for those transformed vectors. Even though the semantic meaning of word combinations might not be justifiable, during the operation of model, the resulting vector possesses the meaning of both vectors for the addition case and the deducted meaning of previous vector for the subtraction case. An example: The closest word embedding of the vector defined as w2v('father')+w2v('woman')-w2v('man') is w2v('mother') The idea to produce a ranking list using word2vec is to give scores to document-query pairs as follows: We first transform the words of the query $(w_1, \ldots, w_T)$ into their vector representations $v_1, \ldots, v_T$. We then define a new vector $v_q = v_1 + \ldots + v_T$. With this result, we aim to achieve the objective that the vector v_q contains the semantic meaning of the whole query. For a document $w_1, \ldots, w_N$ we could perform the similar transformation. Nonetheless, it is not reasonable to transform all words from the document to vectors and add the transformed vectors together to get a semantic meaning of the document. The reason is that a document can consist of multiple paragraphs, where the meaning of each paragraph can be totally different. We also expect the words to answer a query to lie in a single paragraph, or even in a single sentence of a document. Therefore, we first fix a number K (we choose K=12), and then for each position i in the document we want to find the semantic meaning of the K consecutive words of the document at that position i. We thus define for $i = 1, ..., (N-K)$ the vectors u_i as

$$u_i = v_{di} + v_{d(i+1)} + v_{d(i+2)} + \ldots + v_{d(i+K)}$$

Now the u_i can be interpreted as representing the meaning of the part of the document at position i. A simple method to score a document-query pair is to calculate the cosine similarity between v_q with each of the u_i, and then taking the the maximum of all scores. In other words:

$$S(q, d, i) = ...S(q, d) = max_i S(q, d, i)$$

4.2 Experiment process and conclusion

Since for a single document we need to calculate scores for each position, the computation burden is heavy. Therefore, we only consider the top 100 documents preselected by TF-IDF.

The interesting question now is to check whether making use of the semantic interpretation gives better results than the traditional lexical models.

The result of the model performance on the test set compared with TF-IDF is summarized in table 7.

We observe that there is a huge improvement for the current model. Since we preselect top 1000 documents with TF-IDF before we run the word2vec model, we conclude that using semantic meaning to rerank the top100 could enhance the model performance. Word2vec is able to score documents high even when TF-IDF gives a lower score is due to the fact that scores are now based on semantic representations of query and (parts of) documents. For instance,

	TF-IDF	word2vec
ndcg@10	0.2635	0.3943
P@10	0.2492	0.3758

Table 7: Average performance for queries in test set

if we consider a query containing the words "plane" and "fast", and a document concerning jets without mentioning those two words. As query implies semantic meaning close to "jets", the document could be expected to score high in the word2vec. In contrast, since the document does not mention the query words specifically, it will not get a high score in the TF-IDF model.

5 LEARNING TO RANK

5.1 Model explanation

In this task, we begin by training a logistic regression model to predict the relevance of a document. The inputs for the neural network consist of certain document features, and theoretically the output values are binary values 0 or 1. (0/1 for non-relevant and relevant respectively.) To have an IR model which creates a ranking, we want to be able to produce a score. The score stands for the probability of being in class 1. We have chosen for the two document features *"TF-IDF"* and *"BM25"*. These features are selected with the consideration of accuracy and time-efficiency.

5.2 Experiment process and conclusion

For a given query, we first preselect the top 1000 documents according to *"TF-IDF"*. For each of the documents we also calculate the *"BM25"*. Our trained neural network then assigns probabilities for being relevant, and the ranking list is created by setting documents with higher probability of being relevant to the top. Whether this learning-to-rank (LTR) model performs better than the IR models using only TF-IDF or BM25. The results and comparisons could be summarized in table 8. Since we have preselected the top 1000 according to TF-IDF, the recall@1000 of TF-IDF compared with LTR will be the same. In general, the LTR significantly outperformed *"TF-IDF"* , but compared with *"BM25"* the difference is trivial. To conclude, we can make the assumption that since the LTR's performance is much more similar to BM25, the model assigns more weight to the BM25 feature for prediction.

6 APPENDIX

REFERENCES

[1] 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.

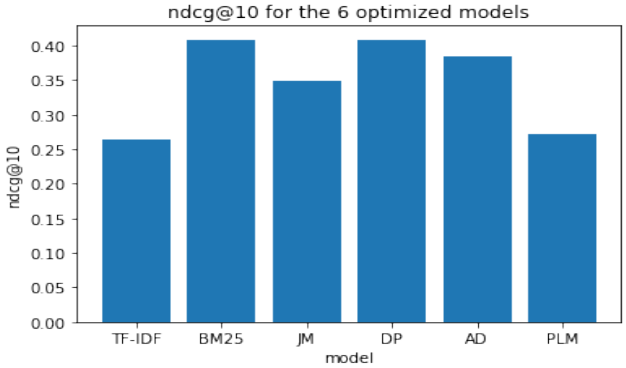
	ndcg@10	map@1000	precision@5
TF-IDF	0.2635	0.1352	0.2700
BM25	0.4086	0.2173	0.4133
LTR	0.4113	0.2135	0.4150

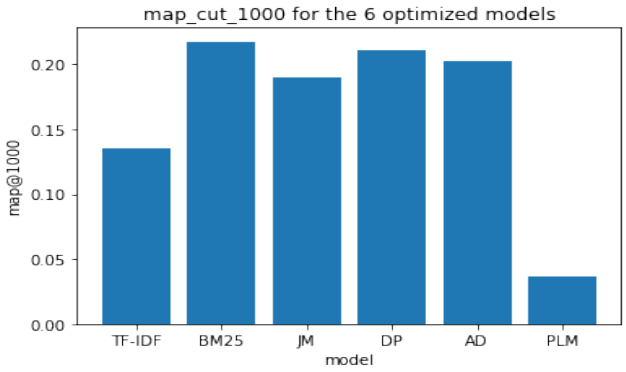
Table 8: Average performance for queries in test set

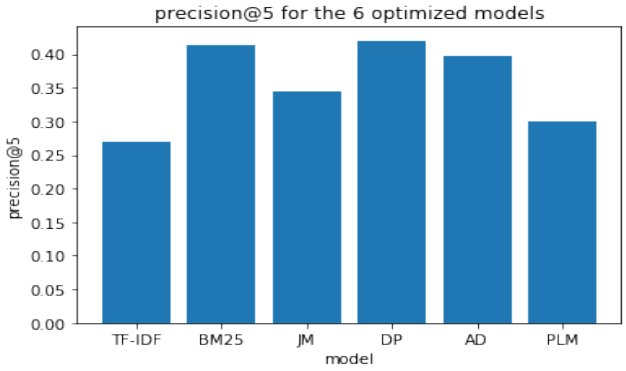
Model	Hyperparameter used	ndcg@10
JM	$\lambda = 0.1$	0.3991
	$\lambda = 0.5$	0.3823
	$\lambda = 0.9$	0.3676
DP	$\mu = 500$	0.4055
	$\mu = 1000$	0.4002
	$\mu = 1500$	0.4026
AD	$\delta = 0.1$	0.3614
	$\delta = 0.5$	0.3768
	$\delta = 0.9$	0.3949
PLM	kernel=Gaussian,$\mu = 500$	0.2754
	kernel=Triangle,$\mu = 500$	0.2782
	kernel=COsine,$\mu = 500$	0.2785
	kernel=CIrcle,$\mu = 500$	0.2767
	kernel=Passage,$\mu = 500$	0.2726
	kernel=Gaussian,$\mu = 1000$	0.2760
	kernel=Triangle,$\mu = 1000$	0.2782
	kernel=COsine,$\mu = 1000$	0.2785
	kernel=CIrcle,$\mu = 1000$	0.2767
	kernel=Passage,$\mu = 1000$	0.2726
	Kernel=Gaussian,$\mu = 1500$	0.2754
	Kernel=Triangle,$\mu = 1500$	0.2782
	Kernel=COsine,$\mu = 1500$	0.2785
	Kernel=CIrcle,$\mu = 1500$	0.2767
	Kernel=Passage,$\mu = 1500$	0.2726

Table 9: Average performance of model on validation set

Figure 1: Performance comparison of optimized models







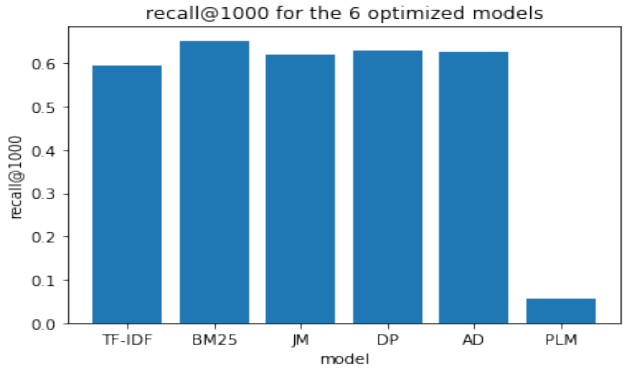


Figure 2: Influence of parameters on language models