Assignment 1: Retrieval models

Antonios Minas Krasakis
University of Amsterdam
UvA-id: 11849584
amkrasakis@gmail.com

Guido Jansen
University of Amsterdam
UvA-id: 5619680
guido.a.jansen@gmail.com

Taxiarchis Papakostas-Ioannidis
University of Amsterdam
UvA-id: 11407387
taxiarchis.papakostas@gmail.com

Keywords

Information Retrieval, tf-idf, BM25, Language models, LSI, LDA, word2vec, k-means, Point-wise ranking

1. INTRODUCTION

The aim of this assignment is to get familiar with offline information retrieval methods and algorithms as well as design an implementation of various methods. We will consider web search as our main retrieval task and therefore use $NDCG@10$ as our golden metric. All methods and models have been tested using *Trec Eval*. In section 1 we analyze the implementation and tuning of the most commonly used lexical vector space and probabilistic models and compare them. In the next section, we experiment with distributed representations (*LSI* & *Word2Vec*) and topic models to exploit semantic matching rather than simple word matching. Finally, we are combining all of the aforementioned methods in a pointwise learning-to-rank system and show that this system outperforms all of the above.

2. LEXICAL IR METHODS

For the assignment's first task, one vector space model and four probabilistic models are implemented, compared and evaluated. An in-depth analysis of the results can be found in the next sections.

2.1 Lexical Vector space models

For the vector space model, the score of a document for a specific query is the summation of all individual scores of query words have been seen inside a document. **Table 1** summarizes the evaluation scores of the *Trec Eval* package on the test set.

Table 1: Evaluation measures for VSMs (Test set)

Measures	TF-IDF
P@5	0.3700
Recall@1000	0.5352
NDCG@10	0.3549
MAP@1000	0.1592

2.1.1 TF-IDF

For the tf-idf model, the following equation was used:

$$tfidf(t;d) = \log(1 + tf(t;d)) \left[\log \frac{n}{df(t)} \right] \quad (1)$$

2.2 Probabilistic models

For scoring a document with a language model, the following query likelihood function [5][6] has been used:

$$f(q,d) = \sum_{i:c(q_i,D)>0} \left[\log \frac{p_s(q_i|D)}{a_D p(q_i|C)} \right] + m \log a_D \quad (2)$$

where $p_s(q_i|D)$ is the smoothed probability of observing a specific query word inside a specific document and $p(q_i|C)$ is the smoothed probability of seeing this query word inside the collection. m indicates the length of the query. The above probabilities had to be identified and extracted from the language models equations. It should be made clear here, that we ignored the last equation term from the original likelihood function ($\sum_{i=1}^{m} \log p(q_i|C)$) [5], since it is independent from the document and so it can be ignored for ranking. Also, we estimated the $p(w|C)$ probability assuming that all words inside the collection are contributing equally.

$$p(w|D) = \frac{tf(w;C)}{|C|} \quad (3)$$

A step by step analysis on how we constructed the final likelihood scoring functions using equation (3) can be found in the appropriate sections. **Table 3** presents the evaluation scores for the three probabilistic models choosing the optimized hyper-parameters from our analysis.

Table 2: Evaluation measures for PMs (Test set)

Measures	BM25	Jelinek M.	Dirichlet P.	Absolute D.
P@5	0.2417	0.3750	0.3817	0.3500
Recall@1000	0.4779	0.6264	0.6038	0.5811
NDCG@10	0.2392	0.3870	0.3732	0.3514
MAP@1000	0.1115	0.1980	0.1977	0.1656

2.2.1 BM25

$$BM25 = \sum_{\substack{unique \\ t \in q}} \frac{(k_1+1)tf_{d,t}}{k_1((1-b)+b(l_d/l_avg))+tf_{d,t}} \frac{(k_3+1)tf_{q,t}}{k3+tf_{q,t}} idf(df_t)$$

An assumption was made regarding the hyper-parameter k_3 setting it to zero ($k_3 = 0$), since it only makes an impact for large queries. Also, log computations have been used inside the equation to avoid under-flows and the run_retrieval function was modified to sum over only unique query terms, rather than all of the terms. Therefore, the final equation for BM25 is the following:

$$BM25 = \sum_{\substack{unique \\ t \in q}} \log(\frac{(k_1+1)tf_{d,t}}{k_1((1-b)+b(l_d/l_avg))+tf_{d,t}} idf(df_t)) \tag{4}$$

2.2.2 *Jelinek-Mercer*

The first part of equation (4) describes the probability of seeing a query word in a document ($P_{seen}(w|d)$), while the second part describes the probability seeing the word inside the collection ($\alpha_d p(w|C)$).

$$P_\lambda(w|d) = (1-\lambda)\frac{tf(w;d)}{|d|} + \lambda \frac{tf(w;C)}{|C|} \tag{5}$$

So, from equation (3) we have:

$$f(q,d) = \sum_{i:c(q_i,D)>0} \left[\log \frac{(1-\lambda)\frac{tf(w;d)}{|d|}}{\lambda \frac{tf(w;C)}{|C|})} \right] + m \log \lambda \Rightarrow$$

$$f(q,d) = \sum_{i:c(q_i,D)>0} \left[\log(1 + \frac{1-\lambda}{\lambda} \frac{tf(w;d)}{|d|p(w|C)}) \right] \tag{6}$$

Since, the last part of the equation($m \log \lambda$) is constant for each query, it does not have an impact on the ranking and has been excluded from the final scoring function of Jelinek-Mercer (equation (5)).
We will perform hyper-parameter tuning in the validation set, for values of λ in range of $[0.1, 0.9]$ with a step of 0.1 (figure 4). Based on the $NDCG$@10 score, we selected 0.9 as a value for our λ hyper-parameter (trade-off between word appearance in a specific document versus appearances in collection). Because of the modified equation, in our case a higher λ illustrates lower weighting for the collection frequencies (in contrast to the lecture slides and following [6]).

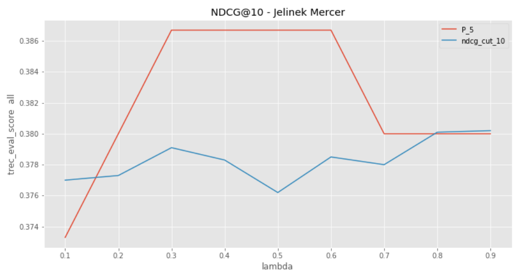

Figure 1: Hyper-parameter tuning (Jelinek Mercer)

2.2.3 *Dirichlet Prior*

$$p_\mu(w|\hat{\theta_d}) = \frac{|d|}{|d|+\mu}\frac{tf(w;d)}{|d|} + \frac{\mu}{m+|d|}p(w|C) \tag{7}$$

Following the same procedure as above, from equation (3) and (6) we have:

$$f(q,d) = \sum_{i:c(q_i,D)>0} \left[\log \frac{\frac{|d|}{|d|+\mu}\frac{tf(w;d)}{|d|}}{\frac{\mu p(w|C)}{m+|d|}} \right] + m \log \frac{\mu}{|d|+\mu} \Rightarrow$$

$$f(q,d) = \sum_{i:c(q_i,D)>0} \left[\log(1 + \frac{tf(w;d)}{\mu p(w|C)}) \right] + m \log \frac{\mu}{|d|+\mu} \tag{8}$$

The equation (7) has been used for scoring all the documents for the Dirichlet Prior probabilistic model.
Using the same method as in the previous section, we select the hyper-parameter $\mu = 1400$ (Figure 2). As in the Jelinek-Mercer case, here again a higher value of μ gives more weight to the collection.

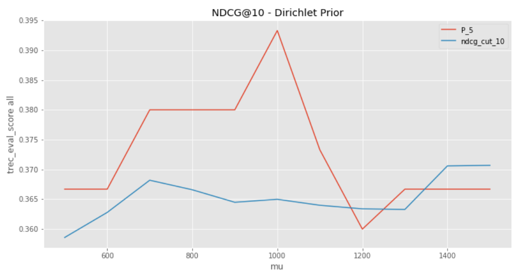

Figure 2: Dirichlet Prior NDCG@10 score- μ hyper-parameter

2.2.4 *Absolute discounting*

$$p_\delta(w|\hat{\theta_d}) = \frac{max(tf(w;d)-\delta,0)}{|d|} + \frac{\delta|d|_u}{|d|}p(w|C) \tag{9}$$

Following the same procedure as above, from equation (3) and (8) we have:

$$f(q,d) = \sum_{i:c(q_i,D)>0} \left[\log(\frac{\frac{max(tf(w;d)-\delta,0)}{|d|}}{\frac{\delta|d|_u p(w|C)}{|d|}}) \right] + m \log \frac{\delta|d|_u}{|d|} \Rightarrow$$

$$f(q,d) = \sum_{i:c(q_i,D)>0} \left[\log(\frac{max(tf(w;d)-\delta,0)}{\delta|d|_u p(w|C)}) \right] + m \log \frac{\delta|d|_u}{|d|} \tag{10}$$

Equation (9) has been used for scoring all the documents for the Absolute discounting probabilistic model.
A value of $\delta = 0.9$ was selected after tuning the hyper-parameter in the validation set (**Figure 3**).

2.3 Significance testing & analysis

In order to evaluate whether the different methods are significantly different, we will use a *two-tailed paired Student's T-test*. Because of the fact that those 5 methods will result in 10 total comparisons/significance tests (the so-called multiple comparisons problem), we will apply the *Bonferroni correction* and drop our significance level to:

$$\alpha = \frac{0.05}{comparisons} = \frac{0.05}{10} = 0.005$$

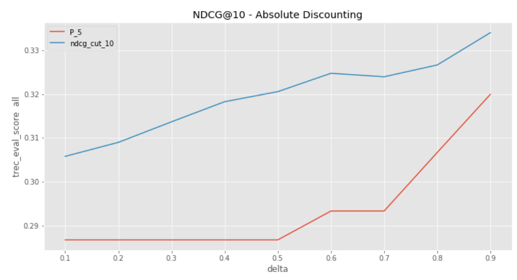

Figure 3: Absolute discounting NDCG@10 score- δ hyper-parameter

Table 3: Paired T-test p-values

	TF-IDF	BM25	Jelinek	Dirichlet	Absolute
TF-IDF	-	0.000	**0.112**	**0.386**	**0.854**
BM25		-	0.000	0.000	0.000
Jelinek			-	**0.224**	**0.038**
Dirichlet				-	**0.243**
Absolute					-

Therefore, the following pairs of ranking methods are considered significantly different:

- *TF-IDF* vs *BM25*
- *BM25* vs *Jelinek-Mercer*
- *BM25* vs *Dirichlet Prior*
- *BM25* vs *Absolute Discounting*

For the rest of the methods, we fail to reject the null hypothesis that they are significantly different at a significance level of 0.005. As a final remark, we can see that *BM25* is the only method that is significantly different versus all the others. This makes us wonder if there is a bug in the implementation of *BM25*, or if the hyper-parameters $(k1, k2, k3)$ were not appropriate (we did not perform hyper-parameter selection for that method).

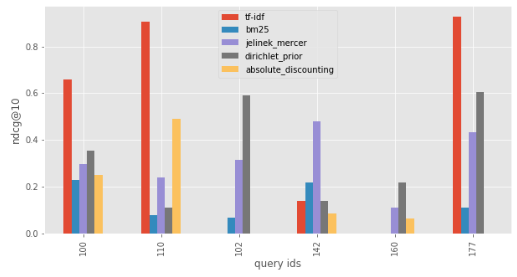

Figure 4: Individual queries comparing methods

The following paragraphs will focus on some individual queries that perform well on some methods and bad in others. By looking closer at these specific queries we aim to understand and explain the intuition behind the scoring & smoothing functions.

# 102: ”Laser Research Applicable to the U.S.’s Strategic Defense Initiative”

The above query performs bad in *TD-IDF*, as it cannot assign any relevant document in the top-10 list. The reason might be that the query can be paraphrased in many different ways, which is partly caused by the length of it. Another effect caused by the length is that the importance of the word laser is undermined. *BM25* also has a poor performance in this query but the language models seem to do a better job, with the exception of *Absolute Discounting*. A possible explanation for this is that *Absolute Discounting* ”blindly” subtracts a value (0.9 in our case) from the term document frequencies. For relevant documents that only mention ”laser” once or a few times throughout the text, this will probably cause their removal from the top-10 list.

- Query # 100: ”Controlling the Transfer of High Technology”
- Query # 110: ”Black Resistance Against the South African Government”
- Query # 177: ”English as the Official Language in U.S.”

For the queries mentioned above, tf-idf performs better than all other methods. What we find common in those queries is that they are well defined. They are to the point, containing words which are equally important for expressing the users ́ınformation need. It is our general observation, that when a user’s information need is expressed well in terms of the query (although it is not always possible), tf-idf performs really good.
We can also mention that it greatly outperforms BM25 which might be due to poor hyper-parameter selection and/or due to the fact that this collection does not have so many duplicate and similar documents (such as the web). Therefore, our main problem is not to select the best/more lengthy and informative document, but rather distinguish some of the relevant documents and rank them high.

- Query # 142: ’Impact of Government Regulated Grain Farming on International Relations’

In this query, *Jelinek-Mercer* performs much better than the rest, while other language models are failing to reach an equally good score. It might be that *Jelinek-Mercer* is performing well since it is the only smoothing method ignoring the $m \cdot loga_D$ in the query likelihood function, and thus does not normalize over document lengths.

- Query # 160: ’Vitamins - The Cure for or Cause of Human Ailments’

This can be considered a challenging query, as it is vaguely defined and the two most important entities involved (vitamins and ailments) are not weighted properly. The ”cure or cause” part of the query can be rephrased, but it is probably considered equally or more important than the vitamins term. Also, we expect the *TF-IDF* to give more weight to the word ”ailments”, but that might cause problems, since it is a very rare word and the language gap will lead to inefficient matching with possibly relevant documents.

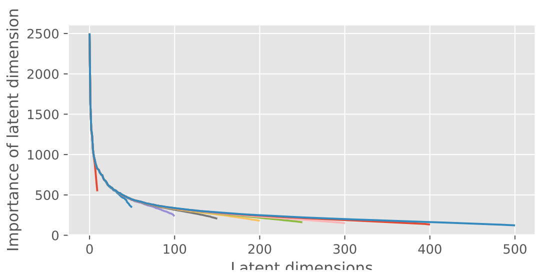

Figure 5: Exploring the effect of latent dimensions in singular values (LSI)

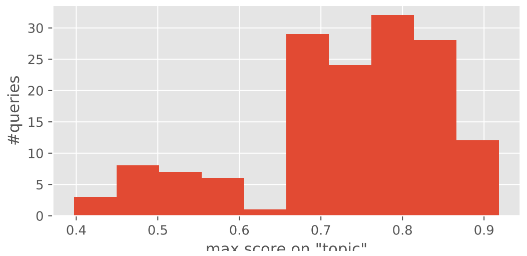

Figure 6: Histogram of the maximum topic distribution per query in the test set

3. LATENT SEMANTIC MODELS

3.1 Latent Semantic Indexing

Latent semantic indexing (*LSI*) performs a singular value decomposition on the Document-Term Frequency matrix (X). Another approach would be to perform the decomposition on the *TF-IDF* values of each document, but that would require us to calculate the inverted index for the whole vocabulary. Due to time and computational constraints, we did not try this. Following, we have to decide the number of latent dimensions used in order to re-calculate a new Document-Term Frequency matrix (X'). This procedure will hopefully remove the noise from the matrix (which partly addresses phenomena such as synonymy by bringing semantically related words closer in the latent space) and result into more accurate results. Hence, selection of the latent dimensions is the most important parameter of the *LSI* model. Manning et.al. [1] recommend using a value in the low hundreds for retrieval purposes. After trying out different values, we plotted their singular values (at the cutoff point) and evaluate them on the validation set (**Table 4**).
For the ranking part, we are treating queries as documents and projecting them to the latent space. Therefore, a document, or a query is treated as a bag of words, and each of it's word is represented by a vector of 300 values. The final representation of the query/document is the sum of the 300-dimensional vectors of its words. To alleviate the problem of shorter documents being closer to queries (which will naturally have shorter length), we are using the cosine similarity.

Table 4: Scores on LSI

	Validation set				Test set
Latent dimensions	100	200	300	500	300
Precision@5	0.166	0.153	0.146	0.180	0.155
NDCG@10	0.149	0.143	0.166	0.185	0.143
MAP@1000	0.100	0.108	0.116	0.121	0.081
Lowest singular value	244.83	181.98	150.8	123.25	

Although results on the validation set are slightly better on the 500 latent dimensions, we decided to proceed with 300, as our validation set consists of only 30 queries and our singular values indicate that an increase of an extra 200 dimensions (which results to almost a double sized vector for each word) does not decrease much the value of the last singular value.

Table 5: Scores on LDA

	Validation set			Test set
# topics	10	50	100	50
Precision@5	0.0600	0.1467	0.1200	0.1217
NDCG@10	0.0523	0.1478	0.1280	0.1296
MAP@1000	0.0795	0.1136	0.0707	0.0800

After running the evaluation on the test set, NDCG@10 reports a value of 14.39% and Precision@5 is at 15.5%.

3.2 Latent Dirichlet Allocation

In the context applied, *LAtent Dirichlet Allocation* (*LDA*) uses a different technique to transform the Document-Term Frequency matrix. Instead of performing matrix decomposition and deleting the least important details, it is learning a topic distribution over words and a word distribution over topics, which is learned from the data and controlled by two Dirichlet distributions. Since *LDA* addresses the structural analysis of corpora, it can be regarded a model for topic search, but it is rarely used (at least as a sole method) in query search [4]. Queries are treated as documents and therefore both are represented as a topic distribution defined by the words used in them. Again, the cosine similarity is used to score each query-document pair, as it considers two documents as identical if they have the same proportion in term frequencies regardless of the number of terms [4].
Due to the fact that *LDA* is very computationally expensive and rarely used for search as a standalone method, we only experimented with a few (and low) number of topics. The results can be found in table 5 (Recall @ 1000 was removed from the table, as it was only depending on the initial list of top-1000 documents derived from the *TF-IDF* rank). Based on these results, a number of 50 topics was selected. It is also interesting to see, that the distribution of queries is mostly skewed towards one of the 50 ”topics” rather than more uniformly distributed across many (Figure 6). This means that the algorithm maps the vast majority of the queries to one specific topic. Although this might help when combined with other methods, it is unlikely to work well by itself, especially when users are looking for specific things within a more vague/general topic.

3.3 Significance testing & analysis

Since those two methods are closely related, we expect a similar behavior - at least in the task of retrieval. Although the number of topics is much different than one another, the *paired T-Test* reported a p-value of 0.5625, hence failing to reject the null hypothesis that those two methods are signif-

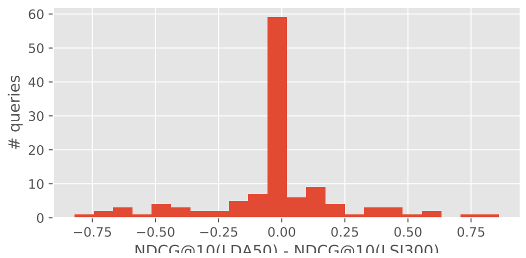

Figure 7: Histogram of score differences (NDCG@10) for different queries in the test set

# Q	Query	NDCG(LDA) - NDCG(LSI)
55	Insider Trading	0.8611
198	Gene Therapy and Its Benefits to Humankind	0.7538
189	Real Motives for Murder	0.6243
71	Border Incursions	0.6021
169	Satellite Launch Contracts	0.4815
77	Poaching	0.429
146	Negotiating an End to the Nicaraguan Civil War	0.4132
134	The Human Genome Project	0.3373
104	Catastrophic Health Insurance	0.3301

Table 6: Queries where LDA outperforms LSI

icantly different at a significance level of $\alpha = 0.05$.
In order to perform an analysis on their behavior, we are computing the differences of the $NDCG@10$ scores. We can see from the histogram at figure 7, that for around half of the queries the differences in the scores are trivial. However, there are certain queries for which one algorithm outperforms the other (Table 3.3 and 7). Our intuition behind those queries, is that *LDA* is performing well on more vague and general queries, such as "Insider Trading" or "Border incursions" and "Poaching". This is consistent with the nature of the *LDA* algorithm which is mainly used for classification and topic extraction and can also be partially explained by Figure 6. On the other hand, the *LSI* method works better on queries where the intuition is that the user is looking for something more specific (such as "Efforts to enact Gun Control Legislation", "Black Resistance Against the South African Government", "South African Sanctions" etc.).
However, we must highlight that those results might be amplified by the fact that the number of topics used in *LSI* is $x6$ bigger than the one in *LDA*. For instance, this might have had as a result to allow the *LSI* algorithm to devote one (or a combination) of its 300 latent dimensions to match South African matters, in comparison to *LDA* which probably could not fit documents regarding South African matters in one of its 50 dimensions.
However, this effect of differentiating the two algorithms is desirable, as it will allow Learning to Rank to utilize more meaningful features.

4. WORD EMBEDINGS FOR RANKING

This paragraph describes the methods used for creating word embedding for ranking documents for queries. The task suggest using *Word2Vec* [3] through *Gensim* implementation. What *Word2Vec* does is translate terms to so called *word vectors* and places them inside a *vector space*. The position of the word vectors will then represent semantic and contextual relations of each word vector by grouping them in close proximity. In other words, terms with a similar semantic meaning are placed closer to each other than term with a different meaning.

Table 7: Queries where LSI outperforms LDA

# Q	Query	NDCG(LDA) - NDCG(LSI)
183	Asbestos Related Lawsuits	-0.4775
188	Beachfront Erosion	-0.4778
82	Genetic Engineering	-0.4818
164	Generic Drugs - Illegal Activities by Manufacturers	-0.5837
52	South African Sanctions	-0.5984
110	Black Resistance Against the South African Government	-0.6063
174	Hazardous Waste Cleanup	-0.6431
70	Surrogate Motherhood	-0.7105
83	Measures to Protect the Atmosphere	-0.7191
156	Efforts to enact Gun Control Legislation	-0.8205

You can do this for single terms, but you can also combine several vectors to create *Sent2Vec* or *Doc2Vec* vectors. Whatever concept you use, the idea behind it is that you can now query texts based on semantic representations, instead of simple lexical matching. A word/query with a similar word vector as a document will now return as relevant, even if all terms in the query do not appear in the document at all. Once you have generated the word vectors you can calculate the difference by calculating the *Cosine Similarity* between two vectors:

$$\cos(\theta) = \frac{A \cdot B}{\|A\|_2 \|B\|_2}$$

4.1 Word vectors

The task suggest applying one of three (increasingly) difficult methods:

1. Average or sum the word vectors;
2. Cluster words in the document using k-means and use the centroid of the most important cluster;
3. Using the bag-of-word-embeddings representation.

The method for this task will be *Average or sum the word vectors*, but will include a method called *Word Movers Distance.*

4.1.1 Creating the word vectors

The first step is to create the word vectors. This can be done by training a *Word2Vec* model using *Gensim* or by using a pre-trained corpus such as the *Google News word embeddings.* We have chosen to train a new corpus based on the sentences of the available data. This results in a model containing a total of 267318 word embeddings of with

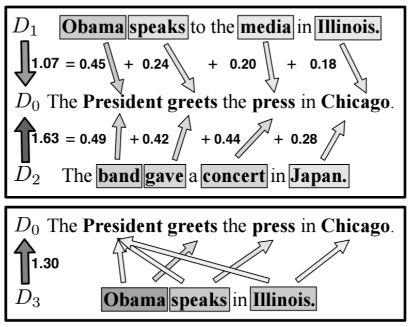
D_1 Obama speaks to the media in Illinois.
1.07 = 0.45 + 0.24 + 0.20 + 0.18
D_0 The President greets the press in Chicago.
1.63 = 0.49 + 0.42 + 0.44 + 0.28
D_2 The band gave a concert in Japan.
D_0 The President greets the press in Chicago.
1.30
D_3 Obama speaks in Illinois.

Figure 8: Word Movers Distance

N=300 dimensions. The vectors have been normalized for proper document comparison.

4.1.2 *Summed word vectors*

The first step was to sum the word vectors of each query and document. Afterwards these have been normalized once again for proper comparison.

Once the summed vectors have been created the *Cosine Similarity* can be calculated to rank documents for the queries. Evaluation of the ranking trough *Trec Eval* can be seen in **table 8**.

Table 8: Scores on summed vectors (Validation Set)

Latent dimensions	score
Precision@5	0.1667
Recall@1000	0.3729
NDCG@10	0.1446
MAP@1000	0.0886

4.1.3 *Averaged word vectors*

The second step was to sum the word vectors of each query and document. Afterwards these have been averaged and normalized for proper comparison.

Once the averaged vectors have been created the *Cosine Similarity* can be calculated to rank documents for the queries. Evaluation of the ranking trough *Trec Eval* can be seen in **table 9**.

Table 9: Scores on averaged vectors (Validation Set)

Latent dimensions	score
Precision@5	0.1400
Recall@1000	0.3736
NDCG@10	0.1462
MAP@1000	0.0886

4.1.4 *Word Movers Distance*

The *Word Movers Distance* [2] tries to optimize the ranking by turning the distance in a transportation problem. Essentially you minimize cumulative cost of replacing all words of one string, with all words of another string (**see figure 8**).

In this case the *Word Movers Distance* between query and text was calculated. This means that everyword in the text was replaced by one of the words of the query. The minimizes cumulative cost of replacing all the words was used to rank the documents for each query. Evaluation of the ranking trough *Trec Eval* can be seen in **table 10**.

Table 10: Scores on Word Movers Distance (Validation Set)

Latent dimensions	score
Precision@5	0.1867
Recall@1000	0.5079
NDCG@10	0.1859
MAP@1000	0.1215

5. LEARNING TO RANK

We use *Learning to Rank* to generate judgments on documents. For this assignment we use *Pointwise LTR* to create a logistic regression model. We use the *Trec Eval* test data as training and validation data. Afterwards the model will be used to score the top 1000 results of every query based on *TF-IDF*.

5.1 Creating the model

Before creating the model there has to be a proper representation of the data. This is done by looking at the query/document pairs. Each query/document pair has a score on all of the methods used in previous sections. This score can be used to predict the relevance score (0 or 1). We have taken the qrel test file to create a list of query/document pairs and used the following features to predict the relevance:

1. Query length
2. Document length
3. Unique words in document
4. TF-IDF score
5. BM25 score
6. Jelinek Mercer score
7. Dirichlet Prior score
8. Absolute discounting score
9. Summed vector cosine similarity
10. Averaged vector cosine similarity
11. Word Movers Distance
12. LSI
13. LDA

This resulted in a *accuracy* of 0.73889 while using a 80/20 split over the query/pairs data. The classification metrics can be see in **table 11**.

Table 11: Precision, Recall and F1

Relevance	Precision	Recall	F1
Non-relevant	0.75	0.98	0.85
Relevant	0.62	0.10	0.17
total	0.72	0.75	0.67

5.2 Ranking documents

For every query we have ranked the documents based on *TF-IDF*. This ranking was used as input for using the model for ranking. First a list of all query/documents pairs was created using the top 1000 documents. This was done for every query resulting in a list of N=145680 . Afterwards, features for every query/document pair were collected and used as input in the model to predict the relevance. Because we are interested in a ranking we did not use the logistic output. Instead we use the probability estimates of scoring a document as relevant. The higher the probability the higher a document will score for a specific query. Evaluation of the ranking trough *Trec Eval* can be seen in **table 12**.

Table 12: Scores on LTR vectors (full set)

Score type	Score
Precision@5	0.4800
Recall@1000	0.9883
NDCG@10	0.4660
MAP@1000	0.3994

6. CONCLUSIONS

To conclude, it is evident from our analysis, that all the traditional scoring methods perform better than *LSI*, *LDA* and the *Word2Vec* implementations we tried. Among them, *Jelinek-Mercer* performs best for $NDCG$@10 with a score of 38.7%, but the differences are rather trivial, especially given our small dataset and binary relevance judgements. Also, if we look at other measures, such as Precision @ 5, *Jelinek-Mercer* is beaten by *Dirichlet Prior*. This makes it even harder to distinguish a single method, since many different tasks (other than web-search) might have different more appropriate evaluation measures.
However, when all of the above are combined with the *Learning-to-Rank* (*LTR* method, it increases our best score from 38.7% to 46.6%. We should also highlight here that there are more appropriate *LTR* methods than the pointwise we just used for the purposes of this assignment and more feature engineering could be put into this. Another promising aspect of (pointwise) *LTR*, might be to use a model other than Logistic Regression which is a linear model. Among the classification algorithms, we believe that Support Vector Machines with a non-linear kernel might be a good candidate, although the computational power and the amount of data needed might be prohibitive.
However, time constraints did not allow us to investigate this matter further.

7. ADDITIONAL AUTHORS

8. REFERENCES

[1] D. M. Christopher, R. Prabhakar, and S. Hinrich. *Introduction to information retrieval*, volume 151. 2008.

[2] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[4] Y. Wang, I. Choi, and J. Lee. Indexing by latent dirichlet allocation and ensemble model. *CoRR*, abs/1309.3421, 2013.

[5] C. Zhai. Statistical language models for information retrieval. *Synthesis Lectures on Human Language Technologies*, 1(1):1–141, 2008.

[6] C. C. Zhai. Text retrieval and search engines.