

## Searching Structured and Unstructured Data: Retrieval Methods

JAN OSOLNIK, University of Amsterdam

GEORGE GOUSIOS, University of Amsterdam

SINAN ERSIN, University of Amsterdam

---

**Abstract.** This paper investigates which information retrieval methods perform better with the query-document pairs. There are different retrieval methods which tries to find the relevant documents to the queries. The queries and documents data is given by the University of Amsterdam: Information Studies department. Lexical IR methods, latent semantic models, word embeddings and learning to rank model are used in order to find the relevancy and are later scored. Analysis and discussion is made according to these scores and the result is presented.

**Keywords:** Lexical Methods, Latent Semantic Models, Word Embedding, Learning to Rank, LSI, LDA, Topic Modeling, Python Programming

---

### 1 INTRODUCTION

Information retrieval has gained importance with the evident rise of interest in data. While Google can be given as example for information retrieval, IR is also used in many sectors of government, research and industry. In today's world, retained data size can reach enormous dimensions and it is impossible to actually read every single document to have an idea about them. This is why, different researches have been made and various techniques are applied to gain information about any document. In this paper, eight different retrieval methods have been applied to 164598 documents and 150 queries. Term frequency – inverse document frequency (TF-IDF) is the first method that has been applied in order to find the most relevant documents to each query. Later, BM25, which can be seen as improved technique of TF-IDF is applied. This method is known as the first probabilistic method ever used in information retrieval. Although the results are not probabilistic, the idea of the technique is based on probabilistic techniques and inspired other probabilistic methods. Later the following three methods, Jelinek Mercer, Dirichlet Prior and Absolute Discounting are the language models which give the results as probabilities are used. Each of these three models are applied with three different parameters in order to decide on the most efficient one that gives more accurate results. After this, LSI and LDA models, which are latent semantic models are applied. These two methods are mostly used in Natural Language Processing. Moreover, Word Embedding is used in order to represent each words in the queries and documents as vectors. With this method, each query and document is reduced to one vector and similarities are calculated. Finally, Learning to Rank method is applied to the data in order to rank the documents according to their relevance to the queries. All of the results of these methods are tested and interpreted. In the following part, the experimental and computational details are described, later, how the results are implemented and the pipeline of the project is explained and finally, the scores and the performance of each method are discussed in Results and Discussion part.

## 2 EXPERIMENTAL AND COMPUTATIONAL DETAILS

The following section is split into 4 parts: Lexical Information Retrieval(IR) Methods, Latent Semantic Models, Word Embeddings and Learning to Rank Methods. Each query is matched with the relevant documents found by these methods and are tested later in order to decide on best models for the given queries and documents. Each method is then tested and the scores are retrieved.

### 2.1 Lexical IR Methods

In lexical IR methods, the meaning of the words are not taken into account and only the words themselves are used for that.

**2.1.1 *Tf-idf*.** This method is used to reflect the importance of each word in a document with a numerical statistic. The value increases proportionally with the appearance time of a word in a document, but decreases as the number of documents that contains the word increases. While most of the recommender systems use this method, many of the sophisticated methods take TF-IDF as the basis.

$$\sum_{\text{unique } t \in q} \log(1 + \text{tf}(t; d)) \left[ \log \frac{n}{\text{df}(t)} \right]$$

**2.1.2 *BM25*.** It is applied using bag of words and offers ranking of a set of documents based on the query terms appearing in each document. This method does not check the connection between the query terms.

$$\sum_{\text{unique } t \in q} \frac{(k_1 + 1)\text{tf}_{d,t}}{k_1((1 - b) + b \cdot (l_d/l_{avg})) + \text{tf}_{d,t}} \cdot \frac{(k_3 + 1)\text{tf}_{q,t}}{k_3 + \text{tf}_{q,t}} \cdot \text{idf}(\text{df}_t)$$

**2.1.3 *Language Models*.** A statistical language model is a probability distribution over sequences of words and are widely used in information retrieval. The model suggests that a document is a good match to a document if the document is more likely to generate the query. A document's probability to generate the query increases as it contains the query words more often.

Instead of modeling the probability of relevance of query to document, the model creates a probabilistic language model from each document and ranks the documents based on the probability of the model generating the query. Each language model works with the same algorithm, although they have different formulas. The following 3 models are implemented with 3 different hyper-parameters individually to find the optimum one and their formulas are shown below. After each calculation, their logarithms are calculated in order to avoid underflows.

#### 2.1.3.1 Jelinek Mercer.

$$\hat{P}_\lambda(w|d) = \lambda \frac{\text{tf}(w; d)}{|d|} + (1 - \lambda) \frac{\text{tf}(w; C)}{|C|}$$

### 2.1.3.2 Dirichlet Prior.

$$p_{\mu}(w|\hat{\theta}_d) = \frac{|d|}{|d| + \mu} \frac{\text{tf}(w; d)}{|d|} + \frac{\mu}{\mu + |d|} p(w|C)$$

### 2.1.3.3 Absolute Discounting.

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

## 2.2 Latent Semantic Models

**2.2.1 Latent Semantic Indexing (LSI).** LSI studies each word in documents and looks for the relation between words and creates topics with different words. The weakness this method has is it does not take into account that words can have different meanings.

**2.2.2 Latent Dirichlet Allocation (LDA).** While LDA also compares each words with others and looks for relationship, unlike LSI, it knows that same words can have different meanings, therefore it is common to see same words in different topics.

## 2.3 Word Embeddings for Ranking

The method is used in order to transform each word into vectors using mathematical embedding. With this method, similar words can be grouped together, and a query or document's one linguistic vector can be calculated. This method has roots in the 1960s and is known with its boost in NLP tasks.

Mapping each word into vectors are not in the scope of this project and the necessary codes that transform each word into vectors are given by University of Amsterdam (UvA). In this project, queries' and documents' representations are found with calculating the average of the word vectors.

**2.3.1 Average of the word vectors** – After transforming each word into vectors, the average of each queries' and documents' words' vector representations are taken and assigned to those queries and documents. Later, the cosine similarity is calculated between each query and document. Higher value implies greater similarity.

## 2.4 Learning to Rank Methods

Logistic regression is a statistical method used for analyzing a dataset with one or more independent variables that affect the outcome. In this method, the final output is either 0 or 1, although the function is continuous between 0-1. Two different models are used, which is with and without cross validation. The purpose of using cross validation is to train the model better without overfitting.

### 3 PROCESS OF WORK - PIPELINE

#### 3.1 Setup, Research and Exploration

After the initial setup of the project requisites, we devoted a few hours to get to know the tools we were using. We explored the several pyndri data structures that we were given and we put some thought in what ways we could make use of it in our initial approach towards the subject. Afterwards we did some research on the TREC Eval tool and particularly in its different evaluation metrics, and the cases each one of them is preferred over the others.

#### 3.2 Implementing the Scoring Functions

Our first touch with the python code, was to engage with the different scoring functions that were to be explored. The language models included were both Vector-Space Models (TFIDF, word2vec, Latent Semantic Indexing (LSI)), and probabilistic (Jelinek-Mercer, Absolute Discounting, BM25, Dirichlet prior, Latent Dirichlet Allocation (LDA)), so we had the opportunity to engage with different methods that either consider each document to be a vector, or a probability distribution..

We took advantage of the log function's "dampening" effect in our scores to keep values in a relatively low level. In the probabilistic models' implementations we applied smoothing to avoid the "probability" parts to get close to 0. That way we tried to avoid underflow problems that occur when many small portions (in our case, probabilities) get multiplied.

#### 3.3 Scores calculation

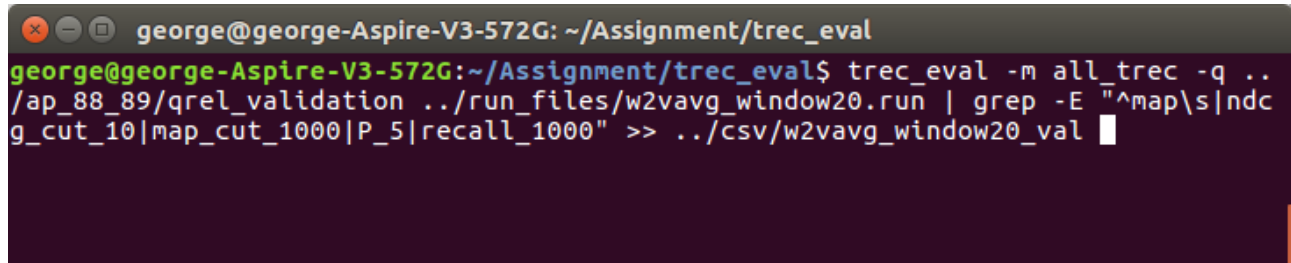
Considering we have access to iterate through 160k documents and 150 queries, each pair ranked by 13 different methods, it seems fairly obvious that unoptimized code and naive use of data structures can greatly affect performance of our code, and particularly speed. Since time was of the essence for this project, so we had to figure out what we could do in order to facilitate our research.

One of the ideas that was not only helpful, but saved a lot of time in practice was the `rel_doc` dictionary, which basically provided a link from every query to a set of the documents that are relevant to it. A similar data structure was given on a `(query_term, doc_id)` level (the `tfs` were provided in the `inverted_index`), but not on a `(query_id, doc_id)` level. There is no point in iterating through documents irrelevant to each respective query (only to get a score of 0 for all scores), and omitting irrelevant documents saves time during execution.

Moreover, possibly our most convenient implementation was that of the `scores_tuple`. It is a essentially a 2D python dictionary that includes all scores for each possible combination of any query to its relevant documents ( `[query_id, doc_id] → [score_tfidf, score_bm25, ...]` ). Once the scores for the population of `scores_tuple` were populated, we were able to proceed to the generation of the run files which led to getting some interesting insights about the data.

What is more is that the aforementioned procedure really helped us to arrange the data in such a way that was fit for the run file generation in the `run_retrieval` function. The run files were later fed to the Trec EVAL program along with the `q_rel` files to provide the evaluation results that we chose to extract to a range of

.csv files (which also facilitated the analysis procedure that followed, as they could easily be loaded to dataframes).



```

george@george-Aspire-V3-572G: ~/Assignment/trec_eval
george@george-Aspire-V3-572G:~/Assignment/trec_eval$ trec_eval -m all_trec -q ..
/ap_88_89/qrel_validation ../run_files/w2vavg_window20.run | grep -E "^map\s|ndc
g_cut_10|map_cut_1000|P_5|recall_1000" >> ../csv/w2vavg_window20_val

```

### 3.4 Extracting results – Generating Stats

It is worth noting that after getting all the scores and generating the run files, we had to evaluate the performance of certain functions using different hyperparameters. For example the Dirichlet model uses the  $\mu$  hyperparameter; the Absolute Discounting model uses  $\delta$  and Jelinek-Mercer uses  $\lambda$ . The selection was based on the MAP@1000 (mean average precision) scores of all variants that were closely scrutinized. T-tests were performed to examine the significance of the mean between all pairs's distributions. LSI and LDA models also have a wide range of hyperparameters to tinker with, but due to the lack of time, we had to limit ourselves to exploring just one hyperparameter from each one, which, in both was the number of topics. As for the Word2vec model, we investigated the impact of window size to the evaluation scores. It would be really interesting to observe the impact of the size (#dimensions) and the epochno as well, should we have more time to work on the project.

With the results of the Trec Eval software at hand, we were able to dive into all kinds of insights concerning the data, which will be explained in the “Results” section.

### 3.5 LTR Process

As far as the 4<sup>th</sup> task of this assignment in concerned, the feature set was built from the scores of all retrieval models and their chosen hyper-parameter values. These features were used to predict the target values - the relevance of a particular query-document pair. Target values were extracted from the qrel test set and matched based on the external identifier of every document. The model was then trained on the testing with a 10-fold cross-validation.

## 4 RESULTS AND DISCUSSION

### 4.1 T-tests on the validation set for the selection of hyper-parameters.

For Absolute discounting language model the null hypothesis is rejected that the sample mean that the MAP for hyper-parameter value  $\lambda = 0.1$  is the same as MAP for hyper-parameter value  $\lambda = 0.5$ . The value of  $\lambda = 0.5$  as a hyper-parameter was used later as it has the higher sample mean and the lowest variance.

	<b>models_compared</b>	<b>p-value</b>	<b>p_values_corr</b>	<b>h0_rejected</b>
<b>0</b>	abs01_abs09	0.040570	0.121709	False
<b>1</b>	abs01_abs05	0.102762	0.308285	False
<b>2</b>	abs05_abs09	0.119061	0.357182	False

For Dirichlet Prior language model all hypothesis are rejected based on the corrected p-values, the hyper-parameter chosen is  $\mu = 1500$  as it has the lowest sample variance.

	<b>models_compared</b>	<b>p-value</b>	<b>p_values_corr</b>	<b>h0_rejected</b>
<b>0</b>	d500_d1500	0.155709	0.467127	False
<b>1</b>	d1500_d1000	0.156332	0.468997	False
<b>2</b>	d500_d1000	0.160351	0.481052	False

For Jelenik-Mercer language model the hyper-parameter with the value of  $\delta = 0.9$  is chosen as it performs with the lowest variance even though we cannot reject none of the hypothesis after the Bonferroni correction.

	<b>models_compared</b>	<b>p-value</b>	<b>p_values_corr</b>	<b>h0_rejected</b>
<b>0</b>	jm05_jm01	0.268264	0.804791	False
<b>1</b>	jm01_jm09	0.270817	0.812451	False
<b>2</b>	jm05_jm09	0.296689	0.890068	False

For Word2vec the hyper-parameter the average window was experimented with where there was no statistically significant difference between using window\_size of 5 compared to window\_size of 20. The hyper-parameter with the higher number of window\_size is used as the mean MAP is higher (but not statistically so, the choice is arbitrary).

	<b>models_compared</b>	<b>p-value</b>	<b>h0_rejected</b>
<b>0</b>	w2vavg_window5_w2vavg_window20	0.738435	False

The null hypothesis is rejected that the MAP sample mean with 100 topics as a hyper-parameter in the LSI model is the same as the MAP sample mean with 20 topics. As the sample mean is two times higher the model with 100 topics is chosen.

	<b>models_compared</b>	<b>p-value</b>	<b>h0_rejected</b>
<b>0</b>	lsi_topics100_lsi_topics20	0.002207	True

#### 4.2 Report on Testing qrel

The hyper-parameters chosen based on the t-tests from the validation set are used on the testing set for the final report on performance. The evaluation on the testing set is performed for NDCG@10, MAP@1000, Precision@5 and Recall@1000 metrics. It is evident in the results that TF-IDF and BM25 outperform the other retrieval models. The worst performing model is LSI, followed by Absolute Discounting.

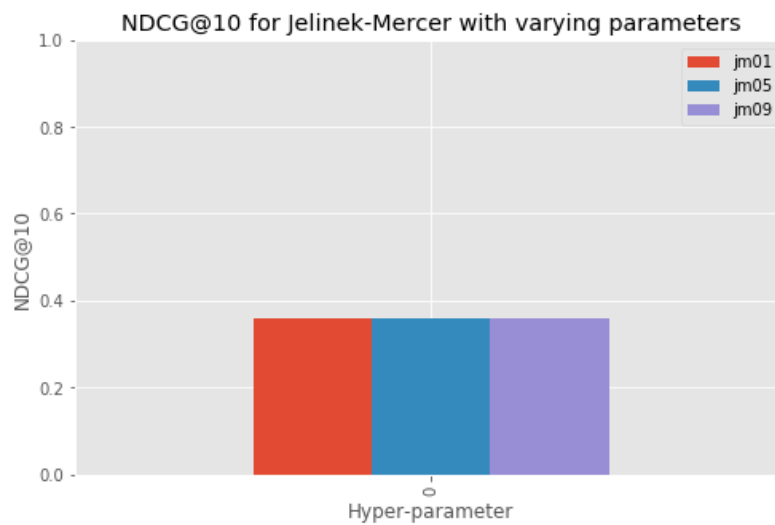
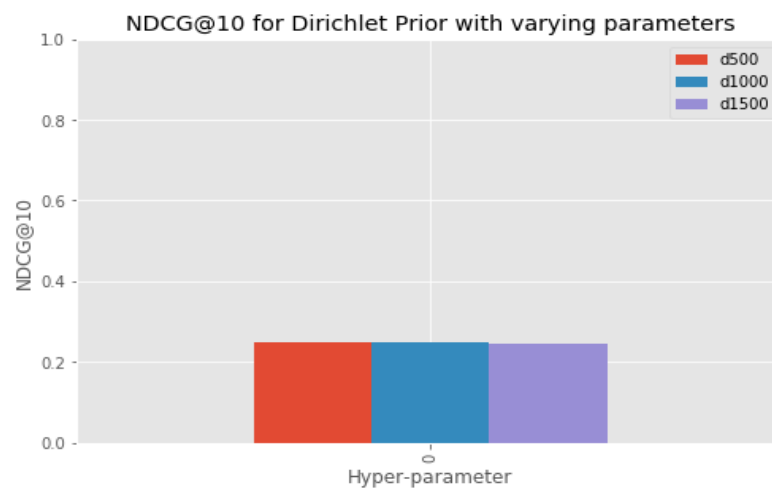
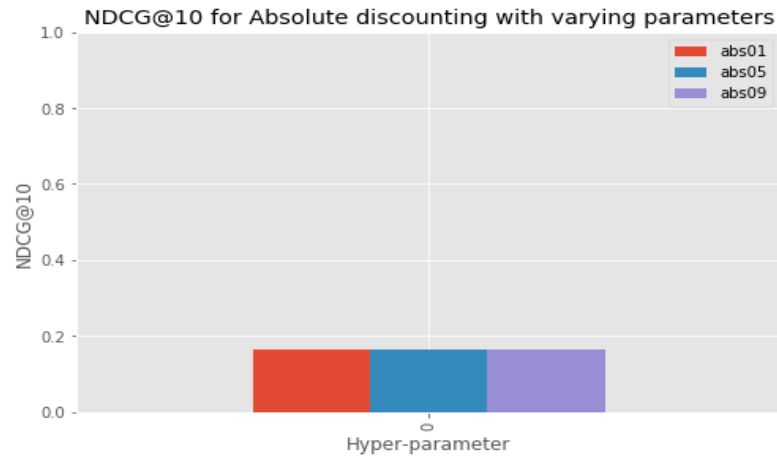
	<b>score</b>	<b>tfidf</b>	<b>bm25</b>	<b>d1500</b>	<b>abs05</b>	<b>jm09</b>	<b>lsi</b>	<b>w2vavg_window20</b>
<b>0</b>	ndcg_cut_10	0.4000	0.3907	0.2459	0.1636	0.3596	0.0321	0.2123
<b>1</b>	map	0.2114	0.2089	0.1125	0.0683	0.1812	0.0193	0.0939
<b>2</b>	P_5	0.4083	0.4017	0.2433	0.1667	0.3617	0.0317	0.2333
<b>3</b>	recall_1000	0.6432	0.6365	0.4581	0.3685	0.5871	0.2016	0.5108

#### 4.3 Visualization (NDCG@10)

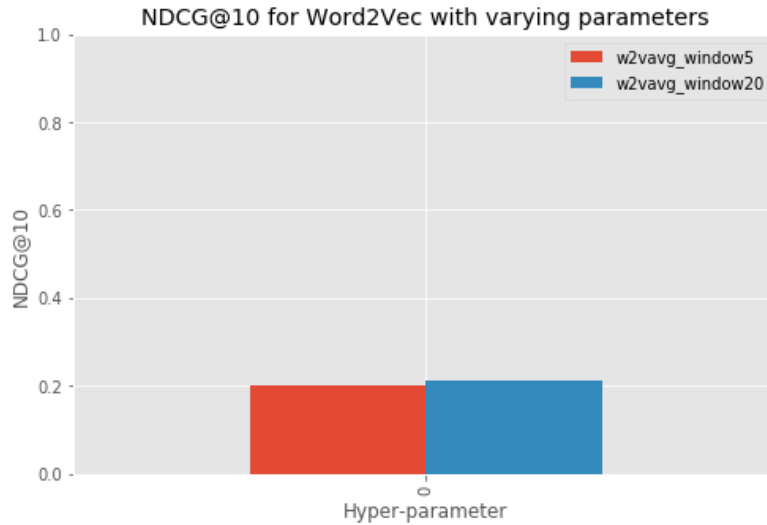
The NDCG@10 scores were calculated on each of the varying parameters for the language models, semantic models (LSI and LDA) and word embedding model.

For different hyper-parameter values in these models (mentioned in the Process of work) there was so apparent visible difference in the performance on NDCG@10 on the test set. This can be seen below in each plot corresponding to a different retrieval model.

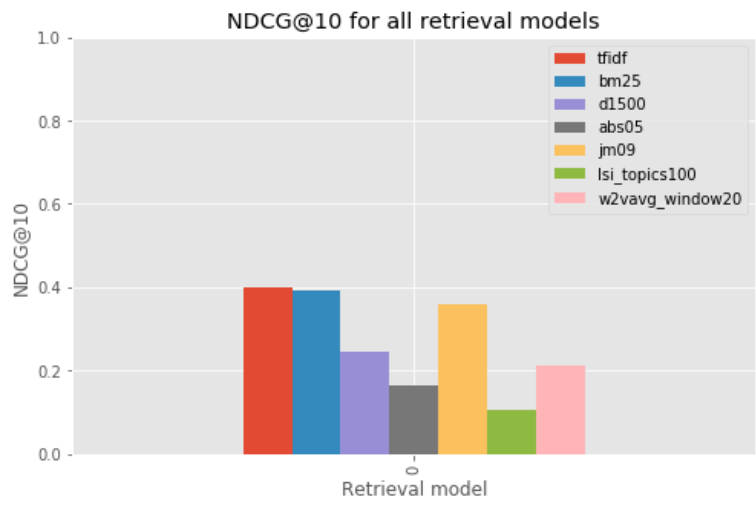
## Searching Unstructured and Structured Data







#### 4.4 Comparison of all retrieval models on NDCG@10



When comparing the models on the NDCG@10 it is apparent that TF-IDF and BM25 outperform other retrieval models. Among the language models the Jelinek-Mercer with hyper-parameter value 0.9 performs the best. In the comparison of semantic methods with word embeddings Word2Vec outperforms LSI.

#### 4.5 Learning to Rank

There was little predictive power extracted from the feature set, the final accuracy was 0.71 which is very similar to the overall class balance in the data. Besides that a model without cross-validation was trained without any difference in the final performance. The performance is close to equal across all folds of the

data, so little variance is present. It is evident that there is high bias in the model as it doesn't fit the data well to be able to predict the ranking.

## 5 CONCLUSION

In summary, we have performed both an experimental and theoretical study for the information retrieval project, but unfortunately no model can be said to have a predictive power. Although all the tests tried, they all failed to have a high test score on finding the relevant documents for the queries. It can be said that among our tests, TF-IDF performed the best.

Although it is seen that by increasing the number of topics in LSI, an evident improvement is seen, LSI still performed the worst. Also, It should be emphasized that we couldn't make LDA work because of the corpus issues, although we tried various methods like making it a list, or serializing it with the built-in function of Gensim. In addition, the test to decide on the hyperparameters didn't end up with an obvious success of any of the hyperparameters, they were very close. It should also be emphasized that there is no silver bullet model that outperforms all others.