A Study of Lexical and Semantic Language Models Applied to Information Retrieval

DANA KIANFAR, University of Amsterdam
JOSE GALLEGO-POSADA, University of Amsterdam

1. INTRODUCTION

In an typical Information Retrieval (IR) task, we are interested in finding a (usually ranked) list of results that satisfy the information need of a user expressed by means of a query. Many difficulties arise as the satisfaction and information need of the user cannot be directly observed, and thus the relevance of a particular document given the query is unobserved. Moreover, the query is merely a linguistic representation of the actual information need of the user and the gap between them can not be measured either. We explore three families of models that are used to measure and rank the relevance of a set of documents given a query: lexical models, semantic models, and machine learning-based re-ranking algorithms that build on top of the former models.

1.1. Data Exploration

We are provided with a dataset of 164,597 news documents from the Associated Press corpus, and a validation set of 30 queries with document-relevance labels (denoting 1 for relevant and 0 for non-relevant), and similarly a test set of 120 queries. There are around 100 unseen tokens in the queries, which are not found in any of the documents. There are also documents which are empty, and documents which do not have an `id`. We excluded such document from our experiments.

We note that the dataset is noisy as there are many spelling mistakes in the documents which are not corrected by the tokenizer. Therefore, differently spelled versions of the same word are represented as different tokens and have different term frequencies. Such examples can be easily identified by a high inverse-document frequency `idf` and a low Levenshtein distance to a dictionary word.

1.2. Pre-Processing

We define a function `normalize()` which performs `pyndri.escape()`, `index.tokenize()` and `pyndri.stem()` respectively to any given string. Input noise reduction is an expected consequence of this step. However, it is important to note that the `pyndri.stem()` function is a naive implementation of a stemming operator. Specifically in the case of named concepts, such as `Telesis`, we observed that `pyndri.stem()` provides `telesi` as an output. However, this stemmed token does not exist in any document or the index `token2id`. Despite this erroneous behavior, we believe that overall it improves our results as there are many plural words such as `predictions` which exist in `token2id` as its singular form `prediction`.

1.3. Assumptions

The distribution of word frequency in a collection generally follows a Zipfian distribution in which the most frequent words are often the so-called stop or functional words (determiners, pronouns, auxiliary verbs). These words can be identified in the given corpus by a token id equal to zero. In lexical models, where the models looks for co-occurence of words in queries and documents, stop words do not provide any useful information for distinguishing the relevance level of a particular document as they appear on every document. For that reason, we do not include stop words in our experiments.

2. EXPERIMENTS AND RESULTS

Despite our limited computational resources, we believe that any IR task faces many computational challenges on both memory and time. Usually for any given query, only a small fraction of the document-space is relevant. Moreover, most documents have a short length of around 400 tokens, and therefore have much smaller word-space compared to the entire collection. Thus it would be unreasonable, given any amount of computational power, to naively traverse the entire document and word spaces for each query. To optimize our models, we have made extensive use of this sparsity characteristic in our implementation of the algorithms. This required us to understand the models in depth, apply theoretical approximations for efficient computation, experiment with different parameters and make certain assumptions on our data.

2.1. Task 1

In Task 1, we focus our efforts on implementing and optimizing several lexical matching models: TF-IDF, BM25, Jelinek-Mercer, Absolute Discounting, Dirichlet Discounting and Positional Language Models. In the case a specific document does not contain any query word, we directly assign a very low score (usually $-\infty$) to it.

For the language models we decided to use a query likelihood approach, as it did not implied the overhead caused by the divergence computation of the KL divergence approach. However, the justification of the query likelihood as a valid measure is guaranteed by Bayesian reasoning, which in turn requires a prior over the documents. Such prior could be established using relevance information (e.g. click signals, time spent on a page, PageRank). Given that we do not have such information at our disposal, we assume a uniform prior over the documents. This can be seen as a monotonous transformation of the document scores, therefore the rankings remain unchanged.

Additionally, as stop words were not taken into consideration for any model, the collection size $|C|$ was reduced accordingly.

2.1.1. TF-IDF. Initially, we implemented this model calculating the normalized score provided by the cosine similarity between the query vector and the document vector of TF-IDF scores. However, we obtained better results by simply accumulating the TF-IDF score for each query term in the given document. This improved our model both in terms of computational complexity and the NDCG@10 score on the test set, and for this reason decided to proceed with this approach.

2.1.2. BM25. Given that the queries in the dataset were on average relatively short, we decided to exclude the normalization factor associated to the frequency of terms in the query. This is controlled by the parameter k_3, whose value was set to -1. As instructed in the assignment, we fixed $k_1 = 1.2$ and $b = 0.75$.

2.1.3. Jelinek Mercer. The results of the optimization of parameter λ on the validation set are shown in Figure 1. We observe the best value found for $\lambda^* = 0.4$. This implies that a slightly higher relevance (60%) is given to the background probability model estimated from the whole collection.

2.1.4. Absolute Discounting. The best value on the validation set obtained for the mixture parameter was $\delta^* = 0.8$, as displayed in Figure 2.

2.1.5. Dirichlet Discounting. Using the range proposed in the assignment for the optimization of μ we obtained the best value at $\mu = 2000$, the upper bound of the set. We decided to increase the range for the parameter search in order to explore whether better performance was obtained for larger values of μ. However, the results did not improve and the best value remains $\mu^* = 2000$, as can be seen in Figure 3.

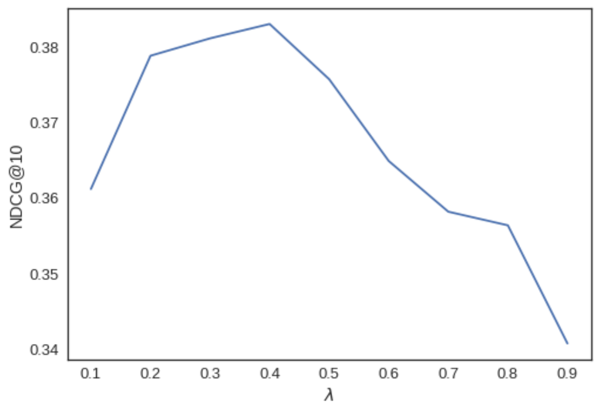

Fig. 1: Performance of Jelinek-Mercer on NDCG@10 for several values of λ

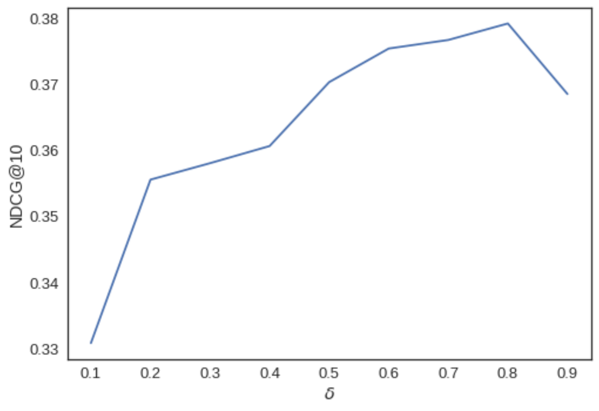

Fig. 2: Performance of Absolute Discounting on NDCG@10 for several values of δ

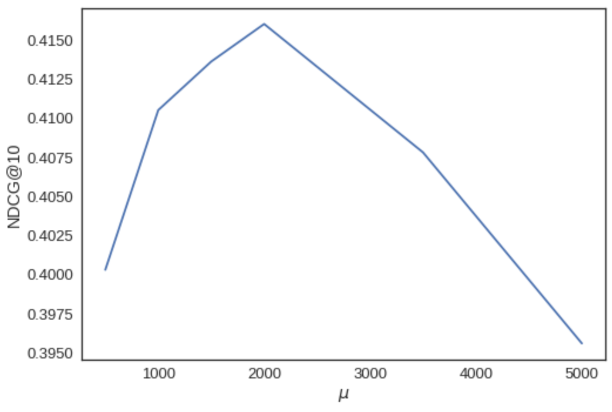

Fig. 3: Performance of Dirichlet Discounting on NDCG@10 for several values of μ

2.1.6. PLM. We performed several optimizations for the PLM model. Firstly, we used the same parameter $\mu^* = 2000$ obtained during the optimization of our Dirichlet smoothing model and set $\sigma = 50$ as instructed.

Secondly, we used the *Best Position Scoring Strategy*, where the PLM model searches for a position i such that $S(Q, D, i)$ is maximized. This is computationally expensive, as it implies estimating and evaluating the positional model at each position in the document. However, given the definition of the employed kernels and $S(Q, D, i)$, it can be seen that $S(Q, D, \cdot)(i)$ holds some kind of *continuity* in the sense that small changes in i generate small changes in S. Therefore, given that most of the kernel densities would be accumulated close to the positions in the document in which there is a query word, we restricted the possible values of i to be the positions of query words in the document. This is a fairly strong assumption, but can be relaxed to considering all i in a $\pm h$ neighborhood of the query word positions, where h may depend on the kernel width σ.

The computation of Z_i also implies traversing the whole document. However, as mentioned by the authors in the paper, this can be efficiently approximated using cumulative density functions for each kernel. We translated the C++ code created by the authors of the paper [Lv 2010], into Python for this assignment.

Additionally, PLM scores the documents according to the Kullback-Leiber divergence between the probability distribution estimated on the query and the smoothed positional language model. Assuming a uniform distribution for the words present in the query (i.e. disregarding repeated words in the query), the entropy computation can be simplified to:

$$\begin{aligned}
S(Q, D, i) &= -\sum_{w \in V} \mathbb{P}(w|Q) \log \frac{\mathbb{P}(w|Q)}{\mathbb{P}(w|D, i)} \\
&= -\sum_{w \in Q} \mathbb{P}(w|Q) \log \frac{\mathbb{P}(w|Q)}{\mathbb{P}(w|D, i)} \\
&= -\sum_{w \in Q} \mathbb{P}(w|Q) \left(\log \mathbb{P}(w|Q) - \log \mathbb{P}(w|D, i) \right) \\
&= -\sum_{w \in Q} \mathbb{P}(w|Q) \log \mathbb{P}(w|Q) + \sum_{w \in Q} \mathbb{P}(w|Q) \log \mathbb{P}(w|D, i) \\
&= -\sum_{w \in Q} \frac{1}{|Q|} \log \frac{1}{|Q|} + \sum_{w \in Q} \frac{1}{|Q|} \log \mathbb{P}(w|D, i) \\
&= \log |Q| + \frac{1}{|Q|} \sum_{w \in Q} \log \mathbb{P}(w|D, i)
\end{aligned}$$

The first term does not change across the documents given a particular query. On the other hand, the second term is penalization factor that affects the score of the document if the language model assigns very low probabilities to the query words.

Finally, we executed PLM on the validation set in order to choose the best kernel function. The results are displayed in Figure 4. As can be seen, the results among the different kernels are very similar and after running two-sided tests on the NDCG@10 for each of the kernel pairs, none of the differences was statistically significant with $\alpha = 0.05$. Therefore, we decided to keep the triangle kernel for the following experiments as it was the fastest to execute on average.

2.1.7. Model Comparison. The comparison between the different models was based on their performance on the test set. Each of the models was executed with their optimal

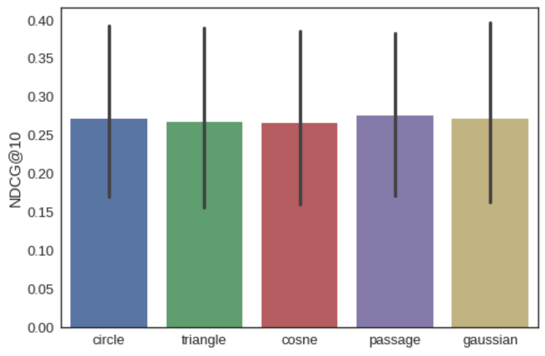

Fig. 4: Performance of PLM on NDCG@10 for several kernel functions (95% c.i.)

parameter(s) found during the optimization stage on the validation set, as mentioned in the previous sections.

In order to speed up the process, the PLM experiment was designed as a re-ranking task where we are given the query and a corresponding top 1000 ranking of the collection by TF-IDF. We used PLM to re-rank these 1000 documents. For that reason, the recall estimates, R@1000, coincide exactly for PLM and TF-IDF.

We clearly observed in Figures 5-8 that all lexical models, with the exception of PLM, exhibit a very similar performance across the different measures.

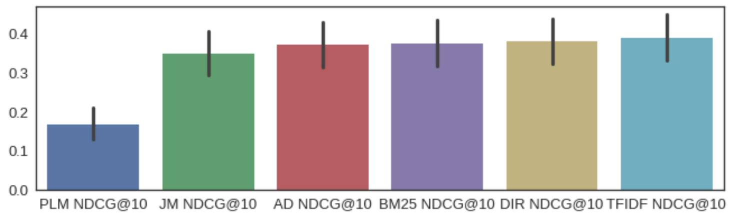

Fig. 5: NDCG@10 on the test set for the models with optimized parameters (95% c.i.)

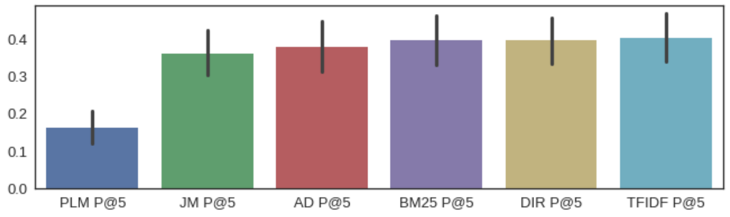

Fig. 6: P@5 on the test set for the models with optimized parameters (95% c.i.)

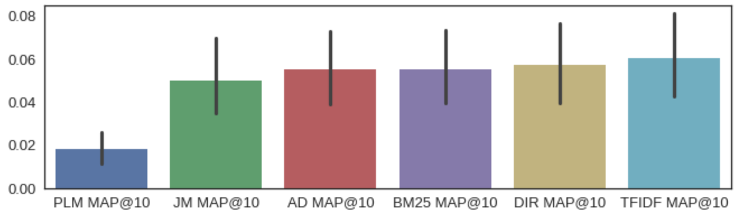

Fig. 7: MAP@10 on the test set for the models with optimized parameters (95% c.i.)

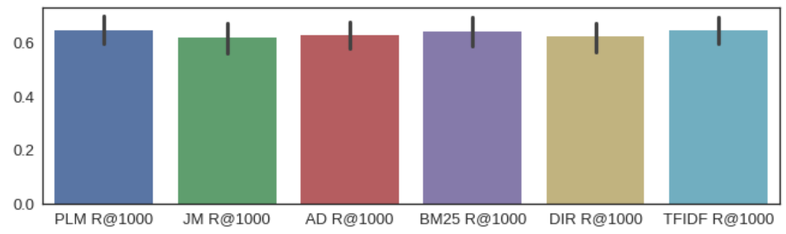

Fig. 8: R@1000 on the test set for the models with optimized parameters (95% c.i.)

Given that our analysis required the use of several metrics for each retrieval model, we employed a correction method on the significance level of the statistical tests to adequately handle the *mutiple comparisons problem* [Miller 1981]. We decided to use the Bonferroni correction method [Bonferroni 1936] instead of the Dunn-Šidák correction [Šidák 1967], as the latter assumes that the individual tests are independent while the former does not. We do not consider our measures to be independent as there is some degree of correlation. For example, rankings with high NDCG would also have high precision and recall.

	TF-IDF	BM25	JM	AD	DIR	PLM
TF-IDF	-	🟩	🟩	🟩	🟩	🟦
BM25	🟩	-	🟦	🟧	🟩	🟦
JM	🟩	🟥	-	🟥	🟥	🟦
AD	🟩	🟥	🟧	-	🟩	🟦
DIR	🟩	🟩	🟦	🟩	-	🟦
PLM	🟥	🟥	🟥	🟥	🟥	-

Table I: Results of one-sided statistical tests with Bonferroni correction on $\overline{\alpha} = 0.05$

The results of the one-sided statistical tests are presented in Table I. These tests were executed with a Bonferroni correction for 4 measurements on $\overline{\alpha} = 0.05$, which yields $\alpha = 0.0125$. The color of cell (i, j) represents the number of times the null hypothesis was rejected on favor of i, where red is 1, orange is 2 and blue is 3. Green is a tie in the case that there was not enough evidence to reject the null hypothesis in favor of i in any of the tests (cell (i, j)) but neither in favor of j (cell (j, i)). For example, the cell (DIR, JM) is orange, which means that the null hypothesis was rejected in favor of the Dirichlet Discounting for three of the tests. The exact value of the 144 p-values can be found in the source code.

These results confirm our observations obtained from the bar plots. We observe many ties between the different models, with a clear disadvantage for the PLM model and a relative disadvantage for the JM model. Note that the diagonal of the matrix is empty as it is redundant to compare a model against itself.

2.2. Task 2

In this part of the assignment, we used latent semantic models to assign a relevance score to document-query pairs. Semantic models represent the semantic and, to some extent, syntactic information by computing an embedding for both the document and the query in high-dimensional latent space. This is opposed to lexical models, which model relevance by a measure of co-occurence of tokens in a query-document pair. In the specific case of LDA, LSI, Word2Vec, and Doc2Vec, a semantic topics of a document is the latent random variable. For each query in our test, we obtain the top 1000 ranked documents from TF-IDF and re-rank them using the aforementioned models. Each model produces a vector representation of the embedding of the document in the latent semantic space for the document and the query, and the relevance score of the query-document pair is determined by their cosine similarity. The 1000 documents are then re-ranked according to their cosine similarity with the query.

Given the short length of the queries in the dataset and the fact that we have not performed any kind of query expansion, we expect to obtain a better performance on the semantic models compared to the lexical matching methods. Besides, in spite of the time and computational power constraints, we have made attempts to optimize the parameters of some of these models.

2.2.1. Latent Semantic Indexing. Using `gensim`, we trained a latent semantic indexing (LSI) model on our corpus. Our `IndriBow` class is responsible for a bag-of-words generator stream from Pyndri to meet the API specifications of `gensim.models.LsiModel`. LSI models latent semantic topics by using the SVD matrix factorization on the token-document occurrence matrix. The number of topics selected for a model dictate a low-rank approximation, ordered by the largest eigenvalue, of the original occurrence matrix. Each topic is represented as a weighted linear combination of several tokens.

For any given document or query, we can construct a vector representation of dimensions $\mathbb{R}^n$, where n is the number of concepts. This functionality is provided by `gensim`, where given a set of tokens (from a document or query), the model is able to estimate a probability distribution over the topics that are relevant to the query or document. Given that each trained LSI model has a fixed number of topics, we can represent a query or document as a vector where each entry holds the probability that the document concerns that topic. The similarity of a query-document pair is the measured by the cosine similarity of their vector representations. To project a document or query in latent space, a bag of words representation is passed to the `LsiModel` object. Using an LSI model trained with 100 topics trained on the entire collection, we obtained the test results shown in Figures 9-12.

2.2.2. Latent Dirichlet Allocation. The latent dirichlet allocation (LDA) model assumes a per-document distribution over topics, and a per-topic probability distribution over words. We used `gensim`'s parallelized `gensim.models.LdaMulticore` model for improved performance. Vector representations of documents and queries are constructed in the same way as for LSI. Using the same connector class `IndriBow`, we trained several models with 10, 20, 50, and 100 topics on the collection. We assumed a uniform prior for parameters η and α. Due to limitations of our computing resources, we were not able to perform cross-validation for parameter selection. Moreover, `gensim`'s inference operation for generating embeddings for bag-of-words repreesntations is an expensive operation that was a major bottleneck in measuring the performance of our models. Due to time constraints, we chose to train a model with 10 topics. The results on the test set are shown in Figures 9-12.

2.2.3. Word2Vec & Doc2Vec. Word2Vec and Doc2Vec compute word embeddings in a dense high-dimensional space. They are trained using a simple multi-layer perceptron with one hidden layer with linear activations, and an in-projection and out-projection layers. These projection layers are matrices with dimensionality $\mathbb{R}^{|\mathbb{V}|\times\mathbb{D}}$ and $\mathbb{R}^{|\mathbb{D}|\times|\mathbb{V}|}$ respectively, where $|V|$ is the size of the vocabulary and D is the dimensionality of the hidden layer. The training data is generated by a sliding window on the tokens which can produce either the inputs (continuous bag-of-words) or the targets (skip-gram). The window size C is a parameter that is anecdotally known to capture syntactic relationships between words when the C is small, and semantic relationships when C is large. The dimensionality of the hidden layer D is a parameter that defines the dimensionality of the word embedding vectors. Naturally, high dimensional representations needs large training sets and powerful computing resources. Note that the linear activation of the hidden layer allows efficient computation of the word vectors, as by removing the non-linearity, the most expensive part of the backpropagation updates is reduced to a linear operation. Both models are trained using stochastic gradient descent. A major bottleneck with the training times of these models is the multinomial softmax calculation at the output layer. The hierarchical softmax and negative subsampling optimizations provide computational relief when training these models. Once trained, the word embedding vectors are rows or columns of the in-projection and out-projection matrices.

For our IR task, to obtain a single vector representation of a document or a query, the vector for each of their tokens must be summarized in a single vector. This can be done by means of finding a centroid using a clustering technique such as k-means, however due to computing power and time constraints we simply averaged the word vectors in a document or query. In the case of Doc2Vec, the vector representation of the document is directly created by the model, while queries are constructed as the average of the words, as in Word2Vec. The relevance score of a query-document pair is measured by the cosine similarity of their vector representations.

Given our extensive previous experience with Word2Vec, we were certain that we could not train a model given our resources. While we have implemented both models, we use pre-trained word vectors on the Associated Press corpus and report test results using these models [Lau and Baldwin 2016]. The Word2Vec model is initialized with the skip-gram architecture and negative subsampling, and trained with $C = 15$ using $D = 300$ dimensional vectors on 4.3M words. Following a similar setup, the Doc2Vec model is trained on the same corpus using the distributed-bag-of-words architecture. The test results are shown in Figures 9-12.

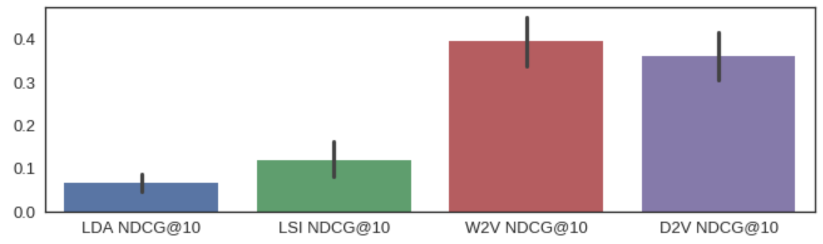

Fig. 9: NDCG@10 on the test set for the models with optimized parameters (95% c.i.)

2.2.4. Model Comparison. In Figures 9-12, we observe a clear trend that reflects our computing challenges with this task. Both LDA and LSI models perform poorly, which we believe is due to our limited training capacity. In contrast, the pre-trained Word2Vec and Doc2Vec models perform better on average. As comparing our LSI and

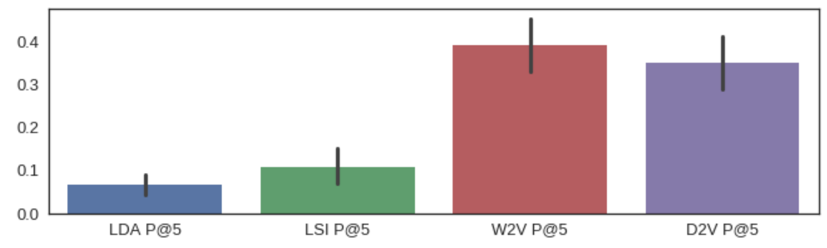

Fig. 10: P@5 on the test set for the models with optimized parameters (95% c.i.)

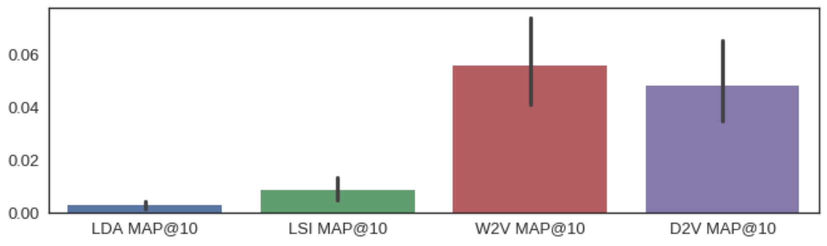

Fig. 11: MAP@10 on the test set for the models with optimized parameters (95% c.i.)

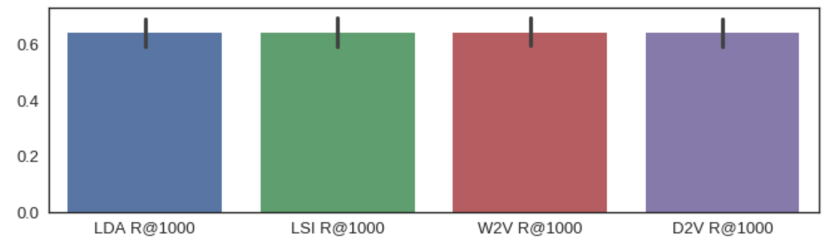

Fig. 12: R@1000 on the test set for the models with optimized parameters (95% c.i.)

LDA model will lead to the obvious conclusion that their performance is inferior to other methods, we will exclude them from our analysis.

We performed several one-sided sign binomial tests where we compared the performance of Word2Vec and Doc2Vec on all metrics excluding of Recall@1000. The reason for excluding recall is that as we are re-ranking the top 1000 results from TF-IDF, the recall is not an informative metric in task 2. With the Bonferroni correction, we reject p-values smaller than 0.0166. In this setting, all statistical tests in favor of Word2Vec provide enough evidence for rejection of the null hypothesis, and naturally there is never enough evidence to reject the null hypothesis in favor of Doc2Vec.

2.3. Query-Based Comparison of Lexical and Semantic Models

As the statistical tests by themselves do not offer detailed insights into how the different models perform internally, we explored specific queries where the models exhibited different performances and produce some hypothesis into why this occurs.

We searched in the test results to find queries where the models varied in their performance. By qualitatively inspecting such queries, we found two examples that offered promising properties. These two test queries are query 193 ”Toys R Dangerous” and query 116 ”Generic drug substitutions”. They provide good examples as the former contains a play-on-words to resemble the Toy manufacturer ”Toys-R-Us”. Without this formulation, we believe the query ”toys dangerous” is discriminative enough to model the information need of the user in a corpus of news articles. The latter query is vague and can provide lexical matching under different topics. Moreover, the word ”drug” can have two interpretations as a medicine or a psychoactive substance.

When evaluating the results for query 193 ”Toys R Dangerous”, we observe that among the lexical models, the PLM performs most poorly with an NDCG@10 of 0. All other lexical models have an NDCG@10 of 0.7 or greater, with the Dirichlet smoothing and BM25 having a perfect score of 1. By manually looking at the top 10 results in both models, we noticed an interesting trend. Using PLM, our model had retrieved many documents containing the token ”r”, most of which concerned US republican senators whose titles are often denoted as ”R. State Last_Name”, where name and state correspond to the name and representing state of the senator. The PLM model will regard a high frequency of any matching work to be highly relevant, especially if they are concentrated into dense regions within the document. This happens to be the case with the top results from PLM, as the news articles mention a sequence of names of Republican senators. In the case of BM25, as the model uses inverse-document frequency and document-length normalization, we hypothesize that it is more robust to such issues.

We also considered the same query between Word2Vec and Doc2Vec. While the former achieves an NDCG@10 score of 0.0784, the latter's score is 0.77. As the NDCG score for Word2Vec indicates, most of the top 10 documents were not labeled as relevant. In contrast, Doc2Vec's top 10 results were highly relevant, often concerning sales of toys or recall of a certain toy product due to some danger posed to children. We hypothesize that as Doc2Vec uses documents as its unit of operation, as opposed to Word2Vec which uses word tokens, the former is more suitable for IR tasks. This hypothesis, however, is not supported by the statistical tests described in Section 2.2.4.

When evaluating the results for the query 116 ”Generic drug substitutions”, with the exception of PLM, all lexical models achieved an NCDG@10 score of 0.18 or lower. The NDCG@10 score for PLM on this query was 0.46 and we hypothesize this is due to articles with dense occurences of the word drugs. The Word2Vec and Doc2Vec models achieved and NDCG@10 score of 0.47 and 0.5 respectively. We observed an interesting correlation in the results between the semantic models. All the top 10 retrieved documents were concerning medicinal drugs. However, the lexical models retrieved results concerning both medicinal drugs and substance drugs. We hypothesize that semantic models were able to capture the context of the query better than the lexical models.

2.4. Task 3

The goal of the final part of the assignment was to implement a Learning To Rank (LTR) model using the methods implemented in Task 1 and 2 as feature extractors for each query-document pair. To this end we use a Logistic Regression (LR) model trained with 10-fold cross-validation on the 120 test queries. This resulted in 10 LR models trained on non-overlapping portions of the dataset. Then, we create a set of *unseen* queries using the remaining section of the dataset that each fold did not use as training data. Finally, we measure the prediction accuracy on the unseen queries by assigning each of the unseen queries to the corresponding trained LR model.

Given a document-query pair (Q, D), a typical training example can be described as a tuple of a feature vector and a target value:

$([TF - IDF(D,Q)$, $\texttt{bm25}(D,Q)$, ..., $PLM(D,Q)$, $\texttt{lsi|lda}(D,Q)$, $\texttt{w2v|d2v}(D,Q)]$, $\texttt{relevant}(D,Q))$

Before training the models we performed feature pre-processing: first we replaced any $-\infty$ values assigned to the document-query scores with adequate values, then we employed several data normalization techniques, such as $[0-1]$, PCA and mean-deviation. However, using normalization no significant improvement on the training error was achieved.

After executing the 10-fold cross-validation we obtain the test set accuracy for each of the models. We perform the 10-fold split on the test queries, where we obtain q_{TR} and q_{TE} denoting the train and test split for each fold. For each q_{TR} and q_{TE} pair, we train a logistic regressor using `sklearn.linear_model.LogisticRegression` on all labeled document-query pairs for all queries in q_{TR}. We keep track of the unseen queries q_{TE} for each 10 models. To evaluate the performance of our LTR model on the test set, for each k-fold model we re-rank the top 100 results obtained from TFIDF on its unseen test queries q_{TR}. This results in a non-overlapping evaluation of all the models on the test set, where each model is presented with unseen queries according to its k-fold split. The results are shown in Table II.

Model	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Accuracy (%)	76.5	76.9	76.9	79.8	73.4	63.5	69.3	68.7	73.4	70.7

Table II: Test set accuracy for each LR model

Even though these results may look satisfying a priori, the off-line metrics for the generated ranking (see Table III) are worse than those generated by most of the individual methods. Note that in this case the recall is calculated at position 100 due to the re-ranking procedure.

NDCG@10	**P@5**	**MAP@10**	**R@10**
0.1165	0.1183	0.0053	0.3163

Table III: Off-line measures for the LTR list

There are several factors that could be responsible for this behavior. As it is customary in IR, given any query the training set is highly imbalanced towards negative samples (irrelevant documents). Specifically, the data is split 27% positive and 73% negative examples. In this setting, a model that generates *irrelevant* for every query has a the baseline accuracy of 73%. Additionally, the choice of a Logistic Regression model implies the assumption that there is a multidimensional hyperplane that separates the data between relevant and irrelevant document-query relations. However, given the inherent syntactical and semantical complexity of language, we consider it is highly unlikely that the data displays a linearly-separable structure.

On the other hand, the training algorithm is performing an optimization procedure to find the linear combination coefficients for the Logistic Regression that generates the best cross-entropy evaluation. Nevertheless, for the IR task per se, we are interested in a measure like NDCG or MAP and not directly in the cross-entropy or accuracy of the model. The problem relies on the fact that the gradient descent algorithm assumes a smooth, differentiable cost function to optimize, such as the one provided by a cross-entropy or squared loss functions, which is not the case for a NDCG measure. This justifies the choice to use the cross-entropy value as a proxy for the quality of a LTR algorithm, although, as shown in the results, they might not always coincide on their assesment of the system.

3. CONCLUSIONS

The results presented in this document lead to the conclusion that lexical and semantic models are effective tools for the proposed IR task. However, there is no model that significantly outperforms the others. Therefore, further exploration on Machine Learning methodologies that provide frameworks to combine the relevance signals provided by each of these models are of great interest, as they could boost the performance of the individual methods and increase the robustness of the system.

Query-based analysis of the results enhance the acquisition of insights on the mechanics inside of each method and the discovery of flaws or features of a particular scoring scheme. Additionally, query expansion techniques might help to alleviate those flaws.

Semantic models are extremely expensive in terms of memory and computation. For that reason, under conditions of scarce time of computational power, pretrained embeddings configure a very useful alternative. It might be relevant to fine-tune the models to accomodate to the particular characteristics of a given dataset.

The assumption of linear separability between relevant and irrelevant documents for a particular query seems unlikely to hold. Therefore, the usage of more complex models such as Neural Networks or Support Vector Machines, which are able to handle non-linear separation boundaries might yield better results.

Even with *curated* data, such as news reports, data exploration and analysis are a fundamental step, as they provide valuable insight on potential shortcomings or errors of a system in operation, and also provide useful considerations when designing data structures in an IR system. Moreover, improvements on the token normalization procedure and the usage of more advanced features like Named Entity Recognition would likely boost the performance of the retrieval system.

REFERENCES

Carlo E Bonferroni. 1936. *Teoria statistica delle classi e calcolo delle probabilita‘* (1 ed.). Seeber.

Jey Han Lau and Timothy Baldwin. 2016. An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation. *CoRR* abs/1607.05368 (2016). http://arxiv.org/abs/1607.05368

Yuanhua Lv. 2010. *PLMRetEval*. University of Illinois Urbana-Champaign. http://sifaka.cs.uiuc.edu/~ylv2/pub/plm/PLMRetEval.cpp

Rupert G Miller. 1981. *Simultaneous statistical inference* (1 ed.). Springer-Verlag.

Zbyněk Šidák. 1967. Rectangular Confidence Regions for the Means of Multivariate Normal Distributions. *J. Amer. Statist. Assoc.* 62, 318 (1967), 626–633. DOI:http://dx.doi.org/10.1080/01621459.1967.10482935