Information Retrieval I - Report

Davide Belli
University of Amsterdam

Gabriele Cesa
University of Amsterdam

Linda Petrini
University of Amsterdam

1 INTRODUCTION

The typical task in Information Retrieval consists in retrieving documents from a collection and ranking them with respect to a specific query. In this assignment we were required to implement and compare a number of different models to pursue that task. We used two different types of models. The first one is based on lexical matching and relies on finding the exact words of the query inside the document. The second one, instead, tries to take into consideration also the semantic meaning of the query. We used TREC Eval to evaluate the performance of a given model against a provided gold standard considering different quality measures.
The Repository used for this Homework can be found on GitHub[1].

2 IMPLEMENTATION

2.1 Dataset and Assumptions

The dataset we were provided contains 150 queries and 164597 documents, which are made from a total of 42208958 unique words. By default, Pyndri assigns the same token id to stop-words (words so frequent that they are uninformative). Throughout the assignment, we do not take them into consideration, for this reason a number of documents in the dataset have length 0 and they are handled separately when scoring.

2.2 Task 1 - Implement and compare lexical IR methods

The first Task in the assignment focuses on creating rankings of documents per query using Lexical Methods such as TF-IDF, BM25 and Language Models. Each method computes the score for a pair (query term, document). The values obtained over all terms in the query are then combined (summed up or multiplied, depending on the method used) to obtain a score for a query and document. To create a unique evaluating function (run_retrieval) and to avoid numeric overflows, we used sum of logscores instead of multiplication of scores for Language Models.

2.2.1 TF-IDF, BM25. The main thing to notice when implementing TF-IDF is how to handle the cases in which the document frequency is zero to avoid numeric exceptions dividing by zero. In this cases the TF-IDF value is manually set to zero. The same check is performed while computing BM25. BM25 was then implemented using suggested values for parameters k1 and b. k1=1.2 controls how quickly an increase in term-frequency results in tf saturation. b=0.75 represent the effect of the normalization on document length from 0 (not present) to 1 (fully normalized).

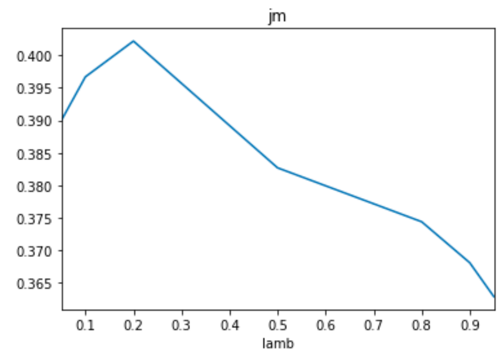
2.2.2 Jelinek-Mercer, Dirichlet Prior, Absolute discounting. One of the problems with TF-IDF and BM25 is that at a query term is assigned score 0 if its term frequency in the document is 0. The following models try to reduce this issue by means of smoothing. The idea is to take into consideration also the background probabilities which are obtained from the whole collection instead of from a single document. We chose to compute them as follows:

$$P(w|C) = \frac{1}{|C|} \sum_{D \in C} \frac{tf(w, D)}{|D|} \tag{1}$$

Hence the background probability for a word is the term frequency per document, averaged on the collection. There are other ways to compute it: one could simply take the overall word frequency (but this could be not very informative, for example if a word appears many time but only in one document, its background probability will be high even if the word is not particularly meaningful in the whole collection) or the normalized document frequency.

We are assuming the document language models follows a multinomial distribution, hence the document score with respect to the query is given by the product over words in the query of the word probability under the document's language model. To increase numerical stability, we compute log-probabilities instead and sum them.

Figure 1: nDCG@10 for different λ values - JM

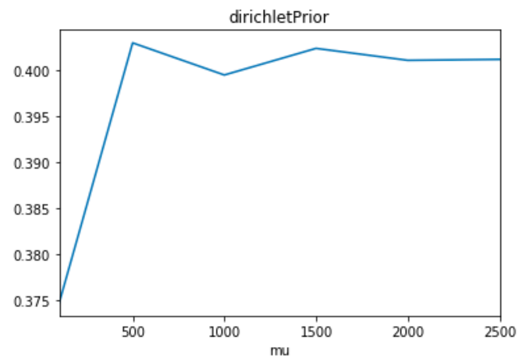


JM. The Jelinek-Mercer model acts as a linear interpolation between the word probability according to the document model and to the collection model. The parameter λ controls how much each term influences the overall probability. We tried different values in the range [0, 1], where 1 means we are only considering the document language model and 0 only the collection language model and we plotted nDCG@10 in Figure 1. $\lambda = 0.2$ gives the best results, meaning that the background probabilities have a strong influence on the model's scores.

Dirichlet Prior. This language model assumes a Dirichlet prior $Dir(\alpha_w)$ distribution on words, where the parameter depends on the word and it is proportional to the background probability of the word, weighted with the parameter μ. We tried different values for μ, in the range [100, 2500] but both nDCG@10 and MAP@1000

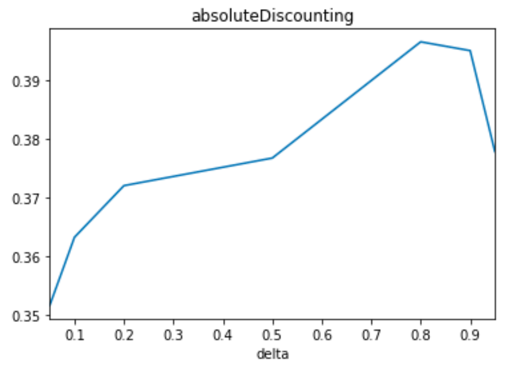
[1]https://github.com/davide-belli/IR_homework_2

Figure 2: nDCG for different μ values - Dirichlet Prior



do not seem to change significantly after the value 500, as shown in Figure 2.

Figure 3: nDCG for different δ values - Absolute Discounting



Absolute Discounting. Absolute Discounting subtracts a fixed value δ from the term frequencies of the words in a document and employs the residual probability mass to weight the background probability. In Figure 3, nDCG@10 is plotted for different values of δ: the best results are obtained for $\delta = 0.8$.

2.2.3 *Positional Language Model.*

The PLM is a Language Model that assigns a probability distribution over the position in the document. Each document is scored using the Best position strategy, which calculates the maximum score over all positions in a document. The score for each position in a document is given by the KL-divergence between the query distribution and the document distribution at that position. This model is computationally very expensive for a number of reasons: scoring a single document requires iterating over all positions in the document, the score at a certain position requires iterating over all words in the vocabulary and the probability for a single word requires iterating over all possible positions in a document. In order to reduce the computation time, we decided to use a number of optimizations:

- The score at position i for the pair document-query D-Q is:

$$S(Q, D, i) = -\sum_{w \in V} p(w|Q) \log \frac{p(w|Q)}{p(w|D, i)} \tag{2}$$

We decided to assume a uniform distribution on the query words, which yields:

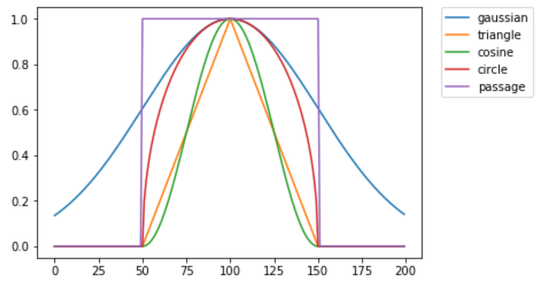
$$p(w|Q) = \begin{cases} \frac{1}{|Q|} & if\ w \in Q \\ 0 & else \end{cases} \tag{3}$$

Then, substituting the query distribution and leaving out constant terms, the score can be simplified as:

$$S(Q, D, i) = \frac{1}{|Q|} \sum_{w \in Q} \log p(w|D, i) \tag{4}$$

This reduces the number of operations since we are only calculating the probabilities $p(w|D, i)$ for the words in the query. A uniform distribution is a reasonable assumption for the query distribution, since queries are usually very short and it is hence hard to estimate a more complex probability distribution.

Figure 4: Kernels for PLM



- Referring to [3], we implemented a function to calculate the normalizing constant for $p(w|D, i)$. The idea is that, for a fixed i the sum $\sum_{k=1}^{N} k(i, j)$ can be approximated with an integral. Then if we normalize the variable of integration by subtracting the mean (which is the position i for which we are calculating the probability in this case) and dividing by the variance (σ in our case), we get the probability distribution associated with the considered kernel. Hence the normalizing constant is just the difference between the cumulative distribution function evaluated at the end and at the beginning of the document. Note that this procedure can't be applied for the passage kernel.
- To score a document, the Best position strategy requires to score all the positions in the document and then take the maximum one. Nevertheless, the PLM relies on the kernel functions to assign higher scores to query words and lower scores to words very far from them. For this reason, we are actually computing the score only for a neighborhood of the positions where query words appear in the document. The exact size of the neighborhood depends on the chosen kernel function.

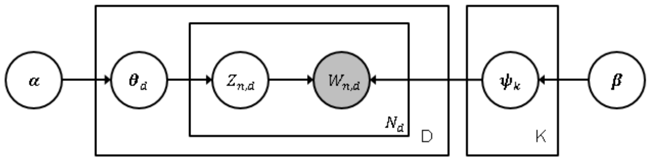
- It is worth noticing that many computations are query independent: indeed, we compute scores for words in different positions of a document; the scores relative to the words of a query are, then, combined to score the document with respect to that query. For this reason, we can save some computations scoring one document at the same time with respect to all the queries. Given a document, we score the set of words in all the queries for which this document can be relevant (we considered all the queries sharing at least one word with the document) and, then, for each query we combine its words.
- Position Language Models employ Dirichlet Prior smoothing when computing the probabilities of words. This means that it is necessary to try different μ values to choose the best one. However, running the whole algorithm for each of the μ values we want to validate would be too expensive. In order to solve this problem, we can run multiple "experiments" at the same time as most of the computations are independent from the value of μ. Indeed, when we compute the probability of a word w in a position i, we need to sum the kernel values over a set of indexes and only in the end to smooth the probability using μ. Therefore, we can first compute that summation (which is the most expensive part) and, then, smoothing the result with a set of different values of μ reducing the complexity of computing these scores from $O(|I| \times |M|)$ to $O(|I| + |M|)$ (where I is the set of indexes where we have to compute the kernels and M is the set of μ values to try).

2.3 Task 2 - Latent Semantic Models

In the second task, we were asked to experiment with Latent Semantic Models, in particular LSI and LDA. Both models are based on the idea of representing documents as a composition of topics. Topics can be described by a linear combination of words (LSI) or by latent variables which regulate the generation of words in documents (LDA). Then, every document is represented as a combination or probability distribution over different topics, depending on the words appearing in its text. The main difference between LSI and LDA is that the first is a vector space model using Singular Value Decomposition, while the latter is a probabilistic model, which introduces prior probabilities over topics and words generation (Fig. 5). This means that in LSI, documents are represented as vectors in the space of topics. On the other side, LDA representation of a document is a distribution of probabilities over the topics.

To implement both models in our homework we used classes from Gensim package. To train the model over our corpus, we first had to represent the corpus in a suitable data-structure. Precisely, the corpus is defined as a list of documents where each document is a list of unique words paired with a weight. Meaningful ways to initialize word weights in the document are using the presence/absence of the word in the document (1 or 0 values), the number of occurrences of the word, or the tf-idf value for that term and document. The results reported in our work are produced using the number of term occurrences as initialization value. Different initialization criteria produced similar results and are therefore not included in score tables because redundant.

Figure 5: Plate Notation of Latent Dirichlet Allocation model.



A crucial component in this experiment is the choice of a suitable number of topics (K value) in order to capture the range of all topics in the corpus without slowing down computations with high number of dimensions. Common choices for similar tasks employ K values ranging from 100 to 300 depending on the size of the corpus. We decided to test values [50, 100, 200, 300]. Other hyper-parameters in the models are alpha and beta, which are used to describe respectively the topic-per-document density and word-per-topic density. Because we couldn't make any assumption on how topics characterize documents in our corpus, we decided to leave these distributions symmetric (which is also the most likely case in commonly used corpora).

On the implementation side, we precomputed a dictionary from document ID to its LSI and LDA representation in order to avoid computing it for every query. To score query-document pairs, we first employed cosine similarity over vector representations. Because LDA representation is a probability distribution over topics, we also implemented KL-divergence as scoring measure for the probabilistic approach.

An interesting positive aspect of semantic models such as LSI and LDA is that it is easy to verify if a model is meaningful by simulating query over known documents. For example, running a query over a relevant document (defined in the validation set), results in a cosine similarity over 0.95, while executing the same query on other random documents mostly return scores around zero. Finally, we can see which topics are most informative in our corpus by looking for the highest singular vectors contained in model.projection.s in LSI, or computing top_topics in LDA.

2.4 Task 3 - Word embeddings for ranking

A common problem in IR is the so called *vocabulary gap*: the user might express his query with words that don't appear in a certain document, even if the semantic meaning corresponds. This is a problem common to many NLP tasks and one of the most used solutions is using *word embeddings*. Each word in the vocabulary is represented as a dense vector that tries to capture some semantic meaning of the word, so that words with similar meaning are associated to vectors close to each other in a high dimensional space (usually of size 100-200).

To exploit this information, we implemented a Generalized Language Model (GLM) as in [2] that extends the Jelinek-Mercer approach. Starting from a simple linear interpolation with the background language model (unlike the background we used in Task 1, here it is based on the distribution of words in the whole collection C), the probability for a word with respect to the document language model:

$$P(w|D) = \lambda \frac{tf(w,D)}{|D|} + (1-\lambda)P(w|C) \tag{5}$$

Two more possible generative processes are, then, taken into account:

- The word w comes from the document language model, but as a synonym or very semantically similar to a word w' in the document. Hence, this new factor is given by:

$$\sum_{w' \in D} P(w, w'|D) = \sum_{w' \in D} P(w|w', D)P(w'|D)$$

$$P(w|w', D)P(w'|D) = \frac{sim(w, w')}{\sum_{w'' \in D} sim(w, w'')} \frac{tf(w', D)}{|D|}$$

  Where sim is defined to be the cosine similarity.
- With a similar process, we also consider the possibility that the word comes from the whole collections (similarly to a smoothing employing a background model). However, considering the similarity of each word with respect to each other word in the corpus would be too expensive. For this reason, for each word only the set of $N_w = 3$ neighbors are considered (i.e. the N_w words whose embeddings are closest w.r.t. the cosine similarity):

$$\sum_{w' \in N_w} P(w, w'|C) = \sum_{w' \in N_w} P(w|w', C)P(w'|C)$$

$$P(w|w', C)P(w'|C) = \frac{sim(w, w')}{\sum_{w'' \in N_w} sim(w, w'')} \frac{tf(w', C)}{|C|}$$

The final probability can be written as:

$$\begin{aligned} P(w|D) = \lambda & \frac{tf(w,D)}{|D|} \\ & + \alpha \sum_{w' \in D} P(w, w'|D) \\ & + \beta \sum_{w' \in N_w} P(w, w'|C) \\ & + (1 - \lambda - \alpha - \beta)\, P(w|C) \end{aligned}$$

We followed the paper method to build embeddings: 200-dimensional words vectors have been built using continuous bag-of-words (CBOW) with a 5 words window and negative sampling.

Though the authors of that paper claimed they found good results with this models, after implementing it and obtaining very poor scores, we tried to understand what were the reasons and we realized the following:

- the cosine similarity ranges in $[-1, 1]$. As a result, using it to build a probability distribution by computing scores for each event and normalizing by the sum of the scores does not lead to a proper probability distribution (though, the "probabilities" will sum to 1, they can take any real value, both negative or greater than 1). We solve this problem by defining $sim = \frac{1}{2}(cossim(w, w') + 1)$.
- we do not think the estimation of $p(w|w', D)$ is meaningful as the normalization makes us think of a probability distribution like $p(w'|w, D)$. Moreover, this choice leads to some problems. For example, let's consider a document whose words w' are all very similar to the query word w: the probabilities $p(w|w', D)$ will be almost equals for all the w' as the similarities will be close. However, the same happens when all the word in the document are very far from the query word. As a result, the computed score can be completely useless to distinguish interesting documents from documents about completely different topics. In order to solve this problem, we tried 2 different models for $p(w|w', D)$.
  - chosen a word w' in the document, that word can be transformed or not in w, independently from other words. Therefore, $p(w|w', D)$ is modeled as a *Bernoulli* distribution with probability equal to $sim(w, w')$ (i.e. we dropped the normalization term). (Bernoulli GLM)
  - chosen a word w' in the document, that word can be transformed in any word in the vocabulary with a probability proportional to their similarity. Therefore, $p(w|w', D)$ is modeled as a categorical/*Multinoulli* distribution with probability equal to $\frac{sim(w, w')}{\sum_{w'' \in V} sim(w'', w')}$ (where V is the vocabulary). (Multinoulli GLM)

We also implemented 2 other simpler models to use as baselines:

- an "extended" TF-IDF (Generalized TF-IDF). Drawing inspiration from the previous models, for each query word we consider the 3 closest words (but now we keep only those whose similarity is above a threshold of 0.6). The term frequency of a word in a document is its term frequency plus the ones of its neighbors weighted by their similarity. Instead, document frequency is computed considering a "fuzzy" membership: w belongs completely to a document if it is actually a word in it; otherwise, it belongs to the document with a degree equal to the maximum similarity of the neighbors appearing in the document.
- each query and document is represented as the average of the normalized word embeddings of its words (Mean GLM). Given a query, documents are ranked according to their cosine similarity with respect to the query

In order to make the computation of the scores with these models more efficient, it is possible to precompute many values. Indeed, most of the summations can be simplified by gathering the terms and re-writing them as the product of the normalized query word vector and another vector computed only from the document. In this way, the documents vectors can be computed in advance, and the scores of the a word evaluated faster.

Furthermore, in order to quickly retrieve the nearest neighbors of a word we tried to use both Scipy's KDTree (more accurate but still slow when dealing with high dimensional vectors) and FLANN library (much faster but approximated).

2.5 Task 4 - Learning to rank

A different way to build Ranking Models for Information Retrieval is to employ Machine Learning algorithms to learn patterns in data. In our case, the task required supervised learning over a partially annotated dataset which contains relevance values related to some documents with respect to certain queries.

A broad range of ML models can be used to approach this task, examples ranging from Logistic and Linear Regression, to SVMs and Neural Networks. Also, algorithms can be distinguished in Point-wise, Pair-wise or List-wise approaches depending on the input representation and Loss Function they use. Finally, a crucial step into creating efficient models is the use of meaningful features (or signals) as input values to represent the data.

2.5.1 Logistic Regression + Cross-Entropy. In our implementation of Learning to Rank, we started by considering one of the basic approaches: Logistic Regression. This is a simple point-wise model that, given a pair document-query, tries to predict the document relevance, treating the problem as a classification task (between "Relevant" and "Non-Relevant") with a Cross-Entropy loss function. Documents are scored according to the Softmax output of the model, i.e. the probability of belonging to the class "Relevant".

2.5.2 RankPy + LambdaMART. Afterwards, we decided to employ models using a Pair-wise approach, which should be more effective than the previous one. For this task, we decided to start by working with RankPy, a library that should provide an easy-to-use implementation of Ranking Models using LambdaMART. Unfortunately, the library is only available in Python 2, so we had to convert core functions to Python 3 to use them in our project. LambdaMART ranking algorithm [1] was first introduced by Microsoft Research following previous algorithms RankNet and LambdaRank. In further details, this algorithm applies a cost function derived from LambdaRank in combination with MART (Multiple Additive Regression Trees), which is based on gradient boosted decision trees. Our dataset was converted in SVM Light format [2] to make it compatible with the library. Finally, we trained the LambdaMART model on the test set for different numbers of epochs, and then we tested performances on the validation set as specified in the Task 4 description. Sadly, the implementation of RankPy to train the model did not allow us to implement cross-validation because the train methods were already defined internally. For this reason, we tried to use early stopping on the training epochs to avoid over-fitting on the training set.

2.5.3 XGBoost + Pairwise rank/LambdaMART. Despite our efforts to achieve good performances with RankPy models, we did not obtain results comparable with other Ranking Models. Thus, we chose to repeat experiments with LambdaMART using XGBoost library, implementing the code to train the model using 10-folds cross-validation. XGBoost makes it possible to choose between basic Pairwise Rank and LambdaMART, and so we decided to run experiments with both algorithms to compare their performances.

To improve the results obtained with XGBoost, we had to tune its (many) hyper-parameters. This operation was performed using Grid Search over a list of values for each parameters. The initial configuration compared to the one optimizing LambdaMART and the one for basic Pairwise ranking can be seen in table 1.

Table 1: Parameters before initialization, after optimization for LambdaRank, after optimization for LambdaMART

	Init Params	Basic Pairwise	LambdaMART
Learning Rate	0.1	0.1	0.1
# Estimators	100	50	50
Max Depth	5	2	9
Min Child Weight	1	6	1
Gamma	0	0	0
Subsample	0.8	0.7	0.9
Colsample bytree	0.8	0.9	0.55
Scale Pos Weight	1	1	1
Eval Metrics	ndcg@10, map@1000		

When deciding which features to use, we considered different combinations of common signals used in IR LTR tasks, joint with the introduction of some features ideated by ourselves. For query-independent features, we considered the document length, the number of different tokens in the document and the number of stop-words. We considered the same signals for queries in query level features. Regarding query-dependent features, we started by including document-query scores computed by other Ranking Models such as TF-IDF, BM25, Jelinek-Mercer, Absolute Discounting and Dirichlet Prior.

In addition, we tried to include mean, max and min values over TF and TF-IDF values over all terms in the query for a certain document.

Afterwards, we decided to consider the semantical distributed representation of queries and documents using LSI Model with 50 topics. For each of the 50 topics, we computed the product between query and document values for that dimension. The idea is that if a document is relevant for a query, the values for topics present in the first one should have the same sign of the values in the latter. This representation was then integrated or replaced by the cosine_similarity over the two representation, which coincide with the scoring values returned in Task 3 for the LSI Model.

Finally, we thought at a feature that could somehow describe if the query (as a sequence of words) is close to appear in the document. Ideally, if the query is a sentence very similar to a sentence in the document, the document is more likely to be relevant than if the query words were spread across the whole document. However, the query can contain some typos or some words can be different, without changing its structure and meaning. For this reason, we thought at the editing distance between the query and the document drawing inspiration from Approximate String Matching[3] (but considering sequences of words, which means we want to edit/change words, not characters). Indeed, Approximate String Matching is commonly used to infer correct meanings in misspelled user queries. A common measure for this task is the Levenshtein distance, which is the minimum number of alterations in a string needed to make it become another string. However, given that queries are usually much shorter than documents, this score would lose in significance

[2]http://svmlight.joachims.org/

[3]https://en.wikipedia.org/wiki/Approximate_string_matching

(as the number of meaningful edits of the queries would be much less than the number of word insertions needed to reach the same length of the documents). For this reason, we defined our distance as the minimum amount of word edits (substitute, remove, insert) required in the query so that the list of query terms becomes a sublist in the edited document. Lower this value is, more the query should match the document.

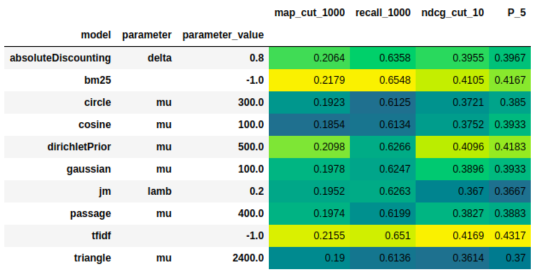
To improve performances in our models, we should have also performed a selection over features. An optimal way to do this is to use Ablation Analysis on features, comparing performances before and after removing a certain information. Due to shortage of time, we could not complete this test, but thanks to XGBoost we could check the f-score of each feature to see which ones were the most informative in creating Decision Trees when using Basic Pairwise ranking and LambdaMART. A description and consideration of this result is reported in 3.4.1.

3 RESULTS AND ANALYSIS

3.1 Task 1

The overall results for the models implemented in Task 1 are shown in Figure 6. The table shows the values for the 4 measures, for the model with the best parameter chosen on the validation set.

Figure 6: Performance on test set for Task 1



model	parameter	parameter_value	map_cut_1000	recall_1000	ndcg_cut_10	P_5
absoluteDiscounting	delta	0.8	0.2064	0.6358	0.3955	0.3967
bm25		-1.0	0.2179	0.6548	0.4105	0.4167
circle	mu	300.0	0.1923	0.6125	0.3721	0.385
cosine	mu	100.0	0.1854	0.6134	0.3752	0.3933
dirichletPrior	mu	500.0	0.2098	0.6266	0.4096	0.4183
gaussian	mu	100.0	0.1978	0.6247	0.3896	0.3933
jm	lamb	0.2	0.1952	0.6263	0.367	0.3667
passage	mu	400.0	0.1974	0.6199	0.3827	0.3883
tfidf		-1.0	0.2155	0.651	0.4169	0.4317
triangle	mu	2400.0	0.19	0.6136	0.3614	0.37

3.1.1 *T-Test.*

Our goal in this assignment was to compare different ranking models. After choosing the parameter that gives the maximum nDCG@10 score on the evaluation set, we computed the following metrics for each model: nDCG@10, precision@5, MAP@1000, Recall@1000 by means of the TREC Eval library. To assure that our results are not due to chance, we perform a statistical test on each pair of models, for each measure. Since we are performing multiple experiments on the same dataset, the problem of multiple comparisons arises. This implies that the probability of committing a Type I error is now dependent on the number of experiments. One of the most common ways to deal with it is the Bonferroni correction, that doesn't require any assumption on the independence and on the distribution of the samples. Since we are comparing for each model four different measures, our significance level of 0.05 is divided by 4 and therefore reduced to 0.0125. To perform the t-test, we use a function implemented in the library Scipy. The function performs a two-tailed t-test for the null hypothesis "the two samples have the same mean". Since we are interested in showing which model is statistically better, we use the function in the following way. The returned variables are the value of the t-test and the associated p-value. If the t-statistic is positive, the first model has a higher mean with respect to the second one. The result is significant if p-value/2 is lower than the significance level, since the two-tailed test considers both sides of the distribution.

3.1.2 *Model Comparison.*

Results for the comparison between lexical models are shown in Figure 7. Even though TF-IDF and BM25 are the most simple models, they clearly outperform all other models.

3.1.3 *Kernel Comparison for PLM.*

For what concern the Position Language Model, we performed a t-test to assess whether a certain kernel performs significantly better than the others. The results are shown in Figure 8. The Gaussian and the Triangle kernel seem to perform better. This might be due to the fact that the shape of the kernel function gives full contribution to the actual position of the word and then decreases slowly, whereas both the Circle and the Cosine kernels drop much faster when getting further, as visible in Figure 4. This means that the first two kernels are more able to capture long-term dependencies. The Passage kernel, that just assigns the same contribution for all words at distance less than σ from the query word, doesn't give results that are statistically significant.

3.1.4 *Query-level comparison.*

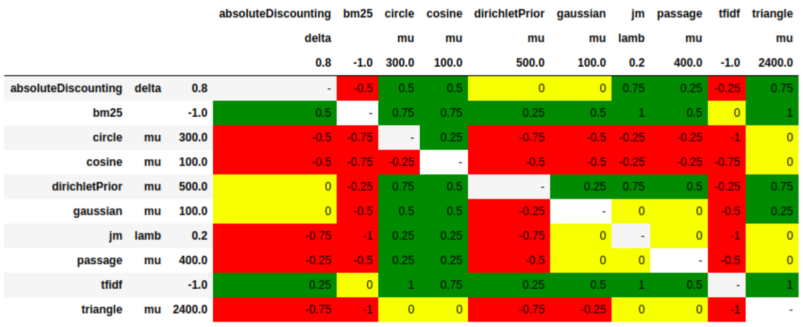
To get a better understanding of the models and how they work, we searched for queries on which different models have very different performances with respect to nDCG@10. Interestingly enough, the biggest overall difference is observed between TF-IDF and PLM (specifically, Circle kernel): for query 103 TF-IDF scores 0.31 more than the PLM model, while for query 182 PLM scores 0.7258 more than TF-IDF (which is a fairly huge difference, given than TF-IDF outperforms the PLM model overall in our experiments). For each query, we considered the documents labeled as relevant in the evaluation dataset and for each document we examined the single word scores for TF-IDF and the single position scores for the PLM, for each query word. The two queries (once tokenized, so with any stopword removed) are:

- 103: welfare reform
- 182: commercial overfishing creates food fish deficit

The main difference between the queries is the length and the frequency of the words. TF-IDF performs better than PLM on query 103 since for most documents, the word "welfare" appears pretty frequently, while "reform" is more rare. Hence the TF-IDF scores will reflect this assigning a high score to "reform" (since is rare) and a high score to "welfare" (since it appears frequently). The PLM on the other hand will be mostly sensible to the (very few in our dataset) cases where the words appear together, as shown in Figure 9. The plot shows the document position on the x-axis, the log-probability on the y axis for single words and for the overall (normalized over the length of the query) score, plus vertical lines where the query words actually appear in the document. Furthermore, the PLM score value, the term frequencies for the single words and the overall TF-IDF for the whole query are displayed.

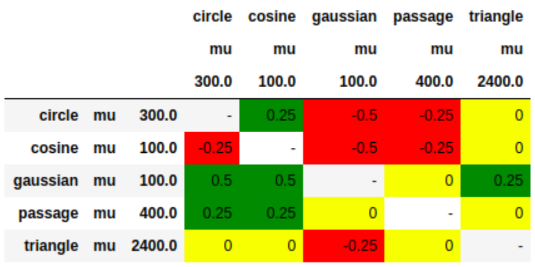
Query 182 on the other hand is pretty long, contains some very unfrequent words ("overfishing", "creates") and some very frequent words ("fish", "food"). Since many documents contain numerous times the more frequent words, TF-IDF correctly weights their scores in order to reduce their contribution to the overall score, as

Figure 7: T-test to compare different Models in Task 1



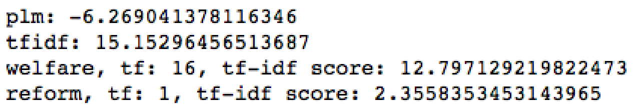
			absoluteDiscounting	bm25	circle	cosine	dirichletPrior	gaussian	jm	passage	tfidf	triangle
			delta		mu	mu	mu	mu	lamb	mu		mu
			0.8	-1.0	300.0	100.0	500.0	100.0	0.2	400.0	-1.0	2400.0
absoluteDiscounting	delta	0.8	-	-0.5	0.5	0.5	0	0	0.75	0.25	-0.25	0.75
bm25		-1.0	0.5	-	0.75	0.75	0.25	0.5	1	0.5	0	1
circle	mu	300.0	-0.5	-0.75	-	0.25	-0.75	-0.5	-0.25	-0.25	-1	0
cosine	mu	100.0	-0.5	-0.75	-0.25	-	-0.5	-0.5	-0.25	-0.25	-0.75	0
dirichletPrior	mu	500.0	0	-0.25	0.75	0.5	-	0.25	0.75	0.5	-0.25	0.75
gaussian	mu	100.0	0	-0.5	0.5	0.5	-0.25	-	0	0	-0.5	0.25
jm	lamb	0.2	-0.75	-1	0.25	0.25	-0.75	0	-	0	-1	0
passage	mu	400.0	-0.25	-0.5	0.25	0.25	-0.5	0	0	-	-0.5	0
tfidf		-1.0	0.25	0	1	0.75	0.25	0.5	1	0.5	-	1
triangle	mu	2400.0	-0.75	-1	0	0	-0.75	-0.25	0	0	-1	-

Figure 8: T-test to compare different kernels for PLM

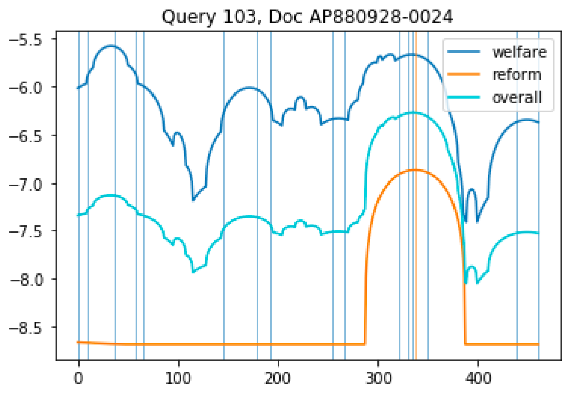


			circle	cosine	gaussian	passage	triangle
			mu	mu	mu	mu	mu
			300.0	100.0	100.0	400.0	2400.0
circle	mu	300.0	-	0.25	-0.5	-0.25	0
cosine	mu	100.0	-0.25	-	-0.5	-0.25	0
gaussian	mu	100.0	0.5	0.5	-	0	0.25
passage	mu	400.0	0.25	0.25	0	-	0
triangle	mu	2400.0	0	0	-0.25	0	-

Figure 9: Query 103, PLM vs TF-IDF



```
plm: -6.269041378116346
tfidf: 15.15296456513687
welfare, tf: 16, tf-idf score: 12.797129219822473
reform, tf: 1, tf-idf score: 2.3558353453143965
```



showed in Figure 10, while assigning a higher score to "commercial". The PLM on the other hand is able to capture better the relative position of the words: in the same Figure is possible to notice how the overall score reaches its maximum in a position close to where the three query words actually appear in the document. This is also noticeable in Figure 11, where TF-IDF assigns a low score since the query words have a low term frequency in the document, while the PLM score is very close the one assigned to the document depicted in Figure 10 since the query words appear close to each other. To summarize, the plots reflect the fact that TF-IDF assigns a score based on the term frequency hence higher for documents containing more times the query words, while the Position Language Model relies on query words co-occurrencies and on the Best position strategy, hence independently from the term frequency two documents receive similar scores if the query words appear close to each other.

Furthermore, the plots allow to get some more insights. The query words that not appear in the document are still assigned a non zero probability due to smoothing and the Dirichlet Prior approach assures this probabilities to be directly related to the word frequency in the collection (hence in Figure 10 "overfishing" has a clearly lower log-probability than "creates" and "deficit"). Finally, the single words log-probability trend makes the understanding of kernel's action more intuitive, for example in Figure 9 the word "reform" gives a single circular-shaped contribution, while the curve for "fish" reflects the more complex interaction between the 16 times the word appears in the document.

3.2 Task 2

To test and compare Language Semantic Models, we chose a set of values (k = 50, 100, 200, 300) for the number of Topics commonly used in literature depending on the type and size of corpus used. These k values were tried on three different models. The first Model is Latent Semantic Indexing with Cosine Similarity as scoring measures between query and document representation. For Latent Dirichlet Analysis we tried both using negative KL-Divergence (commonly used to find how well a probability distribution describes another one) and Cosine Similarity. The idea between using Cosine Similarity on probabilities distributions is that, if two distributions are similar (and than query and document are describing the same topics), they should have similarly high or low values for the same topics.

In Figure 12, we can see how LDA behaves when increasing the number of topics. By looking at how score changes over the graphic, we can infer that our corpus is complex enough in terms of topics. For smaller corpus, $k = 100$ is usually enough to map all topics present in the documents. In our case, performances at 200 topics are noticeably better than at lower values. On the other side,

Figure 10: Query 182, PLM vs TF-IDF

```
plm: -8.752898687996751
tfidf: 18.123988638900364
commercial, tf: 2, tf-idf score: 3.510223853022827
food, tf: 6, tf-idf score: 5.3771922984038465
fish, tf: 8, tf-idf score: 9.23657248747369
```

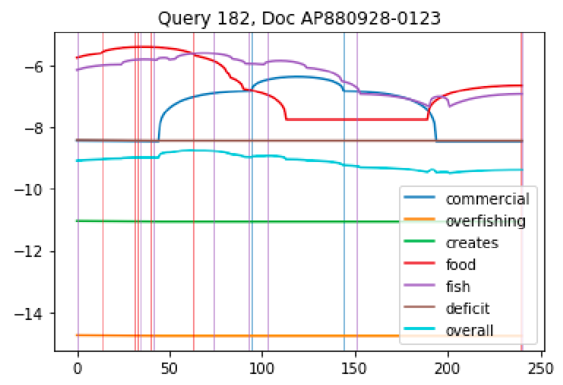
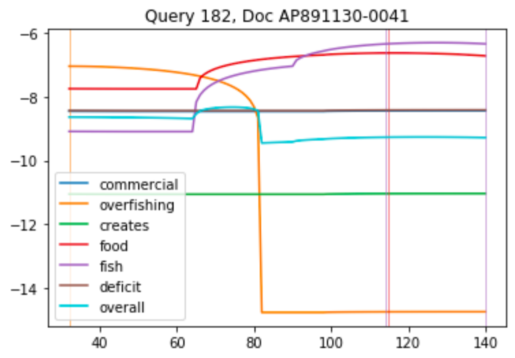


Figure 11: Query 182, PLM vs TF-IDF

```
plm: -8.327628293766287
tfidf: 13.197154213910942
overfishing, tf: 1, tf-idf score: 6.663473427020741
food, tf: 1, tf-idf score: 1.9153945431533548
fish, tf: 2, tf-idf score: 4.618286243736845
```



increasing the k to 300 results in a decrease in score that can be explained with our model describing topics that are actually just noise in the data.

3.2.1 Model Comparison. The overall results for the models implemented in Task 2 are shown in Figure 13. The table shows the values for the 4 measures, for the model with the best parameter chosen on the validation set.

It is evident that scores obtained by distributional models are not as high as the ones obtained in TF-IDF (chosen as comparison). Among the LSMs, LDA with Cosine Similarity appear to have higher performances over nDCG@10, P_5 and map@1000. Because in this

Figure 12: Performances of LDA with Cosine Similarity while varying the number of Topics.

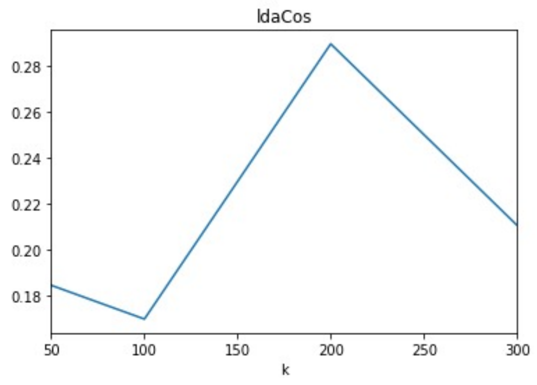
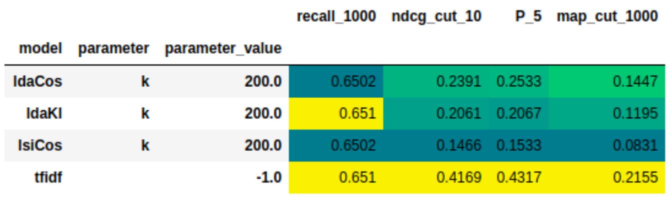


Figure 13: Performance on test set for Task 2

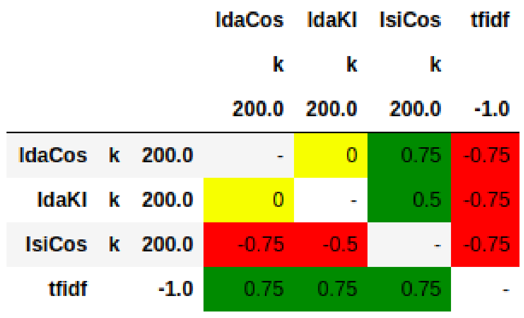


model	parameter	parameter_value	recall_1000	ndcg_cut_10	P_5	map_cut_1000
ldaCos	k	200.0	0.6502	0.2391	0.2533	0.1447
ldaKl	k	200.0	0.651	0.2061	0.2067	0.1195
lsiCos	k	200.0	0.6502	0.1466	0.1533	0.0831
tfidf		-1.0	0.651	0.4169	0.4317	0.2155

Task we are re-ranking the first 1000 documents chosen by TF-IDF, Recall scores are constant.

3.2.2 T-Test. Statistical significance tests were run at the same way of Task 1 to confirm or refuse observation on which Model performed better. In Figure 14 we report statistical test's results for this task.

Figure 14: T-test to compare different Models in Task 2



			ldaCos	ldaKl	lsiCos	tfidf
			k	k	k	
			200.0	200.0	200.0	-1.0
ldaCos	k	200.0	-	0	0.75	-0.75
ldaKl	k	200.0	0	-	0.5	-0.75
lsiCos	k	200.0	-0.75	-0.5	-	-0.75
tfidf		-1.0	0.75	0.75	0.75	-

The interpretation of this tests tells us that LDA is statistically better than LSI models on most features. Differences between using LDA with Cosine Similarity or with KL-Divergence are not large enough to be detected as statistically significant.

Overall, we confirmed our expectation of LSMs not reaching the same scores as TF-IDF or BM25, but our implementations still managed to get noticeable results, even considering that the score is biased by the pre-selection of the first 1000 documents for every query.

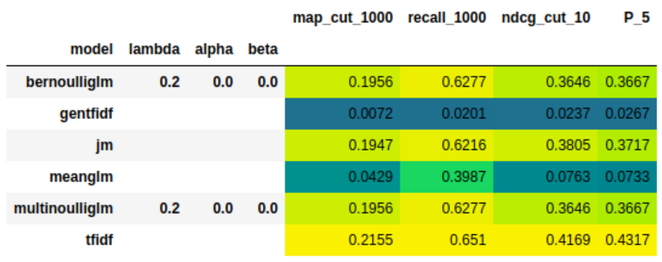
3.3 Task 3

Our versions of Generalized Language Model have 3 parameters as well as the original version [2]. As in the previous tasks, we chose the best set of parameters considering the ndcg@10 measure on the Validation Set. However, we chose to fix λ to 0.2 as it was suggested in the original paper and since this value was also the best parameter found in Task 1 for the JM model. We explored α and β both in the range $[0.0 - 0.4]$ with a step of 0.1.

As expected, the original version of the GLM performed very poorly, scoring almost 0 on all the measures.

Here, we also compare the results of the models implemented in this task with TF-IDF (as one of the best model found so far) and JM (since our 2 versions of the GLM are a generalization of JM).

Figure 15: Performance on test set for Task 3



model	lambda	alpha	beta	map_cut_1000	recall_1000	ndcg_cut_10	P_5
bernoulliglm	0.2	0.0	0.0	0.1956	0.6277	0.3646	0.3667
gentfidf				0.0072	0.0201	0.0237	0.0267
jm				0.1947	0.6216	0.3805	0.3717
meanglm				0.0429	0.3987	0.0763	0.0733
multinoulliglm	0.2	0.0	0.0	0.1956	0.6277	0.3646	0.3667
tfidf				0.2155	0.651	0.4169	0.4317

Figure 15 shows the performances on the test set of the best set of parameters of each model (chosen w.r.t. the validation set).

The first thing to notice is that both BernoulliGLM and Mutlinoulli GLM fall to the standard JM model setting α and β to 0 after validation. The difference in the scores of this 2 models w.r.t JM is due to the different background probabilities employed: this new background probability seems to improve racall@1000 and map@1000 on the test set, but they have worse rsults on p@5 and ndcg@10. Moreover, we found that using one of these two measures for the validation the smoothing from neighbor words happens to be slightly useful reaching a recall of 0.6429 and a map of 0.2369 (statistically significantly better than JM), resulting in the following best configuration (for both of the models and both of the measures): $\alpha = 0.0, \beta = 0.4$.

As regards our 2 baselines, Generalized TF-IDF seems to perform very bad: neighbors may have added noise to the scores or their introduction may have made TF and IDF lose their meaning, penalizing too much words with all the neighbors (not filtered by the threshold) or those with very popular neighbors (with high DF). MeanGLM is performing slightly better and seems to have at least a good recall in the first 1000 documents.

In Figures 16 and 17, ndcg@10 measure is shown w.r.t the value of α for both of the models. The different lines in the plots correspond to each of the 5 values of β. Though the different values of β do not seem to change significantly the results, using a value greater

Figure 16: Perfomances of BernoulliGLM while varying α and β

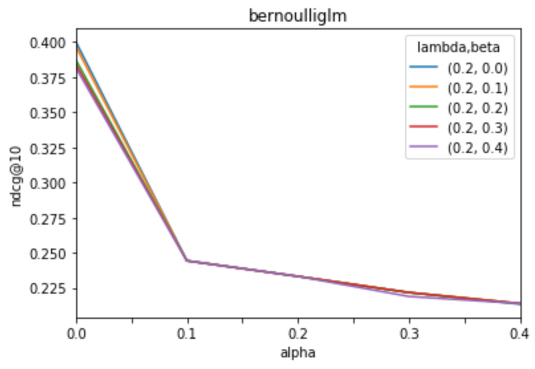
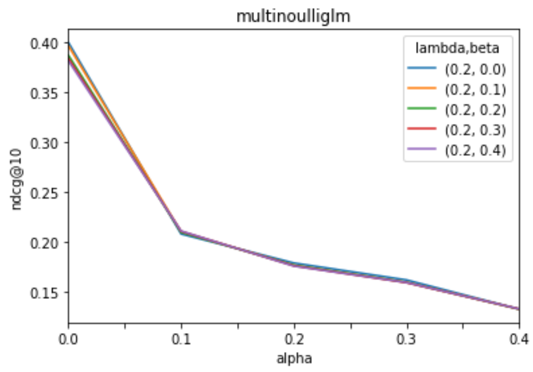


Figure 17: Perfomances of MultinoulliGLM while varying α and β



than 0.0 for α decreases the performances a lot. However, even with high values of α, these models perform much better than the original version. In addition, the performance of Bernoulli GLM seems to decrease more slowly than the one of Multinoulli GLM with respect to α.

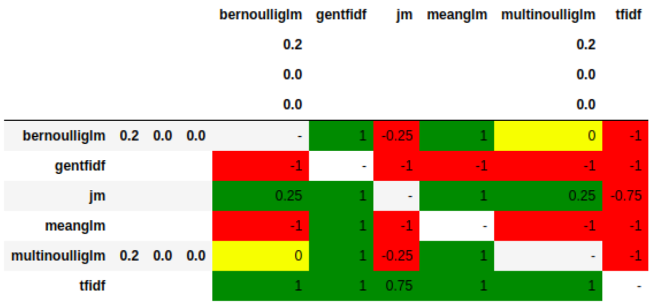
Finally, in Figure 18 the table with the significance tests is shown. These values confirm that: our 2 baselines do not perform very well, though Mean GLM seems better than Generalized TFIDF; TF-IDF remains a powerful model, very hard to defeat. However, comparing the 2 GLM and JM, it emerges not only that the new background probability is not significantly better than the one in Task 1, but also that it is statistically significantly worse on one measure (precisely, the ndcg@10).

3.4 Task 4

3.4.1 About Feature Selection. To have an idea of which features contributed more to the generation of Ranking in our models, we considered f-score reported in XGBoost. In most of the cases, the following features had values significantly higher than the others (in descending order):

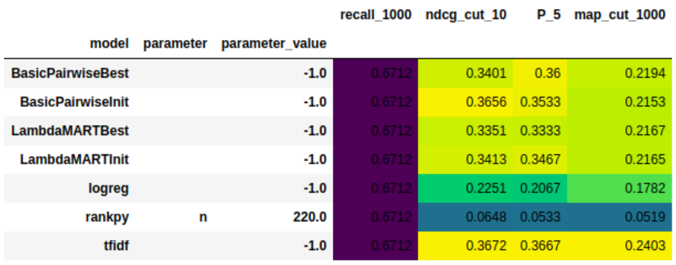
- TF-IDF means over query terms
- Cosine Similarity with LSI

Figure 18: T-test to compare different models in Task 3



				bernoulliglm	gentfidf	jm	meanglm	multinoulliglm	tfidf
				0.2				0.2	
				0.0				0.0	
				0.0				0.0	
bernoulliglm	0.2	0.0	0.0	-	1	-0.25	1	0	-1
gentfidf				-1	-	-1	-1	-1	-1
jm				0.25	1	-	1	0.25	-0.75
meanglm				-1	1	-1	-	-1	-1
multinoulliglm	0.2	0.0	0.0	0	1	-0.25	1	-	-1
tfidf				1	1	0.75	1	1	-

Figure 19: Performance on test set for Task 4



			recall_1000	ndcg_cut_10	P_5	map_cut_1000
model	parameter	parameter_value				
BasicPairwiseBest		-1.0	0.6712	0.3401	0.36	0.2194
BasicPairwiseInit		-1.0	0.6712	0.3656	0.3533	0.2153
LambdaMARTBest		-1.0	0.6712	0.3351	0.3333	0.2167
LambdaMARTInit		-1.0	0.6712	0.3413	0.3467	0.2165
logreg		-1.0	0.6712	0.2251	0.2067	0.1782
rankpy	n	220.0	0.6712	0.0648	0.0533	0.0519
tfidf		-1.0	0.6712	0.3672	0.3667	0.2403

- Absolute Discounting
- TF-IDF minimum value over query terms
- BM-25 score
- Query Length

After obtaining these results, we tried to give an interpretation to find the reason why such features appear to be more relevant than others. TF-IDF, which seem to be more important, is also the scoring method that reported the higher results in previous experiments over all the models employed. It is likely that XGBoost has learned to use this information to give higher or lower scores to query-document pairs accordingly. Probably, the mean value over TF-IDFs for each query term and the overall TF-IDF score carry redundant information and so as the former is used to create branches in decision trees the latter is probably not used and receives a lower score. The same reasoning as for TF-IDF can be applied to BM25, which is the second most performant Model, with results little lower than TF-IDF. Cosine Similarity with LSI represent documents and queries in a semantical distributed way. For this reason, it could carry a different type of information with respect to other Lexical Models scores. Finally, an explanation for the minimum over TF-IDFs values being relevant is that if the number is not null, every word in the query appears in the document. As documents are pretty short, it is likely that in few of them the minimum TF-IDF is a positive number, and then those documents should receive higher scores than ones in which the minimum is zero.

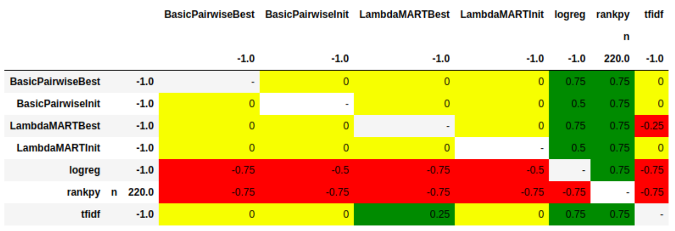
Although f-scores are not a true indication of the importance of a feature (for example, similar features can carry the same information and thus one of the two is reported as irrelevant), we could get an approximate idea to be verified with Ablation Analysis in future work.

3.4.2 Model Comparison. The overall results for the models implemented in Task 4 are shown in Figure 19. The table shows the values for the 4 measures, for the model with the best parameter chosen validating on the test set and tested on the validation one (this is different from previous task, so values are not directly comparable). Parameter n for RankPy is the number of epochs in training.

The initial model we implemented, Logistic Regression, did not perform as good as other Models. This was expected, as it uses Point-wise approach on loss function, which has major pitfalls for ranking. RankPy implementation of LambdaMART was inserted in the report for completeness, although the results show that it didn't work as expected, probably also due to the impossibility to perform cross-validation during training. Finally, models generated through XGBoost achieved noticeable scores, almost reaching TF-IDFs ones in all the measures. No particular differences can be noticed between Pairwise Ranking and LambdaMART. Also, tuning hyper-parameters did not change the results on the test set, despite improvements on the validation set using 10-fold fluctuated around 5% in both models.

3.4.3 T-Test. Statistical significance tests were run at the same way of Task 1 to confirm or refuse observation on which Model performed better. In Figure 20 we report statistical test's results for this task.

Figure 20: T-test to compare different Models in Task 4



			BasicPairwiseBest	BasicPairwiseInit	LambdaMARTBest	LambdaMARTInit	logreg	rankpy	tfidf
								n	
			-1.0	-1.0	-1.0	-1.0	-1.0	220.0	-1.0
BasicPairwiseBest		-1.0	-	0	0	0	0.75	0.75	0
BasicPairwiseInit		-1.0	0	-	0	0	0.5	0.75	0
LambdaMARTBest		-1.0	0	0	-	0	0.75	0.75	-0.25
LambdaMARTInit		-1.0	0	0	0	-	0.5	0.75	0
logreg		-1.0	-0.75	-0.5	-0.75	-0.5	-	0.75	-0.75
rankpy	n	220.0	-0.75	-0.75	-0.75	-0.75	-0.75	-	-0.75
tfidf		-1.0	0	0	0.25	0	0.75	0.75	-

The interpretation of this tests tells us confirms the observation that XGBoost Models reach score levels comparable with TF-IDF. This was actually expected, as TF-IDF values were among the features in the LTR input vector. LambdaMART on RankPy and Logistic Regression were statistically worse than every other model in Task 4

REFERENCES

[1] Chris J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview.* Technical Report. https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/

[2] Debasis Ganguly, Dwaipayan Roy, Mandar Mitra, and Gareth J.F. Jones. 2015. Word Embedding Based Generalized Language Model for Information Retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '15).* ACM, New York, NY, USA, 795–798. https://doi.org/10.1145/2766462.2767780

[3] 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 (SIGIR '09)*. ACM, New York, NY, USA, 299–306. https://doi.org/10.1145/1571941.1571994