Retrieval Methods in Information Retrieval

Comparing different Information Retrieval methods and determining their optimal parameters

Luca Simonetto
University of Amsterdam
11413522

Nuno Mota
University of Amsterdam
11413344

ABSTRACT

With this assignment we set out to implement a wide variety of different retrieval models, determine their optimal parameters and compare their efficacy, in order to pick the best performing one.
For the first task, the optimal parameters obtained were: Jelinek-Mercer-$\lambda = 0.2$, Dirichlet-Prior-$\mu = 1500$, Absolute-Discounting-$\delta = 0.9$, Positional-Language-Model Kernel: Gaussian–$\mu = 1000$, Triangle–$\mu = 500$, Cosine–$\mu = 500$, Circle–$\mu = 500$ and Passage–$\mu = 500$. The best performing model was BM25 after comparing a metric based on all measures, with a significance verification based on a t-student double tailed test with an overall significance level of 0.05.
For the second task we determined that the best model, given the parameters used, was word2vec, after comparing a metric based on all measures but recall, with a significance verification based on a t-student double tailed test with an overall significance level of 0.05.
In the third task we decided to compare all models and came to the conclusion, using the same significance test as the one performed in task 2, that the best model of all those implemented was LTR.

ACM Reference format:
Luca Simonetto and Nuno Mota. 2016. Retrieval Methods in Information Retrieval. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 8 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This report serves to explain the resoning and particularities behind the implementation of several different Retrieval Methods, in the scope of Information Retrieval (IR). The report is divided into three major parts, corresponding to the tasks of the assignment.
In the first part we will cover the implementation and parameter estimation methods, and the motivation that led to the choice of said methods, for 6 models, namely: TF-IDF, BM25, Jelinek Mercer, Dirichlet Prior, Absolute Discounting and Positional Language Models (for 5 distinct kernels).
We will then cover the second part, where we estimate the parameters for 4 Latent Semantic Models (LSM) and construct for each of them a specific query/document representation. The models in question are word2vec, Indexing by Latent Semantic Analysis (LSI), Latent Dirichlet Allocation (LDA) and doc2vec.
Finally, on the third part, we will briefly cover Learning to Rank models, where we will build on what was covered previously.

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

2 TASK 1

On this section we will briefly explain the intuition behind each model, present the estimated optimal parameters, for the models that so require, and, at the end, perform an evaluation of the results of our comparisons between the different models.

2.1 TF-IDF

TF-IDF stands for *Term Frequency - Inverse Document Frequency* and it is a method that tries to estimate how specific to a context a word is. In a way, it tries to distinguish between the words common to most documents in a corpus from those that are closely related to the topic of the document itself. Words like "the" and "that" are often independent from the topic of the document and serve as mere language connectors, as the basic structure of language, failing to add any extra meaning to the document's topic and are, hence, irrelevant from an IR perspective. On the other hand, words like "doctor" and "surgery" will not appear on all documents, allowing us to associate them, and the documents they are on, to a medical context.
That being said, TF-IDF is calculated as follows:

$$\texttt{TF-IDF}(t;d) = \log\left(1 + \texttt{tf}_{t,d}\right) \log\left(\frac{n}{\texttt{df}_t}\right) \tag{1}$$

Where $\texttt{tf}_{t,d}$ is the frequency of term t in document d, n is the total number of documents and $\texttt{df}_t$ is the number of documents on which term t is present.
For this model no parameter estimation was required.

2.2 BM25

BM25 is the short name for Best-Match 25, the 25th model that Robertson and Jones came up with based on TF-IDF-like retrieval methods. The model is based on a Bag-of-Words approach that, like TF-IDF, attempts to estimate the relevance of a document given the frequency of the query terms in a document and the term's document frequency. It also tries to attribute some weighting parameters, to better control the importance of each factor.
The full formula for score$\texttt{BM25}$(t; d) is as follows:

$$\sum_{uniquet \in q} \frac{(k_1+1)\,\texttt{tf}_{t,d}}{k_1\left((1-b)+b\left(\frac{l_d}{l_{avg}}\right)\right)+\texttt{tf}_{t,d}}\texttt{idf}_t \tag{2}$$

BM25 normalizes the term frequency according to the document length, attempting to better represent the importance of a term within the document itself. For example, given document D1 and document D2 and the query term "elephant". If "elephant" occurs in D1 5 times and 50 times in D2 we could expect D2 to be more relevant, given the query term. Let us however consider the possibility that D1 has a total length of 100 words and that D2 has 100000 words. It is then easy to see that whilst D2 has a higher frequency of the term "elephant" it is also a term that has a significantly lower preponderance in the document, being the case that the usage of the term itself could be a by-product of the main topic of the document, allowing for the possibility of D1 actually being more relevant than D2. It is this concept that $(1-b)+b\left(\frac{l_d}{l_{avg}}\right)$ tries to capture. Parameter b allows for a more precise control of the influence of this characteristic.
The term k_1, on the other hand, tries to control how much later increments to $\mathtt{tf}_{t,d}$ contribute to the overall score. That is, if a word has already shown up several times in a document, how much will future occurences contribute to the overall score.

For this model we did not estimate any of the said parameters and used the following values:

k_1	b
1.2	0.75

2.3 Jelinek-Mercer smoothing

Jelinek-Mercer is a Language Model (LM) that ranks documents according to the likelihood that they would generate the requested query, given the LM for that document and the overall LM for the collection of documents. The LM generated for each document is nothing more than the Maximum Likelihood Estimation (MLE) of the probability of a word occurring in the document, that is $\frac{\mathtt{tf}_{t,d}}{|d|}$.
As for the overall LM of the collection, the background language model, it is also the MLE, but of a word occurring in the collection, i.e. $\frac{\mathtt{tf}_{t,C}}{|C|}$.
The formula for calculating the score is as follows:

$$score(q,d)=\prod_{t\in q}\left(\lambda\frac{\mathtt{tf}_{t,d}}{|d|}+(1-\lambda)\frac{\mathtt{tf}_{t,C}}{|C|}\right) \tag{3}$$

The term associated with λ gives us probability estimation according to the LM for the document in question, while the term associated with (1-λ) allows us to interpolate the probability estimation for a word that isn't present in the document, through usage of the background language model. This way, the background language model acts as smoothing agent to the LM of a single document, allowing the possibility of assigning a score to documents which might not have all the query terms. However, if a query term does not exist in the collection at all, it will make the score be 0.
It should be noted that it might be necessary to, instead, compute the logarithm of formula (3), seeing as the probabilities can be extremely small, which could cause the occurrence of underflows. However, we were not faced with this problem and, so, did not use the log form.
The method used for estimating the λ value, and the value itself, will be discussed after the discussion of all LMs, together with the specification of the actual estimated values.

2.4 Dirichlet-Prior smoothing

As with Jelinek-Mercer smoothing the Dirichlet-Prior smoothing arises from an attempt to curb the limitations of pure additive smoothing, which assigns the same probability to all unseen words. In order to achieve that a prior, a Dirichlet conjugate prior to be more specific, is assumed when considering our estimation of the relevance of a document given a query. In the end we get that our score estimation formula is as follows:

$$score(q,d)=\prod_{t\in q}\left(\frac{\mathtt{tf}_{t,d}+\mu\frac{\mathtt{tf}_{t,C}}{|C|}}{|d|+\mu}\right) \tag{4}$$

After some simple manipulation we can get the formula to look like:

$$score(q,d)=\prod_{t\in q}\left(\frac{|d|}{|d|+\mu}\frac{\mathtt{tf}_{t,d}}{|d|}+\frac{\mu}{|d|+\mu}\frac{\mathtt{tf}_{t,C}}{|C|}\right) \tag{5}$$

And finally:

$$score(q,d)=\prod_{t\in q}\left(\frac{|d|}{|d|+\mu}\frac{\mathtt{tf}_{t,d}}{|d|}+\left(1-\frac{|d|}{|d|+\mu}\right)\frac{\mathtt{tf}_{t,C}}{|C|}\right) \tag{6}$$

We can easily conclude that this is the same interpolation formula used in Jelinek-Mercer smoothing, but with $\lambda=\frac{|d|}{|d|+\mu}$. So, the introduction of a Dirichlet conjugate prior leads to a natural interpolaion with the background model. However, the one obtained by the Dirichlet Prior is more expressive than the one present in the Jelinek-Mercer model, since when a document is longer more preponderance is given to the LM of the document itself. The reasoning behind can be somewhat simplified to the assumption that longer documents have a higher chance of having a wider variety of words, which can be interpreted as providing better estimates for the LM, and thus not needing to rely on the background language model or, at least, not rely so heavily.
Nonetheless, we are still left with a μ parameter that allows for a finer tuning of the model. The estimation of this parameter, and its optimal value, will be explained and presented further on the report, along with the explanation for the other language models.
Again, taking the logarithm of the formula can be a good idea to avoid the occurrence of underflows, however we were not faced with that problem.

2.5 Absolute Discounting

Absolute Discounting is also an interpolation method. In this case the method was created because of the problems caused by Laplace smoothing, which, for large vocabularies, adds a high number of counts related to non-existant words that will significantly change the probabilities in an absurd way, misrepresenting the true probabilities of the LM.

Absolute Discounting tries to mitigate that situation by instead removing some probability mass from the seen words and redestributing it among the unseen ones. That intuition yields the following formula:

$$score(q,d) = \prod_{t \in q} \left(\frac{\max\left(\mathtt{tf}_{t,d} - \delta, 0\right)}{|d|} + \frac{\delta |d_u|}{|d|} \frac{\mathtt{tf}_{t,C}}{|C|} \right) \quad (7)$$

Here $|d_u|$ is the number of unique words seen in document d and, together with δ, is what determines how much probability mass is shifted towards the background language model.
δ is a parameter that should be, and was, estimated in order to obtain the best possible performance of the model. Again, the estimation and value of this parameter will be explained further on.
Also, in the presence of underflows it is advised to take the logarithm of the scoring formula.

2.6 Positional Language Model

This kind of approach is distinct from the previous ones, in the sense that instead of attempting to create a LM for the whole document, it creates instead a LM for each position in a document. The document is then ranked according to its Positional Language Models (PLMs). The advantage of this approach is that it allows us to model the best-matching-position of a document.
The idea is to attribute a score for those words in the document that match query terms and to propagate those scores to nearby positions of those where the query words occur. The overall score of a position will then be the sum of all these propagated effects, plus it's own value (which is 0 if it is not a query term and a value different from 0, according to the metric used, otherwise). The propagated effect diminishes with the distance to the position of the respective propagating query term, so that it allows us to model the proximity of query words in the document, since closer positions get a higher propagated value.
There are different ways to calculate the propagated value and they are based on kernel functions. In this assignment we used 5 different kernels, more specifically:

Gaussian Kernel:

$$k(i,j) = \begin{cases} \exp\left(\frac{-(i-j)^2}{2\sigma^2}\right), & if 2|i-j| \leq \sigma \\ 0 & , otherwise \end{cases} \quad (8)$$

Triangle Kernel:

$$k(i,j) = \begin{cases} 1 - \frac{|i-j|}{\sigma}, & if |i-j| \leq \sigma \\ 0 & , otherwise \end{cases} \quad (9)$$

Cosine Kernel:

$$k(i,j) = \begin{cases} \frac{1}{2}\left(1 + \cos\left(\frac{|i-j|\pi}{\sigma}\right)\right), & if |i-j| \leq \sigma \\ 0 & , otherwise \end{cases} \quad (10)$$

Circle Kernel:

$$k(i,j) = \begin{cases} \sqrt{1 - \left(\frac{|i-j|}{\sigma}\right)^2}, & if |i-j| \leq \sigma \\ 0 & , otherwise \end{cases} \quad (11)$$

Passage Kernel:

$$k(i,j) = \begin{cases} 1, & if |i-j| \leq \sigma \\ 0, & otherwise \end{cases} \quad (12)$$

Notice that the kernels are only propagated up to σ distance from the position of the query term in question. For the Gaussian kernel that term is actually 2σ. The advantage of doing this is that we can ignore positions that would receive an extremely small propagation value, allowing us to speed up the computation by simply ignoring those positions.

After the propagation is finished we use a ML estimation to get the probabilities for $p(t|Q)$, that is, a ML estimation for the terms in the query.
Our PLMs themselves, $p(t|D,i)$, are given by the following formula:

$$p(t|D,i) = \frac{c'(t,i) + \mu \frac{\mathtt{tf}_{t,C}}{|C|}}{\sum_{w \in D \wedge w \in Q} (c'(w,i)) + \mu} \quad (13)$$

Where $c'(w,i)$ is the total sum of the propagated counts at position i for word w.
For calculating the final score assigned to a document we can imagine that the PLMs actually give us multiple representations of D. As a consequence, we can adopt the KL-Divergence retrieval model to score each position of the document as follows:

$$Score(Q|D,i) = -\sum_{t \in D \wedge t \in Q} \left(p(t|Q) \log\left(\frac{p(t|Q)}{p(t|D,i)}\right)\right) \quad (14)$$

After having scored each PLM we need to finally decide on how to score the actual document. There are several possible approaches, but the one taken in this assignment is to simply pick the highest of all PLMs of a document, also know as Best Position Strategy.
The sigma considered for this approach was always the same, as a measure of simplification. The chosen value was 50. For this model we needed to estimate the value of the μ parameter and preferably do so for each kernel type. The method followed is the same as the one for the other models, so we explain that in the next subsection. We'll also present the optimal μ values there.

2.7 Parameter Estimation and Choice

As was previously stated some of the models that we talked about have some parameters that, when tuned properly, allow for a better overall performance. It was then of our interest to optimise said parameters.
Care should be taken when tackling the parameter estimation problem, for if the set of possible parameters to be tested is too large, the computation necessary to determine the best one might, in the end, not be cost effective.
With that in mind we considered only a small subset of possible parameteres:

Jelinek-Mercer	Dirichlet-Prior/PLM	Absolute Discounting
[0.1, 0.2, ..., 0.9]	[500, 1000, 1500, 2000]	[0.1, 0.2, ..., 0.9]

The method used to estimate the parameters of the different models is itself important, for the choices made can greatly influence the final chosen parameter.
We decided to use Grid Search and will now explain how the parameter estimation is done for each model (the method is the same for all models, excluding a few exceptions): We pick a parameter value that hasn't been picked yet and for each query in the validation set we create a ranking of all documents. For PLM we only consider documents where at least one of the query terms exists in the document, otherwise the estimation time would be too long. Having ranked the documents we use the TREC Eval tool to report on NDCG@10, Mean Average Precision (MAP@1000), Precision@5 and Recall@1000, for each query.
After having collected all the measures for every single parameter we count the total number of times that each parameter value performs better against every other parameter value for the average, over all queries, of a specific measure. So, let's say, for example, that for $\mu = 0.1$ Jelinek-Mercer has a better average precision than Jelinek-Mercer with $\mu = 0.2$. Then we add a win to parameter $\mu = 0.1$. We repeat this comparison for te other 3 measures and then repeat for the remaining μ values. In the end we'll have a total number of wins for parameter $\mu = 0.1$. After having done the same for every possible value of the μ parameters we pick the value that has the most wins. Notice that we have implicitly assigned equal weighting to all 4 measures, seeing as we consider that all the metrics give an equally important insight of the calculated ranking. Using this evaluation method we arrived to the following values:

Jelinek-Mercer	Dirichlet-Prior	Absolute Discounting
0.2	1500	0.9

For the PLM models with specific kernel we got:

Gaussian	Triangle	Cosine	Circle	Passage
1000	500	500	500	500

We would like to note that one possible improvement to our method could have been based on testing each parameter against each other using a student t-test with the samples corresponding to the measure values obtained for each query. That way we would have performed a significance test on the assumptions that having a greater average implied being the better method, considering a measure.

2.8 Model Performance Evaluation and Comparison

The next step is to determine which of the models performs better on the test set. The method for doing this consists on obtaining new document rankings for each model, using their optimised parameters and the queries belonging to the test set. When the rankings have been collected we perform a similar method as the one used

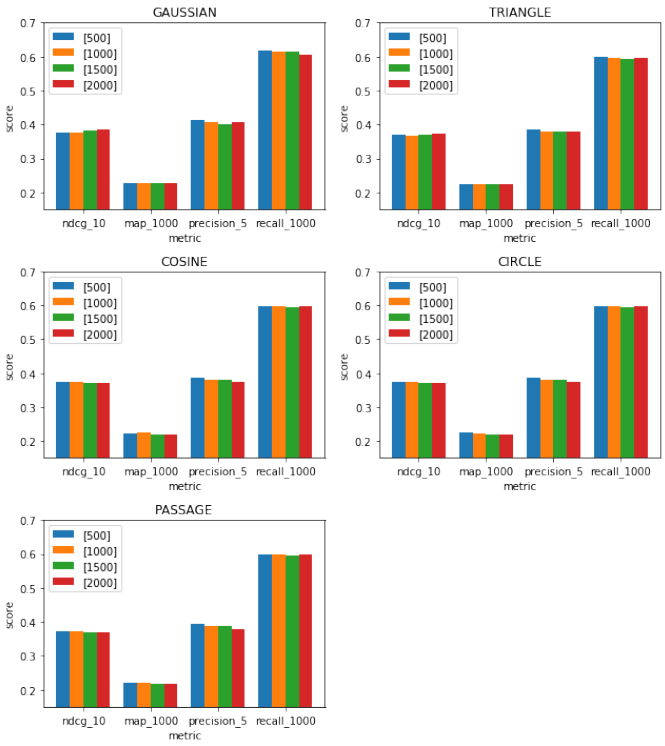

Figure 1: Scores comparison between different positional language models

to estimate the optimal model parameters. However, instead of just comparing the overall average for each measure, here we perform a significance test between all models. The significance test in question is a double-tailed paired t-student test, which determines whether a proposed Null Hypothesis is correct, within a certain confidence interval. In this case our paired data will be the obtained measures for the two comparing models, where the result obtained for each query is considered a sample point. Doing this we are able to determine whether the null hypothesis can be rejected or not. In this case the null hypothesis is that each model will have the same value for the average of the measure in question. If the null hypothesis is rejected we just assume that the highest average is actually representative.
The similarity between this method and the parameter estimation comes from the win-count method. For example, let's say we are comparing model A and B. We first get their measures for each query and then compare, one measure at the time, which one performs better according the the significance test. That is, if the t-test indicates that model A performs better than B under the Precision@5 measure then model A is assigned 1 win. Then we do the same for the other 3 measures, assigning wins accordingly. Keeping on with this method we then compare model A with every other model and then do the same process for the other models (avoiding unecessary repetitions). In the end we consider that the best performing model is the one that has the most wins.
It should be noted that only the Gaussian kernel for the PLM models was used when determining the best performing model using this

method. This was due to the high computation time required to obtain the document rankings for the query test set. Nonetheless, we feel that the Gaussian kernel is the model that would provide the best ranking anyway, since the propagated values occur in a very natural way, diminishing the propagated value smoothly along the surrounding words. The average results obtained for all measures when calculated for the validation set seem to support our reasoning. However, this intuition was not properly tested.

Also, since we will be performing multiple significance test we should beware the multiple testing problem. When testing simultaneously a different number of hypothesis we should understand that there is a probability that at least one of the results is significant just due to chance. The higher the number of hypothesis being tested the higher the probability that this is the case. So, if we wish to maintain an overall level of confidence for the whole experiment there are several different approaches that can be considered.

The method considered for this assignment was a simple one, though extremely conservative. The method is the Bonferroni correction, which simply states that the significance level α should be set to $\frac{\alpha}{N}$, where N is the total number of hypothesis being tested. Again, we stress that this correction method can create a significant portion of false positives, that is, we can get a significant proportion of t-tests that determine the Null Hypothesis is correct, i.e., that the average scores are actually identical or, in other words, that none of the tests performs better than the other.

In our case we wish to have a significance level $\alpha = 0.05$, which gives us a 95% confidence on our results, according to our metric. The total number of hypothesis that were tested was $M \sum_{k=1}^{K-1} k = \frac{M}{2}(K-1)K$, where M is the number of measures, which is 4, and K is the number of models being compared, in this case 6 (TF-IDF, BM25, Jelinek-Mercer, Dirichlet-Prior, Absolute-Discounting and PLM-gaussian-kernel), which yields a final number of hypothesis testing of 60. Our alpha is then $\frac{0.05}{60} = 0.00083$ for each t-test.

Given the previous considerations the number of wins for each model was the following (JM stands for Jelinek-Mercer, DP for Dirichlet-Prior and AD for Absolute-Discounting), where we can see that BM25 was the best, according to our metric:

TF-IDF	BM25	JM	DP	AD	PLM-Gaussian
2	6	0	4	2	0

The performance evaluation gives a ranking of the task 1 models:

(1) **BM25**
(2) **Dirichlet Prior**
(3) **Absolute Discounting** and **TF-IDF**
(4) **Gaussian PLM**

2.9 Queries analysis

We executed a small experiment in order to determine if there were any particularly difficult queries for the models.

Firstly, for each query we calculated each measure's value, as given by each model, and recorded those values. Then we selected each query, one by one, and determined the average and standard deviation of each measure, given the values obtained by each model.

We then took a look at those queries whose sum of the standard

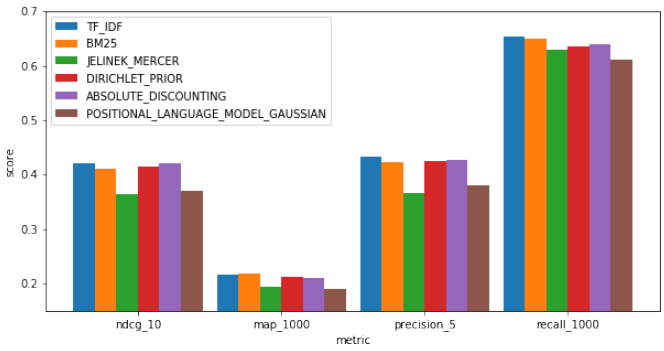

Figure 2: Scores comparison between the task 1 models

deviations acroos the different measures was smaller then 0.05, which is to say that we selected all queries where all the models had a good agreement relative to the scores.

What we found was that all their averaged scores were incredibly low, which is not particularly surprising when examining the queries. Some of them were related to very specific contexts, like "Black Monday", which we would expect to be present in only very few documents. Some queries, like Legality of Medically Assisted Suicides had words that could be considered more complex than usual and which would probably have a poorer representation in models used.

One thing is common though. It seems that when all the models agree on the scorings it is not because they did a good job, but because they actually failed to retrieve relevant documents, which somewhat transposes the problem to the difficulty of resolving said queries given the tested corpus.

For reference, the retrieved queries are:

- Information Retrieval Systems
- McDonnell Douglas Contracts for Military Aircraft
- Efforts to Improve U.S. Schooling
- Find Innovative Companies
- Legality of Medically Assisted Suicides
- "Black Monday"
- Politically Motivated Civil Disturbances
- Criminal Actions Against Officers of Failed Financial Institutions

3 TASK 2

Task 2 is set up for us to have the possibility of using some Distributional Semantics methods, namely word2vec, Indexing by Latent Semantic Analysis (LSI), Latent Dirichlet Allocation (LDA) and doc2vec.

The main idea behind this kind of models comes from the assumption that it is possible to infer, at least up to some degree, a meaning of the words present in a text through their Distributional Semantics, i.e. their usage. The sense of meaning here is interpreted as similarity between words of the corpus. More specifically, it is assumed that words that are often seen in very similar contexts will, themselves, have very similar meanings, if not the same.

The interest for these models is based upon the recognition that the

people querying (a search engine for example) will possibly have limited knowledge on the searched topic. Being that the case, the possibility of naturally extending the query's terms to others that are semantically similar allows the ranking models to broaden their set of documents to rank, in such a way that it might provide a whole new set of possibly relevant documents. It is also interesting due to the ability to build more specific LMs, closely related to the topics of the documents.
For this assignment however, we were not required to infer these topics for the collection of documents used. Instead, we relied on the library gensim to infer those relations for us. Even the models used were themselves the ones implemented in the gensim library. Nonetheless, we did need to create representations for both the queries and the documents, and to calculate the similarity between them.
We now briefly explain the used models and what we had to do for each of them.

3.1 word2vec

Word2vec is the denomination given to a set of models that, not surprisingly, are able to represent words in vector form. Each word will have a corresponding N-dimensional vector and similar words will usually have similar vector representations. This allows a semantic identification of similarity between words by a simple cosine similarity calculation.
The way that this representations are fabricated is done, usually, using one of two methods, or even a combination of both, that extracts the vector's weights through its surrounding context.
The first one consists of the continuous-bag-of-words representation, which will try to predict a word based on a certain number of surrounding words. The way this prediction occurs is using a Neural Network (NN) with one hidden layer of size N, where the inputs are the surrounding words in enconding format and the output is the predicted word. During the training of these NNs the weight matrices continuously adapt their weights, and, somewhat surprisingly, in such a way that we obtain the previously described N-dimensional vectors. Seeing as the both the input and output will have size V, the size of the vocabulary, we have a matrix of size VxN (or NxV depending on the weight matrix picked) and that allows the association of a single N-dimensional vector to each of the V words, creating, thus, the word embeddings.
The second method is extremely similar. In fact, the only difference is that it tries to predict the surrounding words for a given word. It is, in a sense, the complement of the first method.

As it happens, and was previously stated, we did not need to implement the model, but had to choose its hyperparameters and train it on the given corpus in order to be able to later rank a list of document associated with a given query. The inputs of this task is a list of queries each one associated with the first 1000 documents that are more relevant given their TF-IDF score.

Regarding the choice of hyperparameters, we decided to concentrate on the ones that have the highest impact on the final performance of the model: dimensionality of the feature vectors, window size (the maximum distance between the current and predicted word within a sentence), word minimum frequency and number of total iterations through the dataset. Having fixed the window size to 5, minimum frequency to 3 and 25 iterations, we tested two different values for the feature vectors dimensionality, 300 and 150. The final value of the dimensionality was decided based on the scoring obtained by the re-ranking.

In order to re-rank the documents, first we had to transform each document (and query) into a single vector representation, in our case by averaging the projection of each token contained in the document (or query). This aims to output a representation that is supposed to convey the average meaning of the tokens, compressed into a relatively low dimensional space. Having pairs of (query-representation document-representation), a similarity score is computed, in our case the cosine distance. Having calculated each similarity score (2 - cosine distance) we re-ranked the whole document list associated with each query, and got an evaluation of the performance of the model.

size	ndcg@10	map@1000	precision@5	recall@1000
100	0.3918	0.1896	0.3950	0.6528
300	0.4197	0.2031	0.4400	0.6528

Looking at the above table, we can see how an increase in projection dimensions gained an overall increase in score values, settling the dimensions number to 300.

3.2 Doc2vec

This model is extremely similar to the word2vec model. In fact, in its original presentation, the only addition to the word2vec model was that the surrounding words would also be averaged/concatenated with a paragraph/document vector representation, taken from a matrix D where a column would be associated with one paragraph/document in much the same way that the words were in word2vec.

We chose to tune the same hyperparameters as the word2vec model, as they are similar and are mostly affected by the same choices of parameters. We chose a window size of 8, slightly bigger than the word2vec one, in order to account the fact that doc2vec uses surrounding words to compute the vector projections. The minimum word frequency has been set to 3, and the total iterations to 25. As with the word2vec model, we analyzed the scores of two versions of the same model while varying the number of projection dimensions, namely 150 and 300.

In order to be able to re-rank the input documents, we proceeded similarly as before for the query (tokenizing it and returning the average of the projections of each token) and for the document we simply obtained the internal doc2vec projection. Table TABLE NUMBER, shows the results of the two doc2vec re-rankings: we can see that the scores are very similar, and results are marginally higher (less than 0.01) for the model using 300 dimensions, apart from precision. This metric has an increase in score by 0.02 when using 150 dimensions, and this fixed the parameter value to 150.

size	ndcg@10	map@1000	precision@5	recall@1000
150	0.3211	0.1632	0.3450	0.6528
300	0.3235	0.1634	0.3200	0.6528

3.3 Indexing by Latent Semantic Analysis

Indexing by Latent Semantic Analysis, also know simply as LSI, tries to overcome the shortcomings of other models not accounting well for the fact that users usually want to retrieve documents related to the conceptual meaning of their queries, and not the precise meaning. With this in mind LSI attempts to build a conceptual latent space that allows to infer closely related concepts and allow for a wider range of retrieved documents. In the end, the granularity of this concept/topic groupings ends up being one of the most important factors of LSI, having significant impacts if properly determined.

Regarding hyperparameter tuning, this method is heavily influenced by the number of topics to be stored internally, so we concentrated only on the tuning of this value. Our choices were between a 200, a relatively big value that should enable the model to store more finely grained topics and supposedly perform better, and 30 a much smaller value that may increase the quality of the topics, bringing a supposedly better score in this experimental setting.

In order to produce a re-ranking of the document, we also had to compute a measure of similarity, but in this case no explicit vectors are given. For each document or query, the output of the lsi model consists of a list of scores for each topic, indicating which one has stronger associations with the given document. As the topics number is constant, we decided to transform this output into a 200 (or 30) dimensional vector, where each axis represents a topic. This led to a similar setting as the preceding methods, where the score is calculated between two vectors.

The below table shows clearly that the model with 30 topics performs better than the 200 topics one, indicating that having a lower and more general topics space is favorable in this case.

size	ndcg@10	map@1000	precision@5	recall@1000
30	0.1083	0.0718	0.1083	0.6528
200	0.1512	0.0913	0.1500	0.6528

3.4 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a probabilistic topic model. In this case the method tries to infer a probability distribution of topics over the entire provided corpus through a specified inference method and then creates a LM for each of the inferred topics. Thus it allows us to have much more concrete LM and to still have a generality through the latent topic concepts.

Practically speaking, this methods output is very similar to the Latent Semantic Indexing one, returning a list of topics associated with a document. As this method is very time consuming, no parameter tuning has been done, and the chosen value for the topics number is 30, with every other parameter left at its default value.

Contrary to lsi, this method doesn't output every pair topic-score, but this didn't pose an problem, as we considered a missing topic as having a value of zero in the resulting vector. The below table sows the resulting metrics associated with this method.

size	ndcg@10	map@1000	precision@5	recall@1000
30	0.1347	0.0982	0.1367	0.6528

3.5 Model Performance Evaluation and Comparison

Similarly to task one we realized a significance double tailed t-student test with an overall significance level $\alpha = 0.05$ to determine which of the models performed the best. Considering the Bonferroni α correction our individual t-test had a significance level of $\frac{0.05}{24} = 0.00208$, since we realized 24 hypothesis tests ($M \sum_{k=1}^{K-1} k = \frac{M}{2}(K-1)K$, where M is the number of measures, which is 3, and K is the number of models being compared, in this case 4).
The method is exactly the same, except that w eno longer consider the recall measure, since this will be exactly the same for all models, seeing as it consists of the first 1000 documents ranked by TF-IDF that were than re-ranked by the individual models. The obtained number of wins is the following:

Word2vec	LSI	LDA	Doc2vec
9	0	0	6

From the table we are able to conclude that the best performing one is the Word2vec model.

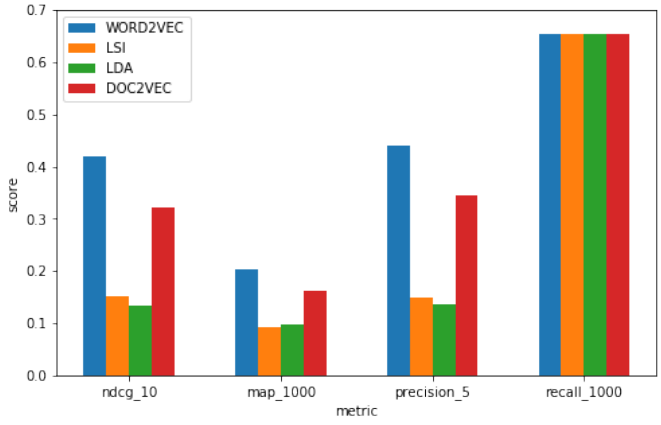

Figure 3: Scores comparison between the task 2 models

With the performance results, the task 2 models can be ordered as follows:

(1) **Word2Vec**
(2) **Doc2Vec**
(3) **LSI and LDA**

4 TASK 3

In this task we implemented the pointwise learning to rank method based on logistic regression using the library scikit-learn. Here we used the retrieval methods previously implemented as the features for the model and then performed a 10-fold cross validation training on the test set.

The training dataset is composed by a data matrix with a number of rows equal to the total number of documents ranked by the TF-IDF model, and columns equal to the number of used scoring models. Due to the long times needed to compute the plm score for each document, we decided to skip that calculation of this metric for the data matrix, resulting in 9 different scores instead of 10.

The target matrix is created based on the relevance labels given by the test set: for every query-document in the file, a label of 0 or 1 is given, indicating whether or not a document is relevant to a specific query. As not every pair query-document is present in the test set, we assigned a value of 0 to every document that was not included in the list. Having produced the entire dataset, we used k-fold splitting to create 10 different dataset to train our logistic regression model. For each validation run we registered the output, and the results are presented in the following table.

fold	ndcg@10	map@1000	precision@5	recall@1000
1	0.6714	0.3961	0.6667	0.7346
2	0.3020	0.1464	0.3067	0.3655
3	0.3765	0.1609	0.4154	0.5439
4	0.3521	0.2094	0.3692	0.6639
5	0.4513	0.1950	0.5231	0.6072
6	0.4609	0.3009	0.4500	0.8114
7	0.5889	0.1979	0.6462	0.5169
8	0.4983	0.3106	0.4923	0.7588
9	0.4138	0.1815	0.4333	0.6182
10	0.4523	0.2076	0.4154	0.5157

We then merged the different runs, and evaluated the results. The below table shows the various scores.

ndcg@10	map@1000	precision@5	recall@1000
0.4806	0.2446	0.4967	0.6528

4.1 Model Performance Evaluation and Comparison

We decided to compare all models, in order to determine which one was the best. Like in task 1 and 2 the comparison was verified with a significance level $\alpha = 0.05$ for a double tailed t-student test. Again, like in task 2, we did not consider the recall measure, since some methods would have exactly the same value. Considering the Bonferroni α correction our individual t-test had a significance level of $\frac{0.05}{165} = 0.00030$, since we realized 165 hypothesis tests ($M \sum_{k=1}^{K-1} k = \frac{M}{2}(K-1)K$, where M is the number of measures, which is 3, and K is the number of models being compared, in this case 11)

TF-IDF	BM25	JM	DP	AD	PLM-Gaussian
10	12	6	11	10	6

Word2vec	LSI	LDA	Doc2vec	LTR
9	0	0	6	19

We can clearly see that LTR outperforms all the models

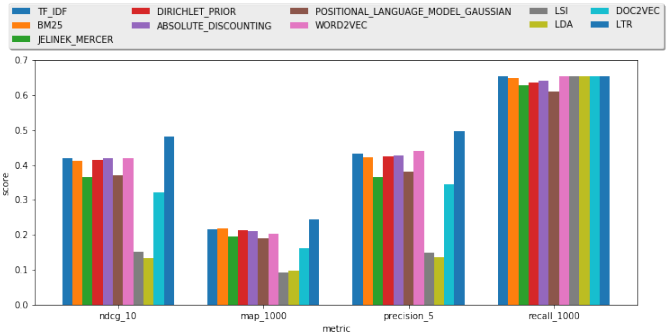

Figure 4: Scores comparison between all the models

Having the t-test results, we can order all models by performance:

(1) **LTR**
(2) **BM25**
(3) **Dirichlet Prior**
(4) **Absolute Discounting** and **TF-IDF**
(5) **Word2Vec**
(6) **Jelinek Mercer** and **Doc2Vec** and **Gaussian PLM**
(7) **LSI and LDA**

5 BRIEF REMARKS

In task one we were surprised that models like Jelinek-Mercer or PLM would perform not so well when compared to the other ones. Regarding the PLM we believe it might have to do with the σ parameters, which wasn't tuned at all. For the remaining ones we are not so sure, but also agree that the metric we used for comparison might not be the best one.
We also found it somewhat surprising that models like LSI and LDA performed so poorly when compared to others simpler models. We believe that this can have been caused by a very reduced parameter estimation phase.