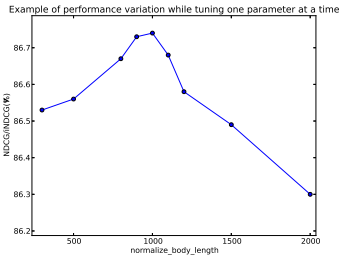
CS 276 Programming Assignment 3 Project Report

Jiaji Hu, Xuening Liu

Task 1: Cosine Similarity

After extensive tuning, we figured out that tuning parameters for cosine similarity does rather little to help with performance. However, we used this task to develop our methodology for tuning parameters.

First of all, we note that if we just tuned one parameter, with the others the same, there was often one peak value for that parameter, e.g. below:



It was usually the case that there is only one such peak value, and that we were able to find it by simple "hill climbing".

After we found the peak for one parameter, we could move on to the next parameter and find its peak value. By making a few iterations through all the parameters, we were able to more or less converge to a set a parameters that gave good performance.

The advantage to such a method is that it is simple, effective and easy to understand. It was often the case that we were able to converge to a local maximum in a few dozen tries. The results were also quite good, especially for our BM25F scorer, which ended up getting huge improvements.

The drawback of hill climbing is that it may get stuck in local optima. We observed that since there is an ordering of which parameters get tuned, the first ones tend to overfit the parameters yet to be tuned. When the time comes to tune the last parameters, little adjustment needs to be made, because the algorithm is already stuck at a local optimum. To combat this, we used randomization and repetition.

Finally, some discussion on the reasoning behind our final parameter weights for cosine similarity, and some attempts to explain the results:

First, the choice to use linear or sublinear scaling for term frequencies: Using sublinear scaling will greatly lower the impact of high frequency appearances of a query term in a field. Note that this would mostly affect the `body` and `anchor` fields. On a high level, this is a design choice of how much emphasis we want to put on the fact that a query term appears many times in the same field. In practice, linear scaling for both the document and query gave better performance on the cosine similarity scorer, so that was our choice.

Next, we discuss the weights for the five fields. Intuitively, higher weights should be given to fields that are good predictors of relevance. In practice, we realized that because we were not doing good normalization, we also needed to take into account the fact that some fields were more likely to gain higher counts of the query terms than others. The following is a list of the field weights and some explanation.

(a) `task1_W_url` : 3.1

The high weight for URL may be because query terms don't always appear in the URL, but when they do, they are great indicators of the document's relevance.

(b) `task1_W_title` : 5.0

As expected, `Title` got the highest weight. It follows intuition that having the query appear in the page title indicates high relevance.

(c) `task1_W_body` : 1.1

Note that the `body` field often contains a high number of occurrences of query terms. Since we are not using sublinear scaling, this has a even larger impact on the document's score. To prevent the influence from overshadowing input from the other fields, the weight for the body field cannot be too high.

(d) `task1_W_header` : 1.6

We note that there are other sets of high-performing parameters that have the weight for

this field to be higher than 1.6. We concluded that the header field is not such a high-impact field as `title`, due to its content being not as indicative.

(e) `task1_W_anchor` : 0.5

We were surprised to find such a low weight. It might be because the anchor field often has a high number of query term occurrences, so the weights need to be lower to normalize for that.

Lastly, we discuss body length smoothing. This directly affects the score, as more smoothing would mean less harsh normalization for longer documents compared to the shorter ones. From our experiments, the best value was found to be 500.

Finally, our performance for task 1:

Dataset	NDCG/iNDCG score
training	0.8671
dev	0.8462

Task 2: BM25F

Our approach for tuning is the same as in task 1, with one difference: In this task, we note that `W` and `B` are best tuned as a couple. This makes sense as `B` influences the field score and `W` decides its weight.

For task 2, the reasoning behind term frequency scaling is the same as task 1. However, in the end, we chose to use sublinear term frequency scaling for both the document and query, based on the results we got from parameter tuning. As for the parameter weights, they are discussed below:

(a) `task2_W_url` : 3.3, `task2_B_url` : 0.0

Same as in task 1, the URL is a good feature. For length normalization, we ended up with no normalization (`B`=0), meaning we don't care about how long the url is. URL are generally similar in length, and different segments serve different purposes, so there being no need to normalize it for length makes sense.

(b) `task2_W_title` : 5.2, `task2_B_title` : 0.2

Our most heavily weighted feature is still the title. This goes on to show that a documents relevance indeed has much to do with the title. Length normalization for titles follows similar logic for the URL, and the weight is 0.2, so there is slight normalization.

(c) `task2_W_body` : 0.9, `task2_B_body` : 0.8

Analysis for the body weight is same as task 1. Now that we finally have length normalization by field, we utilize it well and give a `B` weight of 0.8.

(d) `task2_W_header` : 2.85, `task2_B_header` : 0.5

The header is given a medium weight, and there is some length normalization with `B` = 0.5. Header lengths vary quite a bit so this makes sense.

(e) `task2_W_anchor` : 3.45, `task2_B_anchor` : 0.0

For the anchor field, using sublinear term frequency scaling (which made the vectors smaller in magnitude) positively affected the weight. Now, it can have a rather high weight, even with no length normalization.

(f) `K1` : 4.9

`K1` was found by tuning the parameters. It determines the speed of the saturation.

(g) $V_j = \frac{1}{\lambda' + \exp(-f_j \lambda'')}$, λ : 3.25, λ' : 0.05, λ'' : 0.1

For the function V_j, we tried out all three suggestions and picked the best performing one. The logarithm function was a close second. Both functions look both quite similar for pr ranging from 1 to 9. The parameter λ'' greatly affects the shape of the function and is quite delicate. λ and λ' can have a range of values which all give decent performance.

Lastly, our performance on the datasets:

Dataset	NDCG/iNDCG score
training	0.8883
dev	0.8689

Task 3: Smallest Window

In this task, we add a boosting function onto the BM25F scorer. It turned out that the best set of parameters for BM25F and the best set for smallest window was very similar. This makes sense because we still want the BM25 scorer to do its best – we are just adding a modification to its best effort score.

Therefore, we will not repeat our explanations for the weights that appeared in task 2. Instead, we will list them in the table below, and go on to focus on the boost function and the value of `B`.

Parameter	Value	Parameter	Value
task2_W_url	3.3	task2_B_url	0.0
task2_W_title	5.2	task2_B_title	0.2
task2_W_body	0.9	task2_B_body	0.8
task2_W_header	2.85	task2_B_header	0.5
task2_W_anchor	3.45	task2_B_anchor	0.0

Parameter	Value
tf_scaling	sublinear
K1	4.9
λ	3.25
λ'	0.05
λ''	0.1
B	1.12

Where V_j is the same as in task 2.

For the boosting function, we opted for a function that decreases very rapidly with the increase in window size in the end.

$$score = score \times [1 + (B - 1) \exp(query\_len - win\_size)]$$

For this function, we see that it gives a small boost for the case when the window size is the same as query length. After that, the boost diminishes very rapidly. Only giving a boost of 12% may seen small intuitively, but if we increase that value, performance takes a hit. We also tried a reciprocal function, with similar results if `B` is small.

Finally, our performance on the datasets:

Dataset	NDCG/iNDCG score
training	0.8900
dev	0.8695

Task 4: Extra Credit

For extra credit, we tried many ideas to add to the smallest window scorer.

(a) Number of fields where query term appears

We attempt to reward a document for diversity of where query terms appear. For every field that contains a query term a static reward of r is given. r is set to 0.05.

(b) Number of query terms missing

Similarly, we penalize the document for not containing a specific query term. For every term that does not appear anywhere in the document, we give a penalty of p. p is set to 0.15.

(c) Unevenness in frequency of query terms

We tried to expand the previous idea and suggest that all query terms should appear in the document as evenly as possible. Therefore, we calculate the percentage of each query term's occurrences in total occurrences, and penalize the document by the frequency that is furtherest from an even distribution.

In practice, we did not see improvement after using this feature. Therefore we did not give it a weight in our final model.

(d) Body length

An easy feature to try is the query-independent body length of the document. Using it did not give an improvement in performance, so we did not use it.

Using features (a) and (b) gave a further improvement in scorer performance, as show below:

Dataset	NDCG/iNDCG score
training	0.8908
dev	0.8718

Summary Analysis

Comparing the ranking functions, our implementation of cosine similarity can not do length normalization by field. Also, it does not take the page rank into account. As a result, the performance of BM25F is considerably better. To improve on BM25, we added the feature of window length, which is a good feature to determine document relevance to a query. After that, we tried adding some other features, with some succeeding and some failing to improve performance.

There are always other features that we can use to help with scoring. On one hand, we can work with what we have right now, using more features such as url length, number of anchors, etc, though it remains to be seen whether they are good features. On the other hand, we could alway gather more information from the document to use as features. Things such as the number of images, number of incoming/ outgoint links could be indicators of document quality, and similar to PageRank, may help with ranking the documents. However, as the number of features increase, the job of figuring out good weights will become much much harder, and hand-tuning weights may become quite impossible.