**Your Name: Jiachen Li**

**Your Andrew ID: jiachenl**

**Homework 5**

# Statement of Assurance

*I certify that all of the materials I submit are original works that were done by myself.*

# Experiment: Baselines

Provide information about the effectiveness of your system in three baseline configurations.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **BM25** | **Indri**  **BOW** | **Indri**  **SDM** |
| **P@10** | 0.4600 | 0.3280 | 0.4560 |
| **P@20** | 0.4160 | 0.3420 | 0.4480 |
| **P@30** | 0.4160 | 0.3573 | 0.4480 |
| **MAP** | 0.2417 | 0.1964 | 0.2391 |

The parameter settings for BM25 are  and . For Indri BOW,  and . For Indri SDM, the weight is set as 0.3 for original query, 0.5 for #NEAR/1 part and 0.2 for #WINDOW/8 part.

# Custom Features

Describe each of your custom features, including what information it uses and its computational complexity. Explain the intuitions behind your choices. This does not need to be a lengthy discussion, but you need to convince us that your features are reasonable hypotheses about what improves search accuracy, and not too computationally expensive to be practical.

**Custom Feature 1:**

The first feature I used is the Indri score of a new query generated by the n-gram match part in sequential dependency model (SDM). To be specific, I first used the body field of the original query and the #NEAR/1 operator to generate a new query. For example, if the original query is “Obama family tree”, then the generated query will be “#NEAR/1(Obama family) #NEAR/1(family tree)”. Next, I used the Indri retrieval model to get a score list for this query (Note that the Indri will add the default operator #and for the query). If one document is in the obtained score list, then I set the feature value as the score, otherwise I set the feature value as 0. The computational complexity for processing this feature is similar as processing a general query in Indri, except that it needs to process several #NEAR/1 operators at first, whose computational cost depends on how many tokens the original query has, and it won’t cost much as most queries are not very long (e.g., only have around 3~4 tokens).

The reason why I choose sequential dependency model is that I notice none of the 16 features takes advantage of the phrase information into consideration, which was shown to be very helpful in previous homework (HW3). Moreover, it can be seen from the part 1 baseline of this report, where the result of Indri with SDM are much better than the Indri with BOW. Besides, since we have already used the Indri score for the body field in feature 6, and the #WINDOW/8 operator in sequential dependency model is not as effective as #NEAR/1, so I only pick the #NEAR/1 operator in my feature and I think it is enough to help.

**Custom Feature 2:**

For the second feature, I apply vector space model and use Cosine similarity between query and the document’s title field as the feature value. To be specific, I used “lnc.ltc” to calculate this feature. Since we have already built the term vector in calculating feature 7, so here we can calculate the cosine similarity directly and do not need to bring any additional cost. The computational complexity for calculating cosine similarity is , where  is the term vector length of title field,  is the query length.

The main reason why I use cosine similarity is that I think the cosine similarity gives a different approach to measure the relevance between documents and queries compared with other features, so it may probably provide a different view on the relevance measurement, which I think may be helpful to improve the ranking. Besides, the reason why I use title field is that, on the one hand the vector space model has bias on document length, but the title field is usually much short than other fields, so the bias won’t be a big problem. On the other hand, I think the most important information for a document should be somehow reflected by the title, so title should be a reasonable choice. Moreover, we never tried this in previous homework, so I want to have an idea on whether and how well it works. In addition, the vector space model is flexible and easy to implement (i.e., we do not need to bring additional cost).

# Experiment: Learning to Rank

Use your learning-to-rank software to train four models that use different groups of features.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **IR**  **Fusion** | **Content-**  **Based** | **Base** | **All** |
| **P@10** | 0.4520 | 0.4640 | 0.4880 | 0.4920 |
| **P@20** | 0.4260 | 0.4280 | 0.4620 | 0.4660 |
| **P@30** | 0.4293 | 0.4280 | 0.4560 | 0.4613 |
| **MAP** | 0.2608 | 0.2642 | 0.2683 | 0.2723 |
| **Time** | 27.1 | 28.41s | 28.7s | 31.9s |

Discuss the trends that you observe; whether the learned retrieval models behaved as you expected; how the learned retrieval models compare to the baseline methods; and any other observations that you may have.

**Performance compared to baseline:**

First, it can be seen clearly that all the results of the four models trained by learning-to-rank perform better than the baseline (Note that I think we should mainly compare with the results of BM25, since the re-ranked results are based on the top 100 retrieved documents from BM25), which indicates that the trained models do learn something to improve the ranking from the features and this is exactly what I expect.

**Performance among different models:**

Besides, compare the results among different models (i.e., different group of features), we can see that the overall performance (i.e., MAP) improves as more features are used in the model (i.e., MAP(IR Fusion) < MAP(Content Based) < MAP(Base) < MAP(All)). Also, let’s have a close look on P@k values, we may find that the results of Content Based model are only slight better than the results of IR Fusion model (e.g., P@20 0.4280 vs 0.4260), while the results of the Base model are much better than that of Content Based model (e.g., P@20 0.4620 vs 0.4280), which suggests that the term overlap features may not be very important while features like spam score, url depth and page rank can improve the results well.

**Performance of custom features:**

Moreover, for the model with all the features, I’m glad that it beats the Base model on the overall performance, which indicates that my two custom features work when re-ranking the documents, so that my design should be reasonable. Next, in order to have a clear mind on the performance of each custom feature, I did the following two experiments:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Base** | **Base + SDM** | **Base + Cosine** | **All** |
| **P@10** | 0.4880 | 0.5040 | 0.4840 | 0.4920 |
| **P@20** | 0.4620 | 0.4800 | 0.4580 | 0.4660 |
| **P@30** | 0.4560 | 0.4573 | 0.4520 | 0.4613 |
| **MAP** | 0.2683 | 0.2734 | 0.2699 | 0.2723 |

It is seen from the table that both custom features can improve the final results, especially the SDM one, whose overall performance is even better than using all the features. This result is indeed what I expect and it is also consistent with the conclusion in previous homework that SDM can significantly improve the searching performance. Comparing the results of Base with Base + SDM, we can see that Base + SDM beats Base in all metrics, which indicates that the phrase information is very useful in ranking. For the Base + Cosine, only the MAP is slightly better than Base, which suggests that the cosine similarity feature did rank some relevance documents higher but it also makes some mistakes (e.g., make some higher ranking documents to be lower). So it is obvious that Cosine feature is not as strong as SDM. Taking the individual performance of these two custom features into consideration, it is then acceptable for us that the performance of All features is in the middle of Base + SDM and Base + Cosine. Nevertheless, the results of All features model did the best on the P@30 metric, which suggests that these two custom features together ranked more relevance documents into top 30.

**Running time:**

Since I’m using Approach 1 in my implementation (the overall running time is longer), so there is a chance to compare the running time for each model. There is no doubt that the more features you used, the more time it takes. So, there should be a trade off on the effectiveness of the feature you chose and the computational cost that feature brings. For example, comparing the results of IR Fusion and Content-based model, the MAP improves about 0.034, but it cost another 1.3s. While comparing the result of Content-based and Base model, the MAP improves 0.041 and the time cost is only0.3s.

Also, discuss the effectiveness of your custom features. This should be a separate discussion, and it should be more insightful than “They improved P@10 by 5%”. Discuss the effect on your retrieval experiments, and if there is variation in the metrics that are affected (e.g., P@k, MAP), how those variations compared to your expectations.

# Experiment: Features

Experiment with four different combinations of features.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **All**  **(Baseline)** | **Comb1** | **Comb2** | **Comb3** | **Comb4** |
| **P@10** | 0.4920 | 0.4760 | 0.5120 | 0.5040 | 0. |
| **P@20** | 0.4660 | 0.4760 | 0.4640 | 0.4660 | 0. |
| **P@30** | 0.4613 | 0.4627 | 0.4627 | 0.4613 | 0. |
| **MAP** | 0.2723 | 0.2722 | 0.2709 | 0.2718 | 0. |

My general strategy for feature choosing:

1. In the last section, we have seen that feature 18 (Cosine similarity) doesn’t work well, so I first disable it.
2. According to the baseline that Indri has a bad performance, so I will consider disabling the Indri features;
3. Since feature 1-4 seems to be independent of other features, so I would like to do experiments for disabling each of them;
4. Do experiments for disabling features in each field to identify which field is more important;
5. Consider the SVM weights obtained from next section.

My feature combination

Comb1: 1,2,3,4,5,7,8,10,11,13,16,17 (disabled: 6,9,12,14,15,18)

Comb2: 1,5,7,8,10,11,13,16,17 (disabled: 2,3,4,6,9,12,14,15,18)

Comb3: 1,5,6,7,10,11,13,17 (disabled: 2,3,4,8,9,12,14,15,16,18)

Comb4: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17 (disabled: 2,4,9,11,12,14,15,16,18)

Describe each of your feature combinations, including its computational complexity. Explain the intuitions behind your choices. This does not need to be a lengthy discussion, but you need to convince us that your combinations are investigating interesting hypotheses about what delivers good search accuracy. Were you able to get good effectiveness from a smaller set of features, or is the best result obtained by using all of the features? Why?

# Analysis

Examine the model files produced by SVMrank. Discuss which features appear to be more useful and which features appear to be less useful. Support your observations with evidence from your experiments. Keep in mind that some of the features are highly correlated, which may affect the weights that were learned for those features.

Some of this discussion may overlap with your discussion of your experiments. However, in this section we are primarily interested in what information, if anything, you can get from the SVMrank model files.

The SVM parameters for all the 18 features are shown as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1:  0.501899 | 2:  -0.12852466 | 3:  0.27083552 | 4:  0.023081422 | 5:  0.50871187 | 6:  0.27809715 |
| 7:  0.27935007 | 8:  0.20148262 | 9:-0.038343862 | 10:  0.28761262 | 11:  0.14255269 | 12:  0.091915041 |
| 13:  0.26543868 | 14:  -0.004562771 | 15:  -0.05141204 | 16:  0.073402569 | 17:  0.44743443 | 18:  0.17425808 |

Since in linear SVM, the decision is based on the summation of the products between weights and feature values. Then the value of the weights can indicate the importance of different features. Since we have normalized all the features into the interval [0,1] (I mean we don’t have negative feature value), then the negative weights indicates that these features are not consistent with others, so it is very likely they are not