Optimizing Document Scoring for Query Retrieval

Brent Ellwein
baellwe@cs.stanford.edu

Abstract

The goal of this project was to automate the process of tuning a document query engine. Specifically, I used machine learning to find an optimal scoring function given term frequency, document frequency, and query matching. The accuracy of the results was measured as the greatest average precision of the top ten scored documents.

Background

This project considered 3 statistics when scoring a document's relevance to a given query. These statistics are common throughout Information Retrieval:

- **Term Frequency** (*tf*)
  a measure of the number of times that a given token i appears in a document
- **Inverse Document Frequency** (*idf*)
  a measure of the inverse of the number of documents that the token i appears in
- **Overlap** (*coord*)
  a measure of the number of query terms in the document

For a typical information retrieval system, one would not use the raw number, but would instead use a scaling function of the statistic. One might use square roots or logs to perform the scaling. The total score is reported as a combination of the individual terms. However, it is not generally known what the best scoring function or combination of functions is. In fact, the best scoring function may be different depending on the domain of the corpus and any artifacts of the content[1].

Experimental Conditions

For training, testing, and the collection of results, I used the Cranfield collection. This corpus consists of approximately 1400 abstracts and 226 queries all related to aeronautical engineering. Each document is scored relevant or not relevant for each of the queries.

---

[1] For example, when trying to avoid spam documents on the internet, one might want to severely dampen, or even limit the tf score.

Scoring is calculated as the scaled product of tf, idf, and coord. After computing the score for each document, the top 10 most relevant documents are compared against the known relevancy values. The precision is the ratio of relevant documents to total number of documents returned (10). These precision scores are then averaged and the result is the overall precision of the engine on the queries. The objective is to maximize this overall precision.

Learning Environment

In order to find the optimal combination of scaling functions, I used the following parameterization function weights. Each of the parameters θ_i are optimized by the learning algorithm.

- **tf**   (as a function of frequency f)

$$\theta_0 + \theta_1 f + \theta_2 \log(f) + \theta_3 \sqrt{f} + \theta_4 e^f$$

- **idf**   (as a function of document frequency df and number of documents n)

$$\theta_5 + \theta_6 n/(df+1) + \theta_7 \log(n/(df+1)) + \theta_8 \sqrt{n/(df+1)} + \theta_9 df$$

- **coord** (as a function of term matches m and number of query terms q)

$$\theta_{10} + \theta_{11} m/q + \theta_{12} \log(m/q) + \theta_{13} \sqrt{m/q} + \theta_{14} m$$

- **total score** (product of the parameters)

$$\theta_{15} tf * \theta_{16} idf * \theta_{17} coord$$

Each parameter was optimized according the K-fold cross validation for various values of K. It seems obvious that the parameters θ_{15}, θ_{16}, and θ_{17} are redundant (that is to say their values could in effect be combined with the parameters from the groups they correspond to), but early experiments showed that they improved the convergence rate. Later, as normalized Coordinate Ascent was implemented, their purpose was decoupled from the parameters inside the groups and they became scaling factors for each group as a whole.

The default parameters for the query engine, which are typical default parameters used throughout the field of Information Retrieval, are:

$$\sqrt{tf} * (\log(n/(df+1)) + 1) * m/q$$

In other words, $\theta_0 = \theta_3 = \theta_5 = \theta_7 = \theta_{11} = \theta_{15} = \theta_{16} = \theta_{17} = 1$. All others would be set to zero.

Bounded Coordinate Ascent

The first algorithm I attempted to optimize the parameters with was a bounded coordinate ascent model. It is important to note that each training iteration started with random values for each of the parameters. I discovered that if I initialized the values to the default values, that the algorithm was unable to learn from the local optimum that the values represent. In this model, each parameter is optimized individually according to the Newton-Raphson method. At any given optimization step, the values each theta is allowed to take are strictly bounded by -10 and 10. The model was initialized with random values and optimization was repeated until convergence. The results of this optimization were highly unsatisfactory compared against the default configuration. The following charts summarize the precision results for a couple trial of k-fold validation. For these charts, "training" refers to the queries used as the training set and "test" refers to the queries held out of training and used for validation. Default values refer to the default configuration mentioned above. Learned refers to the optimized parameters.

Average Precision (K=10)

Average Precision (K=20)
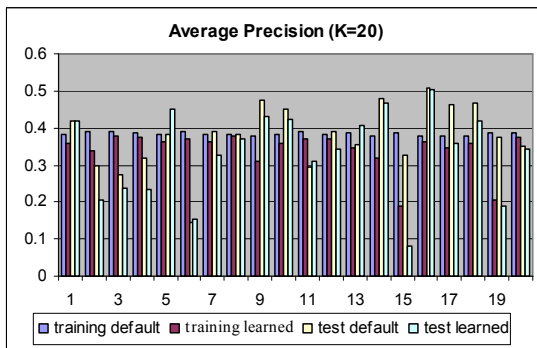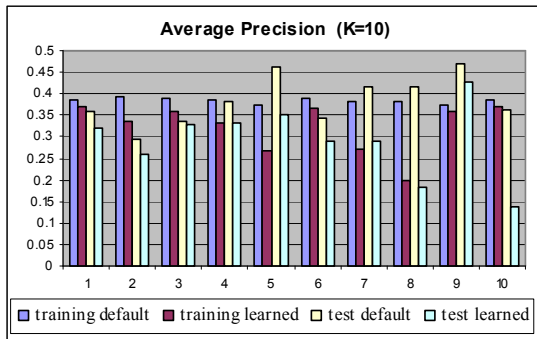
It is easy to see from these graphs that the optimized classifier generally performs worse than the default classifier for both training and test queries. The optimized classifier more closely matches, and even out-performs the default classifier when K is large which suggests that a large number of training examples is necessary in order to properly optimize the parameters. Further

experiments show that even when using the entire dataset, the optimized classifier will typically under perform the default classifier. An explanation for the negative results is that training yielded a high level of variation in the parameters across training datasets. This indicates the presence of many local optima that the algorithm was unable to recover from. Below is a table which summarizes the fluctuations in the values for each theta.

Standard Deviation for each theta
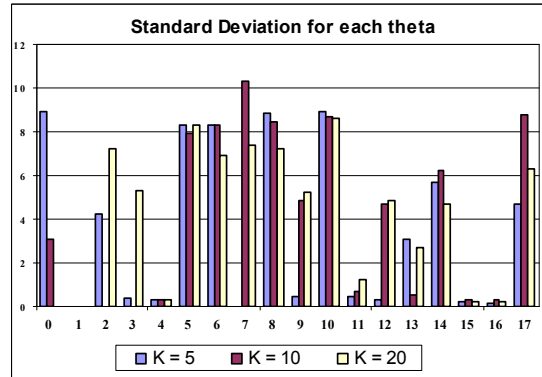
As can clearly be shown in the chart, nearly each value for theta is fluctuating violently across each of K trials of K-fold cross validation. The high degree of variation led me to believe that the parameters were not converging to optima close to the global maximum, but were too highly influence by their initial random values.

Normalized Coordinate Ascent

The next logical step in order to increase the precision of the optimized query engine was to add restrictions on the values each parameter could hold. As before, the values had to be initialized at random. First, I tried restricting the value to be bounded by 0 and 10, but that did not provide noticeable improvement. Then, I added the restriction that, after each optimization, all the values be re-normalized across categories so that they would sum to 1. Naturally, all the values would then be restricted to the range 0-1 after optimization, but the Newton-Raphson steps would still be allowed to explore beyond the 1 boundary. While this yielded slightly better results, the results were still on average only equal to the default classifier. Going another step further, I restricted the parameters to sum to one only within specific groupings. The groupings are enumerated as follows:

- **tf group**

$$\sum_{i=0}^{5} \theta_i = 1$$

- **idf group**

$$\sum_{i=6}^{10} \theta_i = 1$$

- **coord group**

$$\sum_{i=10}^{14} \theta_i = 1$$
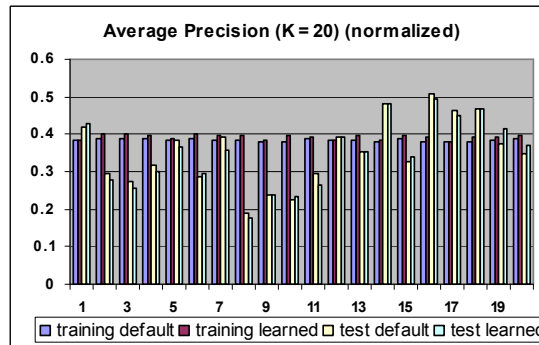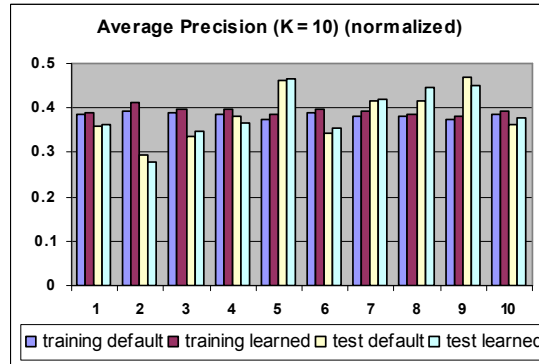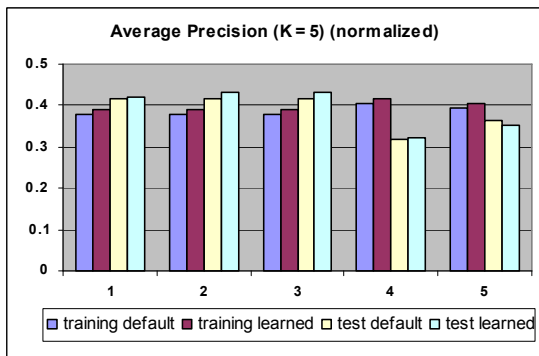
- **total score group**

$$\sum_{i=15}^{17} \theta_i = 1$$

I ran the training and testing examples again and recorded that the fluctuations in the parameters theta had noticeably reduced. While it is true that the forced scaling is cause for some scaled reduction, another thing to notice is that several parameters almost never change. The following chart, as before, summarizes the results.

Standard Deviation for theta (normalized)
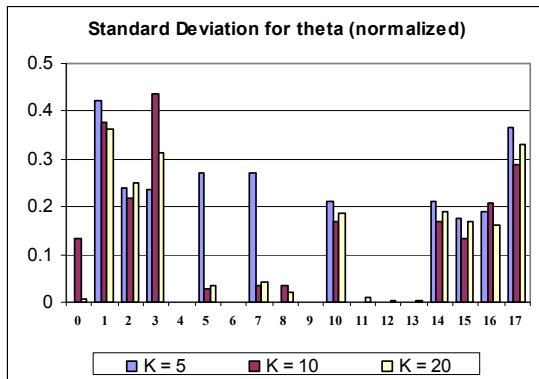
While there is some variation in the data, this can be explained by looking at the raw values for the parameters (which are too numerous to include in this report). I will provide an in-depth discussion for each parameter which will provide some reasoning for the behavior of its optimized value.

Having succeeded in reducing the variance of the optimization, we can now look more closely at the trial results for k-fold cross validation. As before, the following precision charts summarize the behavior for each of K = 5,10,20.

Average Precision (K = 5) (normalized)

Average Precision (K = 10) (normalized)

Average Precision (K = 20) (normalized)

Notice that the precision is consistently better for the learned parameters over the default settings. This generally applies both to the training data (as you would expect) and to the testing data as well! One interesting item to notice is that in comparing K=10 to K=20, we find a higher percentage of the cross-validation precisions to be less than the default for the case of K=20. This suggests that the learning algorithm is overtraining on the training data and that the algorithm has too much variance and not enough bias. Looking for trends in the data, I will suggest why this might be the so.

Parameter optimization and trends

The learning algorithm partitions the parameters into 4 groups by their normalization. In this discussion, it is convenient to think of the parameters in these natural groupings.

- **Total Score Group**
  Parameters θ_{15-17}, behave in one of 2 distinct ways. Either θ_{17} is dominate (i.e. has a value between 0.9 and 1.0), or the weight is roughly evenly distributed among the 3 parameters. For the case where θ_{17} is dominate, the difference in test precisions between default and optimized parameters tend to be larger than when the weight is more evenly distributed. Even though the parameters are multiplied, and thus can be treated as a single entity, arranging them in this fashion allowed the learning algorithm to

"retrain" this parameter multiple times. Experiment tests show that the learning algorithm performs less effectively if the parameters are collapsed into a single entity.

- **TF Group**
  For each trial the optimization of both θ_0 and θ_4 was essentially 0. Effectively, the algorithm has provided experimental evidence to suggest that one would not want to use an exponential or constant term frequency. The other parameters in this group θ_{1-3} were given approximately equal weights and grouped with apparent randomness. That is to say, for one trial, θ_1 and θ_2 might each have 1/2 while θ_3 is 0 while on another trial θ_1, θ_2, and θ_3 might each have 1/3. On other trials, either θ_1, θ_2 or θ_3 might be 1 while the other 2 zero. Because there is not a strong correlation between any combination and a more precise result, the data suggests that any of the scaling functions (liner, logarithmic or root) will have an equal benefit on the overall query precision.

- **IDF Group**
  The parameter θ_7 dominated this group. As is evidenced by the low deviations for each element in this group (ignoring the noisy case of K=5, which is not enough training samples), the values for each theta were essentially the same for each set of training data. While the other values, θ_5, θ_6, θ_8, θ_9 were each zero, or very near zero, θ_7 was consistently at or very near 1. So, the experimental data suggests that a good classifier will use $\log(n/(df+1))$ as a scoring function for idf.[2]

- **COORD Group**
  As evidenced by the low deviations for these parameters, they nearly always optimized to the same value. In the case of θ_{11}, θ_{12}, θ_{13}, that value was always zero. The values of θ_{10}, and θ_{14} were coupled. Either θ_{14} was one and θ_{10} zero, or they both were a half. There is a high correlation (about 66%) of the case where $\theta_{10} \approx \theta_{14} \approx 0.5$ and the test results being lower for the optimized parameters than the default parameters. So, it would seem that it is more advantageous (higher bias) to have θ_{14} equal 1 and the others set to zero. Functionally, this corresponds to only taking into consideration the number of matches between a query and a document and ignoring the size of the document altogether.

---

[2] Recall that n is the number of documents in the corpus while df is the number of documents which contain the term of interest.

Benchmarking

During the discussion of the tests, I have shown that the optimization process can produce gains in average precision over the top ten results. Now, I will move the discussion to include some other tests to see if the optimized query parameters are really better, or if the gains in precision came at too great a cost.

- **Precision - Recall**
  One metric common in IR is to plot a precision-recall curve. Comparing both the default and optimized parameters, the following precision-recall curves are produced:
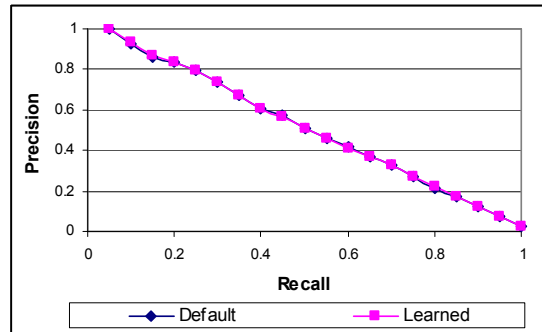
It is easy to tell from the graph that there is not a noticeable difference between the two classifiers when averaged over all the queries. Generally, the learned classifier has a slightly higher precision rate as evidenced by the plot below. For the plot, positive values indicate that the learned classifier performed better, i.e. the differences as calculated as learned - default.
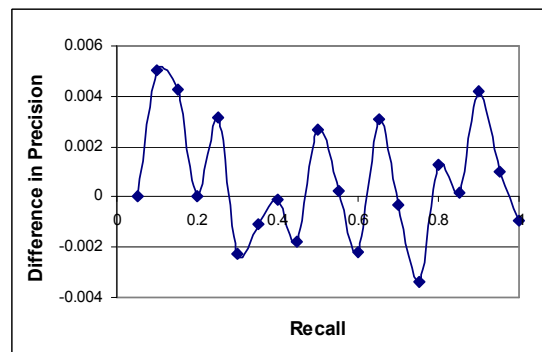
Clearly, by measuring only the rate of precision at a given recall level, there is not a significant difference between the learned and default classifier. While this result may not be terribly impressive, it does show that the learned query parameters have a similar bias and variance as the default query parameters.

Now, having shown that the learned parameters are on average no worse than the default parameters, lets take a closer look at precision in the top 10.

Since we were trying to optimize the precision in the top ten documents, we would expect to see some performance increases for the top ten results. The queries can be grouped into three groupings: those where the precision was the same, those where the learned performed better and those where the default performed better.

- **Default was better (18 queries)**
  Typically, the difference between default and learned was only one, in fact the of the 18 queries in this group, only one query had a difference less than -1. Many of the losses were from documents moving from the 9th or 10th rank to a slightly lower rank. Only two queries which had relevant results in the top ten for the default had none for the learned parameters.
- **Learned was better (42 queries)**
  As was the case when the default was better, the difference in this group was typically only 1. However, there were two cases where the precision was improved by 2, and 1 case where the precision was improved by 4. Moreover, there were two cases where the number of relevant documents increased from 0 to 1.
- **Precision was the same (166 queries)**
  For most of the queries, the precision in the top ten was the same. The only differences in this group were that the ranking of individual documents was somewhat re-ordered. In rare cases, a relevant document was replaced with one whose original rank was greater than 10.

Learning Rate

As mentioned previously, the default values represent a stubborn maximum such that it is hard to explore beyond it. To get around this 'feature,' I was forced to initialize the parameters for each theta randomly for each training trial. Fortunately, the learning algorithm typically converged after only 5 or 6 passes through that data. Below is a chart which summarizes a learning curve from one of the examples.
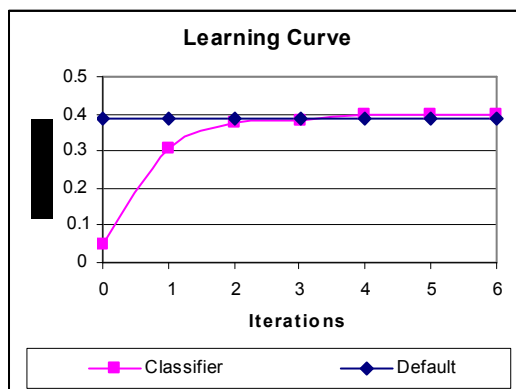
Conclusions

Using the optimizations found by the learning algorithm, one can expect to see more precise results in the top ten queries about 20% of the time, while experiencing less precise results about 8% of the time. This net gain indicates that the learned parameters provide some gain for the top ten results, but as evidenced by the precision-recall figure, these gains do not accumulate over time.

Beyond simply the gain in precision, there are several other results which are worth noting. First, the algorithm was successful in demonstrating that some of the proposed features are bad candidates and should not be used in an information retrieval system. One surprising result was that the algorithm was unable to consistently select parameters for the TF group. The data suggests that using linear, logarithmic or root scaling of the term frequency will have basically the same effect. I had expected that the linear parameter would be optimized to a very low value. Another surprising experimental result was that the default values represented a stubborn local maximum, and if I initialized using these values, the learning algorithm could not recover from this maximum.

Acknowledgements: