CS 276 Programming Assignment 4 Project Report

Jiaji Hu, Xuening Liu

Task 1: Pointwise Approach and Linear Regression

In this part, we use linear regression to give a score to each query-document pair. The query-document pairs are represented by a five-dimensional vector of tf-idf scores.

Since we cannot start adding new features at this point, and there are no parameters to tune for linear regression, there is not much for us to design ourselves in this task. Here, we focus on two design choices:

Firstly, we determined whether or not to use sublinear scaling for term frequencies. By using sublinear scaling, we design the system so that high frequencies of a single term have a smaller impact on the score. Conceptually this makes sense, because having a query term appear 10 times is probably better than having it appear 1 time, but it probably shouldn't be 10 times as good. With sublinear scaling, our linear regression model achieved a higher score, so we used sublinear scaling.

Secondly, we tried applying length normalization to each document. The idea is that since longer documents are more likely to have more query terms appear, we should penalize them for their length. Furthermore, we can smooth the document lengths for better performance. It turned out that document length normalization did not help performance on this task. However, length normalization did help in the following tasks, as we will discuss later.

Finally, our performance for task 1:

Dataset	NDCG/iNDCG score
training	0.8749
dev	0.8429

Task 2: Pairwise Approach and Ranking SVM

The pairwise approach is totally different from the pointwise approach, in that we do not directly try to predict a relevance score for a query-document pair, but train our classifier to compare the relevance of different query-document pairs.

Similar to task 1, sublinear scaling of term frequencies and document length normalization are two things we can do to help the performance of the ranking system. In this task, both sublinear scaling and length normalization gave the NDCG score a considerable boost, so we do both these operations in our final system.

Standardization for the input for the SVM is necessary for fast runtime and good performance. In the assignment guidelines, we were instructed to do standardization before we took the difference of vectors. Also, we were instructed to do standardization independently on the training and test set. We felt that these might not be the correct practice, but rather shortcuts to make the assignment simpler. Therefore, we implemented both versions of standardization to see if they lead to any difference in performance.

Firstly, since the actual vectors that get sent into the SVM for training are the vectors after we take the difference, it makes more sense to do standardization on that dataset. Secondly, it is a good idea to do the same preprocessing to training and test data, so we should not do separate standardization for training and test. In our second implementation, we implemented our own standardization method that remembers the mean and standard deviation of the training data, and apply the same filter to the test data as the training data.

In practice, both the standardization recommended in the guidelines and our own standardization give good results. It makes sense that the shortcuts work. For the first shortcut, we know that we don't need the input data to exactly have zero mean and unit variance for SVM to work well. Standardization before taking the difference gives a distribution that is good enough. For the second shortcut, we rely on the fact that the training and test data have very similar distributions, so doing standardization separately yields very similar results as doing the same transformation on the two datasets.

To tune the SVM parameters, we used grid search on parameter `C` for linear kernel SVM, and parameters C and γ for RBF SVM. We searched on the following range: $C = \{2^{-3}, 2^{-2}, \ldots, 2^{3}\}$, and $\gamma = \{2^{-7}, 2^{-6}, \ldots, 2^{-1}\}$. We noticed that with large C and large γ, the training time goes up considerably, especially because many iterations are needed for the algorithm to terminate. Also, the performance of the linear SVM does not vary a lot with C, and the linear SVM performs very well even under default parameters. For the RBF SVM, we got some good performance with $C = \{2^2, 2^3\}$ and $\gamma = \{2^{-4}, 2^{-5}\}$, though there is a lot of variance in the results.

Comparing the linear and RBF kernel SVMs, both give good performance. The RBF SVM got a slightly higher top NDCG score after grid search, but not by much. On the other hand, the linear SVM is handy because it is generally faster to train and is not as sensitive to parameter adjustment.

Finally, our performance for task 2 is shown in the following table:

Dataset	Linear SVM	RBF SVM
training	0.8627	0.8637
dev	0.8472	0.8479

Task 3: More Features and Error Analysis

In this task, we add extra features to the model in the previous task in order to achieve better results.

First, we tested out our classifier with one extra feature, either the BM25F value, the PageRank score, or the smallest window score. We first computed these three values with our best weights and best choices of functions from PA3. Then, we tried using the weights from the trained linear SVM model from task 2, which gave our score a huge boost of around 0.03. We also tried out other functions for PageRank and smallest window, but they were not as good. On its own, the BM25F score is the most helpful (+0.107), followed by PageRank (+0.054), while smallest window helped just a bit (+0.018).

Then, we tried combinations of two extra features. BM25F in combination with PageRank was by far the most helpful, achieving a score of 0.8735, while the other two combinations were around 0.857. Surprisingly, adding smallest window to BM25F was actually worse than having BM25F alone, though PageRank and smallest window combined got a bigger boost than the combined boost from using the two on their own. Finally, using all three extra features, we got 0.8742 — only a small rise from BM25F + PageRank.

From all this, it seems that the smallest window score is least useful, while BM25F is best, and PageRank is also good. This could mean that we just didn't come up with a good function to utilize the smallest window score.

Next, we take a look at the ranking output of our system and do some error analysis. Below are a few examples that lead us to design some new features:

For the query `climbing wall hours`, our system ranked `http://www.stanford.edu/~clint/index/` fourth in ten documents. With a relevance score of 0.0, it should have been last.

We took a look at the page, and discovered that it was a personal page of Clint Cummins, about a book he wrote on rock climbing. We realized that the site's url gave away the fact that it was a personal homepage (and therefore less likely to be relevant) because it contained `~clint`.

We immediately got the idea to add a binary feature to detect the "~" character in urls. After adding the feature, we tested out the system. To our surprise, the performance stayed the same — the feature didn't work. It took some more analysis for it to dawn on us that having a "~" is not necessarily a bad sign — many queries actually ask for personal homepages for professors, such as the query `jeffery ullman`, which has the url `http://infolab.stanford.edu/~ullman/` at relevance 3.0. This meant that detecting the "~" was not enough — we needed to detect if the character appears in a url while the query does not ask for that person. With this new insight, we came up with an improved feature that is still binary, but is only true if (1), a tilde appears and (2), no query terms appear in the url. With this feature, we improved our system performance by 0.01, which is a good increase. For the specific error case, `http://www.stanford.edu/~clint/index/` dropped a couple places, which was encouraging.

Another error we noticed was for the query `andrew ng computer science stanford`, where our system ranked a link to a New York Times story in the top 3. The story was indeed about Andrew Ng and is a good read, but would probably not fulfill the information need of the person with that query. We noticed that the document url was very long, and contained a lot of characters such as "&", "?" and "=".

Therefore, we came up with two lines of attack:

(1) Add features related to url length. (2) Add features that detect special characters. For url length, we tried number of characters, number of words and number of segments (seperated by slashes). We also tried doing log-scaling for these features. In the end, we came up with the feature $\log(num\_words - 4.5)$, which seemed to work best. With the new url length feature, we observed a slightly lower ranking for docs with really long urls (the log scaling meant the feature was only prominent for exceedingly wordy urls), and an overall performance boost.

For special characters, we tried adding binary features that detected their existence in the url, but they didn't really help. That is understandable because in truth, characters such as "&", "?" and "=" appear quite a lot in urls, and their existence may not be a good indicator of relevance.

Finally, we made further observations on the training and test data and felt that the file type could be a good feature to add. (It seemed to be the case that many PDF files got high scores and PPT files had low relevance scores — possibly because the person with the query is usually not looking for a PPT file when typing his query.) After adding three feature to detect PDF, PPT and TXT files, our ranking performance went up by 0.025, a considerable increase. The feature that detected PDF suffixes had a larger impact, partly because there were a lot more PDF files in the results than PPT and TXT files.

Apart from that, we also imported our extra features from PA3. Some of them worked well, namely "number of fields where query terms appear" and "number of query terms that are missing from document". It makes sense that a document that contains all the query words is probably more relevant than one that only contains a subset of the query words. We also added the log normalized document body length as a feature, which helped a bit. Our theories for this is that the length of a document is a query-independent indicator of its quality, and higher quality documents generally get better scores.

Finally, we did a grid search for parameters. Similar to task 2, the linear kernel SVM performs well for a large range of C values, while RBF SVM has a sweet spot of approximately $C \in [2^2, 2^3]$ and $\gamma \in [2^{-4}, 2^{-5}]$. Peak performance for both are quite similar. With all features in effect, RBF SVM got the highest dev score of 0.8803 with grid search, but linear SVM consistently gets around 0.8795 for a large range of parameters. The training score for both was around 0.8905, so overfitting was not a huge issue for good parameter choices.

Finally, our performance for task 3 (numbers are from linear SVM, but RBF SVM is very similar):

Features	Train	Dev
base	0.8637	0.8479
base+bm25	0.8802	0.8586
base+PR	0.8688	0.8533
base+win	0.8668	0.8497
base+bm25+PR	0.8885	0.8735
base+bm25+win	0.8824	0.8580
base+PR+win	0.8722	0.8568
base+bm25+PR+win	0.8900	0.8742
8 extra features	0.8905	0.8799
after grid search (RBF)	0.8906	0.8803

Task 4: Extra Credit

For extra credit, we experimented with using SVM regression for the pointwise approach. Using the new features, our regression SVM achieves a score of 0.8675, which is a large improvement on task 2. Also, our regression SVM performs better than a linear regression model using the same features.

Our performance for task 4:

Dataset	NDCG/iNDCG score
training	0.8902
dev	0.8675

Interestingly, some of the good features we found for task 3 did not work out for the Regression SVM. We had to take out features such as log body length, number of fields with query terms, and the detector for "∼" in the url. The reason could be that we are doing a totally different task in the pointwise approach, so good features for the pairwise approach may not be as good in the pointwise case — to think of it, the actual features we feed into the SVM classifier in the pairwise approach are not the feature values themselves, but rather the difference of the feature values between the two documents.

On the other hand, for the BM25F feature, we switched to using the weights we got from the linear regression classifier in task 1. This improved performance by 0.025. The switch makes senses because we are moving back to the pointwise approach, and weights learned in that setting might be better than weights learned in the pairwise setting, where the objective is totally different.