# Daniel Ribeiro Silva

# drsilva

# Homework 6

**0. Statement of Assurance**

I certify that all of the material that I submit is original work that was done only by me.

# 1. Performance evaluation (60%)

## 1.1 P@10, NDCG@10 and MAP (10%)

Write down the formula of each metric, and give one example for each of them to show that single metric is not enough to guarantee the ranking quality.

Let’s define as the set of all relevant documents for a query and as the set of all documents. Let be the ranking of a given methodology and the document of that ranking.

So is the number of relevant documents in the ranking of the top-10 documents selected. Let’s consider the example below:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ranking | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Relevant? | Y | Y | Y | N | N | N | N | N | N | N |

This example would have a of . One problem about measuring performance with is that the order is not considered. Let’s compare the previous result with the following result:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ranking | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Relevant? | N | N | N | N | N | N | N | Y | Y | Y |

This result will also have a of but this is clearly a much worse result than the first one, since on the first example all relevant documents appear at the top of the list, while in the second example they appear at the bottom of the top 10.

MAP uses the previous concept to make an “ongoing average” of the score along the ranking and for each query. Let’s first define the average precision AP:

Where is the unitary function, defined as below:

is simply the mean of the for each query in the experiment.

MAP tries to improve by taking the order into account. If we use the same examples as above, the firs one would now have an AP of

while the second one would have an AP of

Nevertheless, AP and MAP don’t take into account the fact that the farther an element is on the ranking, the less relevant it is, and the less likely it is to be even looked by the user. That is where NDCG comes in. It gives higher importance to documents at the top of the list. Besides, it also allows a user to give different levels of relevance to a document (instead of being just binary as for P@K or MAP). In our case, we only use binary relevancy.

To define NDCG, we must first define DCG

where is the relevancy level of (the i-th element in the ranking). Since we are only dealing with binary relevancy, we can rewrite that as:

The NDGC is then calculated by normalizing the DCG by dividing its value by the value that would be obtained by a perfect ranking (all relevant documents first).

where, in our case (binary relevancy, 0 or 1),

An example is given below

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ranking | Relevancy | Gain | Disc. Gain | Disc. Cum. Gain | DCG perfect | NDCG |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 0 | 1 | 2 | 0.5 |
| 3 | 1 | 1 | 0.63 | 1.63 | 2.63 | 0.62 |

## 1.2 Performance and time cost table (20%)

Fill in the table with your experiment results.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| C | Logistic Regression | | | SVM | | |
| P@10 | NDCG@10 | MAP | P@10 | NDCG@10 | MAP |
| 0.0001 | 0.12 | 0.222444 | 0.182743 | 0.09 | 0.123060 | 0.0742470 |
| 0.001 | 0.1 | 0.171854 | 0.157899 | 0.1 | 0.151063 | 0.117227 |
| 0.01 | 0.09 | 0.162860 | 0.101263 | 0.12 | 0.175026 | 0.123039 |
| 0.1 | 0.09 | 0.124049 | 0.075702 | 0.13 | 0.233529 | 0.163304 |
| 1 | 0.09 | 0.122624 | 0.067363 | 0.14 | 0.327028 | 0.267842 |
| 10 | 0.09 | 0.122624 | 0.067235 | 0.09 | 0.143158 | 0.114912 |
| 50 | 0.09 | 0.121510 | 0.070335 | 0.11 | 0.258353 | 0.186244 |
| 100 | 0.09 | 0.122624 | 0.067426 | 0.04 | 0.052009 | 0.030983 |

|  |  |  |
| --- | --- | --- |
| C | Time | |
| Logistic Regression (s) | SVM (s) |
| 0.0001 | 242 | 132 |
| 0.001 | 237 | 130 |
| 0.01 | 241 | 127 |
| 0.1 | 240 | 131 |
| 1 | 239 | 145 |
| 10 | 243 | 210 |
| 50 | 230 | 392 |
| 100 | 244 | 513 |

## 1.3 Plots (10%)

Plots graphs for each metric of Logistic Regression and SVM. You should have six graphs in total.

**For Logistic Regression:**

**For SVM:**

## 1.4 Analysis (20%)

In terms of processing time, we observe that the logistic regression algorithm doesn’t vary a lot with the choice of C. That is actually a consequence of the implementation choice of stopping the algorithm if it reaches certain number o iterations through the data. That decision tries to put an upper bond on the processing time to avoid spending too much time stuck around the optimum point. In this experiment, I limited the number of passes through the data to a maximum of 100 times.

With relation to the SVM algorithm, the processing time is also somewhat limited, since I decided to restrict the maximum number of iterations for the algorithm to 20,000. That breaks the algorithm before it would actually converge. That reduces the quality, but is a necessary measure to avoid huge computation times.

About the results of the Logistic Regression, we observe a dramatic relation between the regularization factor C and the quality of our final LETOR algorithm. That is probably an indication that LETOR algorithms should have some degree of complexity to work well (low penalization by the regularization factor to model complexity). Another possible explanation, which is very possible, is that the learning rate wasn’t perfectly chosen for higher values of C. It is hard to define a learning rate as a function of C and that strongly influences the results of the classifier. In any case, it is important to say that we start with a very high performance for low values of C and that performance quickly decreases with an augmentation of C. Once C gets big enough, it stabilizes and maintains a similar quality.

For the SVM we didn’t exactly observe the same trend. Instead, we observe a peak pattern around the point C=1. That is more plausible to what one would expect: a model complexity penalization that is too high or too low tends to make the model worse. A balance must be found. That is exactly what we observe here: when the penalization for model complexity is too high (high C) and when it is too low (low C) the LETOR quality is enormously decreased if compared with the value found around the optimum point. That is a very clear and interesting way of seeing and understanding the importance of making a good tuning of the regularization term, since it drastically changes the quality of our final model.

In terms of tuning the parameters, it took a very long time to find a reasonable value for the learning rate. A complicating factor is that since we have a big number of training instances, the running time is somewhat slow and that limits the number of experiments one can try (if compared, for example, to the previous homework).

For Logistic Regression, some decisions that were made by me on this experiment are:

* Stochastic gradient descent approach: due to the huge number of training instances (coming from the pair-wise generation of training data), it is less reasonable to use a bulk approach to the stochastic descent algorithm. Because of that, I decided to use the stochastic version of the algorithm.
* Normalization: A few normalizations occur in this algorithm. More specifically, we have the choice to normalize or not the training and/or the testing data. For the training data, we can also choose to normalize it before or after generating the difference vectors (vij). I have tested different configurations and ended up deciding to normalize both the training and testing data using L2 normalization. Moreover, the training data is normalized before the computation of the difference vectors. Vij. That set of decisions, in my particular experiments, tended to give much better results.
* Adaptive learning rate: as discussed in the previous homework, there is an important tradeoff between on the choice of the learning rate. By choosing a larger value of learning rate, we reach the optimum point faster, but it is more difficult to converge close to the optimum point. On the other hand, a smaller learning rate will make the convergence be more precise, but much slower. Adaptive learning is a technique to get the best of both worlds. For this dataset, I decided to decrease the learning rate by 5% every time we go through the training data.
* Learning rate as a function of the regularization factor. Naturally, we must balance the values of the regularization factor and that of the learning rate, since they will be multiplied and will decide the step size of a given iteration of the gradient descent algorithm. I found on this data set to decrease the learning rate in an inverse logarithmic function of the regularization factor.

For the SVM, we have a smaller decision power. The decisions taken by me were:

* Normalization: Once again, we have the choice to normalize or not the training and/or the testing data. For the training data, we can also choose to normalize it before or after generating the difference vectors (vij). I have tested different configurations and ended up deciding to normalize both the training and testing data using L2 normalization. Once again, the training data is normalized before the computation of the difference vectors. Vij. That set of decisions also turned out to give much better results.
* The maximum number of iterations is limited to 5000. That is a constraint to avoid spending too much processing time.

What is particularly interesting about this homework is the different approach we give to the result of a classifier such as LogReg or SVM. Before, we were interested in putting a label in a point. Now, we are interested in ranking it. Before, we were only interested if a given prediction was above the threshold or not, or how above it was. This is not the case here. What we are interested is a relative score among all predictions. Before, the prediction information of a given point would give us enough information about it. Now, that gives us no information. What we care about is that value compared to the value of other points. That was an approach that I had never seen before and that I found to be quite interesting.

Another brilliant aspect of that algorithm is the creation of the pair-wise training set. I was curious enough to run the entire algorithm without creating the v vectors, and the results are significantly worse. A very interesting aspect of this algorithm is that it tries to capture the essentiality of the effects of the features by taking the difference between a point with positive and one with negative label. Another interesting thing is that this algorithm allows us to “expand” our training data, and that is especially relevant in cases such as ours, where we have a very limited amount of positive labels (because they are both rare and expensive to find).

# 2. Custom features (25%)

The results for the new data, using the three new custom features are given below.

**With extra features**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| C | Logistic Regression | | | SVM | | |
| P@10 | NDCG@10 | MAP | P@10 | NDCG@10 | MAP |
| 0.01 |  |  |  |  |  |  |

The three selected features are

Feature 1:

This feature is the number of fields among {url, title, anchor} that have a non-zero term frequency. Those are usually short fields and with few but very representative terms. The idea here is to capture how relevant a document is by checking in how many of those short/representative fields the keywords appear.

Feature 2:

This feature is a very natural feature to imagine. It is the product of the PageRank with the BM25 rank. The main idea here is to count both document relevancy and document importance. BM25 accounts for the relevancy while PageRank accounts for the global importance of the document. By multiplying both, we hope to distinguish important+relevant documents from non-important+relevant, important+nonrelevant and nonimpornant+nonrelevant.

This feature was responsible for a small increase in the MAP score of both SVM and logistic regression.

Feature 3:

Let

HBSPwil = Hyperlink base score propagation: weighted in-link

HBSPwol = Hyperlink base score propagation: weighted out-link

The new feature is

What is interesting is that those two features used (39 and 40) are, according to [1] hybrid features that contain both content and hyperlink information. The attempt here is to simulate a correlation measure between the hyperlink base score propagation for the in-link and out-link. I thought this would be an interesting way of measuring the relation between those two and that could give us some information about the importance of the document, making a parallel with the pagerank. That feature alone gave us a small improvement in the score, but not very significant. Also, when running the 3 together, there is no improvement in both results.

Unfortunatelly, the increases I obtained for the extra features were very little significant. A possible reason is that the scores obtained were already relatively high (up to 0.26 MAP in SVM) and therefore hard to improve.

# 3. The software implementation (15%)

The design of the code is very similar to the one in the previous homework. In a sense, the algorithm for this homework is simpler because it is a one-class classifier. So I basically removed all the multi-class related code and just made it simple to use one-class data.

Both the codes for LogReg and SVM are done in Matlab R2011b. The first step of that code is retrieving the DATA.TXT file using the current directory ‘pwd’. I then run a function that reads the file and retrieves the paths for training and testing data, as well as the value for C.

A function is then called to read the data files and convert them to the proper matlab matrix format. We perform L2 normalization of the training matrix by calling another function.

Followed is the function that takes the training data and creates the difference vectors vij. We duplicate those vectors and adjust the sign to create the positive and negative training data.

What follows is very similar to what was done before, in our previous homework.

For logistic regression, the algorithmic part is given by:

* Vector w is initialized to 0.
* Training data is shuffled
* Stochastic gradient descent runs on the data
* For each training point, we compute the gradient value.
* Update (also for each training data point) the w vector
* After going through the training data, we compute the current LogReg loss function (for the latest w).
* Compare that value to the previous one. If the change is smaller than a defined  threshold, move to next label and store w.
* If change is bigger than threshold, reshuffle the training data and repeat the  procedure.
* Before repeating the procedure, adjust the learning rate by multiplying it by 0.95 (adaptive learning).

For SVM, we just call the system command line through matlab and run the binary files of svm\_light. Nevertheless, in order to do that, we must first have the training and testing matrices in a txt file. That is why we call a function to output those two matrices to a text file before running the binary files.

As allowed by the TA, I don’t output the predictions to stdout, but to a file called "./hw6\_predictions.txt".

References:

[1] LETOR: Benchmark Dataset for Research on Learning to Rank for Information Retrieval, Tie-Yan Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li