Information Retrieval 1 - Homework 3

Philipp Lintl
12152498
philipp.lintl@student.uva.nl

Caitlin Bruijs
11034041
caitlin.bruijs@student.uva.nl

Ruben Woortmann
10208593
ruben.woortmann@student.uva.nl

ACM Reference Format:
Philipp Lintl, Caitlin Bruijs, and Ruben Woortmann. 2020. Information Retrieval 1 - Homework 3. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 LOADING AND PROCESSING THE AP DATASET

2 OFFLINE LEARNING TO RANK

3 POINTWISE LTR

Analysis Questions

AQ2.1 Train a new model with the best performing model's hyperparameters. Every few batches (depending on your available compute power), compute NDCG over the validation set. Plot the loss and NDCG together. Comment on what you observe.

We performed a hyperparameter search for the number of hidden features and learning rate of the Stochastic Gradient Descent Optimizer. For this part, the pointwise LTR, we used a shallow one layer Neural Network with a mean squared Regression error as loss. We tried the following number of hidden unit values: $\{100, 150, 200, 250, 300, 350, 400\}$. For the learning rate, we tried the following values: $\{0.1, 0.01, 0.001, 0.0001\}$. Early stopping was assessed on the ndcg value on the validation data set. As the pointwise LTR approach is not optimizing the ranking directly but rather performing a regression task, we decided to stop once the ranking quality decreases. Our hyper parameter search yielded the best validation NDCG performance for a learning rate of 0.01 and number of hidden units of 150 and a batchsize of 32. We then trained our model based on this specification for the number of epochs that were needed to yield the best validation performance. Respective loss and performance according to NDCG can be seen in Figure 1. We observe a steeper decline in loss for the fist 30000 batches, after which the loss only changes slightly and womewhat converges aroud 0.65. As we plot both measures in one figure, the ndcg curve is not as fine grained. What we observe, matches the loss curve. As in a larger gain in the beginning and a convergence around 0.84. The model reaches a loss of 0.629 and a ndcg value of 0.852 on the test set.

AQ2.2 Compute a distribution of the scores (if you're using a classification loss, use the argmax) output by your model on the validation and test sets. Compare this with the distribution of the actual grades. If your distributions don't match, reflect on how you can fix this and if your solution is sufficient for LTR.

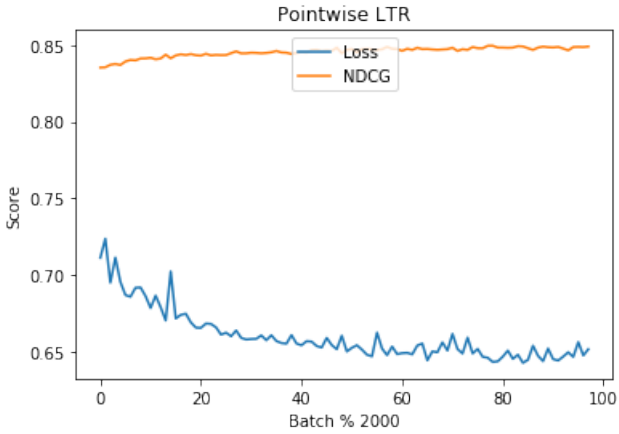

Figure 1: Loss and NDCG for lr=0.01 and hidden=150 for every 1000th validation batch. (Plot states every 2000th batch, which is supposed to be every 1000th)

value	validation	test	actual
mean	1.22	1.25	1.24
std	0.61	0.60	0.98

Table 1: Distribution of ranking scores for validation, test split and actual values.

So far, we decided to state the respective means and stds per dataset split, which is depicted in Table 1,

The mean values seem to match, whereas the std of the actual scores seems to be higher. Meaning, that the regression supposedly yields ranking scores that are closer together with less values being at the larger end. In order to investigate this further, we would need to look at the actual distributions/histograms, which can be seen in Figure 2.

Looking at Figure 2 reveals that the distributions are similar. However, the predicted ranking scores are slightly skewed towards 1.5 with its mode being approximately that value. The actual distribution has its mode at value 1. This graph also explains why the std of the actual scores is higher, as the actual scores are discrete and thus vary more than the predicted values, which more or less resemble a Gaussian. If the distributions did not match at all, we could model the pointwise approach using a multi class classification loss, which at least would enforce a discrete distribution

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

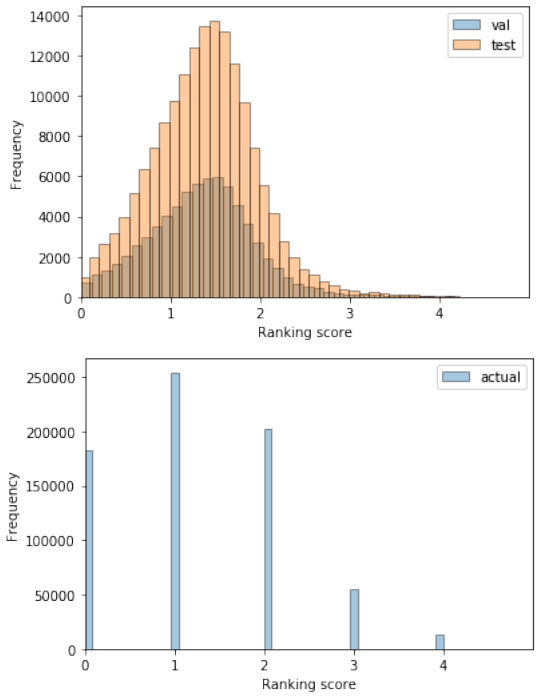

Figure 2: Histogram of ranking scores for test and validation dataset on the top and distribution of actual scores on the bottom.

as well. Considering the distributions match in an ok fashion, we would regard this as sufficient for LTR, which is remarkable for only treating each instant separately as a regression task.

4 PAIRWISE LTR

For RankNet we trimmed search space of the previous hyperparameters due to the addition of the σ parameter. Searched learning rates were now 0.01, 0.001 and the amount of neurons in the hidden layer: 150, 200, 250, 300, 350. Appropriate sigmas of 0.01, 0.001 were determined by performance over few epochs when finding that larger sigmas drastically reduced scores.

Analysis Questions

AQ3.1 Contrast the convergence of the sped up version of RankNet with the previous version.

The RankNet version uses more epochs to converge which can be seen from the fact that it increases for more epochs. For the sped up version, it can be seen that after 2-3 epochs, the model performance does not increase anymore, so it converges faster. This is mostly due to the fact that it has to do less backprops in the model, which makes the model less complex and faster to converge. However, from Table 3, it can be seen that although the model seems to converge, the NDCG to which the model converges is slightly lower for the Sped Up RankNet than the original RankNet model.

AQ3.2 Train a new model with the best performing model's hyperparameters. Every few batches (depending on your available compute power), compute NDCG and ARR (Average Relevant Rank) over the validation set. Plot these numbers. Explain what you observe. In particular, comment on which metric is better optimized and what you can conclude from it.

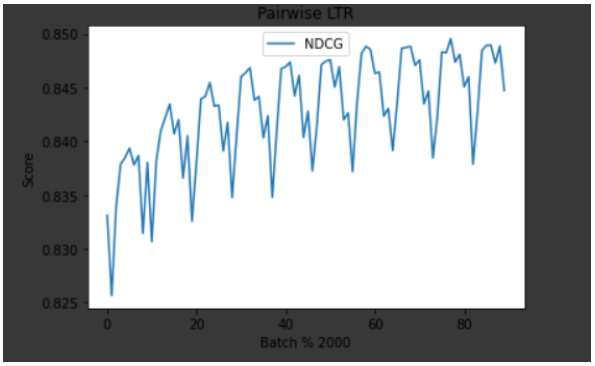

Figure 3: NDCG for lr = 0.1, hidden layers = 300 and sigma = 0.001 for every 1000th validation batch. (Plot states every 2000th batch, which is supposed to be every 1000th)

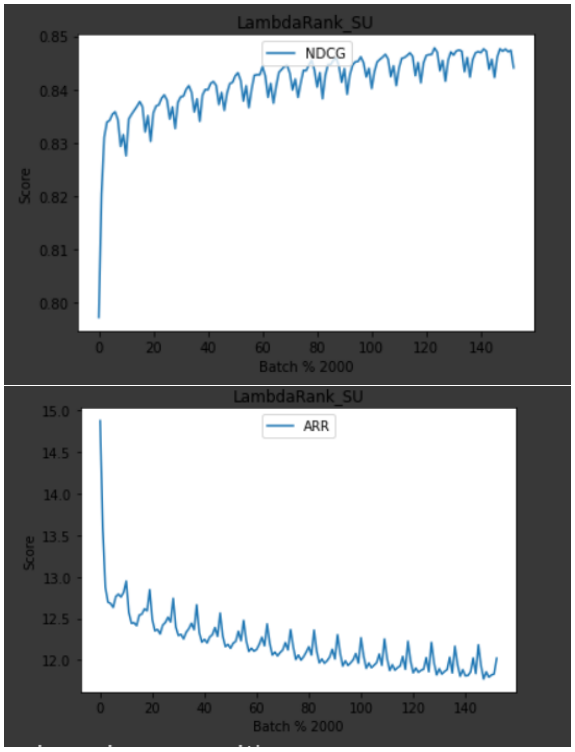

Figure 4: NDCG and ARR for lr = 0.01, hidden layers = 150 and sigma = 0.001 for every 1000th validation batch. (Plot states every 2000th batch, which is supposed to be every 1000th)

The optimal hyperparameters for this model are: {'learning_rate':

0.01, 'n_hidden': 150, 'epoch': 17, 'sigma': 0.001}. From figure 4 it can be seen that the two metrics follow an opposite direction. A good model performance corresponds to a higher NDCG, whereas the average relevant rank (ARR) must be as low as possible, since you want your relevant documents at the top of the rank. The figures both show a similar, exactly opposite pattern.
Moreover, it can be concluded that this metric converges, but this is as expected since the model is trained by minimizing the NDCG. This can be seen from the smaller oscillations in figure of the NDCG (the y-axis of the NDCG is on a lower scale). In conclusion, the metric on which you train and validate your model will likely be the metric that is optimized best.

Measure	RankNet		RankNet Sped Up	
	(mean)	(std.)	(mean)	(std.)
NDCG	0.8416	0.1384	0.8455	0.1381
ERR	0.8324	0.5047	0.8246	0.4696

Table 2: Performance measures of pairwise RankNet evaluated on the test set.

5 LISTWISE LTR

Analysis Questions

AQ4.1 Train a new model with the best performing model's hyperparameters. Every few batches (depending on your available compute power), compute NDCG over the validation set. Plot these results. How does this compare to the previous two plots for RankNet and the Pointwise model?

For listwise with ncdg as an IRM we found that with a learning rate of 0.001 with 200 hidden neurons and a sigma of 0.01 we had the highest ndcg of 0.8353. As listwise with ERR as an IRM was much slower we were only able to evaluate it on the validation set which resulted in a score for ndcg: 0.8354 with parameters

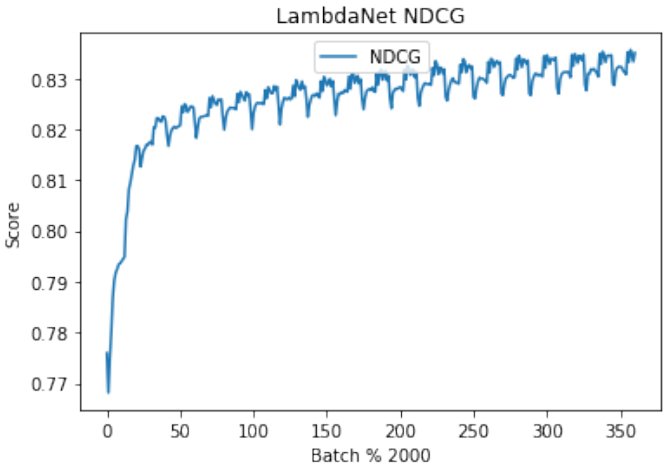

Figure 5: NDCG for lr = 0.001, hidden layers = 200 and sigma = 0.01 for every 1000th validation batch.

Measure	LambdaNet NDCG		LambdaNet ERR	
	(mean)	(std.)	(mean)	(std.)
NDCG	0.8391	0.1377	0.8354 (val)	
ERR	0.8323	0.5051		

Table 3: Performance measures of LambdaNet evaluated on the test set.

We see that it takes much longer for LambdaNet to converge, while no improvement in score shows itself.

AQ4.2 (10 points) We have in total 3 models: Pointwise, RankNet, and LambdaRank, and 3 corresponding losses. Compute the losses we've discussed so far against each model (select the best performing model). Report these in a table. Reflect on what you observe.

One remark to the form of the performance/loss curves in Figures 3,4,5. The pattern arises as a result of not shuffling the queries before iterating over them. Thus, the variance in queries is visible in those step-like curves, as we do not incorporate batching more than one query in our implementation. Shuffling the queries beforehand would eliminate those patterns. Shuffling the queries for each epoch, also could have helped our problem with early stopping as mentioned in the paragraph below.

Table 4 shows the performances for the best parameter setting for each of the depicted models. It is remarkable that the pointwise model performs best according to the ARR and NDCG measure. We came up with a few possible reasons. For once, it seems, as if our pair- and listwise approaches were not converged at all. We used the same early stopping criterion as for the pointwise approach. More precisely, stopping the calculation for one parameter setting, if the validation ndcg score of the current epoch is lower than the previous one. Though this is useful for pointwise methods, which are not based on ranking, it might be a problem for ranking based models. In fact, especially higher learning rates will yield higher jumps in the loss, thus temporary smaller performance values can appear, if the model was caught in a local minimum. Therefore, another early stopping mechanism combined with more epochs could have yielded another parameter setting, which in turn could have resulted in better scores. Furthermore, as hinted previously especially the pair- and listwise approaches do not seem to have converged, which calls for more epochs to be run. Another reason could be caused by the distribution of labels. As many queries contain a large number of documents, which are often irrelevant, e.g. labels zero and one (see Figure 2, bottom), the model gets lot of noise as input. Also, the *ideal rankings* are based on the given relevance scores. Often, ties in relevance score exist, which can cause noise in the rankings, as the ranking is then not representative. In order to overcome these issues, we could have discarded more irrelevant documents especially for queries with a lot of returned documents. Furthermore, we used a smaller grid of possible hyper parameters for the pair- and listwise approach due to time constraints. This together with a different early stopping mechanism and more epochs should have resulted in higher scores compared to the pointwise method. Another reason could be that the models have different

methods of training. The pointwise method is used with proper batching of batch size 32. This stabilizes gradients and leads to a less 'jumpy' loss function. The pair- and listwise approaches on the other hand were trained per query, which can lead to large parameter changes, making learning harder. Thus, either adjusting the pair- and listwise loss functions for batching or by using other measures, such as adjusting the learning rate by time or lower values in general while training longer, could have overcome this issue.

Measure	Pointwise	RankNet	LambdaRank
NDCG	0.853	0.8458	0.8391
ERR	0.814	0.8453	0.8323
ARR	11.535	12.0207	12.3020

Table 4: Performance measures for pointwise Learning to rank, pariwise RankNet and listwise LambdaRank.

6 THEORY QUESTIONS

Theory Questions

TQ 1.1 In this assignment you are already given the feature vector x for a given document and query pair. Assuming you have control over the design over this feature vector, how would you design it for the e-commerce domain?

For the e-commerce domain, it may be usefull to include for example clicks in the vector x to include another relevance measure instead of only using rank. **TQ1.2 Briefly explain the main limitations of Offline LTR.**

First, creating such datasets is expensive and therefore infeasible for smaller search engines, such as small web-store search engines. Second, it may be impossible for experts to annotate documents, as in the case of personalized search.

Third, the relevance of documents to queries can change over time, like in a news search engine.

TQ1.3 Contrast the problem of learning to rank with ordinal regression/classification.

The main difference is what the output of the models in the different problems are. In ordinal regression/classification, the task is to predict a label for a given sample, hence the output of a prediction is a label. On the other hand, in the problem of learning to rank, the output is an order of a sequence of samples. That is, the output of a ranking model can be seen as a permutation that makes the samples to have labels as ordered as possible. Hence, unlike the ordinal regression model, the ranking algorithm is not able to predict a class label. Because of this, the input of a ranking model does not need to specify class labels, but only a partial order between the samples

TQ1.4 Is it possible to directly optimize a given metric? For instance, can we use ERR as a loss function? Explain your answer.

Probably not. A loss function is just a more general evaluation of whether or not the returned ranking was a correct one. However, this does not make clear what should happen (e.g. what pairs to swap) to decrease the loss. For a directly optimizable metric, probably all permutations of the documents per query must be evaluated to 'see' which ranking minimizes the loss. But this takes a lot of computational power and time. Therefore, it is better to indirectly optimize a metric, so you can see what needs to change to create a better ranking. Trying to directly optimize a metric is difficult because most evaluation measures are not continuous functions with respect to ranking model's parameters, and so continuous approximations or bounds on evaluation measures have to be used.

Theory Questions

TQ2.1 Briefly explain the primary limitation of pointwise LTR with an example.

Pointwise LTR considers each document individually, independent of the remaining instances per query. For cases of highly varying number of documents returned per query, this leads to problems. Furthermore, pointwise approaches do not take advantage of the position of each answer in a ranked list. Thus, each candidate answer gets the same weight, meaning that answers at the bottom of the ranked list are treated the same as the ones on top, even though they are less important to that query. Thus, by not leveraging the ranking, the overall performance is expected to be lower than pair- or listwise approaches.

Theory Questions

TQ3.1 Comment on the complexity of training the RankNet (hint: In terms of n, the number of documents for a given query).

The complexity of LTR is $O(|n|^2)$ for a given query, since for each query all possible document pairs are considered. And since a weight update is expensive, since e.g. for a neural net model, it requires a backprop, this is close to quadratic complexity.

TQ3.2 Comment on the complexity of training the sped-up version of RankNet and contrast it with the first version you implemented. (Hint: Consider the number of pairs of documents for a given query)

Since with sped up LTR the λs are accumulated for each query, only backprop is needed for the λ's, the complexity drops from close to quadratic to close to linear. So, the way the problem factorizes, speeds up the process.

TQ3.3 Mention the key issues with pairwise approaches. Illustrate it with an example.

The pairwise approach might suffer from inconsistency. For example, it might happen that, when pairwise LTR has to rank 3 documents, it comes up with the following 'preferences': $D_1 \succ D_2$, $D_2 \succ D_3$, $D_3 \succ D_1$. This results in a loop where no total ranking can occur. Since pairwise LTR is only concerned about predicting relative order (which is closer to the nature of ranking) instead of predicting class label or relevance score in pointwise LTR. So this inconsistency does not happen in pointwise LTR or listwise LTR, since the latter predicts the ranking of the whole list instead of pairs.

Theory Questions

TQ4.1 What does the quantity λ_{ij} represent in LambdaRank?

The λ_{ij} represents the gradient of the cost with respect to the model score. If you look at the function $\lambda_i^q = \sum_{j:(i,j)\in/q} \lambda_{ij}^q - \sum_{j:(j,i)\in Iq} \lambda_{ji}^q$, You may think of the documents as point masses. λ_i is then the (resultant) force on the point q mass d_i. The first sum accounts for all the forces coming from less relevant documents pushes d_i up in the ranking. The second sum accounts for all the forces coming from more relevant documents pushes d_i down in

the ranking.

TQ4.2 You compute a similar λ_{ij} in RankNet, but the approach is still pairwise. In your opinion, does this mean that LambdaRank is still a pairwise approach? Justify your claim.

It might be considered as a pairwise approach, since the lambda considers pairs of candidates, but it actually requires to know the entire ranked list (i.e., scaling the gradient by a factor of the nDCG metric, that keeps into account the whole list) – with a clear characteristic of a Listwise approach.