IR 1 - Homework 3

Puck de Haan
11305150
pdehaan274@gmail.com

Tim van Loenhout
10741577
timvanloenhout@gmail.com

Ard Snijders
12854913
ardsnijders@gmail.com

1 POINTWISE LTR

Implementation

Of all LTR models, the pointwise model is the simplest. All rows of the feature matrix represent a query-document feature vector and we have corresponding labels for each vector. To train the model, we put a batch of feature vectors through a shallow neural net (specifications will follow) and calculate the loss between the predictions and the ground truth by means of the mean squared error (MSE). Thus, we are using a regression loss in our model to predict the relevance labels.

Furthermore, we apply early stopping by monitoring the NDCG of the model on the validation set. The training process is stopped when the NDCG does not improve from the best score so far for 5 consecutive epochs, with a margin of $\delta = 0.001$. The Pointwise model itself has two hidden layers with LeakyReLU activations between each layer – the number of nodes in the hidden layers is a hyperparameter. As optimizer we used Adam [2], since this is usually a solid choice and we didn't experience any problems.

Optimal hyperparameters. See Table 1.

	Value	NDCG
Hidden nodes	50	0.8494
	100	**0.8512**
	150	0.8486
Learning Rate	0.1	0.8355
	0.01	0.8396
	0.001	0.8458
	0.0001	**0.8499**
	0.00001	0.8450
Batch Size	64	0.8471
	128	**0.8480**
	256	0.8479

Table 1: Hyperparameter tuning for pairwise learning to rank

Performance on test set.

- NDCG: 0.8528
- Test Loss: 0.6247

IR 1, February/March, MSc AI, UvA
2020.

Analysis Questions

AQ2.1. Performance for the pointwise ranking model with optimal hyperparameters can be seen in Figure 1. The model is evaluated for loss and NDCG every 5000 batches of size 128. It can be seen that almost instantly, the NDCG on the validation reaches a score of 0.85 or so, whereas the validation loss approaches its steady state more gradually, converging after 11 epochs.

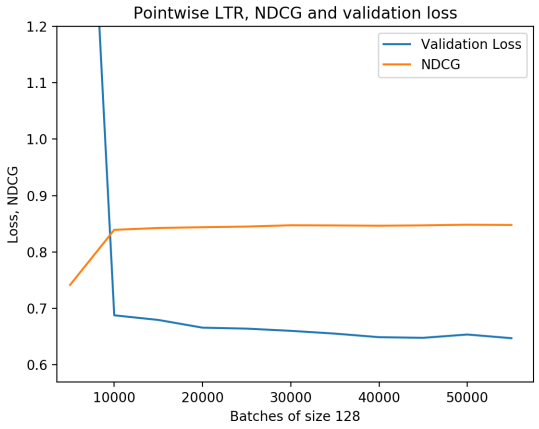
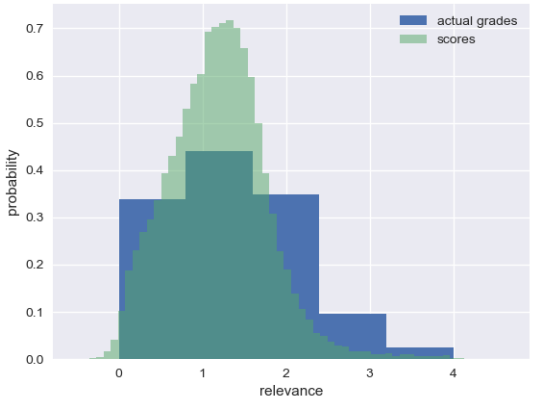
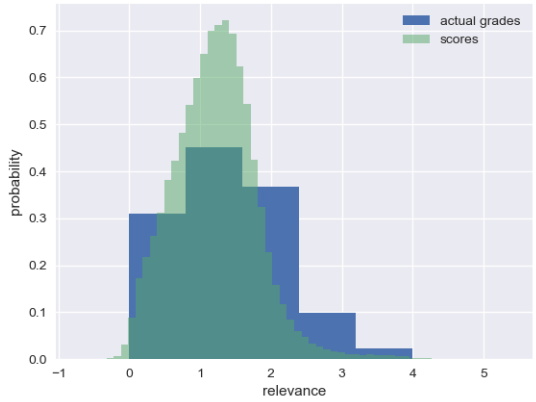


Figure 1: Performance of pointwise LTR on validation set

AQ2.2. When comparing the distribution of the actual validation/test scores to the distribution of the actual validation/test grades (figure Figure 2), it can be seen that the for both data sets the model outputs show a similar mean to the actual grades. The variance on the other hand, is lower than for the actual grades. This suggests that the model puts relatively more emphasis on the the more frequent lower relevance scores, while it is relatively more reserved in assigning high scores. In order to fix this, the already existing (minor) class imbalance could be smoothed out to have a more uniform distribution across the actual labels used for training. Another solution could be the use of a classification loss network instead of the currently used regression approach. That is, the regression algorithm will try to minimize the distance to the real scores, hence putting relatively more emphasis on preventing high scores that could potentially result in substantial contributions to the loss, especially when utilizing the MSE loss, which takes the square root of these distances.



(a) Validation set



(b) Test set

Figure 2: Distribution of Pointwise model scores and actual relevance grades on both the test and validation set.

2 PAIRWISE LTR

RankNet

Implementation. For every forward pass in Pairwise learning to rank, each batch consists of all the document representations for one specific query and its corresponding query/document label. The document representations are fed through the shallow Pairwise model, consisting of two hidden layers, of which the former is followed by a RELU activation. The number of nodes in these layers and the learning rate can be tuned using a hyperparameter.

Subsequently, the outputs and corresponding labels are passed through a function that computes the loss for the whole query. This is done by first constructing two tensors, which for each possible pair contain; the difference between the predicted scores (z_{ij}) and the difference between the actual scores (S_{ij}). Furthermore, for the latter, all scores are converted to a value from the set -1,0,1 by setting all positive differences to 1 and all negative differences to -1. Finally, the loss is computed using the function:

$$C = 0.5(1 - S_{ij})\sigma * z_{ij} + log(1 + e^{-\sigma z_{ij}}) \quad (1)$$

where σ is set to 1. For queries containing only one document score, the loss is automatically set to zero as only one ranking is possible.

The query/batch cost is then used for backpropagation. Furthermore, every few batches, the scores over the validation set are computed to check for convergence. This is done by an early stopping function that stops training when the validation loss does not decrease anymore, within a 5 steps.

Optimal Hyperparameters. This aforementioned process is repeated with different hyperparameter settings, for which the results can be seen in Table 2.

	NDCG
hidden nodes: 50	0.8378
hidden nodes: 100	**0.8381**
hidden nodes: 150	0.8306
learning rate: 0.0001	**0.8381**
learning rate: 0.001	0.8233
learning rate: 0.1	0.7177

Table 2: Hyperparameter tuning for pairwise learning to rank

Performance on test set. Finally, the trained model with optimal hyperparameters of 100 nodes and a learning rate of 0.0001 is used for evaluating the performance on the test set; resulting in an NDCG of 0.8346 and ERR of 0.4410.

RankNet: Sped up

Implementation. The sped-up version of Ranknet is equal to the regular version of Ranknet, with one adjustment: instead of backpropagating over the entire model, comprised of both the gradient of the cost w.r.t. the scores and the gradient of the scores w.r.t. the weights, we only backpropagate over the scores. To do so the gradient of the cost over the scores is computed per query and used as a "force" to multiply with the gradients of the scores over the weights. This way we do include the effect of the cost, but the model only has to calculate the gradients for the scores w.r.t. the weights. Thus, we changed the loss function to calculate the gradients of the cost for all valid document pairs i and j with the gradients detached, so they can be used for backpropagation of the predicted scores.

The model architecture and all other settings were the same as for the regular Ranknet model.

Optimal Hyperparameters. This sped up version is repeated with different hyperparameter settings, for which the results can be seen in table 3.

	NDCG	ERR
hidden nodes: 50	**0.8437**	**0.5132**
hidden nodes: 100	0.8431	0.5075
hidden nodes: 150	0.8443	0.5108
learning rate: 0.0001	**0.8437**	**0.5132**
learning rate: 0.001	0.8423	0.5116
learning rate: 0.1	0.7638	0.3004

Table 3: Hyperparameter tuning for sped-up pairwise learning to rank

Performance on test set. Finally, the trained model with optimal hyperparameters of 50 nodes and a learning rate of 0.0001 is used for evaluating the performance on the test set; resulting in an NDCG of 0.8471 and ERR of 0.5130.

Analysis Questions

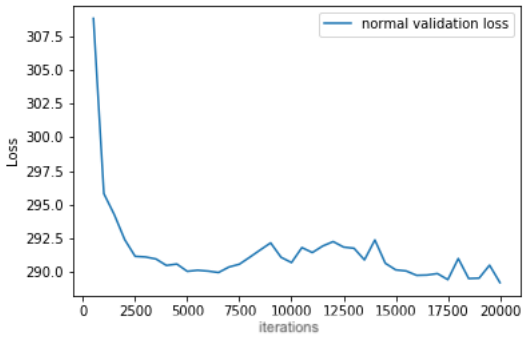
AQ3.1. Convergence of the normal pairwise model and the spedup pairwise model on the validation set can be seen in Figure 3. It should be noted that the losses of the sped-up pairwise model are at a different scale than the losses from the normal model; hence it was chosen to plot these separately. Evluation on the validation set was performed every 500 batches. For the normal pairwise model, it can be seen that convergence is achieved after 20000 batches, which is loosely equivalent to 2 epochs. In case of the spedup pairwise model, we can observe that the validation loss exhibits a large amount of variance - this is somewhat odd, as the NDCG - as can be seen in 4a converges rather nicely.

AQ3.2. See Figure 4 for the NDCG and ARR over the validation set. It seems like the NDCG is better optimized, since there are less fluctuations when closing in on convergence. Despite this, we still observe good performance on the ARR - after 400 queries have been passed through the network, it can be observed that ARR remains between 12 and 13, approximately. A low ARR is desireable, as it means that on average, the relevant documents are found high on the final ranking.

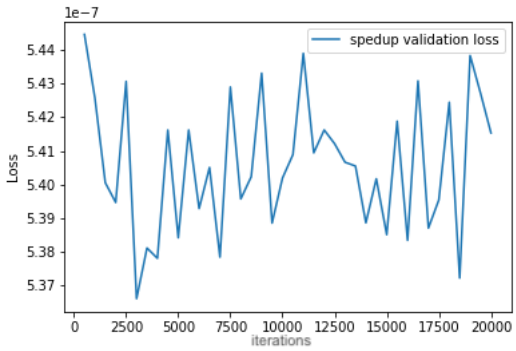
3 LISTWISE LTR WITH LAMBDARANK

Implementation

The implementation is very similar to the implementation of the sped-up Ranknet. Firstly, λ_i is calculated the same way as in the sped-up Ranknet model, by manually calculating the gradients of the cost w.r.t. the scores. The only difference is that we also calculate the ranking measure IRM for all document pairs i, j for each query. Finally, this IRM value is



(a) Normal model



(b) Spedup model

Figure 3: Performance of normal and spedup Pairwise LTR on the validation set.

multiplied with the corresponding *lambda*-value, so we take the entire ranking into account when updating the model. Of course we backpropagate the scores w.r.t. the weights of the model, mulitplied with our $\lambda *$ IRM value.

Unfortunately the listwise model took a very long time to run on our machines, so we were not able to do extensive hyperparameter tuning. That is why we adapted the optimal hyperparameters we found for the similar sped-up pairwise model. The optimizer and early stopping function used were the same as for the other models.

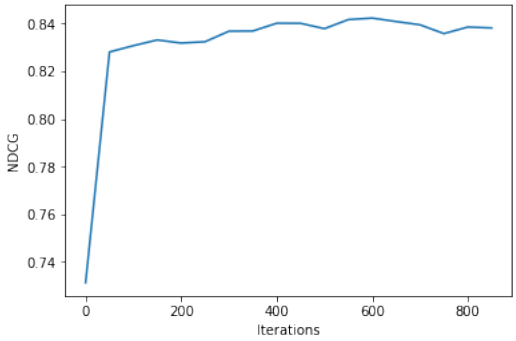
Optimal hyperparameters. Not tuned. See sped-up Ranknet.

Performance on test-set.

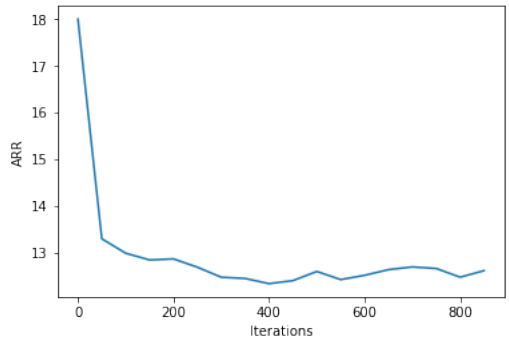
- NDCG: 0.8244
- ERR: 0.1829

Analysis Questions

AQ4.1. The plot of the NDCG Figure 5 looks similar to the plots of the NDCG for the other models. Maybe one could say that the NSCG for the listwise model is less stable and



(a) NDCG



(b) ARR

Figure 4: NDCG and ARR of sped-up pairwise model on the validation set.

fluctuates more. Also, it converges very fast to its optimal value and shoots down afterwards, so it seems to a slightly less reliable method than the pairwise model – at least for the dataset we used.

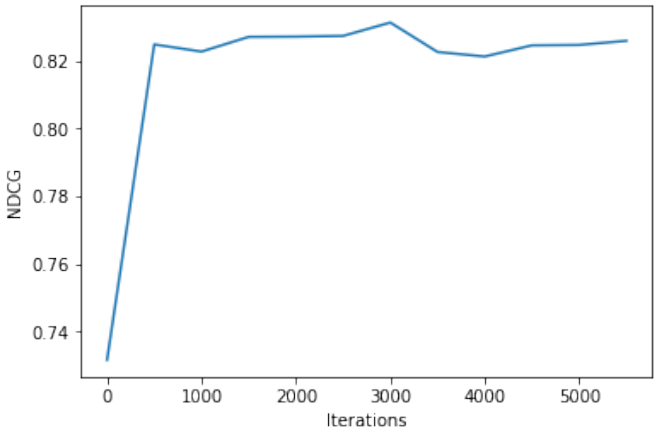


Figure 5: NDCG on the validation set for listwise LTR model

Theory Questions

TQ1.1. If the feature vector had to be designed for e-commerce, we would design the feature vector such that it effectively represents the link between a query and corresponding products. We would probably need a large number of query and product attributes to be able to return relevant products to the consumer. It might also be important to think about features in terms of relevance and popularity and which of these two factors we want to weight more heavily [1].

TQ1.2. Offline LTR relies on annotated datasets; producing such annotated datasets is costly in terms of training annotators. In some cases, a retrieval problem is too personal (e.g. recommendation systems), making it impossible for annotators to assess whether a result is relevant for that user. Annotated datasets are also static and do not account for changes in relevancy. Finally, annotated datasets do not necessarily always align with true user relevance preferences (even citeren naar dat onderzoek in de slides, eventueel).

TQ1.3. In ordinal classification/regression, we attempt to assign some ordinal category to an object, given an initial feature vector; the goal is to optimize for classification accuracy given the set of possible classes. In contrast, the goal of learning to rank is to produce a ranking, and as such we aim to directly optimize for ranking quality.

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

Using a method such as LambdaRank, it is possible to directly optimize for a given metric, by observing how a metric is affected by swapping pairs of documents, and using this information during the update step of gradient descent - however, it should be noted that one cannot use a metric as a loss function directly, because discrete metrics such as DCG and ERR by themselves cannot be differentiated; instead, we just observe the impact of swapping documents on a given metric, and scale the gradients accordingly.

TQ2.1. The task of ranking documents is not a regression or classification problem. In this respect, a document-level loss is somewhat meaningless as document relevancies are not independent - it is hence not possible to optimize for ranking quality with pointwise LRT, which makes it a comparatively poor LTR approach.

TQ3.1. The complexity of training the Ranknet can be seen below in Equation 2, m denotes the number of training queries and n denotes the maximum number of documents per query.

$$O(m * n^2) \tag{2}$$

Since all permutations of document pairs have to be checked to calculate the cost, in the worst case scenario we have to do n^2 operations for all m queries.

TQ3.2. We are unsure on the complexity of the sped-up version of Ranknet. It seems like the complexity would be the same as for the regular Ranknet, since we still have to compare all possible doc pairs for every query. However, there was a significant speed up, which would be slightly strange when the complexity is exactly the same. This indicates that the bulk of the processing time comes from the backpropagation steps and not the forward steps.

TQ3.3. The key problem with pairwise approaches is that the model tries to minimize the number of incorrect pair aversions, while treating every pair as equally important. For instance, consider ranking A (c-c-c-f-f) and ranking B (f-f-c-c-c) where c stands for correct f stands for false. Both rankings contain the same number of correct inversions, and are thus considered equally accurate. In reality however, the correct ordering of the documents at the top of the ranking is more important than the correct ordering of the bottom documents, hence ranking A should be considered better than ranking B.

TQ4.1. The quantity $\lambda_{i,j}$ represents the gradient of the cost given a pair of document i and j over the score of document i. This gradient pushes document i away from document j or pulls it closer to it.

TQ4.2. Similarly to RankNet, in LambdaRank the cost is computed based on a pair of scores, albeit a weighted pair. By considering this weight, the algorithm considers the pair in the context of the whole list. Nonetheless, the pair losses are computed independently from each other, hence we still consider LambdaRank a pairwise approach.

REFERENCES

[1] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. 2017. On Application of Learning to Rank for E-Commerce Search. https://doi.org/10.1145/3077136.3080838

[2] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).