Information Retrieval Assignment 3

David Vos
11295481
vosdavid98@gmail.com

David Knigge
11327782
Knidgey@gmail.com

Marcel Velez Vasquez
11299746
marcel.velez1997@hotmail.com

ABSTRACT

KEYWORDS

Learning to rank

ANALYSIS QUESTIONS

POINTWISE LEARNING TO RANK

Determining optimal hyperparameters

To obtain the best performance from the pointwise model, we implemented a search over a set of hyperparameters. The hyperparameters we tuned were the learning rate, the size of the hidden layers used and the number of hidden layers used. In changing one of these parameters, the other two were kept at a fixed value. The fixed values were 0.001 for the learning rate, 100 for the number of nodes in each hidden layer and 3 for the number of hidden layers used in the model. For the learning rate, a search was performed over the values $\{0.0001, 0.001, 0.01, 0.1\}$. For the number of nodes in each hidden layer a search was done over the values $\{10, 50, 100, 150, 500\}$, and for the number of hidden layers a search was performed over the values $\{1, 3, 5, 10, 50\}$. Results for each configuration can be seen in table 1. From this table we can conclude that the best hyperparameters are a learning rate of 0.001, a hidden layer size of 150 with 5 hidden layers.

Stopping condition

We defined the stopping condition as follows: we keep training until the evaluation and training loss have diverged by at least a ratio of ϵ, where for the pointwise model we took an ϵ of 0.1, and when the last evaluation NDCG value falls below the average of the last κ evaluation NDCG values, where for the pointwise model we took a κ of 5. We thus stop training when evaluation and training loss start to diverge heavily, and the NDCG is on a downward trend.

AQ2.1

With the best set of hyperparameters, a learning rate of 0.001, a hidden layer size of 150 with 5 hidden layers, the results obtained are shown in figure 1. The loss and NDCGs are printed every 50 batches, where each batch has a size of 1024 document-query pairs. In total, before the stopping condition is reached, the model is trained for 191 batches, equalling about 20 epochs. We can see that the training and evaluation loss go down quite rapidly at first, and the NDCG rapidly goes up to a value of around 0.8400. From then on the NDCG slowly but steadily improves to a value of 0.8495, around which it oscillates slightly. We notice a sharp peak in the training and evaluation loss around 115-120 batches, which is reflected as a dip in the NDCG. It is interesting to see the spike in loss reflected in the NDCG value, as we aren't directly optimising for the NDCG, but this co-occurence of volatility shows that the current loss function of the model does in some way relate to the value of the NDCG. Since the training and evaluation loss have not diverged too much at this point, we continue training and can see the evaluation loss ever so slightly decreasing until at around 190 batches the stopping condition is finally reached.

The optimal configuration obtains an NDCG of 0.8503. The full evaluation results for the test dataset using this optimal configuration are given in `'./results/pointwise_evaluation.json'`.

LR	NDCG	Hidden size	NDCG	Hidden layers	NDCG
0.0001	0.8455	10	0.8395	1	0.8455
0.001	0.8465	50	0.8449	3	0.8449
0.01	0.8395	100	0.8472	5	0.8473
0.1	0.8296	150	0.8481	10	0.8472
		500	0.8451	50	0.7227

Table 1: NDCG results for the pointwise model on the evaluation set for different hyperparameter configurations.

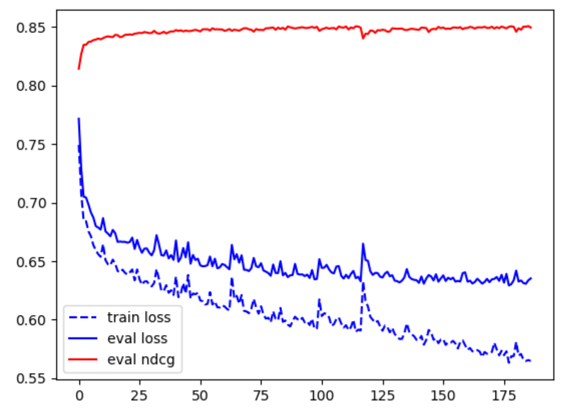

Figure 1: The training and evaluation loss and evaluation NDCG over the number of training iterations for the optimal set of hyperparameters.

AQ2.2

To see whether or not our model creates realistic predictions for a document-query pair, we plotted a histogram of the true ranking score distribution and the ranking score distribution given by our trained model. These results can be seen in figure 2. As we can see, the true ranking scores are way more spread out than the results

Conference'17, July 2017, Washington, DC, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

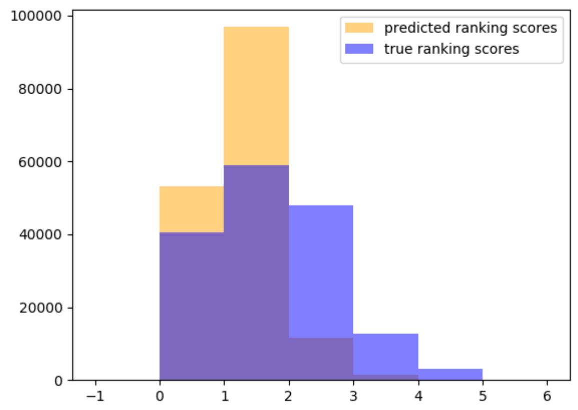

Figure 2: Distribution of true and predicted labels.

our model gives us. The model rarely gives a document query pair a score of or above 2, although the label 2 occurs often in the true ranking scores. This problem is probably caused by the fact that we are using a MSE regression loss to train our model. As this model evaluates document-scores for a given query independently, it is not optimising a ranking but instead it is minimising the error for all documents overall. As we can see, most true ranking scores are either 1, 2 or 0. As the MSE loss is optimising the score for each document independently, the model obtains a low loss when on average the ranking score predicted by the model is close to the true ranking score. Since there is heavy class imbalance - true ranking scores of 3 and 4 barely occur -, the model defaults to ranking most documents in the range from 0 to 2. This way the model obtains a low loss, but the predictions of the model aren't reflective of the true ranking score distribution. To improve the distribution created by our model we could balance the labels before training, but then our training set would not be reflective of the real dataset and the model may not generalise to new data well. We can thus say that this model isn't sufficient as a ranking model. We will improve on this model in further sections.

PAIRWISE LEARNING TO RANK

To train the optimal version of RankNet, a pairwise model, we performed a hyperparameter search over multiple learning rates, amounts of hidden layers and layer sizes. When changing one of the parameters, the other two were fixed. For the learning rate, we trained models with the values 0.00001, 0.0001 and 0.001. We tested hidden layer architectures with 1, 2, 3 and 4 layers. The sizes of the layers were varied between 50, 200, 500 and 1000. When varying one hyperparameter, the others were set to fixed values. The learning rate was set to 0.001, the amount of hidden layers to 1 and the size of the hidden layer to 500. The NDCG values of these networks can be found in Table 2. For the pairwise model we defined a stopping condition that is exactly the same as the pointwise model, with the only difference being the fact that the pairwise loss is used to test performance on the evaluation set.

LR	NDCG	Hidden size	NDCG	Hidden layers	NDCG
0.00001	0.8134	50	0.8304	1	0.8314
0.0001	0.8310	200	0.8313	2	0.8293
0.001	0.8312	500	0.8294	3	0.8308
		1000	0.8246	4	0.8191

Table 2: NDCG results for the pairwise model on the evaluation set for different hyperparameter configurations.

LR	NDCG	Hidden size	NDCG	Hidden layers	NDCG
0.00001	0.7906	50	0.7837	1	0.7880
0.0001	0.7839	200	0.7856	2	0.7901
0.001	0.7826	500	0.7882	3	0.7838
		1000	0.7826	4	0.7879

Table 3: NDCG results for the sped-up pairwise model on the evaluation set for different hyperparameter configurations.

As can be seen in the table, the best hyperparameters are a learning rate of 0.001 and an architecture of 1 hidden layers of each 200 nodes. The performance of this model is an NDCG of 0.8216 and an ERR of 0.8791 on the test set.

SPED-UP PAIRWISE LEARNING TO RANK

To speed up the training process of the pairwise learn-to-rank method, we implemented a weights-updating mechanism described in the assignment. We performed the same hyperparameter search as the non-sped-up version. The results can be found in Table 3. For the sped-up pairwise model we defined a stopping condition that is based on the evaluation NDCG value of our network. We save all the evaluation NDCG scores and when the last evaluation NDCG value falls below the average of the 10 last evaluation NDCG values we stop training.

As can be seen in Table 3 the optimal configuration is a learning rate of 0.00001, 2 hidden layers and a hidden layer size of 500 nodes. When this optimal model is trained, the NDCG on the test set is 0.7757 and the ERR on the test set is 0.9375.

AQ 3.1

It is rather difficult to directly compare the convergence of the evaluation NDCG of the sped-up version to the 'normal version' as they both converge to different values. Both networks improve around 0.06 NDCG points from their baseline of 0.72 in only 100 queries. The sped-up version stays around this NDCG value of 0.78 for the next steps until early stopping is activated. The slower version however keeps improving until an NDCG of above 0.83 over the next 10 steps. This clearly shows us that the slower version takes longer to converge than the sped-up version.

AQ 3.2

The NDCG and ARR on the evaluation set during training can be seen in Figure 3 and Figure 4. These were calculated during the training of the best pairwise model without speedup. This means that the hyperparameters are a learning rate of 0.001 and an architecture of 1 layer of 200 hidden nodes. It can be observed that the total trend is that when the NDCG rises, the ARR decreases. This

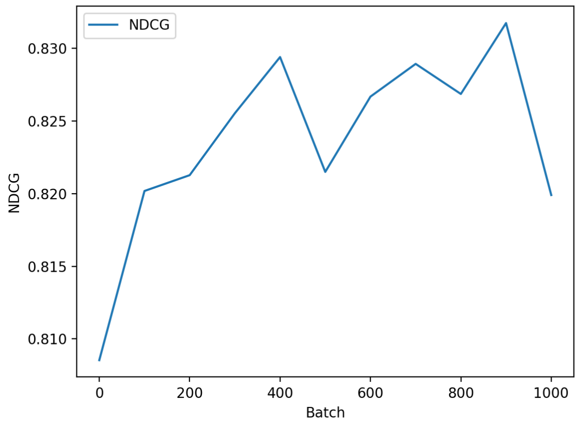

Figure 3: Evaluation NDCG during training.

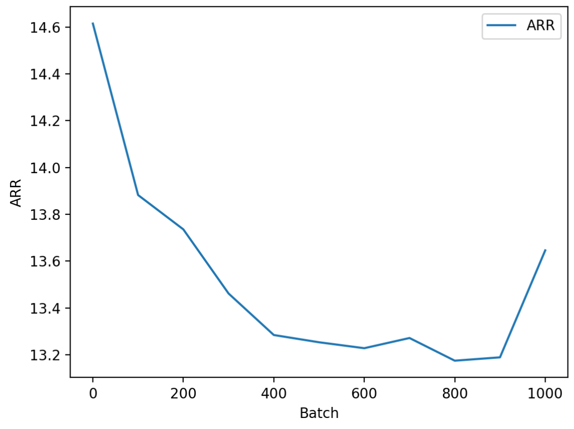

Figure 4: Evaluation ARR during training.

makes total sense as they are both measures that capture how good a ranking is. A difference between both metrics is that the NDCG has graph that moves around more, whereas the ARR shows a more steady decline. The correlation between the two metrics can be clearly seen at the end of both figures, when the NDCG goes against the trend by decreasing, and the ARR does the same by increasing. The relative increase of the NDCG is about $\frac{0.83-0.81}{0.81} = 0.025$ and the relative decrease of ARR is $\frac{14.6-13.2}{13.2} = 0.106$. This means that the ARR is relatively optimized more compared to the NDCG metric.

THEORY QUESTIONS

TQ 1.1

For the e-commerce domain, as our main goal would be ranking products for the user to purchase, we would want to incorporate product information into our vector. This can be very coarse information such as information on the category of item (clothing, electronics, ...) to more fine-grained product type information (for clothing; jeans, jackets, t-shirts). This information would allow the LTR algorithm to learn to make a distinction between different product types based on the words within a query.

TQ 1.2

Offline learning to rank are all LTR methods that makes a clear distinction between training and deployment. For this, we need labeled training data. It is very time-consuming and expensive to obtain this labeled data. Furthermore, the data that the model is deployed on needs to match the data that the model is trained on. When the type of data the model is deployed on changes, which may often happen in the fast-paced online world, the model needs to be trained again to reflect this change. Online LTR uses user-interactions to train the ranking model. This data is way easier to obtain, and the model changes over time as the data it is deployed on changes.

TQ 1.3

The main difference between learning to rank and ordinal regression / classification is that in learning to rank our goal is to obtain the perfect ranking, meaning the scores the model assigns to individual documents does not matter, as long as the way in which these documents are ranked is perfect, whereas in ordinal regression / classification we are optimising for the correct classification of single samples. This is the main reason pointwise LTR doesn't work too well, as in pointwise LTR we are optimising for individual document-query pairs instead of for the best ranking of documents. In short, in LTR the scores given to distinct documents are optimised for simultaneously, whereas in ordinal regression / classification scores given to individual documents are optimised for individually.

TQ 1.4

Most of these metric are non-differentiable, which means they don't allow for backpropagation. As our models are trained using backpropagation, we can only directly optimise for those metrics that are differentiable. ERR is non-differentiable, and therefor cannot be directly optimised for.

TQ 2.1

In pointwise ranking, the relevance of documents with respect to a given query is evaluated for each document separately. The score for each document in a ranking is thus independent of each other document in that ranking. As a ranking of documents is meant to reflect a hierarchy of information based on importance, the independence of documents can lead to mistakes by the model where a worse ranking can obtain a lower loss. Instead we will thus explore methods that do not evaluate documents independently.

Let's say we have 4 documents, $\{A, B, C, D\}$, where only A is relevant to our query, denoted by label 1, and the rest is irrelevant denoted by label 0. If our model was to assign a scores of $\{1, 0.9, 0.9, 0.9\}$ to these documents respectively, the corresponding ranking would be perfect, but we would obtain a relatively high MSE or Cross Entropy Loss. If our model assigned these documents

LR	NDCG	ERR	Hidden size	NDCG	ERR	Hidden layers	NDCG	ERR
0.00001	0.7572	0.7993	20	0.7433	0.7971	1	0.7533	0.7987
0.0001	0.7546	0.7946	50	0.7372	0.8082	2	0.7527	0.7946
0.001	0.7576	0.8121	100	0.7651	0.8078	3	0.7697	0.8037
0.01	0.7564	0.8053	200	0.7726	0.8191	4	0.7661	0.8164

Table 4: NDCG and ERR results for the listwise Learning to Rank model on the evaluation set for different hyperparameter configurations.

scores of $\{0.1, 0.2, 0.2, 0.2\}$, the ranking would not be good at all, but the MSE or Cross Entropy would be lower than for the previously assigned scores. Since we're not directly optimising for a ranking, these errors may occur.

TQ 3.1

In RankNet, the training set is partitioned on queries, and each pair of documents related to a given query is evaluated. This means that we have a complexity in terms of the documents per query n of $O(n^2)$.

TQ 3.2

Since we are now only computing s_i for every document once the complexity in terms of the documents per query ultimately drops to $O(n)$.

TQ 3.3

Let's say we have 4 documents, $\{A, B, C, D, E\}$, where only A, B are relevant to our query, denoted by label 1, and the rest is irrelevant denoted by label 0. If our model obtains the following ranking: $\{A, C, D, E, B\}$, the model gets 3 pairs correct. The ranking $\{C, A, B, D, E\}$ gets 4 pairs correct, even though the highest ranked document is not relevant to our query. The second ranking, according to the pairwise approach, is ranked higher, although this goes against our intuition that says the ranking with the highest ranked document is best. This shows that, since we are only considering pairs of documents, we are still not optimising for the final ranking.

LISTWISE LEARNING TO RANK

In order to find the best configuration for the listwise model, a search over a hyperparameter set has been implemented. The tuned hyperparameters were, LR, Hidden Size, and number of hidden layers. While one of the parameter was being tuned the other parameters were fixed, LR at 0.001, 20 neurons for hidden size, and 1 hidden layer. For the learning rate the used values were 0.0001, 0.001, 0.01, 0.1, for hidden size 20, 50, 100, 200, and for the amount of hidden layers 1, 2, 3, 4. The results for said configurations on the validation set can be found in table 4. Since we had some trouble with the spedup version of ranknet, the C computation of formula (3) from homework pdf has been implemented instead of the lambda, for the loss calculation. The best performing configuration, being hidden size 200, has been ran again and achieved an NDCG of 0.7998 and an ERR of 0.9363 on the test set.

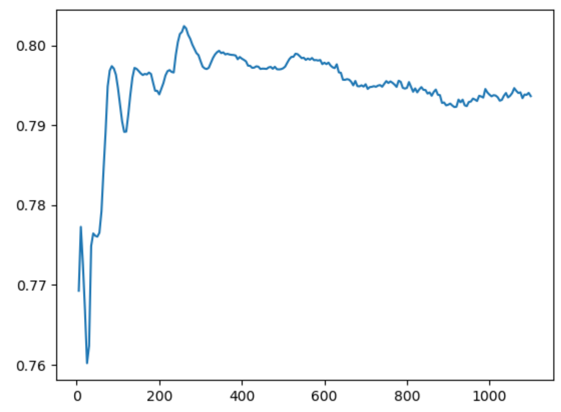

Figure 5: Evaluation NDCG during training of the listwise model.

TQ 4.1

The λ value for a document describes the gradient of the ranking to reduce the cost. A positive lambda means the document needs to move up in the ranking whereas a negative lambda means it should move down.

TQ 4.2

In our opinion, lambdarank is still a pairwise approach, since we're evaluating the cost for our model using the relation between pairs of documents in the ranking. This means that we are still not looking at the ranking as a whole to optimise, but rather we are looking at the ranking of individual pairs of documents and optimising those.

AQ 4.1

When comparing the pointwise and pairwise, the pointwise looks a lot steadier than the pairwise, this might be due to the fact that pointwise was evaluated more frequently. When comparing each of the results of pointwise and pairwise with the NDCG plot of the listwise model, as seen in figure 5, all three NDCG scores increase with each increment at the beginning and all three reach their ceiling in the first quarter of the amount of iterations needed to reach the stopping condition. In reality the listwise should have achieved the best NDCG score, after which pairwise should have come in second and pointwise as third. This is not the case because we suspect a flaw in the sped up version of ranknet, and because we had trouble getting the sped up version working we choose to use C instead of lambda for the calculation of the loss for lambdarank. When using the lambda from the sped up version we achieved even lower scores with the listwise approach.

AQ 4.2

As we can see in table 5 both pointwise and listwise perform somewhat in the same ballpark, and pairwise has an extremely high loss. The loss that is shown is the MSE loss on the test set. Somehow the results contradict the expected result: Listwise performing best,

model	NDCG
pointwise	0.6381
pairwise	4.5264e+11
listwise	1.5267

Table 5: A table containing the loss value of the differently trained models.

pairwise second, and pointwise third. This can be because the loss used to train pointwise was also the MSE loss, thus being trained on the testing loss measure. Concluding, from all results in this paper, pointwise has outperformed both competing models, against all expectations. We think this might be due to uncertainty with implementing sped-up ranknet and lambdarank from the papers.

REFERENCES