Neural Feature Selection for Learning to Rank

Alberto Purpura[1(✉)], Karolina Buchner[2], Gianmaria Silvello[1], and Gian Antonio Susto[1]

[1] University of Padua, Padua, Italy
{purpuraa,silvello,sustogia}@dei.unipd.it
[2] Apple, Cupertino, USA
kbuchner@apple.com

Abstract. LEarning TO Rank (LETOR) is a research area in the field of Information Retrieval (IR) where machine learning models are employed to rank a set of items. In the past few years, neural LETOR approaches have become a competitive alternative to traditional ones like LambdaMART. However, neural architectures performance grew proportionally to their complexity and size. This can be an obstacle for their adoption in large-scale search systems where a model size impacts latency and update time. For this reason, we propose an architecture-agnostic approach based on a neural LETOR model to reduce the size of its input by up to 60% without affecting the system performance. This approach also allows to reduce a LETOR model complexity and, therefore, its training and inference time up to 50%.

Keywords: Learning to rank · Feature selection · Deep learning

1 Introduction

LEarning TO Rank (LETOR) is a research area in the field of Information Retrieval (IR) where machine learning techniques are applied to the task of ranking a set of items [10]. The input to a LETOR system is a set of real-valued vectors representing the items to be ranked – in decreasing order of relevance – in return to a certain user query. The output of such systems is usually a set of relevance scores – one for each item in input – which estimate the relevance of each item and are used to rank them. In the recent years, the attention on neural approaches for this task has grown proportionally to their performance. Starting from [2], where the authors propose to employ a recurrent neural layer to model documents list-wise interactions, to [12], where the now popular self-attention transformer architecture is used. Also, the performance of neural models [12,

A. Purpura—Work done as part of Apple internship.
G. Silvello—Work supported by the ExaMode project, as part of the European Union Horizon 2020 program under Grant Agreement no. 825292.

20] recently became competitive with approaches such as LambdaMART [4] which is often one of the first choices for LETOR tasks. However, neural models performance grew at the expense of their complexity and this hampers their application in large-scale search systems. Indeed, in such context, model latency and update time are as important as model performance. Reducing the input size can help decreasing model architectural complexity, number of parameters, and consequently training and inference time. Also, previous works [5,6,8] showed that the document representations used for LETOR can sometimes be redundant and often reduced [6] without impacting the ranking performance.

Existing feature selection approaches can be organized into three main groups: *filter*, *embedded*, and *wrapper* methods [6][1]. Filter methods, such as the Greedy Search Algorithm (GAS) [5], compute one score for each feature – independently from the LETOR model that is going to be used afterwards – and select the top ones according to it. In GAS the authors minimize feature similarity (Kendall Tau) and maximize feature importance. They rank the input items using only one of the features at a time and consider as importance score the MAP or nDCG@k value. Embedded approaches, such as the one presented in [15], incorporate the feature selection process in the model. In [15], the authors propose to apply different types of regularizations – such as L1 norm regularization – on the weights of a neural LETOR model to reduce redundancy in the hidden representations of the model and improve its performance. Finally, wrapper methods such as the ones presented in [6] and the proposed approach, rely on a LETOR model to estimate feature importance and then perform a selection.

We reimplemented the two best-performing approaches proposed in [6] and consider them as our baselines: eXtended naive Greedy search Algorithm for feature Selection (XGAS) – which relies on LambdaMART to estimate feature relevance – and Hierarchical agglomerative Clustering Algorithm for feature Selection (HCAS) employing single likage [7] – which relies on Spearman's correlation coefficient between feature pairs as a proxy for feature importance. To the best of our knowledge, our approach is the first feature selection technique for LETOR specifically targeted to neural models. The main contributions of this paper are the following:

– we propose an architecture-agnostic Neural Feature Selection (NFS) approach which uses a neural LETOR model to estimate feature importance;
– we evaluate the quality of our approach on two public LETOR collections;
– we confirm the robustness of the extracted feature set evaluating the performance of the proposed neural reranker and of a LambdaMART model using subsets of features of different sizes computed with the proposed approach.

Our experimental results show that the document representations used for LETOR can sometimes be redundant and reduced to up to 40% [6] of the total without impacting the ranking performance.

[1] We purposely omit a comparison with other dimensionality reduction approaches such as PCA since these methods often compute a *combination* of the features to reduce the representation size which is beyond the scope of this paper.

2 Proposed Approach

The proposed Neural Feature Selection (NFS) approach is organized in the following three steps. We first train a neural model for the LETOR task, i.e. to compute a relevance score for each item in the input set to be used to rank it. Second, we use the trained model to extract the most significant features groups considered by the model to rank each item. Finally, we perform feature selection using the previously computed feature information.

Neural Model Training. The NFS model architecture is composed of n self-attention layers [19], followed by two fully-connected layers. We train this model using the ApproxNDCG loss [3]. Before feeding the document vectors to the self-attention layer we apply the same feature transformation strategy described in [20]. In [20], the authors apply three different feature transformations to each feature in the input data and then combine them through a weighted sum. The weights for each transformation are learned by the model so that the best feature transformation strategy for each feature could be used each time. The model architecture is depicted in Fig. 1. Also, we apply batch normalization to the input of each feed-forward layer and dropout on the output of each hidden layer. Note that, since our approach for feature selection is *architecture-agnostic*, we can easily make changes to this neural architecture without impacting the following steps for feature selection.

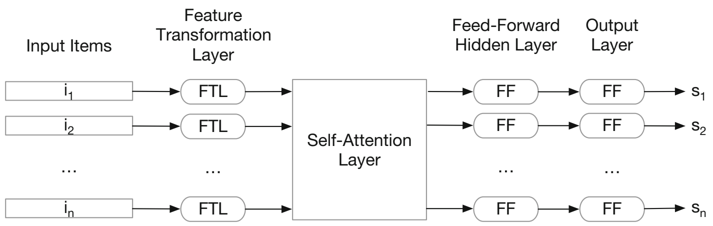

Fig. 1. Architecture of the neural architecture employed in our evaluation.

Feature Groups Mining. At this step, we use the model trained in the previous step to select the most important features used to rank each item in our training data. To do so, we compute the saliency map – a popular approach in the computer vision field to understand model predictions [1,16,17] – i.e. the gradient w.r.t. the each input item feature, corresponding to each item in the training dataset. We then apply min-max normalization on each saliency map M_i to map the values in each vector to the same range $[0, 1]$. Afterwards, we select from each saliency map the groups of features g which have a *saliency* score higher than a threshold t. The set of feature groups G extracted at this step are the most significant features sets that our neural model learned to rely on to compute the relevance score of each item. These features however might not be the same for any possible input instance and – as also pointed out in [1]

– saliency maps can often be noisy and not always represent the behavior of a neural model. For this reason, we propose to apply a further selection step to prune less reliable feature groups similarly to what proposed in [18] where the authors compute the statistical significance of groups of items by comparing their frequency of occurrence in real data to the one in randomly generated datasets. We compute K random sets of saliency maps, each of the same cardinality of the experimental dataset employed. For example, if a dataset contains N queries, each with R documents to be ranked, then we will generate K random datasets, each containing $N \times R$ saliency maps. Then, we apply the same feature groups extraction process on the random saliency maps and compute K different sets of feature groups. The saliency maps are computed sampling values from a uniform distribution with support $[0, 1]$. According to this modeling strategy, each feature can be considered as salient in the current random saliency map with probability $1 - t$; where t is the threshold we used in the previous step to select salient features. Once we computed these K sets of random feature groups $\hat{G}_k$ we use their frequency to prune the original ones. In particular, we consider the frequency f_{g_i} of group $g_i \in G$ and compare it to its frequency in each of the K random datasets $f_{g_{i,k}}$ – the frequency $f_{g_{i,k}}$ might also be 0 if the feature group g_i does not appear in the random dataset k. If $f_{g_i} \leq f_{g_{i,k}}$ in more than 2%[2] of the randomly generated feature groups $\hat{G}_k$, we discard feature group g_i, considering it as noise.

Feature Selection. In this final step, we rely on the feature groups extracted in the previous step and their frequency in the saliency maps to compute a feature similarity matrix. We then use this similarity matrix to perform feature selection. Each feature pair similarity value is computed counting the times the two features appear in the same feature group and normalizing that score by the total number of groups where that feature appears. Finally, we rely on this similarity matrix to perform hierarchical clustering as done in [6]. We consider the number of clusters as the stopping criterion for the single linkage hierarchical clustering algorithm. The final set of features to keep is computed selecting the most frequently occurring feature in the previously computed feature groups, from each feature cluster.

3 Experimental Setup

We evaluate our approach on the first fold of the MSLR-WEB30K [13] and on the whole OHSUMED [14] dataset where the items to rank are represented by 136 and 45 features, respectively[3]. We use the LambdaMART implementation available in the LightGBM[4] library [9] and train and test the proposed neural

[2] This value was set empirically to yield a reasonable number of feature groups for the following feature extraction step.

[3] https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval.

[4] https://github.com/microsoft/LightGBM.

model considering only the top 128 results returned by LambdaMART. We tuned the LightGBM model parameters on the validation sets of both datasets, optimizing the ndcg@3 metric[5]. The proposed neural reranking model is trained for 500 epochs – 100 epochs on the OHSUMED dataset – with batch size 128, using Adam optimizer and a learning rate 0.0005. We consider a feature embedding size of 128 in the feature transformation layer on the MSLR-WEB30K dataset – while we removed it for the experiments on the OHSUMED collection due to its much smaller size and number of features which limited the benefits of it – 4 self-attention heads on the MSLR-WEB30K and 1 on the OHSUMED dataset and a hidden size of 128 for the hidden feed-forward layer. Since each attention head has an output size equal to the total number of features divided by the number of attention heads, to compute the results reported in Table 1, we reduce the number of attention heads to 1 when using 5% and 10% of all the available features (6 and 13 features respectively), we use 4 attention heads when considering 30% (27 features), and 3 when using 40% (54 features). The batch normalization momentum we use is 0.4 and the dropout probability is $p = 0.5$. In the feature groups mining step, we generate 5000 random datasets and the threshold t to extract the feature groups is empirically set to 0.95. For the evaluation of the approach we consider the nDCG@3 measure, similar results are obtained with nDCG at different cutoffs.

4 Experimental Results

In Table 1, we report the results of our experiments on the MSLR-WEB30K dataset. We trained both a LambdaMART model and the proposed neural reranking one on different subsets of features of increasing size. From these experiments, we observe that the proposed Neural Feature Selection (NFS) approach always outperforms all the other baselines when the selected features are used to train a LambdaMART model, and in most of the cases when used with the proposed neural model. The evaluation results on the OHSUMED dataset reported in Table 2 are computed as the previous case. Here, we consider 60%, 70%, 80%, and 90% of the total features in the collection since the total number of feature is much smaller than in the previous dataset. In our evaluation, NFS outperforms HCAS in the majority of the cases, even though the latter approach is slightly more competitive than before.

The main advantage of using a subset of features to represent the inputs to a neural model is that we can reduce the model complexity. We observe this effect mainly when our data is represented by a large number of features as in the MSLR-WEB30K collection. For example, when using 40% of the features of the dataset, the number of attention heads in our model was reduced from 4 to 3 and, since we were considering only 54 out of 136 features, the number of parameters of the self-attention heads – the first layer of our model – was also

[5] We set the learning rate to 0.05, the number of leaves to 200 and the number of trees to 1000 (500) on the MSLR-WEB30K (OHSUMED) collection.

Table 1. Evaluation of the proposed Neural Feature Selection (NFS) approach on the MSLR-WEB30K dataset. We report the ndcg@3 values obtained by LambdaMART and the proposed Neural Reranking model employing different subsets of features.

LambdaMART				Neural reranker		
Features Perc.	XGAS	HCAS (single)	NFS	XGAS	HCAS (single)	NFS
5%	0.3580	0.3589	**0.3753**	**0.3768**	0.3595	0.3749
10%	0.3701	0.4044	**0.4195**	0.3826	0.3923	**0.4117**
20%	0.3781	**0.4672**	**0.4672**	0.3831	**0.4444**	0.4434
30%	0.4169	0.4655	**0.4713**	0.4085	**0.4478**	0.4236
40%	0.4387	0.4709	**0.4730**	0.3943	0.4516	**0.4559**
100%	0.4731	0.4731	0.4731	0.4526	0.4526	0.4526

reduced. As a consequence, training time was halved and inference time also decreased.

Table 2. Evaluation of the proposed Neural Feature Selection (NFS) approach on the OHSUMED dataset. We report the ndcg@3 values obtained by LambdaMART and the proposed Neural Reranking model employing different subsets of features.

LambdaMART				Neural reranker		
Features Perc.	XGAS	HCAS (single)	NFS	XGAS	HCAS (single)	NFS
60%	0.3669	0.3781	**0.3950**	0.4210	**0.4275**	0.4242
70%	0.3669	0.3781	**0.3860**	0.4243	0.4431	**0.4437**
80%	0.3669	0.3993	**0.4007**	0.4374	**0.4369**	0.4205
90%	0.3669	**0.4050**	0.3959	0.3669	0.4050	**0.4221**
100%	0.3968	0.3968	0.3968	0.4973	0.4973	0.4973

It is also interesting to observe the differences between the features selected by the proposed NFS approach and other baselines. We focus on the top 3 features selected from the OHSUMED collection by each of the considered feature selection algorithms over the 5 different dataset folds and refer the reader to [14] for a more detailed description of each feature. NFS most frequently selected features computed with popular retrieval models such as BM25 or QLM [11] (features 4, 12 and 28) based on the document abstract or title. On the other hand, HCAS selected simpler features derived from raw frequency counts of the query terms in each document's title and abstract (features 23, 40 and 36). Finally, XGAS selected a mix of features computed with traditional retrieval approaches such as QL, and simpler frequency counts (features 2, 44 and 13). We conclude that the advantage of NFS is likely due to its ability to recognize and select the most sophisticated and useful matching scores thanks to the information learned during training.

5 Conclusions

In the recent years, neural models became a competitive alternative to traditional Learning TO Rank (LETOR) approaches. Their performance however, grew at the expense of their efficiency and complexity. In this paper, we propose an approach for feature selection for Learning TO Rank (LETOR) based on a neural ranker. Our approach is specifically designed to optimize the performance of neural LETOR models without the need to change their architecture. In our experiments, the proposed approach improved the efficiency of a sample neural LETOR model and decreased its training time without impacting its performance. We also validated the robustness of the selected features testing them using a different – non neural – model such as LambdaMART. We performed our evaluation on two popular LETOR datasets – i.e. MSLR-WEB30K and OHSUMED – comparing our approach to three state-of-the-art techniques from [6]. The proposed approach outperformed the selected baselines in the majority of the experiments on both datasets.

References

1. Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., Kim, B.: Sanity checks for saliency maps. In: Advances in Neural Information Processing Systems, pp. 9505–9515 (2018)
2. Ai, Q., Bi, K., Guo, J., Croft, W.: Learning a deep listwise context model for ranking refinement. Proc. SIGIR **2018**, 135–144 (2018)
3. Bruch, S., Zoghi, M., Bendersky, M., Najork, M.: Revisiting approximate metric optimization in the age of deep neural networks. Proc. SIGIR **2019**, 1241–1244 (2019)
4. Burges, C.J.: From ranknet to lambdarank to lambdamart: an overview. Learning **11**(23–581), 81 (2010)
5. Geng, X., Liu, T., Qin, T., Li, H.: Feature selection for ranking. Proc. SIGIR **2007**, 407–414 (2007)
6. Gigli, A., Lucchese, C., Nardini, F., Perego, R.: Fast feature selection for learning to rank. Proc. ICTIR **2016**, 167–170 (2016)
7. Gower, J.C., Ross, G.: Minimum spanning trees and single linkage cluster analysis. J. Royal Stat. Soc. Ser. C (Appl. Stat.) **18**(1), 54–64 (1969)
8. Han, X., Lei, S.: Feature selection and model comparison on microsoft learning-to-rank data sets. arXiv preprint arXiv:1803.05127 (2018)
9. Ke, G., et al.: Lightgbm: a highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems, pp. 3146–3154 (2017)
10. Liu, T.: Learning to rank for information retrieval. Springer Science & Business Media (2011)
11. Manning, C., Schütze, H., Raghavan, P.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
12. Pobrotyn, P., Bartczak, T., Synowiec, M., Białobrzeski, R., Bojar, J.: Context-aware learning to rank with self-attention. arXiv preprint arXiv:2005.10084 (2020)
13. Qin, T., Liu, T.: Introducing letor 4.0 datasets. arXiv preprint arXiv:1306.2597 (2013)

14. Qin, T., Liu, T.Y., Xu, J., Li, H.: Letor: a benchmark collection for research on learning to rank for information retrieval. Inf. Retrieval **13**(4), 346–374 (2010)
15. Rahangdale, A., Raut, S.: Deep neural network regularization for feature selection in learning-to-rank. IEEE Access **7**, 53988–54006 (2019)
16. Shrikumar, A., Greenside, P., Shcherbina, A., Kundaje, A.: Not just a black box: learning important features through propagating activation differences. arXiv preprint arXiv:1605.01713 (2016)
17. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034 (2013)
18. Tonon, A., Vandin, F.: Permutation strategies for mining significant sequential patterns. In: Proceedings of ICDM 2019, pp. 1330–1335. IEEE (2019)
19. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
20. Zhuang, H., Wang, X., Bendersky, M., Najork, M.: Feature transformation for neural ranking models. In: Proceedings of SIGIR 2020 (2020)