Feature Transformation for Neural Ranking Models

Honglei Zhuang, Xuanhui Wang, Michael Bendersky, Marc Najork
{hlz,xuanhui,bemike,najork}@google.com
Google Research

ABSTRACT

Although neural network models enjoy tremendous advantages in handling image and text data, tree-based models still remain competitive for learning-to-rank tasks with numerical data. A major strength of tree-based ranking models is the insensitivity to different feature scales, while neural ranking models may suffer from features with varying scales or skewed distributions. Feature transformation or normalization is a simple technique which preprocesses input features to mitigate their potential adverse impact on neural models. However, due to lack of studies, it is unclear to what extent feature transformation can benefit neural ranking models. In this paper, we aim to answer this question by providing empirical evidence for learning-to-rank tasks. First, we present a list of commonly used feature transformation techniques and perform a comparative study on multiple learning-to-rank data sets. Then we propose a mixture feature transformation mechanism which can automatically derive a mixture of basic feature transformation functions to achieve the optimal performance. Our experiments show that applying feature transformation can substantially improve the performance of neural ranking models compared to directly using the raw features. In addition, the proposed mixture transformation method can further improve the performance of the ranking model without any additional human effort.

CCS CONCEPTS

• **Information systems → Learning to rank**.

KEYWORDS

Feature transformation; neural ranking model; learning-to-rank

ACM Reference Format:
Honglei Zhuang, Xuanhui Wang, Michael Bendersky, Marc Najork. 2020. Feature Transformation for Neural Ranking Models. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3397271.3401333

1 INTRODUCTION

Many machine learning models cannot easily handle features with drastically skewed distributions or extreme scales. Unfortunately, commonly used features often have these patterns. Particularly in web-related applications such as search and recommendation tasks, data items are often represented by features with a peculiar statistical nature. For example, click counts can be in the scale of billions for some items while other items only have a dozen clicks. Using these numbers directly as input features can result in less optimal models and introduce numerical instability during training. Therefore, feature normalization or transformation is an essential step to improve the effectiveness of many machine learning models [13].

In the learning-to-rank setting, tree-based models [6, 11, 17] have been extensively studied in the past and remain competitive on public data sets which primarily consist of numerical features. The tree-based model architecture is generally immune to the adverse impact of directly using raw features. Recently, neural network based deep learning models attract lots of attention for learning-to-rank tasks [1, 5]. However, few of them investigate the impact of feature transformation. A possible reason is that neural ranking models are regarded as universal function approximators [14], which leads to the misconception that the optimal ranking function can be automatically learned by current algorithms without feature transformation. Therefore, it is still unclear whether feature transformation is important for neural ranking models.

While there are surveys providing empirical comparison of multiple feature transformation techniques in other domains [2, 19, 24], to the best of our knowledge there is no systematic comparison of feature transformation techniques for neural ranking models. Moreover, the optimal transformation could vary by feature due to different statistical nature. Possible solutions are either to manually specify feature transformation technique for each feature based on expert knowledge or to perform tedious empirical comparison for every single feature. Both practices require significant time and effort and become intractable when the number of features is large.

In this paper, we aim to evaluate the importance of feature transformation for neural ranking models. We first enumerate three basic feature transformation functions, including commonly used z-score and CDF transformations, and a simple yet effective symmetric logarithm transformation that has been less explored in prior work on ranking. Then, we propose a mixture feature transformation mechanism. In particular, it takes the set of basic feature transformation functions and automatically derives a mixture of them as the final transformation for each feature. The mixture feature transformation module can be jointly trained with the neural ranking model, which saves the human effort to choose transformation functions for different features.

We perform experiments on multiple learning-to-rank data sets to provide a thorough view of the effects of different feature transformation functions. We show that applying feature transformation can significantly improve the performance of neural ranking models compared to directly using the raw feature values. We also show that our mixture feature transformation can further improve the performance by selecting a proper transformation per feature without any human effort.

SIGIR '20, July 25–30, 2020, Virtual Event, China

ACM ISBN 978-1-4503-8016-4/20/07.
https://doi.org/10.1145/3397271.3401333

2 RELATED WORK

Feature scaling, normalization and transformation is one of the most fundamental data preprocessing techniques in machine learning [12, 13]. There are surveys comparing different feature transformation techniques in various applications, such as disease detection [24], stock market prediction [19] and video classification [2].

In information retrieval, feature transformation is also a common practice. For example, the YAHOO Learning to Rank Challenge data set [8] applies cumulative distribution-based transformation on all features; the LETOR [23] data set also applies query-level min-max scaling on each feature. In traditional information retrieval, feature transformation has been extensively studied on both term frequency and inverse document frequency (e.g., BM25). However, such a study is still missing for neural ranking models.

Automatically learning to perform feature transformation is a relatively novel topic. There are only a few studies with similar objectives [3, 7]. Their methods focus on a family of functions such as linear or logistic ones and are not studied for ranking models. In contrast, our methods focus on neural ranking models and work with different forms of transformation functions.

3 PROBLEM FORMULATION

Learning to rank. We represent a learning-to-rank data set with N lists as $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}_1^N$. For a list $(\mathbf{X}, \mathbf{y}) \in \mathcal{D}, \mathbf{X} = \{\mathbf{x}_i\}_{i=1}^l$ denotes a set of l data items, each represented by an n-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^n$; $\mathbf{y} = \{y_i\}_{i=1}^l$ are relevance labels of corresponding data items, where $y_i \in \mathbb{R}$. We also use the notation $\mathcal{X}$ to represent all the data items in the entire data set, namely $\mathcal{X} = \bigcup_{\mathbf{X} \in \mathcal{D}} \mathbf{X}$.

In a supervised learning-to-rank setting, we are usually given a training data set $\mathcal{D}_L$, where all the relevance labels $\mathbf{y}$ are known. We aim to develop a scoring function $f : \mathbb{R}^n \mapsto \mathbb{R}$ such that for any list $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^l$, the ranked list predicted by scoring and sorting items based on $f(\mathbf{x}_i)$ can be similar to the ground-truth ranked list obtained by sorting items based on y_i.

Feature transformation. We denote the feature transformation function for the k-th feature as $\sigma_k : \mathbb{R} \mapsto \mathbb{R}$. For a feature vector $\mathbf{x}_i = (x_{i1}, x_{i2}, \cdots, x_{in})$, we apply the feature transformation function on each feature vector and obtain the transformed feature vector as $\sigma(\mathbf{x}_i) = (\sigma_1(x_{i1}), \sigma_2(x_{i2}), \cdots, \sigma_n(x_{in}))$. The scoring function f in the neural ranking model will be trained and evaluated based on the transformed feature vectors, i.e., the predicted ranked list will be obtained by sorting items based on $f(\sigma(\mathbf{x}_i))$.

4 FEATURE TRANSFORMATION

In this section, we first introduce several basic feature transformation techniques for single features. Then, we introduce a mixture feature transformation mechanism to automatically derive a mixture of multiple transformed values for each feature.

4.1 Basic Transformation

Gaussian transformation (z-score). We fit a Gaussian distribution for all the data items $\mathbf{x}_i \in \mathcal{X}$ in the given data set. We denote each data item as $\mathbf{x}_i = (x_{i1}, \cdots, x_{in})$. For the k-th feature, we can estimate the mean and standard deviation by:

$$\hat{\mu}_k = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}_i \in \mathcal{X}} x_{ik}, \quad \hat{s}_k = \sqrt{\frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}_i \in \mathcal{X}} (x_{ik} - \hat{\mu}_k)^2}$$

And the derived transformation function is

$$\sigma_{\text{Gauss},k}(x) = \frac{x - \hat{\mu}_k}{\hat{s}_k} \tag{1}$$

We denote the vector containing all transformed features as $\sigma_{\text{Gauss}}(\mathbf{x}_i) = (\sigma_{\text{Gauss},1}(x_{i1}), \cdots, \sigma_{\text{Gauss},n}(x_{in}))$. We will use similar notation for other feature transformation techniques.

CDF transformation. Another idea is to estimate the cumulative distribution function (CDF) for each feature and use the CDF value to represent the feature. Concretely, we still use $\mathcal{X}$ as the set of all data items in the data set. For any value x of the k-th feature, the transformation function is:

$$\sigma_{\text{CDF},k}(x) = \frac{\sum_{\mathbf{x}_i \in \mathcal{X}} \mathbb{I}(x_{ik} < x)}{|\mathcal{X}|} \tag{2}$$

Symmetric log1p transformation. We also use a symmetric logarithm function as a transformation function. A regular $\log(x)$ function cannot be directly applied to any function as it is undefined for $x \leq 0$. Moreover, when x is close to 0, the absolute value of the transformed feature could be extremely large. Hence, we use a symmetric version of $\log(1 + x)$:

$$\sigma_{\text{Log1p},k}(x) = \text{sgn}(x) \cdot \log(1 + |x|) \tag{3}$$

where $\text{sgn}(x)$ is the sign function which returns 1 for positive numbers, -1 for negative numbers and 0 for $x = 0$.

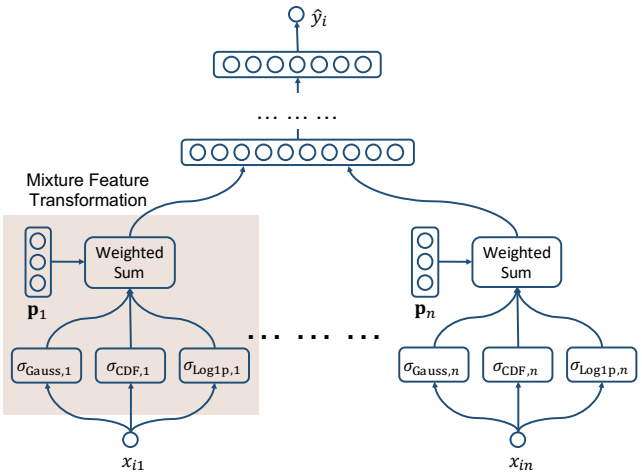

Figure 1: An illustrative example of mixture feature transformation module (shaded) in a neural ranking model.

4.2 Mixture Transformation

There may not be a single optimal feature transformation technique applicable for all scenarios. The effect of different feature transformation techniques depends on both the overall feature value distribution and the model structure. Hence, we introduce a mixture feature transformation module.

Suppose for the k-th feature, we have m feature transformation functions available as a "basis", denoted as $\{\sigma_{1,k}, \ldots, \sigma_{m,k}\}$. We

derive the mixture transformed feature value of the k-th feature $\sigma_k(x)$ as:

$$\sigma_{\text{Mixture},k}(x) = \sum_m p_{m,k}\sigma_{m,k}(x) \tag{4}$$

where $\mathbf{p}_k = (p_{1,k}, \cdots, p_{m,k})$ is a weighting vector. We derive the weighting vector from an embedding of each feature by

$$\mathbf{p}_k = \text{softmax}(\mathbf{W}\mathbf{e}_k)$$

where $\mathbf{e}_k$ is the d-dimensional embedding vector of the k-th feature and $\mathbf{W}$ is an $m \times d$ matrix to be learned. Notice that $\mathbf{e}_k$ is not related to any feature value x_{ik}, but rather an embedding vector of k-th feature per se. There are only n such embedding vectors where n is the number of features.

Note that the mixture feature transformation layer can be jointly trained with the ranking model to automatically determine the best mixture of feature transformations for each feature.

4.3 Ranking Model

The ranking model we employ is a feed-forward network where the first layer takes the transformed feature as input. Formally, for a data item $\mathbf{x}$, we can derive the final ranking score $\hat{y}$ by:

$$\begin{aligned}
\mathbf{z}_1 &= \text{ReLU}(\mathbf{W}_1\sigma_*(\mathbf{x}) + \mathbf{b}_1) \\
\mathbf{z}_2 &= \text{ReLU}(\mathbf{W}_2\mathbf{z}_1 + \mathbf{b}_2) \\
&\cdots \\
\mathbf{z}_H &= \text{ReLU}(\mathbf{W}_H\mathbf{z}_{(H-1)} + \mathbf{b}_H) \\
\hat{y} &= \mathbf{W}_{H+1}\mathbf{z}_H + b_{H+1}
\end{aligned}$$

where $\mathbf{z}_h$ is the output of the h-th hidden layer; $\mathbf{W}_h$ and $\mathbf{b}_h$ are weight matrix and bias vector of the h-th hidden layer to be trained; $\text{ReLU}(\cdot)$ refers to the Rectifier [18] activation function; and $\hat{y}$ is the final output. Notice that $\sigma_*(\mathbf{x})$ is the transformed input vector, where we can plug in any feature transformation methods mentioned above. For basic feature transformations such as CDF and Log1p, the transformed value $\sigma_{\text{CDF}}(\mathbf{x})$ or $\sigma_{\text{Log1p}}(\mathbf{x})$ will be fed into the model. For the mixture feature transformation, the module will be jointly trained with the entire model, as shown in Figure 1.

We insert a batch normalization [15] layer in front of the input of each layer. We also apply a dropout layer for the output of each hidden layer. The model is trained with an approximate NDCG_5 loss [5, 22] with a stochastic treatment as described in [4].

5 EXPERIMENTS

5.1 Data Sets

We use two public learning-to-rank data sets with numerical features in our experiments. Another well-known public data set from YAHOO [8] is not used as its feature values are already transformed.

WEB30K. WEB30K [21] is a public learning-to-rank data set released by Microsoft. The original data contains 5 folds. We only use Fold1 of the data. The data set is partitioned into three subsets: training, validation and testing. Each document is represented by 136 numerical features. In addition, a label with a 5-level relevance grade varying from 0 to 4 is provided for each document.

ISTELLA. Istella released several learning-to-rank data sets to the public. We use the Istella full data set [9]. Each document is represented by 220 numerical features. The data set is also labeled with graded relevance judgments from 0 to 4. We remove 2 features which contain illegitimate values and only use the remaining 218.

Table 1: Performance comparison (%). The best performances are bolded. Performances statistically significantly better ($\alpha = 0.05$) than Raw or Log1p are marked with † and ‡ respectively.

Data set	Method	NDCG_1	NDCG_5	NDCG_{10}
WEB30K	Raw	45.43	45.25	47.26
	Gauss	48.18†	46.81†	48.68†
	CDF	49.17†	47.75†	49.45†
	Log1p	49.12†	48.01†	49.88†
	Mixture	**50.55**†‡	**48.54**†‡	**50.16**†‡
ISTELLA	Raw	65.81	62.32	67.09
	Gauss	66.31	62.41	67.22
	CDF	65.94	62.62†	67.55†
	Log1p	66.60†	63.29†	68.12†
	Mixture	**66.91**†	**63.57**†‡	**68.42**†‡

5.2 Parameter Configurations

In all of our experiments, the ranking model network contains 3 hidden layers with 1024, 512, 256 units respectively. We set the momentum of the batch normalization layers as 0.4 and the dropout rate as 0.5. For all experiments, we set the training batch size to 128 and run for 100,000 steps. We use AdaGrad [10] as the optimizer and tune the learning rate on each data set. The learning rate is set to 0.5 for WEB30K and 0.1 for ISTELLA respectively. The experiments are implemented based on the open-sourced TensorFlow Ranking library [20] and trained on TPU.

5.3 Comparison Settings

We apply each basic feature transformation and compare the performance to the practice of directly using raw input features (represented by "Raw"). We also include the performance of applying the mixture feature transformation function into the comparison. For each feature x_{ik}, the mixture transformation takes the raw feature value along with all the three transformed feature values $\{x_{ik}, \sigma_{\text{Gauss},k}(x_{ik}), \sigma_{\text{CDF},k}(x_{ik}), \sigma_{\text{Log1p},k}(x_{ik})\}$ as the basis and generates the mixture transformation.

We adopt *normalized discounted cumulative gain* (NDCG) [16] as the evaluation metric. In our experiments, we utilize NDCG_k to measure the quality of the top-k ranked items where $k \in \{1, 5, 10\}$.

5.4 Results

The overall results are shown in Figure 1. As one can observe, applying feature transformation techniques to neural ranking models substantially improves the performance, varying from +1% to +3% (NDCG_5) on both data sets. This verifies the importance of utilizing feature transformation methods for neural ranking models.

Among all the basic feature transformation methods, Log1p seems to outperform the other two transformation methods. We believe that Log1p is especially effective to transform features with a power law distribution, which are prevalent in web-related data sets like WEB30K and ISTELLA. The distribution of the transformed feature values will no longer be as skewed as raw feature values,

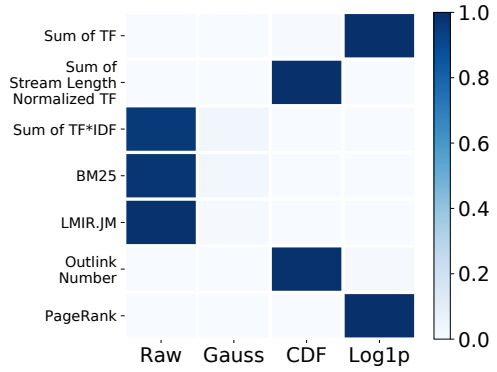

Figure 2: Visualization of the mixture weights $\mathbf{p}_k$ of a selected set of features derived by Mixture method on WEB30K data set. Each row represents a feature, and each column corresponds to a basic feature transformation.

allowing neural ranking models to better handle these features in training.

Another observation is that the Mixture method outperforms all the basic feature transformation methods on both data sets. On the WEB30K data set, the Mixture method improves NDCG_1 by more than +1% compared to the best basic feature transformation method. It shows the effectiveness of the proposed mixture mechanism to automatically derive a better transformation function for each feature from the basic transformation functions.

In-depth analysis of the Mixture feature transformation. We also take a deeper look into the mixture weights derived by the mixture feature transformation. We select a few features in WEB30K and plot their learned weighting vector $\mathbf{p}_k$. By default we select the feature for "body". We visualize these weighting vectors in Figure 2.

First, for some features that are sophisticated ranking scores per se (e.g., Sum of TF*IDF, BM25 and LMIR.DIR), the model only uses their raw scores, suggesting no further transformation is needed. For features with potentially extremely large values and/or power law distribution (e.g., Sum of TF and PageRank), the model learns that applying Log1p transformation methods is beneficial. The mixture method also applies CDF transformation for some features with smaller values (e.g. Sum of Stream Length Normalized TF) or sparse distributions (e.g., Outlink Number).

6 CONCLUSIONS

In this paper, we study the effect of feature transformation for neural ranking models. Through empirical evidence on multiple learning-to-rank data sets, we show that appropriate input feature transformation is crucial to improve the performance of neural ranking models. More importantly, we propose a mixture feature transformation model that can automatically select the optimal transformation given a set of basic transformation functions.

Based on our study, a further examination of feature transformations in the context of neural ranking models can be appropriate. Although our experiments are only conducted on a feed-forward network ranking models, as the focus of this paper is on feature transformation, the techniques studied and proposed in this work are applicable to any neural ranking models. It would be interesting to see whether they can achieve better performances with feature transformations. Moreover, our current work mainly looks into numerical features. It is also intriguing to study how to conduct effective feature transformation for embedding-based features (e.g., text features) in the future.

REFERENCES

[1] Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. 2018. Learning a deep listwise context model for ranking refinement. In *41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 135–144.

[2] Ashikin Ali and Norhalina Senan. 2017. The effect of normalization in violence video classification performance. *IOP Conference Series: Materials Science and Engineering* 226, 1 (2017), 012082.

[3] Danushka Bollegala. 2017. Dynamic feature scaling for online learning of binary classifiers. *Knowledge-Based Systems* 129 (2017), 97–105.

[4] Sebastian Bruch, Shuguang Han, Michael Bendersky, and Marc Najork. 2020. A stochastic treatment of learning to rank scoring functions. In *13th International Conference on Web Search and Data Mining*. 61–69.

[5] Sebastian Bruch, Masrour Zoghi, Mike Bendersky, and Marc Najork. 2019. Revisiting approximate metric optimization in the age of deep neural networks. In *42nd International ACM SIGIR Conference on Research & Development in Information Retrieval*. 1241–1244.

[6] Christopher J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An overview*. Technical Report MSR-TR-2010-82. Microsoft Research.

[7] Xi Hang Cao, Ivan Stojkovic, and Zoran Obradovic. 2016. A robust data scaling algorithm to improve classification accuracies in biomedical data. *BMC Bioinformatics* 17, 1 (2016), 359.

[8] Olivier Chapelle and Yi Chang. 2011. Yahoo! Learning to rank challenge overview. In *Proceedings of the Learning to Rank Challenge*. 1–24.

[9] Domenico Dato, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2016. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. *ACM Transactions on Information Systems* 35, 2 (2016), 1–31.

[10] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[11] Jerome H Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics* 29, 5 (2001), 1189–1232.

[12] Salvador García, Julián Luengo, and Francisco Herrera. 2015. *Data Preprocessing in Data Mining*. Springer.

[13] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data Mining: Concepts and Techniques*. Elsevier.

[14] K. Hornik, M. Stinchcombe, and H. White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 5 (7 1989), 359–366.

[15] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *32nd International Conference on International Conference on Machine Learning*. 448–456.

[16] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.

[17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. 3146–3154.

[18] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted Boltzmann machines. In *27th International Conference on Machine Learning*. 807–814.

[19] Jiaqi Pan, Yan Zhuang, and Simon Fong. 2016. The impact of data normalization on stock market prediction: Using SVM and technical indicators. In *International Conference on Soft Computing in Data Science*. Springer, 72–88.

[20] Rama Kumar Pasumarthi, Sebastian Bruch, Xuanhui Wang, Cheng Li, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. 2019. TF-Ranking: Scalable tensorflow library for learning-to-rank. In *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2970–2978.

[21] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. arXiv:cs.IR/1306.2597

[22] Tao Qin, Tie-Yan Liu, and Hang Li. 2010. A general approximation framework for direct optimization of information retrieval measures. *Information Retrieval* 13, 4 (2010), 375–397.

[23] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. 2010. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval* 13, 4 (2010), 346–374.

[24] Bikesh Kumar Singh, Kesari Verma, and A. S. Thoke. 2015. Investigations on impact of feature normalization techniques on classifier's performance in breast tumor classification. *International Journal of Computer Applications* 116, 19 (2015), 11–15.