Interpretable Ranking with Generalized Additive Models

Honglei Zhuang, Xuanhui Wang, Michael Bendersky, Alexander Grushetsky, Yonghui Wu, Petr Mitrichev, Ethan Sterling, Nathan Bell, Walker Ravina, Hai Qian
{hlz,xuanhui,bemike,grushetsky,yonghui,petya,esterling,nathanbell,walkerravina,hqian}@google.com
Google

ABSTRACT

Interpretability of ranking models is a crucial yet relatively underexamined research area. Recent progress on this area largely focuses on generating *post-hoc* explanations for existing black-box ranking models. Though promising, such post-hoc methods cannot provide sufficiently accurate explanations in general [51], which makes them infeasible in many high-stakes scenarios, especially the ones with legal or policy constraints. Thus, building an *intrinsically interpretable* ranking model with transparent, self-explainable structure becomes necessary, but this remains less explored in the learning-to-rank setting.

In this paper, we lay the groundwork for intrinsically interpretable learning-to-rank by introducing generalized additive models (GAMs) into ranking tasks. Generalized additive models (GAMs) are intrinsically interpretable machine learning models and have been extensively studied on regression and classification tasks. We study how to extend GAMs into ranking models which can handle both item-level and list-level features and propose a novel formulation of *ranking GAMs*. To instantiate ranking GAMs, we employ neural networks instead of traditional splines or regression trees. We also show that our neural ranking GAMs can be distilled into a set of simple and compact piece-wise linear functions that are much more efficient to evaluate with little accuracy loss. We conduct experiments on three data sets and show that our proposed neural ranking GAMs can outperform other traditional GAM baselines while maintaining similar interpretability.

CCS CONCEPTS

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

KEYWORDS

Generalized additive models; learning to rank; interpretable ranking models

ACM Reference Format:
Honglei Zhuang, Xuanhui Wang, Michael Bendersky, Alexander Grushetsky, Yonghui Wu, Petr Mitrichev, Ethan Sterling, Nathan Bell, Walker Ravina, Hai Qian. 2021. Interpretable Ranking with Generalized Additive Models. In *Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining (WSDM '21), March 8–12, 2021, Virtual Event, Israel.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3437963.3441796

WSDM '21, March 8–12, 2021, Virtual Event, Israel

ACM ISBN 978-1-4503-8297-7/21/03.
https://doi.org/10.1145/3437963.3441796

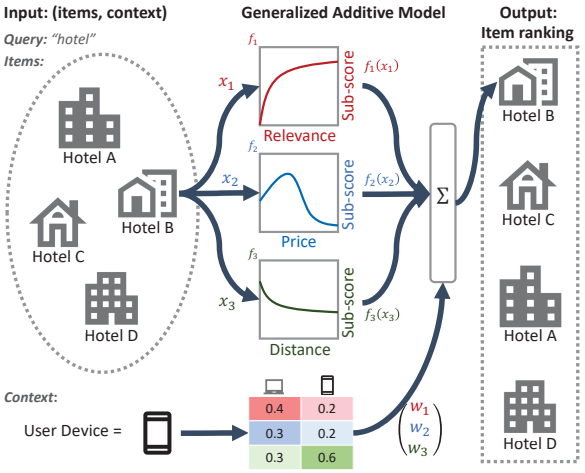


Figure 1: An example of a ranking GAM for local search. For each input feature x_j (*e.g.* price, distance), a sub-model produces a sub-score $f_j(x_j)$. Context features (*e.g.* user device type) can be utilized to derive importance weights of sub-models. The ranking score of each item is a weighted sum of sub-scores. The output is a ranked list of items sorted by their ranking scores.

1 INTRODUCTION

Learning-to-rank (LTR) [29] has been extensively studied [4, 6, 7, 17, 25, 41, 42, 47, 62] with broad applications in various fields [24, 34]. While many studies focus on building more accurate ranking models, fewer pay attention to the model interpretability. The lack of interpretability can lead to many issues in practice, such as difficult troubleshooting, opacity to potential social bias [26], vulnerability to data drifting, and high maintenance costs [52]. The latest SWIRL workshop report [13] lists model transparency as one of the key issues that the IR community should focus on in the following years. Hence, there is an emerging need to develop more transparent and interpretable learning-to-rank models.

There are roughly two categories of techniques to achieve model interpretability [15]: one aims to generate *post-hoc* explanations of an existing black-box model, the other aims to build *intrinsically interpretable* models where the model structure per se is understandable. In the specific application of learning-to-rank, there are a few recent studies [49, 55, 56, 58] focusing on generating post-hoc explanations for existing models. However, intrinsically interpretable models specifically designed for learning-to-rank remain less explored. Tree-based models [6, 17, 25] are often considered more interpretable than neural models. But the state-of-the-art implementations of tree-based models, like LambdaMART, often involve thousands of deep and wide trees, tree ensembles and bagging [5] to

achieve the best performance, making them virtually impossible to interpret by humans. In addition, there are some studies which use attention layers as model interpretation in related fields like CTR prediction [27], but the accuracy of attention-based interpretation has been questioned in recent work [53].

Although the post-hoc explanations are valuable and insightful for some applications, they can also suffer from being inaccurate about actual model behaviors, insufficient to understand model details or unable to incorporate information outside of the database, as pointed out in a recent study [51]. Hence, it is necessary to develop an intrinsically interpretable LTR model as an alternative for scenarios where ranking models need to be fully understandable.

For example, consider the cases where a ranking system is involved in determining bail or parole, loan eligibility assessment, advertisement targeting, or guiding medical treatment decisions – all of which are highly plausible scenarios today [12, 35, 40]. In such high-stakes decision-making scenarios, researchers are morally (and often legally) obligated to develop and deploy interpretable models. The contribution of *each individual feature* in these models should be examinable and understandable to ensure transparency, accountability and fairness of the outcomes.

Generalized additive models (GAMs) [19] provide a promising option to build such intrinsically interpretable models and have been used in many domains like finance [1] and health [8, 60]. The output of a GAM is the sum of multiple sub-models outputs, where each sub-model only takes one feature as input. A GAM has a higher model capacity than a linear model while staying intelligible compared to a black-box model. Each sub-model of a GAM precisely reflects the contribution of each input feature to the final outcome. Previous studies [31, 60] primarily applied GAMs to regression or classification. However, to the best of our knowledge, how to build GAMs for learning-to-rank tasks has not been well-studied.

A learning-to-rank task has different input and objective from classification or regression. The goal is to rank a list of items given some context. Hence, there would be both item-level features and list-level context features available. In Figure 1, we show an example of local search. Based on a user query "hotel", a set of item-level features x_j's are computed (*e.g.* relevance score and distance) for each item (hotel). In addition, there could be list-level context features (*e.g.* user device type). A ranking GAM should not only inherit the intelligibility of GAMs, but can also take the context features into account. As presented in Figure 1, the ranking GAM consists of several isolated sub-models, where each sub-model only takes a single item-level feature x_j as input and outputs a corresponding sub-score $f_j(x_j)$. In addition, the context features are used to derive importance weights for item-level sub-scores before they are summed up as the final ranking score. This ensures that not only the contribution of each item-level feature is interpretable, but also the contributions of the context features, which is crucial in the high-stakes decision-making scenarios.

There are multiple challenges to adapt existing GAMs for ranking tasks. First, traditional GAMs are only trained with loss functions for regression or classification tasks. It is unclear how effective such models can be when trained with ranking losses. In addition, how to leverage list-level context features in a ranking GAM is not clear. As shown later, directly projecting the context features from list-level to item-level is not effective for ranking tasks.

To overcome these challenges, we propose a novel neural GAM model for learning-to-rank tasks. In such a model, we employ standalone neural networks to instantiate sub-models for each individual feature. The flexibility of neural network framework enables us to leverage list-level context features in the ranking GAM structure and to optimize the model with ranking losses. The model can also seamlessly handle both numerical and sparse categorical features.

In summary, we make the following contributions in this paper:

- We formally define the structure of ranking GAMs in both the context-absent and context-present scenarios.
- We propose to instantiate ranking GAMs by neural networks using a novel architecture that can leverage context features effectively and can be trained with ranking losses.
- We develop a novel technique to distill each sub-model of a ranking GAM into a simple piece-wise linear function.
- We evaluate our proposed neural ranking GAMs on three learning-to-rank data sets and show their effectiveness.

2 RELATED WORK

Interpretable learning-to-rank. Model interpretability [61] becomes a popular research topic in recent years [10, 33, 50, 54, 57, 59]. Current studies on interpretable learning-to-rank mainly focus on generating interpretation for an existing black-box model. Singh *et al.* [56] describe a system to quantify and visualize the effect of each term on a certain decision of a given ranking model, such as why a document is ranked higher than another. They also attempt to distill a given ranking model with a subset of interpretable features in [55]. Verma *et al.* [58] also study a similar problem to quantify the contribution of terms in a specific document ranking generated by a given ranking model. Rennings *et al.* [49] propose a diagnostic data set based on a set of axioms to identify potential shortcomings in an existing IR system. However, there are few studies specifically designed to build "intrinsically" interpretable ranking models where each component of the model is understandable.

Generalized additive model. Proposed by Hastie *et al.* [19], generalized additive models (GAMs) provide a class of models which are more flexible than linear models, yet remain more interpretable than complicated models such as deep neural networks. Binder *et al.* [2] provide a thorough study on fitting GAMs with regression splines. Lou *et al.* [31] extend the comparing methods to include regression trees and conduct experiments on both classification and regression tasks. Zhang *et al.* [60] further study training GAMs specifically for multi-class classification tasks.

To obtain more accurate models, higher-order feature interactions were introduced to the additive models. Lou *et al.* [32] propose an efficient algorithm to identify feature pairs with interactions. A series of studies by Hooker [21, 22] also aim to discover additive structure with higher-order feature interactions. Meanwhile, others aim to reduce the number of features utilized in the model [28, 36, 43, 48] by imposing sparsity penalty. Lou *et al.* [30] propose to partially replace the non-linear function of some features by linear function to reduce the model complexity. Chouldechova *et al.* [11] also explore similar ideas. However, to the best of our knowledge, none of these studies focus on applying GAMs to learning-to-rank problems.

3 PROBLEM FORMULATION

In this section, we start by defining notations for a learning-to-rank problem. Then we formalize generalized additive model (GAM) and propose a novel formulation of ranking GAM.

3.1 Learning to Rank

In a ranking problem, we denote a data set with N lists as $\mathcal{D} = \{(\mathbf{q}, \mathbf{X}, \mathbf{y})\}_1^N$. For a specific list $(\mathbf{q}, \mathbf{X}, \mathbf{y}) \in \mathcal{D}$, $\mathbf{q} = (q_1, \cdots, q_m)$ is a context feature vector consisting of m list-level contextual signals (*e.g.* query features in search tasks, user information in recommendation tasks); $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^l$ is a set of l data items, each represented by a feature vector $\mathbf{x}_i$; $\mathbf{y} = \{y_i\}_{i=1}^l$ is a set of relevance labels of corresponding data items where a higher $y_i \in \mathbb{R}$ indicates that the item $\mathbf{x}_i$ is more relevant. Let $\mathbf{\Pi}_l$ denote the set of all permutations of l data items. The optimal ranking $\boldsymbol{\pi}^* \in \mathbf{\Pi}_l$ can always be obtained by ranking items $\mathbf{x}_i$'s according to their relevance labels y_i's from highest to lowest.

The typical learning-to-rank setting aims to learn a ranker φ from a training data set $\mathcal{D}_L$ with given relevance labels, such that for any list $(\mathbf{q}, \mathbf{X})$, the inferred ranking $\hat{\boldsymbol{\pi}} = \varphi(\mathbf{q}, \mathbf{X})$ can be as close to the optimal ground-truth ranking $\boldsymbol{\pi}^*$ as possible.

Specifically, in this work, we infer the ranking by scoring each item individually and sorting them based on their scores. For each list $(\mathbf{q}, \mathbf{X})$ with given context features and item features, we aim to learn a scoring function F which takes both the context features $\mathbf{q}$ and an individual item's features $\mathbf{x}_i$ as input, and output a ranking score $\hat{y}_i \in \mathbb{R}$:

$$\hat{y}_i = F(\mathbf{q}, \mathbf{x}_i)$$

The final predicted ranking $\hat{\boldsymbol{\pi}}$ will be generated by ranking all the items in $\mathbf{X}$ based on their inferred ranking scores $\hat{y}_i$'s. Notice that scoring each item individually does *not* mean this is a regression problem, as the objective is to optimize the ranking performance, and ranking rather regression losses are employed (see Section 4.3 for further discussion).

It is worth noting that there is not yet a common agreement on a formal definition of interpretability in learning-to-rank. Therefore, in this paper, we confine our discussion to the interpretability guarantees offered by generalized additive models (GAMs). We will briefly review the formal definition of GAMs, and then propose a novel ranking GAM definition with similar interpretability.

3.2 GAM

A generalized additive model (GAM) [19, 31] learns a function for each individual input feature respectively. Previous studies typically focus on applying generalized additive models on classification or regression tasks with numerical features. Formally, we denote a data set as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ where each $\mathbf{x}_i = (x_{i1}, \cdots, x_{in})$ is a feature vector containing n features and y_i is the target. For a regression task, $y_i \in \mathbb{R}$ is a real value, while for a binary classification task $y_i \in \{0, 1\}$ is a binary value. A generalized additive model takes a feature vector $\mathbf{x}_i$ and outputs $\hat{y}_i$ with the following structure:

$$g(\hat{y}_i) = f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_n(x_{in}) \tag{1}$$

Each $f_j(\cdot)$ is a function to be learned for the j-th feature. The function can be instantiated by different classes of functions, such as linear functions, splines or trees/ensemble of trees [31]. $g(\cdot)$ is a link function that links the sum of all $f_j(x_{ij})$'s to the model output. For example, $g(\cdot)$ could be an identity function for regression tasks, while for a binary classification task its inverse function $g^{-1}(\cdot)$ could be a logistic function $g^{-1}(u) = \frac{1}{1+\exp(-u)}$.

The model does not involve any interactions between features, which could lead to compromise in performance. However, the simple structure also introduces many appealing benefits in practice. Since each $f_j(\cdot)$ is essentially a univariate function, the relationship between a feature's value and the final response can be accurately quantified and visualized by plotting $f_j(\cdot)$. This transparency is required in high-stake applications in domains such as legal or medical, where black-box models cannot be deployed [31, 51].

3.3 Ranking GAM

Traditional GAMs are designed for classification and regression problems. There are no systematic studies to develop GAMs for ranking problems. In this subsection, we define ranking GAMs, which are designed to tackle the special structure of ranking problems while maintaining intelligibility of traditional GAMs.

Context-absent ranking. We start our discussion by the context-absent ranking scenario, where the list-level context features $\mathbf{q}$ are not available. In this scenario, a ranking GAM essentially applies a traditional GAM for regression on each item $\mathbf{x}_i$ in the list $\mathbf{X}$ as a scoring function. Given each item's representation $\mathbf{x}_i = (x_{i1}, \cdots, x_{in})$, the ranking score $\hat{y}_i$ can be derived by:

$$\hat{y}_i = F(\mathbf{x}_i) = f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_n(x_{in}) \tag{2}$$

Note that the item-level features can be context-dependent. For example, the BM25 scores are item-level features but depend on both queries (contexts) and documents (items).

However, not all context features can be effectively projected to item-level features, e.g., the time of day when a query is sent in search tasks (like 9 a.m. or 9 p.m.). In this case, the context-absent ranking formalization is not compatible with such a context feature.

Context-present ranking. Now we discuss the definition of our ranking GAM in the context-present scenario where list-level context features $\mathbf{q} = (q_1, \cdots, q_m)$ are available. A straightforward solution would be to project context features $\mathbf{q}$ as item-level features and apply a traditional GAM as an item scoring function like in the context-absent setting:

$$\begin{aligned}\hat{y}_i = F(\mathbf{q}, \mathbf{x}_i) = {} & f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_n(x_{in}) \\ & + f_{n+1}(q_1) + f_{n+2}(q_2) + \cdots + f_{n+m}(q_m)\end{aligned} \tag{3}$$

Unfortunately, this model structure cannot fully leverage the signals from context features in learning-to-rank scenarios. On one hand, most ranking losses used for training only involve the differences of predicted ranking scores $(\hat{y}_i - \hat{y}_{i'})$ between items. Obviously, the sub-model terms of context features $f_{n+k}(q_k)$ will be canceled out as all items within the same list share the same context feature values. On the other hand, most ranking metrics only care about the order of items $\hat{\boldsymbol{\pi}}$, which is not related to context feature sub-model terms $f_{n+k}(q_k)$ either.

In order to leverage context features for a GAM model, we propose the following ranking GAM model that utilizes context features to derive importance weights when combining item-level

features. Specifically, the predicted ranking score of each item is:

$$\hat{y}_i = F(\mathbf{q}, \mathbf{x}_i) = \sum_{j=1}^{n} \sum_{k=1}^{m} w_{j,k}(q_k) f_j(x_{ij}) \tag{4}$$

where both $f_j(\cdot)$ and $w_{j,k}(\cdot)$ are arbitrary univariate functions to be learned. It is worth noting that when the context features $\mathbf{q}$ are fixed, the predicted ranking score can still be decomposed as sum of functions of each item-level feature:

$$\hat{y}_i = F(\mathbf{q}, \mathbf{x}_i) = \sum_{j=1}^{n} \Big(w_j(\mathbf{q}) f_j(x_{ij}) \Big) \tag{5}$$

where $w_j(\mathbf{q}) = \sum_{k=1}^{m} w_{j,k}(q_k)$ can be interpreted as the importance weight of the j-th item feature derived from all the context features. For example, in a search task, item features like distance might be more important if users are searching for hotels, while the content relevance might be more important if users are searching for a convention center. And notice that the weight function $w_j(\mathbf{q})$ is also an additive model with regard to each context feature.

4 NEURAL RANKING GAM

In this section, we propose our method that instantiates ranking GAM based on neural networks.

4.1 Context-Absent Neural Ranking GAM

We start with the context-absent scenario, where a ranking data set can be represented as $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}_1^N$ as the context features $\mathbf{q}$ are not available. We will build a scoring function that predicts the ranking score $\hat{y}_i$ for each data item $\mathbf{x}_i$ in each list $\mathbf{X}$.

For simplicity, we focus on a single data item in a list and omit the subscripts. A data item can be represented as $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ where x_j represents the j-th feature of the data item $\mathbf{x}$.

We build a standalone neural network for each feature x_i which outputs a single "sub-score" $s_i \in \mathbb{R}$. This leads to n separate neural networks in total. In principle, users have the flexibility to construct any neural network structure with legitimate input and output. One can even build a different network structure for each feature according to its characteristics.

In our practice, we simply adopt a feed-forward network structure with L hidden layers for each feature as shown in Figure 2. Specifically, for each item feature x_j, we can have

$$\begin{aligned} \mathbf{z}_{j1} &= \sigma(\mathbf{W}_{j1} x_j + \mathbf{b}_{j1}) \\ \mathbf{z}_{j2} &= \sigma(\mathbf{W}_{j2} \mathbf{z}_{j1} + \mathbf{b}_{j2}) \\ &\cdots \\ \mathbf{z}_{jH} &= \sigma(\mathbf{W}_{jH} \mathbf{z}_{j(H-1)} + \mathbf{b}_{jH}) \end{aligned}$$

where $\mathbf{z}_h$ is the output of the h-th hidden layer; $\mathbf{W}_{jh}$ and $\mathbf{b}_{jh}$ are weight matrix and bias vector of the h-th hidden layer to be trained; $\sigma(\cdot)$ is the non-linear activation function. We choose the Rectifier (ReLU) [39] as the activation function.

The sub-score of feature x_j can be obtained by feeding the output of the final hidden layer into a dense layer:

$$f_j(x_j) = s_j = \mathbf{W}_j \mathbf{z}_{jH} + b_j \tag{6}$$

Based on the n neural networks we build for all the n features, we can obtain the final predicted score for item $\mathbf{x}$ by simply taking the sum of all the sub-scores:

$$\hat{y} = F(\mathbf{x}) = \sum_j f_j(x_j) = \sum_j s_j \tag{7}$$

The final predicted ranking $\hat{\boldsymbol{\pi}}$ can be obtained by sorting all the items $\mathbf{x}$'s in the list according to their predicted ranking scores.

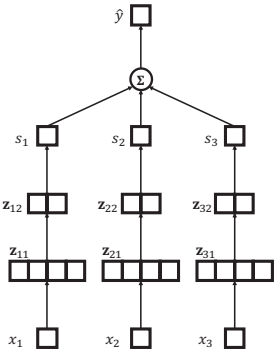


Figure 2: A graphical illustration of a context-absent neural ranking GAM.

4.2 Context-Present Neural Ranking GAM

In this subsection, we describe the neural network instantiation of the context-present ranking GAM where context features $\mathbf{q}$ of each list are available. In this design, the module for item features remains similar to the context-absent setting. We build an additional neural additive module for context features $\mathbf{q}$ which outputs an importance weights vector for item feature sub-models instead of a score.

We again focus on the scoring of a single data item and temporarily omit the subscripts. Specifically, an item in list $\mathbf{X}$ is denoted as $\mathbf{x} = (x_1, \cdots, x_n)$, and the context features are denoted as $\mathbf{q} = (q_1, \cdots, q_m)$. The sub-score $f_j(x_j)$ for each item feature x_j is instantiated in the same way (Equation (6)) as the context-absent model. From each individual context feature q_k, we derive an n-dimensional weighting vector $\boldsymbol{\alpha}_k$. As shown in Figure 3, the weighting vector can be obtained by a T-layer feed-forward neural network structure, where

$$\begin{aligned} \mathbf{z}_{k1} &= \sigma(\mathbf{W}_{k1} q_k + \mathbf{b}_{k1}) \\ \mathbf{z}_{k2} &= \sigma(\mathbf{W}_{k2} \mathbf{z}_{k1} + \mathbf{b}_{k2}) \\ &\cdots \\ \mathbf{z}_{kT} &= \sigma(\mathbf{W}_{kT} \mathbf{z}_{k(T-1)} + \mathbf{b}_{kT}) \end{aligned}$$

Similarly, $\mathbf{z}_{kt}$'s are hidden layer outputs, and $\mathbf{W}_{kt}, \mathbf{b}_{kt}$ are model parameters to be learned. Then we use a softmax layer on top of the final dense layer to derive $\boldsymbol{\alpha}_k$.

$$\boldsymbol{\alpha}_k = \text{softmax}(\mathbf{W}_k \mathbf{z}_{kT}) \tag{8}$$

where the j-th dimension of the weighting vector $\boldsymbol{\alpha}_k^{(j)}$ indicates the importance of the j-th item feature in $\mathbf{x}$ considering the k-th context feature. Notice that $\boldsymbol{\alpha}_k^{(j)}$ exactly corresponds to the value of function $w_{j,k}(q_k)$ in Equation (4). The intuition of using a softmax layer is to prevent the derived importance weights to be negative or to be extremely large on some item features, which would substantially compromise the model interpretability.

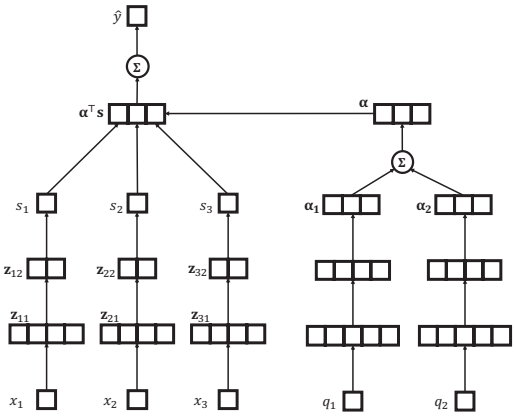


Figure 3: A graphical illustration of a context-present neural ranking GAM.

We denote the overall weighting vector as $\boldsymbol{\alpha} \in \mathbb{R}^n$. We obtain $\boldsymbol{\alpha}$ by taking the sum of weighting vectors $\boldsymbol{\alpha}_k$ from all the m context features as $\boldsymbol{\alpha} = \sum_{k=1}^{m} \boldsymbol{\alpha}_k$. The final ranking score is generated by taking the weighted sum of sub-model $f_j(x_j)$'s as described in Equation (5), where the weight of the j-th item feature $w_j(\mathbf{q})$ equals to $\boldsymbol{\alpha}^{(j)}$, *i.e.* the j-th dimension of the overall weighting vector.

4.3 Remarks

Ranking losses. Both neural ranking GAMs can be trained with any ranking loss functions. In this work, we train our model with approximate NDCG loss from [3, 46]. It is worth noting that regression losses such as Mean Squared Error (MSE) $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = ||\mathbf{y} - \hat{\mathbf{y}}||^2$ can be also used as loss functions for ranking. We will compare traditional GAMs optimizing regression loss with the proposed ranking GAMs in our experiments.

Non-numerical features. Another advantage of adopting neural networks is the flexibility to handle non-numerical features such as categorical or textual features, since handling such features in traditional GAM instantiated by splines or trees is non-trivial. Such features can be seamlessly incorporated into GAMs instantiated by neural networks by deriving embedding vectors. Notice that the introduction of embedding will not affect the final visualization of sub-models for non-numerical features.

5 SUB-MODEL DISTILLATION

Model distillation [20] is a widely-adopted technique to simplify deep neural networks. The general idea is to train a smaller, simpler model by minimizing the loss between its output and that of a more larger, complex model. Given the special architecture of our proposed neural GAM model, we propose to distill each *sub-model* individually. The benefit of sub-model distillation is multi-fold: the distilled sub-models can be faster to perform inference; they are also smaller in size which is beneficial for on-device deployment; they could potentially be more interpretable due to their simpler structure. We focus on distilling sub-models of numerical features where we utilize piece-wise regression (also known as segmented regression) [37].

5.1 Piece-wise Regression

We are given a numerical feature x and its sub-model $f(x)$ learned in a GAM. The goal of piece-wise regression is to find a piece-wise linear function (PWL) that is as close to $f(x)$ as possible. A PWL function can be described by a set of K knots, denoted as $S = \{(x_k, y_k)\}_{k=1}^{K}$ where x_k's in the knots are ordered, *i.e.* $x_k < x_{k+1}$. The definition of PWL function based on the knots S is:

$$PWL_S(x) = \begin{cases} y_1 & \text{if } x < x_1, \\ \frac{y_{k+1}-y_k}{x_{k+1}-x_k}(x - x_k) + y_k & \text{if } x_k \le x \le x_{k+1}, \\ y_K & \text{if } x > x_K. \end{cases}$$

$PWL_S(x)$ is a linear function between any two adjacent knots and becomes flat at two ends. Thus, the K knots uniquely determine the PWL function. For the practice of sub-model distillation, a small K (*e.g.* around 3 to 5) is usually sufficient.

Formally, the goal of piece-wise regression is to find the set of optimal knots $S = \{(x_k, y_k)\}_{k=1}^{K}$ that minimize the empirical loss on a training data set $\mathcal{D} = \{(x_i, f(x_i)\}_{i=1}^{N}$. In particular, we use the Mean Squared Error (MSE) as our loss function and have the following optimization problem:

$$S^* = \underset{S=\{(x_k,y_k)\}_{k=1}^{K}}{\arg\min} \frac{1}{|\mathcal{D}|} \sum_{\mathcal{D}} \left\| f(x_i) - PWL_S(x_i) \right\|^2 \tag{9}$$

Note that knots $S = \{(x_k, y_k)\}_{k=1}^{K}$ are not necessarily a subset of the training data $\mathcal{D} = \{(x_i, f(x_i)\}_{i=1}^{N}$.

Such a formulation can be solved by piece-wise regression packages such as [38]. However, they are usually slow and not scalable to large data sets. In the next section, we propose a greedy algorithm.

5.2 Fitting Algorithm

The major challenge of our fitting algorithm is to determine the x_k's for the K knots: $\{x_k\}_{k=1}^{K}$. With x_k's determined, the optimal y_k's can be computed by a method similar to least squares, which we will not elaborate in this paper due to limited space.

Our method first generates a set of x's as knot candidates P. Given all x's in the training data $\mathcal{D} = \{(x_i, f(x_i)\}$, we construct P by a heuristic using percentile boundary of 0%, 1%, ..., 99%, 100% of $\mathcal{D}$, resulting in at most 101 elements in P.

With a given set of candidate knots P, our problem becomes a combinatorial optimization one, where one needs to find a subset $S \subset P$ with size K that minimizes the loss in Equation (9). We denote the loss as $\mathcal{L}(S)$. A brute-force algorithm is to enumerate all the possible subset of P with size K, but the computational cost is intractable. Thus, we propose a greedy algorithm in Algorithm 1. The algorithm first constructs an initial set of knots in a greedy fashion and then refines it until convergence. More specifically,

- *Knots Initialization.* The algorithm greedily constructs an initial set S of size K by adding x's iteratively from P (See Line 1-5). In each step, the x^* that best minimizes the loss is added into the S. It stops once we have K knots.
- *Knots Refinement.* The algorithm iterates over each knot in S and tries to assess whether it can be replaced by another candidate knot to further minimize the loss (See Line 6-13). The procedure stops when there is no improvement after we loop over all $x_k \in S$ once.

Algorithm 1: Finding Knots for Piece-wise Regression

Input: Training data $\mathcal{D}$, candidates P, output size K.
Output: Knots $\{(x_k, y_k)\}_{k=1}^{K}$.

Let $S \leftarrow \{\min_{x \in P} x\}$
for $k \leftarrow 2$ to K **do**
  $x^* \leftarrow \arg\min_{x \in P \setminus S} \mathcal{L}(S \cup \{x\})$
  $S \leftarrow S \cup \{x^*\}$
end for
repeat
  for all $x \in S$ **do**
    $x^* \leftarrow \arg\min_{x' \in P \setminus S} \mathcal{L}(S \cup \{x'\} \setminus \{x\})$
    if $\mathcal{L}(S \cup \{x^*\} \setminus \{x\})) < \mathcal{L}(S)$ **then**
      $S \leftarrow S \cup \{x^*\} \setminus \{x\}$
    end if
  end for
until Convergence.
return $\{(x_k, y_k) | x_k \in S\}$ after solving y_k's for S.

Though the greedy algorithm may be stuck into local minimums in theory, we found that it outputs reasonably good PWL functions in practice as shown in our experiments. Since K is usually very small, finding y_k when x_k are given can be done efficiently.

6 EXPERIMENTS

In this section, we conduct experiments to evaluate both performance and interpretability of our proposed models. Notice that since our focus is on intrinsically interpretable models for high-stake applications, we do not include black-box models in our comparison such as LambdaMART because they are usually not viable options due to lack of transparency [19, 31]. We first conduct performance comparison in the family of GAMs. Then we illustrate the interpretability of our learned model by visualizing the sub-models. Finally we present the model performance when sub-models are distilled into piece-wise linear functions.

6.1 Data Sets

We use three data sets in our experiments. Two of them are public benchmark ones without context features. The third one is a private data set that contains non-numerical context features.

YAHOO. Yahoo! Learning to Rank Challenge data [9] is a publicly available data set. The original data set contains two sets for different purposes: Set1 and Set2. Set1 is the one commonly used for LTR evaluation. It consists of three partitions for training, validation and testing respectively. There are 19,944 queries in the training partition, 2,994 queries in the validation partition and 6,983 queries in the test partition. In this data set, each document is represented as up to 700 numerical features. Each feature has been normalized into the range of $[0, 1]$ by the inverse cumulative distribution. Documents are labeled with 5-level relevance labels from 0 to 4.

WEB30K. WEB30K [45] is a public learning-to-rank data set released by Microsoft. We use Fold1 out of the total five folds in the original data. Similarly to the YAHOO data set, Fold1 has the training, validation and testing partitions. There are 18,919 queries in the training partition, 6,306 queries in the validation partition and 6,306 queries in the test partition. In this data set, each document is represented by 136 numerical features. The meaning of each feature is disclosed and the range of each feature varies. Some query-dependent features are already captured by item-level features such as "covered query term number". There is no additional list-level context feature in this data set. Documents are also labeled with 5-level relevance labels from 0 to 4.

Chrome Web Store (CWS). CWS is a private data set that we collected by sub-sampling from the Chrome Web Store logs. Each list in this data set corresponds to a visit to Chrome Web Store. The items in the list were the ones shown to a user, and we collect the user actions like clicks or installations as our labels. For each item, we computed 15 numerical features. Moreover, each list contains 3 context features indicating users' region and language settings and the path of user visits and they are non-numerical. Similarly, we also partition the data set into three parts for training, validation and testing. There are 40,500 queries in the training partition, 4,578 in the validation and 4,462 queries in the test partition.

6.2 Experiment Setup

Comparing methods. In our experiments, we compare the performances of various models within the family of GAMs as they have the desired property of being intrinsically interpretable. The methods we compare are:

- *Tree-based GAM (Tree GAM).* We train tree-based GAM with regression loss (mean squared error, *i.e.*, MSE). We use open-sourced software MLTK[1] which implements GAM proposed by Lou *et al.* [31].
- *Neural GAM (Neural GAM).* We use neural networks instead of trees to construct GAMs. The model has similar structure with the context-absent neural ranking GAM, but is trained with regression loss (MSE) instead of ranking loss.
- *Tree-based Ranking GAM (Tree RankGAM).* We also try to train tree-based GAM with ranking loss. Since GAM has not been studied for ranking problems before, there is no existing implementations of GAM specifically designed for optimizing ranking loss. We use LambdaMART implemented by another open-sourced package[2] LightGBM [25], and confine the depths of trees in the model to be 1 so that the final model can be decomposed as Equation (2). The number of trees in LightGBM is set to $1,000$.
- *Context-Absent Neural Ranking GAM (Neural RankGAM).* The context-absent neural ranking GAM is presented in Section 4.1 which only takes item features. We train the model with a ranking loss (approximate NDCG).
- *Context-Present Neural Ranking GAM (Neural RankGAM+).* The context-present neural GAM for ranking is presented in Section 4.2. It can leverage context features to derive importance weights for item features. We also train the model with a ranking loss (approximate NDCG).

Our neural ranking GAMs are implemented and open-sourced[3] in the TensorFlow Ranking library [41].

[1] https://github.com/yinlou/mltk
[2] https://github.com/Microsoft/LightGBM
[3] https://github.com/tensorflow/ranking/releases/tag/v0.3.1

Hyperparameters. For neural ranking GAMs, we tune the hyperparameters and choose the following in our experiments. For YAHOO and WEB30K, we construct a neural network with $L = 2$ layers for each item feature x_i and the dimensions are 16 and 8 respectively. For CWS data set, we also build a neural network with $L = 2$ layers for each item feature x_i with dimensions as 64 and 32. Moreover, we build a neural network with $H = 2$ layers for each context feature q_i with dimensions as 128 and 64 for CWS. Since all context features in CWS data set are categorical, we use a d-dimensional ($d = 300$) embedding layer before the hidden layers for categorical features. We use AdaGrad [16] as the optimizer.

Evaluation Metrics. Normalized Discounted Cumulative Gain (NDCG) is a standard metric for evaluating ranking quality with respect to items with graded relevance [23]. In our experiments, we utilize NDCG_k as the evaluation metrics with different settings of k (i.e., 1, 5, 10) to evaluate the performance of each ranking method.

6.3 Performance Comparison

We first compare the performances of our proposed neural ranking GAMs to baselines on *context-absent* data sets in Table 1. Since there are no context features available, the context-present neural ranking GAM method is not included. Compared with traditional GAMs which optimize regression loss, both Tree RankGAM and Neural RankGAM achieve substantial improvements. For example, Neural RankGAM achieves NDCG_5 of 71.32% on YAHOO, while the Neural GAM achieves only 69.62%. Tree RankGAM also achieves similar improvement over Tree GAM. On WEB30K, Neural RankGAM achieves NDCG_5 of 43.29%, about +10% higher than Neural GAM with regression loss. Similarly, Tree RankGAM also reaches more than +9% higher NDCG_5 than the traditional Tree GAM. This confirms the importance to enable traditional GAMs to optimize ranking losses for LTR problems, as the performance can be extensively improved with exactly the same interpretability.

We further compare the performances of our proposed models on the CWS data set where context features are available. Table 2 shows the results. It can be observed that our proposed context-present neural ranking GAM (Neural RankGAM+) achieves significantly better performance than the context-absent version and Tree RankGAM. As one can observe, Neural RankGAM+ achieves +4.9% improvement in terms of NDCG_5 comparing to the context-absent Neural RankGAM. The improvement shows that context-present Neural RankGAM+ can effectively leverage context features, which is a disadvantage of the traditional tree-based GAMs.

6.4 Interpretability Illustration

It is still an ongoing debate [14, 18, 44] on how to quantitatively measure model interpretability. Considering the challenges and the lack of consensus on a rigorous method to evaluate model interpretability, we instead focus on showing that our proposed LTR models have similar interpretability as traditional GAMs.

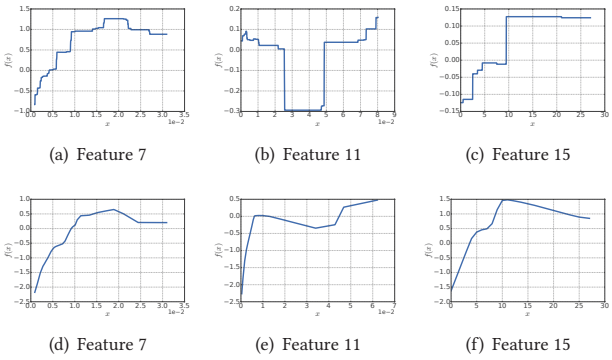
Sub-models of item-level features. We visualize the sub-models of the most important item-level features for both Tree RankGAM and Neural RankGAM+ on CWS data set. The importance of a feature is measured by the NDCG_5 drop on validation data sets (denoted as $\Delta\text{NDCG}_5(j)$) when feature values x_j are perturbed within every lists by random shuffling. For each feature, we plot the

Table 1: Performance comparison on context-absent data sets (%). Results that are statistically significantly better ($\alpha = 0.01$) than *Tree RankGAM* are marked with an asterisk (*), and best GAM result per column is bolded.

Data set	Method	NDCG_1	NDCG_5	NDCG_{10}
YAHOO	Tree GAM	67.61	69.46	73.89
	Neural GAM	67.63	69.62	73.98
	Tree RankGAM	69.12	71.03	75.04
	Neural RankGAM	**69.36**	**71.32**	**75.33***
WEB30K	Tree GAM	29.79	32.79	35.96
	Neural GAM	30.59	33.55	36.54
	Tree RankGAM	41.90	42.04	44.37
	Neural RankGAM	**44.31***	**43.29***	**45.09***

Table 2: Performance comparison on a context-present data set (%). Results that are statistically significantly better ($\alpha = 0.01$) than *Tree RankGAM* are marked with an asterisk (*), and best GAM result per column is bolded.

Data set	Method	NDCG_1	NDCG_5	NDCG_{10}
CWS	Tree GAM	19.74	32.91	36.72
	Neural GAM	20.09	34.01	38.60
	Tree RankGAM	20.16	35.06	39.27
	Neural RankGAM	20.35	34.94	38.93
	Neural RankGAM+	**24.43***	**39.88***	**42.84***



(a) Feature 7 (b) Feature 11 (c) Feature 15

(d) Feature 7 (e) Feature 11 (f) Feature 15

Figure 4: Learned sub-model $f(x)$ of Tree RankGAM (top) and Neural RankGAM+ (bottom) for selected features on CWS data set.

$f(x)$ function for x values above the bottom 5% and below the top 5% of all data points in the training data. Figure 4 show the curves learned by Tree RankGAM and Neural RankGAM+.

As one can observe, the curves of each feature learned by both methods seem to have similar trends, which confirms the neural sub-models are as effective as tree-based sub-models. We also notice that the curves in Tree RankGAM are non-continuous, with potentially steep "steps" (*e.g.* around $x = 2.5$ in Figure 4(b) and around $x = 5$ in Figure 4(c)), whereas the curves learned by Neural RankGAM+ with ReLU activation functions are continuous with less steep slopes. The more continuous curves usually can generalize better, but largely depend on the real applications. By changing the activation function in Neural RankGAMs, it is also possible to change the corresponding behaviors.

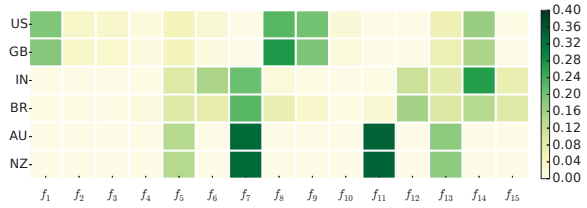


Figure 5: A case study of context feature "region". Each row corresponds to a specific context feature value, where the j-th column corresponds to the derived importance weights on item feature sub-model $f_j(x_j)$.

Sub-models of list-level context features. Neural RankGAM+ can also leverage list-level context features into the model. We show how to visualize the contribution of context features to achieve interpretability. Figure 5 illustrates how item feature sub-models are weighed based on different values of the context feature "region". This visualization is sufficient for users to analyze patterns of context feature contribution to the final results. As a running example, it can be observed in Figure 5 that the learned importance vectors seem to be similar across regions with high geographical and/or cultural proximity. For instance, "US" and "GB" have similar importance vectors as they share similar languages and cultures.

6.5 Sub-Model Distillation

We also show how sub-model distillation can further improve the scalability of our models. We select Neural RankGAM on YAHOO and WEB30K and Neural RankGAM+ on the CWS data set. We use a 5-segment piece-wise linear functions to distill each sub-model $f_j(\cdot)$ in above selected models. Table 3 compares the performance before and after the distillation, where the distilled ones are named with a "(D)" suffix. The results show that the distilled models have only small NDCG losses. On all the data sets, the drop of performance in terms of NDCG_{10} are less than -1%. Moreover, as Table 4 shows, distilling sub-models substantially reduces model inference time. On both YAHOO and WEB30K data sets[4], sub-model distillation achieves 17-23X speed-up.

To better visualize the distilled sub-models, we select a set of most important features for the Neural RankGAM on WEB30K data set. The feature importance is measured by ΔNDCG_5 as defined in Section 6.4. Figure 6 plots the value of originally learned $f_j(x)$'s at some randomly sampled data points from the training data set and the distilled $\hat{f}_j(x)$'s for the selected features. Clearly, the piece-wise linear function fits the data very well, especially in dense areas with a lot of data points from training data.

Admittedly, it could be observed that the error of the distilled sub-model in sparse areas with fewer data points from training data could be larger. (*e.g.* around $x = 200$ in Figure 6(a)). Data points around these areas may actually have high ranking scores. Hence, errors for these data points might lead to larger performance drops in terms of ranking metrics, as shown in the relatively larger performance drop on WEB30K data set. Developing a sub-model distillation technique specifically for ranking is another non-trivial task. We leave it for future study.

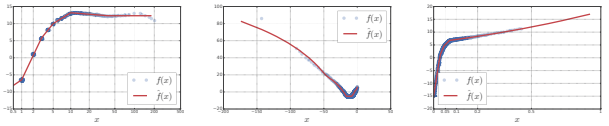
[4]We do not disclose the inference time for the CWS data set due to its proprietary nature, but the conclusions are similar.

Table 3: Comparing performances before and after sub-model distillation. Models with distilled sub-models is marked with suffix "(D)".

Data set	Method	NDCG_1	NDCG_5	NDCG_{10}
YAHOO	Neural RankGAM	69.36	71.32	75.33
	Neural RankGAM (D)	69.41	71.33	75.33
WEB30K	Neural RankGAM	44.31	43.29	45.09
	Neural RankGAM (D)	41.63	41.90	44.15
CWS	Neural RankGAM+	24.43	39.88	42.84
	Neural RankGAM+ (D)	24.11	39.42	42.39

Table 4: Comparison of model inference time τ before and after sub-model distillation measured by CPU time.

Data set	YAHOO		WEB30K	
Method	Total τ	τ / query	Total τ	τ / query
Neural RankGAM	327s	47ms	303s	48ms
Neural RankGAM (D)	19s	3ms	13s	2ms



(a) min of term frequency (whole document)

(b) LMIR.JM (body)

(c) sum of stream length normalized term frequency (whole document)

Figure 6: Curves of distilled sub-model $\hat{f}(x)$ and the originally learned sub-model $f(x)$ values in Neural RankGAM on randomly sampled training data points for selected features on WEB30K data set.

7 CONCLUSIONS

In this paper, we explore building *intrinsically interpretable* learning-to-rank models by adapting generalized additive models (GAMs). Unlike previous GAMs that mainly focus on regression or classification, our proposed ranking GAMs can capture both list-level context features and item-level features of ranking tasks. Our experiments verify that the proposed ranking GAMs outperform traditional GAMs on LTR tasks while maintaining interpretability of GAMs. Effectively leveraging list-level context features further improves the performance. We also present how to distill sub-models into simple piece-wise linear functions for further simplicity.

As an early exploration on building intrinsically interpretable LTR models, we stick to the most basic form of GAM. However, we are aware that there are numerous directions to further develop and understand ranking GAMs: (1) Incorporating limited pairwise feature interactions to improve model accuracy with reasonable interpretability (similar to [32]); (2) Enabling users to dictate constraints such as monotonicity on sub-models; (3) Combining ranking GAMs with fully-fledged neural networks or decision tree models to achieve both high accuracy and high interpretability.

Acknowledgments. We thank Vytenis Sakenas, Dmitry Osmakov, Olexiy Oryeshko for implementing early version of neural ranking GAMs and piece-wise regression for us. We also thank Po Hu, Janelle Lee, and Dan Chary Chen for preparing data sets for our experiments and the anonymous reviewers for their constructive feedback.

REFERENCES

[1] Daniel Berg. 2007. Bankruptcy prediction by generalized additive models. *Applied Stochastic Models in Business and Industry* 23, 2 (2007), 129–143.

[2] Harald Binder and Gerhard Tutz. 2008. A comparison of methods for the fitting of generalized additive models. *Statistics and Computing* 18, 1 (2008), 87–99.

[3] Sebastian Bruch, Masrour Zoghi, Mike Bendersky, and Marc Najork. 2019. Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks. In *SIGIR*.

[4] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *ICML*.

[5] Christopher Burges, Krysta Svore, Paul Bennett, Andrzej Pastusiak, and Qiang Wu. 2011. Learning to rank using an ensemble of lambda-gradient models. In *Proceedings of the learning to rank Challenge*. 25–35.

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

[7] Christopher J Burges, Robert Ragno, and Quoc V Le. 2007. Learning to rank with nonsmooth cost functions. In *NeurIPS*. 193–200.

[8] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. 2015. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *KDD*.

[9] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*.

[10] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. 2018. Neural attentional rating regression with review-level explanations. In *WWW*.

[11] Alexandra Chouldechova and Trevor Hastie. 2015. Generalized additive model selection. *arXiv preprint arXiv:1506.03850* (2015).

[12] Alexandra Chouldechova and Aaron Roth. 2018. The Frontiers of Fairness in Machine Learning. arXiv:cs.LG/1810.08810

[13] J Shane Culpepper, Fernando Diaz, and Mark D Smucker. 2018. Research frontiers in information retrieval: Report from the third strategic workshop on information retrieval in Lorne (SWIRL 2018). In *ACM SIGIR Forum*, Vol. 52. ACM, 34–90.

[14] Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608* (2017).

[15] Mengnan Du, Ninghao Liu, and Xia Hu. 2019. Techniques for interpretable machine learning. *Commun. ACM* 63, 1 (2019), 68–77.

[16] 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.

[17] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[18] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. 2018. Explaining explanations: An approach to evaluating interpretability of machine learning. *arXiv preprint arXiv:1806.00069* (2018).

[19] Trevor Hastie and Robert Tibshirani. 1986. Generalized Additive Models. *Statist. Sci.* 1, 3 (1986), 297–318.

[20] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. arXiv:stat.ML/1503.02531

[21] Giles Hooker. 2004. Discovering additive structure in black box functions. In *KDD*.

[22] Giles Hooker. 2007. Generalized functional ANOVA diagnostics for high-dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics* 16, 3 (2007), 709–732.

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

[24] Alexandros Karatzoglou, Linas Baltrunas, and Yue Shi. 2013. Learning to rank for recommender systems. In *RecSys*.

[25] 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 *NeurIPS*.

[26] Juhi Kulshrestha, Motahhare Eslami, Johnnatan Messias, Muhammad Bilal Zafar, Saptarshi Ghosh, Krishna P Gummadi, and Karrie Karahalios. 2019. Search bias quantification: investigating political bias in social media and web search. *Information Retrieval Journal* 22, 1-2 (2019), 188–227.

[27] Zeyu Li, Wei Cheng, Yang Chen, Haifeng Chen, and Wei Wang. 2020. Interpretable Click-Through Rate Prediction through Hierarchical Attention. In *WSDM*.

[28] Yi Lin, Hao Helen Zhang, et al. 2006. Component selection and smoothing in multivariate nonparametric regression. *The Annals of Statistics* 34, 5 (2006), 2272–2297.

[29] Tie-Yan Liu. 2009. *Learning to Rank for Information Retrieval*. Now Publishers Inc.

[30] Yin Lou, Jacob Bien, Rich Caruana, and Johannes Gehrke. 2016. Sparse partially linear additive models. *Journal of Computational and Graphical Statistics* 25, 4 (2016), 1126–1140.

[31] Yin Lou, Rich Caruana, and Johannes Gehrke. 2012. Intelligible models for classification and regression. In *KDD*.

[32] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. 2013. Accurate intelligible models with pairwise interactions. In *KDD*.

[33] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *NeurIPS*.

[34] Craig Macdonald, Rodrygo LT Santos, and Iadh Ounis. 2013. The whens and hows of learning to rank for web search. *Information Retrieval* 16, 5 (2013), 584–628.

[35] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2019. A Survey on Bias and Fairness in Machine Learning. *ArXiv* abs/1908.09635 (2019).

[36] Lukas Meier, Sara Van de Geer, Peter Bühlmann, et al. 2009. High-dimensional additive modeling. *The Annals of Statistics* 37, 6B (2009), 3779–3821.

[37] Vito Muggeo. 2003. Estimating Regression Models with Unknown Break-Points. *Statistics in medicine* 22 (10 2003), 3055–71. https://doi.org/10.1002/sim.1545

[38] Vito Muggeo. 2008. Segmented: An R Package to Fit Regression Models With Broken-Line Relationships. *R News* 8 (01 2008), 20–25.

[39] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*.

[40] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. 2019. Dissecting racial bias in an algorithm used to manage the health of populations. *Science* 366, 6464 (2019), 447–453.

[41] 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 *KDD*.

[42] Rama Kumar Pasumarthi, Honglei Zhuang, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2020. Permutation Equivariant Document Interaction Network for Neural Learning to Rank. In *ICTIR*.

[43] Ashley Petersen and Daniela Witten. 2019. Data-adaptive additive modeling. *Statistics in medicine* 38, 4 (2019), 583–600.

[44] Forough Poursabzi-Sangdeh, Daniel G Goldstein, Jake M Hofman, Jennifer Wortman Vaughan, and Hanna Wallach. 2018. Manipulating and measuring model interpretability. *arXiv preprint arXiv:1802.07810* (2018).

[45] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* abs/1306.2597 (2013). http://arxiv.org/abs/1306.2597

[46] 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.

[47] Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2021. Neural Rankers are hitherto Outperformed by Gradient Boosted Decision Trees. In *ICLR*.

[48] Pradeep Ravikumar, John Lafferty, Han Liu, and Larry Wasserman. 2009. Sparse additive models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71, 5 (2009), 1009–1030.

[49] Daniël Rennings, Felipe Moraes, and Claudia Hauff. 2019. An Axiomatic Approach to Diagnosing Neural IR Models. In *ECIR*.

[50] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *KDD*.

[51] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 5 (2019), 206.

[52] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*. 2503–2511.

[53] Sofia Serrano and Noah A Smith. 2019. Is Attention Interpretable?. In *ACL*.

[54] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *ICML*.

[55] Jaspreet Singh and Avishek Anand. 2018. Posthoc interpretability of learning to rank models using secondary training data. In *SIGIR Workshop on Explainable Recommendation and Search*.

[56] Jaspreet Singh and Avishek Anand. 2019. EXS: Explainable Search Using Local Model Agnostic Interpretability. In *WSDM*.

[57] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *ICML*.

[58] Manisha Verma and Debasis Ganguly. 2019. LIRME: Locally Interpretable Ranking Model Explanation. In *SIGIR*.

[59] Mike Wu, Michael C Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. 2018. Beyond sparsity: Tree regularization of deep models for interpretability. In *AAAI*.

[60] Xuezhou Zhang, Sarah Tan, Paul Koch, Yin Lou, Urszula Chajewska, and Rich Caruana. 2019. Axiomatic Interpretability for Multiclass Additive Models. In *KDD*.

[61] Yongfeng Zhang, Jiaxin Mao, and Qingyao Ai. 2019. WWW Tutorial on Explainable Recommendation and Search. In *WWW*.

[62] Honglei Zhuang, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2020. Feature transformation for neural ranking models. In *SIGIR*.