Accelerating Ranking in E-Commerce Search Engines through Contextual Factor Selection

Anxiang Zeng,[1,2] **Han Yu,**[1] **Qing Da,**[3*] **Yusen Zhan,**[3] **Chunyan Miao**[1,2*]
[1]School of Computer Science and Engineering, Nanyang Technological University (NTU), Singapore
[2]Alibaba-NTU Singapore Joint Research Institute
[3]Alibaba Group, Hangzhou, China
*Corresponding authors: daqing.dq@alibaba-inc.com, ascymiao@ntu.edu.sg

Abstract

In large-scale search systems, the quality of the ranking results is continually improved with the introduction of more factors from complex procedures. Meanwhile, the increase in factors demands more computation resources and increases system response latency. It has been observed that, under some certain context a search instance may require only a small set of useful factors instead of all factors in order to return high quality results. Therefore, removing ineffective factors accordingly can significantly improve system efficiency. In this paper, we report our experience incorporating our Contextual Factor Selection (CFS) approach into the Taobao e-commerce platform to optimize the selection of factors based on the context of each search query in order to simultaneously achieve high quality search results while significantly reducing latency time. This problem is treated as a combinatorial optimization problem which can be tackled through a sequential decision-making procedure. The problem can be efficiently solved by CFS through a deep reinforcement learning method with reward shaping to address the problems of reward signal scarcity and wide reward signal distribution in real-world search engines. Through extensive offline experiments based on data from the Taobao.com platform, CFS is shown to significantly outperform state-of-the-art approaches. Online deployment on Taobao.com demonstrated that CFS is able to reduce average search latency time by more than 40% compared to the previous approach with negligible reduction in search result quality. Under peak usage during the Single's Day Shopping Festival (November 11th) in 2017, CFS reduced peak load search latency time by 33% compared to the previous approach, helping Taobao.com achieve 40% higher revenue than the same period during 2016.

Introduction

Information retrieval plays an important role in commercial applications, ranging from web searching engines (Google.com, Baidu.com, etc.) to e-commence websites (Taobao.com, Amazon.com). Such applications usually need to rank a large set of data items in response to users' requests. To support these applications, there are generally two issues to be considered: a) effectiveness, how accurate and reliable the search results in the final ranking list are and b) efficiency, how fast the search engine's response to the user's queries can be and whether the computational overhead of the ranking operation is as low as possible from a system perspective. It is a challenge to address both issues in large-scale applications for providing excellent user experience and an efficient performance solution.

To address the high computational cost of more factors as well as large-scale traffic requests, a search engine generally has to degrade the service level in terms of effectiveness, i.e., reducing the number of recalled items, deactivating some unnecessary service and so on, in oder to avoid access delay or even unavailability, which severely affects the users' experience (Li et al. 2009). Liu *et al.* proposed a cascading ranking model to address the trade-off between effectiveness and efficiency in the large-scale e-commerce search applications (Liu et al. 2017), by reducing the number of items in the ranking process with some strategies, which sheds a light on us to optimize the e-commerce search engine in another possible way. In a search engine, a set of factors is applied to the ranking process and not all of those factors are necessary under all circumstances. On the one hand, there may exist redundancy of factors; on the other hand, conversion rates vary on items under different contexts[1]. For instance, for the users with higher purchasing power or for long-tail queries, maybe cheap factors are enough. The above observations shows the possibility of simultaneously improving effectiveness and efficiency by carefully selecting a subset of all factors under certain circumstances, which indeed is a standard combinatorial optimization problem, but with auxiliary context description.

Recently, Bello *et al.* show that reinforcement learning is capable of solving combinatorial optimization problem like TSP via pointer network (Vinyals, Fortunato, and Jaitly 2015; Bello et al. 2016). In this paper, we report our experience deploying an innovative model via reinforcement learning based approach – Contextual Factor Selection (CFS) – in Taobao.com to address the aforementioned challenge. We formally define our optimization problem with a general framework and a loss function which reflects both ranking effectiveness and efficiency. Then, we transform the contextual combinatorial optimization problem into a sequential decision-making problem by incorporating the contex-

[1]Here, the $\langle u, q \rangle$ user-query pair denotes the context.

tual setting and factor selection into the state and action of an MDP, respectively. The reward is designed to encourage saving in the computational cost as well as ensuring that the ranking results are still effective. Our algorithm outperforms the previous algorithm used by Taobao since deployment in Jan 2017. In the Singles' Day Shopping Festival in 2017, we also demonstrated the capabilities of this new method under large-scale real-world usage.

Application Description

Suppose that $\mathcal{O}$ is the set of all available items in a database. $\mathcal{Q}$ is the set of all possible queries and $\mathcal{U}$ denotes the set of all users' information. Let $\{\langle u,q\rangle_1, \langle u,q\rangle_2, \ldots, \langle u,q\rangle_m\}$ be a set of user-query pairs, where $\langle u,q\rangle_i \in \mathcal{U} \times \mathcal{Q}$ denotes the i-th user-query pair from the search requests. $O_i = \{o_{i,1}, o_{i,2}, \cdots, o_{i,n_i}\}$ is the set of items associated with the i-th user-query request, where n_i is the number of the related items. The ranking problem in e-commerce can be formally defined as a task to generate a permutation function $\sigma_i \in \Sigma_i$, where $\sigma_i$is an one-to-one correspondence from $\{1, 2, \ldots, n_i\}$ to itself and Σ_i denotes the set of all the possible permutations on $\mathcal{O}_i$. The goal is to maximize the probability of purchase under the permutation. The permutation is usually generated by a ranking function $F(\langle u,q\rangle_i, o_{i,j}) \rightarrow \mathbb{R}$ which scores each item $o_{i,j} \in O_i$ for the request $\langle u,q\rangle_i$. Let $\boldsymbol{x}^{i,j} \in \mathbb{R}^n$ be the corresponding factor vector of item $o_{i,j} \in \mathcal{O}_i$ under query $\langle u,q\rangle_i$, where $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n_i$. Without loss of generality, the ranking model is defined by

$$F(\langle u,q\rangle_i, o_{i,j}) = f\left(\boldsymbol{x}^{i,j}\right), \tag{1}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the ranking function. It could be any function such as a linear model, a deep neural network or a tree model. The ranking function is usually trained from a dataset $\mathcal{D} = \left\{\left(\langle u,q\rangle_i, \boldsymbol{x}^i, \boldsymbol{y}_i\right)\right\}_{i=1}^N$ logged from a real system, where N is the number of training samples and $\boldsymbol{y}_i = \{y_{1,i}, y_{2,i}, \ldots, y_{i,n_i}\}$ denotes the labels associated with the items. Specifically, $y_{i,j} \in \mathcal{Y} = \{\text{view}, \text{click}, \text{buy}\}$ represents the feedback of the user on the j-th item. The training can be conducted with any *learning to rank* methods (Li 2011). It is worth noting that, in this paper we assume that a trained ranking function is given and consider the general case that the ranking function provided is a black box.

There has been some previous work for resolving the effectiveness and efficiency challenge. Here, we review will some of them which are most relevant to our work. Cascade learning is originally proposed to address the effectiveness and efficiency issue in traditional classification and detection problems (Bourdev and Brandt 2005; Schneiderman 2004; Viola and Jones 2003). Liu *et al.* develop a cascade ranking model for a large-scale e-commerce search system and deploy it in Taobao.com (Liu et al. 2017). However, they only exploited optimization in terms of the number of ranking items. In this work, we mainly focus on factor usage during the ranking process.

Feature selection approaches aim to remove irrelevant and/or redundant features to improve learning performance (Guyon and Elisseeff 2003). Traditional feature selection techniques roughly fall into two categories: 1) filter methods and 2) wrapper methods. Filter methods use learner-irrelevant measurements to evaluate and select features, such as information gain and Relief (Kira and Rendell 1992). Wrapper methods involve the final learner in the feature selection process, such as using the accuracy to measure the goodness of features. Liu *et al.* proposed the TEFE (Time-Efficient Feature Extraction) approach, which balances the test accuracy and test time cost by extracting a proper subset of features for each test object (Liu et al. 2008). In the learning to rank literature, feature selection is a common strategy to improve efficiency. In general, a set of useful factors are selected from a complete set of all possible factors according to some criteria such as importance to ranking (Geng et al. 2007; Wang, Lin, and Metzler 2010; Wang, Metzler, and Lin 2010). Geng *et al.* proposed a selection method based on factor importance in a query-free manner, but they did not consider the real computational cost and query-dependent factors (Geng et al. 2007). There are also some query-dependent methods, in which the cost (delay) of the query is considered (Wang, Lin, and Metzler 2010; Wang, Metzler, and Lin 2010). In contrast, we consider the computational cost (delay) of individual factors.

Use of AI Technology

Contextual Factor Selection for Ranking

In this subsection, we describe the proposed *Contextual Factor Selection* (CFS) approach to help a search engine achieve both high effectiveness and high efficiency. A factor vector $\boldsymbol{x}_{i,j} \in \mathbb{R}^p$ is assigned to the corresponding item $o_{i,j} \in \mathcal{O}_i$, in which each dimension of the factor vector is calculated on-line with different computational cost. Let $\boldsymbol{x}^{i,j} = \{x_1^{i,j}, x_2^{i,j}, \ldots, x_p^{i,j}\}$ be the factor vector associated with a cost vector $\boldsymbol{c} = \{c_1, c_2, \ldots, c_p\}$, where c_k denotes the computational cost of the k-th factor. Let Ω be the set of all factors and S be a subset of Ω. The indicator function for whether a factor belongs to Ω is defined as

$$\mathbb{I}_S(k) = \begin{cases} 1, & x_k \in S, \\ 0, & x_k \notin S. \end{cases} \tag{2}$$

From a practical point of view, some of factors are not necessary for ranking. For example, given a set of factors $\Omega = \{x_k^{i,j} \mid k = 1, 2, \ldots, p\}$, a subset S of Ω with highly confident factors might be sufficient under some contexts. Therefore, given an item $o_{i,j}$ and an indicator function $\mathbb{I}_S$, the computational cost function can be written as $\sum_{k=1}^p \mathbb{I}_S(k)c_k$, where the indicator function determines whether or not we use the factor for the ranking process[2]. Thus, given a set of items $\mathcal{O}_i$, the total computational cost is

$$\sum_{j=1}^{n_i} \sum_{k=1}^{p} \mathbb{I}_S(k)c_k. \tag{3}$$

As defined in Equation 1, the ranking model with all factors can be written as $F_\Omega(o_{i,j}) = f(x_1^{i,j}, x_2^{i,j}, \ldots, x_p^{i,j})$ and

[2]We mainly consider the computational cost of the factors while ignoring other costs.

the one with a subset S is written as

$$F_S(o_{i,j}) = f\left(\mathbb{I}_S(1)x_1^{i,j}, \mathbb{I}_S(2)x_2^{i,j}, \ldots, \mathbb{I}_S(p)x_p^{i,j}\right) \quad (4)$$

Intuitively, we can treat the permutation generated by F_Ω as the optimal one since it includes all the factors we have during the ranking process. Thus, given a $\langle u, q\rangle_i$ request, the objective is

$$\min_{S\subseteq\Omega} \; D^{\mathcal{O}_i}(F_\Omega||F_S) + \lambda n_i \sum_{k=1}^{p} \mathbb{I}_S(k)c_k, \quad (5)$$

where $D^{\mathcal{O}_i}(F_\Omega||F_S)$ denotes the distance between functions F_Ω and F_S over the item set $\mathcal{O}_i$, which could be any distance between two functions, i.e., Kullback-Leibler divergence (Kullback and Leibler 1951). The second term is the computational cost of the factors in the set S. $\lambda > 0$ is the trade-off parameter and n_i is the number of items in query i. Intuitively, the objective function implies that it reduces the usage of factors as much as possible, while approximating the original ranking function F_Ω as close as possible.

However, Equation 5 is intractable even for a single $\langle u, q\rangle_i$ request. Consequently, it is a NP-hard problem in general (Davis, Mallat, and Avellaneda 1997; Natarajan 1995). Moreover, we need to perform contextual factor selection, (i.e., solving a general NP-hard problem for every $\langle u, q\rangle_i$, which is impractical in a large-scale system even with a small number of contexts). To overcome this challenge, we generalize the solution of Equation 5 at the contextual level. That is, we do not directly search for the optimal subset $S^\star$ and define :

$$S_{\langle u,q\rangle} = H\left(\langle u, q\rangle \mid \boldsymbol{\theta}\right) \quad (6)$$

where H is a model parameterized by $\boldsymbol{\theta}$ and the user-query pair $\langle u, q\rangle$ characterizes the context. Such formulation reduces the solution space to a global parameter from the original multiple optimal subset selection problems, based on the assumption that similar $\langle u, q\rangle$ representations should have similar optimal subset structures. Thus, our goal becomes to search for the global parameter vector $\boldsymbol{\theta}$ in order to minimize the loss defined in Equation 5 over all the $\langle u, q\rangle$ requests.

To illustrate our method, we adopt the linear ranking functions as a demonstration, and other representations, (i.e., deep neural network and tree based ranking function, can be derived by similar way). In the linear setting, the score of item $o_{i,j}$ under user-query $\langle u, q\rangle_i$ is

$$f(x_1^{i,j}, x_2^{i,j}, \ldots, x_p^{i,j}) = \sum_{k=1}^{p} w_k^i x_k^{i,j}, \quad (7)$$

where w_k^i is the corresponding weight of factor $x_k^{i,j}$.

From another point of view, the permutation $\sigma_i \in \Sigma_i$ significantly depends on the factors used to calculate the scores. Formally, given a user-query pair $\langle u, q\rangle_i$ and a corresponding weight vector $\boldsymbol{w}^{\langle u,q\rangle_i}$, the linear ranking function is:

$$f\left(\mathbb{I}_{S_{\langle u,q\rangle_i}}(1)x_1^{i,j}, \mathbb{I}_{S_{\langle u,q\rangle_i}}(2)x_2^{i,j}, \ldots, \mathbb{I}_{S_{\langle u,q\rangle_i}}(n)x_p^{i,j}\right)$$
$$= \sum_{k=1}^{p} \mathbb{I}_{S_{\langle u,q\rangle_i}}(k) w_k^i x_k^{i,j} \quad (8)$$

where $\mathbb{I}_{S_{\langle u,q\rangle_i}}(k)$ is the indicator function, which depends on the user-query pair $\langle u, q\rangle_i$. For convenience, $\mathbb{I}_{S_{\langle u,q\rangle_i}} \in \{0, 1\}^p$ denotes the binary vector with respect to the factor vector $\boldsymbol{x}^{i,j}$. Therefore, the ranking permutation σ_i highly depends on the ranking function $f(\cdot)$ and the indicator function $\mathbb{I}_{S_{\langle u,q\rangle_i}}$, assuming the weight vector is fixed if the ranking model is given. Thus, the crucial part of ranking optimization is to learn an indicator function $\mathbb{I}_{S_{\langle u,q\rangle_i}}$ to determine the utilization of the factors. To simplify the notation, we write $\mathbb{I}_{S_{\langle u,q\rangle_i}}$ as $\mathbb{I}_{\boldsymbol{\theta}}$, where the parameter $\boldsymbol{\theta}$ characterizes the factor subset $S_{\langle u,q\rangle_i}$. Hence, the ranking permutation $\sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}$ can be derived by the ranking function $f(\cdot)$ and the indicator function $\mathbb{I}_{\boldsymbol{\theta}}$. Thus, we can rewrite the distance function $D^{\mathcal{O}_i}(F_\Omega||F_S)$ as $D^{\mathcal{O}_i}(\sigma_\Omega||\sigma_S)$, where σ_Ω and σ_S are permutations derived by ranking function F_Ω and F_S, respectively.

With the optimal ranking permutation σ_Ω, we then define the distance over a item set $\mathcal{O}_i$ between a permutation $\sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}$ and the optimal ranking permutation σ_Ω as

$$D^{\mathcal{O}_i}(\sigma_\Omega||\sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}) = \frac{2}{n_i(n_i-1)} \sum_{\substack{j,k=1, j\neq k \\ \sigma_\Omega(j)\geq\sigma_\Omega(k)}}^{n_i} \mathbf{1}(\sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}(j) < \sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}(k)), \quad (9)$$

where $\mathbf{1}(\sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}(j) < \sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}(k))$ equals 1 if $\sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}(j) < \sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}(k)$ and 0 otherwise. The definition of the distance D is analogous the averaged pairwise loss in learning to rank literature. The distance measures how *far away* the derived permutation is to the optimal one. With the distance and total cost function defined above, our goal is, given a user-query pair $\langle u, q\rangle_i$, the corresponding item set $\mathcal{O}_i$ and the ranking function f, to learn an indicator function $\mathbb{I}_{\boldsymbol{\theta}}$ that minimizes the the distance D function and the total computational costs. Formally, the objective in Equation 5 can be further rewritten as

$$\mathcal{L}(\langle u, q\rangle_i, \mathcal{O}_i, f \mid \boldsymbol{\theta}) = D^{\mathcal{O}_i}(\sigma_\Omega||\sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}) + \lambda \sum_{j=1}^{n_i}\sum_{k=1}^{p} \mathbb{I}_{\boldsymbol{\theta}}(k)c_k$$
$$= \underbrace{\frac{2}{n_i(n_i-1)} \sum_{\substack{j,k=1, j\neq k \\ \sigma_\Omega(j)\geq\sigma_\Omega(k)}}^{n_i} \mathbf{1}(\sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}(j) < \sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}}(k))}_{\text{Ranking Effectiveness}}$$
$$+ \underbrace{\lambda\, n_i \sum_{k=1}^{p} \mathbb{I}_{\boldsymbol{\theta}}(k)c_k}_{\text{Ranking Efficiency}} \quad (10)$$

RankCFS: A Reinforcement Learning Approach

As the optimization problem defined in Equation 5 is NP-hard in general, finding the exact solution is computationally intractable. Inspired by recent work (Bello et al. 2016; Benbouzid, Busa-Fekete, and Kégl 2012), we propose to learn an indicator function $\boldsymbol{\theta}$ using reinforcement learning, by transforming the assignment of each element in the indicator vector as a sequential decision-making problem. We refer to this approach as RankCFS.

Reinforcement Learning and Actor-Critic Methods In this subsection, we will review some basic concepts in reinforcement learning. This subsection could be skipped If the readers are similar with reinforcement learning.

In reinforcement learning, an agent sequentially selects actions to maximize total expected pay-off. These problems are typically formalized as Markov decision processes (MDPs) with a tuple of $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S} \subseteq \mathbb{R}^d$ and $\mathcal{A} \subseteq \mathbb{R}^m$ denote the state and action spaces. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ represents the transition probability governing the dynamics of the system, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function quantifying the performance of the agent and $\gamma \in (0, 1)$ is a discount factor specifying the degree to which rewards are discounted over time. At each step t, the agent is in state $\boldsymbol{s}_t \in \mathcal{S}$ and must choose an action $\boldsymbol{a}_t \in \mathcal{A}$, transitioning it to a successor state $\boldsymbol{s}_{t+1} \sim p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ as given by $\mathcal{P}$ and yielding a reward r_t. A policy $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is defined as a probability distribution over state-action pairs.

Policy gradients (Kober and Peters 2011; Sutton and Barto 1998) are a class of reinforcement learning algorithms that have shown successes in solving complex robotic problems (Kober and Peters 2011). Such methods represent the policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)$ by an unknown vector of parameters $\boldsymbol{\theta} \in \mathbb{R}^d$. The goal is to determine the optimal parameter vector $\boldsymbol{\theta}^\star$ that maximize the expected discounted cumulative reward $\mathcal{J}(\boldsymbol{\theta}) = \sum_{\boldsymbol{\tau}} P(\boldsymbol{\tau}|\boldsymbol{\theta})\mathfrak{R}(\boldsymbol{\tau})$, where $\boldsymbol{\tau} = [\boldsymbol{s}_{0:T}, \boldsymbol{a}_{0:T}]$ denotes a trajectory over a possibly finite horizon T and $P(\boldsymbol{\tau}|\boldsymbol{\theta})$ denotes the probability of acquiring a trajectory under the policy parameterization $\pi_{\boldsymbol{\theta}}(\cdot)$. the policy gradient of $\mathcal{J}(\boldsymbol{\theta})$ can be estimated using the the likelihood ratio trick as

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \sum_{\boldsymbol{\tau}} P(\boldsymbol{\tau}|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log P(\boldsymbol{\tau}|\boldsymbol{\theta}) \mathfrak{R}(\boldsymbol{\tau}) \tag{11}$$

which is usually approximated with empirical estimate for m sample trajectories under the policy $\pi_{\boldsymbol{\theta}}$, i.e., $\frac{1}{m} \sum_{j=1}^{m} \nabla_{\boldsymbol{\theta}} \log P(\boldsymbol{\tau^j}|\boldsymbol{\theta}) \mathfrak{R}(\boldsymbol{\tau^j})$. The gradient can be applied in every step t and further improved by introducing a learned bias $V^{\pi_{\boldsymbol{\theta}}}(s_t|\mu)$ to reduce the variance of this estimate as in (Mnih et al. 2016)

$$d\theta \leftarrow \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t)(\mathfrak{R}_t - V^{\pi_{\boldsymbol{\theta}}}(s_t|\mu)) \tag{12}$$

where $\mathfrak{R}_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$ is the discounted cumulative reward from step t and $V^{\pi_{\boldsymbol{\theta}}}(s_t|\mu)$ is the function approximation of $\mathfrak{R}_t$ parameterized by μ.

Converting CFS into an MDP Setting It is possible to learn a selected factor, by a reinforcement learning policy, instead of approximating the indicator vector $\mathbb{I}_{\boldsymbol{\theta}}$ directly. However, it results in a combinatorial action space which leads to computational intractability and searching failure with a high probability.

To reduce the action space, we introduce a fixed factor sequence so that a policy can sequentially determine the corresponding utility of factors. Formally, for each user-query $\langle u, q \rangle_i$ request, the vector function $\mathbb{I}_{\boldsymbol{\theta}}$ can be determined in p steps, where in the k-th step ($1 \leq k \leq p$), we need to decide whether the k-th factor should be included in the ranking function or not for this request, (i.e. $\boldsymbol{a}_k \in \mathcal{A} = \{\text{Skip}, \text{Keep}\}$

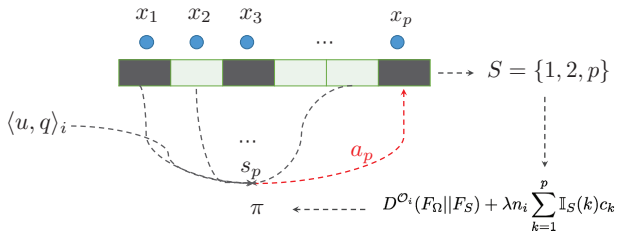


Figure 1: An example of the pruning procedure via reinforcement learning

is the action taken at step k and $\mathcal{A}$ is the action space). $\boldsymbol{a}_k$ is obtained through a policy

$$\boldsymbol{a}_k = \boldsymbol{\pi}(\boldsymbol{s}_k|\boldsymbol{\theta}), \tag{13}$$

where $\boldsymbol{s}_k$ is the state representation of k-th step. Then we can obtain

$$\mathbb{I}_{\boldsymbol{\theta}}(k) = \begin{cases} 0, & \text{if } \boldsymbol{a}_k = \text{Skip} \\ 1, & \text{if } \boldsymbol{a}_k = \text{Keep}. \end{cases} \tag{14}$$

After p steps, $\mathbb{I}_{\boldsymbol{\theta}}$ is determined and so is the ranking permutation $\sigma^{\mathbb{I}_{\boldsymbol{\theta}}}$. Then, we can directly calculate the loss $\mathcal{L}(\langle u, q \rangle_i, \mathcal{O}_i, f \mid \boldsymbol{\theta})$ to evaluate the effectiveness of selected actions, which can be further used to define the total reward of the actions generated in p steps during the episode. (Figure 1). The key idea lies in the state design (base on which the action is generated), the reward design (how to evaluate each action) and the optimization method for this reinforcement learning problem (how to find the optimal policy) which will be described in details below.

The State and Reward Design The optimal policy should generalize over the state space, and the optimal actions for an episode only depend on the $\langle u, q \rangle_i$ request. Thus, ideally, the state should be designed as

$$\boldsymbol{s}_k = (v_{\langle u,q \rangle_i}, k) \in R^{l+1} \tag{15}$$

where $v_{\langle u,q \rangle_i} \in R^l$ is the representations for the user-query pair $\langle u, q \rangle_i$. The corresponding reward r_k is then defined as

$$r_k = \begin{cases} 0, & 1 \leq k < p \\ -\mathcal{L}(\langle u, q \rangle_i, \mathcal{O}_i, f \mid \boldsymbol{\theta}), & k = p. \end{cases} \tag{16}$$

The agent obtains a reward of 0 when the episode is not terminated, (i.e. $1 \leq k < p$), and a reward of -$\mathcal{L}(\langle u, q \rangle_i, \mathcal{O}_i, f \mid \boldsymbol{\theta})$ when the episode ends, like in the goal-directed tasks. Following this definition and by assigning γ to 1, we can then conclude that the objective in this reinforcement learning problem is exactly the negative of the objective in Equation 10:

$$\mathfrak{R}(\boldsymbol{\tau}) = \sum_{k=1}^{p} \gamma^{k-1} r_k = -\mathcal{L}(\langle u, q \rangle_i, \mathcal{O}_i, f \mid \boldsymbol{\theta}) \tag{17}$$

Algorithm 1 RankCFS

Input:
D: Training data set $\mathcal{D} = \{(\langle u, q\rangle_i, \mathcal{O}_i)\}_{i=1}^N$
f: The ranking function
$\gamma, \lambda, \beta, r_c, T_{max}$: Parameters of the algorithm
Output:
$\boldsymbol{\theta}$: Parameters of actor model
1: Initialize $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$
2: $T \leftarrow 1$
3: **repeat**
4: **for** each $(\langle u, q\rangle_i, \mathcal{O}_i) \in \mathcal{D}$ (For each page view) **do**
5: $T \leftarrow T + 1$
6: Initialized the initial state $\boldsymbol{s}_1$ as in Eq. 19
7: **for** $k = 1, 2, \ldots, p$ **do**
8: Taking action $\boldsymbol{a}_k \in \{\text{Skip, Keep}\}$ on the k-th factor based on $\pi_{\boldsymbol{\theta}}(\boldsymbol{s}_k)$, observe r_k and $\boldsymbol{s}_{k+1}$.
9: Cache the tuple $(\boldsymbol{s}_k, \boldsymbol{a}_k, r_k, \boldsymbol{s}_{k+1})$
10: **end for**
11: $\mathfrak{R} \leftarrow 0$
12: **for** $k = p, p-1, \ldots, 1$ **do**
13: $\mathfrak{R} \leftarrow r_k + \gamma\mathfrak{R}$
14: $\boldsymbol{\theta} \leftarrow \text{Adam}(\boldsymbol{\theta}, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_k|\boldsymbol{s}_k)(\mathfrak{R} - V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t|\mu)))$
15: $\boldsymbol{\mu} \leftarrow \text{Adam}(\boldsymbol{\mu}, (\mathfrak{R} - V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_k|\boldsymbol{\mu}))\nabla_{\boldsymbol{\mu}} V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_k|\boldsymbol{\mu}))$
16: **end for**
17: **end for**
18: **until** $T > T_{max}$

This means that maximizing $\mathfrak{R}(\boldsymbol{\tau})$ can directly minimize $\mathcal{L}$, allowing us to find the optimal solution of $\mathcal{L}$ through deep reinforcement learning.

However, empirically there are two issues that make learning the optimal policy for above reinforcement learning problem difficult. Firstly the reward is sparse over states. This is known as the *sparse feedback problem* (Kulkarni et al. 2016). Secondly the rewards ard distributed widely in a continuous space, making the critic model difficult to converge. Inspired by the *reward shaping* (Ng, Harada, and Russell 1999) technique, we consider slightly adjusting the representations of states and rewards, to mitigate these issues.

We firstly initialize $\mathbb{I}_{\boldsymbol{\theta}}^{'} = [1, 1, ..., 1] \in R^p$ as an all-1 vector. Thus at a given step k, we update $\mathbb{I}_{\boldsymbol{\theta}}^{'}$ as

$$\mathbb{I}_{\boldsymbol{\theta}}^{'}(t|k) = \begin{cases} \mathbb{I}_{\boldsymbol{\theta}}(t), & 1 \le t < k \\ 1, & k \le t \le p. \end{cases} \tag{18}$$

Then, we extend our state vector as

$$\boldsymbol{s}_k = (v_{\langle u,q\rangle_i}, k, \mathbb{I}_{\boldsymbol{\theta}}^{'}(\cdot|k)) \in R^{l+p+1}. \tag{19}$$

Thus, each state memorizes the decisions made before during an episode. At each step k, the reward is calculated based on $\mathbb{I}_{\boldsymbol{\theta}}^{'}(\cdot|k)$, (i.e. at each step it is pre-evaluated for the decisions made so far), assuming the remaining decisions are all "1"s by default. For each reward r_k, we decompose it into the effectiveness part $\mathcal{T}(\boldsymbol{s}_k, \boldsymbol{a}_k)$ and the efficiency part $\mathcal{G}(\boldsymbol{s}_k, \boldsymbol{a}_k)$, i.e., $r_k = \mathcal{T}(\boldsymbol{s}_k, \boldsymbol{a}_k) + \mathcal{G}(\boldsymbol{s}_k, \boldsymbol{a}_k)$. For the efficiency part, we simply add a penalty when selecting the k-th factor as

$$\mathcal{G}(\boldsymbol{s}_k, \boldsymbol{a}_k) = \begin{cases} 0, & \text{if } \boldsymbol{a}_k = \text{Skip} \\ -\lambda n_i c_k, & \text{if } \boldsymbol{a}_k = \text{Keep.} \end{cases} \tag{20}$$

This part is consistent with the Equation 10. For the effectiveness part, we choose to give a constant penalty if the ranking loss under $\mathbb{I}_{\boldsymbol{\theta}}^{'}$ exceeds a pre-defined threshold as

$$\mathcal{T}(\boldsymbol{s}_k, \boldsymbol{a}_k) = \begin{cases} -r_c, & D^{\mathcal{O}_i}(\sigma_\Omega || \sigma_i^{\mathbb{I}_{\boldsymbol{\theta}}^{'}}) > \beta \\ 0, & \text{otherwise.} \end{cases} \tag{21}$$

With such a design, we enable help the critic module to distinguish *bad* and *good* ranking results much more easily. Moreover, we can avoid generating poor ranking performance with a large penalty r_c.

Learning the Policy After transforming the original problem into a reinforcement learning problem, we then apply an existing reinforcement learning methods to learn a solution policy. In this paper, we choose the well-known policy gradient method with actor-critic models as described in (Mnih et al. 2016). It is worthing noting that, the difficulty of the original optimization problem does not decrease with the introduction of reinforcement learning techniques. The RL-based approach here acts as a solver whose solution space contains the optimal solutions, and provides an efficient search path to the optimal solutions through trial-and-error.

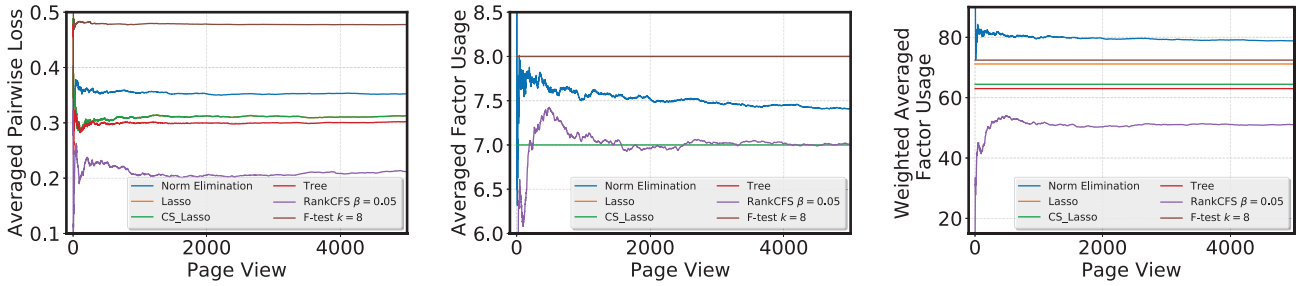
Algorithm 1 shows the training details. The data $\mathcal{D} = \{(\langle u, q\rangle_i, \mathcal{O}_i)\}_{i=1}^N$, the reward discount factor γ, the parameters used in the reward definition λ, β, r_c, and the maximum number of training steps T_{max} are given as inputs to the algorithm. The parameter of the actor network $\boldsymbol{\theta}$ is the output of the algorithm. We firstly initialize the parameters of the actor and critic network, as well as the step counter T. The training phase starts with the iteration of each of the page views, with which an episode will be generated during the Lines 6-10. Then, the standard policy gradient method is applied in Lines 14-15, where the tuple $(\boldsymbol{s}_k, \boldsymbol{a}_k, r_k, \boldsymbol{s}_{k+1})$ is organized in descending order so that the discounted cumulative reward $\mathfrak{R}$ can be updated incrementally as specified in Line 13. The training process ends when the number of steps exceeds the given threshold T_{max}.

Application Development and Deployment

Before the actual deployment of the technique, we compared our method based on an offline dataset from Taobao with the norm elimination method, the l_1-based feature selection method, the tree-based feature selection method and the F-test feature selection method in an off-line evaluation setting in order to facilitate decision-making about which technique shall be deployed.

Norm Elimination this method removes those factors for which the absolute values of weights are less than a predefined threshold ϵ.

l_1-based feature selection is a model-based feature selection method. It selects factors according to the l_1 regularizer. The basic idea is to eliminate those factors whose corresponding l_1 coefficients are zero. Since this method



(a) Pairwise loss (b) Factor usage. (c) Weighted factor usage

Figure 2: Off-line comparison of pairwise loss, factor usage and weighted averaged factor usage among various approaches.

must be based on a supervised machine learning model, we need to convert our ranking problem into a supervised one. We define the training dataset as follows. Let the label $l^{i,j} = f(x_1^{i,j}, x_2^{i,j}, \ldots, x_p^{i,j}) = \sum_{k=1}^{p} w_k^i x_k^{i,j}$ and corresponding factor vector $\boldsymbol{x}^{i,j}$, thus the set of training samples has the form $\mathcal{D}_{\mathcal{T}} = \{l^{i,j}, \boldsymbol{x}^{i,j}\}$ such that $j = 1, \ldots, n_i$ and $i = 1, \ldots, N$. Therefore, we can train a regressor using the training set $\mathcal{D}_{\mathcal{T}}$ and select the factors base on the trained model. We adopt Lasso as our comparison method.

Cost-sensitive l_1-based feature selection cost-sensitive lasso. It adds a cost to l_1 regularizer according to the computational time of each factor.

Tree-based feature selection is similar to the l_1-based feature selection. The main difference is that we replace the Lasso model with a non-linear regression tree model.

F-test feature selection is a model-free feature selection method that selects top k factors based on F-test scores.

Rank Contextual Factor selection (RankCFS) algorithm is the proposed actor-critic method which is capable of adjusting the selection of factors based on the contexts of the search queries.

For l_1-based feature selection, tree-based feature selection and F-test feature selection, we adopt their implementations in scikit-learn (Pedregosa et al. 2011). We implement RankCFS with Tensorflow (Abadi et al. 2016). For the optimal ranking model in the off-line evaluation, we select one of the linear ranking models from Taobao.com and treat it as a black box so that only the inputs and the outputs of the ranking model are considered during the experiments. We set the constant $\epsilon = 0.1$ in the Norm Elimination method and the constant that multiplies the l_1 term α equals 0.05 in the Lasso model. We choose the default ExtraTreeRegressor in scikit-learn package as our tree model. The actor and critic are construct by two deep neural networks (DNN) with three fully connected layers, respectively. The DNN structure of the actor is $266 \times 128 \times 128 \times 20$ and that of the critic is $266 \times 128 \times 128 \times 1$. We adopt *relu* as the activation functions for the hidden layers, Adam as our optimizer and the learning rates of actor and critic are set to 0.0001 and 0.001, receptively.

We sample a dataset with $100,000$ samples train the l_1-based, Tree-based and F-test approaches[3] on $50,000$ samples and then test them on the remaining of $50,000$ samples [4]. For the l_1, Tree-based and F-test methods, feature selection is performed after the training. That is, we use a fixed feature selection policy during the testing stage. Since our method considers the computational costs of factors, the computational cost vector $\boldsymbol{c}$ is obtained from the on-line operational environment of Taobao.com.

We test our methods on $5,000$ page views and each page view contains 10 items so that there are $50,000$ testing examples. Then, we evaluate the averaged pairwise loss defined in Equation 9 and factor usage over page views. Figures 2a-2b show our experiment results. Generally, the Norm Elimination method removes those factors whose absolute values are small under different contexts. Therefore indicators such as loss and factor usage may vary over page views. Figure 2a shows that RankCFS with the threshold $\beta = 0.05$ outperforms all other methods in terms of pairwise loss. In addition, in Figure 2b, it can be observed that the averaged factor usage of RankCFS is also close to the lowest Tree-based method. Figure 2c illustrates the weighted factor usage, in which the weights are the corresponding computational costs. This metric is more appropriate for describing the factor usage in terms of efficiency due to the variations in computational costs among factors. For example, only the absolute values of weights are considered in the Norm Elimination method, while RankCFS tends to eliminate those factors with high computational costs. Although, RankCFS with $\beta = 0.05$ and Tree-based method have similar averaged factor usage, RankCFS with $\beta = 0.05$ has much lower weighted factor usage. This demonstrates that our approach reduces the computational burden in context-aware manner and saves more computational resources. Empirically, our method is capable of exploring for the better solution in a combinatorial solution space with significantly reduced factor usage. The F-test method suffers high pairwise losses since it is a model-free method, it does not consider the ranking model we adopt.

[3] It is not necessary to train Norm Elimination method since it only removes those factors whose absolute values of weights are less than a positive constant ϵ.

[4] $10,000$ page views and 10 items in each page view.

Table 1: Comparisons of different methods in terms of Averaged Pairwise Loss (APL), Averaged Factor Usage (AFU) and Weighted Factor Usage (WFU)

Algorithm	APL	AFU	WFU
F-test / $k = 8$	0.47	8	72.44
F-test / $k = 11$	0.40	11	84.63
F-test / $k = 14$	0.29	14	107.93
Norm Elimination	0.35	7.41	78.84
Lasso	0.31	8	71.16
Cost-sensitive Lasso	0.32	7	65
Tree-based	0.3	7	63
RankCFS / $\beta = 0.05$	**0.21**	**7.01**	**51.06**
RankCFS / $\beta = 0.15$	0.27	9.07	67.40
RankCFS / $\beta = 0.25$	0.25	8.4	63.72

Overall, the experiments show that our RankCFS algorithm successfully explores the function space and find an high quality approximations to the optimal ranking function. We summarized the complete experimental results in Table 1 under different parameter settings, which shows that RankCFS with $\beta = 0.25$ outperforms the other methods in general. Therefore, RankCFS was deployed in Taobao.com to replace the previous search algorithm in Jan 2017.

Application Use and Payoff

To demonstrate the payoff generated, we conduct the experiment in the real-world large-scale operational environment of Taobao.com with a standard A/B test setting. We adopt the same learning structure as the off-line setting, but with a more complex nonlinear optimal ranking model. The training is conducted with more than $\times 10^9$ training samples on a distributed streaming system in an on-line learning fashion.

We conducted a standard A/B test experiment in our operational environment, where roughly 6% of the users are selected at random for testing. The parameter $\beta \in \{0.25, 0.15, 0.05\}$ and $\lambda \in \{0.9, 0.8, 0.7\}$ are tuned through the GMV and search latency. The goal is to minimize the impact on the GMV as much as possible, while reducing the latency as much as possible. Figure 3a shows the best results with $\beta = 0.05$ and $\lambda = 0.9$. Our method reduces the average search latency by approximately 40%, compared to the control group. For the maximum search latency, out algorithm has reduced it by roughly 25%.

The Singles' Day Shopping Festival Performance

Alibaba Singles' Day shopping festival is one of the biggest shopping extravaganzas around the world which can be thought of as the Chinese version of Black Friday. In 2017, by the end of day (November 11), sales hit a new record of $25.3 billion, more than 40% higher than sales on the same day in 2016. It attracted over hundreds of millions users from more than 200 different countries. The infrastructure system manages to handle 0.325 millions orders per second during peak usage[5]. The e-commerce search system played a crucial role in this event.

On November 11th, the search traffic burden of the e-commerce search engine abruptly increases by multiple times compared to a regular day. On the one hand, the e-commerce search engine faces the high traffic challenge, which might lead to system degradation. On the other hand, it is still crucial to provide accurate search results.

During this event, the proposed approach was deployed on top of (Liu et al. 2017) in order to perform optimization at the search engine system level. The CLOES approach enables the system to optimize the number of items in the ranking process via a cascading model, while our method concentrates on optimizing the set of selected ranking factors during the ranking process. Thus, both of these approaches can be applied on the search engine simultaneously. We use the CLOSE approach as our control group, and the CLOES+RankCFS approach as the experimental group, in which the parameter $\beta = 0.05$[6]. Figure 3b depicts the average latency change during the course of the experiments. The proposed approach reduced the latency by more than 20% compared to using only the CLOES approach. In addition, our method saves approximately 33% peak usage latency compared to using only the CLOES approach.

Maintenance

Since deployment in Jan 2017, we conducted internal reviews on the effectiveness of the algorithm on a quarterly basis to ensure efficiency operation. Each revision has revealed the need for some modifications to the system. However, the framework of the algorithm has not been changed in any significant way thus far.

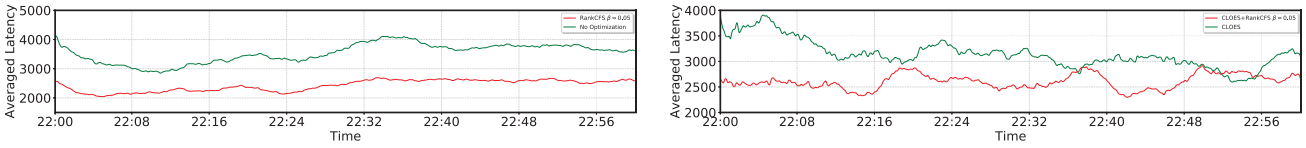
Conclusions and Future Work

In this paper, we investigated the effectiveness and efficiency issues in a real-world large-scale e-commerce search system. We formally defined the contextual factor selection problem in ranking system. Then, we converted the problem into a reinforcement learning problem through reward shaping to account for the scarcity and wide distribution of reward signals in a practical search engine. We then proposed an efficient approach to select the optimal set of factors to considered based on the context of each search query in order to enhance efficiency and effectiveness. The algorithm has been deployed in Taobao which is China's largest e-commerce platform. Actual usage data demonstrate that our method is a practical solution for real-world large-scale e-commerce search systems which significantly outperformed the previous algorithm in the platform.

In future research, we plan to explore other optimization approaches such as memory usage and load balancing, in conjunction with search latency. We will also incorporate the considerations for fairness (Yu et al. 2018), persuasiveness (Liu et al. 2019) and privacy preservation (Gao et al. 2019; Yang et al. 2019) aspects related to search and recommendation to further enhance user experience.

[5]https://techcrunch.com/2017/11/11/alibaba-smashes-its-singles-day-record/

[6]Due to limited on-line resources available for the purpose of A/B testing in the system, we are only able to use CLOES as our control group.



(a) Latency during regular operation

(b) Latency during the Singles' Day Shopping Festival A/B test

Figure 3: Latency in the e-commerce search engine on Taobao.com. The lower the value, the better the performance.

Acknowledgments

This research is supported by the Nanyang Assistant Professorship (NAP), AISG-GC-2019-003, NRF-NRFI05-2019-0002, NTU-SDU-CFAIR (NSC-2019-011), and Alibaba-NTU-AIR2019B1.

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, 265–283.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Benbouzid, D.; Busa-Fekete, R.; and Kégl, B. 2012. Fast classification using sparse decision dags. In *ICML*, 747–754.

Bourdev, L., and Brandt, J. 2005. Robust object detection via soft cascade. In *CVPR*, volume 2, 236–243.

Davis, G.; Mallat, S.; and Avellaneda, M. 1997. Adaptive greedy approximations. *Constructive Approximation* 13(1):57–98.

Gao, D.; Liu, Y.; Huang, A.; Ju, C.; Yu, H.; and Yang, Q. 2019. Privacy-preserving heterogeneous federated transfer learning. In *IEEE BigData*.

Geng, X.; Liu, T.-Y.; Qin, T.; and Li, H. 2007. Feature selection for ranking. In *SIGIR*, 407–414.

Guyon, I., and Elisseeff, A. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research* 3(Mar):1157–1182.

Kira, K., and Rendell, L. A. 1992. The feature selection problem: traditional methods and a new algorithm. In *AAAI*, 129–134.

Kober, J., and Peters, J. 2011. Policy search for motor primitives in robotics. *Machine Learning* 1(84):171–203.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *NIPS*. 3675–3683.

Kullback, S., and Leibler, R. A. 1951. On information and sufficiency. *The Annals of Mathematical Statistics* 22(1):79–86.

Li, B.; Yu, H.; Shen, Z.; and Miao, C. 2009. Evolutionary organizational search. In *AAMAS*, 1329–1330.

Li, H. 2011. *Learning to Rank for Information Retrieval and Natural Language Processing*. Morgan & Claypool Publishers.

Liu, L.-P.; Yu, Y.; Jiang, Y.; and Zhou, Z.-H. 2008. Tefe: A time-efficient approach to feature extraction. In *ICDM*, 423–432.

Liu, S.; Xiao, F.; Ou, W.; and Si, L. 2017. Cascade ranking for operational e-commerce search. In *KDD*, 1557–1565.

Liu, C.; Dong, Y.; Yu, H.; Shen, Z.; Gao, Z.; Wang, P.; Zhang, C.; Ren, P.; Xie, X.; Cui, L.; and Miao, C. 2019. Generating persuasive visual storylines for promotional videos. In *CIKM*, 901–910.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Harley, T.; Lillicrap, T. P.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *ICML*, volume 48, 1928–1937.

Natarajan, B. K. 1995. Sparse approximate solutions to linear systems. *SIAM Journal on Computing* 24(2):227–234.

Ng, A. Y.; Harada, D.; and Russell, S. J. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 278–287.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Schneiderman, H. 2004. Feature-centric evaluation for efficient cascaded object detection. In *CVPR*.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *NIPS*. 2692–2700.

Viola, P., and Jones, M. 2003. Rapid object detection using a boosted cascade of simple features. In *CVPR*, I–511–I–518.

Wang, L.; Lin, J.; and Metzler, D. 2010. Learning to efficiently rank. In *SIGIR*, 138–145.

Wang, L.; Metzler, D.; and Lin, J. 2010. Ranking under temporal constraints. In *CIKM*, 79–88.

Yang, Q.; Liu, Y.; Chen, T.; and Tong, Y. 2019. Federated machine learning: Concept and applications. *ACM TIST* 10(2):12:1–12:19.

Yu, H.; Shen, Z.; Miao, C.; Leung, C.; Lesser, V. R.; and Yang, Q. 2018. Building ethics into artificial intelligence. In *IJCAI*, 5527–5533.

Corrigendum

The spelling of coauthor Yusen Zan in the paper "Accelerating Ranking in E-Commerce Search Engines through Contextual Factor Selection" has been changed from Zan to Zhan to correct the typographical error.