Learning to Augment Imbalanced Data for Re-ranking Models

Zi-Hao Qiu,[1] Ying-Chun Jian,[1] Qing-Guo Chen,[2] Lijun Zhang[1]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
[2]Alibaba Group, Hangzhou, China
{qiuzh,jianyc}@lamda.nju.edu.cn,qingguo.cqg@alibaba-inc.com,zhanglj@lamda.nju.edu.cn

ABSTRACT

The conventional solution to learning to rank problems ranks individual documents by prediction scores greedily. Recent emerged re-ranking models, which take as input initial lists, aim to capture document interdependencies and directly generate the optimal ordered lists. Typically, a re-ranking model is learned from a set of labeled data, which can achieve favorable performance on average. However, it can be suboptimal for individual queries because the available training data is usually highly imbalanced. This problem is challenging due to the absence of informative data for some queries and furthermore, the lack of a good data augmentation policy.

In this paper, we propose a novel method named *Learning to Augment* (LTA), which mitigates the imbalance issue through learning to augment the initial lists for re-ranking models. Specifically, we first design a data generation model based on Gaussian Mixture Variational Autoencoder (GMVAE) for generating informative data. GMVAE imposes a mixture of Gaussians on the latent space, which allows it to cluster queries in an unsupervised manner and then generate new data with different query types using the learned components. Then, to obtain a good augmentation strategy (instead of heuristics), we design a teacher model that consists of two intelligent agents to determine how to generate new data for a given list and how to rank both the raw data and generated data to produce augmented lists, respectively. The teacher model leverages the feedback from the re-ranking model to optimize its augmentation policy by means of reinforcement learning. Our method offers a general learning paradigm that is applicable to both supervised and reinforced re-ranking models. Experimental results on benchmark learning to rank datasets show that our proposed method can significantly improve the performance of re-ranking models.

CCS CONCEPTS

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

KEYWORDS

Learning to rank; Data augmentation; Re-ranking

ACM Reference Format:
Zi-Hao Qiu,[1] Ying-Chun Jian,[1] Qing-Guo Chen,[2] Lijun Zhang[1] . 2021. Learning to Augment Imbalanced Data for Re-ranking Models. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482364

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482364

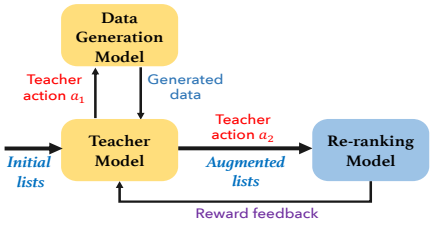


Figure 1: An overview of our data augmentation framework, which contains a data generation model to produce informative data and a teacher model for augmentation decision-making.

1 INTRODUCTION

Learning to rank (LTR) is an important machine learning area in the IT industry, powering online advertisement, search engines and various content recommendation services [20, 25]. Many existing LTR applications aim to generate and display an ordered list of "documents" to users (called a "slate" in Swaminathan et al. [28]; Sunehag et al. [27]), based on both user preferences and documents content. For large scale LTR systems, a common scalable approach is to first select a small set of candidate documents out of the entire document pool to form the initial lists. Then a function approximator such as a neural network called the "re-ranking model" [37] is used to rank the initial lists and generate result ordered lists to users. This two-step process is widely popular to solve large scale LTR problems due to its scalability and fast inference at serving time. The re-ranking models play an important role in this process because they have to model not only the relations between documents and users but also the mutual influence between documents, and directly generate optimal lists that maximize some type of user engagement feedback.

The re-ranking models are typically learned on labeled datasets and able to achieve favorable performance on average. However, such a model may be not optimal for individual queries because the available training data is usually imbalanced [1, 21, 30]. The imbalance problem mainly comes from two sources [35]: (1) given a query, the documents judged with high relevance level are much fewer than judged irrelevant documents; and (2) different queries typically have different amounts of relevance judgments. Some pioneering work has been proposed to mitigate this problem. Resampling methods [4] and ensemble methods [12] attempt to avoid learning a biased model with imbalanced data, but none of them provide informative augmentation for the training data. Generative Adversarial Networks (GAN) [14] based models can generate data

by adversarial learning. However, they capture the input data distribution and will intrinsically inherit the imbalance in the training set. Yu and Lam [35] employs query embedding and clustering to discover several query types first, then uses Adversarial Autoencoder (AAE) to generate new data with different query types. However, such query types may be biased and suboptimal for data generation.

While generating informative data is important to handle the data imbalance problem, we find that there are very limited attempts on building a good augmentation policy in LTR area. Existing data augmentation methods [19, 35] generate new data based on heuristic rules, which rely on the strong assumptions of the tasks and may be not optimal for some raw data. Therefore, an effective augmentation policy for re-ranking models should first decide *what kind of data should be generated* to mitigate the imbalance problem. Moreover, previous work suggests that the performance of re-ranking models is often sensitive to the order in which the input is processed [2, 3, 5]. Thus such policy should then decide *the order of both raw items*[1] *and generated items in the augmented list*. Besides, the augmentation policy should be formulated towards optimizing the overall performance of the re-ranking models, which is a much more principled way than manually crafting heuristics.

In this paper, we propose a novel and modular data augmentation framework for re-ranking models. Fig. 1 shows an overview of our framework, which contains two main models to achieve different goals: (1) a *data generation model* that can generate new data given different query types and different relevance levels; and (2) a *teacher model* that can make the augmentation decisions and be optimized towards the overall performance of the re-ranking model. To achieve the first goal, we devise the data generation model as a GMVAE [13]. GMVAEs learn a latent space by imposing a prior of a mixture of Gaussians and allowing the means and variances of each component to be determined by training. Each component of the mixture thus learns to encode a meaningful subset of training data. When applied to data generation, such a model could help discover clusters of query types with similar semantics. Moreover, inspired by Yu and Lam [35], the relevance information is disentangled from the latent representations of query types, so each component of the Gaussians mixture could be used as a generative model to produce data for a particular query type with different relevance levels. In our framework, this data generation model will be trained on the labeled datasets first and used by the teacher model for data augmentation next.

To achieve the second goal, we construct a teacher model to decide when given an initial list, how to generate new data and how to produce augmented lists for the re-ranking model. Specifically, there are two intelligent agents in the teacher model: a *generation agent*, determining the query types and relevance levels needed for new data generation (teacher action a_1 in Fig. 1), and a *ranking agent*, rearranging both raw items and new generated items and produce augmented lists (teacher action a_2 in Fig. 1). The training phase of the teacher model contains several episodes of sequential interactions between the teacher model and the re-ranking model. Based on the state information in each step, the teacher model updates its actions to improve the performance of the re-ranking model. The re-ranking model then performs its learning process based on the augmented lists from the teacher model, and provides reward signals (e.g., the NDCG@k on the held-out validation set) back to the teacher afterward. The teacher model then updates its parameters via policy gradient methods (e.g., REINFORCE [33]) by such rewards. The whole process is end-to-end trainable, exempt from the limitations of human-defined heuristics, thus we name our method as *Learning to Augment* (LTA).

To demonstrate and understand the effectiveness of our method, we conduct empirical experiments on large-scale learning to rank corpora and employ both supervised and reinforced re-ranking models. Experimental results show that our approach can significantly improve the performance. In addition, our analyses show that our data generation model can produce new data with high quality and the teacher model is able to learn a good strategy for handling data imbalance. To the best of our knowledge, this is the first work for handling data imbalance in learning to rank area from both data generation and augmentation strategy optimization.

2 RELATED WORK

A basic task for information retrieval is to rank a set of documents given a query, based on the relevance between a document and a query [9]. Learning to rank (LTR) refers to a group of techniques that attempts to solve ranking problems by using machine learning algorithms with the feature representations of query-document pairs. There are several types of LTR models, including point-wise, pair-wise, list-wise, and so on. Point-wise models [11, 18] treat the ranking problems as classification or regression tasks for each item. Pair-wise models [7, 8] convert the original problem into the internal ranking of pairs. List-wise models [2, 34] use well designed loss functions to directly optimize the ranking criteria. Group-wise models [3] and page-wise models [36] are proposed recently, which are similar to re-ranking models [37].

Re-ranking models aim to rank the initial lists given by the global ranking function and generate new ranked lists for users. They use the whole initial list as input and model the complex dependencies between items in different ways. Ai et al. [2] uses an unidirectional Gated Recurrent Unit (GRU) to encode the information of the whole list into the representation of each item. Pei et al. [22] optimizes the whole list by employing a transformer structure to encode the items in the list. Zhuang et al. [37] uses Long Short-Term Memory (LSTM) and Bello et al. [5] uses pointer network [31] not only to encode the whole list information, but also to generate the ranked list by a decoder. Slate optimization [5] is a close topic with LTR. Similar to the objective of re-ranking, it also aims to optimize some criteria of the whole slate (i.e. a list or a webpage).

Data augmentation is a promising direction for mitigating the imbalance problem, which includes a set of methods that introduce unobserved data via generative models. Such techniques have been employed in some LTR scenarios. Li et al. [19] proposes a attention-based sequence-to-sequence generative model for data augmentation in Point-Of-Interest (POI) recommendation. Yu and Lam [35] proposes a data generation model based on Adversarial Autoencoder (AAE) for tackling the data imbalance in LTR. They disentangle the relevance level information from the latent representations, so that their model can reconstruct data with specific

[1]The terms "document" and "item" are synonymous references and used interchangeably in this paper.

relevance levels. However, in order to discover different query types, they employ a separate query embedding and clustering process, which may not be optimal for data generation. In our work, we devise the data generator as a GMVAE, which is able to discover clusters of queries during training by imposing a prior of a mixture of Gaussians. Thus it can generate data with different query types naturally and improve the performance of re-ranking models. Besides, we employ an end-to-end trainable method to learn a good augmentation strategy, which is much more principled than heuristics rules used in previous methods.

3 PROBLEM FORMULATION

Each training sample $\boldsymbol{v}_{q,i}$ for LTR is the *feature vector derived from a pair of query q and document i*. Besides, a *relevance level* $r_{q,i}$, which measures the relevance between query q and document i, is associated with $\boldsymbol{v}_{q,i}$. Let n_r denote the number of relevance levels, so $r_{q,i}$ is an integer value in the range $[0, n_r - 1]$.

Given a query q, practical LTR systems first employ global ranking functions to select a small set of candidate items and form the initial list $L_q = [\boldsymbol{v}_{q,1}, \cdots, \boldsymbol{v}_{q,n}] = [\boldsymbol{v}_{q,i}]_{i=1}^{n}$. To further model the mutual influence between item-pairs and the interaction between users and items, a re-ranking model M is introduced and its loss function can be formulated as:

$$\mathcal{L}_{re-ranking} = \sum_{q \in Q} \ell\left(\{r_{q,i}, P(r_{q,i}|\boldsymbol{v}_{q,i}; \theta_M) | i \in L_q\}\right), \quad (1)$$

where Q is the set of all users' queries. $P(r_{q,i}|\boldsymbol{v}_{q,i}; \theta_M)$ is the predicted relevance level of item i given by the re-ranking model with parameter θ_M. ℓ is the loss computed with $r_{q,i}$ and $P(r_{q,i}|\boldsymbol{v}_{q,i}; \theta_M)$.

Our goal is to address the imbalance issue of the initial lists $L = \{L_q : q \in Q\}$ and improve the performance of re-ranking models. First, we employ a **data generation model** G to produce informative data. The model is trained on the whole dataset, and it can discover query clusters containing queries that are semantically close during training. Each cluster can be regarded as a *query type* labeled with a type ID. At generation time, the model produces a new item $\boldsymbol{v}^*$ given a query type e^* and a relevance level r^*:

$$\boldsymbol{v}^* = G(r^*, e^*; \theta_G), \quad (2)$$

where θ_G is the parameters for the data generation model. More details of this model can be found in section 4.

We further design a **teacher model** to make augmentation decisions for initial lists as follows:

- Decide *how to produce new items*. Specifically, the *generation agent* in the teacher model needs to decide k relevance levels $R^* = \{r_1^*, \cdots, r_k^*\}$ and k query types $E^* = \{e_1^*, \cdots, e_k^*\}$ for an given initial list L_q.
- Generate k new items $\boldsymbol{V}^* = \{\boldsymbol{v}_1^*, \cdots, \boldsymbol{v}_k^*\}$ by feeding R^* and E^* into the data generation model as in Eq. (2).
- Decide *how to rank both raw items in L_q and generated items in $\boldsymbol{V}^*$ to produce a result list L_q' with n items* for the re-ranking model. The *ranking agent* in the teacher model is responsible for this process.

The loss function for the re-ranking model is presented in Eq. (1). Thus, the training process of this model actually corresponds to the following optimization problem:

$$\theta_M^* = \underset{\theta_M}{argmin}\, \mathcal{L}_{re-ranking} \triangleq \mu_M(L).$$

Here we use $\mu_M(\cdot)$ to denote the learning algorithm for re-ranking model M. As a summary, this learning algorithm takes the input lists L as inputs, and outputs a function with parameters θ_M^* by minimizing the empirical risk $\mathcal{L}_{re-ranking}$.

The goal of the teacher is to augment the initial input lists for the re-ranking model such that it can achieve better performance, e.g., measured by the evaluation measure $\mathcal{M}$ on a held-out validation set D_{valid}. So the goal of the teacher model is

$$\max_{L}\, \mathcal{M}(\mu_M(L), D_{valid}).$$

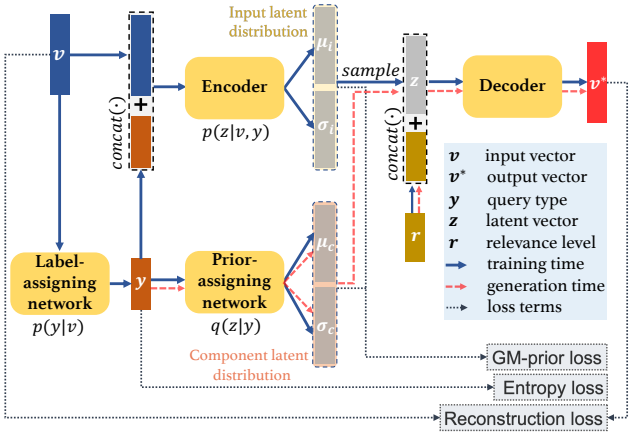


Figure 2: The data generation model based on GMVAE, where GM-prior loss is the KL-divergence between the input latent distribution and the component latent distribution.

4 DATA GENERATION MODEL

In this section, we first demonstrate the architecture of our proposed data generation model. Then we introduce the training and generation process of the model.

4.1 Architecture

In a Variational Autoencoder (VAE) [17], the encoder takes in an input vector and outputs a mean vector and a variance vector that parameterize the input latent distribution corresponding to the input vector. A latent vector is then sampled from such distribution and forwarded through the decoder to produce the reconstruction of the input vector. Two losses are minimized in the training process: (1) the reconstruction loss between the input and output vectors and (2) the KL-divergence between the latent distribution and the Gaussian prior. Fig. 2 demonstrates our data generation model based on a GMVAE [13]. A GMVAE with a m-component GM prior makes the following two modifications to such a VAE:

Label-assigning network The input vector $\boldsymbol{v}$ is first passed to a *label-assigning* network, whose last layer, the Gumbel-Softmax layer [15], produces a m-dimensional label. Its i-th dimension contains the probability that the input vector belongs to the i-th GM component. This set of probabilities is gradually enforced to be concentrated on one component during training. Therefore, during

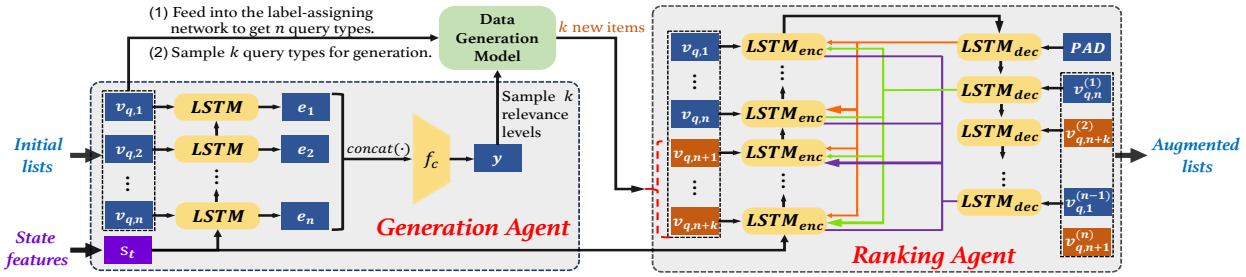


Figure 3: The structure of our teacher model, which consists of a generation agent and a ranking agent.

later stages in training and generation, given the input vector $\boldsymbol{v}$, this network outputs one-hot labels $\boldsymbol{y} \in \mathbb{R}^m$. This allows our model to cluster items in an unsupervised manner, and the labels produced by the label-assigning network are the *query types* for input items. So the label-assigning network models a distribution $p(\boldsymbol{y}|\boldsymbol{v})$. The concatenation of the input vector $\boldsymbol{v}$ and the label $\boldsymbol{y}$ is then fed into the encoder and get a latent representation $\boldsymbol{z}$ for the decoder. Thus the encoder actually models a distribution $p(\boldsymbol{z}|\boldsymbol{v}, \boldsymbol{y})$.

Prior-assigning network For each input vector $\boldsymbol{v}$, we now minimize the KL-divergence between its latent distribution and its assigned *component latent distribution*, which is parameterized by the *component mean* vector and *component variance* vector obtained by forwarding its label $\boldsymbol{y}$ through the *prior-assigning* network. This new loss term is called the *GM-prior loss*. Let $\boldsymbol{z}$ denote the latent representation sampled from the component latent distribution for label $\boldsymbol{y}$, then this network models a distribution $q(\boldsymbol{z}|\boldsymbol{y})$.

Having the prior-assigning network allows us to produce latent representations $\boldsymbol{z}$ for specific one-hot query types $\boldsymbol{y}$. To facilitate the support of generating a feature vector with a target relevance level, inspired by Yu and Lam [35], the relevance levels are encoded into one-hot relevance indicator vector $\boldsymbol{r}$ and added as the input for the decoder. This disentanglement enforces the encoder to generate relevance-independent latent representations. Therefore the generation procedure can be controlled via concatenating a specific target relevance level with the latent representation produced by a specific query type. The detailed structure of each network will be described in section 6.1.3.

The main advantage of our data generation model is that it is a *unified* framework for queries clustering using learnable GM prior and data reconstruction, which allows us to generate new data with specific query types accurately. On the contrary, the data generation model proposed by Yu and Lam [35] employs a *separate* process, including query embedding and K-means clustering, to discover the query type for each item first. Then they randomly select a fixed mixture of Gaussians as a prior, and train the AAE model by reconstructing the items belonging to each query type from the corresponding Gaussian distribution. Therefore, these query types may be biased and suboptimal for data generation.

4.2 Training and Generation

To train our data generation model, we seek to minimize the following objective, which is the sum of the reconstruction loss, the GM prior loss and an additional entropy term:

$$\mathcal{L}_G = \|\boldsymbol{v} - \boldsymbol{v}_{recon}\|_2^2 + \mathbb{E}_{p(\boldsymbol{y},\boldsymbol{z}|\boldsymbol{v})}\left(\alpha \log \frac{p(\boldsymbol{z}|\boldsymbol{v},\boldsymbol{y})}{q(\boldsymbol{z}|\boldsymbol{y})} + \beta \log p(\boldsymbol{y}|\boldsymbol{v})\right),$$

where $\boldsymbol{v}$ and $\boldsymbol{v}_{recon}$ are the input vector and reconstructed vector respectively, α is a coefficient to balance these two terms, and $p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{v}) = p(\boldsymbol{y}|\boldsymbol{v}) \cdot p(\boldsymbol{z}|\boldsymbol{v}, \boldsymbol{y})$. Besides, we add an additional entropy loss to encourage the model to explore more GM components for the given inputs. β is a hyper-parameter balancing the last entropy term.

When generation, given a query type $\boldsymbol{y}$ and a relevance level $\boldsymbol{r}$, a new item (feature vector) $\boldsymbol{v}^*$ is generated by first sampling a latent representation $\boldsymbol{z}$ form the learned prior-assigning network $q(\boldsymbol{z}|\boldsymbol{y})$, concatenating with the one-hot encoding of the relevance level $\boldsymbol{r}$ and passed into the learned decoder, and finally generating $\boldsymbol{v}^*$.

5 TEACHER MODEL

As reviewed in section 2, existing works that consider augmentation strategies simply employ some heuristic rules. In this section, we propose to model the augmentation strategy via a sequential decision process. We first elaborate in detail on the structure of our teacher model. Then we introduce how to leverage reinforcement learning to model the interaction between the re-ranking model and teacher model, and the corresponding optimization process.

5.1 Structure

The structure of the teacher model is shown in Fig. 3. We can see that it contains two main components: generation agent and ranking agent. We will illustrate the details of the architecture and the corresponding forward process in this section.

Generation Agent This agent decides *what kind of new data should be generated* for the given initial lists. As show in Fig. 3, it contains an LSTM cell as the encoder for the input lists. At each encoding step $i \leq n$, the encoder reads the input vector $\boldsymbol{v}_{q,i}$ and outputs a result vector $\boldsymbol{e}_i$, thus transforming the input sequence $[\boldsymbol{v}_{q,i}]_{i=1}^n$ into a sequence of latent states $[\boldsymbol{e}_i]_{i=1}^n$. These latent states can be seen as a compact representation of the initial list. Then the latent states $[\boldsymbol{e}_i]_{i=1}^n$ are concatenated and passed through a fully connected network f_c with softmax as the output layer and produce a result probability matrix $\boldsymbol{y} \in \mathbb{R}^{k \times n_r}$, which can be regarded as k n_r-dimensional distributions. Such process can be formulated as:

$$\boldsymbol{e}_i = LSTM(\boldsymbol{v}_{q,i}),$$
$$\boldsymbol{y} = softmax\left(f_c(concat([\boldsymbol{e}_i]_{i=1}^n))\right) \in \mathbb{R}^{k \times n_r}.$$

The teacher model then can sample k relevance levels from $\boldsymbol{y}$. This is the first action that the teacher takes, so we set the action as a_1 and denote the decision process as $\phi_T(a_1|s_t)$. s_t is the state features for both agents and will be introduced in section 5.2.

To get the query types that are needed for generating new data, we first feed all items in the initial list into the *label-assigning network* mentioned in section 4.1 and get query types for each item, then sample k query types from them. Now we can produce k new items by feeding these k query types and relevance levels into the data generation model. The detailed data generation process has been described in section 4.2.

Ranking Agent This component is used to *rank both raw items and generated items to produce augmented lists* for the re-ranking model. As shown in Fig. 3, we frame this process as a sequence-to-sequence (Seq2Seq) problem and adopt the pointer network, which is a Seq2Seq model with an attention mechanism for pointing at position in the input [31]. Such a model contains an encoder and a decoder, both of which use LSTM cells. At encoding stage, the encoder transforms the input sequence $[\boldsymbol{v}_{q,i}]_{i=1}^{n+k}$, where $[\boldsymbol{v}_{q,i}]_{i=1}^{n}$ are raw items and $[\boldsymbol{v}_{q,i}]_{i=n+1}^{n+k}$ are generated items, into a ρ-dimensional latent states sequence $[\boldsymbol{e}_i]_{i=1}^{n+k}$. At each decoding step j, the decoder outputs a vectors $\boldsymbol{d}_j \in \mathbb{R}^{\rho}$, which are used as queries in the attention function. The attention function takes as input the query $\boldsymbol{d}_j$ and latent states $[\boldsymbol{e}_i]_{i=1}^{n+k}$ and produces a probability distribution over the next item to include in the output list as follows:

$$t_i^j = \boldsymbol{W}_v^T \tanh(\boldsymbol{W}_{enc} \cdot \boldsymbol{e}_i + \boldsymbol{W}_{dec} \cdot \boldsymbol{d}_j)$$

$$p(\pi_j = i | \pi_{<j}, \boldsymbol{v}) = \begin{cases} e^{t_i^j} \Big/ \sum_{k \notin \pi_{<j}} e^{t_k^j} & if\ i \notin \pi_{<j} \\ 0 & if\ i \in \pi_{<j} \end{cases}$$

where $\boldsymbol{W}_{enc}, \boldsymbol{W}_{dec} \in \mathbb{R}^{\rho \times \rho}$ and $\boldsymbol{W}_v \in \mathbb{R}^{\rho}$ are learnable parameters for the attention function, $\pi \in \Pi$ denotes a permutation of item and each $\pi_j \in \{1, \cdots, n+k\}$ denotes the index of the item in position j. Then an item can be selected by sampling, and its embedding is fed as input to the next decoding step. The input to the first decoding step is a learnable vector denoted as *'PAD'* in Fig. 3. The decoding step will run for n times and finally we get the augmented list for the re-ranking model. This process can be regarded as the second action that the teacher takes, so we set the action as a_2 and denote the decision process as $\phi_T(a_2|s_t)$.

5.2 Interaction and Optimization

The overall interactive process between the teacher model and the re-ranking model can be modeled via reinforcement learning: (1) at the t-th step, given the *state* s_t and initial lists, the teacher model takes augmentation *action* $a_{1,t}$ and $a_{2,t}$ to produce augmented lists; (2) the re-ranking model updates itself based on these lists, changes the environment to s_{t+1} and then provides a *reward* r_t to the teacher model; and (3) the teacher model then updates its parameters θ_T to maximize the accumulated reward. Finally, the whole interactive process stops when the re-ranking model gets converged, forming one *episode* of the teacher model training.

To encourage good performance, assuming the length of input lists is n, we directly employ the ranking measure NDCG@n on validation data as the reward r_t. For state features s_t, we adopt three categories features to compose s_t:

- *Data features*, contain feature vectors and relevance levels (we use one-hot encoding) of items in the input lists. Such features are commonly used in curriculum learning [6, 29].
- *Re-ranking model features*, include the signals reflecting how well the current model is trained (measured by NDCG@10) and the model training progress (i.e., iteration).
- *Features for the combination of data and re-ranking model*. Here assume the length of the initial lists is n. The initial lists will be ranked by the re-ranking model first, and then we calculate NDCG@k ($k \in \{1, \cdots, n\}$) on the ranked lists. Such n values for each list can form a n-dimensional vector representing how the re-ranking model performs on the initial input lists.

The state features s_t are computed after the arrival of each mini-batch training data, and then added as the input for both agents as in Fig. 3. They can effectively and efficiently represent the state for the teacher model and enable the model to learn a better policy. The teacher model is trained by maximizing the expected reward:

$$J(\theta_T) = \mathbb{E}_{\phi_T(a_1|s)\phi_T(a_2|s)} R(s, a_1, a_2),$$

where a_1 and a_2 are the actions made by the generation agent and the ranking agent respectively, $R(s, a_1, a_2)$ is the state-action value function. Since $R(s, a_1, a_2)$ is non-differentiable w.r.t. the parameters of teacher model θ_T, we use REINFORCE [33], a likelihood ratio policy gradient algorithm to optimize $J(\theta_T)$ based on the gradient:

$$\nabla_{\theta_T} = \mathbb{E}_{\phi_T} \left[\sum_{t=1}^{T} \nabla_{\theta_T} \left(\log \phi_T(a_{1,t}|s_t) + \log \phi_T(a_{2,t}|s_t) \right) G_{t:T} \right],$$

where $G_{t:T} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ is the discounted cumulative reward. To reduce the variance of policy gradient method, a typical technique is subtracting baseline values from the original rewards. Inspired by the self-critical algorithm [24], at each training iteration, the teacher samples several different actions and uses the average of their cumulative rewards as the baseline. So the final gradient is:

$$\nabla_{\theta_T} = \mathbb{E}_{\phi_T} \left[\sum_{t=1}^{T} \nabla_{\theta_T} \left(\log \phi_T(a_{1,t}|s_t) + \log \phi_T(a_{2,t}|s_t) \right) (G_{t:T} - V_t) \right],$$

where V_t is the average cumulative reward of different actions in iteration t. As a result, rewards will be increased as the teacher increase the generation probability of good lists while decreasing the chances of worse lists generation. Finally, at serving time, the well-trained teacher model can generate not just a single but *multiple* augmented lists for any given list using sampling policy [5].

Our proposed augmentation method is flexible and controllable by using both the generation agent and the ranking agent. Furthermore, we bridge the gap between the isolated processes of data augmentation and re-ranking model optimization by the interactive learning process, which allows our method to generate more proper training samples for the re-ranking model than heuristic rules.

6 EXPERIMENTS

We conduct experiments to answer the following questions:
RQ1: How is the performance of our proposed LTA compared with existing baseline methods?
RQ2: How do the components of LTA affect its performance?
RQ3: How do the hyper-parameters affect its performance?
RQ4: Where are the improvements of our framework come from?

We first present experimental settings, followed by results and analyses to answer each research question.

Table 1: Statistics of Datasets.

Dataset	MS 10K	MS 30K	Yahoo! set 1
Query	10,000	30,000	29,921
Q-D pair	1,200,192	3,771,125	709,877
Q-D pair with r=4	8,881	30,435	13,402
Q-D pair with r=3	21,317	69,010	54,473
Q-D pair with r=2	159,451	504,958	202,700
Q-D pair with r=1	386,280	1,225,770	254,110
Q-D pair with r=0	624,263	1,940,952	185,192
max Q-D pair per query	908	1,245	135
min Q-D pair per query	1	1	1

6.1 Experimental Settings

6.1.1 Datasets. In our experiments, we use three benchmark datasets, Microsoft 10k, Microsoft 30k [23][2] and Yahoo! Webscope v2.0 set 1[3]. We purposefully choose them because as far as we know, these are the largest public learning to rank datasets from commercial English search engines. Due to privacy concerns, these datasets do not disclose any text information and only provide feature vectors for each query-document pair (Q-D pair). For these datasets, documents for each query are annotated with 5 different relevance levels, namely from 0 for irrelevance to 4 for high relevance. The Microsoft datasets are partitioned into five folds and define cross validation by using three folds for training, one fold for validation and one fold for testing. The Yahoo! dataset splits the queries arbitrarily and uses 19944 for training, 2994 for validation and 6983 for testing.

The statistics of these datasets are listed in Table 1. We can observe that query-document pairs with high relevance levels are relatively scarce in the training set. In addition, the large range of the number of Q-D pairs for a query indicates the data imbalance regarding queries. To reflect such imbalance in our experiments, when the global ranking models do the initial retrieval for each query, they will generate initial lists that are proportional to the number of Q-D pairs for this query.

6.1.2 Baselines. To the best of our knowledge, there is no existing data augmentation method for re-ranking models. To back up our claim, we compare LTA with several relevant approaches. Considering different queries having different number of initial lists, we employ a simple strategy for all baselines. Assuming there are at most m initial lists belonging to one query, in order to augment a query with n ($n \leq m$) lists, baseline methods will take as input each list for this query and produce $\frac{m-n}{n}$ augmented lists. Thus, every query will have m lists in the end. All baseline methods employ this strategy and differ in *how to produce augmented lists given the initial list.* The details of each method are as follows:

- **Random Augment.** This method produces augmented lists by randomly rearrange the items in the initial list. Experiments conducted by Ai et al. [2] have shown that this strategy can mitigate the overfitting of the model and improve the performance.
- **AAE-based Augment [35].** This method employs an adversarial autoencoder (AAE) to generate informative data and uses heuristic augmentation rules to handle the data imbalance issue. Here we adopt its heuristic rules to produce augmented lists for re-ranking models. The dimension of latent representations in the AAE model and the number of query types are tuned in the range of {5,10,15,20,25} and {5,10,15,20}, respectively.
- **GMVAE-based Augment.** This method employs GMVAE (our data generation model) instead of AAE to generate new items. To produce augmented lists, it uses the same heuristic rules as in AAE-based Augmentation. For fair comparisons, we adapt the same encoder and decoder architecture with the AAE model and match their number of parameters to 4.1 M.

6.1.3 Model Training. We use two types of global learning to rank models to do the initial retrieval: SVMrank [16] and LambdaMART [8]. SVMrank is a well-known ranking model trained with pairwise losses while LambdaMART is the famous LTR algorithm trained with listwise losses. In this paper, we use the implementation of SVMrank from Joachims[4] and the implementation of LambdaMART from RankLib[5]. We tune the global ranking model on the validation set based on NDCG@10 and select the best one as our initial ranking function. For SVMrank, we tune parameter c from 20 to 200; for LambdaMART, we tune tree numbers from 100 to 1000.

For the re-ranking models, we try both a supervised model and a reinforced model. We implement a supervised re-ranking model named PRM [22], which optimizes the whole list by employing a transformer structure. The hidden dimensionality is set to 1024 and the batch size is set to 256. The learning rate of Adam optimizer in our implementation is the same with [22]. For the reinforced model, we employ Seq2Slate [5], which uses a sequence-to-sequence model for ranking and can be optimized by REINFORCE algorithm [33]. Inspired by literature [5], the batch size is set to 256 and the LSTM cells have 128 hidden units. The model is trained with Adam optimizer and an initial learning rate of 0.0003 decayed every 1000 steps by a factor of 0.96.

Our framework contains a data generation model and a teacher model. In the data generation model for Microsoft datasets, the label-assigning network, encoder, prior-assigning network, and the decoder are all fully connected networks with the layer size as 136-512-10, 146-512-256, 10-256, and 261-512-136, respectively. As for Yahoo! dataset, the network structures are 700-512-20, 720-512-256, 20-256, and 261-512-700, respectively. The number of GM components is set to 10 for Microsoft datasets and 20 for Yahoo! set 1. We set 5 new items to be generated for each initial list. The teacher model contains a generation agent and a ranking agent. The generation agent has an LSTM cell with 256 units and a fully connected network with the layer size as 256-25. The ranking agent has two LSTM cells with 256 units. All network parameters are initialized uniformly at random in [-0.1,0.1] and optimized by Adam optimizer. The batch size is 256 and the learning rate for the data generation model and teacher model are 0.003 and 0.0002, respectively. We train the teacher model till convergence, i.e., the terminal reward of the re-ranking model stops improving for several episodes.

6.1.4 Hyper-parameter Settings. There are five hyper-parameters for our LTA: the size of initial lists n, the number of GM components

[2]https://www.microsoft.com/en-us/research/project/mslr/
[3]https://webscope.sandbox.yahoo.com
[4]https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html
[5]https://sourceforge.net/p/lemur/wiki/RankLib/

Table 2: Performance (NDCG@10 / ERR@10 / Precision@10) of baseline methods and LTA on MS 10K, MS 30K and Yahoo! set 1. $*$ indicates significant improvements over the corresponding original re-ranking models, at level 0.01.

Global Ranking Algorithm	SVMrank		LambdaMART	
Re-ranking model	PRM	Seq2Slate	PRM	Seq2Slate
	MS 10K			
Original	0.3618 / 0.2614 / 0.6048	0.3635 / 0.2622 / 0.6069	0.3977 / 0.3087 / 0.6309	0.4041 / 0.3125 / 0.6371
Random Augment	0.3689 / 0.2732 / 0.6109	0.3710 / 0.2759 / 0.6125	0.4024 / 0.3168 / 0.6379	0.4125 / 0.3247 / 0.6433
AAE-based Augment	0.3789 / 0.2774 / 0.6151	0.3802 / 0.2797 / 0.6189	0.4125 / 0.3206 / 0.6452	0.4197 / 0.3289 / 0.6498
GMVAE-based Augment	0.3826*/ 0.2808*/ 0.6179*	0.3875*/ 0.2831*/ 0.6209*	0.4161*/ 0.3243*/ 0.6487*	0.4208*/ 0.3337*/ 0.6533*
LTA(ours)	**0.3943*/0.2976*/0.6247***	**0.3986*/0.2994*/0.6279***	**0.4239*/0.3353*/0.6546***	**0.4301*/0.3412*/0.6608***
	MS 30K			
Original	0.3624 / 0.2613 / 0.6044	0.3689 / 0.2677 / 0.6108	0.4107 / 0.3179 / 0.6401	0.4162 / 0.3214 / 0.6462
Random Augment	0.3712 / 0.2709 / 0.6132	0.3724 / 0.2788 / 0.6175	0.4170 / 0.3249 / 0.6461	0.4221 / 0.3287 / 0.6520
AAE-based Augment	0.3812 / 0.2793 / 0.6185	0.3843 / 0.2832 / 0.6204	0.4269 / 0.3263 / 0.6502	0.4287 / 0.3311 / 0.6568
GMVAE-based Augment	0.3895*/ 0.2846*/ 0.6236*	0.3912*/ 0.2915*/ 0.6279*	0.4282*/ 0.3332*/ 0.6579*	0.4313*/ 0.3391*/ 0.6607*
LTA(ours)	**0.3959*/0.2991*/0.6303***	**0.4018*/0.3027*/0.6385***	**0.4321*/0.3405*/0.6627***	**0.4405*/0.3468*/0.6702***
	Yahoo! set 1			
Original	0.6919 / 0.4079 / 0.7094	0.6942 / 0.4132 / 0.7142	0.6976 / 0.4189/ 0.7144	0.7019 / 0.4231 / 0.7183
Random Augment	0.6983 / 0.4123 / 0.7129	0.7004 / 0.4192 / 0.7198	0.7021 / 0.4259/ 0.7193	0.7084 / 0.4318 / 0.7234
AAE-based Augment	0.7043 / 0.4164 / 0.7179	0.7054 / 0.4234 / 0.7234	0.7104 / 0.4329/ 0.7205	0.7139 / 0.4367 / 0.7278
GMVAE-based Augment	0.7098*/ 0.4245*/ 0.7236*	0.7126*/ 0.4287*/ 0.7279*	0.7149*/ 0.4387*/ 0.7233*	0.7209*/ 0.4396*/ 0.7301*
LTA(ours)	**0.7152*/0.4308*/0.7289***	**0.7182*/0.4327*/0.7311***	**0.7193*/0.4419*/0.7284***	**0.7252*/0.4438*/0.7316***

m in the data generation model, the number of new generated items k for each initial list, coefficients α and β for training data generation model. We tune n from 10 to 50, m from 5 to 20, k from 2 to 10, and α and β in {0.5,0.7,1.0,1.2,1.5}. We present hyper-parameter studies in section 6.4 and set them according to the results to achieve satisfactory performance. We train our models on one Nvidia Tesla V100 GPU with 32 GB memory. The training of the data generation model takes about 1 hour for convergence, while the training of the teacher model takes about 11 hours for convergence. The whole augmentation process takes about 2 to 3 ms for each initial list.

6.1.5 Evaluation Protocols. Our datasets have five-level relevance judgments, from 0 (irrelevant) to 4 (perfectly relevant), so we use two types of multi-label ranking metrics to evaluate the re-ranked lists. They are Normalized Discounted Cumulative Gain (NDCG) and Expected Reciprocal Rank (ERR) [10]. We also use Precision to reflect the fraction of clicked items in the re-ranked lists for all test samples. For these three metrics, we report results at rank 10 to show the performance. Statistical differences are computed with the Fisher randomization test [26] ($p \leq 0.01$).

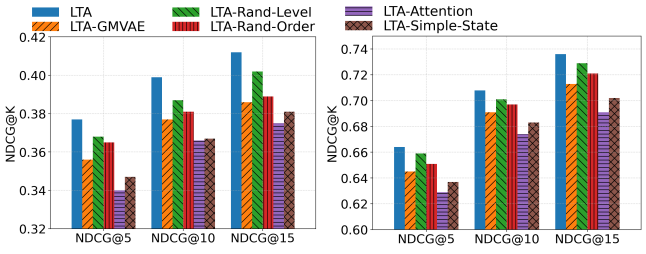
6.2 Overall Performance (RQ1)

The overall performance of two re-ranking models with different data augmentation methods on MS 10K, MS 30K and Yahoo! set 1 are presented in Table 2. 'Original' represents training the re-ranking models using raw data without any augmentation operation. From the table, we have the following observations:

- According to the results, Random Augment generally improves LTR performance over Original, which indicates even a simple strategy can mitigate data imbalance regarding queries. Besides, we can observe that both AAE-based Augment and GMVAE-based Augment perform better than Random-Augment, which implies that new informative data is critical for mitigating data imbalance regarding relevance levels.
- In particular, GMVAE-based Augment outperforms AAE-based Augment on all datasets. These results signify the effectiveness of our data generation model, which is attributed to the dedicated design of the model. Our model unifies unsupervised query types discovering and supervised data reconstruction as a whole, which allows us to generate features with different query types naturally. We will present detailed analyses of these two data generation models in section 6.5.
- LTA consistently produces better results than GMVAE-based Augment. It validates our belief that the sequence of the input lists is important to the final performance of the re-ranking model. Our method shows strength in modeling the relationship between items and producing augmented lists which can help the re-ranking models learn well. In-depth analyses about our proposed method will be given in section 6.5.
- We also observe that our method is able to bring stable and significant improvements to both supervised re-ranking model PRM and reinforced re-reranking model Seq2Slate. This demonstrates that our proposed method is general and can be applied to different ranking models.
- Compared to other datasets, we notice that the improvements from the augmentation baselines are relatively small on Yahoo! set 1. This, however, is not surprising considering the special properties of this dataset. From Table 1, we can observe that Yahoo! set 1 is more balanced than the other two datasets. These results indicate that the more imbalanced the raw data is, the greater the improvement our method will bring.

6.3 Ablation Studies (RQ2)

The strengths of LTA come from two novel components: (1) the data generation model that can discover queries types automatically and generate new item features given specific query types and relevance levels; and (2) the teacher model with two agents that augments the initial lists with a learned policy. To justify the rationality of LTA, we study the following five variants:



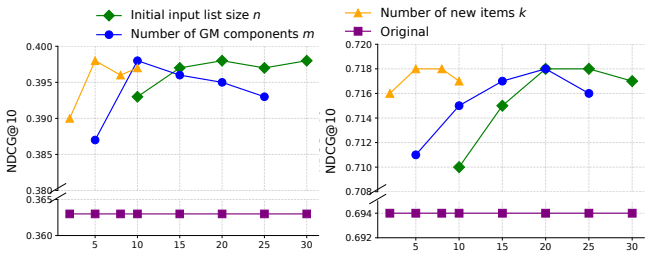
(a) MS 10K. (b) Yahoo! set 1.

Figure 4: Augmentation performance of LTA and its five variants with different designs being disabled.

- **LTA-GMVAE**, which removes the data generation model from LTA. At this time, we augment the initial lists by training the ranking agent in the teacher model to rearrange the initial lists.
- **LTA-Rand-Level**, which disables the generation agent in the teacher model. We simply use random relevance levels to generate new items.
- **LTA-Rand-Order**, which disables the ranking agent in the teacher model, and we randomly replace the items in the initial lists with new generated items to produce augmented lists.
- **LTA-Attention**, which uses attention model introduced by Pei et al. [22] instead of LSTM cells in the teacher model to extract features from the input lists. For fair comparisons, we match the number of parameters of the attention model and LSTM cell.
- **LTA-Simple-State**, which removes the *features for the combination of data and re-ranking model* that mentioned in section 5.2 from the state representation s_t.

For the five variants, we keep all hyper-parameters the same as the optimal settings of LTA. Fig. 4 shows the performance of our proposed LTA and the five variants on Microsoft 10k and Yahoo! set 1. To save space, we omit the results on Microsoft 30k and the results of ERR@K and Precision@K which show the same trend. We have the following observations:

- Regarding the design of our data generation model, LTA performs better than LTA-GMVAE by 6.34% and 4.56% on NDCG@10 in MS 10K and Yahoo! set 1, which shows the effectiveness of the informative data. This finding is consistent with prior work [35], which also verifies the importance of new generated informative data for handling data imbalance in LTR.
- Compared with LTA-Rand-Level in MS 10K, LTA achieves 3.43%, 4.17%, and 6.63% improvements on NDCG@5, NDCG@10, and NDCG@15, respectively. These results signify the effectiveness of the generation agent, which is attributed to its ability to determine what types of data should be generated in a principled manner instead of heuristics. Besides, compared with LTA-Rand-Order in MS 10K, LTA achieves 3.87%, 4.65%, and 7.05% improvements on NDCG@5, NDCG@10, and NDCG@15, respectively. These results demonstrate the necessity of the ranking agent, which is able to capture the relationship between items and arrange them properly to produce augmented lists. Existing studies on the re-ranking problem also demonstrate the critical role of the sequence of input lists [3, 5].
- Regarding the structure of our models, our method significantly outperforms LTA-Attention by 9.88% and 7.56% on NDCG@10 in MS 10K and Yahoo! set 1, respectively. This indicates the LSTM structure we employed in our model is a better choice for modeling the sequence relationship between items than attention models.
- When the teacher model is trained with a simpler state representation, the performance of LTA in MS 10K drops by 8.64%, 9.41% and 10.12% on NDCG@5, NDCG@10, and NDCG@15, respectively. These results indicate that the feature, which represents the combination of both data and re-ranking model, enables the teacher model to learn a better augmentation policy.



(a) MS 10K. (b) Yahoo! set 1.

Figure 5: The performance of LTA on MS 10K and Yahoo! set 1 with different hyper-parameters.

6.4 Hyper-parameter Studies (RQ3)

We study how important hyper-parameters affect the performance, more specifically, the size of initial input lists, the number of GM components in the data generation model, and the number of new generated items for each initial list.

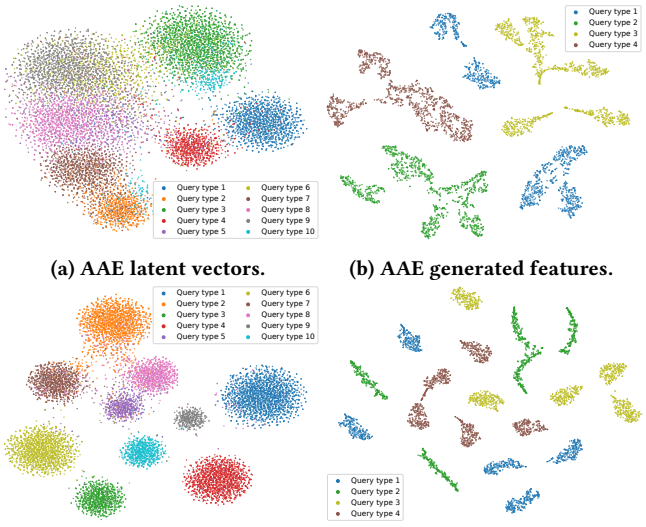
6.4.1 Size of initial input lists n. We fix other hyper-parameters and adjust the size of initial input lists from [10,15,20,25,30]. As shown in Fig. 5, the performances of LTA become better when n increases from 10 to 20 and are stable when n increase from 20 to 30. This is probably because when n is relatively small, the increase of n would introduce more relevant items in the inputs for the teacher model and give the model more chance to learn a good augmentation policy. When n is large enough, however, increasing n only brings irrelevant items into the input lists.

6.4.2 Number of GM components m. We adjust m from [5,10,15,20,25] and observe that LTA achieves the best performance on MS 10K and Yahoo! set 1 with 10 and 20 components, respectively. On both datasets, further increasing GM components degrades the performance. These results suggest that the optimal m varies, depending on the features of the dataset.

6.4.3 Number of new generated items k. We tune k in range [2,5,8,10]. It can be observed that for both datasets the improvement increases when k increases from 2 to 5. The performance improvement is negligible when k exceeds a certain value. In both cases, using $k = 5$ is beneficial.

6.5 In-depth Analyses (RQ4)

In this section, we conduct in-depth analyses to shed some light on how our framework improves the performance compared with other baseline methods. Our analyses focus on two questions: (1) What is the quality of the item features generated by our data



(a) AAE latent vectors. (b) AAE generated features.

(c) GMVAE latent vectors. (d) GMVAE generated features.

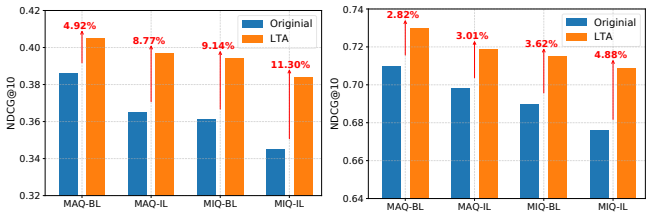
Figure 6: The arrangement of latent vectors and generated features (projected using t-SNE) of AAE and GMVAE. Due to limited space, we include four query types in (b) and (d).

generation model? (2) What kind of initial list received more improvements from our teacher model? For analysis purposes, we use the initial lists ranked by SVMrank on Microsoft 10k and employ PRM as the re-ranking model.

6.5.1 Visual Inspection of GMVAE Generations. Our data generation model can discover different query types in an unsupervised manner during training, and then generate new item features given specific query types and relevance levels. From section 6.2, we can see that our data generation model achieves better performance than that based on AAE. To better understand these results, we employ t-SNE, a cluster-preserving algorithm, to visualize the arrangement of the latent vectors extracted by the encoders and generated features from AAE and GMVAE in Fig. 6.

One can observe that the latent vectors and generated features from GMVAE are well-separated. Moreover, the generated features produced by GMVAE from the same query type are in 5 well-separated clusters, corresponding 5 different relevance levels. On the other side, for the AAE, we observe that the cluster boundaries of both latent vectors and generated features from the same query type are unclear. These results demonstrate the effectiveness of our data generation model based on GMVAE, which can discover query types in a principled way and generate high-quality item features with different query types and relevance levels.

6.5.2 Performance of Different Initial Lists Types. We divide queries into two groups: *majority* queries mean those queries with more than N lists, otherwise *minority* queries mean those queries with less than N lists. For simplicity, we make these two groups have the same size and set the threshold N accordingly. For initial lists, we divide them into two groups: *balanced* lists mean those lists with balanced relevance levels, otherwise *imbalanced* lists mean those lists with imbalanced relevance levels. Here we employ the entropy of relevance levels to measure the degree of balance, and set the entropy threshold to make these two groups have the same size.



(a) MS 10K. (b) Yahoo! set 1.

Figure 7: Augmentation performance of LTA on MS 10K and Yahoo! set 1 grouped by four input list types: majority queries-balanced lists (MAQ-BL), majority queries-imbalanced lists (MAQ-IL), minority queries-balanced lists (MIQ-BL), and minority queries-imbalanced lists (MIQ-IL).

Then we cross query groups and list groups, dividing all initial lists into four types: *majority queries-balanced* lists (MAQ-BL), *minority queries-balanced* lists (MIQ-BL), *majority queries-imbalanced* lists (MAQ-IL), and *minority queries-imbalanced* lists (MIQ-IL). We then perform the evaluation on each type of lists. Fig. 7 shows the performance of the re-ranking models that are trained on original lists and LTA-augmented lists, respectively, where red arrows indicate the improvement percentages.

We observe that LTA improves the performance of *minority queries-imbalanced* lists (MIQ-IL) by the largest margin. On *minority queries-balanced* lists (MIQ-BL) and *majority queries-imbalanced* lists (MAQ-IL), LTA also achieves significant improvements. From these results, we conclude that the improvements of our framework over the original models are mainly from the augmentation for minority queries and imbalanced lists. These results show the strong ability of our framework in mitigating data imbalance regarding both queries and relevance levels.

7 CONCLUSION

In this work, we investigate the data augmentation for re-ranking models. To address the data imbalance issues, we propose Learning to Augment (LTA) approach, which consists (1) a well-performed data generation model based on GMVAE that can generate high-quality informative data; and (2) an end-to-end trainable teacher model that consists of two agents to decide how to generate new data and how to produce augmented lists, respectively. We conduct experiments on three datasets, providing extensive results and analyses on the effectiveness and rationality of LTA.

There are many directions to explore in future. First, we plan to investigate more useful features as the state representation (section 5.2) to improve the performance of the teacher model. Second, we do not consider an online augmentation policy currently, and we plan to take it into account. Third, we will extend our generic learning paradigm to a wide range of LTR scenarios, such as session-based recommendations [32], which has attracted increasing attention in recent years.

8 ACKNOWLEDGMENTS

This work was partially supported by the Open Research Projects of Zhejiang Lab (NO. 2021KB0AB02), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

REFERENCES

[1] Arvind Agarwal, Hema Raghavan, Karthik Subbian, Prem Melville, Richard D Lawrence, David C Gondek, and James Fan. 2012. Learning to rank for robust question answering. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. 833–842.

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

[3] Qingyao Ai, Xuanhui Wang, Sebastian Bruch, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2019. Learning groupwise multivariate scoring functions using deep neural networks. In *Proceedings of the 42nd ACM SIGIR International Conference on Theory of Information Retrieval*. 85–92.

[4] Rukshan Batuwita and Vasile Palade. 2010. Efficient resampling methods for training support vector machines with imbalanced datasets. In *Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[5] Irwan Bello, Sayali Kulkarni, Sagar Jain, Craig Boutilier, Ed Chi, Elad Eban, Xiyang Luo, Alan Mackey, and Ofer Meshi. 2018. Seq2slate: Re-ranking and slate optimization with rnns. *arXiv preprint arXiv:1810.02019* (2018).

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*. 41–48.

[7] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine learning*. 89–96.

[8] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[9] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. 2006. Adapting ranking SVM to document retrieval. In *Proceedings of the 29th International ACM SIGIR Conference on Research & Development in Information Retrieval*. 186–193.

[10] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. 2009. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM International Conference on Information and Knowledge Management*. 621–630.

[11] David Cossock and Tong Zhang. 2008. Statistical analysis of Bayes optimal subset ranking. *IEEE Transactions on Information Theory* 54, 11 (2008), 5140–5154.

[12] Thomas G Dietterich. 2000. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*. Springer, 1–15.

[13] Nat Dilokthanakul, Pedro AM Mediano, Marta Garnelo, Matthew CH Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. 2016. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648* (2016).

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*. 2672–2680.

[15] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[16] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 217–226.

[17] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[18] Ping Li, Qiang Wu, and Christopher Burges. 2007. McRank: Learning to rank using multiple classification and gradient boosting. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*. 897–904.

[19] Yang Li, Yadan Luo, Zheng Zhang, Shazia Sadiq, and Peng Cui. 2019. Context-aware attention-based data augmentation for poi recommendation. In *Proceedings of the 35th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 177–184.

[20] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. 2015. Recommender system application developments: a survey. *Decision Support Systems* 74 (2015), 12–32.

[21] 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.

[22] Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, Junfeng Ge, Wenwu Ou, et al. 2019. Personalized re-ranking for recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 3–11.

[23] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 datasets. *arXiv preprint arXiv:1306.2597* (2013).

[24] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *Proceedings of the 32nd IEEE Conference on Computer Vision and Pattern Recognition*. 7008–7024.

[25] J Ben Schafer, Joseph A Konstan, and John Riedl. 2001. E-commerce recommendation applications. *Data Mining and Knowledge Discovery* 5, 1 (2001), 115–153.

[26] Mark D Smucker, James Allan, and Ben Carterette. 2007. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the 16th ACM International Conference on Information and Knowledge Management*. 623–632.

[27] Peter Sunehag, Richard Evans, Gabriel Dulac-Arnold, Yori Zwols, Daniel Visentin, and Ben Coppin. 2015. Deep reinforcement learning with attention for slate markov decision processes with high-dimensional states and actions. *arXiv preprint arXiv:1512.01124* (2015).

[28] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miroslav Dudík, John Langford, Damien Jose, and Imed Zitouni. 2017. Off-Policy Evaluation for Slate Recommendation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 3635–3645.

[29] Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Brian MacWhinney, and Chris Dyer. 2016. Learning the curriculum with bayesian optimization for task-specific word representation learning. *arXiv preprint arXiv:1605.03852* (2016).

[30] Suzan Verberne, Hans van Halteren, Daphne Theijssen, Stephan Raaijmakers, and Lou Boves. 2011. Learning to rank for why-question answering. *Information Retrieval* 14, 2 (2011), 107–132.

[31] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Proceedings of the 29th International Conference on Neural Information Processing Systems*. 2692–2700.

[32] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z Sheng, Mehmet Orgun, and Defu Lian. 2019. A survey on session-based recommender systems. *arXiv preprint arXiv:1902.04864* (2019).

[33] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[34] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th International Conference on Machine learning*. 1192–1199.

[35] Qian Yu and Wai Lam. 2019. Data augmentation based on adversarial autoencoder handling imbalance for learning to rank. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 411–418.

[36] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 95–103.

[37] Tao Zhuang, Wenwu Ou, and Zhirong Wang. 2018. Globally Optimized Mutual Influence Aware Ranking in E-Commerce Search. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 3725–3731.