AUTOREC: AN AUTOMATED RECOMMENDER SYSTEM

A Thesis

by

TING-HSIANG WANG

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Xia "Ben" Hu
Committee Members,	Theodora Chaspari
	Na Zou
Head of Department,	Scott Schaefer

August 2022

Major Subject: Computer Science

ABSTRACT

Recommender systems are highly specialized to handle specific data and tasks. For example, Neural Collaborative Filtering [1] takes the implicit interaction between user and item IDs as the input data for the rating prediction task. Wide & Deep learning [2] ingests user and application attributes to predict app downloads for Google Play. And DeepFM [3] leverages both numerical and categorical data to estimate the click-through rate (CTR) for ad campaigns. However, a high degree of specialization comes at the expense of model adaptability and model tuning complexity. As shown in Figure 1, the originally apt model often either becomes obsolete or requires hyper-parameter tuning as the recommendation task at hand changes and additional types and amounts of data are collected over time. The efforts required to re-tune or re-build a model is often high.

So far, several modular pipelines for building recommender systems, such as OpenRec [4] and SMORe [5], have been proposed to address the adaptability issue. They contribute to the community by defining unified pipeline schema which divide recommendation models into a series of components (blocks) with specific functions and provide selectable modules for each. This design allows developers to quickly build and iterate recommendation models by assembling and swapping for the promising parts. Nevertheless, 1) determining which modules to use for each block and 2) hyper-parameter tuning for recommendation models remain challenging when models need to be adapted for continuously changing tasks and data.

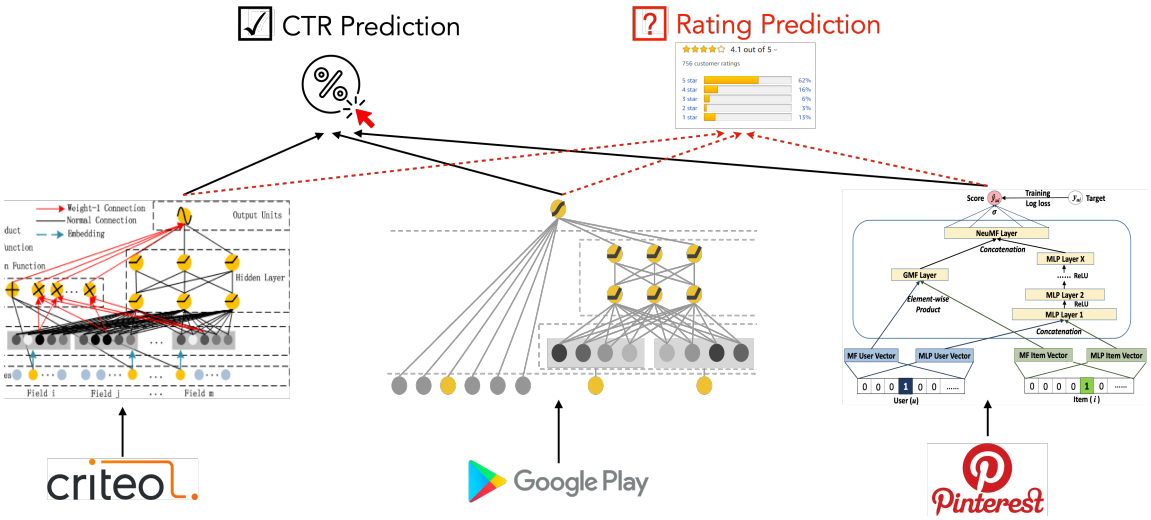

Figure 1: Recommender systems are optimized for specific data and tasks. For example, *NeuMF* [1] (right) predicts ratings based on user and item interactions. *Wide & Deep Learning* [2] (center) predicts app downloads based on user and application attributes. *DeepFM* [3] (left) predicts CTR for ads based on numerical and categorical data. However, as the data and tasks change over time, the models become obsolete. Despite better architectures loom in the background, designing and testing new models are no easy tasks.

ACKNOWLEDGMENTS

The past three years at Texas A&M University is an unforgettable chapter in my life. I have received a great deal of support and assistance. Here, I would like to thank all the people who have helped me in my research.

First and foremost, I would like to thank my advisor Dr. Xia "Ben" Hu, for his help, encouragement, and guidance.

Second, I would like to thank my committee members, Professor Na Zou and Professor Theodora Chaspari for their guidance and support throughout the course. This includes during projects, class, and even interviews.

Also, I would like to thank my group members and collaborators at the DATA (Data Analytics at Texas A&M) Lab in the Department of Computer Science and Engineering, for helping me and providing advice to my work.

I owe my most sincere gratitude to my parents for their support and love.

Finally, I would like to thank the Texas A&M University for supporting my research during my master's program.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by the thesis committee consisting of Professor Xia "Ben" Hu (advisor) and Professor Theodora Chaspari from the Department of Computer Science and Engineering, and Professor Na Zou from the Department of Engineering Technology and Industrial Distribution.

The conceptualization, design, implementation, and experimentation in this work is a collaborated work between Ting-Hsiang Wang (author), Qingquan Song, Haifeng Jin, Xiaotian Han, and Zirui Liu.

Funding Sources

The research work was supported under the Student Worker title from Texas A&M University.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Background and Motivation

As the main driving force responsible for generating venues in today's eCommerce companies, e.g., more than 1 billion US dollars per year for Netflix [6, 7], about 35% of sales for Amazon, and 60% of the clickable contents on YouTube [8], recommender systems across the world face a fundamental challenge: the engineering cost needed to adapt to new recommendation scenarios. Specifically, realistic recommender systems are required to have the capacity to quickly adapt to the constantly evolving data and tasks or to explore different models systematically. One of the most prominent examples of this is Netflix has never deployed the champion recommendation model emerged from their $1M contest due to its engineering cost and the business' shifting from movie recommendation, which targets rating prediction, to video streaming, which targets click-through-rate (CTR) prediction [9]. In addition, although recommender systems start to capitalize on the power of deep learning, they have yet been able to convert model depth into raw performance because of the tendency to overfit, which leads to severe online-offline performance mismatch. Therefore, most of the active recommendation models in the industry are shallow when compared to their computer vision counterparts [10]. This calls for a new approach of recommendation model development which emphasizes both flexibility and the systematic exploration of both existing and new neural architectures alike.

In the industry, most recommender systems are highly specialized to handle specific data and tasks. For example, NeuMF [1] takes user-item implicit feedback data as inputs for the rating prediction task; DeepFM [3] leverages both numerical and categorical data for the CTR prediction task. However, a high degree of specialization comes at the expense of model adaptability and tuning complexity. As recommendation tasks evolve over

time and additional types of data are collected, the originally apt model can either become obsolete or require tremendous tuning efforts. So far, several pipelines for recommender systems, e.g., OpenRec [4] and SMORe [5], tried to address the adaptability issue via providing modular base blocks that can be selected according to the context of recommendation. Nevertheless, both determining the blocks to use and tuning the model parameters are not straightforward when facing new data and changing tasks.

In order to bridge the gap, we present AutoRec, an open-source automated machine learning (AutoML) platform extended from the TensorFlow [11] ecosystem which focuses on neural architecture search (NAS) for deep recommendation models. While many AutoML libraries, such as Auto-Sklearn [12] and TPOT [13] have shown promising results in general-purpose machine learning tasks (e.g., regression and hyperparameter tuning) and our fruitful efforts with AutoKeras [14] extended AutoML to multi-modal data and multi-task training (e.g., text and image classification), few models incorporate AutoML for recommendation tasks. And for the few which do, their approaches are often too narrow for general recommendation models. For example, AutoInt [15] and AutoCTR [16] focus on searching interactions for only CTR prediction tasks. In contrast, AutoRec supports a highly flexible pipeline that accommodates both sparse and dense inputs, rating prediction and click-through rate (CTR) prediction tasks, and an array of recommendation models. Lastly, AutoRec provides a simple, user-friendly API. Our preliminary experiments conducted on the benchmark datasets reveal AutoRec can automate the finding of new state-of-the-art deep recommendation models without prior knowledge.

1.2 Thesis Contributions

As realistic recommender systems need to adapt to ever-changing data and tasks, this research aims to contribute to the community by automating model selection and hyperparameter tuning for recommendation models. There are two major components in this

thesis: 1) Design and implement a recommendation model architecture schema which supports searching optimal model interactions and hyper-parameters and 2) incorporate the searcher functionality to enable an efficient and systematic search by leveraging the power of AutoML. The major contributions of our work are summarized as follows:

- We formally define the modular architecture for recommendation models to enable NAS algorithms to conduct model search and hyperparameter tuning for the said models.
- The architecture surpasses existing frameworks in usability by accommodating sparse and dense inputs, rating prediction and CTR prediction tasks, and both deep and non-deep recommendation models.
- Experimental results show AutoRec is reliable and can identify models which resemble the best human-engineered models, showing the potential of AutoML in maintaining the reliability and efficiency of recommendation models in production.

In addition, the proposed architecture provides a user-friendly application programming interface (API), which allows users to define a searchable recommendation model using as few as 10 lines of code.

2. RELATED WORK

In this section, we discuss two lines of work that are relevant to our research: 1) different types of recommender systems and their evolution and 2) neural architecture search (NAS) introduced to automate the design of general neural network models. Together, these works contextualize the contributions of AutoRec.

2.1 Recommender Systems

Different from recommender systems in academia, where researchers design and optimize monolithic machine learning models based on fixed benchmark datasets, recommender systems in the industry need to handle data on the order of billions. Since it is computationally infeasible for models to rank the entire inventory of items according to user preference, the process of recommendation in the industry are divided into two stages: 1) the recall stage and 2) the ranking stage, as shown in Figure 2.1. First, in the recall stage, efficient heuristics (e.g., same category, k-nearest neighbors algorithm [17, 18]) retrieve promising items to drastically shrink the pool of candidate items. Then, in the ranking stage, models personalize items according to user preference to achieve accuracy. This research is focused on the machine learning models being used in the ranking stage.

2.1.1 Rating Prediction and CTR Prediction

The goal of recommender systems is to learn the relationship between users and items to help with curating a subset of items that solicits targeted behaviors from users (i.e., not limited to buying). In general, the user-item interaction data are classified as being either "explicit", where the data labels have numerical meaning (e.g., star ratings on Amazon, Tomatometer score on Rotten Tomatoes), or "implicit", where the data labels are binary (e.g., clicked or not for online ads, bought or not on Ebay). Consequently, recommendation

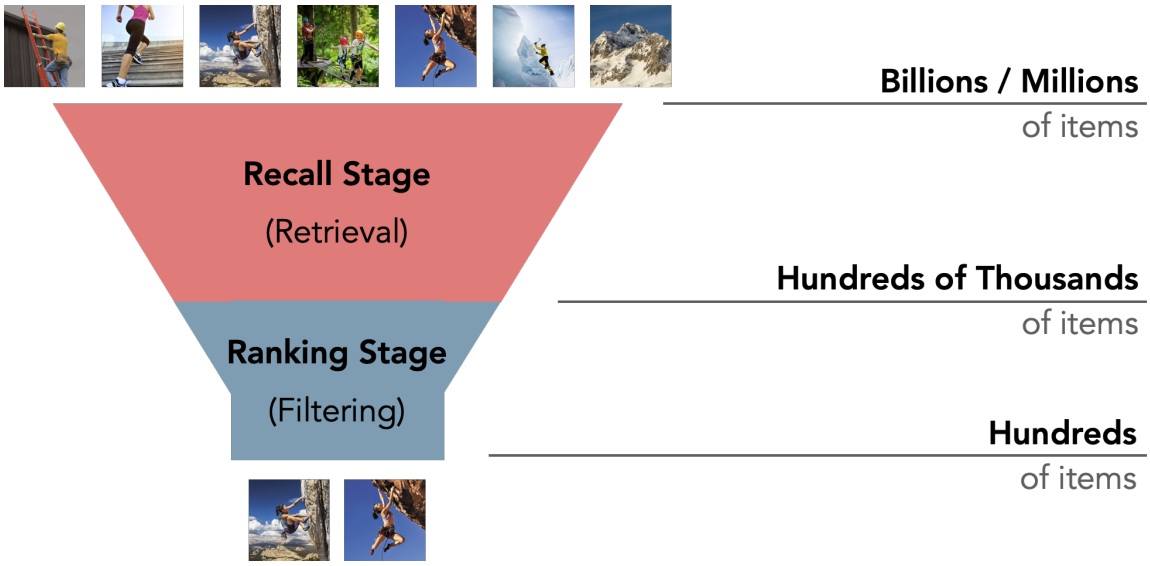

Figure 2.1: In the recall stage, the number of candidate items are greatly reduced using heuristics to ensure efficiency in the ranking stage. In the ranking stage, items are ranked using high-order features and complex models to achieve effectiveness.

tasks are also divided into two categories: "rating prediction" and "click-through rate (CTR) prediction", where the words "rating" and "CTR" are general terms describing the numerical and binary properties of the predictive outcomes, respectively.

The models which make recommendations solely based on user-item interaction data are said to have taken the "collaborative filtering" approach since they effectively recommend items based on whether other users with tastes similar to that of the target user like the candidate items (i.e., collaboration) and discard the rest (i.e., filtering). For example, collaborative filtering algorithms such as [19, 20, 21, 22] conduct rating prediction, while [1, 23, 24, 25] conduct CTR prediction.

Despite early recommender system research focused on the rating prediction task, the CTR prediction task has gained a lot of popularity in recent years. This is because, when compared to explicit data, implicit data is much easier to collect since users' explicit actions (e.g., submitting a score) are not required [26]. On the other hand, compared to rating

prediction, CTR prediction is considered to be more practical since the ultimate goal of recommender systems is maximizing the probability for targeted user behaviors to happen, not estimating the extent of user preference [1].

2.1.2 Deep Learning in Recommender Systems

The collaborative filtering approach was proposed at a time when: 1) leveraging the user-item interaction data alone is sufficient to make high quality recommendations [4] and 2) the data was considered abundant for its contemporary applications (e.g., product recommendation on Amazon [27]). However, as the industry evolved with time, two problems emerged: 1) more modes of data (e.g., text, image) became available but were not utilized and 2) data sparsity (the growth of the number of user-item interactions cannot keep up with those of the users and items) took increasing toll on the performance of collaborative filtering methods [28].

Deep learning is sought after by recommender systems mainly for two reasons: 1) its raw modeling power has brought tremendous advancements in other machine learning research areas (e.g., computer vision, natural language processing); and 2) its ability to convert item content information (e.g., product descriptions, product images) into latent embeddings, which can then be fed into recommender systems as auxiliary information to help alleviating the user-item interaction data sparsity problem.

Incidentally, most of the current deep recommender systems for rating prediction, while exploiting the superior modeling power of deep learning, do not utilize the auxiliary information embedded in item contents. The deep neural network structures being used for rating prediction include restricted Boltzmann machines (RBM) [21], autoencoder [29], etc. In contrast, deep recommender systems for CTR prediction take full advantage of different modes of data to facilitate recommendation. For example, textual descriptions are modeled by autoencoder [30], audio signals by multi-layer perceptron

(MLP) and convolutional neural network (CNN) [31], images by convolutional autoencoder [32], and metadata by attention network [15] and cross network [33]. Notice linear mondeling is still essential to recommender systems since most of the these methods still integrate latent features into matrix factorization.

2.1.3 Modular Framework for Recommender Systems

Although standard machine learning frameworks (e.g., TensorFlow [11], PyTorch [34]) have provided modular components for general machine learning models, the modular components tailored for recommender systems are still under-developed. Currently, existing works towards this problem usually provide modular base blocks that can be selected according to the context of recommendation. For example, OpenRec [4] divides the algorithm component of recommender systems into three stages: extraction, fusion, and interaction, where the extraction stage can utilize embeddings pretrained from different modes of content information (e.g., text, image). SMORe [5] approaches the recommendation problem from a graph perspective and divides a model into three blocks: sampler, mapper, and optimizer. The sampler module is responsible for extracting high-order interactions between entities in heterogeneous graphs (e.g., user, items, metadata) using various sampling heuristics (e.g., adjacency, neighborhood, random walk). Nevertheless, both determining the blocks to use and tuning the model hyperparameter are not straightforward when facing new data and changing tasks.

2.2 Neural Architecture Search

Neural architecture search (NAS) is a sub-field of automated machine learning (AutoML) [35] and has become the center of focus for machine learning research as deep learning models sophisticate and proliferate across different industries over the years. Essentially, NAS aims to automatically discover the optimal deep learning solutions given specific data and tasks, thus allowing domain experts to contribute without being limited

by their lack of experience in machine learning.

2.2.1 Search Space and Search Algorithms

NAS methods are formulated in terms of three components: 1) search space, 2) search strategy, and 3) performance estimation strategy [36]. Intuitively, the search space defines the candidate "architectures" for search strategies to pick from. For example, in the case of hyperparameter tuning for MLP, the search space is composed of the number of layers and the number of neurons in each layer. In the case of model search, the search space can include any permitted modeling operations (e.g., MLP, dot product, factorization machine (FM) [16]) supported by the machine learning framework being searched.

After a search space is defined, a search strategy is responsible for selecting the most promising architecture from the search space. The process is often a trade-off between exploitation (i.e., efficiency) and exploration (i.e., effectiveness) [36]. Some of the mainstream search strategies include Bayesian optimization, reinforcement learning, and evolutionary algorithm [35].

At the end of the NAS iteration, a predefined performance estimation strategy is used to evaluate the performance of the selected architecture. Since the intuitive solution of comparing training performance against validation performance is computationally infeasible, many other strategies are proposed (e.g., lower fidelity, learning curve exploration, network morphism, and weight sharing [36]). We remark that AutoKeras [14], the basis of our work, employs network morphism to address the efficiency issue of NAS.

2.2.2 Libraries for Automated Machine Learning

The notion of automating the process of finding the optimal model for particular data and tasks is not new to the community. In fact, most of the AutoML libraries are designed for non-deep-learning algorithms (e.g., TPOT [13], SMAC [37], Auto-WEKA [38], Auto-Sklearn [12]). AutoKeras [14] is one of the first open-source AutoML frameworks extend-

ing to deep neural networks (DNNs). However, these frameworks are designed for general machine learning and not for recommender systems. On the other hand, for the few which support NAS for recommender systems (e.g., AutoInt [15]. AutoCTR [16]), they search the optimal models only for CTR prediction task. Hence, one major novelty of this work is its modular and searchable pipeline tailored for both rating prediction and CTR prediction tasks for recommender systems.

To sum up, our work aims to fill the gap of an AutoML framework that satisfies the needs of general recommender systems, where the system support both rating prediction and CTR prediction tasks and can utilize user-item interaction data as well as item feature information to find the most optimal model. The best of all is that the entire process can be automated by search algorithms once input data, recommendation task, and architecture search space are provided.

3. PROBLEM STATEMENT

In this section, we introduce the necessary notations as well as the problem of neural architecture search (NAS) for recommender systems. Essentially, NAS aims to discover the optimal machine learning solution for a recommendation scenario by automating model selection and hyperparameter tuning with search algorithms. We give the prerequisites for formulating the problem below and list the notations in Table 3.1 for reference.

There are two specifications pertaining to recommender systems which are required to formulate the problem: dataset and loss function. Given a dataset $\mathcal{D}$, $\mathcal{D}$ is split into a training set $\mathcal{D}_{train}$ and a validation set $\mathcal{D}_{valid}$. A training loss function $\mathcal{L}_{train}$ and a validation loss function $\mathcal{L}_{valid}$ should be provided to facilitate model training and validation, respectively. Notice $\mathcal{L}_{train}$ and $\mathcal{L}_{valid}$ are often different for recommender systems for pragmatic reason. Namely, $\mathcal{L}_{train}$ is required to be differentiable for model training (e.g., MSE, cross entropy), while $\mathcal{L}_{valid}$ is chosen to resemble real-life recommendation scenario as close as possible (e.g., AUC, Precision@k).

To automate model selection and hyperparameter tuning using NAS techniques, there are two specifications required to formulate the problem: model search space and hyperparameter search space. A model search space $\mathcal{M} = \{m^1, m^2, ...\}$ is a collection of trainable candidate models m^i. And each m^i is associated with a hyperparameter search space $\mathcal{H}^i$, where $\mathcal{H}^i = \{h^{i,1}, h^{i,2}, ...\}$ is a collection of candidate hyperparameters $h^{i,j}$. Finally, the problem of NAS for recommender systems is formulated as the optimization problem shown below:

$$m^*_{h^*} = \underset{m^i \in \mathcal{M}, h^{i,j} \in \mathcal{H}^i}{\operatorname{argmin}} \mathcal{L}_{valid}(m^i_{h^{i,j},w^*}, \mathcal{D}_{valid}),$$

$$\text{s.t. } w^* = \underset{w}{\operatorname{argmin}}\, \mathcal{L}_{train}(m^i_{h^{i,j},w}, \mathcal{D}_{train}),$$

Table 3.1: Important symbols and definitions.

Notations	Definitions
$\mathcal{D}$	Dataset
$\mathcal{D}_{train}$	Training set
$\mathcal{D}_{valid}$	Validation set
$\mathcal{L}_{train}$	Training loss function
$\mathcal{L}_{valid}$	Validation loss function
$\mathcal{M}$	Model search space
m^i	Model instance
$\mathcal{H}^i$	Hyperparameter search space associated with m^i
$h^{i,j}$	Hyperparameter instance associated with m^i
w	Model weight
w^*	Optimal model weight
m^*	Optimal model instance
h^*	Optimal hyperparameter for m^*
$m^*_{h^*}$	Optimal machine learning solution m^* trained with h^*

where w^* is the optimal state of (i.e., trained) weight w for a model m. The objective of NAS for recommender systems is, therefore, to find the optimal model m^* trained with its optimal hyperparameter h^*, namely the optimal machine learning solution $m^*_{h^*}$.

NAS for Recommender Systems: Given training set $\mathcal{D}_{train}$, validation set $\mathcal{D}_{valid}$, training loss function $\mathcal{L}_{train}$, validation loss function $\mathcal{L}_{valid}$, model search space $\mathcal{M}$, and hyperparameter search space $\mathcal{H}$, the objective of NAS for recommender systems is to find the optimal model m^* trained with its optimal hyperparameter h^*, namely the optimal machine learning solution $m^*_{h^*}$.

4. THE PROPOSED FRAMEWORK: AUTOREC

In this section, we introduce the AutoRec framework. The AutoRec framework operates based on two major components: 1) **Recommender:** the search space and evaluation metrics module which provides a model search space $\mathcal{M}$ and its associated hyperparameter search space $\mathcal{H}$. And 2) **Seacher:** the search functionality module (e.g., random search, Bayesian optimization) which iteratively decides whether to exploit existing candidate model m^i and its associated hyperparameter $h^{i,j}$ or to explore new ones. We remark that the majority of our work is focused on the design and implementation of the Recommender, while the Searcher in our framework is extended from the TensorFlow [11] and AutoKeras [14] ecosystems.

4.1 Overview of AutoRec Pipeline

Figure 4.1 details the process of using the proposed AutoRec pipeline to set up a Recommender instance for Searcher to perform NAS. There are four steps involved in the process: **data preprocessing**, **search space construction**, **search algorithm selection**, and **NAS**. Below, we will discuss about each step in more detail.

In the data preprocessing step, a Preprocessor is used to transform numerical features (e.g., log transformation) and categorical features (e.g., fit dictionary). As data preprocessing is highly circumstantial, abstract interface is defined for users to implement their own solutions. However, we do provide ready-to-use Preprocessors for mainstream benchmark datasets such as Netflix, Movielens 1M, Avazu, and Criteo. At the end of the data preprocessing step, the input dataset $\mathcal{D}$ is split into a training set $\mathcal{D}_{train}$, a validation set $\mathcal{D}_{valid}$, and a test set $\mathcal{D}_{test}$.

In the search space construction step, Recommender is built from chaining Mapper, Interactor, and Optimizer. Recommender is literally an unspecified recommendation model

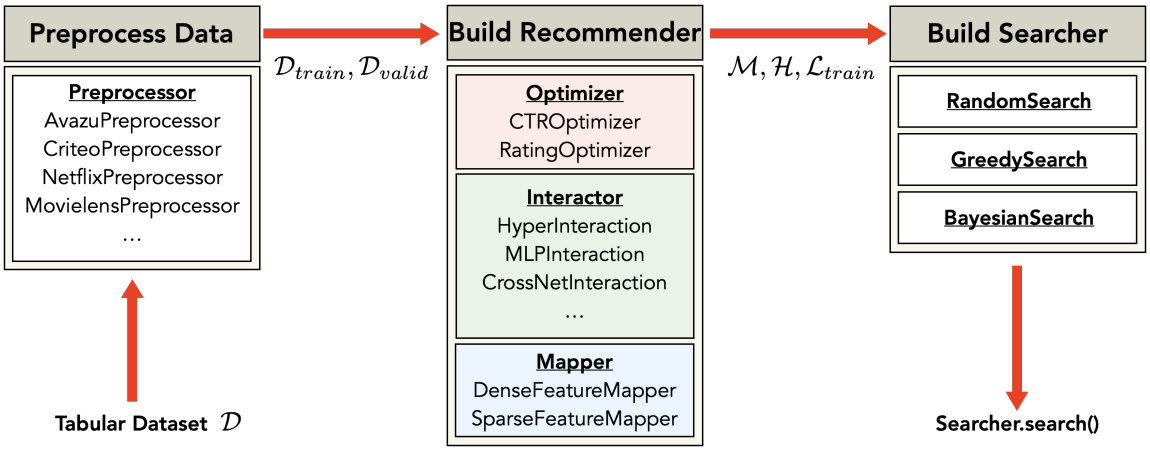

Figure 4.1: Pipeline of the proposed AutoRec framework. First, Preprocessor prepares and splits $\mathcal{D}$ into $\mathcal{D}_{train}$ and $\mathcal{D}_{valid}$. Second, Recommender is built from chaining Mapper, Interactor, and Optimizer, where Interactor specifies $\mathcal{M}$ and $\mathcal{H}$ and Optimizer specifies $\mathcal{L}_{train}$. Third, Searcher is instantiated with specific search algorithms and chained with Recommender. This forms an AutoRec instance ready to undergo NAS.

waiting to differentiate. Among its three components, Mapper and Optimizer are fixed, while Interactor is mutable. Mapper is responsible for converting data into input features (e.g., embeddings). Optimizer defines the training loss function $\mathcal{L}_{train}$ depending on the recommendation task (i.e., MSE for rating prediction, cross entropy for CTR prediction). And Interactor provides model search space $\mathcal{M}$ and its associated hyperparameter search space $\mathcal{H}$ for the purpose of NAS.

In the search algorithm selection step, Searcher is instantiated with the specified search algorithm (e.g., random search, greedy search, Bayesian optimization) and is chained with Recommender. This forms an AutoRec instance ready to undergo NAS.

Finally, in the NAS step, the AutoRec instance initiates NAS and uses the provided training loss function $\mathcal{L}_{valid}$ to help with finding $m^*_{h^*}$. In our research, we let the loss function for $\mathcal{L}_{train}$ and $\mathcal{L}_{valid}$ to be the same due to the difficulty in obtaining more realistic labels in an academic setting, despite they are often different in the industry. The process is iterative and, depending on whether search space is provided by Interactor, may automate

both, either, or none of model selection and hyperparameter tuning.

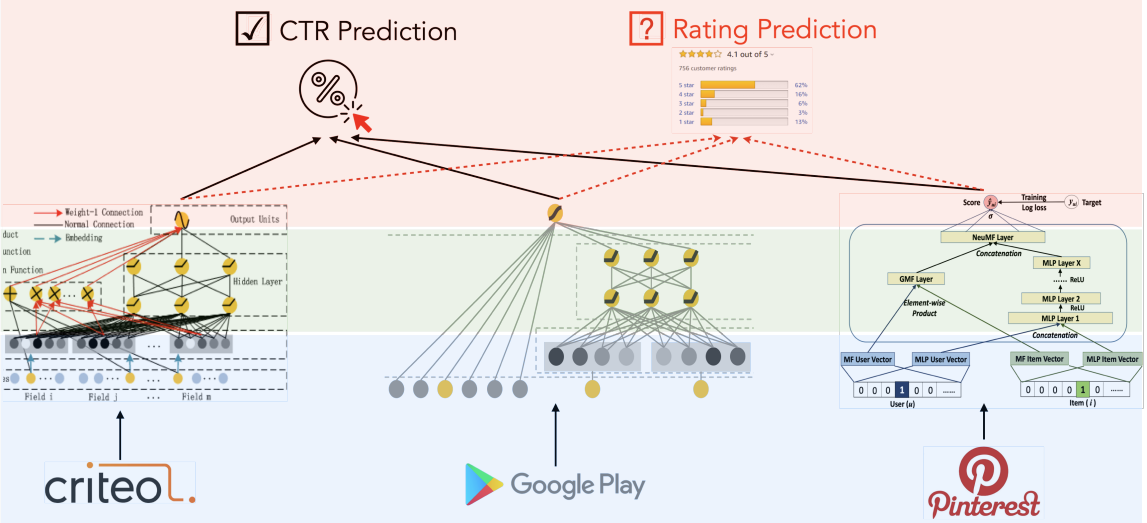

Figure 4.2: The three-block schema in in recommender systems. (Bottom) Data inputs are mapped into feature vectors. (Center) Feature vectors interact to produce additional information. (Top) Output value is scaled and optimized for the specific recommendation task. (From left to right) *DeepFM* [3], *Wide & Deep Learning* [2], and *NeuMF* [1].

4.2 Design of Recommender Module

As Figure 4.2 shown, the structure of recommender systems can be generalized using a three-block schema. When data comes in, it is mapped into different feature vectors depending on its data type (e.g., dense vector for numerical feature, one-hot encoding for categorical feature). The feature vectors then interact with each other following intricate neural networks to produce new information. Finally, the output value is scaled and optimized for the specific recommendation recommendation task to help with model training. This motivates us to design Recommender with three dedicated blocks: **Mapper**, **Interactor**, and **Optimizer**, to handle data mapping, feature interaction, and output optimization. Figure 4.3 shows the division of labor between the triad in greater detail.

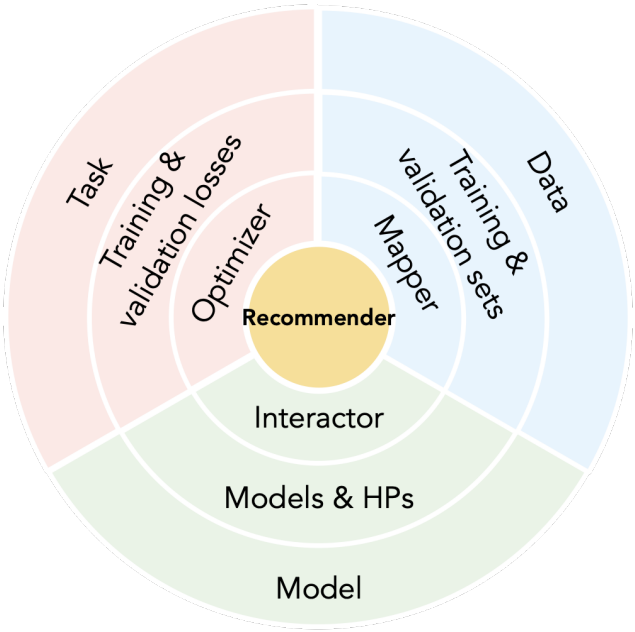

Figure 4.3: Recommender is consisted of Interactor, Mapper, and Optimizer, which handles the data variables (training and validation sets), model variables (models and hyperparameters), and task variables (training and validation losses), respectively.

4.2.1 Mapper Block

The Mapper block is designed to convert data into feature vectors so that different entities can interact with each other to produce additional information. Numerical (dense) features are naturally comparable and thus directly usable, while categorical (sparse) features needs to be fit and transformed by Preprocessor. Both numerical and categorical features are mapped into embeddings to reduce data noise and dimensionality before being passed to Interactor to produce new information.

4.2.2 Interactor Block

The Interactor block defines the model search space $\mathcal{M}$ and its associated hyperparameter search space $\mathcal{H}$ for NAS. Through Interactors, the input features can interact to produce additional information, which help with modeling. The Interactor is designed

under three guidelines to ensure a comprehensive and flexible search space: 1) Linear, nonlinear, and deep learning based architectures should all be included. 2) Mainstream recommendation models must be present in the search space. And 3) Interactor should have the flexibility to remain ambiguous until Searcher decides which specific Interactor it should become during NAS.

Among the nine supported Interactor classes, five are non-deep: RandomSelectInteraction, ConcatenateInteraction, ElementwiseInteraction, InnerProductInteraction, and FMInteraction. Three are deep: MLPInteraction, CrossNetInteraction, and SelfAttentionInteraction. Lastly, HyperInteraction is the meta Interactor which can parallel multiple Interactors in the same level. With these modules, state-of-the-art recommender systems such as NeuMF [1], DeepFM [3], DLRM [39], AutoInt [15], DCN [33], MF [19], and MLP are able to be discovered by NAS. We remark that RandomSelectInteraction, ConcatenateInteraction, and InnerProductInteraction are not searchable. For Interactors which are searchable, we list their search spaces below:

- **ElementwiseInteraction**:

$$\mathcal{H}^{Rand} = \{h^{sum}, h^{average}, h^{multiply}, h^{max}, h^{min}\}$$

- **MLPInteraction**:

$$\mathcal{H}^{MLP} = \{h^{units}, h^{num\_layers}, h^{batch\_norm}, h^{drop\_rate}\}$$

- **FMInteraction**:

$$\mathcal{H}^{FM} = \{h^{embedding\_dim}\}$$

- **CrossNetInteraction**:

$$\mathcal{H}^{CrossNet} = \{h^{layer\_num}\}$$

- **SelfAttentionInteraction**:

$$\mathcal{H}^{SelfAtt} = \{h^{embedding\_dim}, h^{att\_embedding\_dim}, h^{head\_num}, h^{residual}\}$$

- **HyperInteraction**:

$$\begin{aligned}\mathcal{M} = \{&m^{MLPInteraction}, m^{ConcatenateInteraction}, m^{RandomSelectInteraction},\\ &m^{ElementwiseInteraction}, m^{FMInteraction}, m^{CrossNetInteraction},\\ &m^{SelfAttentionInteraction}, m^{InnerProductInteraction}\}\\ \mathcal{H}^{Hyper} = \{&h^{interactor\_type}, h^{meta\_interactor\_num}\}\end{aligned}$$

4.2.3 Optimizer Block

The Optimizer block defines training loss function $\mathcal{L}_{train}$ for a Recommender instance for measuring the deviation between predicted values and labels. $\mathcal{L}_{train}$ for rating prediction task and CTR prediction task are mean squared error (MSE) and cross entropy (CE), respectively. The two loss functions are defined below:

- **Mean Squared Error (MSE)**: measures the average squared distance between the predicted values and the true values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(r_i - \hat{r}_i)^2,$$

where n is the number of observations, r_i is the true rating of the i^{th} observation, and $\hat{r}_i$ is the predicted rating of the i^{th} observation.

- **Cross Entropy (CE)**: measures the average difference between the predicted probability distribution and the true probability distribution.

$$CE = -\frac{1}{n}\left(\sum_{i=1}^{n} y_i \cdot log(\hat{p}_i)\right),$$

where n is the number of observations, y_i is the true label of the i^{th} observation, and $\hat{p}_i$ is the predicted probability of the i^{th} observation.

4.3 About the Searcher Module

As Figure 4.4 shown, the Searcher class provides the search strategies which conduct NAS. On the other hand, Recommender provides the model search space $\mathcal{M}$ and hyperparameter search space $\mathcal{H}$ to be searched with and the loss functions $\mathcal{L}_{train}$ and $\mathcal{L}_{valid}$ as performance estimator for guiding the search process. The Seacher class supports three types of search algorithms: random search, greedy search, and Bayesian optimization. As our work aims to study the impact of NAS for recommender system development and not search algorithms themselves, the Searcher class is directly extended from the Tensorflow [11] and AutoKeras [14] ecosystems. Without going into too much detail, we discuss the characteristics of the provided search algorithms below:

- **Random Search [40]:** The search algorithm explores a larger, less promising configuration space.
- **Greedy Search:** The search algorithm exploits the architecture that locally optimize the objective function at each iteration.
- **Bayesian Optimization [41]:** The search algorithm actively tries to balance the trade-off between exploration and exploitation by reasoning before running.

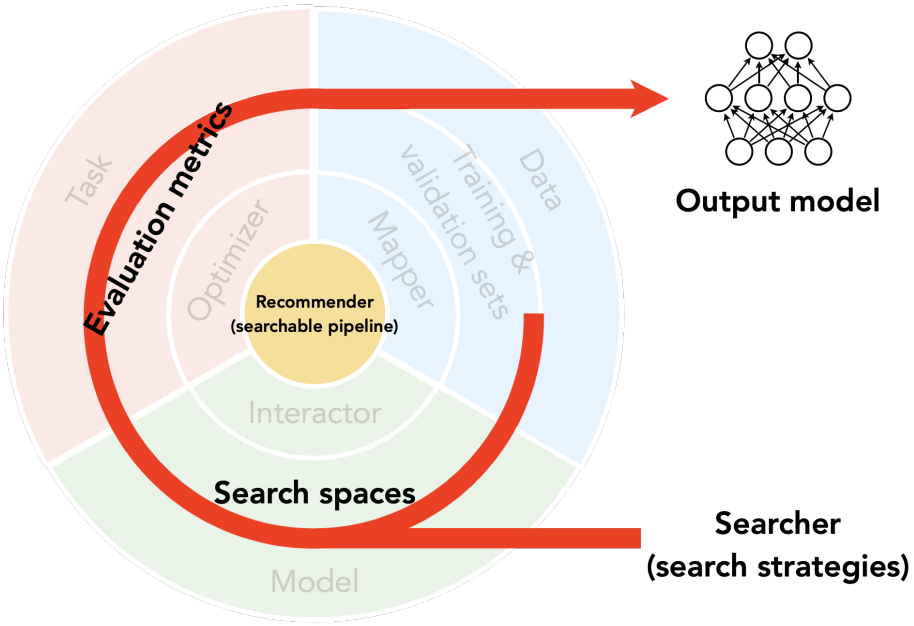

Figure 4.4: Searcher's search strategies conducts NAS and performance evaluation based on the search spaces and losses provided by Recommender. At the end of each search iteration, a winning model along with its optimal hyperparameters are decided.

To perform model search and hyperparameter tuning for a Recommender, simply pass the Recommender to a Searcher and call the search function to undergo NAS. The Searcher will display diagnostic information for developer's reference. Such information include loss, explored Interactors, and their hyperparameters.

5. EXPERIMENTS

We evaluate the performance of the proposed AutoRec framework base on four real-world benchmark datasets. Specifically, we aim to answer the following questions:

- **Q1.** How effective is the proposed AutoRec compared to state-of-the-art recommender systems in rating prediction task?
- **Q2.** How effective is the proposed AutoRec compared to state-of-the-art recommender systems in CTR prediction task?
- **Q3.** How much effort is required to build a recommender system with automated model selection and hyperparameter tuning using AutoRec?

5.1 Datasets

In the experiments, four real-world datasets are used to assess the performance of the proposed AutoRec framework. For the rating prediction task, two movie rating datasets are used, i.e., Netflix [20] and Movielens 1M [42]. And for the CTR prediction task, two advertisement click datasets are used, i.e., Avazu [43] and Criteo [44]. All four of these datasets are well-studied benchmark datasets in their respective task area.

5.1.1 Datasets for Rating Prediction

For the rating prediction task, we employ the full Netflix and Movielens 1M datasets. The dataset statistical information for rating prediction is reported in Table 5.1.

Table 5.1: Dataset statistics for rating prediction task.

	# Users	# Items	# User-item Interactions
Netflix	480,189	17,770	100,480,507
Movielens 1M	6,040	3,900	1,000,209

Table 5.2: Dataset statistics for CTR prediction task.

	# Dense Features	# Sparse Features	# Total Records	# Sampled Records
Avazu	0	22	40,428,967	500K
Criteo	13	26	45,840,617	500K

- **Netflix** [20]: The movie rating dataset contains four kinds of data: movie ID, customer ID, rating ranged from 1 to 5 stars, and the date of interaction. It was released by Netflix in 2006 as the benchmark dataset for the Netflix Prize on Kaggle.

- **Movielens 1M** [42]: The movie ratings file (i.e., "ratings.dat") is one of the four dataset files from the Movielens 1M dataset and contains four data columns: user ID, movie ID, rating ranged from 1 to 5 stars, and timestamp of the interaction. It was released in 2003 by GroupLens.

5.1.2 Datasets for CTR Prediction

For the CTR prediction task, due to the large number of feature columns and the computational overhead involved during the ranking stage of the recommendation process, we sampled 500K data records from each of Avazu and Criteo dataset. The dataset statistical information for rating prediction is reported in Table 5.2.

- **Avazu 500K** [43]: The mobile advertising dataset contains 24 data columns, where column 0 is ad ID, column 1 is label indicating whether the user clicked or not, and column 2-23 are categorical features. It was released by Avzu in 2014 as the benchmark dataset for the CTR Prediction Contest on Kaggle.

- **Criteo 500K** [44]: The online advertising dataset contains 40 data columns, where column 0 is label indicating whether the user clicked or not, column 1-13 are numerical features, and columns 14-39 are categorical features. It was released by Criteo in 2014 as the benchmark dataset for the CTR Prediction Contest on Kaggle.

5.2 Baseline Methods

In our experiments, two different types of recommendation tasks: 1) rating prediction and 2) CTR prediction, are conducted to evaluate the proposed AutoRec. For the rating prediction task, one baseline is linear and the other two are deep learning based: MF [19], MLP, and NeuMF [1]. And for the CTR prediction task, we employ four deep learning baselines: DeepFM [3], DLRM [39], DCN [33], and AutoInt [15]. Notice existing recommender systems, whether automated or not, are designed for only one type of recommendation task, while we aim to provide an AutoML solution for recommender systems that can generalize to both tasks.

5.2.1 Baselines for Rating Prediction

- **MF** [19]: takes the user-item interaction matrix as input and decompose it into two smaller matrices: a user latent matrix and a item latent matrix, where every user and item is associated with a latent vector. Taking the dot product of the two latent matrices recreates the user-item interaction matrix with the missing entries now substituted with predicted rating score.

- **MLP**: takes a single user-item interaction feature vector as input by concatenating a user one-hot encoding with a item one-hot encoding. The feature vector is then fed into an MLP to model the high-order interaction between user and item. The embedding of the last hidden layer is passed through a ReLU activation function to produce a predicted rating score.

- **NeuMF** [1]: is a two-tower model constructed by fusing the MF model and the MLP model. Specifically, the last hidden layer of MF and the last hidden layer of MLP are concatenated to form the last hidden layer of NeuMF. The embedding is then passed through a ReLu activation function to produce a predicted rating score.

5.2.2 Baselines for CTR Prediction

- **DeepFM** [3]: is a two-tower model which takes categorical features as inputs, where the wide component is a factorization machine (FM) modeling the pairwise feature interactions, and the deep component is an MLP modeling the high-order feature interactions. The output embeddings of the two components are added together and passed through a sigmoid function to produce the probability of a click.

- **DLRM** [39]: takes both numerical and categorical features as inputs, which are converted into feature vectors by an MLP and one-hot encoding, respectively. The two features are then passed to an MLP, whose output is passed through a sigmoid function to produce the probability of a click.

- **AutoInt** [15]: takes both numerical and categorical features as inputs. Whereas each numerical field is converted into its own feature vector by multiplying with a learned embedding vector, categorical fields are converted into a single feature vector by multiplying a learned embedding matrix with their one-hot encodings. The features are then concatenated and passed in to a multi-head self-attention unit, whose output is passed through a sigmoid function to produce the probability of a click.

- **DCN** [33]: takes both numerical and categorical feature as inputs. Whereas the numerical fields are treated as a feature vector, the categorical fields are converted into a single feature vector by multiplying a learned embedding matrix with their one-hot encodings. The two feature vectors are cocatenated and fed into a cross network and an MLP to provide feaure-crossing and high-order interaction information, respectively. Their output embeddings are concatenated again before being passed through a sigmoid function to produce the probability of a click.

5.3 Experimental Setup

In this section, we introduce the overall experimental setup of this work, including data sampling and preprocessing, evaluation metrics for the rating prediction task and the CTR prediction task, and parameter settings.

5.3.1 Data Sampling and Preprocessing

5.3.1.1 Data Sampling

As mentioned in Section 2.1, a recommendation model actually plays the pivotal role only in the ranking stage of the grand scheme of a recommender system. Without a preceding recall stage to reduce the amount of candidate items, ranking the entire inventory of items based on tens to hundreds of features for each user is simply computationally infeasible. As information retrieval is not the focus of this work, we take the liberty to sample data as needed.

For the rating prediction task, we use the entire **Netflix** [20] and **Movielens 1M** [42] datasets. Whereas, for the CTR prediction task, we sample 500K records from the Avazu [43] and the Criteo [44] datasets to make the **Avazu 500K** and the **Criteo 500K** datasets. Despite the 100M user-item interactions in the Netflix dataset (Table 5.1) seem to trump the 40M and 45M total records in the Avazu dataset and the Criteo dataset, respectively (Table 5.2), computing results for the latter are orders of magnitude more expensive due to the number of features involved. Notice the rating prediction datasets have only two features (i.e., user and item) while the CTR prediction datasets have tens of features (i.e., the total number of numerical and categorical features).

5.3.1.2 Data Preprocessing

To reduce data skewness in numerical data, we scale the data of each feature type by log transformation. To facilitate easy lookup and one-hot encoding for categorical data,

we create a fit dictionary to associate each feature type as well as each identifier in the feature type with a number (starting from zero). Categorical data is then converted into one-hot encodings before being ingested by models. To procure training, validation, and test datasets and avoid label imbalance, we employ stratified sampling to split the data according to the 8:1:1 training-validation-test ratio.

Notice dates in the Netflix dataset, timestamps in the Movielens 1M dataset, and ad IDs in the Avazu 500K dataset are not used in our research and therefore not preprocessed. We would also like to remark that the user and item features in the Netflix and Movielens 1M datasets are considered as categorical (i.e., sparse) features because user and item IDs are incomparable and discrete.

5.3.2 Evaluation Metrics

In our experiments, the performance of recommender systems is measured by two metrics: 1) mean squared error (MSE) for the rating prediction task and 2) cross entropy (CE) for the CTR prediction task. Since we are measuring loss, the smaller the value, the better the performance. The two metrics are defined below:

- **Mean Squared Error (MSE)**: measures the average squared distance between the predicted values and the true values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(r_i - \hat{r}_i)^2,$$

  where n is the number of observations, r_i is the true rating of the i^{th} observation, and $\hat{r}_i$ is the predicted rating of the i^{th} observation.

- **Cross Entropy (CE)**: measures the average difference between the predicted prob-

ability distribution and the true probability distribution.

$$CE = -\frac{1}{n}\left(\sum_{i=1}^{n} y_i \cdot log(\hat{p}_i)\right),$$

where n is the number of observations, y_i is the true label of the i^{th} observation, and $\hat{p}_i$ is the predicted probability of the i^{th} observation.

5.3.3 Parameter Settings

In our work, most of the model parameters (e.g., number of neurons, number of layers) are automatically discovered by search algorithms. Specifically, AutoRec searchers tune the hyperparameters for all models, with AutoRec-RP and AutoRec-CTR further subject to model search. This is because the architectures of baseline models are structured in advance, while the architectures of AutoRec-RP and AutoRec-CTR are defined by searchable virtual blocks.

In our experiments, the only unspecified model parameter is: input mapping dimension=64. On the other hand, the training parameters are: epoch=10, early stop=1, and trial=10. Due to its large size, the Netflix dataset has batch size set to 512,000 while the other datasets have batch size set to 1,024.

Table 5.3: Performance for the rating prediction task.

Mean Squared Error (MSE)						
Dataset	Netflix			Movielens 1M		
Searcher	Random	Greedy	Bayesian	Random	Greedy	Bayesian
MF	0.749	0.740	0.729	0.755	<u>0.750</u>	0.752
MLP	0.755	0.765	0.755	0.768	0.771	0.760
NeuMF	0.706	<u>0.644</u>	0.707	0.772	0.752	0.772
AutoRec-RP	**0.637**	0.740	0.645	<u>0.750</u>	0.751	**0.749**

5.4 Model Performance in Rating Prediction (Q1)

Table 5.3 lists the rating prediction results of the four compared methods (i.e., MF, MLP, NeuMF, and AutoRec-RP) trained on the Netflix and Movielens 1M datasets. First, we report that the numbers are on par with our experience working with these recommendation models. Overall, AutoRec-RP scored the best model-searcher combination (**boldface**) in both datasets. This demonstrates the effectiveness of AutoRec in discovering optimal machine learning solutions for the rating prediction task.

Notice that the deep learning based NeuMF and the linear MF are the runner-up (<u>underline</u>) models for the Netflix dataset and the Movielens 1M dataset, respectively. Their high performance is likely attributed to the one architecture they share in common, the generalized MF component. The phenomenon may not come as a surprise because MF is still the main staple for modeling user-item interaction information (i.e., CF). Often, deep learning is employed to extract auxiliary information from item contents. It is then integrated with MF to help with CF [1].

Table 5.4: Performance for the CTR prediction task.

Cross Entropy (CE)						
Dataset	Avazu 500K			Criteo 500K		
Searcher	Random	Greedy	Bayesian	Random	Greedy	Bayesian
DeepFM	0.406	0.401	0.403	0.481	0.480	0.472
DLRM	0.401	0.402	0.404	0.476	0.474	0.478
DCN	**0.399**	0.400	0.403	**0.471**	0.479	0.476
AutoInt	0.403	0.405	0.409	0.476	0.472	0.474
AutoRec-CTR	0.400	0.405	0.409	0.475	0.476	**0.471**

5.5 Model Performance in CTR Prediction (Q2)

Table 5.4 lists the CTR prediction results of the five compared methods (i.e., DeepFM, DLRM, DCN, AutoInt, and AutoRec-CTR) trained on the Avazu 500K and Criteo 500K datasets. In this experiment, the numbers are also on par with our experience working with these recommendation models. Here, we observe that DCN scored not only the best (**boldface**) but also the runner-up (underline) model-searcher combinations. Notice the Avazu 500K and Criteo 500K datasets both have large amounts of categorical features (see Table 5.2). And the cross network component of DCN is particularly suitable for modeling the high-order feature interactions through feature crossing.

On the other hand, AutoRec-CTR and DCN tied for first place in the Criteo 500K dataset, and AutoRec-CTR is a runner-up in the Avazu 500K dataset. This shows, while AutoRec is still effective in discovering optimal machine solutions for the CTR prediction task, AutoML does have limitation. While the DCN components are present in the AutoRec search space, it is computationally infeasible for search algorithms to iterate through all architectural combinations. Consequently, an exploitation-exploration tradeoff was made, resulting in a worse but close solution (less than 0.25% difference in CE).

5.6 Coding Simplicity Case Study (Q3)

The advantages of using the AutoRec framework for developing recommender systems include: 1) AutoRec is integrated with the TensorFlow ecosystem, which supports parallel computing and allows discovered models to be saved as Keras model instances for reuse. 2) AutoRec facilitates systematic exploration of candidate models and hyperparameters as the variables are managed internally by the system. 3) It is intuitive for users to build new models by chaining components following the computational graph style. 4) Search results are shared in terminal to provide feedback and further analysis, as shown in Figure 5.1. 5) Coding is minimized due to high-level of encapsulation, as shown in Figure 5.2.

```
[Trial summary]
 > Hp values:
 |-hyper_interaction_1/fm_interaction_2/embedding_dim: 8
 |-hyper_interaction_1/interactor_type_0: FMInteraction
 |-hyper_interaction_1/interactor_type_1: SelfAttentionInteraction
 |-hyper_interaction_1/self_attention_interaction_1/att_embedding_dim: 8
 |-hyper_interaction_1/self_attention_interaction_1/embedding_dim: 8
 |-hyper_interaction_1/self_attention_interaction_1/head_num: 2
 |-hyper_interaction_1/self_attention_interaction_1/residual: True
 |-hyper_interaction_2/cross_net_interaction_1/layer_num: 1
 |-hyper_interaction_2/fm_interaction_1/embedding_dim: 8
 |-hyper_interaction_2/interactor_type_0: CrossNetInteraction
 |-hyper_interaction_2/interactor_type_1: FMInteraction
 |-hyper_interaction_3/cross_net_interaction_2/layer_num: 1
 |-hyper_interaction_3/interactor_type_0: RandomSelectInteraction
 |-hyper_interaction_3/interactor_type_1: CrossNetInteraction
 |-optimizer: adam
 |-Score: 0.5980250239372253
 |-Best step: 0
```

Figure 5.1: Search results are displayed in the terminal for feedback and analysis.

Figure 5.2: Searchable model for rating prediction (AutoRec-RP) built in 10 lines of code.

6. CONCLUSION

Realistic recommender systems need to adapt to ever-changing recommendation scenarios as well as explore machine learning architectures systematically, and AutoML fits right into the picture. However, we surveyed current mainstream AutoML frameworks and discovered most of them are built for general-purpose machine learning tasks [14, 13, 37, 38, 12] or lack comprehensive support for recommender systems [16, 15]. This motivates us to build a comprehensive AutoML framework specifically for recommender systems. In this work, we present AutoRec, an open-source AutoML framework based on the TensorFlow ecosystem [11, 14] for recommendation tasks. AutoRec automates both model selection as well as hyperparameter tuning. Experiments conducted on the benchmark datasets [20, 43, 44, 42] reveal AutoRec is reliable and can identify close-to-the-best recommendation models without prior knowledge in machine learning. Lastly, AutoRec provides a simple, user-friendly API to lower the hurdles for people with limited experiences in coding to utilize the package. In the near future, we look forward to adding more model components to the search space and additional plug-and-play demo examples into AutoRec open-source repository.

REFERENCES

[1] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, pp. 173–182, 2017.

[2] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st workshop on deep learning for recommender systems*, pp. 7–10, 2016.

[3] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," *arXiv preprint arXiv:1703.04247*, 2017.

[4] L. Yang, E. Bagdasaryan, J. Gruenstein, C.-K. Hsieh, and D. Estrin, "Openrec: A modular framework for extensible and adaptable recommendation algorithms," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pp. 664–672, 2018.

[5] C.-M. Chen, T.-H. Wang, C.-J. Wang, and M.-F. Tsai, "Smore: modularize graph embedding for recommendation," in *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 582–583, 2019.

[6] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Transactions on Management Information Systems*, 2016.

[7] D. Jannach and M. Jugovac, "Measuring the business value of recommender systems," pp. 1–23, 2019.

[8] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, *et al.*, "The youtube video recommendation system," in *Proceedings of the fourth ACM conference on Recommender systems*, pp. 293–296, 2010.

[9] "Netflix recommendations: Beyond the 5 stars (part 1)," 2012. https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[11] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

[12] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 2962–2970, Curran Associates, Inc., 2015.

[13] R. S. Olson and J. H. Moore, "Tpot: A tree-based pipeline optimization tool for automating machine learning," in *Automated Machine Learning*, pp. 151–160, Springer, 2019.

[14] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1946–1956, ACM, 2019.

[15] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang, "Autoint: Automatic feature interaction learning via self-attentive neural networks," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1161–1170, 2019.

[16] Q. Song, D. Cheng, E. Zhou, J. Yang, Y. Tian, and X. Hu, "Towards automated neural interaction discovering for click-through rate prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 2020.

[17] E. Bernhardsson, *Annoy: Approximate Nearest Neighbors in C++/Python*, 2018. https://github.com/spotify/annoy/.

[18] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019. https://github.com/facebookresearch/faiss.

[19] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[20] J. Bennett, S. Lanning, *et al.*, "The netflix prize," Citeseer, 2007.

[21] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning*, pp. 791–798, 2007.

[22] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, pp. 285–295, 2001.

[23] I. Bayer, X. He, B. Kanagal, and S. Rendle, "A generic coordinate descent framework for learning from implicit feedback," *CoRR*, vol. abs/1611.04666, 2016.

[24] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua, "Fast matrix factorization for online recommendation with implicit feedback," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 549–558, 2016.

[25] D. Liang, L. Charlin, J. McInerney, and D. M. Blei, "Modeling user exposure in recommendation," in *Proceedings of the 25th international conference on World Wide Web*, pp. 951–961, 2016.

[26] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," *arXiv preprint arXiv:1205.2618*, 2012.

[27] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[28] K.-H. Lai, T.-H. Wang, H.-Y. Chi, Y. Chen, M.-F. Tsai, and C.-J. Wang, "Superhighway: Bypass data sparsity in cross-domain cf."

[29] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th international conference on World Wide Web*, pp. 111–112, 2015.

[30] H. Wang, N. Wang, and D. Yeung, "Collaborative deep learning for recommender systems," *CoRR*, vol. abs/1409.2944, 2014.

[31] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," *Advances in neural information processing systems*, vol. 26, 2013.

[32] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 353–362, 2016.

[33] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *Proceedings of the ADKDD'17*, pp. 1–7, 2017.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[35] Y. Chen, Q. Song, and X. Hu, "Techniques for automated machine learning," 2019.

[36] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," 2018.

[37] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*, pp. 507–523, Springer, 2011.

[38] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms," 2012.

[39] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[40] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[41] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," 2012.

[42] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.

[43] “Click-through rate prediction,” 2015.

[44] “Display advertising challenge,” 2014.

APPENDIX A

CODES

AutoRec GitHub: https://github.com/datamllab/AutoRec