Recommending Courses in MOOCs for Jobs: An Auto Weak Supervision Approach

Bowen Hao[1], Jing Zhang[1]✉, Cuiping Li[1], Hong Chen[1], and Hongzhi Yin[2]

[1] Key Laboratory of Data Engineering and Knowledge Engineering of Ministry of Education, School of Information, Renmin University of China
[2] The University of Queensland
{jeremyhao, zhang-jing, licuiping, chong}@ruc.edu.cn
h.yin1@uq.edu.cn

Abstract. The proliferation of massive open online courses (MOOCs) demands an effective way of course recommendation for jobs posted in recruitment websites, especially for the people who take MOOCs to find new jobs. Despite the advances of supervised ranking models, the lack of enough supervised signals prevents us from directly learning a supervised ranking model. This paper proposes a general automated weak supervision framework (*AutoWeakS*) via reinforcement learning to solve the problem. On the one hand, the framework enables training multiple supervised ranking models upon the pseudo labels produced by multiple unsupervised ranking models. On the other hand, the framework enables automatically searching the optimal combination of these supervised and unsupervised models. Systematically, we evaluate the proposed model on several datasets of jobs from different recruitment websites and courses from a MOOCs platform. Experiments show that our model significantly outperforms the classical unsupervised, supervised and weak supervision baselines.

1 Introduction

Massive open online courses, or MOOCs, are attracting widespread interest as an alternative education model. Lots of MOOCs platforms such as Coursera, edX and Udacity have been built and provide low cost opportunities for anyone to access a massive number of courses from the worldwide top universities. As reported by Harvard business review[3], a primary goal of 52% of the people surveyed who takes MOOCs is to improve their current jobs or find new jobs. We call this group of MOOCs' users as career builders. Meanwhile, people usually resort to the online recruitment platforms such as LinkedIn.com and Job.com to seek jobs. However, there always exists a "Skill Gap" [1] between the career builders and the employers. The career builders who expect themselves to fit a job through taking MOOCs, need to deeply understand the demands of the job skills and then take the matchable courses. Clearly, to help career builders

[3] https://hbr.org/2015/09/whos-benefiting-from-moocs-and-why

improve their skills for finding gainful jobs, it has been an essential task that is able to automatically match jobs with suitable courses.

Straightforwardly, to solve this problem, unsupervised methods such as BM25 [2], Word2vec [3] or the network embedding methods such as DeepWalk [28] and LINE [6] can be used to calculate the relevance between a queried job and a candidate course. However, such unsupervised methods aim at modeling the implicit structures of the input data, i.e., the clustering of words in jobs and courses, while the ranking of different courses to a queried job cannot be obviously learned. In another word, the unsupervised models do not explicitly compare the relevance of the positive courses and the negative courses to a queried job. Although the supervised neural ranking models are demonstrated to have good performance in the information retrieval (IR) tasks [7, 12], they cannot directly solve our problem, as the supervision signals about which courses can be recommended to a job are not easily available.

To alleviate the problem of lacking supervision signals, weak supervision models are proposed to train supervised IR models upon the pseudo labels provided by unsupervised models. For example, Dehghani et al. leverage the output of BM25 as the weak supervision signals [8] and Zamani and Croft extend a single pseudo signal to multiple signals to guide multiple supervised ranking models [9]. However, for different tasks, human efforts are demanded to determine the suitable weak signals and the supervised models. Even if each component is carefully selected by humans, their combination may not result in the best performance (which is also justified in our experiments). Thus, it is imperative to automatically identify an optimal combination of different components.

To address the above challenge, we propose a general automated weak supervision model *AutoWeakS*, which can automatically select the optimal combination of weak signals, supervised models and hyperparameters for a given ranking task and dataset. Specifically, the auto model trains a weak supervision model and a controller iteratively through reinforcement learning, where the weak supervision model aims to train a group of sampled supervised ranking models upon the pseudo labels (i.e., weak signals) provided by a group of sampled unsupervised ranking models, and the controller targets at automatically sampling an optimal configuration for the weak supervision model, i.e., it sequentially determines which unsupervised models should be sampled, how to set the hyperparameters for merging the unsupervised models, and which supervised models should be sampled. Our proposed model is a general framework to rank courses for jobs in this paper, but it is general enough to solve other ranking problems. Besides, we can incorporate any unsupervised and supervised models as candidate components to be selected by the controller.

Our contributions can be summarized as: (1) we are the first to explore the problem of recommending courses in MOOCs to jobs posted in online recruitment websites, which can help to eliminate the "Skill Gap" between the career builders who take MOOCs and the employers in the recruitment; (2) we propose a general automated weak supervision model, *AutoWeakS*, to rank courses for jobs. With reinforcement joint training of the weak supervision model and

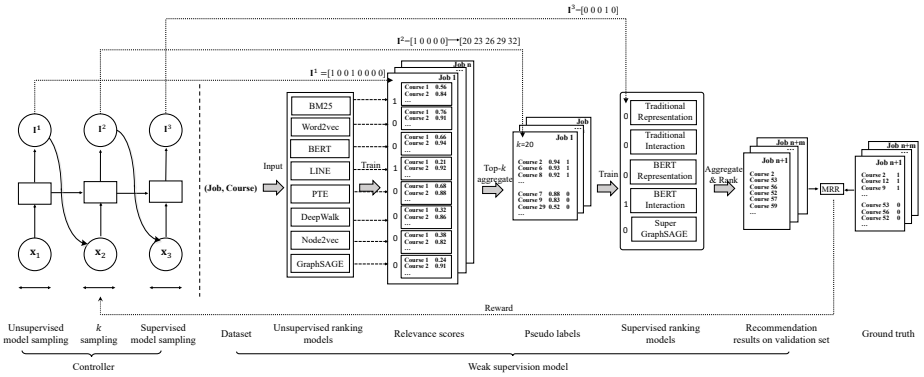


Fig. 1. Overview of the auto weak supervision model.

the controller in *AutoWeakS*, we can automatically find the best configuration of the weak supervision model; (3) experiments on two real-world datasets of jobs and courses show that *AutoWeakS* significantly outperforms the classical unsupervised, supervised and weak supervision baselines.

2 The Auto Weak Supervision Model

We denote the jobs in a recruitment website as J and the courses in a MOOCs platform as C, where each job $j \in J$ contains maximal N words, i.e., $j = \{j_1, \cdots, j_N\}$, and each course $c \in C$ contains maximal M words, i.e., $c = \{c_1, \cdots, c_M\}$. The words are extracted from the descriptions of the jobs or courses. Given J and C, the goal is to learn a predictive function $\mathcal{F} : (J, C) \rightarrow Y$ to predict the label $y \in Y$ for each pair of a queried job j and a candidate course c, where y is a binary value with $y = 1$ if c is relevant to j and $y = 0$ otherwise.

2.1 Model Overview

In our problem, the set of the true labels Y about which courses should be recommended to a given job are not easily to be obtained, which motivates us to use a weak supervision method to solve this task, i.e., training supervised ranking models upon the pseudo labels provided by unsupervised models. However, selecting only one unsupervised model may suffer from the issue of ranking bias, while combing multiple unsupervised models may bring in additional noises. Besides, we also have many different choices for the supervised ranking models. Thus we explore an optimal combination of the pseudo labels from various unsupervised ranking models, together with various supervised ranking models.

Fig. 1 illustrates the proposed auto weak supervision framework *AutoWeakS*, which consists of a weak supervision model and a controller, where the weak supervision model aggregates the results of multiple unsupervised ranking models as the pseudo labels and trains multiple supervised ranking models upon them,

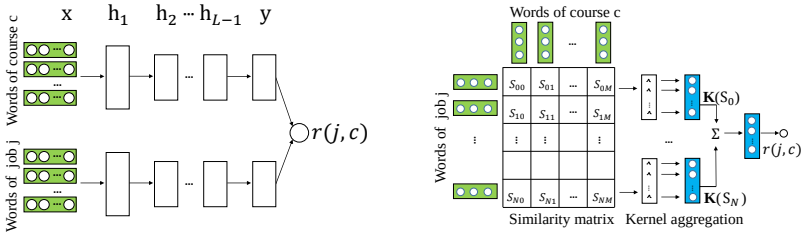
and the controller is responsible for automatically searching the optimal configuration of the unsupervised and the supervised models. Specifically, first, the controller is to sample the unsupervised models, sample the number k of the top ranked courses to a job after aggregating the results of the unsupervised models, and sample the supervised models to be trained upon the top-k pseudo labels, and since the above three sampling processes should be sequentially determined, we formalize the controller by a three-step LSTM model; second, the sampled supervised models are trained on the pseudo labels and evaluated on the validation data with a few human annotated labels; finally, the evaluation metric is returned as the reward signal to guide the training of the controller. When the whole training process is converged, we can obtain an optimal combination of different components, which can be regarded as the final model to predict the courses for new jobs.

2.2 Weak Supervision Model

In the weak supervision model, we conduct N^u unsupervised ranking models to calculate N^u relevance scores for each pair of a queried job and a candidate course, aggregate the N^u relevance scores to generate the pseudo labels, and train N^s supervised ranking models on these pseudo labels. Although we select the following unsupervised and supervised models in our framework, the framework is general to incorporate any kinds of unsupervised and supervised models.

Unsupervised Ranking Model. We define two types of the unsupervised ranking models, namely unsupervised text-only matching models and unsupervised graph-based matching models. Unsupervised text-only matching models calculate a relevance score between a queried job and a candidate course based on their descriptions. For example, *BM25* [2] exactly matches the words between a job and a course. *Word2vec* [3] and *BERT* [4] first embed the descriptions of a course and a job into two vectors, and then calculate the cosine similarity between these two vectors.

Different from the unsupervised text-only matching models which represent the jobs and the courses independently, unsupervised graph-based matching models leverage the global correlations between the jobs and courses to represent them. Specifically, we first build a job-word-course heterogeneous graph $G = (V, E)$, which consists of three types of nodes, i.e., job, course and word, and two types of edges that connect courses and words, and connect jobs and words. Then we apply different unsupervised network embedding models to map each node in G into a low-dimensional vector to capture the structural properties. For example, *LINE* [6] and PTE [10] maximize the first-order and the second-order proximity between two nodes. *DeepWalk* [28] extends the first-order neighbors to distinct neighbors which can be reached by random walks. *Node2vec* [11] further proposes the biased random walks to balance the homophily by BFS search and the structural equivalence by DFS search. *GraphSAGE* [5] aggregates the neighbors' embeddings of the nodes to represent them. Finally, we can calculate the relevance based on the learned embeddings of job and course.



(a) Traditional Representation Model (b) Traditional Interaction Model

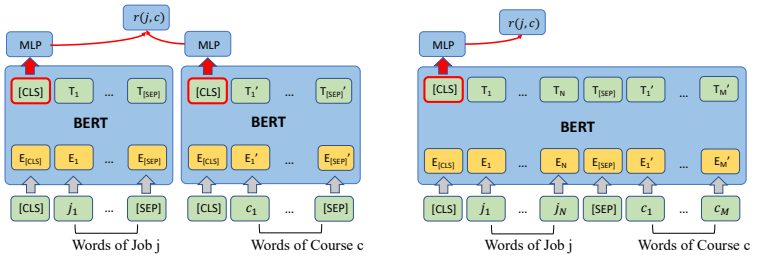
Fig. 2. Traditional supervised text-only matching models.

Pseudo Label Generator. To avoid the labeling bias from a single unsupervised model, we aggregate the results of the N^u unsupervised ranking models to generate the pseudo labels. Specifically, for each queried job, we average the N^u relevance scores for each candidate course, rank all the courses according to their average relevance scores, and then annotate the top-k courses as positive instances and the other courses as the negative instances for the queried job.

Supervised Ranking Model. We define two types of the supervised ranking models, including supervised text-only matching models and supervised graph-based matching models. For the supervised text-only matching models, we first explore two traditional models, namely traditional representation model and traditional interaction model, and inspired by the recently proposed pre-training model BERT [4], which has advanced the state-of-the-art in various NLP tasks, we further explore two BERT-based models, namely BERT representation model and BERT interaction model. Finally, we explore one supervised graph-based matching model, GraphSAGE.

Traditional Representation Model directly compares the embeddings of a queried job and a candidate course to capture their semantic relevance. Fig 2(a) illustrates the architecture of the model. We first transform the input word representations $\mathbf{x} \in \mathbb{R}^{N \times d_0}$ of a job into low-dimensional embeddings, and then apply multi-layer nonlinear projections on them to get the intermediate embeddings $\mathbf{h}_l \in \mathbb{R}^{d_l}$ and the final embedding $\mathbf{y} \in \mathbb{R}^{d_L}$ of a job, where N is the maximal number of words included in all the jobs, d_0, d_l and d_L represent the embedding dimensions. The paired inputs of courses are transformed in the same way. Finally, we estimate the relevance score $r(j, c)$ of c to j as the cosine similarity between the job embedding $\mathbf{y}_j$ and the course embedding $\mathbf{y}_c$.

Traditional Interaction Model compares each pair of the words in a queried job and a candidate course. Fig. 2(b) illustrates the model. Inspired by [13], we first build a similarity matrix $\mathbf{S}$ between the word embeddings of a queried job and a candidate course, where each element S_{ik} in the similarity matrix $\mathbf{S}$ stands for the cosine similarity between the embedding of the i-th word in job j and the embedding of the k-th word in course c. Then we trans-



(a) BERT Representation Model (b) BERT Interaction Model

Fig. 3. BERT-based supervised text-only matching models.

form the i-th row $\mathbf{S}_i = \{S_{i0}, \ldots, S_{iM}\}$ of the similarity matrix $\mathbf{S}$ into a feature vector $\mathbf{K}(\mathbf{S}_i) = \{K_1(\mathbf{S}_i), \ldots, K_H(\mathbf{S}_i)\}$, where $\mathbf{S}_i = \{S_{i0}, \ldots, S_{iM}\}$ represents the similarities between the i-th word of the queried job j and every word of the course c. Each of the h-th element is converted from $\mathbf{S}_i$ by the h-th RBF kernel with the mean value μ_h and the variance value σ_h, i.e., $K_h(\mathbf{S}_i) = \sum_{k=1}^{M} \exp[(S_{ik} - \mu_h)^2/2\sigma_h^2]$. Next, the similarity vectors of all the words in j are summed up into a similarity feature vector, i.e., $\sum_{i=1}^{N} \log\mathbf{K}(\mathbf{S}_i)$, which is then mapped into a one-dimension relevance score $r(j, c)$ to represent the relevance between the job j and the course c.

BERT Representation Model compares the embeddings of a queried job and a candidate course through independently encoding the descriptions of a queried job and a candidate course by the pre-training model BERT [4]. Fig. 3(a) illustrates the proposed model. Specifically, for each job j, we take [CLS], $j_1, \cdots, j_N$, [SEP] as the input, where $j_1, \cdots, j_N$ represent job tokens, [CLS] and [SEP] are special tokens. Then we add a multi-layer perceptron (MLP) layer on top of the first output [CLS] embedding to get the representation of job j. A course c is represented in the same way. Finally, we calculate the cosine similarity between the embeddings of the job and the course to obtain the relevance score $r(j, c)$.

BERT Interaction Model compares each pair of the words in a queried job and a candidate course through a unified BERT model, where the multi-head attention in the BERT unit spans over the interactions between the job and the course so that the job-course interactions can be captured. Fig. 3(b) illustrates the proposed model. Specifically, we take [CLS], $j_1, \cdots, j_N$, [SEP], $c_1, \cdots, c_M$ as the input, where $j_1, \cdots, j_N$ represent the job tokens and $c_1, \cdots, c_M$ represent the course tokens. Then we add a MLP layer on top of the first output [CLS] embedding to obtain the relevance score of the job-course pair.

SuperGraphSAGE Model aggregates the embeddings of the nodes' neighbors to represent them in a supervised end-to-end fashion. Specifically, given the job-word-course graph $G = (V, E)$, we invoke the bert-as-server API[4] to generate

[4] https://github.com/hanxiao/bert-as-service

the features for all nodes in G. For each job j, at the l-th convolutional time, it aggregates the embeddings of all its neighbors to obtain its new embedding $\mathbf{h}_j^l$, i.e., $\mathbf{h}_j^l{=}\sigma(\mathbf{W}^l \cdot \text{CONCAT}(\mathbf{h}_j^{l-1}, \mathbf{h}_{\mathcal{N}(j)}^l))$, where σ is a nonlinear function, $\mathbf{W}^l$ is the parameter matrix, $\mathbf{h}_{\mathcal{N}(j)}^l$ is the aggregated embedding of the job's neighbors, $\mathbf{h}_j^{l-1}$ is the previous embedding of the job, and CONCAT is the concatenate operation. We can obtain the l-th course embedding $\mathbf{h}_c^l$ in the same way. The cosine similarity between the final L-th embeddings of the course and the job can be viewed as their relevance score $r(j, c)$.

We use the loss function, $\mathcal{L} = \sum_{j,c+} \log \sigma(r(j, c^+)) + \sum_{j,c-} \log(1{-}\sigma(r(j, c^-)))$, to train the supervised models based on the pseudo labels provided by the unsupervised models. For a new (j, c) pair in the test set, we estimate their relevance $r(j, c)$ as the average of all the relevance scores predicted by the N^s supervised models, i.e., $r(j, c) = \frac{1}{N^s} \sum_{i=1}^{N^s} r_i(j, c)$.

2.3 Automated Model Search

To avoid human efforts to determine the suitable weak signals and the supervised models, in this paper, we propose to automatically search the optimal configuration of the weak supervision model. As the weak supervision model is a sequential process that first trains the unsupervised ranking models, then aggregates their outputs as the pseudo labels and finally trains the supervised ranking models based on the pseudo labels, we formalize the controller as a three-step LSTM model to sequentially determine which unsupervised models to select, which value of top-k to select and which supervised models to select. The controller maintains a representation for each choice of different components, i.e., each unsupervised model, each value of k and each supervised model. At step t, the representations of all the selections at $t-1$ are viewed as the input $\mathbf{x}_t \in \mathbb{R}^{d'}$, which is taken together with the previous hidden vector $\mathbf{h}_{t-1} \in \mathbb{R}^{d'}$ and the cell state $\mathbf{e}_{t-1} \in \mathbb{R}^{d'}$ to produce the hidden vector $\mathbf{h}_t$ and the cell state $\mathbf{e}_t$, i.e.,

$$\mathbf{h}_t, \mathbf{e}_t = \mathbf{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{e}_{t-1}, \Phi), \tag{1}$$

where Φ are the parameters of LSTM and d' is the hidden vector dimension. Finally, the component at step t is determined according to the hidden vector $\mathbf{h}_t$ and the representations of all the choices of the component at time t. Now we present the details of the sampling process:

Step 1: Unsupervised model sampling is to sample unsupervised models. At the beginning, the controller selects none of the components and has no memory, thus we set the initial hidden state $\mathbf{h}_0$, the cell state $\mathbf{e}_0$ as empty embeddings, and randomly initialize the input $\mathbf{x}_1$. The controller takes $\mathbf{h}_0$, $\mathbf{e}_0$ and $\mathbf{x}_1$ as input, and output $\mathbf{h}_1$ and $\mathbf{e}_1$ by Eq. (1). Then given $\mathbf{h}_1$, the controller samples the unsupervised models. Intuitively, an unsupervised model is more likely to be sampled if it is more related to the hidden vector $\mathbf{h}_1$ at this step. We randomly

initialize the representation $\mathbf{w}_i \in \mathbb{R}^{d'}$ for each unsupervised model, multiply $\mathbf{w}_i$ with $\mathbf{h}_1$ to represent their relevance, based on which we perform sampling:

$$I_i^1 \sim \text{softmax}(f(\mathbf{h}_1^T \times \mathbf{w}_i)), \tag{2}$$

where $\mathbf{h}_1^T \times \mathbf{w}_i$ is a d'-dimensional element-wise product between the two embeddings and f is a fully-connected layer that converts the product into a 2-dimensional vector. Softmax is used to convert the vector into a probability distribution, from which the indicator variable $I_i^1 \in \{0, 1\}$ is sampled to represent whether the i-th unsupervised model should be selected or not. Essentially, we sample each model from a binomial distribution. After this step, we can get the indicator vector $\mathbf{I}^1 = [I_1^1, \cdots, I_{N^u}^1]$ to indicate the selected unsupervised models. For example in Fig. 1, $\mathbf{I}^1 = [10010000]$ indicates the controller selects BM25 and LINE as the unsupervised models.

Step2: k sampling is to sample the value of k for selecting the top-k ranked positive instances in the pseudo labels. After sampling the unsupervised models, we multiply the indicator vector $\mathbf{I}^1 = [I_1^1, \cdots, I_{N^u}^1]$ with the model representations $[\mathbf{w}_1, \cdots, \mathbf{w}_{N^u}]$ as the input $\mathbf{x}_2$ of the second step:

$$\mathbf{x}_2 = [I_1^1, \cdots, I_{N^u}^1] \cdot [\mathbf{w}_1, \cdots, \mathbf{w}_{N^u}]^T, \tag{3}$$

where $\mathbf{x}_2$ denotes the summation of the representations of all the sampled models. With $\mathbf{x}_2$, $\mathbf{h}_1$ and $\mathbf{e}_1$ as the input, we can obtain $\mathbf{h}_2$ and $\mathbf{e}_2$ by Eq. (1). Since the value space of k can be very large, to simplify the sampling process, we first categorize all the values of k into τ categories and sample one category for k. The sampling process is defined as:

$$\mathbf{I}^2 \sim \text{softmax}(g(\mathbf{h}_2)), \tag{4}$$

where g is a full-connected layer that converts the hidden vector $\mathbf{h}_2$ into a τ-dimensional vector. Softmax is used to convert the vector into a probability distribution, from which the indicator vector $\mathbf{I}^2 \in \{0, 1\}^\tau$ is sampled to represent which category of k is selected. Note that $\mathbf{I}^2$ is a one-hot vector with only one dimension as one, whose index indicates the sampled category of k. Essentially, we sample the category of k from a multinomial distribution. For example in Fig. 1, $\mathbf{I}^2 = [10000]$ indicates the controller selects the first category for k and its corresponding value is 20.

Step 3: Supervised model sampling is to sample supervised models. After sampling k, we multiply the sampling indicator $\mathbf{I}^2$ with the concatenation of the k's category representations $[\mathbf{z}_1, \cdots, \mathbf{z}_\tau]$ as the input $\mathbf{x}_3$ of the third step:

$$\mathbf{x}_3 = [I_1^2, \cdots, I_\tau^2] \cdot [\mathbf{z}_1, \cdots, \mathbf{z}_\tau]^T, \tag{5}$$

where the category representations $[\mathbf{z}_1, \cdots, \mathbf{z}_\tau]$ for each category of k are randomly initialized. The input $\mathbf{x}_3$ denotes the representation of the selected category. With $\mathbf{x}_3$, $\mathbf{h}_2$ and $\mathbf{e}_2$ as the input, we can obtain $\mathbf{h}_3$ and $\mathbf{e}_3$ by Eq. (1). Given $\mathbf{h}_3$, we can sample the indicator $\mathbf{I}^3$ following the same sampling process of step 1 to determine which supervised models should be selected, i.e., $I_i^3 \sim \text{softmax}(q(\mathbf{h}_3^T \times \mathbf{u}_i))$, where q is a fully-connected layer that converts the product $\mathbf{h}_3^T \times \mathbf{u}_i$ into a 2-dimensional vector, $\mathbf{u}_i \in \mathbb{R}^{d'}$ is a randomly initialized embedding for the i-th supervised model. As shown in Fig. 1, $\mathbf{I}^3 = [00010]$ indicates the controller selects BERT interaction model.

2.4 Reinforcement Joint Training

Once the controller finishes searching the configurations of the weak supervision model, i.e., the unsupervised models, the top-k value and the supervised models, a combination with this architecture is built and trained. When the searching architecture achieves convergence, it will get an accuracy $\mathcal{R}$ on a small hold-out annotated dataset (validation set). The accuracy $\mathcal{R}$ is viewed as reward and the parameters of the controller LSTM are then optimized in order to search the best configurations that can achieve the maximal expect validation accuracy. In this paper, we propose a reinforcement joint training process to update the parameters of the controller LSTM, denoted by Φ, and the parameters of the weak supervision model, denoted by Θ. The reinforcement joint training process consists of two interleaving phrases (Algorithm 1), the first phrase trains Θ, while the second phrase trains Φ, the details are as follows:

Training Θ. When training Θ, we fix the controller's sampling policy $\pi(\mathbf{m}; \Phi)$, i.e., the three-step sampling strategy, and perform stochastic gradient descent on Θ to maximize the expected loss $\mathbb{E}_{\mathbf{m}\sim\pi}[\mathcal{L}(\mathbf{m}; \Theta)]$, where $\mathcal{L}(\mathbf{m}; \Theta)$ is the loss computed on a minibatch of training data, with a weak supervision model $\mathbf{m}$ sampled from $\pi(\mathbf{m}; \Phi)$. The gradient is computed using Monte Carlo estimate:

$$\nabla_\Theta \mathbb{E}_{\mathbf{m}\sim\pi}[\mathcal{L}(\mathbf{m}; \Theta)] \approx \frac{1}{N_m} \sum_{p=1}^{N_m} \sum_{q=1}^{N'_u+N'_s} \nabla_\Theta \mathcal{L}_q(\mathbf{m}_p, \Theta), \tag{6}$$

where N_m is the sampling times of the weak supervision model in one epoch. It is empirically proven that N_m=1 works just fine [14]. Notations N'_u and N'_s represent the number of the sampled unsupervised and the supervised models respectively. The whole loss $\mathcal{L}(\mathbf{m}_p, \Theta)$ is the summed losses of all the sampled models. No matter which unsupervised ranking models are sampled, they are always trained on the same training data. So we can pre-train each unsupervised ranking model on the training data, and directly fetch the relevance scores between jobs and courses from all the sampled unsupervised models during each update of Eq. (6). Thus after each sampling of $\mathbf{m}$, we only need to re-optimize the loss $\mathcal{L}_q$ of each sampled supervised model. As a result, the loss $\mathcal{L}(\mathbf{m}_p, \Theta)$ is the summed losses of all the sampled supervised models.

Input: A set of jobs J and a set of courses C.
Output: Parameters Θ of the weak supervision model and Φ of the controller.
Initialize $\Phi = \Phi^0$, $\Theta = \Theta^0$;
Pre-train the N^u unsupervised ranking models;
repeat
  Sample a weak supervision model $\mathbf{m}$ from $\pi(\mathbf{m}; \Phi)$;
  Train Θ of $\mathbf{m}$ by Eq. (6);
  Calculate $\mathcal{R}^s + \mathcal{R}^u$ of $\mathbf{m}$ on the validation set;
  Update Φ in the controller by REINFORCE;
until *Convergence*;

Algorithm 1: Reinforcement Jointly Training

Training Φ. When training Φ, we fix Θ of the weak supervision model and perform REINFORCE algorithm [15] on Φ to maximize the expected reward $\mathbb{E}_{\mathbf{m}\sim\pi(\mathbf{m};\Phi)}[\mathcal{R}(\mathbf{m}, \Theta)]$. The actions of the controller are to sample the unsupervised models, k, and the supervised models sequentially. The reward is regarded as the evaluated mean reciprocal rank (MRR) of the sampled supervised models on the validation set, which is the set of a few job-course pairs annotated by human beings. Besides, the MRR achieved by the aggregation results of the sampled unsupervised models can also be regarded as the additional reward to accelerate the training process [16]. Thus, the final reward is defined as $\mathcal{R} = \mathcal{R}^s + \mathcal{R}^u$, where $\mathcal{R}^s$ and $\mathcal{R}^u$ are the rewards from the supervised and unsupervised models.

3 Experiment

In this section, we evaluate our proposed model *AutoWeakS* against several unsupervised, supervised, and weak supervision baselines. We also explore whether the selections for each component in *AutoWeakS* (i.e., the selections for the unsupervised models, the supervised models and the top-k values) are necessary.

3.1 Experimental Setup

Dataset. We collect all the courses from XuetangX[5], one of the largest MOOCs in China, and this results in 1951 courses. The collected courses involve seven areas: computer science, economics, engineering, foreign language, math, physics, and social science. Each course contains 131 words in its descriptions on average. We also collect 706 job postings from the recruiting website operated by JD.com[6] (JD) and 2,456 job postings from the website owned by Tencent corporation[7] (Tencent). The collected job postings involve six areas: technical post, financial post, product post, design post, market post, supply chain and engineering post. Each job contains 107 and 151 words in its posting on average in

[5] http://www.xuetangx.com
[6] http://campus.jd.com/home
[7] https://hr.tencent.com/

Table 1. Overall performance of recommending courses for jobs. We try different k for WeakS and report its best performance.

Model	JD-XuetangX			Tencent-XuetangX		
	HR@5	NDCG@5	MRR	HR@5	NDCG@5	MRR
BM25	0.162	0.151	0.173	0.072	0.046	0.070
Word2vec	0.301	0.212	0.217	0.142	0.107	0.114
BERT	0.348	0.239	0.238	0.159	0.104	0.122
LINE	0.489	0.362	0.409	0.396	0.284	0.279
PTE	0.378	0.244	0.334	0.295	0.204	0.210
DeepWalk	0.390	0.249	0.258	0.370	0.262	0.261
Node2vec	0.374	0.279	0.284	0.386	0.282	0.277
GraphSAGE	0.312	0.252	0.232	0.186	0.121	0.139
Traditional Representation	0.407	0.261	0.262	0.201	0.125	0.148
Traditional Interaction	0.470	0.429	0.414	0.324	0.215	0.214
BERT Representation	0.350	0.232	0.231	0.294	0.195	0.204
BERT Interaction	0.564	0.537	0.497	0.405	0.254	0.222
SuperGraphSAGE	0.263	0.176	0.186	0.231	0.144	0.155
WeakS	0.704	0.548	0.592	0.370	0.255	0.227
LINE+AllS	0.736	0.550	0.624	0.408	0.289	0.236
AutoWeakS	**0.793**	**0.615**	**0.671**	**0.631**	**0.522**	**0.540**

JD and Tencent respectively. To evaluate the model performance, for both JD and Tencent dataset, we randomly select 200 jobs, and ask ten volunteers to annotate the relevant courses to the jobs. Specifically, for a queried job, we first use each unsupervised model in Section 2.2 to calculate a relevance score for each course, average all the scores over all the models, select top 60 candidate courses, annotate each candidate and obtain the ground truth by majority voting of all the volunteers' annotations. The Dataset and the code are online now[8].

Settings. For training the unsupervised ranking models, we use all the 706 job postings from JD and all the 2,456 job postings from Tencent to learn the embeddings of the jobs and courses. For the supervised ranking models, we hold out the human annotated jobs and only use the 506 unlabeled jobs from JD and 2,256 unlabeled jobs from Tencent for training. On each dataset, we averagely partition the annotated 200 jobs into a validation set and a test set and sample 99 negative instances for each positive instance (1 positive plus 99 negatives) [17]. We use Hit Ratio of top K items (HR@K), Normalized Discounted Cumulative Gain of top K items (NDCG@K) and Mean Reciprocal Rank (MRR) as the evaluation metrics for ranking, where K is set as 5.

[8] https://github.com/jerryhao66/AutoWeakS

3.2 Experimental Results

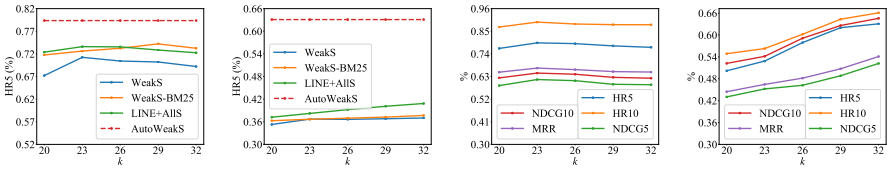
Comparison with Baselines. In this experiment, we evaluate our model *AutoWeakS* against the unsupervised and supervised models in Section 2.2, the weak supervision model WeakS, which includes all the unsupervised models and the supervised models without model search, and one competitive baseline, LINE+AllS, which includes LINE as the unsupervised model and all the supervised models. Note that due to the lack of enough labeled data, we only use a small annotation data (i.e., the validation set) to train the supervised models.

Table 1 shows the results on two datasets. We vary the value k for WeakS, LINE+AllS and report its best performance in Table 1. From the results, we can see that the proposed *AutoWeakS* performs clearly better than other baselines. Compared with the unsupervised graph-based matching methods, unsupervised text-only matching models perform worse, as only using the descriptive words of the jobs and the courses can not capture high-order relationships between the jobs and the courses. Some supervised methods such as BERT interaction model and traditional interaction model perform better than the unsupervised methods, as the unsupervised methods do not explicitly compare the relevance of the positive courses and the negative courses to a queried job. However, due to the lack of enough training labels, the performance of the supervised models is worse than WeakS, LINE+AllS and our proposed method *AutoWeakS*.

Table 2. Performance of different choices of unsupervised models in AutoWeakS with k and the supervised component fixed.

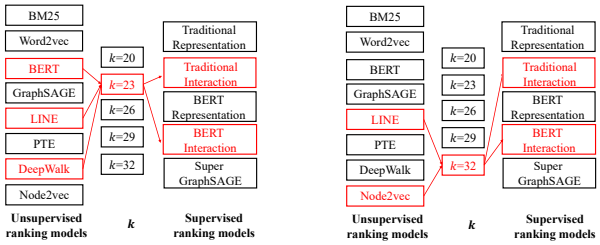
Unsuper. choices	JD-XuetangX			Tencent-XuetangX		
	HR@5	NDCG@5	MRR	HR@5	NDCG@5	MRR
BM25+	0.203	0.194	0.182	0.183	0.139	0.159
Word2vec+	0.435	0.392	0.336	0.333	0.312	0.321
BERT+	0.705	0.511	0.511	0.393	0.373	0.387
LINE+	0.722	0.559	0.516	0.589	0.478	0.499
PTE+	0.657	0.488	0.505	0.471	0.451	0.497
DeepWalk+	0.677	0.503	0.462	0.508	0.461	0.411
Node2vec+	0.684	0.507	0.451	0.534	0.449	0.463
GraphSAGE+	0.642	0.495	0.402	0.563	0.471	0.491
All unsupervised+	0.609	0.423	0.458	0.415	0.396	0.426
AutoWeakS	**0.793**	**0.615**	**0.671**	**0.631**	**0.522**	**0.540**

WeakS performs better than all the unsupervised models on JD-XuetangX, as it explicitly learns the ranking of the candidate courses to queried jobs. However, on Tencent-XuetangX, WeakS underperforms several unsupervised models, because BM25 performs particularly poorly on this dataset, which reduces the effect of the aggregated pseudo labels from all the unsupervised models. This



(a) JD-XuetangX (b) Tencent-XuetangX (c) JD-XuetangX (d) Tencent-XuetangX

Fig. 4. (a), (b) show that the baselines with any choice of top-k underperform AutoWeakS with the automatically searched top-k value. (c), (d) further present the results of AutoWeakS under different top-k values, which indicates that the automatically searched top-k value performs the best against all the other top-k values.



(a) JD-XuetangX (b) Tencent-XuetangX

Fig. 5. The optimal selections of AutoWeakS.

also indicates that indiscriminately combing all the models may not result in the best performance. Besides, *AutoWeakS* beats LINE+AllS, which implies selecting only one unsupervised model may suffer from the issue of ranking bias.

We further remove BM25 from WeakS, name the model as WeakS-BM25 and show the performance of WeakS, LINE+AllS and WeakS-BM25 in Fig. 4(a) and Fig. 4(b). The k value of *AutoWeakS* is fixed as the automatically searched value. The results show that even if the worst performed BM25 is removed, given any value of k, WeakS-BM25 still underperforms *AutoWeakS*, which indicates the advantage of the automated model search in *AutoWeakS*. For training *AutoWeakS*, at each epoch, it takes only a few seconds to train the controller, about 30 minutes to train the sampled weak supervision model on JD-XuetangX and 1 hours on Tencent-XuetangX. The controller can converge after 30 epochs.

Analysis of Unsupervised Component. We evaluate the performance of different choices of the unsupervised models, when fixing the sampled k and the supervised component in *AutoWeakS*. We name the model as BM25+ if only BM25 is chosen to produce pseudo labels. Other single model is named in the same way. All unsupervised+ means we combine the labels of all the unsupervised models. Fig. 5(a) shows that on JD-XuetangX, *AutoWeakS* selects the combination of BERT, LINE, and DeepWalk, which performs better than any single unsupervised model and All unsupervised+ shown in Table 2. On Tencent-

Table 3. Performance of different choices of supervised models in AutoWeakS with the unsupervised component and k fixed.

Super. choices	JD-XuetangX			Tencent-XuetangX		
	HR@5	NDCG@5	MRR	HR@5	NDCG@5	MRR
Traditional Representation+	0.525	0.349	0.335	0.362	0.283	0.269
Traditional Interaction+	0.679	0.508	0.483	0.601	0.502	0.504
BERT Representation+	0.604	0.483	0.462	0.318	0.209	0.209
BERT Interaction+	0.729	0.537	0.609	0.576	0.464	0.481
SuperGraphSAGE+	0.348	0.212	0.368	0.265	0.181	0.193
All supervised+	0.652	0.482	0.507	0.402	0.317	0.301
AutoWeakS	**0.793**	**0.615**	**0.671**	**0.631**	**0.522**	**0.540**

XuetangX, *AutoWeakS* also obtains the best performance, and it selects the combination of LINE and Node2vec as the unsupervised component. The results indicate the advantage of automatically searching the unsupervised models.

Analysis of k. We evaluate the performance of different choices of k to generate the pseudo labels, when fixing the sampled unsupervised and the supervised components in *AutoWeakS*. Fig. 5 presents that the automatically searched k is 23 on JD-XuetangX and is 32 on Tencent-XuetangX. Fig. 4(c) and Fig. 4(d) show that *AutoWeakS* with other k values underperforms the searched k values. The results indicate the advantage of automatically searching k.

Analysis of Supervised Component. We evaluate the performance of different choices of the supervised models, when fixing the sampled unsupervised component and k in *AutoWeakS*. We name the model as BERT Interaction+ if only the BERT interaction model is trained. Other single supervised model is named in the same way. All supervised+ means we train all the supervised models. Fig. 5(a) and Fig. 5(b) show that on both of the JD-XuetangX and the Tencent-XuetangX datasets, *AutoWeakS* selects the combination of the BERT interaction model and the traditional interaction model. The results show that *AutoWeakS* performs better than all the other choices shown in Table 3, which indicates the advantage of automatically searching the supervised models.

4 Related Work

This paper first attempts to match the jobs in the recruitment websites and the courses in MOOCs to help the career builders who take MOOCs fill their skill gap in their pursuing jobs. The related works include:

Weak Supervision Model. Training neural ranking models on pseudo-labeled data has been attracted attentions. For example, Dehghani et al. [8] leverage the output of traditional IR models such as BM25 as the weak supervision signal

to generate a large amount of pseudo labels to train effective neural ranking models. Zamani et al. [9] train a neural query performance predictor by multiple weak supervision signals, and they also provide a theoretical analysis of this weak supervision method [18]. The same idea is employed in [19, 20]. However, for different tasks, human efforts are demanded to determine the suitable weak signals and the supervised models. Even if each signal is carefully selected by humans, their combination may not be optimal.

Automated Machine Learning (AutoML). The goal of AutoML is to automatically determine the optimal configurations such as selecting the optimal models [21, 22], features [23, 24], and neural architecture [25, 26], which can help people use machine learning models easily. Different types of techniques are studied to search the optimal configuration. For example, Bayesian optimization methods such as Auto-sklearn [21] and Auto-Weka [22] model the relationship between a configuration and the corresponding performance in a probabilistic way. Reinforcement learning trains the optimal search policies according to the feedbacks of the searched configurations [27], where the search policy can be modeled by RNN [14, 25]. Inspired by the above works, we propose a RL-based joint training framework to search an optimal combination of the unsupervised/supervised models and the hyperparameter k in the proposed weak supervision model for recommending courses for jobs.

5 Conclusion

We present the first attempt to solve the problem of recommending courses in MOOCs for jobs by a general automated weak supervision model. With reinforcement joint training of a weak supervision model for recommending courses and a controller for searching models, we can automatically find the best configuration of the weak supervision model. Experiments on two real-world datasets of jobs and courses show that the proposed AutoWeakS significantly outperforms the classical unsupervised, supervised and weak supervision baselines.

ACKNOWLEDGMENTS

This work is supported by National Key R&D Program of China (No.2018YFB1004401) and NSFC (No.61532021, 61772537, 61772536, 61702522).

References

1. Xu, T., Zhu, H., Zhu, C., Li, P., Xiong, H.: Measuring the popularity of job skills in recruitment market: A multi-criteria approach. In AAAI'17, pp.2572-2579.
2. Robertson, S., Zaragoza, H.: The probabilistic relevance framework: BM25 and beyond. Foundations and Trends® in Information Retrieval, 2009, 3(4), 333-389.
3. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J.: Distributed Representations of Words and Phrases and their Compositionality. In NeurIPS'13.

4. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In NAACL'19, pp.4171-4186.
5. Hamilton, W. L., Ying, Z., Leskovec, J.: Inductive Representation Learning on Large Graphs. In NeurIPS'17, pp.1024-1034.
6. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In WWW'15, pp. 1067-1077, pp.1067-1077.
7. Qiu, X., Huang, X. Convolutional neural tensor network architecture for community-based question answering. In IJCAI'15, pp. 1305-1311.
8. Dehghani, M., Zamani, H., Severyn, A., Kamps, J., Croft, W. B.: Neural ranking models with weak supervision. In SIGIR'17, pp. 65-74.
9. Zamani, H., Croft, W. B., Culpepper, J. S.: Neural query performance prediction using weak supervision from multiple signals. In SIGIR'18, pp. 105-114.
10. Tang, J., Qu, M., Mei, Q.: Pte: Predictive text embedding through large-scale heterogeneous text networks. In KDD'15, pp. 1165-1174.
11. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In SIGKDD'16, pp. 855-864.
12. Pang, L., Lan, Y., Guo, J., Xu, J., Wan, S., Cheng, X.: Text matching as image recognition. In AAAI'16, pp.2793-2799.
13. Xiong, C., Dai, Z., Callan, J., Liu, Z., Power, R.: End-to-end neural ad-hoc ranking with kernel pooling. In SIGIR'17, pp. 55-64.
14. Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., Dean, J. (2018). Efficient neural architecture search via parameter sharing. In ICML'18, pp. 4092-4101.
15. Williams, R. J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4), 229-256.
16. Ghavamzadeh, M., Mahadevan, S.: Hierarchical policy gradient algorithms. Computer Science Department Faculty Publication Series, 173.
17. He, X., He, Z., Song, J., Liu, Z., Jiang, Y. G., Chua, T. S.: NAIS: Neural attentive item similarity model for recommendation. IEEE TKDE, 30(12), 2354-2366.
18. Zamani, H., Croft, W. B.: On the theory of weak supervision for information retrieval. In SIGIR'18, pp. 147-154.
19. Dehghani, Mostafa, et al.: "Avoiding your teacher's mistakes: Training neural networks with controlled weak supervision." arXiv preprint arXiv:1711.00313 (2017).
20. Luo, C., Zheng, Y., Mao, J., Liu, Y., Zhang, M., Ma, S.: Training deep ranking model with weak relevance labels. In Australasian Database Conference.
21. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In NeurIPS'15, pp. 2962-2970.
22. Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., Leyton-Brown, K. (2017). Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. The Journal of Machine Learning Research, 18(1), 826-830.
23. Katz, G., Shin, E. C. R., Song, D.: Explorekit: Automatic feature generation and selection. In ICDM'16, pp. 979-984.
24. Huang, S., Wang, C., Ding, B., Chaudhuri, S.: Efficient identification of approximate best configuration of training in large datasets. In AAAI'19, pp. 3862-3869.
25. Zoph, B., Le, Q. V.: Neural architecture search with reinforcement learning. In ICLR'17.
26. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L, Murphy, K.: Progressive Neural Architecture Search. In ECCV'18, pp. 19-34.
27. Sutton, R. S., Barto, A. G. (2011).: Reinforcement learning: An introduction. In AI Magazine, pp. 15-34.
28. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In SIGKDD'14, pp. 701-710.