Jobs Filter to Improve the Job Seeker Experience at Indeed.com

Shichuan Ma
Indeed.com
Sunnyvale, USA
shichuannm@indeed.com

Haiyan Luo
Indeed.com
Sunnyvale, USA
hluo@indeed.com

Jianjie Ma
Indeed.com
Sunnyvale, USA
jma@indeed.com

Ziying Liu
Indeed.com
Sunnyvale, USA
ziyingl@indeed.com

Yu Sun
Indeed.com
Sunnyvale, USA
sunyu@indeed.com

Xingang Huang
Indeed.com
Sunnyvale, USA
xingangh@indeed.com

Fengdan Wan
Indeed.com
Sunnyvale, USA
fwan@indeed.com

Veeresh Beeram
Indeed.com
Sunnyvale, USA
veeresh@indeed.com

Henry Oh
Indeed.com
Sunnyvale, USA
henryoh@indeed.com

Santosh Raghuram Kumar
Indeed.com
Sunnyvale, USA
santoshk@indeed.com

Sreenivasa Reddy Ambati
Indeed.com
Sunnyvale, USA
sreenivasa@indeed.com

ABSTRACT

Indeed.com has grown rapidly over the past few years with the mission to help more people get jobs. Many applications have sprouted up to better serve job seekers' different needs, such as search, recommendations, email invitation and so on. Ensuring same level of great user experience across these different channels have become an increasing challenge for the company, due to slightly different objectives and optimization goals of these different applications.

In this work, we solved the challenge by building a large scale jobs filter that focuses on identifying and removing negative matches of jobs and job seekers, which would normally have markedly adverse impact on user experience. The system encompasses a rule-based engine and various machine learning technologies. It has been running in production since 2019 last year with significantly improved performance results.

1 INTRODUCTION

Indeed is the #1 job site in the world with over 250 million unique visitors every month. As a company, Indeed strives to put job seekers first, giving them free access to search for jobs, post resumes, and research companies. Job seekers come to indeed.com to find their perfect jobs. Employers come to our website to post and sponsor jobs. Every day, we connect millions of people to new opportunities.

As Indeed continues to grow, we're finding more ways to help people get jobs. We're also offering more ways job seekers can see those jobs. For instance, job seekers can search directly on Indeed.com, receive job recommendations, view jobs through Targeted Display Ads, or even receive email invitations to apply – to name just a few. While each channel presents jobs in a slightly different way, our goal for each remains the same: showing the right jobs to the right job seekers.

However, if we miss the mark with the jobs we present, job seekers may lose trust in our ability to connect them with their next opportunities. Our mission is to help people get jobs, not waste their time. Thus, ensuring a universal high quality job to job seeker match across all channels becomes critically important to ensure good user experience that our job seekers have been enjoying with the Indeed provided services.

Some of the common ways we would consider a job to be wrong for a job seeker are if it:

- Pays less than their expected salary range
- Requires special licenses or certificates they do not have
- Is located outside their preferred geographic area
- Is in a completely unrelated field, such as CTOs being offered barista jobs
- Is in a related filed but mismatched, such as nurses and doctors being offered the same jobs

In order to mitigate this issue, we built an large scale intelligent jobs filter, which can be applied to user facing applications so that we can filter out jobs that are obviously mismatched with the job seeker before they are shown to the job seeker. Our solution uses a combination of carefully designed rules and machine learning technologies, and our analysis shows it to be very efficient and effective. The major contributions of our work include

- This is a real scalable production system in a large ecosystem with successful metrics improvement. We present a few design challenges on some of the innovative engineering considerations. We also share some of our operational experience in our unique business environment.
- We innovatively use a combination of a rule based engine and machine learning technologies especially deep learning to achieve both efficiency and effectiveness and to achieve excellent performance enhancement.

- All these rules can be easily plugged in and adapted for different applications dynamically, so that they are very easy to use and scale.

We hope our work can shed some light on similar problems in the industry. The rest of the paper is organized as follows. In Section 2 we provide the peer work in the related fields. Section 3 gives the system architecture. Section 4 and Section 5 describe in details two of the machine learning based rules we use in the system and their performance evaluation. Section 6 gives some operational experience while section 7 concludes this paper.

2 RELATED WORK

In the work [2], the authors present Wide & Deep learning to jointly train wide linear models and deep neural networks so as to combine the benefits of memorization and generalization for recommender systems. In our jobs filter work, one of our rules are based on this research with some adaptation and engineering innovation.

Authors in [8] use embedding to combine two approaches to represent longer pieces of text while having minimal computational complexity, i.e. meaningful combining word vectors through simple vector addition and learning representations of phrases through a single token.

Further, in our previous work [7], we presented to use graph embedding to learn job representation based on job seeker's job position progress and employment change and used it in career move prediction at Indeed.com with encouraging performance. A few of our embedding-based rules are based on our previous research work.

For the machine learning and rule-engine hybrid applications, an example can be found in [12], where the authors combine a machine learning algorithm with a rule-based expert system to improve the accuracy of results by filtering false positives and dealing with false negatives, where the machine learning algorithm provides a base model trained with a label corpus. Our work is also designed on the basis of a hybrid of machine learning and a rule-based engine to achieve job and job seeker match efficiency and effectiveness.

3 SYSTEM ARCHITECTURE

Engineering wise, the jobs filter is a complex system due to its scalability requirement and the support of various applications. At a high level, it consists of the following components, as shown in Figure 1:

- **Jobs Filter Service.** This is a high throughput, low latency application service that evaluates potential match-ups of jobs to users, identified by ID. If the service determines that the job is appropriate for the user ID, it returns an ALLOW decision; otherwise it returns a VETO. This service is horizontally scalable so it can serve many real-time Indeed applications.
- **Job Profile.** This is a data storage service that provides high throughput, low latency performance job meta data. It retrieves job attributes such as estimated salary, job titles, and job locations at serving time. The job profile uses Indeed NLP libraries and machine learning technologies to extract or aggregate user attributes.
- **User Profile.** Similar to the job profile, but it provides attributes about the job seeker rather than the job. Like the job profile, it is a data storage service that provides high throughput, low latency performance. It retrieves job seeker attributes such as expected salary, current job title, and preferred job locations at serving time. Like the job profile, it also uses Indeed NLP libraries and machine learning technologies to extract or aggregate user attributes.
- **Offline Evaluation Platform.** It consumes historic data to evaluate rule effectiveness without actually integrating with the upstream applications. It is also heavily used for fine-tuning existing rules, identifying new rules, and validating new rules and new models.
- **Offline Model Training.** This component consists of our offline training algorithms, with which we train models that can be used in the jobs filter rules at serving time for evaluation.

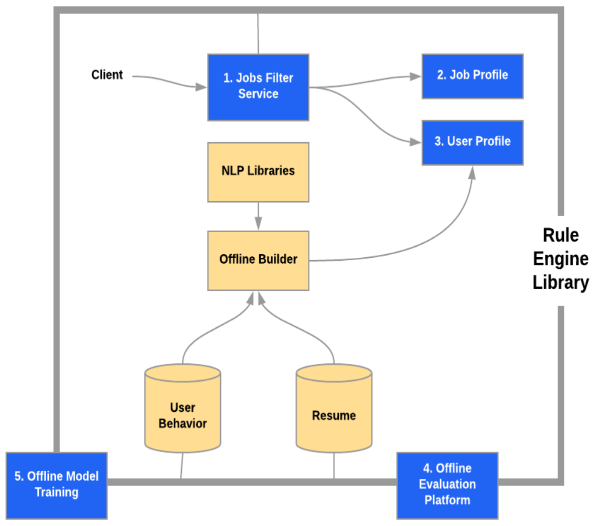

Figure 1: The system architecture of our jobs filter to improve the job seeker experience.

The jobs filter uses a set of rules to improve the quality of jobs displayed to any given job seeker. Rules can be simple: "Do not show a job requiring professional licenses to job seekers who don't possess such licenses," or "Do not show jobs to a job seeker if they come with a significant pay cut." They can also be complex: "Do not show jobs to the job seeker if we are confident the job seeker will not be interested in the job titles," or "Do not show jobs to the job seeker if our complex predictive models suggest the job seeker will not be interested in them."

All rules are compiled into a decision engine library. We share this library in our **Jobs Filter Service** and **Offline Evaluation Platform**, so that they are completely in sync for cross-validation, performance evaluation.

4 FILTER RULES TO IMPROVE JOB MATCH QUALITY

Although the underlying data for building jobs filter rules might be complex to acquire, most of the heuristic rules themselves are straightforward to design and implement. These rules are very effective at capturing edge cases to avoid degraded user experience.

To improve efficiency, we also extensively used machine learning, especially deep learning when designing the rules. In this section, we present two deep learning based rules to illustrate our design and implementation.

4.1 Inferred Title Similarity

In this rule, we build title transition embedding and use that to decide if a job should be shown to the job seeker.

The purpose of title transition embedding is to convert titles into a sparse vector in the space of all normalized titles, in a way that the transition relation between normalized tiles can be captured. We can then use it for similarity calculations. A variety of graph embedding algorithms exists for computing title transition embedding [5, 9, 13], in addition to our previous work [7].

Here we present another algorithm to compute title transition embedding, as illustrated in Algorithm 1. Unlike graph embedding models, this algorithm simply takes the distribution of the first and second order transitions of a given normalized title as its representation. Its advantage lies in the interpretability of the resulting embedding vectors, which facilitates troubleshooting in production environment. The modeling result generated from this algorithm is then loaded into **Jobs Filter Service** , which can be retrieved in real-time with extremely low latency.

Algorithm 1: Title transition embedding generation used in the jobs filter.

```
Result: The sparse representation of each normalized title
        transition in the space of all normalized titles.
1) Collect aggregation of most recent job titles and the
   second recent job titles from the resume database. Place a
   threshold $\tau$ on the counts to exclude sparse entries. ;
2) For each current title, group all previous titles of the job
   seekers together along with the number of instances. ;
3) Normalize the counts by dividing them by total
   occurrence of each current title. ;
4) Filter out entries with weights that are too low to mitigate
   false signals ;
```

We weight the job seeker's resume title and titles from their clicks and applies and treat them as the job seeker's inferred titles. Then we use Weighted Jaccard Similarity to decide if the job match is high quality. Let $v1$ and $v2$ be the embedding vectors of two normalized titles, $v1 \cap v2$ be the set of normalized titles corresponding to non-zeros entries in both $v1$ and $v2$, $v1 \cup v2$ be the set of normalized titles corresponding to non-zeros entries in $v1$ or $v2$. Then the similarity is computed as:

$$Sim(v1, v2) = \frac{\sum_{i \in v1 \cap v2} (\min(v_{1,i}, v_{2,i})/f_i)}{\sum_{j \in v1 \cup v2} (\max(v_{1,i}, v_{2,i})/f_j)} \tag{1}$$

where f_i and f_j are the frequencies of the corresponding normalized title. The procedure of making job veto decision using this similarity score is shown in Algorithm 2.

Algorithm 2: The Weighted Jaccard Similarity used in Inferred Title Similarity to allow or veto the job to the job seeker match.

```
Result: Allow or Veto the job to the job seeker
vector_job_titles = get_vector(job_title) ;
while user_inferred_title in user_inferred_titles do
    vector_user_inferred_title =
      get_vector(user_inferred_title) ;
    max_similarity_score = Max(max_similarity_score,
      compute_similarity_(vector_job_title,
      vector_user_inferred_title)) ;
    if max_similarity_score > $\tau$ then
        Allow the pair ;
    end
    else
        Veto the pair;
    end
end
```

4.2 User Response Prediction Model

In this rule, we use a user response prediction model to filter out jobs that the job seeker is less likely to be interested in.

In online advertising and recommender systems, user response prediction models are usually used to predict the probability that users respond positively to certain items by clicking on them or performing other actions. These probability scores are then used to rank all candidate items to generate top-K choices [1]. A common practice is to set a threshold to filter out those items with low scores. Although the models themselves are trained to predict positive reactions from users, the assumption here is that the low scores generated by the models also indicate poor match quality, which is usually true for properly-trained models.

When designing our machine learning based rules we adopted a slightly different approach: we combine negative response prediction and positive response prediction together to get predictions of higher quality. For our email job recommendation use case, if the job pushed to a user is relevant, the user may apply to the job. If the job is not relevant, the user may ignore this recommendation or reject it. In some cases, the user could even unsubscribe the service after seeing the recommended job, this could be a strong indicator of low quality recommendations. We log the user responses and use them as progressively stronger negative signals to train our models.

We build deep neural network models, including AutoInt [11], DeepFM [6], Wide&Deep [3] and multi-task FwFM [10] to predict users responses. When building these models, the features set we used covers user information, job information and some context information. Both the categorical features and the numerical features are converted to embedding vectors before they are fed to deep and interaction components.

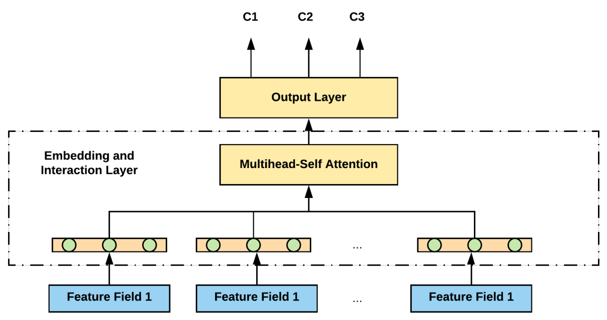

Figure 2: Label encoding in AutoInt for ordinal regression.

Since we have 4 different types of user reactions (labels): unsubscribe, reject, no response and apply, we want to be able to use the full information they carry when training the models. A straightforward approach is to use a multi-head structure and treat the prediction of each type of user action as a separate tasks. However this does not take the relation between these labels into consideration. To address this problem, we combine the label encoding trick in ordinal regression [4] with the deep models to capture the relation between all the labels. Taking AutoInt model as example, the structure of the network is shown in Figure 2. The embedding and interaction layers are largely unchanged except that in the output layer we produce multiple binary outputs (e.g. $C_i, i \in \{i, 2, 3\}$ in Figure 2). These binary outputs are used to encode all the labels.

In Table 1 we compare the offline performance of three different model configurations: ordinal regression via label encoding, multi-class classification, and binary classification, all implemented with AutoInt model. The numbers reported are all relative to a baseline DeepFM [6] model trained on binary label (reject and unsubscribe as 1, 0 otherwise). It can be observed that the multi-class version does not perform well, while the ordinal regression model is slightly better than the model trained with binary labels.

After we train a model that performs well, we export it using the Tensorflow SimpleSave API. We load the exported model into our online systems and serve requests using the Tensorflow Java API. Besides traditional classifier metrics such as AUC, precision, and recall, we also load our model into our **Offline Evaluation Platform** to validate the performance.

Model	AUC Improvement
Ordinal regression + AutoInt	0.9%
Multi-class AutoInt	-1.57%
Binary label AutoInt	0.3%

Table 1: Offline evaluation results, comparing with a baseline DeepFM model.

5 PERFORMANCE EVALUATION

In this section, we present the performance evaluation results of the two rules we have described above. The results are achieved through online A/B testing with different applications as well as offline evaluation with historic data through our **Offline Evaluation Platform**. The applications we used for evaluation include content-based recommendations (an application that recommends jobs that are similar to the jobs the job seeker has clicked on and applied), user-based recommendations (an application that recommends jobs that other similar job seekers are interested in) and email invite recommendations (an email application that identifies would-be interested candidates for employers).

An Indeed proprietary metric is used to evaluate our performance by measuring the match quality of the job seeker and the given jobs, which is defined as M in this work. Within Indeed, we sometimes look at M over seen events and M over impression events when evaluating matching quality of similar kinds.

From Table 2 we can see using Inferred Title Similarity rule alone can improve the M over impression events by 11.61%, 11.48% and 4.3% for the three applications we show.

Application	M/Impression Improvement
Content-based Recommendation	11.61%
User-based Recommendation	11.48%
Email Recommendation	4.3%

Table 2: Performance evaluation result by using the Inferred Title Similarity filter rule alone.

Further, using Response Prediction rule alone on the Email invitation application shows it can improve M over impression events by 11.35%, M over seen events by 18.74%, as shown in Table 3.

Finally, it is worth mentioning that the overall metrics improvements using the Jobs Filter product are 24.57% and 39.29%, also shown in Table 3.

Metrics	Response Prediction Rule	Jobs Filter
M/Impression Improvement	24.57%	11.35%
M/Seen Improvement	39.29%	18.74%

Table 3: Performance evaluation result by using the Response Prediction Model rule for the Email invitation application vs that by using the whole jobs filter product.

6 OPERATIONAL EXPERIENCE AND LESSONS LEARNED

Rule-based engines work well in solving corner cases. It is especially great at removing false negatives and false positives. However, the number of rules can easily spiral out of control. A hybrid approach of deep learning and a rule-based engine can effectively give us the best of both worlds. Our design's hierarchy of rules and machine learning technologies effectively solve this challenge and keep our system working efficiently.

Another experience is the **Offline Evaluation Platform** can be very powerful on identifying new rules and enhancing the existing rules to continue to improve overall effectiveness of the jobs filter. Rules backtested by the **Offline Evaluation Platform** barely needs online A/B tests when being launched, which hugely decreased development cycle.

In the future, we are also planning to use multihead model to predict all user responses together (multi-task learning). This should lead to more robust feature representation, and can further improve prediction performance.

Our system currently only gives a yes or no answer based on the matching quality. In the near future, we are planning to expose our matching quality as scores, so that the upstream applications can have more flexibility on how to make decisions based on the evaluation results.

7 CONCLUSIONS

Jobs Filter was born due to spiraling business applications to serve job seekers of Indeed.com and in the meantime we want to maintain a universal top user experience. We have applied our jobs filter in several applications within Indeed, which significantly increases user experience across the board. More specifically, it also works especially well on the corner cases, in which cases many job seekers actually used to complain about.

In this work, we have shared our engineering practice on building such as production quality jobs filter and have described in detail the system design and algorithms of two of the rules. We have also shown our performance results to prove the effectiveness and efficiency of the engineering achievements.

REFERENCES

[1] Charu C. Aggarwal. 2016. *Recommender Systems: The Textbook* (1st ed.). Springer Publishing Company, Incorporated.

[2] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. *CoRR* abs/1606.07792 (2016). arXiv:1606.07792 http://arxiv.org/abs/1606.07792

[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Gregory S. Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide Deep Learning for Recommender Systems. In *DLRS 2016*.

[4] Jianlin Cheng, Zheng Wang, and Gianluca Pollastri. 2008. A neural network approach to ordinal regression. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (2008), 1279–1284.

[5] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* (2018). https://doi.org/10.1016/j.knosys.2018.03.022

[6] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. *ArXiv* abs/1703.04247 (2017).

[7] Haiyan Luo, Shichuan Ma, Anand Joseph Bernard Selvaraj, and Yu Sun. 2019. Learning Job Representation Using Directed Graph Embedding. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data* (Anchorage, Alaska) *(DLP-KDD '19)*. Association for Computing Machinery, New York, NY, USA, Article 14, 5 pages. https://doi.org/10.1145/3326937.3341263

[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *CoRR* abs/1310.4546 (2013). arXiv:1310.4546 http://arxiv.org/abs/1310.4546

[9] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*. ACM Press, 1105–1114. https://doi.org/10.1145/2939672.2939751

[10] Junwei Pan, Yizhi Mao, Alfonso Lobos Ruiz, Yu Sun, and A. Flores. 2019. Predicting Different Types of Conversions with Multi-Task Learning in Online Advertising. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining* (2019).

[11] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (2019).

[12] Julio Villena-Román, Sonia Collada-Pérez, Sara Lana-Serrano, and José Carlos González. 2011. Hybrid Approach Combining Machine Learning and a Rule-Based Expert System for Text Categorization. In *FLAIRS Conference*.

[13] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable Graph Embedding for Asymmetric Proximity. *AAAI* (2017), 2942–2948.