LiJAR: A System for Job Application Redistribution towards Efficient Career Marketplace

Fedor Borisyuk
LinkedIn Corporation, USA
fborisyuk@linkedin.com

Liang Zhang
LinkedIn Corporation, USA
lizhang@linkedin.com

Krishnaram Kenthapadi
LinkedIn Corporation, USA
kkenthapadi@linkedin.com

ABSTRACT

Online professional social networks such as LinkedIn serve as a marketplace, wherein job seekers can find right career opportunities and job providers can reach out to potential candidates. LinkedIn's job recommendations product is a key vehicle for efficient matching between potential candidates and job postings. However, we have observed in practice that a subset of job postings receive too many applications (due to several reasons such as the popularity of the company, nature of the job, etc.), while some other job postings receive too few applications. Both cases can result in job poster dissatisfaction and may lead to discontinuation of the associated job posting contracts. At the same time, if too many job seekers compete for the same job posting, each job seeker's chance of getting this job will be reduced. In the long term, this reduces the chance of users finding jobs that they really like on the site. Therefore, it becomes beneficial for the job recommendation system to consider values provided to both job seekers as well as job posters in the marketplace.

In this paper, we propose the job application redistribution problem, with the goal of ensuring that job postings do not receive too many or too few applications, while still providing job recommendations to users with *the same level* of relevance. We present a dynamic forecasting model to estimate the expected number of applications at the job expiration date, and algorithms to either promote or penalize jobs based on the output of the forecasting model. We also describe the system design and architecture for *LiJAR*, LinkedIn's Job Applications Forecasting and Redistribution system, which we have implemented and deployed in production. We perform extensive evaluation of *LiJAR* through both offline and online A/B testing experiments. Our production deployment of this system as part of LinkedIn's job recommendation engine has resulted in significant increase in the engagement of users for underserved jobs (6.5%) without affecting the user engagement in terms of the total number of job applications, thereby addressing the needs of job seekers as well as job providers simultaneously.

CCS CONCEPTS

• Information systems → Recommender systems; • Applied computing → Forecasting;

KEYWORDS

Job application forecasting; Job application redistribution; Personalized search and recommendation systems

ACM Reference format:
Fedor Borisyuk, Liang Zhang, and Krishnaram Kenthapadi. 2017. LiJAR: A System for Job Application Redistribution towards Efficient Career Marketplace. In *Proceedings of KDD '17, Halifax, NS, Canada, August 13-17, 2017,* 10 pages.
https://doi.org/10.1145/3097983.3098028

1 INTRODUCTION

Online professional social networks such as LinkedIn play a key role in helping job seekers find right career opportunities and job providers reach out to potential candidates. LinkedIn's job ecosystem has been designed to serve as a marketplace for efficient matching between potential candidates and job postings, and to provide tools to connect job seekers and job providers. LinkedIn's job recommendations product is a crucial mechanism to help achieve these goals.

The problem of recommending jobs to users is fundamentally different from traditional recommendation system problems such as recommending books, products, or movies to users. While all of the above have a common objective to maximize the engagement rate of the users, one key difference is that a job posting is typically meant to hire one or a few employees only, whereas the same book, product, or movie could be potentially recommended to hundreds of thousands of users for consumption. As LinkedIn's job posters pay to post jobs on the site, on the one hand it is critical to deliver sufficient number of applications from qualified candidates for each job, so that the job poster can interview the top candidates and finally hire one or a few of them. On the other hand, it is not desirable for the system to deliver too many applications for any posted job with one or a few openings, as the amount of effort for the job poster to interview would become much greater than expected. Also, if too many job seekers compete for the same job posting, each job seeker's chances of getting that job will be dramatically reduced. Historically, since our job recommendation system at LinkedIn used to only optimize for user engagement, we have noticed that there are jobs that receive too few applications at expiration, and ones with too many applications as well. As these often lead to job poster complaints and the discontinuation of the job posting contracts, it hurts the liquidity of the jobs in the marketplace, and in the long term in turn reduces the chances that a user can find the jobs that she really likes on the site.

Therefore, we note that an ideal job recommendation system would need to achieve three goals simultaneously: (1) Recommend the most relevant jobs to users. (2) Ensure that each job posting receives sufficient number of applications from qualified candidates.

(3) Ensure that each job posting does not receive too many applications. All the three goals above are meant to link users with the best career opportunities that excite them, with the first goal mostly focusing on the short term user engagement optimization, and the other two trying to address the health of the marketplace for the site’s long-term growth.

Intuitively, these goals can be achieved by assigning jobs to users so as to maximize the likelihood that the user will *apply* for the job, *pass* the interview, and also *take* the job offer. Maximizing such objective is desirable from the perspective of job seekers as well as job providers, since it takes into account both the competition from all the applicants for a job as well as the interest of the users, and is likely to lead to better assignment of jobs to users for a healthy ecosystem. However, this approach is usually infeasible to take, since the data of whether a job applicant passed the interview is often highly confidential, and understanding why a user takes one job offer over another would need a lot of confidential data and study as well. Hence, we instead adopt a pragmatic approach towards achieving the three goals above.

In this paper, we propose the job application redistribution problem, with the goal of ensuring that job postings do not receive too many or too few applications, while providing job recommendations to users with *the same level* of relevance as the system that purely optimizes for user engagement (§2). We present a dynamic forecasting model to estimate the expected number of applications at the job expiration date (§3), and algorithms to either promote or penalize jobs based on the output of the forecasting model in real time (§4). We then describe the design and architecture for *LiJAR*, LinkedIn’s Job Applications Forecasting and Redistribution system, which we have implemented and deployed in production (§5). We perform extensive evaluation of the proposed job forecasting and boosting models (§6). Our online A/B testing experiments demonstrate the effectiveness of our approach in increasing the engagement of users for underserved jobs by 6.5%, while keeping the user engagement in terms of the total number of job applications the same. We finally discuss related work (§7) and future work (§8).

2 PROBLEM SETTING

As the world’s largest professional social network, LinkedIn provides plenty of great opportunities for professionals to find new career opportunities, and also sends good candidates to companies who paid LinkedIn for hiring for their jobs. One of LinkedIn’s most important products is the Jobs Homepage (see Figure 1), which serves as a one-stop shop for our users who are interested in finding jobs to apply for. On the homepage, a key module is called “Jobs you may be interested in”, where we show relevant jobs to users according to the match between user profile and job descriptions, and user’s past activities as well. Every time a user visits the jobs homepage, the job recommendation system picks the “best” set of jobs from the candidate pool [5], and displays them for the user to apply for. The jobs shown are mostly posted by LinkedIn’s customers who paid a fixed price ahead of time in return for their job postings to be eligible for inclusion in the job recommendations. Thus, this module is responsible for both ensuring a good user experience and distributing applications from users to jobs such that LinkedIn’s customers can be satisfied as well.

Historically, the job recommendation problem has been mostly solved by considering it as a traditional recommendation system

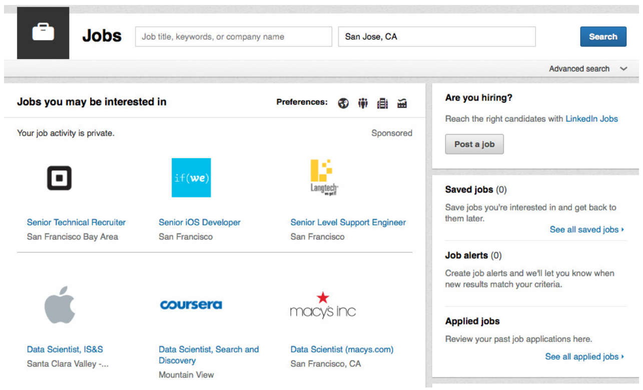

Jobs
Job title, keywords, or company name
San Jose, CA
Search
Advanced search
Jobs you may be interested in
Preferences:
Your job activity is private.
Sponsored
Senior Technical Recruiter
San Francisco Bay Area
Senior iOS Developer
San Francisco
Senior Level Support Engineer
San Francisco
Data Scientist, IS&S
Santa Clara Valley -...
Data Scientist, Search and Discovery
Mountain View
Data Scientist (macys.com)
San Francisco, CA
Are you hiring?
Reach the right candidates with LinkedIn Jobs
Post a job
Saved jobs (0)
Save jobs you're interested in and get back to them later.
See all saved jobs ›
Job alerts (0)
Create job alerts and we'll let you know when new results match your criteria.
Applied jobs
Review your past job applications here.
See all applied jobs ›

Figure 1: A snapshot of the LinkedIn jobs homepage.

problem, where the objective is to rank the jobs corresponding to the probability of clicks or applications for each user visit. An example of such models is called *GLMix* [22], where a generalized linear mixed model is used to predict the binary response of apply or no apply given an impression from a user to a job. The model not only contains global features such as interactions between member features (e.g., title, industry, work history, skills) and job features (e.g., title, company, qualifications), but also includes a per-member model component for member-specific preferences to job features, and a per-job model component to model what types of members like to apply for a given job. The success of the model was measured by A/B testing using metrics such as the total number of applications, which corresponds to one of our three goals outlined in §1: *recommend the most relevant jobs to users*. However, it does not consider at all the other two objectives that require *each job posting to not receive either too many or too few applications at expiration*.

To justify why the number of applications per job at expiration is a good measure for our customer satisfaction, we studied the relationship between the probability of at least one hire from the pool of job applicants versus the total number of applications a job receives, and found that this probability increases with the number of applications, but with a non-linear relationship (see Figure 2): the incremental value an additional application provides to the hiring probability diminishes exponentially as the job receives more and more applications. In the meantime, it is clear that the amount of effort to review the applications and interview the candidates grows linearly with the number of applications. Therefore, on the one hand, if a job receives very few applications, most likely the job posting company cannot find anyone to hire; hence they may become dissatisfied and not renew the job posting contract. On the other hand, if a job receives too many applications (say, 1000 for one position), it becomes very difficult for the job posting company to interview all the candidates; hence the job posters may also complain. It decreases the value that we are providing to our users as well, since too many applications essentially make the job offer to become too competitive to obtain from the perspective of job seekers.

Our idea to solve the problem is motivated by the concept of “early intervention”. At any time, assuming that an oracle could tell us which jobs are likely to receive too few or too many applications at expiration, we can then intervene early to boost or penalize the scores from the GLMix ranking model for such jobs to achieve a good trade-off between short-term user engagement and our customer satisfaction for the long-term marketplace health. For the jobs that are expected to receive a reasonable range of the

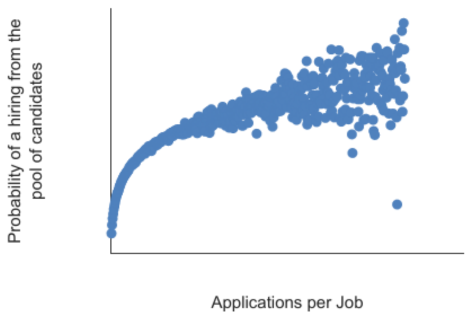


Figure 2: Probability of finding a hire among the set of candidates who applied for the job. The data for figure was estimated using LinkedIn's marketplace statistics.

number of applications, it is better not to intervene at all and the ranking of these jobs would be purely driven by relevance. Hence, the success of this approach would heavily depend on whether we can accurately forecast the number of applications a job would receive at expiration. Our problem can thus be stated as follows: *How do we forecast the number of applications that a job is likely to receive at expiration (along with a confidence interval), given the data about this job observed so far? How do we either promote or penalize the job based on the forecast?* We describe our solutions in §3 and §4 respectively.

3 JOB APPLICATIONS FORECASTING MODEL

We next describe the forecasting model for the number of applications a job would receive at its expiration time. We consider three types of signals: (1) Seasonality. Since a job usually takes 30 days to expire, the number of impressions and applications depends heavily on the time of the day and the day of week. We also noticed that new jobs tend to receive more impressions and applications than old jobs (See Figure 4). (2) Job attributes, including title, company, industry, qualifications, and so forth. (3) The number of impressions and applications the job has received so far. In the following text, we describe the details of our forecasting model.

Without loss of generality, assume that the timestamp is 0 when a job gets posted and ticks to T when the job expires. We describe our model to forecast the number of applications a job would receive when it expires at time T, based on the data we collected for the job from time 0 to t ($t < T$), seasonality features and job attributes. The objective is to provide the confidence interval of the number of applications at time T, so that we can penalize or boost the job score in ranking when we "strongly believe" that a job is going to be under-delivered or over-delivered without intervention.

For any job j, let c_{jt} be the number of applications received *at* time t, and v_{jt} be the number of impressions *at* time t. Let $c_{j,1:t} = \sum_{i=1}^{t} c_{ji}$, and $v_{j,1:t} = \sum_{i=1}^{t} v_{ji}$ be the total number of applications and impressions job j has received *up to* time t respectively. Also denote the feature vector for job j at time t as $\boldsymbol{x}_{jt}$ which includes both the seasonality features as well as all the attributes of this job (e.g., job title, company, industry). Although $\boldsymbol{x}_{jt}$ depends on time t, in this paper we assume $\boldsymbol{x}_{jt}$ is always known, since the seasonality features are mostly day of the week, time of the day and so forth; and the attributes of the job are static, i.e. not dependent on time.

At time t, our main task is to obtain the distribution of $c_{j,(t+1):T}$, based on the observed $c_{j,1:t}$ and $v_{j,1:t}$, along with the feature vector $\boldsymbol{x}_{jt}$. Since $c_{j,1:T} = c_{j,1:t} + c_{j,(t+1):T}$, the distribution of $c_{j,1:T}$ is simply a mean shift of the distribution of $c_{j,(t+1):T}$ with the observed $c_{j,1:t}$, i.e., treating the observed $c_{j,1:t}$ as a constant,

$$E[c_{j,1:T}|c_{j,1:t}] = c_{j,1:t} + E[c_{j,(t+1):T}], \tag{1}$$

$$Var[c_{j,1:T}|c_{j,1:t}] = Var[c_{j,(t+1):T}]. \tag{2}$$

Assume that the CTR estimate of job j at time t is μ_{jt}. Our idea is to forecast $c_{j,(t+1):T}$ by $\mu_{jt} v_{j,(t+1):T}$, where both μ_{jt} and $v_{j,(t+1):T}$ have a statistical model. Once the distributions of μ_{jt} and $v_{j,(t+1):T}$ are available, the distribution of $c_{j,(t+1):T}$ can be estimated accordingly.

Gamma Distribution. We denote a Gamma distribution as $x \sim Ga(m, s)$ with the associated density function

$$p(x) = \frac{s^{ms}}{\Gamma(ms)} x^{ms-1} e^{-sx}, \tag{3}$$

where m is the mean of the distribution, and s is called *sample size*. The variance of x is hence m/s.

3.1 Per-job CTR Estimate μ_{jt}

We consider a dynamic Gamma-Poisson model following [2] for the estimated CTR μ_{jt} for job j at time t. Specifically, given a Gamma prior of μ_{jt} as $Ga(\alpha_{jt}, s_{jt})$, where α_{jt} is the prior mean and s_{jt} is the prior sample size, we assume

$$c_{jt}|v_{jt}, \mu_{jt} \sim Poisson(v_{jt}\mu_{jt}), \tag{4}$$

The posterior distribution of μ_{jt} is then

$$\mu_{jt}|v_{jt}, c_{jt} \sim Ga(\frac{c_{jt} + s_{jt}\alpha_{jt}}{v_{jt} + s_{jt}}, v_{jt} + s_{jt}). \tag{5}$$

At time $t+1$, the prior of $\mu_{j,t+1}$ becomes $Ga(\alpha_{j,t+1}, s_{j,t+1})$, where

$$\alpha_{j,t+1} = \frac{c_{jt} + s_{jt}\alpha_{jt}}{v_{jt} + s_{jt}}, \tag{6}$$

$$s_{j,t+1} = \delta(v_{jt} + s_{jt}), \tag{7}$$

where $\delta \in (0, 1]$ is called the variance discounting parameter which controls how much the CTR changes over time, as the prior variance of $\mu_{j,t+1}$ is proportional to $1/\delta$. When $\delta = 1$, the applications and impressions observed from time 1 to t are equally weighted to obtain the CTR estimate at time t. When $\delta < 1$, the data observed closer to time t gets exponentially bigger weight than the data observed earlier.

At time 0, i.e., when there is no data observed for job j, we can start with letting α_{j1} be the global CTR and s_{j1} be a tuning parameter (e.g., 5 or 10).

3.2 Forecasting Model of $v_{j,(t+1):T}$

For any job j, given its number of impressions we observed up to time t, i.e., $v_{j,1:t}$, we would like to predict its number of impressions from time $t+1$ to T. Note that the number of impressions usually has a strong pattern of seasonality, and the job attributes such as title, job function, company, and industry can impact the expected number of impressions significantly. At time t, we hence use the feature set $\boldsymbol{x}_{jt}$ which includes both the seasonality features as well as the job-specific attributes to first build a baseline Poisson model as

$$v_{jt} \sim Poisson(\exp(\boldsymbol{x}'_{jt}\boldsymbol{\beta})), \tag{8}$$

where $\boldsymbol{\beta}$ is the coefficient vector to be learned. Let $\hat{\boldsymbol{\beta}}$ be the Maximum log-likelihood estimate. Then, $\exp(\boldsymbol{x}'_{jt}\hat{\boldsymbol{\beta}})$ essentially provides the expected number of impressions job j will receive at time t, **without** considering the job's past history of impressions from time 0 to $t-1$. To be able to incorporate such past data into consideration, we thus introduce an additional adjustment parameter λ_j for each job j, i.e.,

$$v_{jt} \sim Poisson(\lambda_j \exp(\boldsymbol{x}'_{jt}\boldsymbol{\beta})), \tag{9}$$

where λ_j has a prior

$$\lambda_j \sim Ga(1, r), \tag{10}$$

with r being the prior sample size that can be a tuning parameter. Given the observed number of impressions for job j from time 1 to t, i.e., $v_{j,1:t}$, and also the fitted coefficient vector $\hat{\boldsymbol{\beta}}$, the posterior of λ_j at time t then becomes

$$\lambda_j | v_{j,1:t} \sim Ga(\frac{r + v_{j,1:t}}{r + \sum_{i=1}^{t} \exp(\boldsymbol{x}'_{ji}\hat{\boldsymbol{\beta}})}, r + \sum_{i=1}^{t} \exp(\boldsymbol{x}'_{ji}\hat{\boldsymbol{\beta}})). \tag{11}$$

Intuitively, λ_j can be thought of as a multiplicative factor, following Gamma distribution with prior mean of 1. By taking into account the observed number of impressions for job j till time t, we obtain the posterior Gamma distribution for λ_j, wherein the posterior mean "corrects" for the observed number of impressions compared to the expected number of impressions: the posterior mean increases with the ratio of the observed number of impressions to the expected number of impressions, with the prior sample size r acting as a smoothing parameter. In this manner, we obtain the forecasting model for $v_{j,(t+1):T}$ that takes into account both the baseline feature set based prediction model and the past history of impressions. Since

$$v_{j,(t+1):T} | \lambda_j \sim Poisson(\lambda_j \sum_{i=t+1}^{T} \exp(\boldsymbol{x}'_{ji}\hat{\boldsymbol{\beta}})), \tag{12}$$

by integrating out λ_j, the marginal distribution of $v_{j,(t+1):T}$ given $v_{j,1:t}$ becomes

$$\begin{aligned} p(v_{j,(t+1):T} | v_{j,1:t}) = & \int p(v_{j,(t+1):T} | \lambda_j) p(\lambda_j | v_{j,1:t}) d\lambda_j \\ \sim \quad & NB(v_{j,1:t} + r, \frac{\sum_{i=t+1}^{T} \exp(\boldsymbol{x}'_{ji}\hat{\boldsymbol{\beta}})}{\sum_{i=1}^{T} \exp(\boldsymbol{x}'_{ji}\hat{\boldsymbol{\beta}}) + r}), \end{aligned} \tag{13}$$

where $NB(\gamma, p)$ is a negative binomial distribution with density

$$p(x|\gamma, p) = \frac{\Gamma(x+\gamma)}{\Gamma(\gamma)\Gamma(x+1)} (1-p)^{\gamma} p^{x}. \tag{14}$$

To give some intuition, we note that

$$E[v_{j,(t+1):T} | v_{j,1:t}] = \frac{\sum_{i=t+1}^{T} \exp(\boldsymbol{x}'_{ji}\hat{\boldsymbol{\beta}})}{\sum_{i=1}^{t} \exp(\boldsymbol{x}'_{ji}\hat{\boldsymbol{\beta}}) + r} (v_{j,1:t} + r), \tag{15}$$

$$Var[v_{j,(t+1):T} | v_{j,1:t}] = \frac{(\sum_{i=t+1}^{T} \exp(\boldsymbol{x}'_{ji}\hat{\boldsymbol{\beta}}))(\sum_{i=1}^{T} \exp(\boldsymbol{x}'_{ji}\hat{\boldsymbol{\beta}}) + r)}{(\sum_{i=1}^{t} \exp(\boldsymbol{x}'_{ji}\hat{\boldsymbol{\beta}}) + r)^2} * (v_{j,1:t} + r). \tag{16}$$

Suppose that the prior sample size, $r = 0$. Then, $E[v_{j,(t+1):T} | v_{j,1:t}]$ becomes a simple projection of the number of impressions observed up to time t, with a ratio of the expected number of impressions from $t+1$ to T and that from time 1 to t with the baseline model.

3.3 Forecasting Model of $c_{j,(t+1):T}$

Since we predict $c_{j,(t+1):T}$ by $\mu_{jt} v_{j,(t+1):T}$ with μ_{jt} having a Gamma distribution from (5), and $v_{j,(t+1):T}$ having a Negative Binomial distribution from (13), the distribution of $c_{j,(t+1):T}$ does not have a closed form based on our knowledge.

Ideally the distribution of $c_{j,(t+1):T}$ can be obtained empirically through Monte-Carlo sampling, i.e., we obtain K random samples from posteriors of μ_{jt} and $v_{j,(t+1):T}$, and multiply each pair of such samples to obtain a sample of $c_{j,(t+1):T}$. However, since the forecasting needs to happen online and Monte-Carlo sampling might be too expensive, we instead take some approximations to obtain $E[c_{j,(t+1):T}]$ and $Var[c_{j,(t+1):T}]$ as follows, assuming μ_{jt} and $v_{j,(t+1):T}$ are independent random variables:

$$E[c_{j,(t+1):T}] = E[\mu_{jt} v_{j,(t+1):T}] = E[\mu_{jt}] E[v_{j,(t+1):T}], \tag{17}$$

$$\begin{aligned} Var[c_{j,(t+1):T}] &= Var[\mu_{jt} v_{j,(t+1):T}] \\ &= E[\mu_{jt}^2] E[v_{j,(t+1):T}^2] - (E[\mu_{jt}] E[v_{j,(t+1):T}])^2 \\ &= (Var[\mu_{jt}] + E[\mu_{jt}]^2)(Var[v_{j,(t+1):T}] + \\ & \quad E[v_{j,(t+1):T}]^2) - (E[\mu_{jt}] E[v_{j,(t+1):T}])^2 \end{aligned} \tag{18}$$

The confidence intervals (CI) of $c_{j,(t+1):T}$ can thus be approximately obtained as

$$\begin{aligned} CI(c_{j,(t+1):T}) = [&E[c_{j,(t+1):T}] - \eta\sqrt{Var[c_{j,(t+1):T}]}, \\ &E[c_{j,(t+1):T}] + \eta\sqrt{Var[c_{j,(t+1):T}]}], \end{aligned} \tag{19}$$

where $\eta = 1.96$ for 95% confidence interval. Using Equations 1 and 2, we obtain the corresponding confidence interval of $c_{j,1:T}$ given $c_{j,1:t}$ as

$$\begin{aligned} CI(c_{j,1:T} | c_{j,1:t}) = [&c_{j,1:t} + E[c_{j,(t+1):T}] - \eta\sqrt{Var[c_{j,(t+1):T}]}, \\ &c_{j,1:t} + E[c_{j,(t+1):T}] + \eta\sqrt{Var[c_{j,(t+1):T}]}], \end{aligned} \tag{20}$$

3.4 Discussions

Feature-based CTR model for μ_{jt}. We note that the CTR model for μ_{jt} can be extended to a feature-based model as well if necessary. In this case we let

$$c_{jt} | v_{jt}, \mu_{jt} \sim Poisson(v_{jt} \mu_{jt} \exp(\boldsymbol{x}'_{jt}\boldsymbol{\gamma})), \tag{21}$$

where $\exp(\boldsymbol{x}'_{jt}\boldsymbol{\gamma})$ forms the baseline model to predict the CTR based on the seasonality and job attribute features. μ_{jt} still has the prior as $Ga(\alpha_{jt}, s_{jt})$ that comes from the posterior of μ_{jt} at time $t-1$, and at time 0, $\alpha_{j1} = 1$ instead of the global CTR. Similar to how we learned $\boldsymbol{\beta}$ in Section 3.2, $\boldsymbol{\gamma}$ can be estimated first from a baseline

Poisson model assuming all the $\mu_{jt} = 1$. Once $\boldsymbol{\gamma}$ is learned, we learn the posterior of μ_{jt} by plugging in the estimates of $\boldsymbol{\gamma}$.

Negative binomial model for $c_{j,(t+1):T}$ directly. It might be tempting to consider using the model presented in §3.2 to model $c_{j,(t+1):T}$ directly, instead of breaking $c_{j,(t+1):T}$ into the CTR μ_{jt} and number of impressions $v_{j,(t+1):T}$ and modeling the two separately. We note that this idea sacrifices the predictive accuracy by not being able to distinguish jobs with high CTR but low number of impressions versus those with high number of impressions but low CTR; having both number of applications as well as impressions considered in the model is always better than just number of applications alone, as the two are highly correlated.

4 JOB BOOSTING AND PENALIZATION

At time t, assume that the predicted confidence interval, $CI(c_{j,1:T}|c_{j,1:t})$ of the number of applications a job would receive at expiration time T is $[l_t, u_t]$, where l_t and u_t can be estimated via the job applications forecasting model specified in §3. As we would like a job to receive not too few or too many number of applications and simultaneously try to serve the most relevant jobs to users, we adopt a pragmatic and effective approach to intervene the ranking of jobs when either u_t is below the targeted minimum number of applications (denoted as *minApps*), or l_t is above the targeted maximum number of applications (denoted as *maxApps*).

As discussed in §2, we make use of a model to predict the probability of a user applying for a job given an impression [22], and use this probability score (in the range $[0, 1]$) to rank the candidate set of jobs to maximize the user engagement. Our idea is to boost the job's score when $u_t < minApps$, and penalize the score when $l_t > maxApps$, so that we can achieve a good trade-off between user engagement and the desired range of $[minApps, maxApps]$ for the number of applications at the expiration time for as many jobs as possible.

4.1 Penalization of the Job Score

At time t, if the lower bound of CI for the number of applications of a job, i.e., l_t, is projected to be more than targeted maximum number of applications, i.e., *maxApps*, we would like to stop showing the job to users unless it is really necessary (e.g., no other jobs are available with better or similar quality). Our approach is to multiply the score with an exponential decay penalization factor, based on the number of applications the job has received up to the current moment, i.e., when $l_t > maxApps$, we let

$$newScore = originalScore \cdot e^{-\frac{\#applications}{softness}}, \tag{22}$$

where *originalScore* is computed by the model to predict the probability of the user applying for the job, $\#applications$ is the number of applications a job has received up to now, and $softness$ is a tuning parameter of the exponential decay upon the number of applications. In §6, we show how the choice of the softness parameter influences our metrics (e.g., the total number of job applications received) through online experiments.

Our choice of the exponential decay function upon the number of applications a job has received is based on an intuition obtained from Figure 2, where the incremental value of probability of hiring given an additional application decreases exponentially upon the number of applications that a job has received.

4.2 Boosting of the Job Score

Similarly, at time t, if the upper bound of CI for the number of applications of a job, i.e., u_t, is projected to be less than the targeted minimum number of applications, i.e., *minApps*, we would like to boost the job score so that it can obtain more impressions from users, without significant degradation of the relevance of the job recommendations. Our approach is to boost the original model score by a multiplicative constant, which is a tuning parameter named *boostFactor*, provided the original score is greater than a certain threshold h. In other words, when $u_t < minApps$ and *originalScore* $> h$, we let

$$newScore = \min\{originalScore \cdot boostFactor, 1.0\} \tag{23}$$

We show how the choices of *boostFactor* and h impact our metrics such as the total number of applications through online experiments in §6. We also note that having the threshold $h > 0$ is critical as boosting a non-relevant job to a user would hurt the user engagement and even the reputation of the site heavily.

We chose a small constant multiplicative factor function (at most $\frac{1}{h}$) for boosting since excessive boosting can affect the relevance of jobs shown to users, and further we used the same factor across jobs for simplicity.

5 SYSTEM ARCHITECTURE

We now describe the overall architecture of *LiJAR*, the job application redistribution system at LinkedIn, focusing on the details of the job applications forecasting and job boosting modules that we implemented and deployed in production as part of LinkedIn's job recommendation engine. Our system can be subdivided into an online system for serving job recommendations and an offline workflow for updating the forecasting model regularly (see Figure 3). For the offline workflow, every day we take the most recent user-job interaction log data, and retrain our job applications forecasting model on Hadoop. The learned model parameters are then pushed to the forecasting model store implemented using Voldemort [20], which are accessed online by the job recommendation service tier.

We now provide a description of the overall flow for how a client request (query) is processed in our online job serving system. It makes use of a multi-tier service architecture, wherein the recommendation request queries are handled in a distributed fashion across hundreds of production servers. The online query processing proceeds as follows:

- Whenever a job recommendation needs to be shown to a user, the client application issues a query containing the userId and other metadata to the backend distributed job recommendation application service tier (step 1).
- The job recommendation application service then retrieves relevant structured user data from the user fields store (step 2), determines the appropriate A/B testing experimental treatment that the user falls under (step 3), and fetches the necessary ranking models based on the experimental treatment (step 4). Then, it creates a request object containing the user data and the models, which is issued to the distributed search service tier (step 5).
- Our search based retrieval system is built on top of LinkedIn's Galene search platform [19]. This system is responsible for applying the candidate selection model [5] to the user data

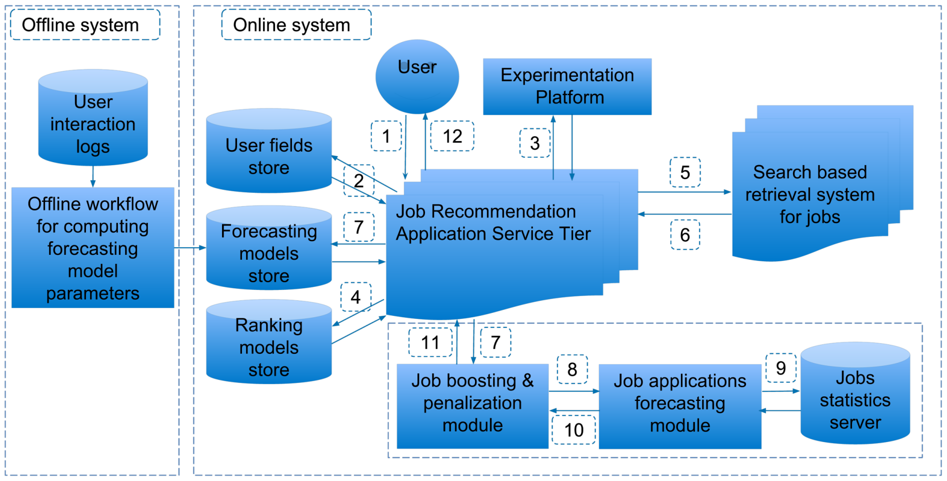

Offline system
User interaction logs
Offline workflow for computing forecasting model parameters
Online system
User
Experimentation Platform
User fields store
Forecasting models store
Ranking models store
Job Recommendation Application Service Tier
Search based retrieval system for jobs
Job boosting & penalization module
Job applications forecasting module
Jobs statistics server
1
2
3
4
5
6
7
7
8
9
10
11
12

Figure 3: Architecture of *LiJAR*, LinkedIn's Job Applications Forecasting and Redistribution system.

to generate a corresponding Galene query, issuing this query to the distributed search index service, scoring the resulting jobs using the second-pass GLMix ranking model [22], and returning back the top ranked job recommendation results (step 6).

- The job recommendation application service then performs post-processing (such as applying filters and certain business rules), and in particular, invokes the job boosting and penalization module (step 7; shown separately for clarity). For each specified jobId, the module needs to first obtain the (as-of-the-current-moment) expected number of applications by the job expiration date, and hence queries the job applications forecasting module (step 8), which in turn obtains the real-time job statistics (#impressions, #applications so far for the jobId) from the job statistics server (step 9). Based on the obtained statistics and the features of the associated job posting (e.g., job title, company, industry), the forecasting module computes and sends back the expected number of applications by the expiration date, along with the confidence interval as well as the current job statistics (step 10). The job boosting module uses this information to determine whether to intervene, if yes, either penalize or boost the score as the case may be, and returns back the modified score (step 11). The job recommendation service then returns the final ranked list of recommended jobIds to the requesting client application.

6 EXPERIMENTS

We present an extensive evaluation of our methodology for job application redistribution, through both offline and online A/B testing experiments on the job recommendation module ("Jobs You May Be Interested In") on LinkedIn jobs homepage. We first provide data insights on how the number of impressions and the number of applications correlate with the time after a job gets posted (§6.1). Then, we present an offline evaluation of the performance of our job application forecasting model as well as boosting and penalization strategy (§6.2). Finally, we demonstrate the performance of our job redistribution methodology in the production system through online A/B testing experiments (§6.3).

6.1 Data analysis

To help us better understand how the number of impressions and the number of applications correlate with the time after a job gets posted, we performed data analysis using the entire 2015 year's data collected from the job recommendation module. Figure 4 shows a normalized view of the number of impressions and applications a job receives on average for each day after a job gets posted, until 30 days which is usually when most of the jobs expire. Since users like to apply for new jobs, our job recommendation system tends to always show new jobs more often than old jobs. Hence, we observe that both the number of impressions and the number of applications a job receives decay exponentially over time after it gets posted. It is also very interesting to observe a weekly pattern in the graph as well, which is due to the fact that most of the jobs are posted on weekdays instead of weekends; hence even if we align the daily job impressions with the number of days elapsed after posting, the weekly pattern of impressions and applications can still be observed.

6.2 Offline Experiments

We next show the performance of our job application forecasting model (§3), using the entire year of 2015's data as training, and Sep-Dec 2016's data as test. We consider three different models with several variations of the feature set $\boldsymbol{x}_{jt}$ for the forecasting model of $v_{j,(t+1):T}$ and μ_{jt}, where the feature set $\boldsymbol{x}_{jt}$ is used to construct the baseline model for v_{jt} and μ_{jt} (See Equations (9) and (21)). The three models we consider are:

- IMP-WEEKLY: The forecasting model of v_{jt} with only the day of week features in $\boldsymbol{x}_{jt}$. No features are used for μ_{jt}, i.e., the model for μ_{jt} follows precisely as in §3.1.
- IMP-FULL: The forecasting model of v_{jt} with a full set of features in $\boldsymbol{x}_{jt}$, including the day of week, time elapsed after job posting, and job attributes such as title, industry, job function and so forth. No features are used for μ_{jt}.

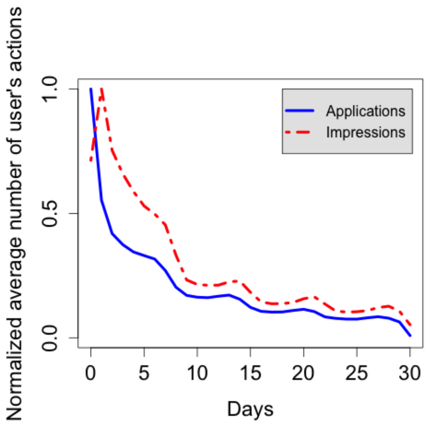


Figure 4: Normalized average number of applications a job receives daily after it gets posted (normalized by maximum number of actions). The dashed red line represents the average number of impressions a job receives daily after the posting. This data is obtained by averaging over all the jobs posted within the year 2015.

- IMP-CTR-FULL: The forecasting models of v_{jt} and of μ_{jt} use full set of features as used in IMP-FULL. The model for μ_{jt} is as described in §3.4.

In Figure 5(a), we show the RMSE performance of the above three models over 30 days. The experiment was performed as follows: For each job and each of the 30 days of the job's lifetime, we used the historical impressions and applications observed up to this day, and the feature set $\boldsymbol{x}_{jt}$ to obtain a forecasted expectation of the number of applications the job would receive at expiration (i.e., day 30), following Equation (17). The RMSE for each day was obtained by computing the square of the difference between the forecasted and the observed value (at expiration) of the number of applications for each job, averaging over all jobs, and taking the square root. Agreeing with our intuition, the prediction becomes more accurate as we get closer to the job expiration time. We note that compared to IMP-WEEKLY, IMP-FULL reduces the overall RMSE (the average RMSE across all days) by 6.3%, while IMP-CTR-FULL reduces the overall RMSE by 7.5%. Hence adding more features to the baseline forecasting model does help improve prediction accuracy significantly.

We now evaluate the estimation of the forecasted confidence interval for $c_{j,1:T}$ (Equation (20)) with model IMP-CTR-FULL, with the consideration of how it will be used in our problem setting of job application redistribution. At time t, let the estimated confidence interval for job j be $[l_{jt}, u_{jt}]$. We evaluate the performance in terms of a binary classification problem as follows:

- Boosting a job correctly: For each job j, the true label is positive if $c_{j,1:T} < minApp$, and negative otherwise. The predicted label is positive if $u_{jt} < minApp$, and negative otherwise. Here we use $minApp = 8$.
- Penalizing a job correctly: For each job j, the true label is positive if $c_{j,1:T} > maxApp$, and negative otherwise. The predicted label is positive if $l_{jt} > maxApp$, and negative otherwise. Here we use $maxApp = 100$.

As the confidence interval is defined as $[E[c_{j,1:T}]-\eta\sqrt{Var[c_{j,1:T}]}, E[c_{j,1:T}]+\eta\sqrt{Var[c_{j,1:T}]}]$, we can pick different values of η, and generate precision and recall values for both boosting and penalization strategies. The resulting ROC curves, which reflect the trade-off between precision and recall, corresponding to different values of t (the number of days after the job gets posted) are shown in Figures 5(b) and 5(c). As expected, the closer the current time to the expiration date, the better the trade-off curves are.

It is also worth noting that if we average across all the values of t (day 1, 2, ..., 30) , with $\eta = 1.96$ (i.e., the 95% confidence interval), we can achieve $\sim$ 90% recall for the boosting strategy with false positive rate of only ~3%, and ~86% recall for the penalization strategy with false positive rate being only $\sim$ 0.4%. This shows that our model is pretty accurate.

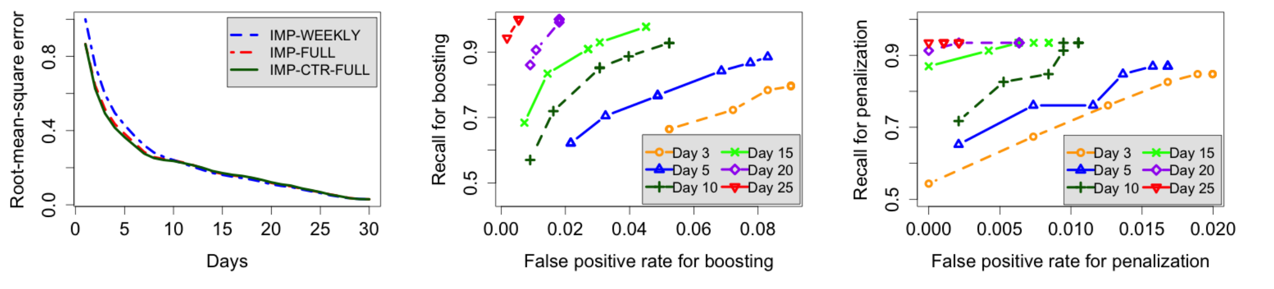
6.3 Online A/B Testing Experiments

We next present the results from online A/B testing of our job application redistribution methodology, for several variations and parameter choices. Our experiment was performed using the forecasting model IMP-CTR-FULL during the period of Sep-Dec 2016, with the control being no job application redistribution at all (i.e., pure relevance driven model). Since we would like to drive more applications to the jobs that receive very few applications, and avoid sending extra applications to the jobs that have already received too many applications, every day we split the jobs in our system into the following three buckets and report the number of applications for each bucket individually:

- Bucket 1: The number of applications for this job at this moment is less than *minApps*.
- Bucket 2: The number of applications for this job at this moment is in the range, [*minApps*, *maxApps*].
- Bucket 3: The number of applications for this job at this moment is greater than *maxApps*.

Table 1 shows the performance of three strategies, namely, boosting only, penalization only, and boosting plus penalization, in terms of the gain in the number of application for the three buckets and overall, compared to the control. We used the following parameter choices in this experiment: $minApps = 8$; $maxApps = 100$; the boosting factor, $boostFactor = 1.05$; the GLMix model score threshold, $h = 0.8$; the softness parameter for job penalization exponential decay function, $softness = 300$. Since the loss of applications in Bucket 3 is considered as a positive signal, we note that boosting and penalization together provide the best trade-off between the overall number of applications and the application distribution among the three buckets. We also note that we are able to redistribute significant number of applications in Bucket 3 to Bucket 1, while keeping the total applications to be flat or slightly positive. The fact that there is no decline in the total number of applications is due to the behavior of our ranking system, where user's attention was redistributed from relevant jobs with high number of applications to relevant jobs with lower number of applications.

As we rolled out the job redistribution methodology to the online production system, we also wanted to measure its impact on the distribution of applications among all the jobs. One appropriate measure is the *entropy* of the job application distribution, defined as follows: Let the number of applications at expiration time T for job j be $c_{j,1:T}$. The probability of a given job receiving an application



(a) RMSE of the forecasting model prediction for the number of job applications over time.

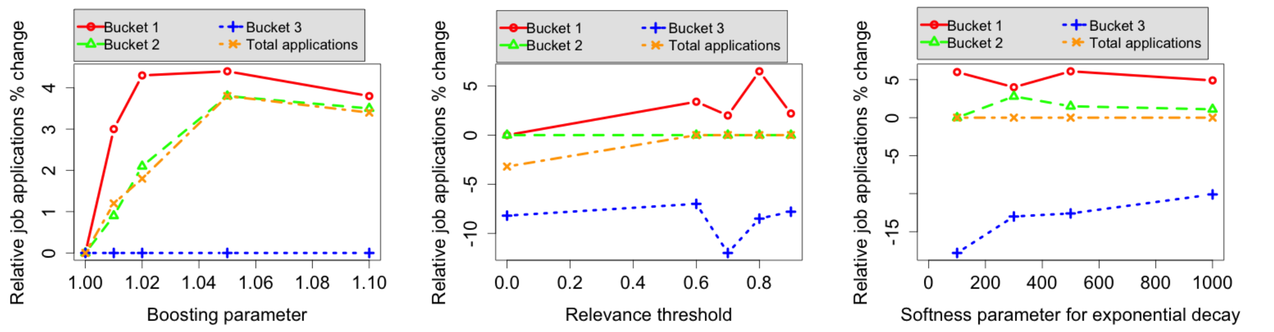
(b) ROC curves for job boosting strategy corresponding to different number of days used in training.

(c) ROC curves for job penalization strategy corresponding to different number of days used in training.

Figure 5: Offline evaluation for our job application forecasting model.

Table 1: Online A/B testing performance of the job boosting and penalization strategies, compared to the control.

Method	Applications % change in Bucket 1	Applications % change in Bucket 2	Applications % change in Bucket 3	Total applications % change
Boosting only	+4.4%	+3.8%	+2.7% (non-significant; p-val=0.22)	+3.8%
Penalization only	+4.0%	+1.9% (non-significant; p-val=0.24)	-13.2%	-1.1% (non-significant; p-val=0.4)
Boosting + Penalization	+6.5%	+3% (non-significant; p-val=0.05)	-8.7%	+2.3% (non-significant; p-val=0.1)



(a) The impact of the boosting factor on the number of job applications (with relevance threshold 0.8 for boosting).

(b) The impact of the relevance model score threshold for boosting on the number of job applications.

(c) The impact of the softness parameter for job penalization exponential decay function on the number of job applications.

Figure 6: Performance of the boosting / penalization methodology with different parameters in online A/B testing experiments.

from the pool of all applications is thus $p_j = c_{j,1:T}/\sum_j c_{j,1:T}$, so that the entropy could then be computed as $\sum_j p_j \log(1/p_j)$.

Since the entropy is maximized when each job receives exactly the same number of applications, the higher the entropy is, the more evenly the applications are distributed across jobs. Table 2 shows the change of entropy after deploying our job application redistribution methodology to LinkedIn's jobs recommendation system. The 12% increase in the entropy demonstrates the effectiveness of our methodology towards distributing the job applications more evenly across the entire job marketplace.

Table 2: Relative change in the entropy of job application distribution after the deployment of our application redistribution methodology.

Deployment	% change in the entropy of job application distribution
Control	0%
Application Redistribution	+12%

6.4 Effect of the Boosting and Penalization Parameters

To understand how the choice of the job boosting parameters (the boosting factor and the GLMix model score threshold, h) and the job penalization parameters (softness for the exponential decay function) impacts the online performance, we experimented with a set of different values for these three parameters. Figure 6 shows the performance of the overall number of applications as well as that of the three buckets. It is clear that with a careful tuning of these parameters, our methodology can achieve significantly better performance.

One lesson learned from our online experimentation is that the existence of the relevance model score threshold, h is critical, as shown in Figure 6(b). Initially, our boosting strategy did not have the threshold set up (i.e., $h = 0$), and the A/B testing results for the boosting showed a very significant loss of the job applications overall with little gain of applications in Bucket 1. By further analysis, we found that the loss of user engagement was mostly due to boosting of jobs with very low relevance scores. Therefore, the threshold was introduced to mitigate this problem and after selecting a good value of h, the performance of boosting becomes much better as shown in Table 1.

7 RELATED WORK

Our work resides in the domain of online recommender systems, which are widely adopted across many web applications, e.g., movie recommendations [9], e-commence item recommendations [17], job recommendations [5] and so forth, where authors mainly concentrate on the relevance retrieval and ranking aspects of the recommendation system.

There is insightful research and modeling of the hiring processes within job marketplaces. Such research includes work related to estimation of employee reputation for optimal hiring decisions [8], as well as work related to ranking and relevance aspects of job matching in labor marketplaces [10, 12, 16]. There has been work related to the theory of optimal hiring process, e.g., on the problem of finding the right hire for a job (the hiring problem), as well as on the classical secretary problem, where a growing company continuously interviews and decides whether to hire applicants [6, 13]. Authors of [4] investigated job marketplace as a two-sided matching market using locally stable matching algorithms for solving the problem of finding a new job using social contacts. In this paper, we treat mechanisms for retrieval and ranking of jobs for users as given (following [5, 22]), and instead focus on the problem of redistributing users' attention from overly popular jobs to jobs that do not have enough applications. The probability of hire given an application is not considered in this paper due to high confidentiality of such data.

The early intervention idea that motivates this work has also been adopted in the problem of ads pacing [3, 14, 21]; the objective there is to help advertisers reach as many targeted users as possible and ensure smooth budget spending over time. The pacing algorithm throttles the ad delivery in real time if it believes that the ad would run out of budget sooner than expected, based on the current forecast of the total number of impressions the ad is expected to receive. Although the idea seems similar, there are a few major differences. The forecasting models for the number of impressions an ad would receive are usually based on the targeting information the advertisers provide when posting an ad, while the matching of user profile and job profile is purely driven by our job recommendation system; no targeting is currently in place for job postings at LinkedIn. Also, unlike the hard daily budget constraints for ads, the constraints of the minimum and maximum number of applications a job can receive are soft. For example, consider a job that has received too many applications, but happens to be the only job matching a user's profile; we would still show the job to the user so that the user can have a good experience when visiting LinkedIn jobs homepage.

The Gamma-Poisson model to estimate the CTR dynamically originates from [2], which shows that this model can provide similar or even better performance than time series models [11] in terms of accuracy. The Negative Binomial model for estimating the number of impressions is similar to [1]; however the application setting is entirely different, namely, estimating rates of rare events in the domain of computational advertising.

Our idea of penalizing the job score for over-delivered number of applications is similar to impression discounting which is widely adopted in recommender systems [15]. However, impression discounting is designed to avoid over-exposure of the same items to the same user, which is a different objective than ours. The idea of boosting the relevance scores in the case of jobs with very few applications is similar to the explore-exploit methodology for discovering new relevant content to users [7, 18]. However, the context and the problem are entirely different, as the objective of the exploration is to reduce the variance of the parameters for new items so that we can find good and relevant items and show them to users (exploitation).

8 CONCLUSION

We presented the problem of distributing job applications in the job recommendation system to optimize for customer satisfaction while keeping the recommendations to be relevant. We developed a statistical job application forecasting model to provide confidence intervals of job applications at expiration time, and used the confidence intervals to boost or penalize the jobs which we believe are likely to receive too few or too many applications respectively. We demonstrated the efficacy of our approach through both offline evaluation and online A/B testing experiments.

One promising future work is to incorporate several other important customer satisfaction and member engagement metrics into the consideration of our jobs ranking model. We note that our customers not only care about the number of applications they receive; it is also critical to make sure that these applicants are well qualified so that some of them can be finally hired. Building models to predict who would qualify for interviews / hiring for the job posting and ensuring that we can distribute the *qualified* applicants across jobs to better optimize customer satisfaction is a challenge we would like to address in the future. Also, besides the total number of applications from members, other long-term user engagement metrics such as retention rates can potentially be incorporated into our ranking model.

9 ACKNOWLEDGMENTS

The authors would like to thank Deepak Agarwal, Rahim Daya, Monica Lewis, Jason Phan, Kaushik Rangadurai, Chris Walker and Jesse Ward for insightful feedback during development, deployment

and integration of our system as part of LinkedIn's job recommendation engine.

REFERENCES

[1] Deepak Agarwal, Rahul Agrawal, Rajiv Khanna, and Nagaraj Kota. 2010. Estimating rates of rare events with multiple hierarchies through scalable log-linear models. In *KDD*. https://doi.org/10.1145/1835804.1835834

[2] Deepak Agarwal, Bee-Chung Chen, and Pradheep Elango. 2009. Spatio-temporal Models for Estimating Click-through Rate. In *WWW*. https://doi.org/10.1145/1526709.1526713

[3] Deepak Agarwal, Souvik Ghosh, Kai Wei, and Siyu You. 2014. Budget Pacing for Targeted Online Advertisements at LinkedIn. In *KDD*. https://doi.org/10.1145/2623330.2623366

[4] Esteban Arcaute and Sergei Vassilvitskii. 2009. Social Networks and Stable Matchings in the Job Market. In *WINE*. https://doi.org/10.1007/978-3-642-10841-9_21

[5] Fedor Borisyuk, Krishnaram Kenthapadi, David Stein, and Bo Zhao. 2016. CaSMoS: A Framework for Learning Candidate Selection Models over Structured Queries and Documents. In *KDD*. https://doi.org/10.1145/2939672.2939718

[6] Andrei Z. Broder, Adam Kirsch, Ravi Kumar, Michael Mitzenmacher, Eli Upfal, and Sergei Vassilvitskii. 2008. The Hiring Problem and Lake Wobegon Strategies. In *SODA*. https://doi.org/10.1137/07070629X

[7] Olivier Chapelle and Lihong Li. 2011. An Empirical Evaluation of Thompson Sampling. In *NIPS*. http://dl.acm.org/citation.cfm?id=2986459.2986710

[8] Maria Daltayanni, Luca de Alfaro, and Panagiotis Papadimitriou. 2015. WorkerRank: Using Employer Implicit Judgements to Infer Worker Reputation. In *WSDM*. https://doi.org/10.1145/2684822.2685286

[9] Carlos A. Gomez-Uribe and Neil Hunt. 2015. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* (2015). https://doi.org/10.1145/2843948

[10] Viet Ha-Thuc, Ye Xu, Satya Pradeep Kanduri, Xianren Wu, Vijay Dialani, Yan Yan, Abhishek Gupta, and Shakti Sinha. 2016. Search by Ideal Candidates: Next Generation of Talent Search at LinkedIn. In *WWW*. https://doi.org/10.1145/2872518.2890549

[11] Jeff Harrison and Mike West. 1999. *Bayesian forecasting & dynamic models*. Springer New York.

[12] Marios Kokkodis, Panagiotis Papadimitriou, and Panagiotis G. Ipeirotis. 2015. Hiring Behavior Models for Online Labor Markets. In *WSDM*. https://doi.org/10.1145/2684822.2685299

[13] Ravi Kumar, Silvio Lattanzi, Sergei Vassilvitskii, and Andrea Vattani. 2011. Hiring a Secretary from a Poset. In *EC*. https://doi.org/10.1145/1993574.1993582

[14] Kuang-Chih Lee, Ali Jalali, and Ali Dasdan. 2013. Real Time Bid Optimization with Smooth Budget Delivery in Online Advertising. In *ADKDD*. https://doi.org/10.1145/2501040.2501979

[15] Pei Lee, Laks V.S. Lakshmanan, Mitul Tiwari, and Sam Shah. 2014. Modeling Impression Discounting in Large-scale Recommender Systems. In *KDD*. https://doi.org/10.1145/2623330.2623356

[16] Jia Li, Dhruv Arya, Viet Ha-Thuc, and Shakti Sinha. 2016. How to Get Them a Dream Job?: Entity-Aware Features for Personalized Job Search Ranking. In *KDD*. https://doi.org/10.1145/2939672.2939721

[17] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.Com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (2003), 76–80. https://doi.org/10.1109/MIC.2003.1167344

[18] Omid Madani and Dennis DeCoste. 2005. Contextual Recommender Problems [Extended Abstract]. In *UBDM*. https://doi.org/10.1145/1089827.1089838

[19] Sankar Sriram and Asif Makhani. 2014. LinkedIn's Galene Search engine. (2014). http://engineering.linkedin.com/search/did-you-mean-galene.

[20] Roshan Sumbaly, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, and Sam Shah. 2012. Serving Large-scale Batch Computed Data with Project Voldemort. In *FAST*. http://dl.acm.org/citation.cfm?id=2208461.2208479

[21] Jian Xu, Kuang-chih Lee, Wentong Li, Hang Qi, and Quan Lu. 2015. Smart Pacing for Effective Online Ad Campaign Optimization. In *KDD*. https://doi.org/10.1145/2783258.2788615

[22] XianXing Zhang, Yitong Zhou, Yiming Ma, Bee-Chung Chen, Liang Zhang, and Deepak Agarwal. 2016. GLMix: Generalized Linear Mixed Models For Large-Scale Response Prediction. In *KDD*. https://doi.org/10.1145/2939672.2939684