MANAGEMENT SCIENCE
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0025-1909 | EISSN 1526-5501 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000

The Good, the Bad, and the Unhirable: Recommending Job Applicants in Online Labor Markets

Marios Kokkodis
Boston College
kokkodis@bc.edu

Panagiotis G. Ipeirotis
New York University
panos@stern.nyu.edu

Choosing job applicants to hire in online labor markets is hard. To identify the best applicant at hand, employers need to assess a heterogeneous population. Recommender systems can provide targeted job-applicant recommendations that help employers make better-informed and faster hiring choices. However, existing recommenders that rely on multiple user evaluations per recommended item (e.g., collaborative filtering) experience structural limitations in recommending job applicants: Because each job application receives only a single evaluation, these recommenders can only estimate noisy user-user and item-item similarities. On the other hand, existing recommenders that rely on classification techniques overcome this limitation. Yet these systems ignore the hired worker's performance—and as a result, they uniformly reinforce prior observed behavior that includes unsuccessful hiring choices—while they overlook potential sequential-dependencies between consecutive choices of the same employer.

This work addresses these shortcomings by building a framework that uses job-application characteristics to provide recommendations that (1) are unlikely to yield adverse outcomes (performance-aware) and (2) capture the potentially evolving hiring preferences of employers (sequence-aware). Application of this framework on hiring decisions from an online labor market shows that it recommends job applicants who are likely to get hired and perform well. A comparison with advanced alternative recommender systems illustrates the benefits of modeling performance-aware and sequence-aware recommendations. An empirical adaptation of our approach in an alternative context (restaurant recommendations) illustrates its generalizability and highlights its potential implications for users, employers, workers, and markets.

1. Introduction

Online labor markets such as Peopleperhour, Freelancer, and Upwork facilitate global short-term contracts or freelance work. Employers can purchase services from online workers that complete diverse jobs, including web development, graphic design, accounting, sales, marketing, and data science. Like other online platforms, online labor markets grew exponentially during the past decade (Freelancers-union 2017). Upwork, for example, hosts fourteen million workers and five million employers and reports a total annual transaction volume of $1 billion (Lauren 2017, Brier and Pearson 2018). Similarly, Peopleperhour connects 750,000 employers to 1.5 million workers around the world (Atkins 2019). This growth is projected to continue as automation and the sharing economy structure the future of work (Sundararajan 2016, Institute of Business Value 2019).

Similar to offline settings, identifying capable workers to hire in online labor markets is hard (Klazema 2018). To make hiring decisions, employers need to assess the observed and latent characteristics of the available job applicants. The observed characteristics include the applicants' education, skills, work histories, and certifications, as listed on their resumes (Kokkodis et al. 2015). The latent characteristics are the workers' actual knowledge and abilities (when skill certifications are absent), as well as the workers' motivation, drive, and willingness to collaborate and do a good job (Geva and Saar-Tsechansky 2016). The existence of latent characteristics, the heterogeneity that appears in the observed ones (Kokkodis and Ipeirotis 2014), and the interactions between the two create an uncertain environment of information asymmetry (Akerlof 1970, Pelletier and Thomas 2018).

Besides, because job applications are free, workers often broadcast their availability widely to increase their chances of getting hired (Kokkodis et al. 2015). Large numbers of job applications increase employers' search costs (Guo et al. 2017) and may even result in unfilled openings (Snir and Hitt 2003, Carr 2003). By some estimates, around 60% of openings in online labor markets never reach a contract (Zheng et al. 2015). As a result, increased search costs hurt both the market (through lack of revenue) and its users (employers and workers), many of whom opt to quit (Guasch et al. 2003, Autor 2001).

Information asymmetry and search costs are not unique to online labor markets. Both are present in almost every type of online market (Dimoka et al. 2012, Chen et al. 2004, Ba and Pavlou 2002). By providing targeted product or service recommendations, recommender systems are a popular solution to these issues (Pathak et al. 2010, Brynjolfsson et al. 2011, Fleder and Hosanagar 2009). Specifically, in our context, recommender systems can help employers make better-informed decisions *by ranking job applicants according to their likelihood of getting hired and performing well.*

Yet, when applied to job-applicant recommendations, existing recommender systems experience shortcomings. In particular, recommenders that rely on *many assessments* per item to provide recommendations (e.g., collaborative filtering; we refer to these systems as many-assessment recommenders; see Adomavicius and Tuzhilin 2005, Ricci et al. 2011, Quadrana et al. 2018) have limited information to estimate the required user-user and item-item similarities. This is because, in online labor markets, (1) task requirements are diverse (i.e., no two jobs are identical), (2) job applicants evolve by gaining experience or learning new skills, (3) different job openings attract different pools of applicants and, as a result, tasks have non-overlapping choice sets, and (4) the ratio of employers to workers is significantly lower than conventional many-assessment contexts (e.g., movie recommendations). These characteristics allow each *job application*—the focal recommended item—to be evaluated only once by a single employer. Hence, when applied in this context, many-assessment systems will underperform as they will rely on a single observed assessment per job application to estimate noisy user-user and item-item similarities.

On the other hand, systems that rely on a *single assessment* to provide recommendations overcome these limitations (we refer to these systems as single-assessment recommenders; see Kokkodis et al. 2015, Abhinav et al. 2017, Baba et al. 2016, Mao et al. 2015). However, existing single-assessment systems have two shortcomings when adapted to recommend job applicants. First, they ignore the performance of the hired applicant. Instead, they learn and uniformly reinforce previously observed behavior, including employer choices that yielded unsuccessful outcomes (Kokkodis et al. 2015, Abhinav et al. 2017). Second, they overlook potential sequential-dependencies between

hiring decisions of the same employers. Hence, they implicitly assume that employer hiring preferences remain the same over repeated hiring choices. As a result, they make recommendations that regress to a mean that uniformly aggregates behaviors of varying-experience and varying-ability employers.

This work identifies three principles for designing job-applicant recommenders that address these limitations of existing many-assessment and single-assessment systems: A job-applicant recommender should be a *single-assessment* system that is both *performance-aware* and *sequence-aware*. Single-assessment systems overcome the limitations of many-assessment approaches and learn to recommend items that only a single user evaluates. Performance-aware systems identify prior unsuccessful hiring choices and learn to promote job applicants who are not only hireable but also likely to perform well. Sequence-aware systems allow repeat employers to adjust their hiring preferences and evolve independently.

We instill these principles into a new *single-assessment* framework that conceptualizes three discrete job-application outcomes ("No-hire," "Hire-negative," "Hire-positive"—*performance-aware*) and captures any changes in hiring-preferences through a Hidden Markov Model (HMM— *sequence-aware*). Implementation of the proposed approach on hiring decisions from a major online labor market highlights the advantages of the three design principles. Across four different evaluation metrics, our framework significantly outperforms alternative job-applicant recommenders, including existing and new single-assessment systems (logistic regression, gradient boosting, random forests, support vector machines, recurrent neural networks) and adaptations of popular many-assessment systems such as collaborative filtering-based approaches (singular value decomposition, HMM for collaborative filtering; see Sahoo et al. 2012) and deep sequential recommenders (Kula 2018). Repeat employers benefit the most from our approach, as these employers receive personalized sequence-aware recommendations by evolving across their distinct hiring-preference paths. Application of our approach in an alternative context (restaurant recommendations) shows its generalizability in contexts where both the recommended items and the user preferences change.

This work extends research in recommender systems and online labor markets by identifying and addressing critical shortcomings of existing many-assessment and single-assessment approaches when recommending job applicants. By conceptualizing the necessary design principles, this paper is the first to directly incorporate worker performance into the recommendation process and allow employer hiring preferences to change. Methodologically, compared with other HMM-based recommenders (Sahoo et al. 2012, Hosseinzadeh Aghdam et al. 2015, Zhang et al. 2016b) and traditional HMM approaches for classification (Murphy 2012), the proposed HMM offers a unique structure that models choices according to observed applicant-employer-task characteristics and allows hiring-preferences to evolve only after the completion of each task. As a result, because our framework provides job-applicant recommendations that lead to successful outcomes, it can benefit (1) workers to differentiate, (2) employers to make better-informed and faster (reduced search cost; see Bakos 1997) decisions, and (3) markets to increase their transaction efficiency, which in turn results in increased revenue and employer satisfaction.

2. Research context

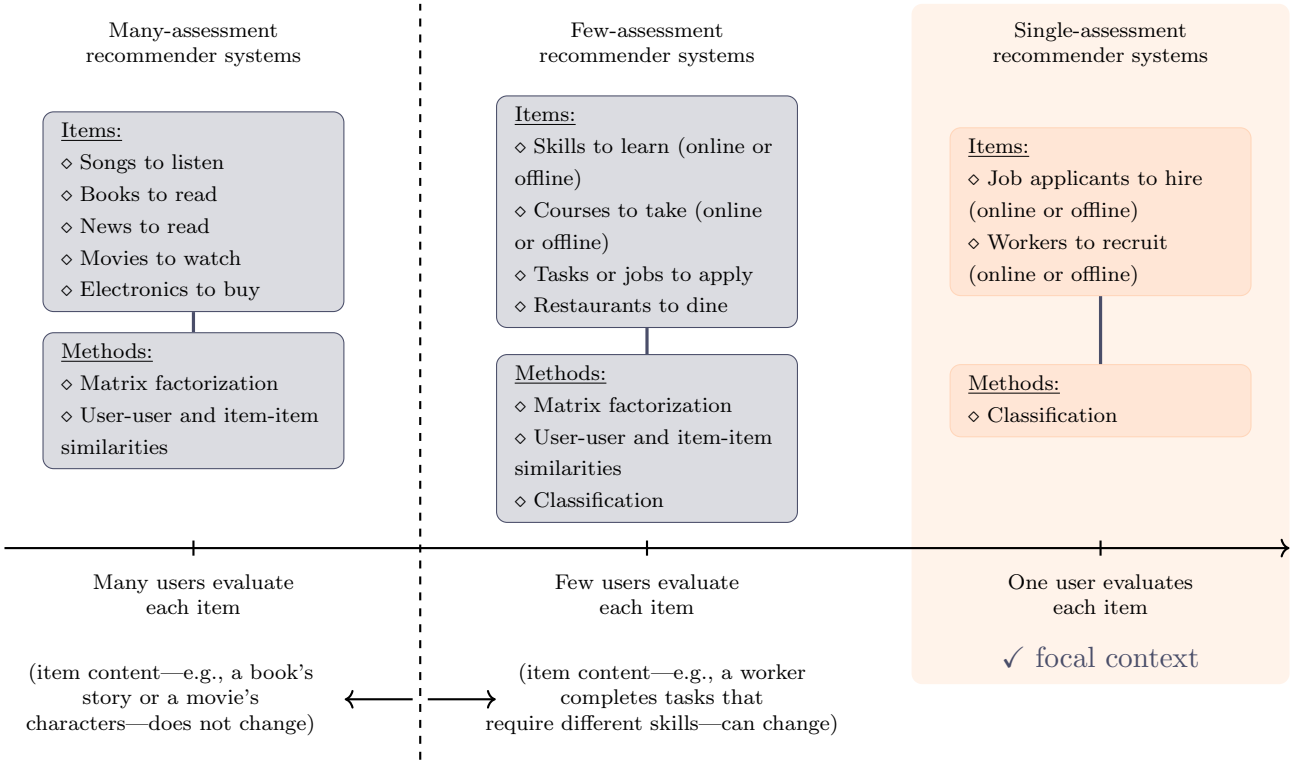
The ultimate goal of this work is to provide relevant job-applicant recommendations in the context of an online labor market.

Problem definition: *Assume a given job opening with a set of job applicants. We are interested in ranking these applicants according to their likelihood of getting hired and performing well.*

At the problem's core is a *recommender system* that ranks job applications within a given opening. A rich literature on recommender systems offers diverse approaches (Adomavicius and Tuzhilin 2005, Kantor et al. 2011, Quadrana et al. 2018, Zhang et al. 2019), some of which focus specifically on recommending job applicants and job openings in online labor markets (Kokkodis et al. 2015, Abhinav et al. 2017, Goswami et al. 2014, Baba et al. 2016, Färber et al. 2003).

These existing recommender systems can be many-assessment or single-assessment depending on the items they recommend (Figure 1):

Figure 1 Recommender systems vary according to their focal recommended items



Many-assessment systems recommend items that multiple users evaluate by estimating user-user and item-item similarities. Single-assessment systems recommend items that a single user evaluates by modeling item characteristics through classification techniques. Few-to-many systems use either many-assessment or single-assessment methods to recommend items that receive few user evaluations.

- Many-assessment recommenders: Many users evaluate each recommended item as the underlying item content (e.g., a book's story, a movie's finale, or a camera's chip) does not change. Examples include systems that recommend books, songs, and movies (Adomavicius and Tuzhilin 2005, Ricci et al. 2011, Quadrana et al. 2018). The availability of multiple evaluations per item allows these systems to estimate item-item and user-user similarities. Through matrix factorization techniques, they predict user-item affinities and make recommendations (Koren et al. 2009).
- Single-assessment recommenders: Only a single user evaluates each recommended item, as the underlying item content (e.g., a worker completing tasks that require different skills) changes. Examples include systems that recruit workers or rank job applicants (Kokkodis et al. 2015,

Abhinav et al. 2017, Goswami et al. 2014, Mao et al. 2015). These systems use classification approaches that model observed item characteristics to provide recommendations.

Between these two types, Figure 1 identifies few-assessment systems that recommend items that either change or receive only a few user evaluations. These systems borrow techniques from either many-assessment or single-assessment systems to provide recommendations. The next paragraphs present a brief overview of existing recommender systems and explain in detail the limitations of these systems when applied to the focal context.

2.1. A brief overview of existing recommender systems

2.1.1. Many-assessment recommenders: Depending on whether they model sequential user actions, many-assessment recommender systems can be sequence-independent or sequence-aware (Adomavicius and Tuzhilin 2005, Quadrana et al. 2018).

Sequence-independent recommenders: Conventional content-based, collaborative filtering, and hybrid approaches do not explicitly model the sequence of user actions. *Content-based* systems focus on understanding the commonalities between items that a user has rated highly in the past to recommend similar items (Adomavicius and Tuzhilin 2005, Billsus and Pazzani 2000, Zheng et al. 2017, Gong and Zhang 2016). *Collaborative filtering* systems suggest items that similar users with the targeted user have liked in the past (Billsus and Pazzani 1998, Breese et al. 1998, Adomavicius and Tuzhilin 2005, Breese et al. 1998, Delgado and Ishii 1999). *Hybrid* systems combine notions from content-based with characteristics of collaborative approaches to provide systems that overcome some of the limitations of the two approaches (Adomavicius and Tuzhilin 2005, Balabanović and Shoham 1997, Si and Jin 2003, Tso-Sutter et al. 2008). Recent advances in deep learning extend these approaches by allowing neural networks to structure new collaborative filtering frameworks (Li et al. 2015, Wang et al. 2015).

Sequence-aware recommenders: Sequence-aware systems explicitly model sequences of past events and recommend items according to the short-term behavior of the user (Quadrana et al. 2018). There are three main classes of such recommenders: sequence learning, sequence-aware matrix factorization, and hybrid systems (Quadrana et al. 2018). *Sequence-learning* methods include frequent

pattern mining (Hsueh et al. 2008, Zang et al. 2010, Jannach et al. 2015) and sequence modeling (recurrent neural networks, Markov models, reinforcement learning; see Garcin et al. 2013, He et al. 2009, Sahoo et al. 2012, Moling et al. 2012, Natarajan et al. 2013, Song et al. 2016, Twardowski 2016, Zhang et al. 2019). *Sequence-aware matrix factorization* approaches combine information from timestamps with algorithms that rely on matrix completion (Koren 2009, Quadrana et al. 2018). *Hybrid* approaches combine matrix factorization techniques with Markov chains (Rendle et al. 2010, Lian et al. 2013, He and McAuley 2016) to provide next-item recommendations.

The rich literature of sequence-aware systems further includes approaches that rely on Hidden Markov Models (HMMs) to recommend items that their content does not change such as news, songs, and movies. In particular, HMM-based collaborative filtering allows user preferences to evolve over months in terms of (1) how many news articles they will read and (2) which of the available articles they will pick (Sahoo et al. 2012). Hierarchical HMMs allow for two levels of hidden states to provide personalized song recommendations (Hosseinzadeh Aghdam et al. 2015). Hidden semi-Markov models allow users to stay in a (latent) interest state for varying time intervals to provide movie, song, and webpage recommendations (Zhang et al. 2016b).

2.1.2. Single-assessment recommenders: Single-assessment systems rely on classification techniques to provide recommendations (Figure 1). Prior research provides a multitude of sequence-independent classification-based approaches. For instance, job-applicant recommenders evaluate unique choice sets of applicants within each job opening to make personalized recommendations (Kokkodis et al. 2015, Abhinav et al. 2017). These systems adapt popular classifiers (logistic regression, random forest, support vector machines) to rank job-applicants according to their likelihood of getting hired. Besides, automated recruiters rely on similar classification techniques (logistic regression, random forest, latent aspect models, Decision Trees, K-NN, Bayesian inference) to model both sides of the market. Such systems recruit candidates to apply to different jobs (Färber et al. 2003, Malinowski et al. 2006, Goswami et al. 2014, Mao et al. 2015) or redistribute job applications to create a more balanced marketplace (Borisyuk et al. 2017).

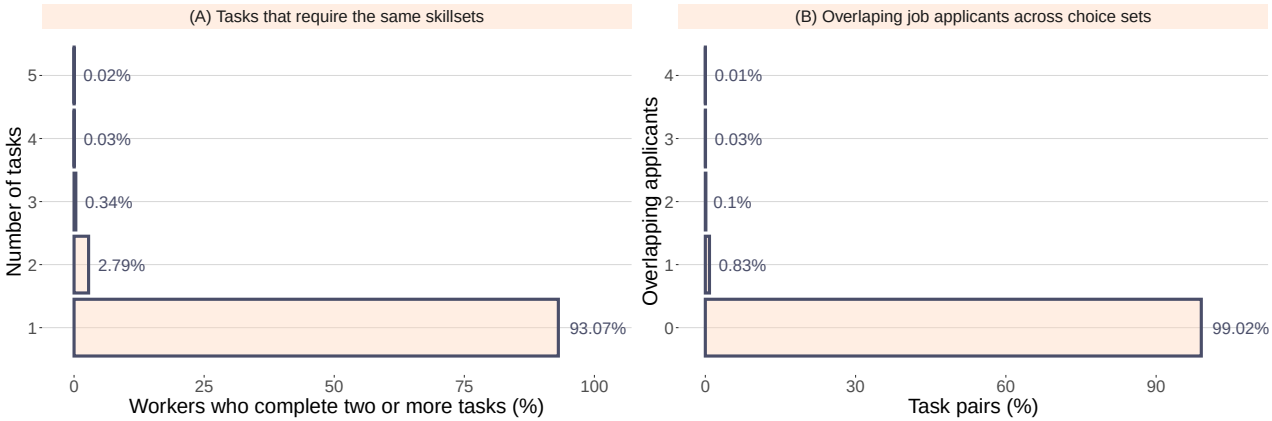
2.1.3. Few-assessment recommenders: Between many-assessment and single-assessment recommenders, there exist systems that recommend items that either change (e.g., a restaurant or a course that change over time) or receive very few user evaluations (e.g., a task that multiple users (workers) evaluate until it gets filled; see Figure 1). Such few-assessment systems borrow techniques from both many-assessment and single-assessment approaches. For instance, they can cluster the recommended items into static entities and apply matrix-factorization-based techniques similar to many-assessment approaches; Or, they can acknowledge that the items are relatively unique and apply classification algorithms similar to single-assessment recommenders.

Task recommenders in crowdsourcing environments are a prime example of few-assessment recommenders. Many tasks on Amazon Mechanical Turk require multiple workers (e.g., human intelligence tasks such as labeling). Because these tasks are often very similar, clustering them into task categories creates an environment where multiple workers choose to complete the same types of tasks (i.e., multiple users evaluate the same recommended item). As a result, through this type of clustering, matrix factorization techniques (Lin et al. 2014, Yuen et al. 2012, 2015, Kurup and Sajeev 2017) or algorithms that estimate user-item similarities (Safran and Che 2017) can recommend within-task-category tasks to workers.

On the contrary, when task recommenders focus on more complex jobs that are often unique and harder to cluster into categories, few-assessment systems use classification-based recommenders similar to single-assessment ones. For instance, logistic regression and support vector machines can provide job recommendations to workers in online labor markets (Hossain and Arefin 2019). Similarly, softmax functions can model the probability of winning a crowdsourcing contest and provide relevant task recommendations (Baba et al. 2016).

Besides task recommenders, there are many alternative few-assessment contexts that existing literature proposes both many-assessment and single-assessment techniques. These include (1) restaurant or venue recommendations (Farooque et al. 2014, Kurashima et al. 2013, Li and Li 2014), (2) online courses recommendations (Bousbahi and Chorfi 2015, Symeonidis and Malakoudis 2018, Pardos et al. 2017, Prabhakar et al. 2017), and (3) career path recommendations (Patel et al. 2017, Kokkodis and Ipeirotis 2020).

Figure 2 Characteristics of the focal context suggest that a single-assessment framework would be a better fit



93% of workers who complete multiple tasks in our dataset complete tasks that require unique skillsets; 99% of pairwise matched tasks have entirely unique choice sets of job applicants (zero overlap).

2.2. Overview of the focal single-assessment context

Can existing recommenders provide rankings of job applicants according to their likelihood of getting hired and performing well?

To understand the limitations of existing recommender systems we first need to identify the salient features of the focal problem. Our research context assumes a marketplace where workers apply to job openings, some of them get hired, and once they complete the required task, they receive a feedback score (rating) about their performance. This behavior suggests that some workers might get hired (and hence evaluated) multiple times over a period of time. Hence, one could argue that many-assessment frameworks could provide efficient solutions to our research question as they would be able to leverage user (employer) evaluations across multiple items (workers) to estimate the required user-user and item-item similarities.

Yet, recommending job applicants generates additional, unique characteristics that suggest that single-assessment recommenders would likely be more appropriate. Specifically:

- Diverse, heterogeneous tasks: When workers get hired multiple times, they usually complete different tasks. As a result, the ratings they receive represent their performance across different task requirements. For instance, consider the following real example from our data (described in detail in Section 4):

Data evidence 1 *A worker in our dataset completed two tasks that required the following skillsets:*

— *Skillset 1:* {`sql,.net-framework,asp.net-mvc,c#`}

— *Skillset 2:* {`jquery,css,html5,css3,asp.net-mvc` }

For the first task, the worker received a very high evaluation (89%). For the second task, the worker received a lower evaluation (67%). □

Even though the two tasks are contextually similar, they are not identical. The differences between the tasks generate evaluation inconsistencies: the received feedback scores could be specific to `sql` and `c#` or to `jquery`—i.e., the skills that *do not overlap* between the two skillsets. A many-assessment recommender system would assume that these two tasks are identical, and try to estimate user similarities based on how these two employers have evaluated this worker, even though these employers have rated this worker for two different tasks.

Data evidence 2 *Figure 2A shows that the vast majority (93%) of workers who complete more than one task in our dataset complete tasks that require unique skillsets.* □

Combined, these observations suggest that the underlying recommended item in our context should not be the worker, but instead, the combination of worker and task, i.e., a *job application*. Because only a single employer evaluates each job application, a single-assessment framework would likely be able to make more appropriate recommendations.

- Ratio of employers to workers: Another characteristic of our context is its low user-to-item ratio (i.e., employer-to-worker ratio). In many-assessment contexts, the user-to-item ratio is usually very large:

Data evidence 3 *Netflix has 73 million registered users*[1] *and offers 3,781*[2] *movies. The user-to-item ratio for Netflix is 19,307. In our data, there are 11,461 employers and 162,976 workers (Section 4). The user-to-item ratio in our context is 0.07.* □

[1] https://www.statista.com/statistics/250937/quarterly-number-of-netflix-streaming-subscribers-in-the-us/

[2] https://www.foxbusiness.com/technology/how-many-movies-on-netflix

The user-to-item ratio is particularly important for many-assessment recommender systems, as they rely on *multiple users evaluating multiple items* to estimate user-user and item-item similarities. When the available items are significantly more than the available users, many items do not receive any rating. This is generally true for online labor markets, where many workers are new or have never got hired on the platform before applying to a focal job opening (see p.3576, Table 1, Pallais 2014). When ratings are not available, many-assessment matrix factorization techniques that require such ratings to make predictions will likely fail (Appendix A). On the other hand, single-assessment systems will learn to rely on alternative, observed characteristics (beyond ratings, Table 1) to make recommendations, and as a result, they are more likely to generate better rankings of job applicants.

- Unique choice sets: In many-assessment contexts, the choice sets have large overlapping segments of product offerings over time. Even though new items enter the choice sets and old items exit frequently (e.g., new products on Amazon or new movies on Netflix), the vast majority of items remain available for long periods. On the contrary, in the focal context, each task attracts an almost unique set of job applicants that employers can choose from.

Data evidence 4 *Many movies and TV shows remain available on Netflix for multiple years. For instance, the TV show* "The Office" *has been in the choice sets of tens of millions of Netflix users from 2013 until 2020.*[3][4] *On the other hand, Figure 2B shows that each employer makes hiring decisions across vastly unique choice sets: 99% of the pairwise combinations of tasks in our data do not share any overlapping job applicant.* □

Conceptually, overlapping choice sets are critical for many-assessment approaches, as these approaches operate on the assumption that all items are available to all users (user-item matrix). Hence, in contexts such as ours, where very few workers of the marketplace are

[3] https://variety.com/2013/digital/news/netflix-adds-the-office-and-30-rock-final-seasons-other-nbc-shows-on-oct-1-1200682400/

[4] https://en.wikipedia.org/wiki/The_Office_(American_TV_series)

available to each employer at each point in time, many-assessment systems will likely generate noisy predictions that incorporate workers that will likely never apply (become available) to the focal job opening. On the contrary, single-assessment systems do not rely on the availability of workers, but instead, they make predictions based on the observed characteristics that each applicant-task combination offers (Table 1).

Appendix A provides additional examples and describes in detail the structural problems that many-assessment frameworks face when applied to the focal context; Table 2 summarizes the similarities and differences between the focal and many-assessment contexts; Figure 11 shows that the implementation of such approaches results in very poor recommendations; Figure 16 generalizes the poor performance of these systems in a few-assessment context, restaurant recommendations.

2.3. Limitations of existing single-assessment recommenders in the focal context

The previous discussion illustrates that our context requires a single-assessment framework. *What are the limitations of current single-assessment approaches?*

Existing single-assessment approaches can rank job-applicants according to their likelihood of getting hired (Kokkodis et al. 2015, Abhinav et al. 2017). Besides, classification algorithms proposed in automated recruiters can be adapted to address the focal problem (Goswami et al. 2014, Färber et al. 2003, Malinowski et al. 2006, Mao et al. 2015). However, similar to implicit-feedback many-assessment recommenders, these approaches ignore the performance of the hired worker (predict only "Hire" or "No-hire"). Hence, they learn and uniformly reinforce previously observed behavior, including choices that resulted in unsuccessful outcomes.

Furthermore, existing single-assessment approaches are sequence-independent: they uniformly consider all previous employer decisions to predict future behavior. As a result, they implicitly assume that employer hiring preferences remain the same over repeated hiring choices. This assumption, however, might not be entirely accurate. Over repeated hiring decisions and through gaining experience by remotely managing workers and by increasing their familiarity with the platform,

many employers might evolve and adjust their hiring preferences.[5] Hence, existing single-assessment systems will likely offer recommendations that might not fully capture the current hiring preferences of each employer.

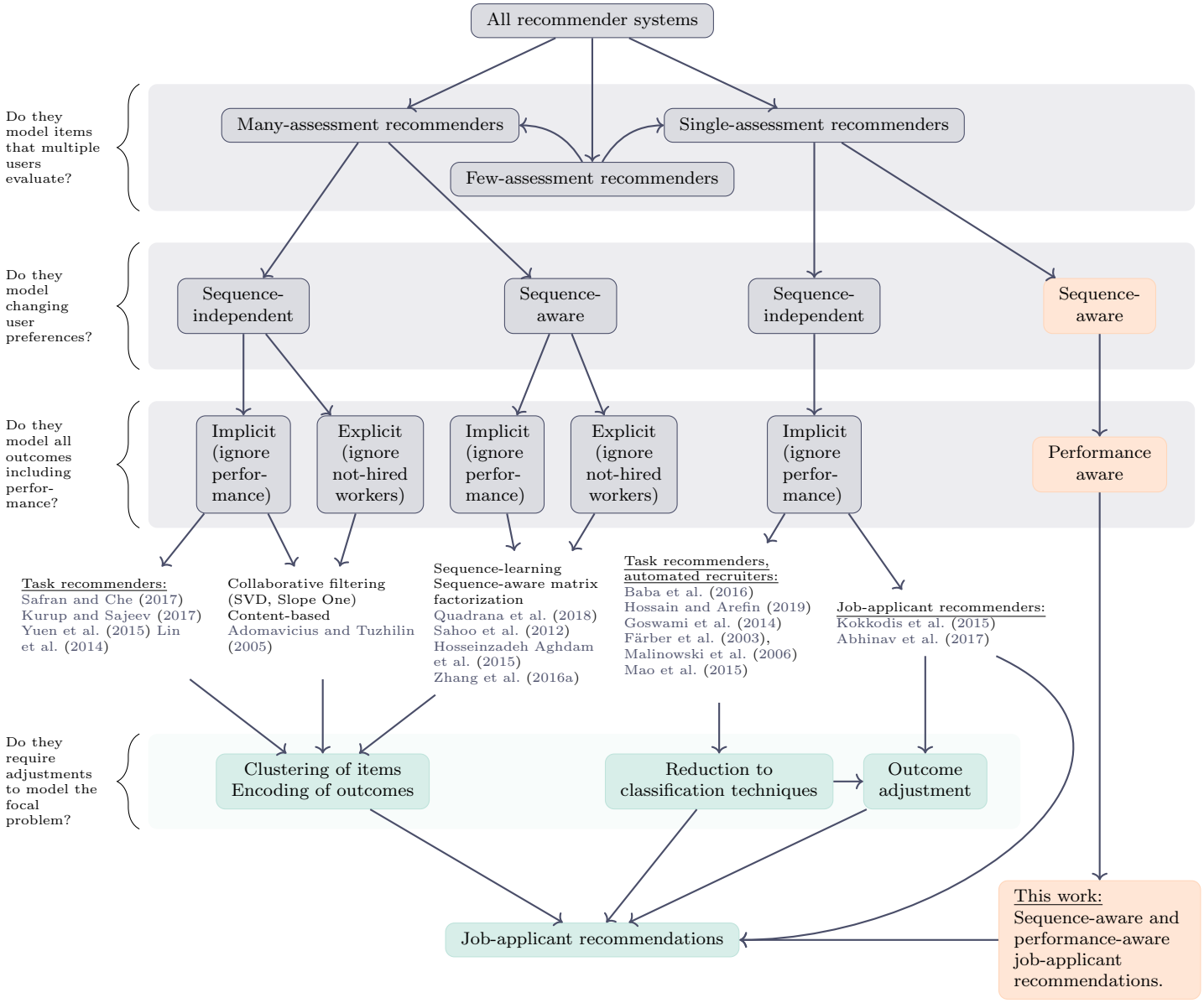
2.4. Design principles of a performance-aware and sequence-aware job-applicant recommender

Based on the above discussion on the shortcomings of existing recommender systems, we argue that an efficient system that ranks job applicants according to their likelihood of getting hired and performing well should be a *single-assessment recommender* that is both *performance-aware* and *sequence-aware*. Single-assessment systems overcome the limitations of many-assessment recommenders (Section 2.2, Appendix A) and allow models to recommend unique items through their time-varying characteristics. Performance-aware systems identify previously unsuccessful hiring decisions and learn to promote job applicants who are not only hireable but also likely to perform well. Sequence-aware systems allow employers to independently adjust their hiring preferences over time and offer recommendations that capture employers' current hiring preferences. Together, these three design principles structure frameworks that provide personalized recommendations of job applicants who are likely to get hired and perform well.

Figure 3 builds a decision tree that positions this work in the current literature of recommender systems. The first layer divides recommender systems into many-assessment and single-assessment (few-assessment systems map to either of the two). The second layer subdivides into sequence-aware and sequence-independent systems. The third layer further splits existing systems according to their outcome modeling (implicit or explicit feedback and performance-aware). The leaves of the decision tree identify examples of each subtree. Finally, the bottom of the decision tree describes the necessary adjustments that various approaches require in order to fit the focal problem.

[5] Employer hiring preferences might evolve over repeated hiring choices for multiple reasons. Some employers might learn through "trial and error," while others might adjust their confidence levels in managing and controlling workers remotely. As we discuss in Section 6.4, identifying the exact mechanism that employers learn or gain experience requires a deeper investigation that is outside the scope of this work. Instead, in this work we argue that job-applicant recommenders should capture these possibilities by allowing employer hiring preferences to evolve.

Figure 3 Positioning of our work in the current literature of recommender systems



None of the existing systems are single-assessment and, at the same time, sequence-aware and performance-aware: *Our work is the first to meet these three design principles.*

The figure annotates that none of the existing systems satisfy the three design principles: there are no existing single-assessment systems that are sequence-aware and performance-aware. Our work fills this gap by implementing a single-assessment framework that is both performance-aware and sequence-aware, and as a result, it provides relevant rankings of job-applicants according to their likelihood of getting hired and performing well. We discuss this framework next.

3. Sequence-aware and performance-aware job-applicant recommendations

A transaction in online labor markets starts with an employer creating a job opening. The opening description reveals characteristics that the employer is looking for, such as the required set of skills and experience. Workers who are looking for opportunities observe these characteristics and self-select to submit their job applications to openings that they see fit. Employers then assess the available job applicants and decide which ones to hire. The next paragraphs summarize a single-assessment system that provides sequence-aware and performance-aware rankings of such job applicants, hence guiding employers to make better-informed and faster decisions.

3.1. Latent hiring preferences and observed outcomes

An employer's hiring-decision process is latent. The market, however, observes the characteristics and outcomes of each job application. Specifically, for each applicant, the employer must first choose whether or not to hire. If the employer does not hire the applicant at hand, the market observes a "No-hire" outcome. However, if the employer chooses to hire the applicant at hand, the market eventually (when the task is completed) observes an outcome that describes the worker's performance: A "Hire-positive" outcome occurs when the performance of the hired worker is satisfactory, while a "Hire-negative" outcome occurs when the performance of the hired worker is not satisfactory. Formally, upon completion of a task, the market observes the employer's decisions on the task's job-applicants, which fall in the following set of possible outcomes ($\mathcal{Y}$):[6]

$$\mathcal{Y} = \{\text{"No-hire,"} \ \text{"Hire-negative,"} \ \text{"Hire-positive"}\}. \tag{1}$$

Over time, repeat employers who hire workers on multiple tasks might adjust their hiring preferences as they get more familiar with the challenges of hiring and managing remote workers.

[6] Section 4 discusses performance thresholds that separate "Hire-positive" from "Hire-negative" outcomes. Extensions to more granular outcomes are conceptually trivial but increase sparsity without a clear benefit for the platform, as the platform's ultimate goal is to predict the likelihood of getting hired and being successful, which is sufficiently captured by the three classes in $\mathcal{Y}$. Note that including both a "Hire-negative" and a "No-hire " class is crucial, as it allows markets to create flexible rankings across different objectives (Section 6.4 and Appendix G).

A Hidden Markov Model (HMM) can formally facilitate this possible evolution of hiring preferences. In particular, an HMM allows employers to operate from a latent state that captures their current hiring preferences. As employers hire workers over multiple tasks, they emit task-specific "No-hire," "Hire-positive," and "Hire-negative" observations. These observations reveal new information about the current employer hiring preferences. If the employers' preferences change, the HMM allows for employers to stochastically transition to new states that better capture their updated hiring preferences. Otherwise, employers remain in the state that best describes their observed behavior.

3.2. HMM structure

The definition of an HMM requires (1) a transition matrix T that describes the transition probabilities between states that capture different employer hiring preferences, and (2) an emission matrix E that describes the state-specific probability distributions across the set of observations $\mathcal{Y}$. In the next paragraphs, we assume that the HMM has K states such that $\mathcal{S} = \{s_1, s_2, ..., s_K\}$; Appendix F shows the tuning process of choosing an appropriate number of states K.

Transition probabilities: Employers emit observations $Y_t \in \mathcal{Y}$ for every job application they receive (Equation 1). Conceptually, hiring-preferences might only change after an employer hires a worker for a given task and observes the hired-worker's performance (i.e., at the completion of the task). The HMM encodes this by allowing employers to transition to a new state *only when* when the employer has:

1. Chosen a job applicant to hire,[7]
2. Chosen job applicants to not hire ("No-hire" observations), and,
3. Observed the outcome of the hired worker ("Hire-negative" or "Hire-positive" observation).

This context-specific requirement separates the transition probability matrix of our HMM from all prior HMM designs that allow stochastic transitions to occur *after every observation* (Murphy

[7] Employers' decisions on any given task happen instantaneously: the moment that one applicant gets hired, the remaining applicants emit a "No-hire" outcome.

2012, Bishop 2006, Sahoo et al. 2012). (Appendix J shows that this constraint yields significantly better results in practice.) Formally, the task-specific transition probability of a given employer to move from state s_k to state s_l when evaluating job application t for task o after observing the outcomes of $o-1$ tasks is as follows:

$$\lambda^{s_k s_l}_{\boldsymbol{\gamma}_{kl}\boldsymbol{X}_{o-1}} := \Pr(S_t = s_l | S_{t-1} = s_k; \boldsymbol{\gamma}_{kl}, \boldsymbol{X}_{o-1})$$
$$= \begin{cases} 0, & \text{if } t \text{ is not the first application of task } o \text{ to be evaluated and } s_l \neq s_k \\ 1, & \text{if } t \text{ is not the first application of task } o \text{ to be evaluated and } s_l = s_k, \\ \text{softmax}(\boldsymbol{\gamma}_{kl}\boldsymbol{X}_{o-1}), & \text{if } t \text{ is the first application of task } o \text{ to be evaluated} \end{cases} \quad (2)$$

where $\boldsymbol{X}_{o-1}$ is a vector of *employer characteristics* that captures the employer's behavior on the previously $o-1$ tasks with observed outcomes, and $\boldsymbol{\gamma}_{kl}$ is a parameter vector of state s_k that weights vector $\boldsymbol{X}_{o-1}$. (Note that Equation 2 assumes a sequence of ordered tasks according to their completion dates.) Based on Equation 2, for a job applicant t the transition matrix has the following form:

$$T(\boldsymbol{\gamma}, \boldsymbol{X}_{o-1}) = \begin{bmatrix} \lambda^{s_1 s_1}_{\boldsymbol{\gamma}_{11}\boldsymbol{X}_{o-1}} & \lambda^{s_1 s_2}_{\boldsymbol{\gamma}_{12}\boldsymbol{X}_{o-1}} & \cdots & \lambda^{s_1 s_K}_{\boldsymbol{\gamma}_{1K}\boldsymbol{X}_{o-1}} \\ \lambda^{s_2 s_1}_{\boldsymbol{\gamma}_{21}\boldsymbol{X}_{o-1}} & \lambda^{s_2 s_2}_{\boldsymbol{\gamma}_{22}\boldsymbol{X}_{o-1}} & \cdots & \lambda^{s_2 s_K}_{\boldsymbol{\gamma}_{2K}\boldsymbol{X}_{o-1}} \\ \vdots & \vdots & \vdots & \vdots \\ \lambda^{s_K s_1}_{\boldsymbol{\gamma}_{K1}\boldsymbol{X}_{o-1}} & \lambda^{s_K s_2}_{\boldsymbol{\gamma}_{K2}\boldsymbol{X}_{o-1}} & \vdots & \lambda^{s_K s_K}_{\boldsymbol{\gamma}_{KK}\boldsymbol{X}_{o-1}} \end{bmatrix}, \quad (3)$$

where $\boldsymbol{\gamma} = [\boldsymbol{\gamma}_{11}, \boldsymbol{\gamma}_{12}, \ldots, \boldsymbol{\gamma}_{KK}]'$.

Unlike most HMMs that provide state-specific static transition matrices (Murphy 2012, Bishop 2006, Sahoo et al. 2012, Hosseinzadeh Aghdam et al. 2015, Zhang et al. 2016b), the elements of the focal matrix T of Equation 3 are state-specific, employer-specific, and task-specific. Simply put, the transition probabilities of each employer are personalized, and they change according to the employer's current state and history of observed outcomes as captured by vector $\boldsymbol{X}_{o-1}$.

Emission probabilities: The second component of the HMM framework identifies the emission probabilities across the three hiring outcomes in $\mathcal{Y}$. The HMM assumes that these probabilities are (1) state-specific, representing the current hiring preferences of the employer, and (2) job-application specific, capturing the observed characteristics of the focal job application. In particular,

for a given job application, an employer at state s_k will make a hiring choice according to the following:

$$\mu^{s_k}_{\boldsymbol{\beta}^y_k \boldsymbol{Z}_t \boldsymbol{X}_{o-1}} := \Pr(Y_t = y | S_t = s_k; \boldsymbol{\beta}^y_k, \boldsymbol{Z}_t, \boldsymbol{X}_{o-1}) = \text{softmax}(\boldsymbol{\beta}^y_k [\mathbf{Z}_t, \boldsymbol{X}_{o-1}]') \,, \tag{4}$$

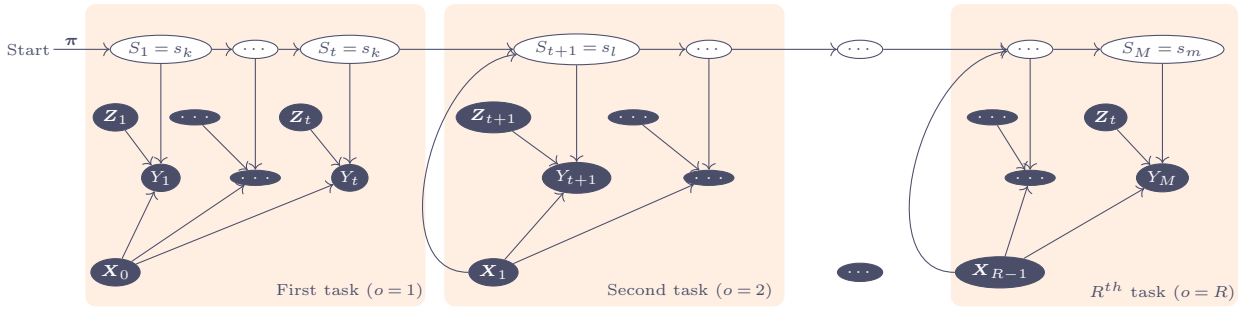
where $\mathbf{Z}_t$ is a vector of *job-application characteristics*, $\boldsymbol{X}_{o-1}$ is the same vector of employer characteristics that affect transition probabilities (Equation 2), $y \in \mathcal{Y}, s_k \in \mathcal{S}$, and $\boldsymbol{\beta}^y_k$ is a parameter vector that weights $\mathbf{Z}_t$ and $\mathbf{X}_{o-1}$ in estimating the probability to observe outcome y when being in state s_k. The emission matrix then is as follows:

$$E(\boldsymbol{\beta}, \boldsymbol{Z}_t, \boldsymbol{X}_{o-1}) = \begin{bmatrix} \mu^{s_1}_{\boldsymbol{\beta}^{\text{No-hire}}_1 \boldsymbol{Z}_t \boldsymbol{X}_{o-1}} & \mu^{s_1}_{\boldsymbol{\beta}^{\text{Hire-negative}}_1 \boldsymbol{Z}_t \boldsymbol{X}_{o-1}} & \mu^{s_1}_{\boldsymbol{\beta}^{\text{Hire-positive}}_1 \boldsymbol{Z}_t \boldsymbol{X}_{o-1}} \\ \vdots & \vdots & \vdots \\ \mu^{s_K}_{\boldsymbol{\beta}^{\text{No-hire}}_K \boldsymbol{Z}_t \boldsymbol{X}_{o-1}} & \mu^{s_K}_{\boldsymbol{\beta}^{\text{Hire-negative}}_K \boldsymbol{Z}_t \boldsymbol{X}_{o-1}} & \mu^{s_K}_{\boldsymbol{\beta}^{\text{Hire-positive}}_K \boldsymbol{Z}_t \boldsymbol{X}_{o-1}} \end{bmatrix} , \tag{5}$$

where $\boldsymbol{\beta} = [\boldsymbol{\beta}^{\text{No-hire}}_1, \boldsymbol{\beta}^{\text{Hire-negative}}_1, \boldsymbol{\beta}^{\text{Hire-positive}}_1, \ldots, \boldsymbol{\beta}^{\text{Hire-positive}}_K]'$. Similar to the transition matrix T, and unlike popular HMM approaches that provide state-specific user-independent emission distributions (Murphy 2012, Sahoo et al. 2012, Hosseinzadeh Aghdam et al. 2015, Zhang et al. 2016b), the emission probabilities of the proposed framework depend not only on the current employer state, but also, on both the observed job-application $\mathbf{Z}_t$ and employer $\mathbf{X}_{o-1}$ characteristics.

HMM over time: Figure 4 shows the evolution of hiring-preferences of a single employer across R tasks that attract M job applications with outcomes $Y_1, ..., Y_t, ..., Y_M$, $Y_t \in \mathcal{Y}$. Aligned with the conventional presentation of probabilistic graphical models (Koller and Friedman 2009), shaded ellipses identify observed outcomes and characteristics, while clear ellipses identify latent hiring-preference states. The figure identifies that a new employer who joins the platform starts at state $s_k \in \mathcal{S}$ according to an initial probability vector $\boldsymbol{\pi}$ and remains to that state until the completion of the first task. From that state, the employer chooses which applicant to hire for the first task. Once the task is completed, the platform observes the performance of the hired applicant, and annotates observations $Y_1 \in \mathcal{Y}$ to $Y_t \in \mathcal{Y}$. The experience that the employer gained from the first

Figure 4 Interactions of the HMM framework across a sequence of completed tasks by a single employer



The Figure illustrates a sequence of M hiring choices ("No-hire," "Hire-negative," "Hire-positive") across R tasks of a single employer. During these choices, the focal employer transitions over states $S_1, S_2, ..., S_M$ according to the previously observed employer characteristics $\boldsymbol{X}_0, \boldsymbol{X}_1, ..., \boldsymbol{X}_{R-1}$ and parameters $\boldsymbol{\gamma}$ (Equation 2). For each received job application t, the employer emits a hiring choice Y_t based on the employer's current state S_t, employer characteristics $\mathbf{X}_{o-1}$, job-application characteristics $\boldsymbol{Z}_t$, and parameters $\boldsymbol{\beta}$ (Equation 4). The employer lands on state $S_1 \in \mathcal{S}$ according to an initial probability vector $\boldsymbol{\pi}$. Aligned with the conventional presentation of probabilistic graphical models (Koller and Friedman 2009), latent states form clear ellipses and observed characteristics form shaded ones.

task accumulates to vector $\boldsymbol{X}_1$. This vector affects the transition of the employer to a potentially new state s_l (Equation 2), which better describes the employer's hiring preferences. From that state, the employer evaluates new applications $t+1, ...,$ for task $o = 2$. This process continues until the completion of the R^{th} task.

Figure 4 further shows how job-application and employer characteristics ($\mathbf{Z}_t$,$\mathbf{X}_{o-1}$) affect the propensity of each hiring decision $Y_t \in \mathcal{Y}$: Through Equation 4, the model can predict the likelihood of each job-application to yield a "No-hire," a "Hire-negative," and a "Hire-positive" outcome. As we show later in Section 5.3 and Appendix G, estimation of these likelihoods yields performance-aware rankings of job-applicants that are not only likely to get hired, but they also have better chances of performing well.

Vectors $\boldsymbol{\pi}, \boldsymbol{\beta}$, and $\boldsymbol{\gamma}$ are parameters that the model needs to estimate. Appendix C presents the derivation of the likelihood and the subsequent process of estimating these parameters. Appendix F discusses the process of choosing the number of states K, and Appendix I provides implementation details with code examples from our Github repository (`https://github.com/repudime/gbu`).

4. Empirical context

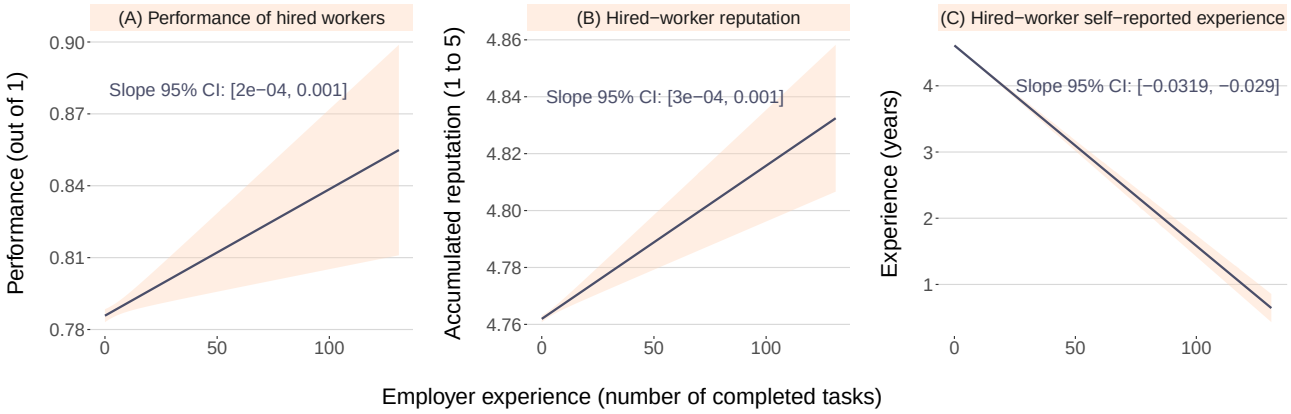
We build and evaluate the proposed framework on a set of real transactions from a major online labor market. The focal dataset includes 762,802 hiring decisions by 11,461 employers that led to 45,331 completed tasks and observed performance outcomes. The dataset tracks employers for twelve months and includes all their hiring decisions from the time that they joined the platform. The dataset also uses the internal log of the marketplace that takes snapshots of each worker's profile at the time of a job application.

Figure 5 illustrates characteristics of employer behavior in the focal dataset, which shows some initial model-free evidence that some employers might adjust their hiring preferences over time. Figure 5A shows that as employers gain experience, they hire workers who perform significantly better ($p < 0.05$). Similarly, Figures 5B and 5C show that over time, employers hire workers with higher reputation scores ($p < 0.05$) and lower self-reported expertise ($p < 0.05$). These trends show how employers' average behavior change over time. However, not all employers are average; Our approach allows employers to evolve or not evolve independently, hence better-capturing the current status of their hiring preferences. (Appendix D provides a more detailed description of the dataset and the focal market.)

Outcomes, employer, and job-application characteristics The HMM framework requires outcomes ($Y_t \in \mathcal{Y}$) as well as employer ($\mathbf{X}_{o-1}$) and job-application ($\mathbf{Z}_t$) characteristics (Figure 4).

Target variable (Y_t): Equation 1 models three outcomes: "Hire-positive," "Hire-negative," and "No-hire." The "No-hire" outcomes are readily available through the job applicants that employers did not choose to hire. The platform further labels a hiring outcome as "Hire-positive" when, upon the competition of the task, the employer privately rates the performance of the hired worker with a score greater or equal to 80% (i.e., performance threshold = 0.8). Otherwise, the platform labels a hiring outcome as "Hire-negative." (Appendix H illustrates the robustness of our approach across alternative performance thresholds.)

Figure 5 Model-free evidence suggests that some repeat employers adjust their hiring behaviors



Performance scores are private and only observed by the platform; Accumulated reputation scores are publicly available; Self-reported years of experience include the self-reported experience of the worker on the skills listed (not verified by the platform). CI: Confidence interval.

Employer and job-application characteristics ($\mathbf{X}_{o-1}, \mathbf{Z}_t$): Previous works on job-applicant recommendations have proposed various features that capture employer and job-application characteristics to predict the likelihood of each applicant to get hired (Kokkodis et al. 2015, Abhinav et al. 2017). Appendix E presents the set of 22 predictive variables we consider for this analysis; Table 1 shows their descriptive statistics. These statistics represent sequential observations of the same variables over time. We log-transform variables with long tails. We normalize (min-max) all variables to accelerate the convergence speed of the numeric optimization. (Note that the variables of Table 1 are context-specific. Appendix B shows how alternative contexts—such as restaurant recommendations—require different variable choices.)

5. Framework evaluation

The next paragraphs describe the training process of the HMM framework and compare its performance with alternative advanced recommender systems.

5.1. Nested cross validation

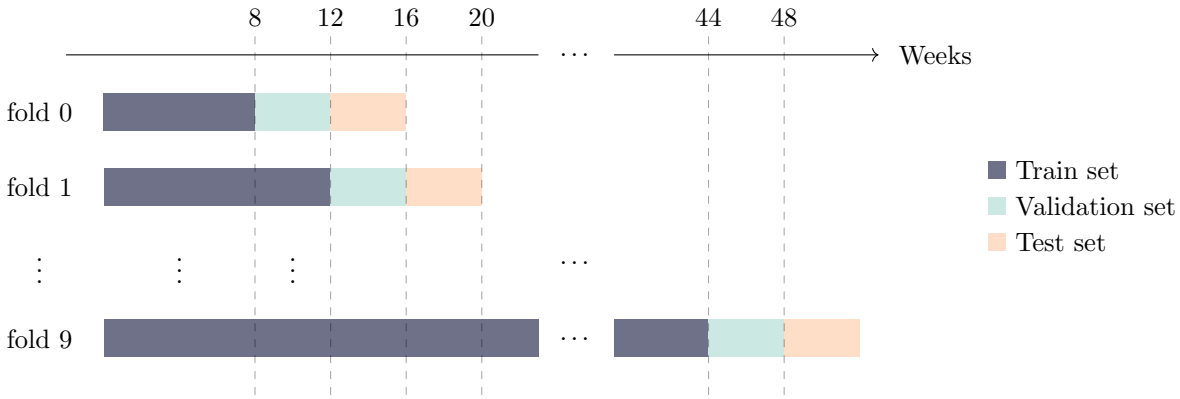
Our data has a time component (Figure 4). To ensure that we do not use information from the future to predict the past we evaluate all approaches through a nested cross validation setup (see

Table 1 Descriptive statistics

	Mean	Median	StD	Min	Max
Variable that creates outcomes Y_{it}					
Worker performance (privately reported to the platform)	0.79	0.89	0.28	0	1
Employer characteristics ($\boldsymbol{X}_{o-1}$)					
Employer money spent after completing $o-1$ tasks	178	4	841	0	36806
Employer most recent outcome $(o-1)$	0.62	1	0.49	0	1
Employer total competed tasks $(o-1)$	2.3	1	6	0	131
Employer number of fixed contracts after completing $o-1$ tasks	1.5	0	4.7	0	94
Employer number of hourly contracts after completing $o-1$ tasks	0.84	0	3.1	0	120
Employer total hire-positive outcomes after completing $o-1$ tasks	1.4	0	4	0	104
Employer fixed contract jobs with hire-positive outcomes after completing $o-1$ tasks	0.96	0	3.3	0	89
Employer hourly contract jobs with hire-positive outcomes after completing $o-1$ tasks	0.46	0	2.1	0	99
Job-application characteristics (snapshots of worker profiles at the time of application, $\boldsymbol{Z}_t$)					
Applicant completed work-hours	578	45	1472	0	37766
Skills IP	1	1	1.2	0	18
Applicant bid price	89	11	549	1	50000
Received order of application	26	15	32	0	291
Applicant accumulated reputation score (publicly available)	4.8	4.9	0.4	1	5
Applicant completed jobs	5	0	15	0	403
Employer-applicant countries PMI	−0.49	−0.46	0.4	−3.7	4.4
Applicant's self-reported years of experience (not verified by the platform)	4.5	4	4.1	0	30
Certifications PMI	2.3	2.1	1.7	−0.71	7
Invited	0.15	0	0.35	0	1
Contract type	0.48	0	0.5	0	1

Statistics describe time-varying sequential observations.

Figure 6 The nested cross validation process



For all models, parameter tuning and feature selection happens in the training and validation sets. Reported performance across all metrics is estimated on the previously unseen test sets.

for instance Cochrane 2018). Figure 6 describes this process. First, we assume 10 folds. Each fold includes a training, a validation, and a test set. Each training set includes only tasks that have been completed before the beginning of the validation set; each validation set includes only tasks that have been completed before the beginning of the test set. The necessary data transformations (e.g., normalization), hyperparameter tuning, and feature selection[8] happens in each training and validation sets, guaranteeing no data leakage in the previously unseen test sets. Once tuning is done, each algorithm uses both the training and validation sets of each fold to build a final model that is tested once on the previously unobserved test set.

5.2. Alternative recommender systems

Alternative approaches could also generate rankings of applicants. Prior work on recommending job-applicants (Kokkodis et al. 2015, Abhinav et al. 2017) along with other sequence-aware machine learning approaches provide a variety of alternative single-assessment recommenders. To benchmark the performance of the HMM framework against such advanced alternative models, we implement and compare the following systems:

◇ *Current reputation*: Upon completion of each job, workers receive a publicly available rating. These ratings accumulate to form each worker's public reputation. The most straightforward and

[8] For all models we use a step-forward feature selection process (Ferri et al. 1994). See Appendix I.1 for a detailed description of the this process.

transparent recommender system ranks applicants according to their accumulated reputation scores (Kokkodis et al. 2015, Abhinav et al. 2017).

◇ *Single-assessment recommenders*: Classification techniques can estimate the likelihood of an applicant getting hired and completing a job successfully. These systems model the relationship:

$$\Pr(Y_t|\boldsymbol{Z}_t, \boldsymbol{X}_{o-1}) \sim G(\boldsymbol{Z}_t, \boldsymbol{X}_{o-1})\,, \tag{6}$$

where:

- ○ Logistic regression: G represents the logistic sigmoid.
- ○ Support Vector Machines (SVM): G captures the relationships between vectors $\boldsymbol{X}_{o-1}, \boldsymbol{Z}_t$, and Y_t through Support Vector Machines.
- ○ Gradient boosting classification (XGBoost): G captures the relationships between vectors $\boldsymbol{X}_{o-1}, \boldsymbol{Z}_t$, and $Y_t \in \mathcal{Y}$ through gradient boosting classification (Chen and Guestrin 2016).
- ○ Random forest: G captures the relationships between vectors $\boldsymbol{X}_{o-1}, \boldsymbol{Z}_t$, and $Y_t \in \mathcal{Y}$ through a multitude of decision trees (Ho 1998).
- ○ Recurrent neural networks (LSTM): G captures sequence-aware relationships between vectors $\boldsymbol{X}_{o-1}, \boldsymbol{Z}_t$, and $Y_t \in \mathcal{Y}$ through Long Short Term Memory networks (Hochreiter and Schmidhuber 1997).

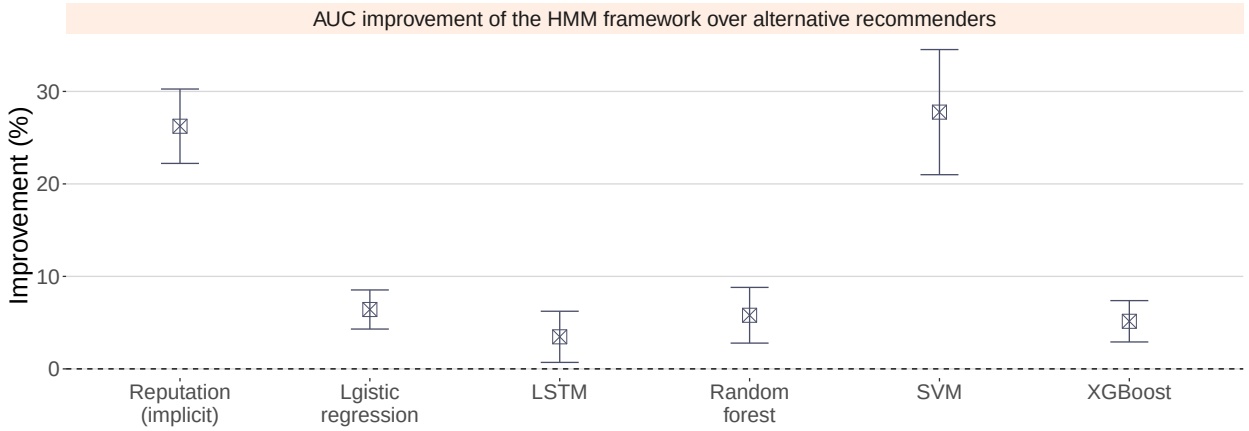
Appendix F describes the grid search process we follow for tuning all recommenders.

5.3. Results

Our goal is to rank job applicants according to their likelihood of getting hired and performing well. The next paragraphs benchmark the performance of the HMM framework against alternative recommender systems across four ranking measures:

- Job-applicant rankings for all employers,
- Job-applicant rankings for repeat employers,
- Performance of top-ranked applicants,
- Performance of top-ranked applicants within tasks.

Figure 7 Comparison of alternative recommenders: job-applicant rankings for all employers

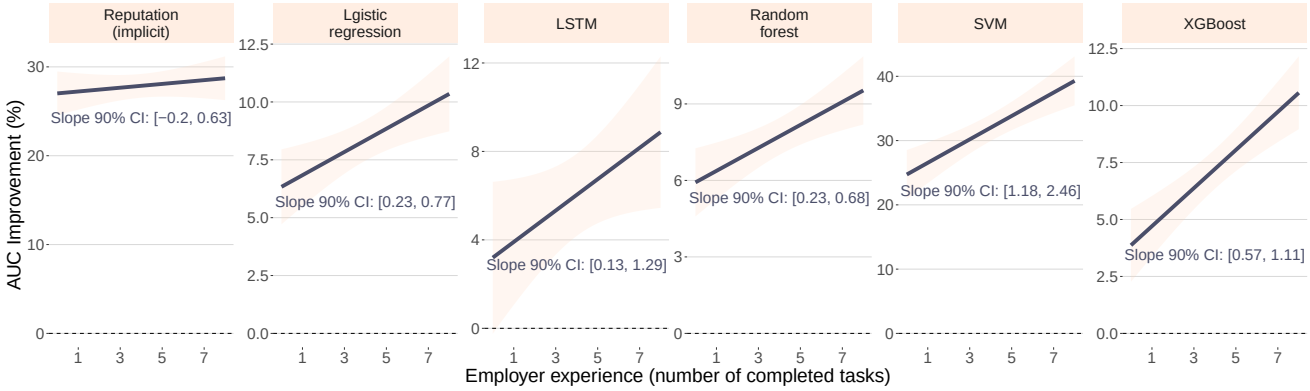


The proposed approach ranks job applicants according to their likelihood of getting hired and performing well significantly better (at least $p < 0.1$) than the alternative systems. The $y-$axis shows the 10-fold nested cross-validated AUC percentage improvement of the proposed framework over the $x-$axis recommender systems. Error bars show 90% confidence intervals. Implicit identifies implicit-feedback systems ("No-hire," "Hire"). Confidence intervals are estimated across the improvements of the 10 folds.

5.3.1. Job-applicant rankings for all employers: The first ranking measure that we use is the area under the ROC curve (AUC; see Provost and Fawcett 2001). Our AUC score of interest measures the probability that a model ranks applicants who are hireable and likely to perform well ("Hire-positive") higher than applicants who are not likely to get hired ("No-hire"), or they are likely to get hired and perform poorly ("Hire-negative"; see p.864 of Fawcett 2006). (Appendix G shows alternative rankings that trade-off "Hire-positive" with "Hire-negative" outcomes.)

Figure 7 compares the AUC performance of the HMM framework with that of the alternative recommender systems. The $y-$axis shows the average percentage AUC-improvement of the HMM framework compared to the $x-$axis alternative system (Table 3 in Appendix M shows the AUC scores for each fold and approach). The error bars show the nested 10-fold cross-validated 90% confidence intervals. The Figure shows that the performance of the HMM framework is significantly ($p < 0.05$) better than the performance of all alternative systems. The percentage improvement over the existing single-assessment models (Kokkodis et al. 2015, Abhinav et al. 2017) ranges, on average, between 4% and 28%. The most competitive (with the HMM framework) models are the LSTM,

Figure 8 Comparison of alternative recommenders: job-applicant rankings for repeat employers



As employers hire more workers, the proposed approach's improvement over the alternative recommender systems increases (positive slope). This is due to the task-specific employer transitions of Equation 2. The $y-$axis shows the 10-fold nested cross-validated AUC percentage improvement of the proposed framework over each alternative recommender system. The $x-$axis captures employer experience in terms of hired-workers (completed tasks). CI stands for confidence interval. Implicit identifies implicit-feedback systems ("No-hire," "Hire"). Confidence intervals are estimated across the improvements of the 10 folds.

and XGBoost, none of which has been previously proposed for job-applicant recommendations. Yet, even against these powerful approaches, the unique structure that allows task-specific transitions of our HMM framework allows it to perform on average 4% to 6% ($p < 0.05$) better. As we discuss next, the outperformance of our HMM follows an increasing trend when models recommend job-applicants to repeat employers.

5.3.2. Job-applicant rankings for repeat employers: The real power of the HMM approach resides in modeling employers who hire workers repeatedly over different tasks as these employers are more likely to adjust their hiring preferences and evolve across the HMM states (see also Appendix L that quantifies employer transitions). To test how each approach performs in terms of such *repeat employers*, we estimate the AUC scores for hiring decisions after employers complete n or more tasks (AUC-n). Specifically, for any given n, we consider only hiring decisions that occurred after each decision-making employer has hired and evaluated $n-1$ workers. For instance, if a given employer appears in a test set with a sequence of four tasks and their respective hiring choices and outcomes, then AUC-2 would consider that employer's second, third, and fourth

tasks' choices; AUC-3 would consider the third and fourth tasks' choices; AUC-4 would consider the fourth task's choices; for $n > 4$, the estimation of AUC-n would exclude this employer.

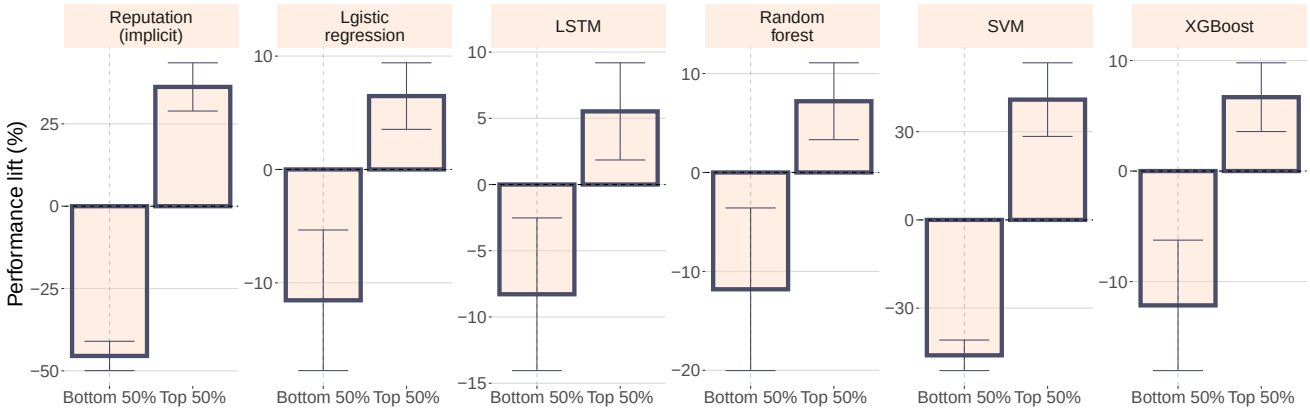
Figure 8 shows the 10-fold cross-validated AUC-n percentage improvement scores of the HMM approach over the alternative recommender approaches as employers hire workers across different job tasks. The $x-$axis captures the number of previously hired workers (completed collaborations) for each employer. Intuitively, as employers hire and manage more workers, the HMM should provide better recommendations as its state structure allows employers to adjust their hiring-preferences individually. As a result, the HMM improvement over the alternative recommenders should increase over time.

Figure 8 shows this expected improvement increase over time. Across all alternative recommender systems, the slope of the linear regression of percentage improvement over the number of completed jobs is positive (at least $p < 0.1$; Figure 8 shows the 90% confidence intervals for the slopes of each improvement line; for the reputation baseline the slope is positive but not statistically significant). This increasing trend illustrates that allowing employers to evolve according to the feature vector $\mathbf{X}_{o-1}$ and Equation 2 captures employer changing hiring preferences and yields progressively better results compared with the alternative recommender systems that do not capture employer evolution as accurately. (Note that sequence-aware models such as LSTM do not encode the task-specific sequences of Equation 2; Instead, the sequences these systems model identify sequential dependencies across all hiring outcomes: "No-hire," "Hire-negative," and "Hire-positive." Appendix O discusses and conceptually explains why our HMM approach outperforms the LSTM recommender.)

5.3.3. Performance of top-ranked applicants: Given a ranking of candidates, perhaps the most crucial performance metric is whether the top ranked candidates actually get hired and perform better than the bottom-ranked ones. To evaluate this behavior, we compare all alternative approaches by estimating the performance lift as follows:

$$\text{Performance lift}(p) = \frac{\#(\text{HMM "Hire-positive"} \in p) - \#(\text{Alternative "Hire-positive"} \in p)}{\#(\text{Alternative "Hire-positive"} \in p)}, \quad (7)$$

Figure 9 Comparison of alternative recommenders: performance of top-ranked applicants



Compared with alternative recommenders, the proposed approach ranks applicants who are more likely to get hired and perform well in the top 50^{th} percentiles, while it ranks applicants who are less likely to get hired and perform well in the bottom 50^{th} ones. The $y-$axis shows the 10-fold nested cross-validated performance lift (Equation 7). Error bars show 90% confidence intervals. Implicit identifies implicit-feedback systems ("No-hire," "Hire"). Confidence intervals are estimated across the improvements of the 10 folds.

where "Alternative" captures a ranking by an alternative recommender system and $p \in \{$Bottom 50%, Top 50%$\}$.

Intuitively, as we move from the bottom-ranked to the top-ranked job applicants the performance lift should increase. If ranked applicants by the HMM framework purely outperform ranked applicants from an alternative approach, the estimated performance lift should be negative for the bottom 50^{th} percentile and positive for the top 50^{th} percentile.

Figure 9 shows exactly this behavior for all alternative recommenders. The $y-$axis shows the nested 10-fold cross-validated performance lift; the $x-$axis separates applicants into the bottom and the top 50^{th} percentiles according to their predicted likelihood of getting hired and performing well. Indeed, the observed performance lift is negative ($p < 0.05$) for all bottom-ranked job applicants, and positive ($p < 0.05$) for all the top-ranked ones. Overall, compared with all alternative recommenders, the HMM framework presents significantly ($p < 0.05$) better applicants in the top ranking positions and significantly worse applicants in the bottom ones.

5.3.4. Performance of top-ranked applicants within tasks: The ranking (AUC, AUC-n) and performance lift evaluations capture a model's behavior across all available tasks; they do not,

however, evaluate how each algorithm performs within tasks. To do so, we rank applicants within each task, and we measure the actual performance of the top-ranked applicants as follows:

$$\text{Within-task hired-applicant performance}(k) = \frac{\frac{\#(\text{“Hire-positive”} \in \text{top}-k \text{ recommended})}{\#(\in \text{top}-k \text{ hired})}}{\frac{\#(\text{“Hire-positive”} \notin \text{top}-k \text{ recommended})}{\#(\notin \text{top}-k \text{ hired})}}, \tag{8}$$

where “$\in$ top$-k$” captures recommended applicants in the top$-k$ who got hired and “$\notin$ top$-k$” captures recommended applicants not in the top$-k$ who get hired. Conceptually, this performance ratio measures how much better the top-recommended job applicants perform compared with the rest non-top-ranked applicants who got hired.

Figure 10 shows the results for $k = 3$[9] in a similar form to Figure 7: The $y-$axis captures the 10-fold cross-validated performance improvement of the HMM framework over the $x-$axis recommender. The 90% confidence intervals clearly show that the within-task performance of the HMM framework is statistically significantly ($p < 0.05$) better than all alternative recommender systems. The improvement ranges from an average of 15% over the LSTM to 43% over the SVM.

5.4. Robustness and generalizability

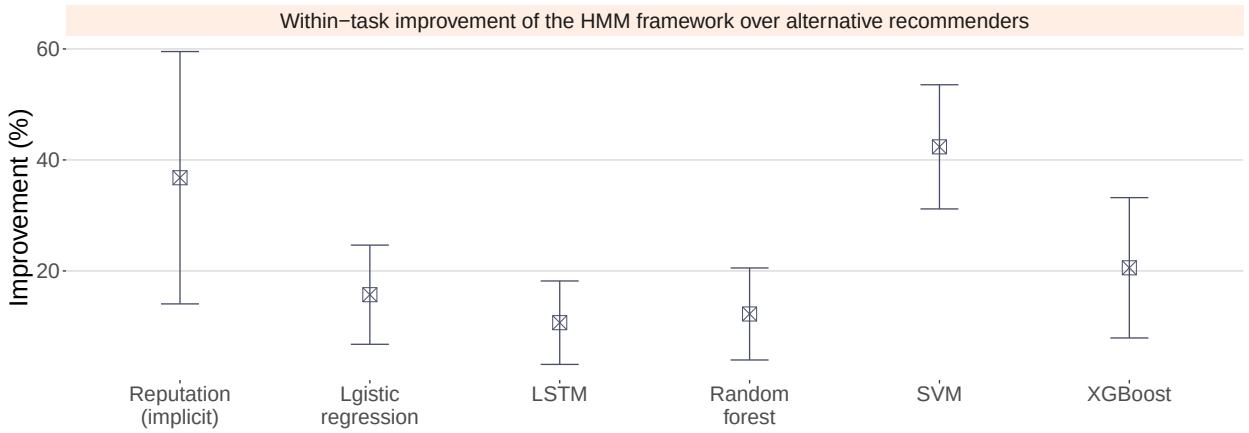
Multiple appendices illustrate the robustness and generalizability of the HMM framework:

- ◇ **Alternative ranking mechanisms:** The proposed approach ranks job applicants according to their likelihood of getting hired and performing well; it ignores, however, the likelihood of each candidate to get hired and perform poorly (“Hire-negative”). Given that employers who hire poor-performing workers are likely to exit the market (Tripp and Grégoire 2011), alternative ranking mechanisms could trade-off “Hire-positive” outcomes for fewer “Hire-negative” ones. Such approaches could minimize the likelihood of “Hire-negative” outcomes while keeping the likelihood of observing “Hire-positive” outcomes at sufficient levels. Appendix G presents such alternative ranking approaches.

- ◇ **Robustness to alternative thresholds:** The platform’s choice of performance threshold equal to 0.8 (80%) might appear ad-hoc. Appendix H illustrates that our results are robust across alternative thresholds of separating “Hire-positive” from “Hire-negative” outcomes.

[9] Results are robust for $k = 4, 5$.

Figure 10 Comparison of alternative recommenders: performance of top-ranked applicants within tasks



The proposed approach ranks job applicants within tasks according to their likelihood of getting hired and performing well significantly better (p <0.05) than the alternative systems. The $y-$axis shows the nested 10-fold cross-validated within-task performance (Equation 8) improvement of the proposed framework over the $x-$axis recommender systems. Error bars show 90% confidence intervals. Implicit identifies implicit-feedback systems ("No-hire," "Hire"). Confidence intervals are estimated across the improvements of the 10 folds.

◇ **Generalizability:** Our approach can generalize to other single-assessment and few-assessment contexts (Figure 1). One such context is restaurant recommendations on reputation platforms such as TripAdvisor and Yelp. In these platforms, user preferences evolve as reviewers grow older. At the same time, restaurants change significantly, as they go through renovations, update their menus, and hire new stuff. As a result, both the users (reviewers) and the recommended items (restaurants) change. In this few-assessment context, we can build recommender systems that rank restaurants within a location according to their likelihood of getting reviewed positively (i.e., a user will visit them, self-select to review them, and review them positively). Appendix B implements our approach and compares its performance with alternative recommender systems on a set of TripAdvisor restaurant reviews: the proposed approach significantly outperforms ($p < 0.05$) the alternative recommender systems in this different context, providing evidence that our framework generalizes in contexts where both the recommended items and the user preferences might change over time.

◇ **Comparison with many-assessment recommenders:** Section 2.2 illustrated the characteristics of job-applicant recommendations that make it a single-assessment context. Appendix A provides additional details and examples on why many-assessment systems will likely perform poorly in this context. Appendix A.2 implements eight such systems, and illustrates in practice that they perform poorly compared with our proposed framework (Figure 11); Figure 16 shows that their underperformance extends in the alternative, restaurant recommendation context.

Overall, by encapsulating the design principles discussed in Section 2.4, the proposed framework performs particularly well, especially among repeat employers. Because no alternative approach can model employer sequences at the task level and at the same time provide "No-hire" outcomes, our HMM framework ends up providing significantly better job applicant recommendations.

6. Discussion

This work conceptualized that job-applicant recommenders in online labor markets should be single-assessment systems that are performance-aware and sequence-aware. Based on these principles, an HMM framework modeled performance-aware emissions and allowed employer hiring preferences to change. Comparison of the proposed framework with various alternative recommender systems highlighted the advantages of the three design principles. The empirical evaluation further showed that repeat employers benefit the most from our approach, as these employers receive personalized sequence-aware recommendations by following their distinct hiring-preference paths. Application of the HMM framework in a restaurant recommendation context showed its generalizability in environments where both the recommended items and the user preferences change.

6.1. Research contributions

Given the projected growth of the number of online workers in the coming years (Agile-1 2016, Sundararajan 2016), accurate job-applicant recommendations could be a significant factor of the ultimate reach of online work. This paper is the first to outline the limitations of existing recommender systems and explain why such systems underperform when ranking job-applicants according to their likelihood of getting hired and performing well. By identifying and addressing these

shortcomings, this work provides significantly enhanced recommendations of hireable and capable job applicants, especially for repeat employers.

From a design perspective, this work conceptualizes three principles that job-applicant recommender systems should have. First, they need to be *single-assessment* systems and facilitate the modeling of uniquely recommended items. Second, they need to be *performance-aware*; Our work is the first to formulate the job-application recommendation problem as a trinary classification problem that includes both implicit (the choice to hire) and explicit (performance of the hired worker) feedback. Because this formulation separates successful from unsuccessful collaborations, it does not reinforce all prior hiring decisions, but instead, it learns from unsuccessful collaborations and identifies job applicants that have a high propensity of performing well. Third, job-applicant recommender systems should be *sequence-aware*, allowing employer hiring preferences to evolve over repeated collaborations with remote workers. Our work is the first to capture employer changing hiring preferences through task-specific customized transition probabilities (Equation 2).

The unique design of the HMM framework extends the rich literature of existing recommender systems (Figure 3). Compared with previous HMM-based systems (Sahoo et al. 2012, Hosseinzadeh Aghdam et al. 2015, Zhang et al. 2016b), the proposed approach allows the (1) modeling of uniquely evaluated items, (2) task-specific transition probabilities (Equation 2), (3) history-driven stochastic transitions (affected by vector $\mathbf{X}_{t-1}$), and (4) item-driven emission probabilities (affected by vectors $\mathbf{X}_{t-1}, \mathbf{Z}_t$). Two empirical contexts (job-applicant and restaurant recommendations) show in practice the significance of these unique characteristics (Section 5.3, Appendices B, J, N).

The conceptualizations of performance-awareness and sequence-awareness can transfer to other types of recommender systems in online labor markets and crowdsourcing. In particular, both automated recruiters and task recommenders could adapt to be performance-aware and sequence-aware. Automated recruiters could include observed outcomes on top of hireability requirements while allowing both the worker's experience and the employer's hiring preferences to evolve. Similarly, task recommenders can incorporate performance when they allocate jobs to available workers, while they can also allow worker abilities to evolve. Future work on these two types of recommender systems can instill these ideas that will likely improve performance and reduce adverse outcomes.

6.2. Contributions to practice and generalizability

This paper provides a detailed guideline along with sample code[10] for market practitioners that are interested in developing performance-aware and sequence-aware recommender systems. Specifically, it addresses empirical challenges that include the conceptualization, modeling, and estimation of the framework:

- ◇ HMM architecture: Section 3 describes how practitioners can conceptualize and formulate a suitable structure for an HMM that allows a series of observed signals to shape the transition and emission probabilities of employers with different hiring preferences. It further illustrates how transitions can be task-specific (Equation 2), which is conceptually sound when modeling evolving hiring-preferences.
- ◇ Parameter estimation: Appendix C guides practitioners through the derivation of the global likelihood of the model and the estimation process of all the parameters. Appendix F presents the process of selecting an appropriate configuration for the HMM.
- ◇ Evaluation: Section 5.3 guides practitioners in generating meaningful evaluation metrics that compare the performance of various recommender systems in terms of ranking candidates according to their likelihood of performing well.

These methodological contributions generalize beyond the focal context of online work. The HMM framework can be adjusted and successfully implemented in any context where (1) recommended items change or receive very few user evaluations, (2) user preferences evolve, and (3) choice sets do not significantly overlap. Offline job-applicant recommendations form one such context: by analyzing career trajectories of LinkedIn users, sequence-aware frameworks can identify performance outcomes (e.g., through "upward trajectory" or "downward trajectory") while allowing employer hiring preference to change as job requirements change. Recommending restaurants is another such context: Appendix B shows that reputation platforms such as Yelp and TripAdvisor could use a similar framework to develop performance-aware and sequence-aware restaurant

[10] See our Github repository: https://github.com/repudime/gbu

recommendations. Similarly, travel platforms such as Airbnb can also use the proposed framework as both the recommended items (rented apartments) and user preferences evolve: Our approach models these sequential changes while controlling for implicit (choice of an apartment to stay) and explicit (rating of the chosen apartment) feedback. Figure 1 identifies multiple other suitable contexts, including recommending skills to learn (both recommended skills and users change over time) and courses to take (courses evolve based on teachers and year; students also evolve).

6.3. Implications for platforms, workers, employers, and the future of work

Online labor markets stand to benefit through implementing our approach as job-applicant recommendations that lead to successful outcomes help (1) workers to differentiate, (2) employers to make better-informed and faster (reduced search cost; see Bakos 1997) decisions, and (3) the markets to increase their transaction efficiency, which in turn results in increased revenue and customer satisfaction. Through recommendations of appropriate job applicants, some low-quality workers that currently flood the market might get marginalized. This marginalization can create room for potentially high-quality workers to pursue tasks that they see fit.

Furthermore, employers who make faster decisions that lead to productive collaborations are more likely to keep using the platform (Jerath et al. 2011). More successful employers will create more job openings, which in turn will attract more workers and widen the reach of the online labor economy. Besides, through improved recommendations, markets will increase their transaction efficiency as more openings will reach contracts (recall that currently, as many as 60% of job openings never reach a contract; see Zheng et al. 2015).

The performance of the proposed approach in terms of recommending candidates to repeat employers (Section 5.3, Figure 8) is of particular importance to market managers. These employers represent a significant client segment for online labor markets. Providing them with relevant recommendations reduces the number of adverse outcomes and increases the employers' likelihood to keep participating in the marketplace (Tripp and Grégoire 2011).

Finally, through the proposed framework, platforms can better understand how employers evolve. By observing employer paths across various latent states of changing hiring preferences, platform

managers have opportunities to intervene to groups of employers that are likely to drop out (e.g., through discount offers, or through helping them to make better hiring choices).

6.4. Limitations

Our approach learns to reinforce previously observed successful hiring behavior. As a result, it can miss applicants with characteristics that might be really good and valuable but have not been previously chosen by employers (i.e., "No-hire" choices). Markets can explore this possibility by developing rankings that assign a higher weight to "No-hire" outcomes (similar to the examples presented in Appendix G). By introducing more "No-hire" candidates into the curated top-ranked applicant recommendations, platforms can empirically measure the outcomes of hired workers who would (probabilistically) not have been hired otherwise, retrain their systems, and eventually learn to provide more holistic and better recommendations.

Another limitation of our work is that it does not investigate the underlying mechanisms that drive employers to adjust their preferences. In fact, our approach is agnostic to the factors that cause employers to transition to new states. Instead, our framework is being proactive and expects that some employers will likely evolve over time. Hence, it provides the infrastructure for modeling these changing employers (see Appendix L) while allowing employers who do not change to remain in the same state. Regardless, given that our work shows empirically that for some employers hiring preferences change, future research can properly investigate the mechanisms that such changes happen.

6.5. Conclusion

Conclusively, this work implements a single-assessment job-applicant recommendation framework that is both performance-aware and sequence-aware. Application of this framework in a large dataset of hiring decisions from an online labor market shows that it provides recommendations of job applicants who are both hireable and likely to perform well. The framework's structure generalizes to other contexts where the recommended items either change or receive very few evaluations, and where user preferences evolve. As a result, its deployment in different types of online platforms could have significant implications for workers, employers, businesses, and the future of work.

References

Abhinav, Kumar, Alpana Dubey, Sakshi Jain, Gurdeep Virdi, Alex Kass, Manish Mehta. 2017. Crowdadvisor: A framework for freelancer assessment in online marketplace. *International Conference on Software Engineering*. 93–102.

Adomavicius, Gediminas, Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Transactions on Knowledge and Data Engineering* **17** 734–749.

Agile-1. 2016. Gig economy. http://www.hrotoday.com/wp-content/uploads/2016/07/Whitepaper_Agile2016-single.pdf. [Online; accessed 28-November-2020].

Akerlof, George A. 1970. The market for "lemons": Quality uncertainty and the market mechanism. *The Quarterly Journal of Economics* **84** 488–500.

Atkins, Olivia. 2019. Peopleperhour plans on growing community with cross-media campaign. https://www.thedrum.com/news/2019/10/23/peopleperhour-plans-growing-community-with-cross-media-campaign. Accessed: 08-December-2019.

Autor, David H. 2001. Wiring the labor market. *Journal of Economic Perspectives* **15** 25–40.

Ba, Sulin, Paul A. Pavlou. 2002. Evidence of the effect of trust building technology in electronic markets: Price premiums and buyer behavior. *MIS Quarterly* 243–268.

Baba, Yukino, Kei Kinoshita, Hisashi Kashima. 2016. Participation recommendation system for crowdsourcing contests. *Expert Systems with Applications* **58** 174–183.

Bakos, Yannis. 1997. Reducing buyer search costs: Implications for electronic marketplaces. *Management Science* **43** 1676–1692.

Balabanović, Marko, Yoav Shoham. 1997. Fab: content-based, collaborative recommendation. *Communications of the ACM* **40** 66–72.

Billsus, Daniel, Michael J. Pazzani. 1998. Learning collaborative information filters. *International Conference on Machine Learning*, vol. 98. 46–54.

Billsus, Daniel, Michael J. Pazzani. 2000. User modeling for adaptive news access. *User Modeling and User-Adapted Interaction* **10** 147–180.

Bishop, M. Christopher. 2006. *Pattern Recognition and Machine Learning*. Springer.

Borisyuk, Fedor, Liang Zhang, Krishnaram Kenthapadi. 2017. LiJAR: A system for job application redistribution towards efficient career marketplace. *International Conference on Knowledge Discovery and Data Mining*. 1397–1406.

Bousbahi, Fatiha, Henda Chorfi. 2015. MOOC-Rec: A case based recommender system for MOOCs. *Procedia Social and Behavioral Sciences* **195** 1813–1822.

Breese, S. John, David Heckerman, Carl Kadie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. *Conference on Uncertainty in Artificial Intelligence*. 43–52.

Brier, Elisabeth, Rich Pearson. 2018. Upwork's SVP of marketing explains what it takes to perfect an offering that relies on people. https://www.prnewswire.com/news-releases/snagajob-appoints-former-upwork-ceo-to-board-of-directors-300417689.html. [Online; accessed 28-November-2020].

Brownlee, Jason. 2017. Stacked long short-term memory networks. https://machinelearningmastery.com/stacked-long-short-term-memory-networks/. [Online; accessed 28-November-2020].

Brownlee, Jason. 2018. How to develop lstm models for time series forecasting. https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/. [Online; accessed 28-November-2020].

Brynjolfsson, Erik, Yu Hu, Duncan Simester. 2011. Goodbye pareto principle, hello long tail: The effect of search costs on the concentration of product sales. *Management Science* **57** 1373–1386.

Byrd, H. Richard, Peihuang Lu, Jorge Nocedal, Ciyou Zhu. 1995. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing* **16** 1190–1208.

Carr, M. Scott. 2003. Note on online auctions with costly bid evaluation. *Management Science* **49** 1521–1528.

Chen, Pei-Yu, Shin-yi Wu, Jungsun Yoon. 2004. The impact of online recommendations and consumer feedback on sales. *International Conference on Information Systems*.

Chen, Tianqi, Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. *International Conference on Knowledge Discovery & Data Mining*. ACM, 785–794.

Cochrane, Courtney. 2018. Time series nested cross-validation. https://towardsdatascience.com/time-series-nested-cross-validation-76adba623eb9. [Online; accessed 03-March-2021].

Delgado, Joaquin, Naohiro Ishii. 1999. Memory-based weighted majority prediction. *Special Interest Group on Information Retrieval Workshop on Recommender Systems*.

Dimoka, Angelika, Yili Hong, Paul A. Pavlou. 2012. On product uncertainty in online markets: Theory and evidence. *MIS Quarterly* **36** 395–426.

Eckhardt, Karsten. 2018. Choosing the right hyperparameters for a simple lstm using keras. https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046. [Online; accessed 28-November-2020].

Färber, Frank, Tim Weitzel, Tobias Keim. 2003. An automated recommendation approach to selection in personnel recruitment. *Americas Conference on Information Systems*.

Farooque, Umar, Bilal Khan, Abidullah Bin Junaid, Akash Gupta. 2014. Collaborative filtering based simple restaurant recommender. *International Conference on Computing for Sustainable Global Development*. IEEE, 495–499.

Fawcett, Tom. 2006. An introduction to roc analysis. *Pattern recognition letters* **27** 861–874.

Ferri, FJ, Pavel Pudil, Mohamad Hatef, Josef Kittler. 1994. Comparative study of techniques for large-scale feature selection. *Machine Intelligence and Pattern Recognition*, vol. 16. Elsevier, 403–413.

Fleder, Daniel, Kartik Hosanagar. 2009. Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. *Management Science* **55** 697–712.

Freelancers-union. 2017. Freelancing in america. https://s3-us-west-1.amazonaws.com/adquiro-content-prod/documents/Infographic_UP-URL_2040x1180.pdf. [Online; accessed: 02-December-2019].

Garcin, Florent, Christos Dimitrakakis, Boi Faltings. 2013. Personalized news recommendation with context trees. *Conference on Recommender Systems*. ACM, 105–112.

Geva, Tomer, Maytal Saar-Tsechansky. 2016. Who's a good decision maker? Data-driven expert worker ranking under unobservable quality. *International Conference on Information Systems*.

Gong, Yuyun, Qi Zhang. 2016. Hashtag recommendation using attention-based convolutional neural network. *International Joint Conference on Artificial Intelligence*. 2782–2788.

Goswami, Anjan, Fares Hedayati, Prasant Mohapatra. 2014. Recommendation systems for markets with two sided preferences. *International Conference on Machine Learning and Applications*. 282–287.

Guasch, J. Luis, Stéphane Straub, Jean-Jacques Laffont. 2003. *Renegotiation of concession contracts in Latin America*. The World Bank.

Guo, Xue, Jing Gong, Paul Pavlou. 2017. Call for bids to improve matching efficiency: Evidence from online labor markets. *International Conference on Information Systems*.

He, Qi, Daxin Jiang, Zhen Liao, Steven CH Hoi, Kuiyu Chang, Ee-Peng Lim, Hang Li. 2009. Web query recommendation via sequential query prediction. *International Conference on Data Engineering*. 1443–1454.

He, Ruining, Julian McAuley. 2016. Fusing similarity models with Markov chains for sparse sequential recommendation. *International Conference on Data Mining*. 191–200.

Ho, Tin Kam. 1998. The random subspace method for constructing decision forests. *Transactions on Pattern Analysis and Machine Intelligence* **20** 832–844.

Hochreiter, Sepp, Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* **9** 1735–1780.

Hossain, Md Sabir, Mohammad Shamsul Arefin. 2019. Development of an intelligent job recommender system for freelancers using clients feedback classification and association rule mining techniques. *Journal of Software* **14** 312–339.

Hosseinzadeh Aghdam, Mehdi, Negar Hariri, Bamshad Mobasher, Robin Burke. 2015. Adapting recommendations to contextual changes using hierarchical hidden Markov models. *Conference on Recommender Systems*. 241–244.

Hsueh, Sue-Chen, Ming-Yen Lin, Chien-Liang Chen. 2008. Mining negative sequential patterns for e-commerce recommendations. *Asia-Pacific Services Computing Conference*. 1213–1218.

Institute of Business Value. 2019. The enterprise guide to closing the skills gap. https://www.ibm.com/downloads/cas/EPYMNBJA. [Online; accessed 02-December-2019].

Jannach, Dietmar, Lukas Lerche, Michael Jugovac. 2015. Adaptation and evaluation of recommendations for short-term shopping goals. *Conference on Recommender Systems*. 211–218.

Jerath, Kinshuk, Peter S. Fader, G.S. Bruce Hardie. 2011. New perspectives on customer "death" using a generalization of the pareto/nbd model. *Marketing Science* **30** 866–880.

Kantor, B. Paul, Lior Rokach, Francesco Ricci, Bracha Shapira. 2011. *Recommender systems handbook*. Springer.

Keras, API. 2021. Dense layer. https://keras.io/api/layers/core_layers/dense/. [Online; accessed 28-November-2020].

Klazema, Michael. 2018. Why is hiring the right employee so difficult? *Workplaces* [Online; accessed: 28-November-2020].

Kokkodis, Marios, Panagiotis G. Ipeirotis. 2014. The utility of skills in online labor markets. *International Conference on Information Systems*.

Kokkodis, Marios, Panagiotis G. Ipeirotis. 2020. Demand-aware career path recommendations: A reinforcement learning approach. *Management Science* (forthcoming).

Kokkodis, Marios, Panagiotis Papadimitriou, Panagiotis G. Ipeirotis. 2015. Hiring behavior models for online labor markets. *International Conference on Web Search and Data Mining*. 223–232.

Koller, Daphne, Nir Friedman. 2009. *Probabilistic graphical models: Principles and techniques*. MIT press.

Koren, Yehuda. 2009. Collaborative filtering with temporal dynamics. *International Conference on Knowledge Discovery and Data Mining*. 447–456.

Koren, Yehuda, Robert Bell, Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* **42** 30–37.

Kula, Maciej. 2017. Neural networks for recommendations. https://bit.ly/2UD4wH4. [Online; accessed: 02-December-2019].

Kula, Maciej. 2018. Deep recommender models using pytorch. https://github.com/maciejkula/spotlight. [Online; accessed: 02-December-2019].

Kurashima, Takeshi, Tomoharu Iwata, Takahide Hoshide, Noriko Takaya, Ko Fujimura. 2013. Geo topic model: Joint modeling of user's activity area and interests for location recommendation. *International Conference on Web Search and Data Mining*. 375–384.

Kurup, R. Ayswarya, G.P. Sajeev. 2017. Task recommendation in reward-based crowdsourcing systems. *International Conference on Advances in Computing, Communications and Informatics*. 1511–1518.

Lauren, Dyke. 2017. Snagajob appoints former Upwork CEO to board of directors. https://www.prnewswire.com/news-releases/snagajob-appoints-former-upwork-ceo-to-board-of-directors-300417689.html. [Online; accessed 28-November-2020].

Li, Jialing, Li Li. 2014. A location recommender based on a hidden Markov model: Mobile social networks. *Journal of Organizational Computing and Electronic Commerce* **24** 257–270.

Li, Sheng, Jaya Kawale, Yun Fu. 2015. Deep collaborative filtering via marginalized denoising auto-encoder. *International on Conference on Information and Knowledge Management*. 811–820.

Lian, Defu, Vincent W. Zheng, Xing Xie. 2013. Collaborative filtering meets next check-in location prediction. *International Conference on World Wide Web*. 231–232.

Lin, H. Christopher, Ece Kamar, Eric Horvitz. 2014. Signals in the silence: Models of implicit feedback in a recommendation system for crowdsourcing. *Conference on Artificial Intelligence*. 908–914.

Malinowski, Jochen, Tobias Keim, Oliver Wendt, Tim Weitzel. 2006. Matching people and jobs: A bilateral recommendation approach. *Hawaii International Conference on System Sciences*, vol. 6. 137c–137c.

Mao, Ke, Ye Yang, Qing Wang, Yue Jia, Mark Harman. 2015. Developer recommendation for crowdsourced software development tasks. *Symposium on Service-Oriented System Engineering*. 347–356.

Meyer, Frank. 2012. Recommender systems in industrial contexts. *arXiv preprint* 1203.4487.

Moling, Omar, Linas Baltrunas, Francesco Ricci. 2012. Optimal radio channel recommendations with explicit and implicit feedback. *Conference on Recommender Systems*. ACM, 75–82.

Murphy, Kevin P. 2012. *Machine learning: A probabilistic perspective*. The MIT Press.

Natarajan, Nagarajan, Donghyuk Shin, Inderjit S Dhillon. 2013. Which app will you use next? Collaborative filtering with interactional context. *Conference on Recommender Systems*. 201–208.

Pallais, Amanda. 2014. Inefficient hiring in entry-level labor markets. *American Economic Review* **104** 3565–3599.

Pardos, A. Zachary, Steven Tang, Daniel Davis, Christopher Vu Le. 2017. Enabling real-time adaptivity in MOOCs with a personalized next-step recommendation framework. *Conference on Learning@ Scale*. 23–32.

Patel, Bharat, Varun Kakuste, Magdalini Eirinaki. 2017. CaPaR: A career path recommendation framework. *International Conference on Big Data Computing Service and Applications*. 23–30.

Pathak, Bhavik, Robert Garfinkel, Ram D Gopal, Rajkumar Venkatesan, Fang Yin. 2010. Empirical analysis of the impact of recommender systems on sales. *Journal of Management Information Systems* **27** 159–188.

Pelletier, Adeline, Catherine Thomas. 2018. Information in online labour markets. *Oxford Review of Economic Policy* **34** 376–392.

Prabhakar, Sankalp, Gerasimos Spanakis, Osmar Zaiane. 2017. Reciprocal recommender system for learners in massive open online courses. *International Conference on Web-Based Learning*. 157–167.

Provost, Foster, Tom Fawcett. 2001. Robust classification for imprecise environments. *Machine Learning* **42** 203–231.

Quadrana, Massimo, Paolo Cremonesi, Dietmar Jannach. 2018. Sequence-aware recommender systems. *ACM Computing Surveys* **51** 66:1–66:36.

Rendle, Steffen, Christoph Freudenthaler, Lars Schmidt-Thieme. 2010. Factorizing personalized Markov chains for next-basket recommendation. *International Conference on World Wide Web*. 811–820.

Ricci, Francesco, Lior Rokach, Bracha Shapira. 2011. Introduction to recommender systems handbook. *Recommender systems handbook*. Springer, 1–35.

Safran, Mejdl, Dunren Che. 2017. Real-time recommendation algorithms for crowdsourcing systems. *Applied Computing and Informatics* **13** 47–56.

Sahoo, Nachiketa, Param Vir Singh, Tridas Mukhopadhyay. 2012. A hidden Markov model for collaborative filtering. *MIS Quarterly* **36** 1329–1356.

Si, Luo, Rong Jin. 2003. Flexible mixture model for collaborative filtering. *International Conference on Machine Learning*. 704–711.

Snir, Eli M, Lorin M Hitt. 2003. Costly bidding in online markets for it services. *Management Science* **49** 1504–1520.

Song, Yang, Ali Mamdouh Elkahky, Xiaodong He. 2016. Multi-rate deep learning for temporal recommendation. *Conference on Research and Development in Information Retrieval*. 909–912.

Su, Xiaoyuan, Taghi M. Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence* **2009** 19.

Sundararajan, Arun. 2016. *The sharing economy: The end of employment and the rise of crowd-based capitalism*. Mit Press.

Symeonidis, Panagiotis, Dimitrios Malakoudis. 2018. MoocRec: Massive open online courses recommender system. *Collaborative Recommendations*. 627–651.

Tripp, M. Thomas, Yany Grégoire. 2011. When unhappy customers strike back on the internet. *MIT Sloan Management Review* **52** 37–44.

Tso-Sutter, H.L. Karen, Leandro Balby Marinho, Lars Schmidt-Thieme. 2008. Tag-aware recommender systems by fusion of collaborative filtering algorithms. *Symposium on Applied Computing*. 1995–1999.

Twardowski, Bartłomiej. 2016. Modelling contextual information in session-aware recommender systems with neural networks. *Conference on Recommender Systems*. 273–276.

Wang, Hao, Naiyan Wang, Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. *International Conference on Knowledge Discovery and Data Mining*. 1235–1244.

Yuen, Man-Ching, Irwin King, Kwong-Sak Leung. 2012. Task recommendation in crowdsourcing systems. *Workshop on Crowdsourcing and Data Mining*. 22–26.

Yuen, Man-Ching, Irwin King, Kwong-Sak Leung. 2015. Taskrec: A task recommendation framework in crowdsourcing systems. *Neural Processing Letters* **41** 223–238.

Zang, Hao, Yue Xu, Yuefeng Li. 2010. Non-redundant sequential association rule mining and application in recommender systems. *International Conference on Web Intelligence and Intelligent Agent Technology*, vol. 3. 292–295.

Zhang, Dongsong, Lina Zhou, Juan Luo Kehoe, Isil Yakut Kilic. 2016a. What online reviewer behaviors really matter? Effects of verbal and nonverbal behaviors on detection of fake online reviews. *Journal of Management Information Systems* **33** 456–481.

Zhang, Haidong, Wancheng Ni, Xin Li, Yiping Yang. 2016b. Modeling the heterogeneous duration of user interest in time-dependent recommendation: A hidden semi-Markov approach. *Transactions on Systems, Man, and Cybernetics: Systems* **48** 177–194.

Zhang, Shuai, Lina Yao, Aixin Sun, Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys* **52** 5.

Zheng, Alvin, Yili Hong, Paul Pavlou. 2015. Value uncertainty and buyer contracting: Evidence from online labor markets. *International Conference on Information Systems*.

Zheng, Lei, Vahid Noroozi, Philip S Yu. 2017. Joint deep modeling of users and items using reviews for recommendation. *International Conference on Web Search and Data Mining*. 425–434.

Appendix A Comparison with many-assessment recommender systems

In this section we explain in detail the structural limitations of implementing many-assessment systems in our context. We start by providing an example that highlights the conceptual differences between our context and many-assessment systems and then we discuss the necessary adaptations that we need to make in practice in order to implement many-assessments recommenders. The last subsection of this Appendix empirically illustrates the poor performance of these approaches compared with our HMM framework.

A.1 Comparison of the focal context with movie recommendations

To further clarify the conceptual differences of the focal context that we described in Section 2.2, consider the following comparison with a traditional many-assessment context, movie recommendations:

Scenario 1: A worker applies to two different jobs posted by two employers:

- Job 1 requires `Python` and `R`. The offered contract is hourly, the employer is experienced and located in Ukraine, and the task attracts a lot of applicants because `Python` and `R` are fairly popular on this platform.
- Job 2 requires `Stata`. Our focal worker knows both `R` and `Stata`, and hence applies to this task. This task is posted by an inexperienced employer from the U.S., and attracts very few applicants because `Stata` is not a popular skill on the platform.

Scenario 2: Two users choose to watch a movie from the same streaming platform:

- A user located in Ukraine chooses to watch the movie "Star Wars" among a set of available movies offered by the platform in Ukraine.
- A user located in the U.S. chooses to watch the movie "Star Wars" among a set of available movies offered by the platform in the U.S.

Are these two scenarios contextually identical?

Similar to the movie context, the users' (employers') tastes and backgrounds might differ. On the other hand, while in our context the underlying item and choice sets are task-specific, the two

Table 2 Comparison of the focal context with movie recommendations

Context	User	Evaluated item	Similarities	Differences
Watching movies	Service subscriber	Movie	• Users have different tastes.	• The movie content does not change. A given movie has the same story-line independent of who is watching it. • The total number of subscribers is significantly larger than the total number of movies, allowing for each movie to be evaluated thousands of times. • Subscribers who choose to watch the same movie made that choice among overlapping choice sets.
Hiring workers (1)	Employer	Worker	• Employers have different tastes.	• Workers complete unique tasks. • When they receive multiple evaluation scores, these scores represent their performance across different skills. • Employers who evaluate the same worker did not choose that worker among overlapping choice sets. • The total number of employers is significantly smaller than the total number of items (workers), hence each item is evaluated (on average) very few times.
Hiring workers (2)	Employer	Task-specific job application	• Employers have different tastes.	• Each item (job-application) is unique and receives only a single evaluation score. • Direct implementation of many-assessment systems is impossible as there are no multiple evaluations per item for the frameworks to estimate user-user and item-item similarities. Strong assumptions and adaptations are necessary (see Appendix A.2).

users who are using the same streaming platform have vastly overlapping sets of movies to choose from. Even further, in our context, the two employers evaluate the same worker for completely different tasks that require different skills. Yet, the movie users evaluate the exact same version of "Star Wars": they will watch and evaluate the exact same storyline with the same actors and the same finale. Table 2 summarizes these similarities and differences between the two contexts, illustrating the shortcomings of many-assessment frameworks in our context.

A.2 Implementing adaptations of many-assessment recommender systems

Despite these shortcomings, can we still modify many-assessment systems to address the focal problem?

Definition of items: The most significant adjustment we need to make is to define the items that multiple employers evaluate. We consider two options for modeling items:

- Option 1: The recommended item is the worker.
- Option 2: The recommended item is a mapping of job-applicant-task characteristics.

The first option is straightforward as it assumes that each worker is an item. This approach, ignores task-specific characteristics. The second option controls for such characteristics by clustering job applications according to vectors $\mathbf{X}_{o-1}, \mathbf{Z}_t$. Specifically, for this option, we use a Gaussian Mixture Model (GMM; see Murphy 2012). The model estimates the conditional probability $\Pr(\text{Clustered item}|\boldsymbol{Z}_t, \mathbf{X}_{o-1})$, where the clustered item becomes the focal recommended item. We choose the number of clustered items C according to the resulting BIC scores (Murphy 2012). We consider $C \in \{5, 10, ..., 100\}$. $C = 35$ yields the lowest BIC score; hence, we cluster job-applications into one of the 35 mapped items.

Definition of ratings: The next adaptation focuses on encoding the necessary ratings of a many-assessment system. In particular, many-assessment recommenders require an ordered rating scale (e.g., star ratings). Such ordering is absent in our context: the focal labels are "No-hire," "Hire-negative," and "Hire-positive." Even though "Hire-positive" is better than "Hire-negative," the relationship between "No-hire" and "Hire-negative" and between "No-hire" and "Hire-positive" cannot be uniquely defined.

To implement a performance-aware many-assessment recommender we consider the following ordering of ratings:

- "No-hire" $\mapsto 0$
- "Hire-negative" $\mapsto 1$
- "Hire-positive" $\mapsto 2$

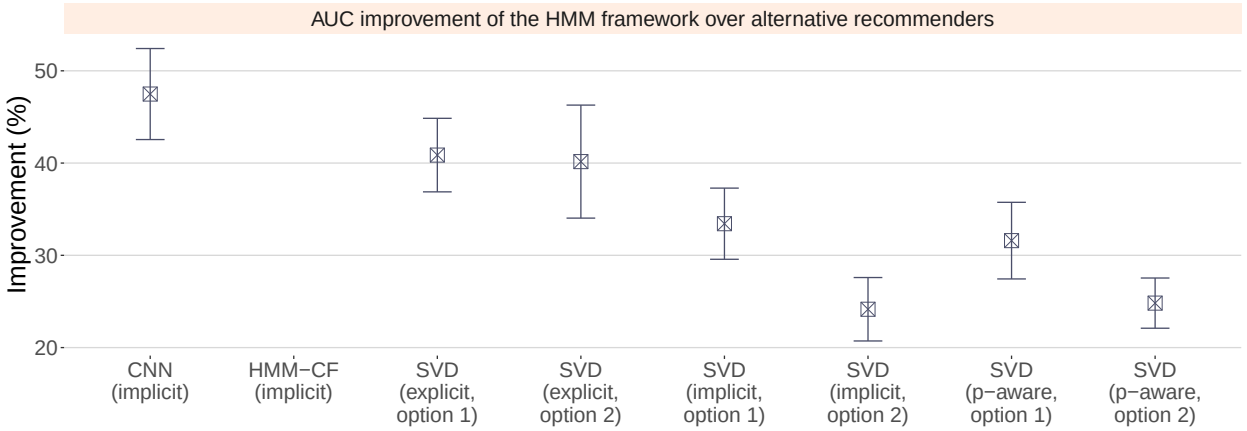
Many-assessment systems, however, do not have to be performance-aware. Traditionally, these systems are either implicit (i.e., modeling the choice to buy or not buy a product and ignoring the rating) or explicit (i.e., modeling only rated products). Hence, we can implement these types of systems in our context by ignoring the outcome of the hired candidate (implicit) and by focusing only on the hired workers and their outcomes (explicit). We discuss the specific implementations of many-assessment systems next.

A.3 Results

We build the following many-assessment recommenders:

- Collaborative filtering (SVD): Collaborative filtering recommenders are among the most popular and widespread industry approaches (Adomavicius and Tuzhilin 2005, Meyer 2012, Su and Khoshgoftaar 2009). In this analysis, we use singular value decomposition (SVD; see Kantor et al. 2011). The three different mappings of ratings and the two options for items generate six different SVD versions:
  1. Explicit feedback SVD with option 1 items
  2. Implicit feedback SVD with option 1 items
  3. Performance-aware SVD with option 1 items
  4. Explicit feedback SVD with option 2 items
  5. Implicit feedback SVD with option 2 items
  6. Performance-aware SVD with option 2 items
- HMM-based Collaborative Filtering (HMM-CF): Adaptations of many-assessment HMM-based recommender approaches could model employer evolution over changing hiring preferences. To showcase that their application experiences the same limitations with other many-assessment recommenders, we implement the proposed algorithm by Sahoo et al. (2012). Appendix N discusses the details of this adaptation. Since this approach models next-item recommendations (e.g., which news article to read next), it requires implicit-feedback sequences of observations (see Appendix N and Sahoo et al. 2012).

Figure 11 Comparison with many-assessment adaptations



Across three different outcome mappings (implicit, explicit, and p-aware) and two different item mappings (option 1 and option 2), many-assessment systems significantly underperform the proposed HMM approach. The $y-$axis shows the 10-fold nested cross-validated AUC improvement of the HMM approach compared with the $x-$axis many-assessment framework. Error bars show 90% confidence intervals. Implicit identifies implicit-feedback systems ("No-hire," "Hire"). Explicit identifies explicit-feedback systems ("Hire-negative," "Hire-positive"). P-aware identifies performance-aware systems ("No-hire," "Hire-negative," "Hire-positive"). Confidence intervals are estimated across the improvements of the 10 folds.

- Neural network sequence recommender systems (CNN): The rise and popularity of neural networks have motivated approaches that use such networks to build deep recommender systems (Section 2.1.1). Deep sequential recommenders can capture latent employer changing hiring preferences (Zhang et al. 2019). Convolutional neural networks (CNN) often model such sequential recommender systems (Kula 2017, 2018). As next-item recommenders, these systems require implicit-feedback sequences of observations (Kula 2017).

Figure 11 shows the results. As expected, none of the many-assessment approaches is competitive with the HMM framework: the AUC improvement ranges between 24% for the implicit SVD with option 2 items to up to 48% for the HMM-CF approach. Appendix B and Figure 16 further shows that many-assessment systems significantly underperform our approach in an alternative, few-assessment context. Overall, this analysis provides empirical support that the presented conceptual shortcomings of many-assessment systems significantly hurt their performance.

Appendix B Generalizability: Restaurant recommendations

Figure 1 identifies the type of contexts that our framework can provide relevant recommendations. In particular, our approach can generalize to other single-assessment and few-assessment contexts where the recommended items evolve or receive very few user evaluations, the user preferences evolve, or the choice sets do not significantly overlap. To illustrate, we implement and test our framework in an alternative few-assessment context: restaurant recommendations.

Our goal is to rank restaurant choices according to their likelihood of being reviewed—on an online reputation platform such as TripAdvisor—positively. In this context, both user preferences and item characteristics evolve. For instance, college students (reviewers) might be more interested in cheap fast food; As they grow older and become young professionals they might be interested in alternative types of cuisines. Similarly, the restaurants (recommended items) also evolve: through changing management, going through renovations, hiring new chefs and changing menus they offer different experiences over time.

These characteristics suggest that our framework, which captures evolving user preferences and recommends items based on their observed characteristics, should provide accurate restaurant recommendations. To illustrate, we focus on a popular zip-code area in the Boston area and create a dataset of 35,492 restaurant choices as captured on a set of TripAdvisor reviews.[11] Similar to our primary context, we generate three outcome levels: "No-review," "Review-negative," and "Review-positive." To separate "Review-negative" and "Review-positive" we use a threshold equal to 0.8 (4 out of 5 stars).

Similar to our primary analysis (Section 4 and Appendix I.1), we use a feature-selection process to identify the most informative variables. We consider the following predictive variables:

1. Number of posted reviews: the total number of previously posted reviews of the reviewer at hand after making $o-1$ restaurant choices (numeric).
2. Quality of reviewed restaurants: the average reputation of all the restaurants that the reviewer has previously reviewed after making $o-1$ restaurant choices (numeric).

[11] The dataset is available on our Github repository: `https://git.io/JmFBz`

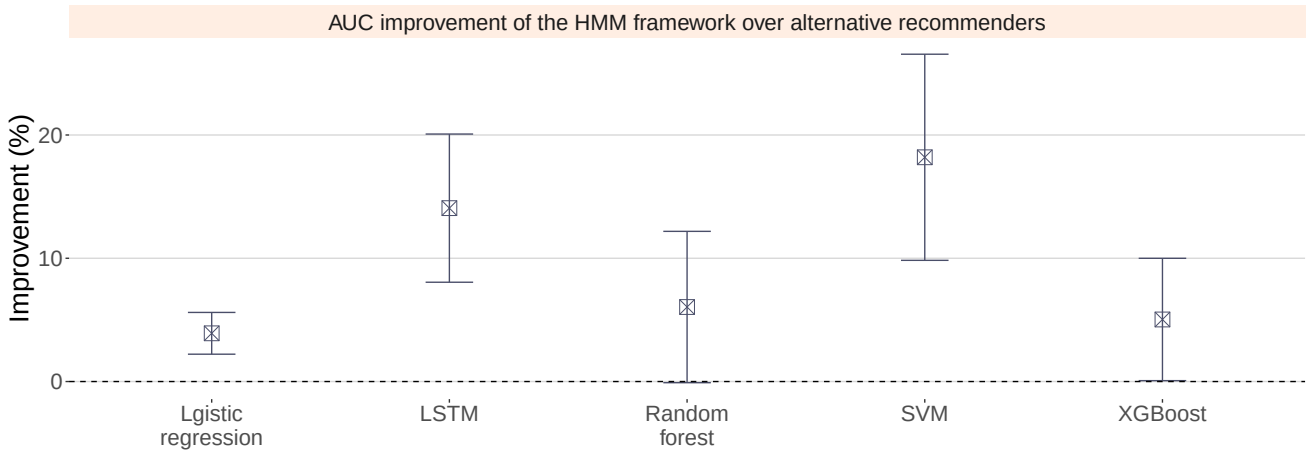
3. Restaurant rating: the restaurant's current reputation score at time t (numeric).
4. Restaurant number of reviews: the restaurant's total number of reviews at time t (numeric).
5. Whether the restaurant is categorized by the platform as "cheap" (binary).
6. Whether the restaurant is categorized by the platform as "affordable" (binary).
7. Whether the restaurant is categorized by the platform as "expensive" (binary).

We tune our framework and the alternative recommender systems following a similar process as the one presented in Appendix F.[12] We follow the same evaluation process as shown in Figure 6 and Section 5.3. Many-assessment recommenders in this few-assessment context do not require clustering, as this is a few-assessment context where multiple users evaluate the recommended items (restaurants). Yet, many-assessment recommenders require similar rating assumptions as in our main context to become performance-aware (Appendix A.2).

Figures 12 to 16 show the results in the same order as in Section 5.3. Across all measures, the proposed framework significantly (at least $p < 0.1$) outperforms all alternative recommender systems. Overall, this application of our approach in this alternative context illustrates its generalizability and provides empirical evidence that our framework can be successfully implemented in contexts where the recommended items evolve or receive very few user evaluations and where user preferences evolve.

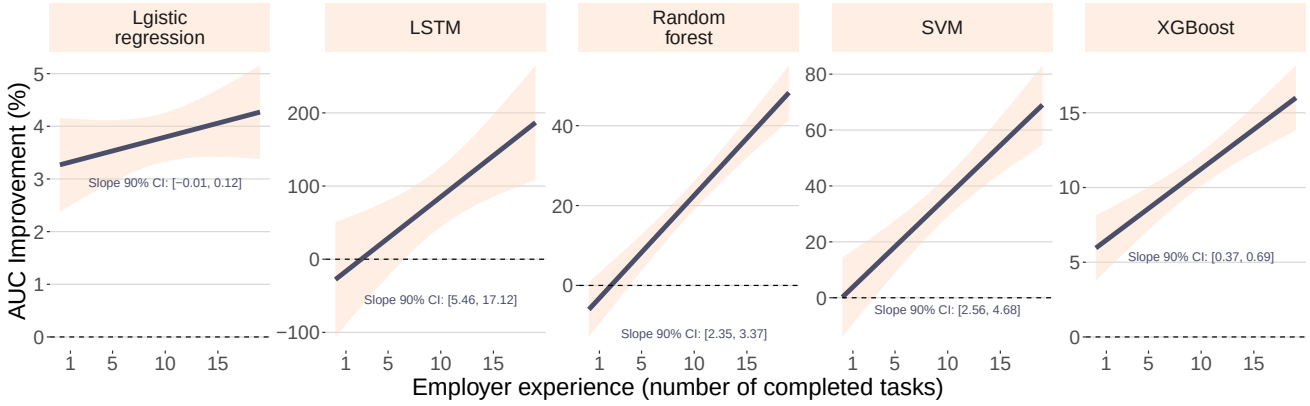
[12] We consider greater number of HMM states (i.e., $K \in \{5, 6\}$) as the average timelines of reviewers (i.e., posted reviews R) are longer compared with our main worker dataset.

Figure 12 Comparison of alternative recommenders: restaurant rankings for all reviewers



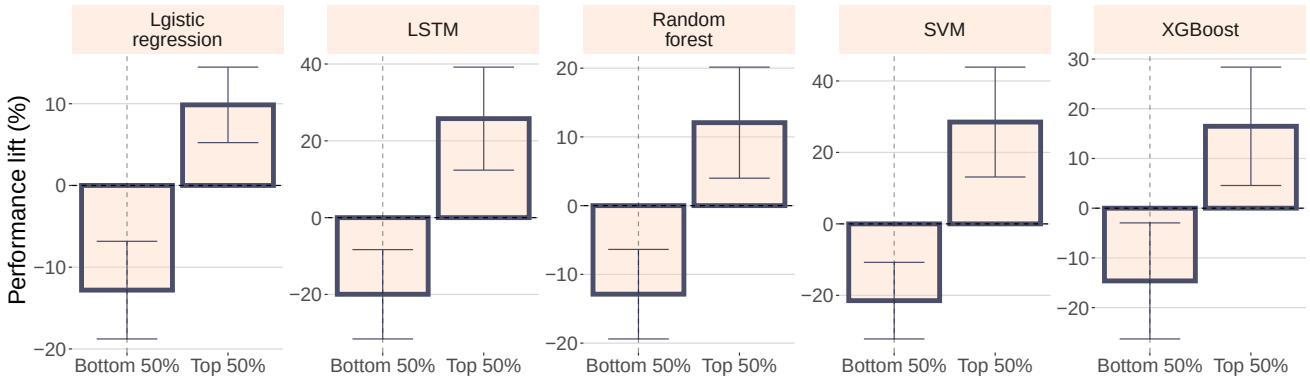
The proposed approach ranks restaurants according to their likelihood of getting reviewed positively significantly better (at least $p < 0.05$) than the sixteen alternative systems. The $y-$axis shows the 10-fold nested cross-validated AUC percentage improvement of the proposed framework over the $x-$axis recommender systems. Error bars show 90% confidence intervals.

Figure 13 Ranking performance of alternative recommenders for repeat reviewers



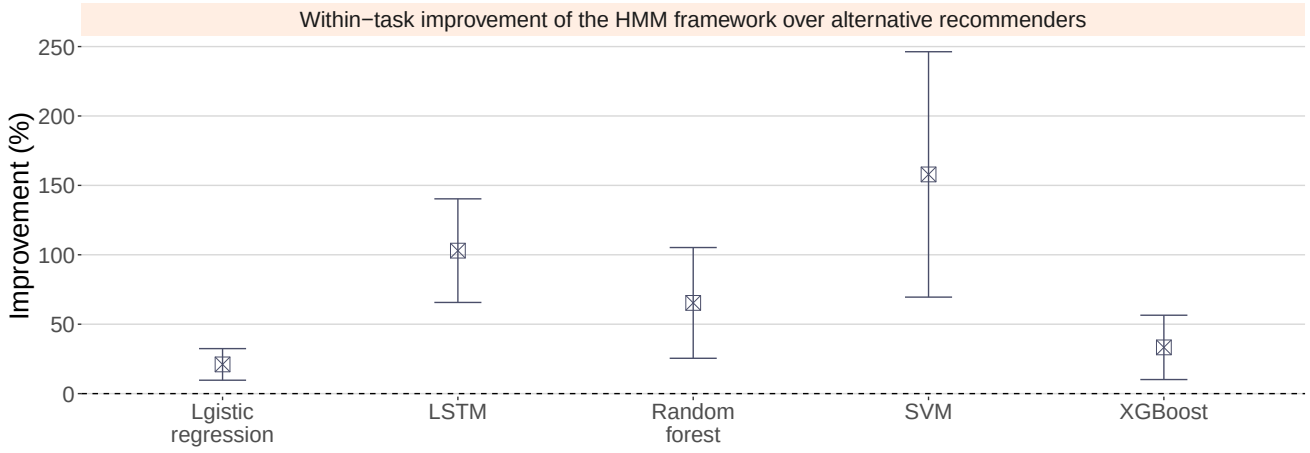
As reviewers evaluate additional restaurants, the proposed approach's improvement over the alternative recommender systems relatively increases (positive slope, $p < 0.1$ in all alternative approaches except logistic regression). The $y-$axis shows the 10-fold nested cross-validated AUC percentage improvement of the proposed framework over each alternative recommender system. The $x-$axis captures reviewer tenure in terms of posted reviews. Error bars show 90% confidence intervals. CI stands for confidence interval. Confidence intervals are estimated across the improvements of the 10 folds.

Figure 14 Comparison of alternative recommenders: performance of top-ranked restaurants



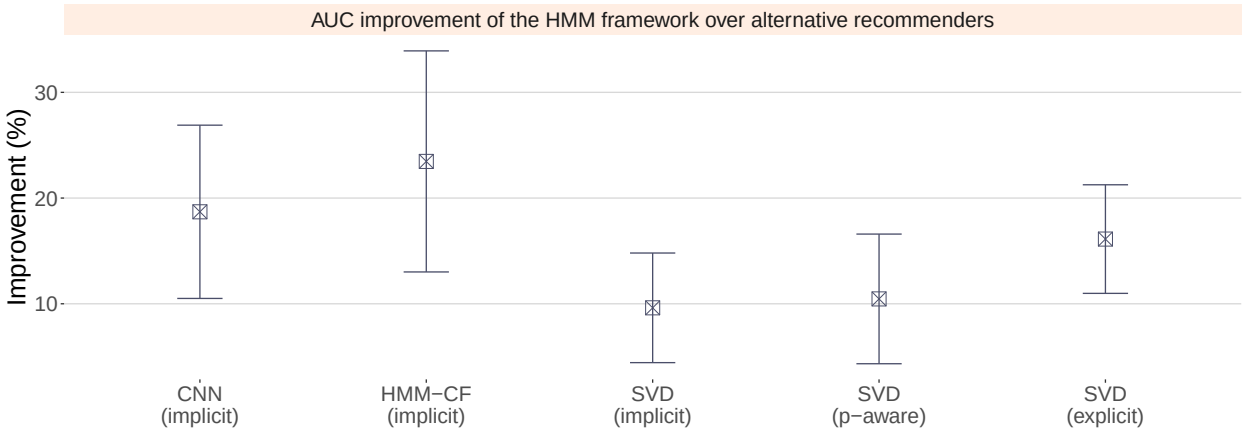
Compared with all alternative recommenders, the proposed approach ranks restaurants that are more likely to get reviewed positively in the top 50^{th} cohorts, while it ranks restaurants that are less likely to get reviewed positively in the bottom 50^{th} cohorts. The $y-$axis shows the 10-fold nested cross-validated performance lift (Equation 7). The $x-$axis ranks restaurants into cohorts according to their likelihood of getting reviewed positively. Error bars show 90% confidence intervals. Confidence intervals are estimated across the improvements of the 10 folds.

Figure 15 Comparison of alternative recommenders: performance of top-ranked restaurants within choice sets



The proposed approach ranks restaurants within choice sets according to their likelihood of getting reviewed positively significantly better (at least $p < 0.1$) than the alternative systems. The $y-$axis shows the 10-fold nested cross-validated within-choice-set performance (Equation 8) improvement of the proposed framework over the $x-$axis recommender systems. Error bars show 90% confidence intervals. Confidence intervals are estimated across the improvements of the 10 folds.

Figure 16 Comparison with many-assessment adaptations (restaurant recommendations)



Across three different outcome mappings (implicit, explicit, and p-aware), many-assessment systems significantly underperform the proposed HMM approach in this alternative context. The $y-$axis shows the 10-fold nested cross-validated AUC improvement of the HMM approach compared with the $x-$axis many-assessment framework. Error bars show 90% confidence intervals. Implicit identifies implicit-feedback systems ("No-review," "Review"). Explicit identifies explicit-feedback systems ("Review-negative," "Review-positive"). P-aware identifies performance-aware systems ("No-review," "Review-negative," "Review-positive"). Confidence intervals are estimated across the improvements of the 10 folds.

Appendix C HMM likelihood derivation and estimation

Given the structure of the HMM, the framework needs to estimate the parameter vectors $\boldsymbol{\pi}, \boldsymbol{\beta}, \boldsymbol{\gamma}$. To do so, it maximizes the conditional probability of the set of observations given the HMM. Let us assume a sequence of M hiring decisions across R tasks for a given employer i:

$$\boldsymbol{Y}_i = Y_{i1}, Y_{i2}, \ldots, Y_{iM} \,, \tag{9}$$

where $Y_{im} \in \mathcal{Y}, m \in \{1, 2, ..., M\}$. These observations correspond to a sequence of employer and job-application characteristics (Figure 4):

$$\boldsymbol{X}_{i0:R-1} = \boldsymbol{X}_{i0}, \boldsymbol{X}_{i1}, \ldots, \boldsymbol{X}_{iR-1} \,, \tag{10}$$

$$\boldsymbol{Z}_{i1:M} = \boldsymbol{Z}_{i1}, \boldsymbol{Z}_{i2}, \ldots, \boldsymbol{Z}_{iM} \,. \tag{11}$$

Furthermore, let us assume that $\boldsymbol{Y}_i$ is the result of a sequence of latent states, $\boldsymbol{S}_i$:

$$\boldsymbol{S}_i = S_{i1}, S_{i2}, \ldots, S_{iM} \,, \tag{12}$$

where $S_{im} \in \mathcal{S}$.

Based on the structure of the graph in Figure 4, the conditional likelihood of observing $\boldsymbol{Y}_i$ is:

$$\Pr(\boldsymbol{Y}_i | \boldsymbol{S}_i; \boldsymbol{\beta}, \boldsymbol{Z}_{i1:M}, \boldsymbol{X}_{i1:R-1}) = \prod_{t=1}^{M} \Pr(Y_{it} | S_{it}; \boldsymbol{\beta}, \boldsymbol{Z}_{it}, \boldsymbol{X}_{io-1}) \,, \tag{13}$$

where Equation 4 estimates the right hand side (recall that the subscript o is connected with subscript t). The conditional probability of observing the sequence $\boldsymbol{S}_i$ is (Figure 4):

$$\Pr(\boldsymbol{S}_i | \boldsymbol{\gamma}, \boldsymbol{X}_{i1:R-1}) = \boldsymbol{\pi}(S_1) \prod_{t=2}^{M} \Pr(S_{it} | S_{it-1}; \boldsymbol{\gamma}, \boldsymbol{X}_{io-1}) \,, \tag{14}$$

where $\boldsymbol{\pi}(S_1)$ is the the prior probability of being at state $S_1 \in \mathcal{S}$. Equation 2 estimates the right hand side.

Based on this analysis and the graph in Figure 4, the likelihood of this sequence of observations for employer i is as follows:

$$\begin{aligned} l(\boldsymbol{Y}_i;\boldsymbol{\pi},\boldsymbol{\beta},\boldsymbol{\gamma}) &= \Pr(\boldsymbol{Y}_i|\boldsymbol{\pi},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{Z}_{i1:M},\boldsymbol{X}_{i1:R-1},\mathbf{Y}_{i1:M-1}) \\ &= \boldsymbol{\pi}(S_1)\sum_{\forall \boldsymbol{S}_i}\Pr(\boldsymbol{Y}_i,\boldsymbol{S}_i|\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{Z}_{i1:M},\boldsymbol{X}_{i1:R-1},\mathbf{Y}_{i1:M-1}) \\ &\overset{\text{Figure 4}}{=} \boldsymbol{\pi}(S_1)\sum_{\forall \boldsymbol{S}_i}\Pr(\boldsymbol{Y}_i|\boldsymbol{S}_i;\boldsymbol{\beta},\boldsymbol{Z}_{i1:M},\boldsymbol{X}_{i1:R-1})\Pr(\boldsymbol{S}_i|\boldsymbol{\gamma},\boldsymbol{X}_{i1:R-1}) \\ &= \boldsymbol{\pi}(S_1)Pr(Y_{i1}|S_{i1};\boldsymbol{\beta},\boldsymbol{Z}_{i1},\boldsymbol{X}_{i0}) \\ &\cdot \sum_{\forall \boldsymbol{S}_i}\prod_{t=2}^{M}\Pr(Y_{it}|S_{it};\boldsymbol{\beta},\boldsymbol{Z}_{it},\boldsymbol{X}_{io-1}) \\ &\cdot \Pr(S_{it}|S_{it-1};\boldsymbol{\gamma},\boldsymbol{X}_{io-1})\,, \end{aligned} \tag{15}$$

where the structure of the HMM (Figure 4) decomposes the joint probability of $\Pr(\boldsymbol{Y}_i,\boldsymbol{S}_i|\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{Z}_{i1:M},\boldsymbol{X}_{i1:R-1},\mathbf{Y}_{i1:M-1})$. Note that $\boldsymbol{X}_{i0}$ captures the initial characteristics of each employer before they complete their first task. The complete likelihood for a dataset with N employers is as follows:

$$L(\boldsymbol{\beta},\boldsymbol{\gamma}) = \prod_{i=1}^{N} l(\boldsymbol{Y_i};\boldsymbol{\pi},\boldsymbol{\beta},\boldsymbol{\gamma})\,. \tag{16}$$

To estimate the parameters $\boldsymbol{\pi},\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ that maximize this complete likelihood we can use common solvers such as the L-BFGS-B (Byrd et al. 1995) or the COBYLA[13] algorithms. (In practice, we minimize the negative log-likelihood of Equation 16. See Appendix I and our Github repository[14] for additional implementation details.) Equation 16 assigns more weight to employers who make more hiring decisions (i.e., larger M and R values). As a result, it is expected (and desired) that the resulting HMM will assigned greater weight to (and as a result represent better) repeat employers than one-time employers.

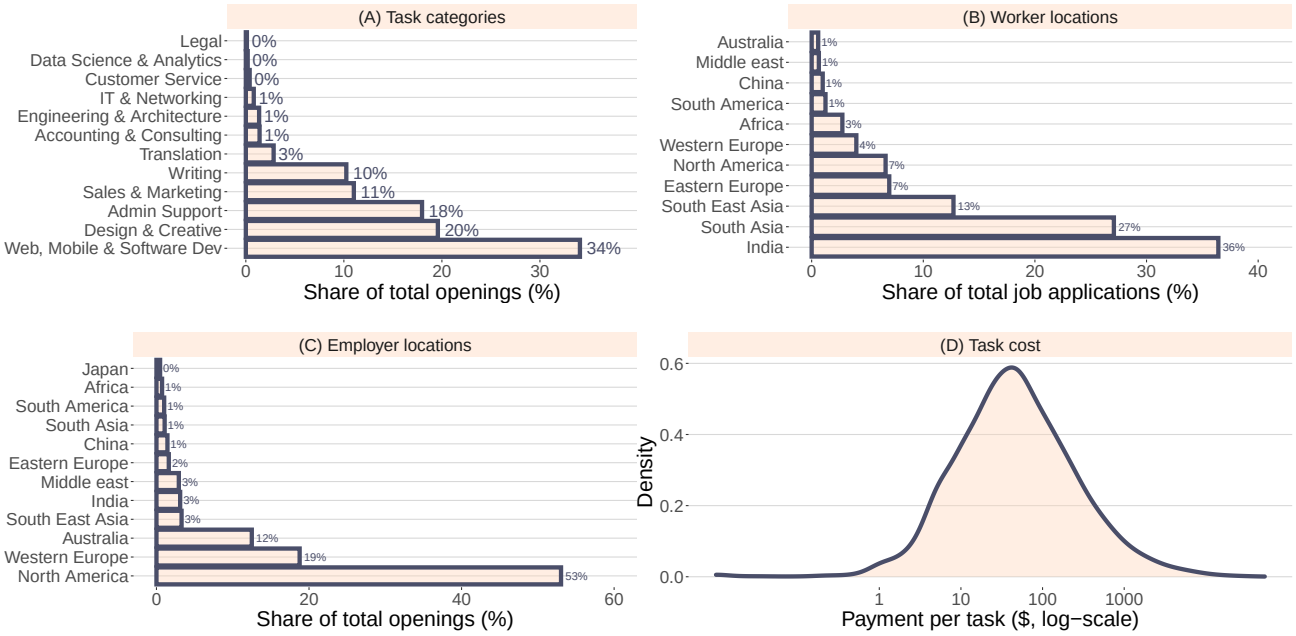
[13] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

[14] https://github.com/repudime/gbu

Appendix D The marketplace

The focal marketplace provides a variety of opportunities for high-skilled workers in categories such as web and software development (i.e., programming), graphic design, marketing, and writing (Figure 17A). Job applicants come from a variety of locations (Figure 17B), including India (36%), south and south-east Asia (40%), Europe (11%) and North America (7%). The majority of the employers reside in North America (53%, Figure 17C). However, a significant portion comes from Europe, Australia, and Asia, highlighting the global dimension of the focal platform. Finally, employers spend significant amounts of money on the platform, often hundreds or thousands of dollars per completed task (Figure 17D). This non-trivial cost of each task suggests that employers likely expect fruitful collaborations with the hired workers.[15]

Figure 17 Characteristics of the focal market



The figures show information about the types of tasks (A), worker locations (B), employer locations (C), and task cost.

[15] At the time that we queried the market's database, the platform did not offer an advanced task recommender. Workers could look for tasks based on keyword searches. After applying to a job opening, employers could rank applicants according to their reputation and arrival time.

Appendix E Predictive variables

Below is the description of the predictive variables we consider in our feature selection process:

1. *Employer money spent after completing $o-1$ tasks*: The total amount of money that the focal employer has spent so far in the previous $o-1$ completed tasks (numeric).
2. *Employer most recent outcome ($o-1$)*: The outcome of the most recent $(o-1)$ completed task of the employer (numeric).
3. *Employer total competed tasks ($o-1$)*: the employer's total number of completed jobs (numeric).
4. *Employer number of fixed contracts after completing $o-1$ tasks*: the employer's total number of completed jobs that used fixed contracts after completing $o-1$ tasks (numeric).
5. *Employer number of hourly contracts after completing $o-1$ tasks*: the employer's total number of completed jobs that used hourly contracts after completing $o-1$ tasks (numeric).
6. *Employer total hire-positive outcomes after completing $o-1$ tasks*: the employer's total number of "Hire-positive" outcomes after completing $o-1$ tasks (numeric).
7. *Employer fixed contract jobs with hire-positive outcomes after completing $o-1$ tasks*: the employer's total number of "Hire-positive" outcomes in previously completed fixed contracts (numeric).
8. *Employer hourly contract jobs with hire-positive outcomes after completing $o-1$ tasks*: the employer's total number of "Hire-positive" outcomes in previously completed hourly contracts (numeric).
9. *Applicant completed work-hours*: the number of hours that the job applicant has worked on the platform at the time of application (numeric).
10. *Skills IP*: the inner product between the skillset of the job-applicant and the required skills by the task at the time of application (numeric).
11. *Applicant bid price*: the bid price of the applicant (numeric).
12. *Received order of application*: the order that a job application was received (numeric).

13. *Applicant accumulated reputation score (publicly available)*: the accumulated publicly available reputation score of the job applicant at the time of application (numeric).
14. *Applicant completed jobs*: the applicant's total number of completed jobs (numeric).
15. *Employer-applicant countries PMI*: the pairwise mutual information between the job applicant's country and the employer's country (numeric; see below Equation 18). Conceptually, it captures the average affinity of employers from a given country to hire workers from the applicant's country.
16. *Applicant's self-reported years of experience (not verified by the platform)*: the self-reported job applicant's experience (numeric).
17. *Certifications PMI*: the pairwise mutual information between the listed certificates of the job-applicant and the required skills by the task (numeric; see Equation 17).
18. *Invited*: whether or not the employer invited the job applicant to apply to the focal task (binary).
19. *Contract type*: whether or not the job is fixed priced or hourly rated (binary).
20. *Applicant bid price* $\times$ *Contract type*: an interaction term between the job applicant's bid price and the contract type (numeric).
21. Employer number of fixed contracts after completing $o-1$ tasks $\times$ Contract type: interaction between the employer's total number of completed fixed-contract jobs and the contract type of the job at hand (numeric).
22. Employer hourly contract jobs with hire-positive outcomes after completing $o-1$ tasks $\times$ Contract type: interaction between the employer's total number of completed hourly-contract jobs and the contract type of the job at hand (numeric).

Most of these variables are self-explained. To estimate the pairwise mutual information (PMI) of countries and certifications we rely on prior work on hiring decisions (Kokkodis et al. 2015):

$$\text{Certifications PMI}(E_a, E_i) = \log \frac{\Pr(E_a, E_i)}{\Pr(E_a)\Pr(E_i)}\,, \tag{17}$$

$$\text{Employer-applicant countries PMI}(C_a, C_i) = \log \frac{\Pr(C_a, C_i)}{\Pr(C_a)\Pr(C_i)}\,, \tag{18}$$

where E_a is the set of certifications of the job applicant, E_i is the set of required skills of the specific job task, C_a is the country of the job-applicant, and C_i is the country of the employer. PMI measures how much the probability of a co-occurrence between two events differs from what we would see if the two events were independent. It can be both positive and negative, and it is zero when the two events are independent.

To avoid overfitting, we estimate these PMI scores on a separate set of 10,000 previously hired workers. The employers who hired these 10000 workers are not included in the focal dataset described in Section 4. Because we use hired workers, the PMI scores reflect employers' average affinities towards certain countries and certifications.

Appendix F Hyperparameter tuning

The proposed framework and many of the alternative recommender systems discussed in Section 5.2 require hyperparameter tuning. This Appendix discusses the grid search approaches we followed to tune these parameters for our HMM and the alternative recommender systems that rely on random forest, gradient boosting, and recurrent neural networks.

HMM tuning: For the proposed framework we need to identify the number of states K for each fold in the nested cross validation structure (Figure 6). We consider the following 40 combinations:

$$\text{HMM tested combinations:} \left\{ \overbrace{\{2,3,4,5\}}^{K} \times \overbrace{\{0,1,...,9\}}^{folds} \right\}. \tag{19}$$

For each one of these combinations, we follow an HMM-specific step-forward feature selection process (Ferri et al. 1994) to identify the best-performing predictive variables on the validation set (see also Appendix I and our Github repository for more information on the feature selection process).

Random forest tuning: For the Random forest recommender, we use the Python package `sklearn.ensemble.RandomForestClassifier`. We experiment with two hyper-parameters: the maximum number of levels in each decision tree ("`max_depth`"), and the number of trees in the forest ("`n_estimators`"). We consider the following 90 combinations:

$$\text{Random forest tested combinations:} \left\{ \overbrace{\{3,10,15\}}^{\texttt{max\_depth}} \times \overbrace{\{10,50,100\}}^{\texttt{n\_estimators}} \times \overbrace{\{0,1,...,9\}}^{folds} \right\}. \tag{20}$$

Similar with the HMM training process, for each one of these combinations, we follow a random-forest-specific step-forward feature selection process to identify the best-performing predictive variables on the validation set.

XGBoost tuning: For Gradient boosting we use the Python package `xgboost`. We tune three hyperparameters: the number of trees to fit ("`n_estimators`"), the maximum tree depth ("`max_depth`"), and the subsample ratio of the training instance ("`subsample`"). We consider the following 180 combinations:

$$\text{XGBoost tested combinations:} \left\{ \overbrace{\{50,100,150\}}^{\texttt{n\_estimators}} \times \overbrace{\{3,10,15\}}^{\texttt{max\_depth}} \times \overbrace{\{0.8,1\}}^{\texttt{subsample}} \times \overbrace{\{0,1,...,9\}}^{folds} \right\}. \tag{21}$$

Similar with the HMM training process, for each one of these combinations, we follow an XGBoost-specific step-forward feature selection process to identify the best-performing predictive variables on the validation set.

LSTM tuning: To build LSTM networks we use the Python packages `keras.models.Sequential` and `keras.layers.LSTM`. To get probability estimates, we use a `softmax` activation function and we optimize according to the `categorical_crossentropy`. We tune two hyperparameters: the number of "`epochs`" to train the model, and the number of samples per gradient update "`batch_size`." In addition, we explore stacking hidden LSTM layers, in an effort to improve performance (Brownlee 2017). We consider the following 180 combinations:

$$\text{LSTM tested combinations:} \left\{ \overbrace{\{10,20,30\}}^{\texttt{epochs}} \times \overbrace{\{32,64,128\}}^{\texttt{batch\_size}} \times \{\text{Stacked, Not stacked}\} \times \overbrace{\{0,1,...,9\}}^{folds} \right\}. \tag{22}$$

To set the parameter "`units`" (i.e., the dimensionality of the output space of the hidden layer; see the dense layer implementation of Keras 2021), we use the formula (Eckhardt 2018):

$$\texttt{units} = 0.67 * (n_{features} + n_{steps}), \tag{23}$$

where $n_{features}$ is the total number of predictive variables and n_{steps} is the length of the sequence (see also Appendix O). Note that the units (also known as neurons) refer to the hidden layer of the network; our final layer has three neurons that capture the total number of output classes we predict (see the run_lstm function on our Github repository: `https://git.io/JmF80#L88`).

Similar with the previous models, for each one of these combinations, we follow an LSTM-specific step-forward feature selection process to identify the best-performing predictive variables on the validation set.

Appendix G Alternative ranking functions

The main goal of our research is to rank job-applicants according to their likelihood of getting hired and performing well. However, alternative ranking mechanisms could trade-off "Hire-positive" outcomes for fewer "Hire-negative" outcomes to provide risk-averse recommendations.

In this appendix, we consider four alternative ranking approaches:

- Difference: This approach subtracts the likelihood of observing a "Hire-negative" outcome from the likelihood of observing a "Hire-positive" outcome.
- 75%-Hire-negative 25%-Hire-positive: This approach interpolates the probability of the two outcomes and creates rankings according to the following:

$$0.75 * \Big(1 - \Pr(\text{“Hire-negative”})\Big) + 0.25 * \Pr(\text{“Hire-positive”})\,. \tag{24}$$

- 25%-Hire-negative 75%-Hire-positive: This approach interpolates the probability of the two outcomes and creates rankings according to the following:

$$0.25 * \Big(1 - \Pr(\text{“Hire-negative”})\Big) + 0.75 * \Pr(\text{“Hire-positive”})\,. \tag{25}$$

- Odds ratio: This approach estimates the ratio of the likelihood of the two outcomes:

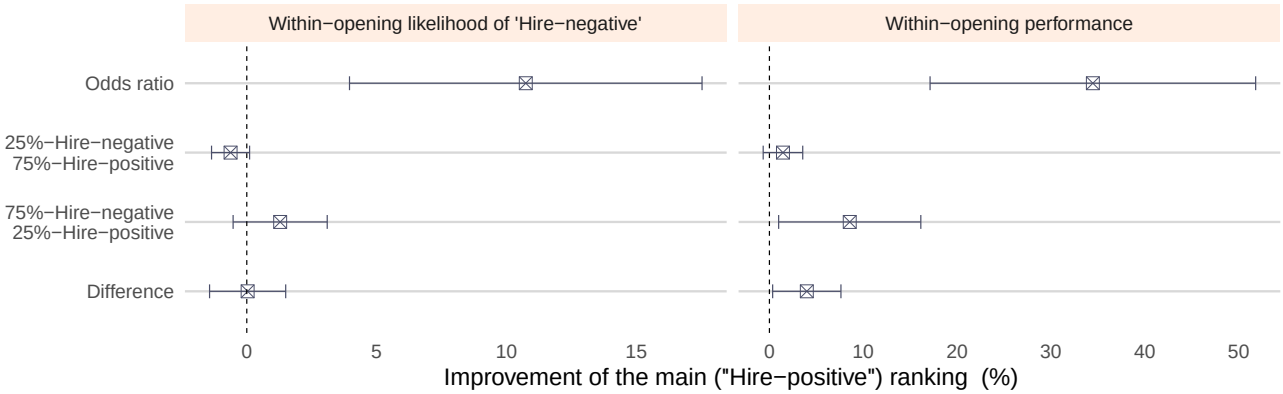
$$\text{Odds ratio} = \frac{\Pr(\text{“Hire-positive”})}{\Pr(\text{“Hire-negative”})}\,. \tag{26}$$

Figure 18 compares these four different ranking approaches with the main ranking approach that ranks applicants according to their estimated likelihood of getting hired and performing well. The Figure shows the within-task performance as defined in Equation 8, as well as the within-task likelihood of "Hire-negative":

$$\text{Within-task likelihood of “Hire-negative”}(k) = \frac{\#(\text{“Hire-negative”} \in \text{top}{-}k \text{ recommended})}{\#(\in \text{top}{-}k \text{ hired})}\,. \tag{27}$$

The results show that in terms of within-task performance, the main ("Hire-positive") ranking performs on par or better than the four alternative ranking approaches. However, in terms of minimizing the likelihood of "Hire-negative" outcomes, the main ranking performs on par or worse. In practice, managers can experiment with such alternative ranking mechanisms and adjust according to their objectives and costs of "Hire-positive" and "Hire-negative" outcomes.

Figure 18 Comparison of alternative rankings

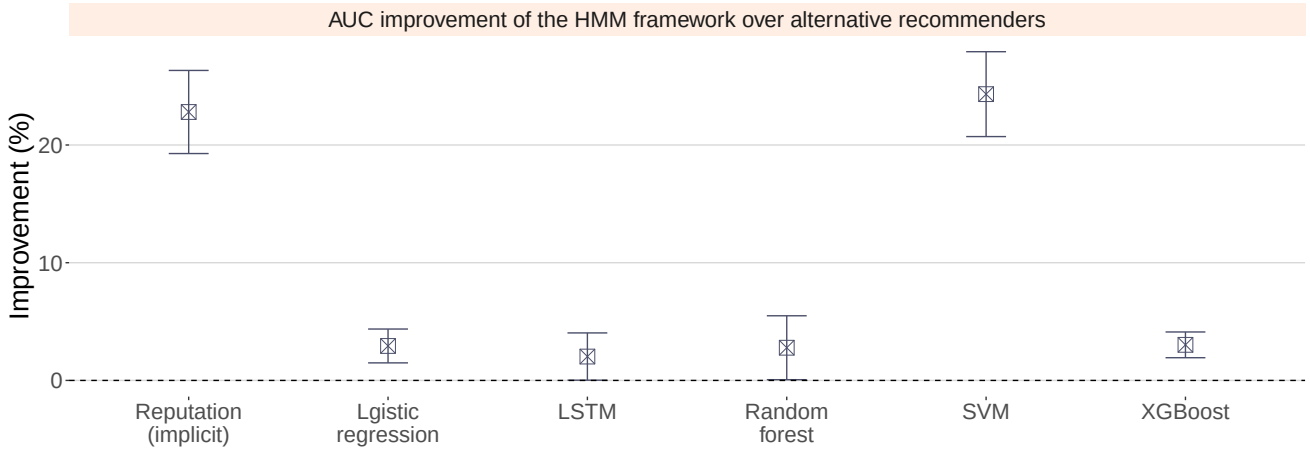


Most rankings yield comparable results; The odds-ratio rankings yield worse results than the main ranking in terms of within-task performance (Equation 8), but better results in terms of the within-task likelihood of getting a "Hire-negative" outcome (Equation 27). The main raking approach ranks applicants according to their "Hire-positive" probabilities. Error bars show 95% confidence intervals.

Appendix H Robustness with alternative thresholds

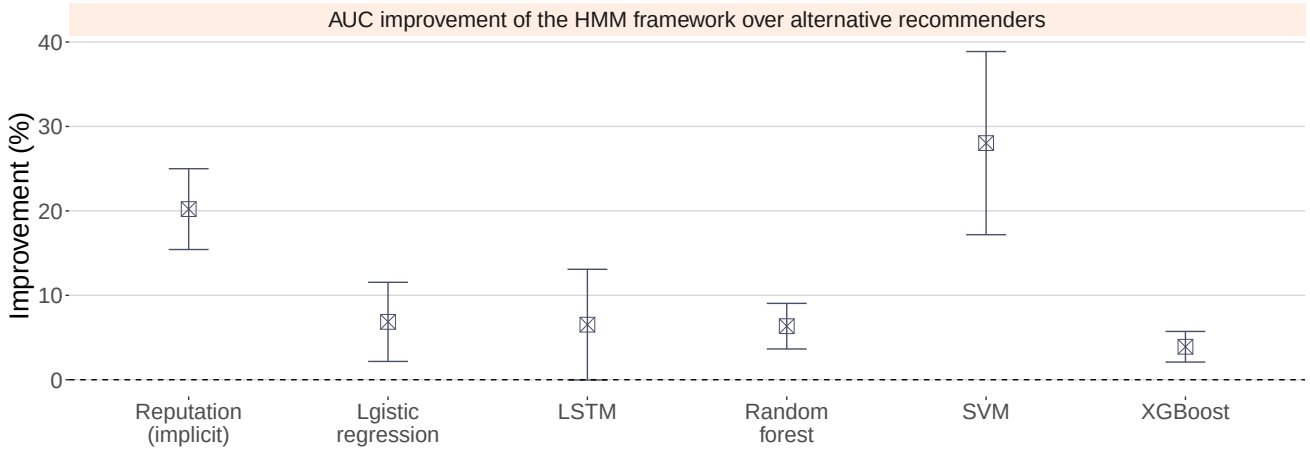
The choice of a performance threshold equal to 0.8 to separate "Hire-positive" from "Hire-negative" might bias the observed results. To check the robustness of our approach across alternative thresholds, we repeat the evaluation process for performance thresholds equal to 0.7 and 0.9. Figures 19 and 20 show the results that are qualitatively the same with the results in our primary analysis.

Figure 19 Comparison of alternative recommenders: job-applicant rankings for all employers (performance threshold = 0.7)



The proposed approach ranks job applicants according to their likelihood of getting hired and performing well significantly better (at least $p < 0.1$) than the alternative systems. The $y-$axis shows the 10-fold nested cross-validated AUC percentage improvement of the proposed framework over the $x-$axis recommender systems. Error bars show 90% confidence intervals. Implicit identifies implicit-feedback systems ("No-hire," "Hire"). Confidence intervals are estimated across the improvements of the 10 folds.

Figure 20 Comparison of alternative recommenders: job-applicant rankings for all employers (performance threshold = 0.9)



The proposed approach ranks job applicants according to their likelihood of getting hired and performing well significantly better (at least $p < 0.1$) than the alternative systems. The $y-$axis shows the 10-fold nested cross-validated AUC percentage improvement of the proposed framework over the $x-$axis recommender systems. Error bars show 90% confidence intervals. Implicit identifies implicit-feedback systems ("No-hire," "Hire"). Confidence intervals are estimated across the improvements of the 10 folds.

Appendix I Implementation details

This section presents the implementation details of our framework. You can find the respective Python code in our Github repository: `https://github.com/repudime/gbu`[16]

I.1 Feature selection

Figure 6 describes the nested cross validation process we follow to train our framework and the alternative approaches. Feature selection happens in the training and validation sets of each fold. We use a step-forward feature selection (Ferri et al. 1994) process. In particular, we use the Python package `mlxtend.feature_selection.SequentialFeatureSelector`, and we build each respective model by choosing the most informative predictive variables among the ones presented in Appendix E. The function `do_feature_selection` inside the `custom_utils.py` file shows the Python code for this process.

I.2 Complexity and running time

One potential concern about our approach is that it might be too expensive—in terms of training time—to estimate. Indeed, the numeric optimization presented in Appendix C is computationally tedious. In fact, Equation 16 can take a significant amount of time to be estimated even for tiny amounts of data and parameters (see function `get_individual_log_l` and relevant time tests).

Instead of estimating this equation, we estimate its vectorized version (function get_single_timeline_likelihood_map). The complexity of this factorized function is $O(N * M)$, where N is the number of employers (timelines) in our data and M is the median length of a timeline (number of received job applications across completed tasks). In cases such us ours, where M is small compared to N, complexity grows almost linear with the number of timelines $\approx O(N)$. And as we discuss in Appendix I.4, complexity can even be reduced to a constant $\approx O(1)$ through parallelization.

Furthermore, our implementation uses state of the art linear algebra libraries[17] that significantly speed up the process. Overall, even though the training time of our HMM will vary depending on

[16] Our README.md file provides rich information about our HMM implementation and explains in detail how we impose the structure of the HMM, and how we derive the emission and transition probabilities (matrices T and E).

[17] BLAS: http://www.netlib.org/blas/

the number of iterations, it is comparable with the training time of our LSTM approaches even without parallelization (see relevant time tests).

I.3 Parameters to be estimated

The total number of parameters to be estimated depends on the number of states, the number of outcomes, and the number of emission and transition variables. Assuming n_{states} to be the number of states, $n_{outcomes}$ to be the number of outcomes, and $n_{transitions}, n_{emissions}$ to be the number of transition and emission variables respectively, the total number of parameters that the model will need to estimate will be:

$$\begin{aligned}\text{Total number of parameters} = n_{states} * (n_{states} - 1) * n_{transitions} \\ + n_{states} * n_{emissions} * (n_{outcomes} - 1) + n_{states} \,. \end{aligned} \tag{28}$$

Hence, the parameters that we estimate for our main models range from 42 ($n_{states} = 2,\ n_{transitions} = 2,\ n_{emissions} = 9,\ n_{outcomes} = 3$) to 410 ($n_{states} = 5,\ n_{transitions} = 9,\ n_{emissions} = 23,\ n_{outcomes} = 3$). We share the details of the parameter vector in our Github repository as follows:

- `hmm_gbu.py`: line 120 calls the function `createPriors` (https://git.io/JmF4h)
- `hmm_functions.py`: line 348 defines the function `createPriors` that initializes our parameter vector to be optimized (https://git.io/JmFBo)

I.4 Implementation in practice

Finally, we argue that our approach is overall appealing to practitioners for two more reasons:

- Offline training: In practice, platforms can train our approach once a week (or a month) based on new data. This training process happens offline. Once the model learns its parameters, real-time recommendations happen instantaneously.
- High potential for parallelization: The log of Equation 16 that forms the negative log-likelihood we minimize is a sum of individual user likelihoods (function `get_single_timeline_likelihood_map`). Hence, we can estimate the individual user-likelihoods in parallel and aggregate them at the end. We can do this through batches assigned in parallel cores, or, through a MapReduce formulation.

Appendix J Comparison of alternative transition constraints

One of the distinct characteristics of our framework is that it allows transitions only after the completion of a task (Equation 2). Does this matter?

We compare our task-specific approach with an HMM that allows task-independent transitions after every observation $Y_{t-1} \in \mathcal{Y}$. Specifically, we can assume that transitions have the following form:

$$\lambda^{s_k s_l}_{\boldsymbol{\gamma}_{kl} \boldsymbol{X}_{o-1} Y_{t-1}} := \Pr(S_t = s_l | S_{t-1} = s_k; \boldsymbol{\gamma}_{kl}, \boldsymbol{X}_{o-1}) = \text{softmax}(\boldsymbol{\gamma}_{kl} \boldsymbol{X}_{o-1}) \,. \tag{29}$$

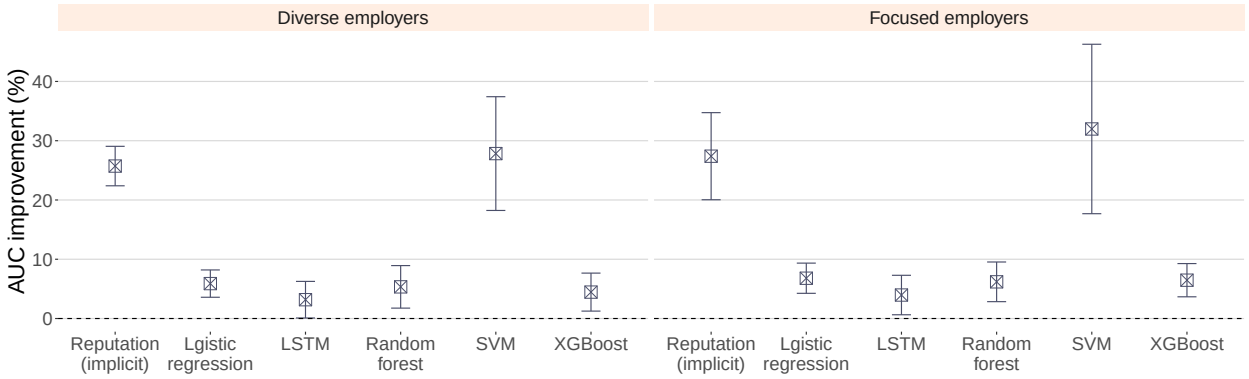
When we test this implementation, we observe that constraining transitions after the completion of a task yields up to 4% significantly ($p < 0.001$) better results. Hence, our unique design choice to model transitions through Equation 2 significantly improves the performance of our approach.

Appendix K Heterogeneous employers

Does our framework perform well for different kinds of employers?

To test, we split our test sets into focused (i.e., employers who hire workers only in a single category) and diverse (i.e., employers who hire workers in multiple task categories) employers. Figure 21 shows the results. Our framework performs better compared with alternative approaches across both groups, but it does slightly better across focused employers, as perhaps it is able to better learn their preferences over time.

Figure 21 Comparison of alternative recommenders: job-applicant rankings for all employers(Diverse vs. focused employers)

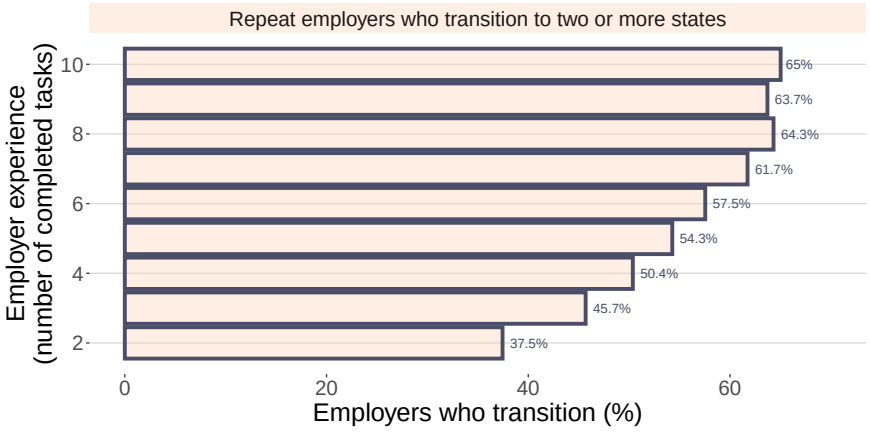


The proposed approach significantly outperforms alternative approaches for both diverse (hire workers across multiple task categories) and focused (hire workers only in a single task category) employers. Error bars show 90% confidence intervals. Confidence intervals are estimated across the improvements of the 10 folds.

Appendix L Do employers transition across different states?

Another relevant question to our proposed framework is whether employers indeed evolve over time: If they do not, then there is no need for implementing a complex system such as the HMM. Figure 8 shows that our framework performs increasingly better on repeat employers, illustrating that such repeat employers likely evolve. To better capture this evolution, Figure 22 shows the percentage of employers who complete more than two tasks and transition to more than two states. Indeed, a significant portion of repeat employers (that ranges between 37.5% and 65%) transition to two or more states during our observational window.

Figure 22 A significant portion of employers evolves across multiple HMM states



Appendix M AUC scores

Table 3 presents the AUC scores of our framework and the alternative approaches, for all three performance thresholds.

Table 3

		Model	AUC (performance threshold =0.8)	AUC (performance threshold =0.7)	AUC (performance threshold =0.9)
Average		HMM	0.71	0.68	0.681
		Lgistic regression	0.668	0.661	0.642
		Random forest	0.673	0.664	0.643
		SVM	0.561	0.548	0.543
		XGBoost	0.677	0.66	0.657
		LSTM	0.688	0.667	0.647
		Reputation (implicit)	0.564	0.555	0.569
Per fold	0	HMM	0.683	0.621	0.646
	0	Lgistic regression	0.584	0.576	0.5
	0	Random forest	0.579	0.553	0.562
	0	SVM	0.523	0.521	0.353
	0	XGBoost	0.603	0.595	0.587
	0	LSTM	0.578	0.608	0.563
	0	Reputation (implicit)	0.547	0.541	0.567
	1	HMM	0.704	0.689	0.649
	1	Lgistic regression	0.65	0.655	0.566
	1	Random forest	0.638	0.625	0.607
	1	SVM	0.463	0.556	0.499
	1	XGBoost	0.629	0.662	0.621
	1	LSTM	0.693	0.652	0.616
	1	Reputation (implicit)	0.6	0.585	0.591
	2	HMM	0.712	0.675	0.683
	2	Lgistic regression	0.67	0.68	0.683
	2	Random forest	0.656	0.667	0.624
	2	SVM	0.598	0.497	0.524
	2	XGBoost	0.684	0.663	0.633
	2	LSTM	0.698	0.69	0.494
	2	Reputation (implicit)	0.554	0.536	0.561
	3	HMM	0.732	0.719	0.719
	3	Lgistic regression	0.696	0.677	0.685
	3	Random forest	0.723	0.704	0.699
	3	SVM	0.63	0.542	0.566
	3	XGBoost	0.727	0.683	0.701
	3	LSTM	0.705	0.66	0.658
	3	Reputation (implicit)	0.603	0.603	0.624
	4	HMM	0.717	0.666	0.682
	4	Lgistic regression	0.676	0.655	0.664
	4	Random forest	0.688	0.653	0.644
	4	SVM	0.483	0.548	0.606
	4	XGBoost	0.68	0.649	0.64
	4	LSTM	0.698	0.631	0.688
	4	Reputation (implicit)	0.566	0.561	0.561
	5	HMM	0.696	0.669	0.679
	5	Lgistic regression	0.656	0.656	0.65
	5	Random forest	0.666	0.672	0.638
	5	SVM	0.557	0.557	0.56
	5	XGBoost	0.665	0.64	0.668
	5	LSTM	0.678	0.691	0.676
	5	Reputation (implicit)	0.557	0.54	0.553
	6	HMM	0.714	0.688	0.676
	6	Lgistic regression	0.692	0.663	0.628
	6	Random forest	0.654	0.649	0.594
	6	SVM	0.579	0.609	0.597
	6	XGBoost	0.678	0.656	0.673
	6	LSTM	0.699	0.682	0.642
	6	Reputation (implicit)	0.563	0.557	0.573
	7	HMM	0.708	0.687	0.673
	7	Lgistic regression	0.691	0.668	0.677
	7	Random forest	0.701	0.705	0.66
	7	SVM	0.582	0.561	0.618
	7	XGBoost	0.691	0.678	0.65
	7	LSTM	0.708	0.675	0.695
	7	Reputation (implicit)	0.594	0.561	0.598
	8	HMM	0.71	0.688	0.695
	8	Lgistic regression	0.682	0.682	0.682
	8	Random forest	0.722	0.714	0.691
	8	SVM	0.552	0.525	0.542
	8	XGBoost	0.705	0.699	0.703
	8	LSTM	0.693	0.672	0.696
	8	Reputation (implicit)	0.559	0.563	0.561
	9	HMM	0.726	0.698	0.711
	9	Lgistic regression	0.687	0.699	0.686
	9	Random forest	0.706	0.693	0.708
	9	SVM	0.644	0.565	0.566
	9	XGBoost	0.706	0.678	0.69
	9	LSTM	0.732	0.71	0.74
	9	Reputation (implicit)	0.498	0.5	0.499

Appendix N Adaptation of an HMM-based recommender

In section 5.3 we have used an adaptation of the HMM-based recommender system (HMM-CF) proposed by Sahoo et al. (2012). Because this HMM serves article recommendations, its structure does not directly apply in our context. Below we list the differences between our approach and the HMM-CF and identify adaptations that the HMM-CF requires to provide job-applicant recommendations:

- The HMM-CF updates user states monthly. Monthly aggregations are not relevant in our setting. Hence, we update employer preferences in the HMM-CF adaptation after each hiring decision.
- Similar to other many-assessment recommender systems (Section 2.2), HMM-CF uses items that multiple users evaluate. Hence, to apply the HMM-CF in the focal context we use the clustering of job-applications presented in Appendix A.
- The HMM-CF does not allow individual employer characteristics (i.e., vector $\mathbf{X}_{o-1}$) to affect state transitions.
- The HMM-CF includes a component of negative Binomial that identifies the number of articles to recommend per month. Then it uses multinomial emissions to choose items. In our context, the negative Binomial step is unnecessary, as we do not model how many job-applicants the employer will hire. Hence in the focal adaptation, we only use the second step of the HMM-CF that models multinomial emission and transition probabilities.
- The HMM-CF uses implicit feedback to model choices. It does not allow for performance-aware outcomes.

Based on these modifications, Section 5.3 and Appendices H and B compare the performance of the HMM-CF with our proposed approach. Our framework significantly ($p < 0.001$) outperforms the HMM-CF across all alternative evaluation measures we consider.

Appendix O Shortcomings of the LSTM approach

One of the most competitive single-assessment framework is the Long-Short Term Memory (LSTM). Yet, even this approach, which by definition is sequence aware, underperforms compared with the proposed HMM framework. In this section, we discuss the implementation details of the LSTM approach and we conceptually identify why the LSTM networks perform worse than the curated HMM.

Implementation details: To apply an LSTM framework, we need to create the sequences that the network will learn from. Specifically, we need to reshape the data such as each row is a job application and each column is a separate time series (see Multiple Input Series; Brownlee 2018). Then, we need to structure the data into samples with input and output elements. We can assume that for a job application t, the previous n job applications are also relevant. Function `split_sequences` shows this process that generates sequences of job applications. The function assumes that the parameter `n_steps` is the median of the number of applications per task (i.e., median M) in the training set. As a result, this function assumes that *ranked job applications* (independent of task and employer) generate sequences, and not tasks (which is the case in our context as shown in Figure 4). For more information regarding our LSTM implementation please visit the following files:

- `lstm_sklearn.py`: https://git.io/JmFRv
- `train_lstm.py`: https://git.io/JmF80

Discussion: The above details illustrate that it is likely impossible for the LSTM to capture employer evolution through completed tasks and at the same time provide job-application recommendations. An encoder-decoder architecture (Brownlee 2018) could potentially model task-specific sequences, but it would require hundreds of parallel columns that describe the characteristics of each application multiplied by the number of applications per task. That would potentially be fine if we had long sequences, but in our context, task sequences are short.

Furthermore, the length of sequences in our context is not (and cannot be) constant. When defining the `n_steps`, we only approximate the length of previous applications that the LSTM

will consider as relevant. This generates a lot of noise, when job-applications from different tasks become relevant (e.g., when tasks receive fewer than n_{steps} applications, and when the number of job applicants is greater than `n_steps`).

These two shortcomings explain conceptually why the LSTM cannot capture employer evolving preferences as good as our proposed framework, and why it ends up underperforming for repeat employers as shown in Figures 7 to 10 and in Figures 12 to 15.