Eliciting Worker Preference for Task Completion

Mohammad Esfandiari, Senjuti Basu Roy, Sihem Amer-Yahia

Eliciting Worker Preference for Task Completion

Mohammad Esfandiari [#1], Senjuti Basu Roy [#1], Sihem Amer-Yahia [*2]

[#] *NJIT, USA* [1] me76@njit.edu, senjutib@njit.edu

[*] *Univ. Grenoble Alpes, CNRS, LIG, France*

[2] sihem.amer-yahia@univ-grenoble-alpes.fr

***Abstract*—Current crowdsourcing platforms provide little support for worker feedback. Workers are sometimes invited to post free text describing their experience and preferences in completing tasks. They can also use forums such as Turker Nation[1] to exchange preferences on tasks and requesters. In fact, crowdsourcing platforms rely heavily on observing workers and inferring their preferences implicitly. In this work, we believe that *asking workers to indicate their preferences explicitly* improve their experience in task completion and hence, the quality of their contributions. Explicit elicitation can indeed help to build more accurate worker models for task completion that captures the evolving nature of worker preferences. We design a worker model whose accuracy is improved iteratively by requesting preferences for task factors such as required skills, task payment, and task relevance. We propose a generic framework, develop efficient solutions in realistic scenarios, and run extensive experiments that show the benefit of explicit preference elicitation over implicit ones with statistical significance.**

I. Introduction

The main actors of a crowdsourcing platform are tasks and workers who complete them. A range of studies point out the importance of designing incentive schemes, other than financial ones, to encourage workers during task completion [1], [2]. In particular, it is expected that a crowdsourcing system should "*achieve both effective task completion and worker satisfaction*". The ability to characterize the workforce with *factors that influence task completion* is recognized to be of great importance in building such a system [3], [4], [5], [6], [7], [8], [9], [10], [11]. Those efforts have focused on implicitly observing workers and inferring their preferences. In this paper, we argue that solely relying on implicit observations does not suffice and propose to *elicit preferences from workers* explicitly, as they complete tasks. Any computational model, designed for the workers to understand their task completion likelihood needs to consume worker preference. The evolving nature of worker preference requires to periodically ask workers and refine such models. To the best of our knowledge, this work is the first to examine the benefit of explicit preference elicitation from workers and its impact on effective task completion. Our proposed approach of explicit preference elicitation does not incur additional burden to the workers; in fact, platforms such as, Amazon Mechanical Turk[2] already seek worker feedback in the form of free text. Our effort is to judiciously select questions for elicitation in a stuctured and holistic fashion.

In this paper, *our objective is to design a framework that advocates for explicit preference elicitation from workers to develop a model that guides task completion*. That differs from developing solutions for task assignment. The objective behind preference elicitation is to use obtained feedback to effectively maintain a *Worker Model*. Given a task t that a worker w undertakes (either via self-appointment or via an assignment algorithm), there could be one of two possible outcomes : 1. the task is completed successfully. 2. otherwise. In reality, worker preferences are *latent*, i.e., they are to be inferred through task factors and task outcomes. Popular platforms, such as Mechanical Turk or Prolific Academic,[3], have characterized tasks using factors, such as *type, payment* and *duration*. While our framework is capable to consume any available task factors, our effort nevertheless is to propose a *generic solution that characterizes workers by understanding their preferences for a given set of task factors* [3], [4].

Our overarching goal is to seek feedback from workers to effectively maintain a *Worker Model* for task completion. To achieve this goal, the first challenge is to define an accurate model that predicts, per worker, how much each task factor is responsible for the successful completion of that task or for its failure. We propose to *bootstrap* this model by selecting a small set of tasks a worker needs to complete initially to learn her model. An equally important challenge is to update the *Worker Model*, as workers complete tasks. Indeed, unless that model is updated periodically, it is likely to become outdated, as worker's preferences evolve over time. To update the model, we advocate the need to explicitly elicit from a worker her preferences. That is a departure from the literature where workers are observed and their preferences computed implicitly. We claim that the explicit elicitation of preferences results in a more accurate *Worker Model*. Preferences are elicited via the **Question Selector** that selects a set of k task factors and asks a worker w to rank them. For example, a worker may be asked *"Rank task relevance and payment"*. A higher rank for payment will indicate the worker's preference for high paying tasks over those most relevant to her profile. Once the worker provides her preference, the *Worker Model* is updated with the help of the **Preference Aggregator**. **Question Selector** and **Preference Aggregator** constitute the two computational problems of our framework.

Worker Model. Our natural choice is to use a graphical model [12], such as a Bayesian Network where each node

[1] http://turkernation.com/

[2] https://www.mturk.com/

[3] https://www.prolific.ac/

is a random variable (task factors/worker preference/task outcome) and the structure of the graph expresses the conditional dependence between those variables. The observed variables are task factors and task outcomes and the *Worker Model* contains worker preferences in the form of latent variables that are inferred through that model. It is however known that structure learning in Bayesian Network is NP-hard [12] and that the parameters could be estimated through methods such as Expectation Maximization, that also are computationally expensive. As both **Question Selector** and **Preference Aggregator** have to invoke the model many times, it becomes prohibitively expensive to use it in real time. We therefore propose a simplified model that has a one-to-one correspondence between task factors and worker preference. The preference of a worker for a task factor is construed as a weight and the *Worker Model* becomes a linear combination of the task factors. This simplification allows us to design efficient solutions.

Question Selector. The question selector intends to select the k-task factors whose removal maximizes the improvement of the *Worker Model* $\mathcal{F}$. The idea is to present those *uncertain* factors to the worker and seek her explicit preference. We prove that optimally selecting k questions, i.e., k task factors for a worker, is NP-hard, even when the *Worker Model* is linear. We develop an efficient alternative using an iterative greedy algorithm that has a provable approximation bound.

Preference Aggregator. The second technical problem is to update the *Worker Model* with the elicited preference. Given a set of k task factors, worker w provides an absolute order on these factors. The obtained ranking is expressed as a set of $k(k-1)/2$ pairwise linear constraints, as $i > j$, $i > l$, etc. We design an algorithm that updates the *Worker Model* using the same optimization function as the one used to build it initially, modified by adding those constraints. With a Bayesian Network as the underlying model, the addition of dummy variables would encode the constraints aptly. However, with one variable per constraint, the solution would not scale. For the simplified linear *Worker Model*, we add them as pairwise linear constraints. The problem then becomes a constrained least squares problem that could be solved optimally in polynomial time.

We run experiments that measure the accuracy of our model and the scalability of our approach with real tasks and workers: $165,168$ tasks from CrowdFlower involving 58 workers from Amazon Mechanical Turk. We measure the accuracy of the *Worker Model* against several baselines: a random selection of which task factors to invoke preferences for, and implicit preference computation [13]. We show that soliciting preferences explicitly and using them to update the model greatly reduces error and largely outperforms implicit solutions with statistical significance. We also show that our approach scales well.

In summary, our contributions are:

- Problem Formalism (Section II): A framework that has a *Worker Model* that captures, per worker, her preference for task factors to predict the likelihood of task completion. We present an innovative formulation to bootstrap the model. An important aspect of the model is that it could be easily adapted to other crowdsourcing processes, such as, task assignment or worker compensation. We present two core problems around the model: **Question Selector** that asks a worker to rank the k task factors that cause the highest error in the model, and **Preference Aggregator** that updates the model with elicited preferences.
- Technical Results (Section III): We study the hardness of our problems and their reformulation under realistic assumptions, as well as design efficient solutions with provable guarantees.
- Experimental Results (Section IV): We present extensive experiments that corroborate that explicit preference elicitation outperforms implicit preferences [13], and that our framework scales well.

II. Formalism and Framework

We present our formalism, following which we provide an overview of the proposed framework and the problems we tackle.

Example 1: We are given a set of tasks, where each task is characterized by a set of factors (e.g., *type, payoff, duration* are some examples). A task could be of different types, such as, image tagging, ranking, sentiment analysis. Payoff determines the \$ value the workers receives as payment, whereas, duration is an indication of the time a worker needs to complete that task. Generalizing this, one can imagine that each task could be described as a vector of different factors and a set of tasks together gives rise to a task factor matrix $\mathcal{T}^f$. One such matrix of 6 tasks is presented below:

id	$tagging$	$ranking$	$sentiment$	$payoff$	$duration$	$outcome$
t_1	1	0	0	$high$	$long$	1
t_2	1	0	0	low	$short$	0
t_3	0	1	0	low	$short$	1
t_4	0	1	0	low	$long$	0
t_5	0	0	1	$high$	$short$	1
t_6	0	0	1	$high$	$long$	0

Given a task t that a worker w undertakes (either via self-appointment or via an assignment algorithm), there could be one of two possible outcomes : 1. the task is completed successfully (denoted by 1). 2. otherwise (denoted by 0). The last column of the task factor matrix indicates outcomes of each of the tasks. These could be known from prior history or predicted using a mathematical model.

A. Formalism

Task Factors. In a crowdsourcing platform, each task t in a set of n given tasks $\mathcal{T} = \{t_1, ..., t_n\}$ is characterized by a set of m factors whose values are either explicitly present or could be extracted. For this work, we assume that for a task t, its factors give rise to a vector $\vec{t^f}$ that is given and we do not focus on how to obtain them. This gives rise to the task-factor matrix $\mathcal{T}^f$ of dimension $n \times m$. For the simplicity of exposition, task factors are presented as binary, although our proposed solutions adapt when they are continuous or categorical.

Using Example 1, the task factor matrix consists of 6 tasks and each task is described by 5 factors.

Worker Preferences. The preferences of a worker w are represented by a vector $\vec{w}^f$ of length p that takes real values and determines the preferences of w for tasks. Using Example 1, $\vec{w}^f$ could be represented as *latent variables*, such as, {*skill, motivation, reputation*}. The correspondence between task factors and worker preference is surjective (e.g., task type $\rightarrow$ skill, $\{duration, \text{payoff}\} \rightarrow$ motivation). Worker preference variables cannot be observed and must be inferred.

Explicit Questions. An explicit question is asked to elicit w's preference on a particular task factor, as the tasks dictate worker preference and hence her performance thereof. In fact, there is an one to one correspondence between the questions and task factors. A set of k questions is asked to obtain a preferred order among a set of k task factors (where k is part of the input). As an example, one may ask to rank *"task duration, tagging tasks, ranking tasks, sentiment analysis tasks, payment"*. A worker provides an absolute order among these 5 factors as her preference. The ranking can be simply interpreted as the worker's preference for the first factor followed by the second, etc. For example, if the worker ranks *payment* first, she means a preference for high paying tasks to any others including those most relevant to her profile.

B. Framework and Challenges

We propose an iterative framework (refer to Figure 1) that is designed to ask personalized questions to a worker to elicit her preference in a crowdsourcing platform. The rationale for our proposal is that while task factors are stable, a worker's preference evolves as workers undertake tasks. Workers' skills improve as they complete tasks and their motivation varies during task completion [10], [13]. How to define an iteration is orthogonal to our problem. Indeed, an iteration could be defined by discretizing time into equal-sized windows, by the number of available/completed tasks, by the number of workers, or a combination thereof. We will see in Section IV how we define an iteration in our experiments.

Central to our framework is a model $\mathcal{F}$ that consumes task factors and given a worker's history, infers her preference vector to predict the outcome of a new task to be undertaken by her. Even though we develop a worker model to predict task completion quality, this information could be used in many places to characterize the workforce of a crowdsourcing platform and enable several improvements such as the analysis of workers' fatigue [10] and motivation, as well as task assignment [14], [15], [16], [17], [13]. However, unless the *Worker Model* is refreshed or updated periodically, it is likely to become outdated, as worker preference evolves over time [10], [4], [13]. To update the model, one has to periodically invoke an explicit preference elicitation step through **Question Selector** that selects a set of k task factors and asks worker w to rank them. Once the worker provides her preference, the *Worker Model* is updated by the **Preference Aggregator**. These two components form the heart of our computational challenges. *The first is to quickly select the bst set of k factos to invoke feedback for. The second is to quickly relearn the model while satisfying the answers the worker has provided.*

Notation	Definition
$\mathcal{T}$	a set of tasks $\{t_1, \ldots, t_n\}$
$\vec{t}^f$	a vector describing task factors
$\mathcal{T}^f$	task factor matrix
$\vec{w}^f$	worker w's preference vector
$\mathcal{Q}$	a set of questions (task factors)
$\mathcal{Q}^s$	a set of selected k questions
t^O	a label signaling task outcome
$\mathcal{T}^O$	labels signaling task outcomes of a set $\mathcal{T}$
$\mathcal{B}$	a set of b tasks for bootstrapping
$\mathcal{E}$	reconstruction error of the model $\mathcal{F}$

TABLE I
TABLE OF IMPORTANT NOTATIONS

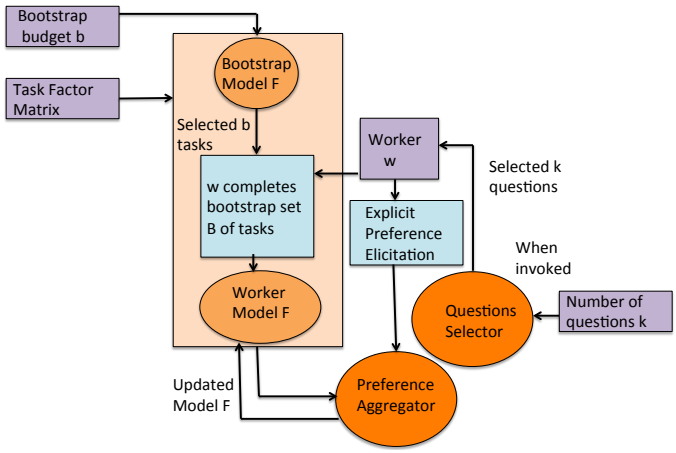


Fig. 1. Explicit Preference Elicitation Framework

The remainder of the paper focuses on a particular worker w, unless otherwise stated - i.e., each of the components of the framework is designed or invoked for her.

C. Problem Definitions

Worker Model. Given the task factor matrix $\mathcal{T}^f$ of a set of tasks $\mathcal{T}$, where each task t is associated with a known outcome ($t^O = 1/0$ successfully completed or not), the *Worker Model* $\mathcal{F}$ estimates the preference vector $\vec{w}^f$ of a worker w. $\mathcal{F}$ is a function of $\mathcal{T}^f$ and $\vec{w}^f$, denoted as $\mathcal{T}^f \otimes \vec{w}^f$. The correctness of $\mathcal{F}$ is estimated using *its reconstruction error*, i.e., $\mathcal{E}(\mathcal{T}^O - (\mathcal{T}^f \otimes \vec{w}^f))$ is minimized.

Once the model is built, it can predict the outcome of a task undertaken by w. On Example 1, the model predicts the likely outcome of each of the 6 tasks, by consuming $\mathcal{T}^f$ and inferring $\vec{w}^f$.

Bootstrapping. Initially, as no past history of a worker w is available, she is treated akin to a "cold user" and the *bootstrap process* selects a subset of tasks, $\mathcal{B} \subset \mathcal{T}$ to be completed by w. The size of $\mathcal{B}$, denoted by b, is given as a budget to bootstrapping. The idea is to leverage the contributions of w for those b tasks to estimate her preference and build the *Worker Model* $\mathcal{F}$. During bootstrapping, the algorithm is *offline*, i.e., it apriori decides all b tasks, without actually

observing outcomes of tasks completed by w. For that, it selects the b tasks whose feedback minimizes the *expected reconstruction error* over the remaining $\mathcal{T} - \mathcal{B}$ tasks, i.e., $\mathbb{E}((\mathcal{T} - \mathcal{B})^O - ((\mathcal{T} - \mathcal{B})^f \otimes \vec{w}^f))$ is minimized. The same approach is adopted in subsequent steps to refine $\mathcal{F}$.

Given Example 1 with $b = 3$, the objective would be to select 3 tasks that minimize the expected reconstruction error.

We are now ready to describe the two fundamental problems that our system needs to solve.

Question Selector. This module selects the best set of k questions for a worker w. The objective is to select those task factors that are responsible for the model's inaccuracy, i.e., removing them would improve the most the reconstruction error of $\mathcal{F}$. Let $\mathcal{E}$ denote the current reconstruction error of $\mathcal{F}$ and $\hat{\mathcal{E}}$ denote it when k task factors are removed. Given $\mathcal{Q}$, the k questions are selected such that the model reconstruction error improves the most, i.e., $argmax_{\{\mathcal{Q}^s \in \mathcal{Q}: |\mathcal{Q}^s| = k\}}(\mathcal{E} - \hat{\mathcal{E}}_{m-\mathcal{Q}^s})$.

Using Example 1, if $k = 2$, this will select any two of the five task factors in the task factor matrix.

Preference Aggregator. The preferences $\mathcal{P}$ provided by a worker for task factors, could be expressed as *a set of* $k(k-1)/2$ *pairwise linear constraints* of the form, $i > j$, $i > l$, $j > l$. Worker's preferences are taken as hard constraints. Given $\mathcal{P}$, the objective is to relearn $\mathcal{F}$ that satisfies $\mathcal{P}$ such that its reconstruction error is minimized. The objective therefore is to minimize $\mathcal{E}(\mathcal{T}^O - (\mathcal{T}^f \otimes \vec{w}^f))$ such that the constraints in $\mathcal{P}$ are satisfied.

Using Example 1, if worker w explicitly states that she prefers `annotation` tasks to `ranking` tasks, this preference is translated into constraints expressed on the worker preference vector. Those are then used by the preference aggregator to update $\mathcal{F}$.

III. Solutions

We now propose a generic approach for building and bootstrapping the *Worker Model* and for solving the two computational problems enunciated in Section II, namely, **Question Selector** and **Preference Aggregator.** In each case, we present our generic approach and then develop a simplified, yet realistic framework, that enables efficient solutions with theoretical guarantees.

A. Worker Model

As described in Section II-B, central to our framework is a supervised model $\mathcal{F}$, designed for a worker w, that consumes task factors and predicts task outcomes by inferring w's preferences for those factors. There are several machine learning models that could potentially be used. A natural choice is a probabilistic graphic model [12], such as, Bayesian Networks, where each node is a random variable (task factors/worker preferences/task outcome) and the structure of the graph expresses the conditional dependence between them. In particular, it could be expressed as a Bayesian Network, which is represented as a directed acyclic graph (DAG). As described in Figure 2, the observed variables are task factors and task outcomes and the worker preferences are captured as latent variables that are inferred.

Furthermore, we can impose constraints, such as, the task factors are independent from each other, and the worker preference variables are correlated/not, or the worker preference variables determine task outcomes, which are in turn dependent on the task factors. Given a set of tasks, each with an outcome and associated vector of factors, the model $\mathcal{F}$ corresponds to the factorization of the joint probability distribution over these variables:

$$\begin{aligned} Pr(\text{task factors, worker preferences, outcome}) = \\ Pr(outcome|\text{worker preferences}) \times \\ \prod_{i=1}^{p} Pr(\text{worker preference i}|\text{parent task factors of i}) \\ \times \prod_{j=1}^{m} Pr(\text{task factor j}) \end{aligned} \quad (1)$$

There are two primary computational difficulties when building $\mathcal{F}$: learning the structure of the graph and estimating parameters to obtain a probability distribution at each node. The overall objective is to minimize reconstruction error. Once the model is built, we use it for inference, i.e., it will infer the probability distribution over a worker's preference, given the values of task factors and task outcomes.

The proposed non-linear model is prohibitively expensive to compute. The structure learning part is known to be NP-hard [12] while the parameters could be estimated through methods such as Expectation Maximization, also computationally expensive. The known efficient algorithms that propose a workaround are primarily heuristics [18] without approximation guarantees.

Therefore, we suggest to *simplify our model under the assumption that the preference of a worker for a task factor is a weight and the Worker Model is a linear combination of a given set of task factors*. As a result, the *Worker Model* can be expressed as a linear regression function. A one-to-one correspondence between task factors and worker preferences (# task factors = length of $\vec{w}^f$, i.e., $p = m$) would treat worker preferences as weights that are to be estimated. The model takes the form,

$$\mathcal{T}^O = \vec{w}^f \times \mathcal{T}^f$$

Given the task factor matrix $\mathcal{T}^f$ and the observed task outcome vector $\mathcal{T}^O$, the objective is to estimate $\vec{w}^f$, such that the reconstruction error is minimized, i.e.,

$$\underset{\vec{w}^f \in \mathbb{R}^m}{\operatorname{argmin}} ||\mathcal{T}^O - \vec{w}^f \times \mathcal{T}^f||_2 \quad (2)$$

Algorithm for Worker Model. As long as the task factor matrix $\mathcal{T}^f$ is invertible, or could be inverted by adding an additional term [19], the objective function expressed in Equation 2 has an equivalent and alternative representation of the form $\left(\mathcal{T}^{f^T}\mathcal{T}^f\right)^{-1}\mathcal{T}^{f^T}\mathcal{T}^O$, where $\left(\mathcal{T}^{f^T}\mathcal{T}^f\right)^{-1}\mathcal{T}^{f^T}$ is known as the Moore-Penrose pseudo-inverse matrix of $\mathcal{T}^f$. The proof could be found in [20].

We design an ordinary least squares (OLS) [21] solution to estimate the regression coefficient or worker preferences $\vec{w}^f$.

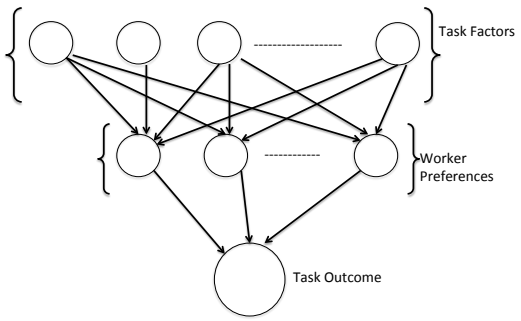


Fig. 2. Worker Model

It first transforms the objective function in Equation 2 to a Moore-Penrose pseudo-inverse matrix of $\mathcal{T}^f$. This equivalent representation is proved to have a global minimum, hence the obtained OLS estimator is optimal. The overall running time of OLS is dictated by matrix multiplication (multiplying $({\mathcal{T}^f}^T\mathcal{T}^f)$, Matrix inversion (inverting ${\mathcal{T}^f}^T\mathcal{T}^f)^{-1}$, followed by matrix multiplication (to obtain $({\mathcal{T}^f}^T\mathcal{T}^f)^{-1}{\mathcal{T}^f}^T$), and a final matrix multiplication (to obtain $({\mathcal{T}^f}^T\mathcal{T}^f)^{-1}{\mathcal{T}^f}^T$). The asymptotic complexity is $\mathcal{O}(m^2n + m^3)$.

Bootstrapping the Worker Model. Unfortunately, the *Worker Model* can not be built for a brand new worker, who has not completed any task before in the platform. For such workers, we propose to bootstrap the *Worker Model*. Bootstrapping is a common practice in recommender systems and is adopted by largely used platforms such as Netflix and MovieLens.[4]. The objective in our case is to select the set of tasks $\mathcal{B}$ and learn the *Worker Model* $\mathcal{F}$ that minimizes the *expected reconstruction error* over the remaining $\mathcal{T} - \mathcal{B}$ tasks. For a selected set of b tasks, we compute a set of 2^b *Worker Models*, where each model is learned by encoding one of the 2^b possible combinations of the task outcomes, and capturing the reconstruction error over the remaining set of $\mathcal{T} - \mathcal{B}$ based on the learned model. This gives rise to a bootstrapping tree with 2^b branches, as the one shown in Figure 3. Each branch is associated with a probability value that represents the probability of that combination of b task outcomes. As an example, in Figure 3, the leftmost branch captures the model where all three tasks would have a successful completion, the branch represents that probability, and the corresponding leaf represents the reconstruction error.

Generic and Simplified Probability Model for Bootstrapping. To capture the probability of a branch in the bootstrap tree, we need to calculate the probability of successful/unsuccessful outcome of each task by the "cold" worker. Technically, we are interested to compute $Pr(t^0 = 1|\vec{t^f}, w)$ (probability of successful completion) and $Pr(t^0 = 0|\vec{t^f}, w)$ (probability of unsuccessful completion). In a real crowdsourcing platform, however, there is little to no information available about a new worker that could be potentially used to capture similarity between her and other existing workers in the system. Thus, the only way to obtain this information is to see if there are *past tasks with similar characteristics* (although undertaken by different workers) and analyze the outcome of those tasks. Therefore, these values should rather rely only on task factors, $Pr(t^0 = 1|\vec{t^f})$ and $Pr(t^0 = 0|\vec{t^f})$ should be computed as the joint distribution of the task factors and outcomes. Using Example 1, if we assume that the task factors, such as, duration, payoff, and the task types are correlated with each other, then the outcome of a task relies on the joint distribution over these factors.

The optimal solution of this problem could be obtained by computing the *joint distribution* using a structure similar to a contingency table where each cell represents a possible set of values for each of the m task factors and the value of that cell represents the probability of successful completion. Classical algorithms such as iterative proportional fitting (or IPF) [22], could be used for estimating this joint distribution. Unfortunately, these algorithms do not scale when the number of factors and their possible values are large [23].

Given a task t, $Pr(t^0 = 1|\vec{t^f})$ (probability of successful completion) and $Pr(t^0 = 0|\vec{t^f})$ (probability of unsuccessful completion), s the joint distribution over the task factors and outcomes is very expensive, as this will require us to compute a joint distribution over a v^m space, if each of the m task factor variables takes v possible values. A more realistic probability model relies on a conditional independence assumption. It assumes that the task factors are themselves independent but the task outcome is conditionally dependent on each of the task factors. This Bayesian assumption is not an overstretch. Using Example 1, one can see that the different task types, annotation, ranking, or sentiment analysis are independent from each other, as is duration, but the task outcome is conditionally dependent on each of these factors. In practice, some dependence may exist between factors, e.g., task duration and payment may be correlated. However, that is not the case in general as tasks are posted by different requesters and in different countries. Therefore, we can confidently claim that a conditionally independent probability model captures a large number of cases in practice.

Under the conditional independence assumption, we have

$$Pr(t^0 = 1|\vec{t^f}) = \prod_{i \in \vec{(t)}^f} Pr(t^0 = 1|t^{(i)}) \tag{3}$$

Using Bayes' Theorem, this could be rewritten as

$$Pr(t^0 = 1|\vec{t^f}) = \prod_{i \in \vec{(t)}^f} \frac{Pr((t^0 = 1)|t^{(i)}) \times Pr(t^0 = 1)}{Pr(t^{(i)})}$$

Computing the probability formula requires us to know the value of quantities such as $Pr((t^0 = 1)|t^{(i)})$, $Pr(t^0 = 1)$, and $Pr(t^{(i)})$. However, singleton and pairwise probabilities can be computed in a pre-processing step considering other tasks and workers. For example, $Pr((t^0 = 1)|t^{(i)})$ can be estimated as the fraction of previous tasks with successful outcomes that also have the i-th factor as $t^{(i)}$ and $Pr(t^0 = 1)$ can be estimated as the fraction of tasks with successful outcomes, whereas, $Pr(t^{(i)})$ is the fraction of tasks that have $t^{(i)}$ as the

[4]https://www.netflix.com/,https://movielens.org/

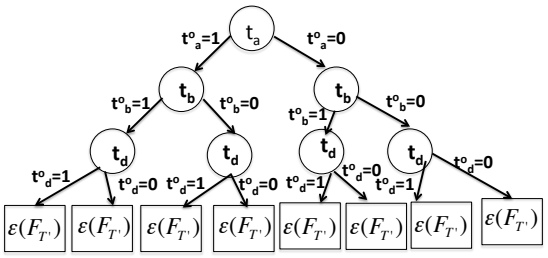


Fig. 3. Bootstrapped tree of three chosen tasks $\{t_a, t_b, t_d\}$

i-th factor. Each of these calculations are efficient and could be done in a pre-processing phase.

Assuming independence among tasks, the probability of a combination of outcomes of b tasks is then obtained by multiplying the probabilities of the individual task outcomes. Considering Figure 3, the probability of the left most branch is $Pr({t_a}^0 = 1|\vec{t}^f) \times Pr({t_b}^0 = 1|\vec{t}^f) \times Pr({t_d}^0 = 1|\vec{t}^f)$

Bootstrapping Algorithm. Given a budget of b tasks, these tasks are chosen a priori, therefore, one has to explore all possible $\binom{n}{b}$ tasks, and for each task, compute its two possible outcomes probabilistically. This gives rise to an exponential search space of $\binom{n}{b}$ possible choices. Even when the best set of b tasks are chosen, each task has one of two possible outcomes, which gives rise to a bootstrap tree with 2^b branches [24], as shown in the Figure 3. Each branch from the root to the leaf corresponds to a set of b tasks and their outcomes, and each leaf is associated with a reconstruction error. The reconstruction error of the branch is the product of the respective probabilities of the outcomes and the reconstruction error of the model $\mathcal{F}$ thereof. The expected reconstruction error is the sum of the errors over all 2^b branches. The objective, as mentioned in Section II-C, is to design the tree that improves the expected reconstruction error the most.

Theorem 1: Bootstrapping the *Worker Model* is NP-hard.

Proof: (Sketch:) If the underlying model is a graph whose structure computation is NP-hard [12], naturally the bootstrapping becomes NP-hard, as it has to invoke that model as a subroutine. Even for an arbitrarily simple model that is polynomial time computable, the NP-hardness could be proved using a reduction from the Set Cover Problem, even when each factor is only binary [25]. ■

We now design an algorithm that is greedy in nature and avoids searching the $\binom{n}{b}$ space to select the b tasks. It runs in b iterations and in the i-iteration selects the task out of the remaining set of tasks that has the highest marginal improvement over the objective function. However, even when the b tasks are selected, computing the bootstrap tree is exponential in b (recall Figure 3). This algorithm makes $\mathcal{O}(n \times b)$ comparisons to select the best set of b tasks. However, while the greedy selection is in progress, given an already selected b' tasks, it still has to build the bootstrap tree with $2^{b'}$ branches. The worst case asymptotic complexity is therefore, $\mathcal{O}(n \times b \times 2^b)$.

Running Example: Using Example 1, tasks $\{t_1, t_3, t_6\}$ are chosen for bootstrapping. Once $\mathcal{F}$ is developed, it assigns the following weights to the task factors. tagging=0.4, ranking= 0.69, sentiment=0.1, payoff=0.4, duration =0.42. This is intuitively explainable, as the tasks that the worker complete successfully are tagging and ranking (hence gets higher weights), but the sentiment analysis task is not completed successfully (thus, gets a lower preference value).

B. Question Selector

The objective is to select k questions (task factors) eliminating which would maximize the reconstruction error reduction of the model $\mathcal{F}$. Ideally, out of m task factors (a set $\mathcal{Q}$ of questions), k factors should be chosen as a set. This gives rise to an exponential search space that could be modeled using a decision tree like structure with [24], $\binom{m}{k}$ possible branches. Each branch from the root to the leaf corresponds to a set of k questions, and the leaf is associated with a reconstruction error. The objective, as mentioned in Section II-C, is to design that tree that improves the reconstruction error the most.

Theorem 2: Optimally selecting k questions to elicit preferences is NP-hard.

Proof: (Sketch:) A careful review of the objective function (refer to Section II-C) shows that since $\mathcal{E}$ is a constant at a given point - thus, maximizing $(\mathcal{E} - \hat{\mathcal{E}}_{m-\mathcal{Q}^s}) : \{\mathcal{Q}^s \in \mathcal{Q} : |\mathcal{Q}^s| = k\}$ is same as minimizing the reconstruction error of $\hat{\mathcal{E}}_{m-\mathcal{Q}^s}$, i.e., retaining the best $m-k$ factors (thus eliminating the worst k factors) that has the smallest reconstruction error of $\mathcal{F}$. The problem thus becomes selecting the best $m-k$ factors that have the smallest reconstruction error. The remaining k factors would therefore be chosen as the explicit questions for preference elicitation.

The reduction is done using the Set Cover problem. We omit the details for brevity, but elaborate later on that the problem remains NP-hard even when we consider a simple *Worker Model*. ■

Efficient Solution for Question Selector. Since $\mathcal{E}$ is a constant at a given point, maximizing $(\mathcal{E} - \hat{\mathcal{E}}_{m-\mathcal{Q}^s}) : \{\mathcal{Q}^s \in \mathcal{Q} : |\mathcal{Q}^s| = k\}$ is same as minimizing the reconstruction error of $\hat{\mathcal{E}}_{m-\mathcal{Q}^s}$, i.e., retaining the best $m-k$ factors (thus eliminating the worst k factors) that has the smallest reconstruction error of $\mathcal{F}$. The problem thus becomes selecting the best $m-k$ factors that have the smallest reconstruction error. The remaining k factors would therefore be chosen as the explicit questions for preference elicitation.

Theorem 3: Optimally selecting k questions for explicit worker preference is NP-hard even when the *Worker Model* is linear.

Proof: (Sketch): When a linear model such as the one in Equation 2 is assumed, as described above, the objective function is equivalent to minimizing $\left({\mathcal{T}^f}^T \mathcal{T}^f\right)^{-1} {\mathcal{T}^f}^T \mathcal{T}^O$, where $\left({\mathcal{T}^f}^T \mathcal{T}^f\right)^{-1} {\mathcal{T}^f}^T$ is known as the Moore-Penrose pseudo-inverse matrix of $\mathcal{T}^f$.

Therefore, the problem of identifying and removing the k worst factors, i.e., retaining the best $m-k$ factors, is akin to selecting a subset of $m-k$ columns from the task factor matrix $\mathcal{T}^f$ such that the pseudo-inverse of this sub-matrix has the smallest norm. Under the Frobenius or L_2 norms this problem is proved to be NP-hard [19].

Using the NP-hardness proof described in [19], our reduction is rather simple. Given an instance of that problem, we set k (the k worst factors to remove) as the difference between the total number of columns and k' (k'= the best set of k' columns giving rise to the submatrix whose pseudo-inverse has the smallest norm). The rest of the proof is trivial and omitted for brevity. ■

Greedy Algorithm for Question Selector. Under the linear model such as the one described in Equation 2 and its equivalent representation using a pseudo-inverse matrix, the objective of identifying the set $\mathcal{Q}_s$ of k selected questions (thereby identifying $m-k$ best factors) out of a set $\mathcal{Q}$ of m questions (a task factor is a question) is equivalent to retaining the task factor submatrix with $m-k$ columns that is of the following form [26]:

$$\underset{\mathcal{Q}_s \subset \mathcal{Q}, |\mathcal{Q}_s|=k}{\operatorname{argmin}} \; Trace({\mathcal{T}^f}^T_{Q\setminus Q_s} \mathcal{T}^f_{Q\setminus Q_s})^{-1} \tag{4}$$

Algorithm 1 Algorithm `k-ExFactor`: Greedy Question Selector for a Linear Model

Require: Task factor matrix $\mathcal{T}^f$, set of questions $\mathcal{Q}$
Ensure: $\mathcal{Q}^s$ with k factors

1: $\mathcal{T}_Q \leftarrow \mathcal{T}^f$
2: $\mathcal{Q}^s \leftarrow \mathcal{Q}$
3: **for** $j \leftarrow 1$ to k **do**
4: $\quad q_j \leftarrow argmin_{q \in \mathcal{Q}}$ Trace$(\mathcal{T}_Q {}^T_{Q\setminus q} \mathcal{T}_{Q\setminus q})^{-1}$
5: $\quad \mathcal{T}_Q \leftarrow \mathcal{T}_{Q\setminus j}$
6: $\quad \mathcal{Q}^s \leftarrow \mathcal{Q} \setminus q_j$
7: **end for**
8: Return $\mathcal{Q} - \mathcal{Q}^s$

We now describe a greedy algorithm `k-ExFactor` to identify k worst task factors (thus retaining $m-k$ best factors). Our algorithm makes use of Equation 4 and has a provable approximation guarantee. It works in a backward greedy manner and eliminates the factors iteratively. It works in k iterations, and in the i-th iteration, from the not yet selected set of factors, it selects a question q_j and eliminates it which marginally minimizes Trace$({\mathcal{T}^f}^T_{Q\setminus f_j} \mathcal{T}^f_{Q\setminus f_j})^{-1}$. Once the k^{th} iteration completes the eliminated k questions are the selected k-factors for explicit elicitation. The pseudo code of the algorithm is presented in Algorithm 1. Line 4 in Algorithm 1 requires a $\mathcal{O}(m^2n + m^3)$ time for matrix multiplication and inversion for the question under consideration. Therefore, the overall complexity is $\mathcal{O}(km^2n^2 + m^3nk)$. Notice that most of the complexity is actually in the process of recomputing the model error and the actual question selection is rather efficient.

Theorem 4: Algorithm `k-ExFactor` has an approximation factor of $\frac{m}{m-k}$.

Proof: (sketch): The proof adapts from an existing result [27], [28] that uses backward greedy algorithm for *subset selection* for matrices and retains a given smaller number of columns such that the pseudo-inverse of the smaller sub-matrix has as smallest norm as possible. This is akin to removing k worst task factors and retaining the best $m-k$ factors and the proof is a simple adaptation of [27], [28]. Exploration of a better approximation factor is deferred to future work. ■

Running Example: Using Example 1, if $k = 3$, {sentiment, Payoff, Duration} are the three task factors for which worker feedback is solicited.

C. Preference Aggregator

The second technical problem is to update $\mathcal{F}$ with the worker's preferences. Given a set of k task factors, worker w provides an absolute order on these factors. We design an algorithm that updates $\mathcal{F}$ using the same optimization function as the one used to build it initially, modified by adding the set of constraints that represent obtained preferences.

The worker provides a full order among the selected questions (task factors) in the form $i > j > r > l$. We express this full order using a set of pairwise constraints of the form $i > j$. If the preferences contain a full order among k constraints, this gives rise to a total of $k(k-1)$ linear pairwise constraints.

Updating a Bayesian Network with constraints has been studied in the past [29]. The idea is to add additional dummy variables that encode the constraints aptly. To satisfy $k(k-1)/2$ pairwise constraints, we will have to add that many number of dummy variables which considerably blows the size of the network. Unfortunately, such algorithms are expensive and unlikely to scale.

Efficient Solution for Preference Aggregator. For the simplified *Worker Model*, with the linear constraints added to our objective function in Equation 2, the preference aggregation problem becomes a constrained least squares problem.

Specifically, our problem corresponds to a box-constrained least squares one as the solution vector must fall between known lower and upper bounds. The solution to this problem can be categorized into active-set or interior-point [30]. The active-set based methods construct a feasible region, compute the corresponding active-set, and use the variables in the active constraints to form an alternate formulation of a least squares optimization with equality constraints [31]. We use the interior-point method that is more scalable and encodes the convex set (of solutions) as a barrier function. It uses primal Newton Barrier method to ensure the KKT equality conditions to optimize the objective function [30]. The primal Newton Barrier interior-point is iterative and the exact complexity depends on the barrier parameter and the number of iterations, but the algorithm is shown to be polynomial [31].

Running Example: Using Example 1 again, if the worker says that she prefers `Duration > Sentiment > Payoff`, then the new weights that the preference aggregator estimates for $\mathcal{F}$ are, tagging=0.1, ranking= 0.1, sentiment=0.12, payoff=0.11, duration=0.97. Notice that the order of the task factors provided by the worker is satisfied in the updated model.

IV. Experimental Evaluations

We describe our experimental setup, steps, and findings in this section. All the algorithms are implemented in Python 3.5.1. The experiments are conducted on a machine with

Intel Core i7 4GHz CPU and 16GB of memory with Linux operating system. All the numbers are presented as an average of 10 runs. We run both quality and scalability experiments and implement several baselines.

A. Dataset Description

We use $165,168$ CrowdFlower micro-tasks. A task belongs to one of the 22 different categories, such as, tweet classification, searching information on the web, audio transcription, image tagging, sentiment analysis, entity resolution, or extracting information from news. Each task type is assigned a set of keywords that best describe its content and a payment, ranging between $\$0.01$ and $\$0.12$. Our tasks are *micro-tasks* that take less than a minute to complete.

Initially, we group a subset of micro-tasks into 240 HITs and publish them on Amazon Mechanical Turk. Each HIT contains 20 tasks and has a duration of 30 minutes. When a worker accepts a HIT, he is redirected to our platform where he completes the tasks. A worker may complete several HITs in a work session. Workers get paid for every *micro-task* completed.

To qualify for our experiment, we require the workers to have previously completed at least 100 HITs that are approved, and to have an approval rate above 80%. Overall, 58 different workers complete tasks. When a worker is hired for the first time, she is asked to select a set of keywords from a given list of keywords that capture her preferences.

The task types along with other factors, such as, payment and duration, form the task factors. Our original data has 41 task factors that are categorical or binary; after binarization of all the categorical factors, we obtain a total of 100 factors. The length of any worker preference vector is therefore 100. Of course, our proposed framework adapts even when the task factors are continuous.

Ground Truth. Each micro-task has a known ground-truth. If a task, undertaken by a worker is completed successfully (i.e., the outcome matches the ground-truth), it is marked successful (value 1). Otherwise, if the task is accepted but either not finished or not completed correctly, its label is unsuccessful (value 0). This information is used as the ground-truth for the *Worker Model*.

Iteration. We define an iteration as the completion of a HIT. When a worker finishes a HIT, we compute the error in the *Worker Model*. We update the *Worker Model*, when there is a non-zero error and start another iteration.

B. Implemented Algorithms

We now describe the algorithms that are implemented and compared for evaluation purposes.

1) Worker Model & Bootstrapping: The linear model in Section III-A is implemented with a regularization parameter α. When implementing statistical models, this is a standard practice to avoid overfitting of the model. The overall objective function thus becomes,

$$\min_{\vec{w}^f \in \mathbb{R}^m} \left\| \mathcal{T}^O - \vec{w}^f \times \mathcal{T}^f \right\|_2 + \alpha \left\| \vec{w}^f \right\|_2^2 \tag{5}$$

The best value of α is chosen by generalized cross validation [30].

Additionally, there are three algorithms that are implemented for bootstrapping the *Worker Model*.

Random Bootstrapping. `RandomBoot` selects a random subset $\mathcal{B}$ of data as the initial tasks to present to the worker and records their outcome to estimate the *Worker Model*.

Uniform Bootstrapping. `UniformBoot` does not learn anything to build the *Worker Model* but bootstraps the model by assigning uniform weights to the worker preference vector.

Optimization-Aware Bootstrapping. `OptBoot` implements our algorithm given in Section III-A.

2) Explicit Feedback: This has two important components - one is the **Question Selector** that selects the task factors for explicit preference elicitation, the other is **Preference Aggregator** that updates the *Worker Model* using elicited preferences.

Optimization-Aware Question Selector. `k-ExFactor` is our proposed algorithm described in Section III-B.

k-random Question Selector. `k-Random` is a simple baseline that randomly selects k-task factors for preference elicitation.

Preference Aggregator: This is our implemented solution for preference aggregation, as described in Section III-C.

3) Implicit Feedback: We also implement implicit feedback computation to serve as a comparison alternative to explicit feedback approaches. Algorithm `Implicit-1` is an adaption of a recent related work [13] that investigates how to implicitly capture worker motivation and use that for task assignment. While we do not necessarily focus on motivation as a factor in this work, we adapt the algorithm in [13] to estimate and update the worker preference vector over time. Since our focus is not on task assignment, once we estimate the worker preference vector using `Implicit-1`, we use that in conjunction with our *Worker Model* to predict a task outcome. Algorithm `Implicit-2` is a further simplification. It relearns the *Worker Model* at the end of every iteration as the worker completes tasks and does not factor in the preference of the worker in the *Worker Model*.

C. Summary of Results

There are two primary takeaways:

1. Our proposed explicit preference elicitation framework outperforms existing implicit ones with statistical significance. We compare our approach `k-ExFactor` for question selection with another explicit baseline `k-Random` and two implicit preference computation algorithms `Implicit-1` [13] and `Implicit-2`. For qualitative evaluation, we present error (mean square error or MSE) with statistical significance test and find that `k-ExFactor` convincingly and significantly outperforms the other three baselines under varying parameters: # iterations, # task factors, k. For bootstrapping, we compare our solution `OptBoot` with two baselines `UniformBoot` and `RandomBoot`. Again, our observation here is `OptBoot` is superior qualitatively.

2. **Our proposed solution is scalable.** In our scalability study, we vary # tasks, # task factors, k, and bootstrap sample

size. While k-ExFactor is slower than the other three algorithms, as it performs a significantly higher number of computations, it still scales very well. Unsurprisingly, our bootstrapping algorithm OptBoot is slower than the two other baselines. Despite that, it scales reasonably well. These results demonstrate the effectiveness of eliciting explicit preferences making our framework usable in practice.

D. Quality Experiments

The objective of these experiments is to capture the effectiveness of our explicit feedback elicitation framework and compare it with appropriate baselines. Unless otherwise stated, we capture effectiveness as reconstruction error, according to the objective function in Section II-C using Mean Square Error or MSE.

Invocation of the Framework. For quality experiments, the proposed framework is invoked iteratively as follows: in the beginning, we filter out the tasks and task completion history by worker id since the framework is personalized per worker. On average, a worker undertakes 100 tasks. We randomly sample 70% of each worker's data for training and the rest as the holdout set.

For bootstrapping, the *Worker Model* is initially built by selecting a subset $\mathcal{B}$ of b tasks from the training data and MSE is computed over the holdout set. After that, in an iteration, we select a set x of 25 tasks (unless otherwise stated), randomly from the holdout set and we calculate the score over the remaining set of tasks in the holdout set. Next, we will check if there is an error in the prediction of *Worker Model* for set x. If yes, then we invoke the **Question Selector** that seeks explicit feedback from the same worker. Upon receiving worker feedback, the *Worker Model* is updated using the **Preference Aggregator**. All these steps construe a single iteration of the framework. We periodically perform the aforementioned steps to get multiple iterations of the framework.

Parameter Setting. For a given worker, there are three parameters to vary: # task factors, k, and # iterations. For bootstrapping, we additionally vary the budget b. Unless otherwise stated, defaults values are 90, 4, and 7, respectively. The best 90 features are retained by performing feature selection using *Chi-squared test*[32]. We also notice that the error does not reduce significantly beyond $k = 4$ questions and 7 iterations. By default, we always maintain the full history of worker preference while updating the *Worker Model* under varying iterations and the default size of the bootstrapping set is 15.

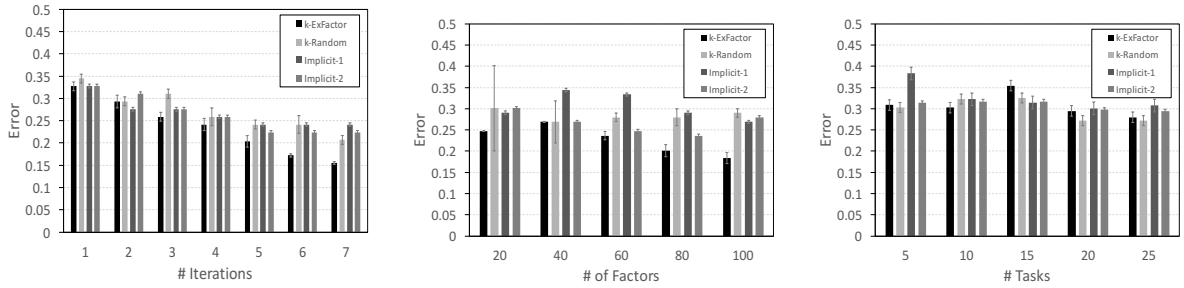
1) Explicit vs. Implicit Feedback: Figure 4 presents a comparative study between explicit, implicit, and no preference elicitation. We compare two explicit solutions with two implicit ones. We vary # iterations, # task factors, and x (# tasks assigned to a worker after which the framework is invoked). Figure4(a) presents the error of each of the four algorithms by varying the number of iterations, where we compare two explicit algorithms ((k-ExFactor and k-Random) with implicit ones Implicit-1 [13] and Implicit-2. Our method k-ExFactor significantly outperforms the other three. After 7 iterations its error drops from 41% to 15% almost 10% lower than other methods. A similar observation holds for k-ExFactor when we vary # task factors (Figure 4(b)), and # tasks (Figure 4(c)). Our proposed solution convincingly and significantly outperforms k-Random and implicit preference computation.

2) Explicit Feedback: Figure 5 presents the error of the two explicit preference elicitation methods as a function of the number of questions k. Notice that the two implicit preference algorithms do not have an input parameter k but for the sake of comparison we have include their results in Figure 5. Overall, our method, k-ExFactor, clearly outperforms k-Random and the other two implicit methods. More importantly, increasing the number of questions does not necessarily yield better results as it is shown in Figure 5. This could be justified by the fact that adding more constraints to the model will result in poor optimization results. That indicates that a small number of questions is good enough to elicit worker preferences and improve the *Worker Model*.

3) Explicit Feedback: Full History vs Partial History: We now present a comparative study between capturing the full history of worker preferences vs just the most recent preferences in the preference aggregation step. The size of history is the total number of explicit feedbacks received from the workers from beginning until a given point in time. As an example, if we ask 4 explicit questions to a worker in each iteration, after the second iteration, her full history size is 8, whereas, her most recent history size is 4; i.e., the recent history represents the number of feedbacks in the current iterations. Figure 6 demonstrates the results by varying # iterations. Clearly, the rate of error reduction for the model with a full history is higher and the error of the model in the end is slightly smaller than the model with just the most recent history. Taking into account the evolution of worker preferences in the whole session is therefore a better option.

4) Bootstrapping: Figure 7 presents the error of the three bootstrapping algorithms. For RandomBoot and OptBoot we set $b = 15$ tasks, whereas, UniformBoot just sets uniform weights to the worker preference vector. We continue to add an additional number of $b = 15$ tasks from the training set and measure MSE. Initially, OptBoot has the best error which signals the effectiveness of our method to pick the best set of tasks that gives us the smallest error. In the preceding iterations, OptBoot converges to a lower error faster than the other two methods. The rate of decrease in error is the same after the fourth iteration which shows that the *Worker Model* is stable and performs well.

5) Worker Model: We profiled three workers randomly from our database and analyzed their models in conjunction with the keywords they have initially chosen. Table II presents the 6 keywords chosen by the workers and the top-2 worker preferences. It is easy to notice that they are highly correlated, which shows that our proposed model successfully captures worker preference.



(a) Error - varying iterations (b) Error - varying task factors (c) Error - varying x

Fig. 4. Comparison between Explicit and Implicit Preferences with Statistical Significance Test (standard error)

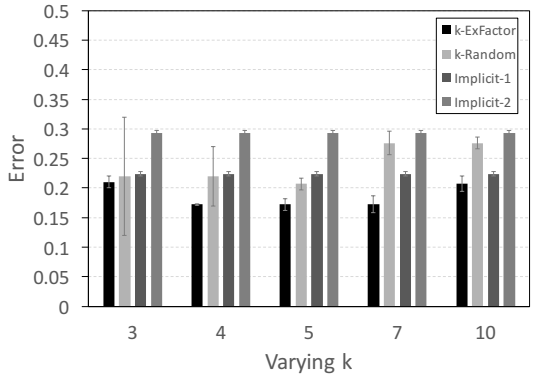


Fig. 5. Error varying k with Statistical Significance Test (standard error)

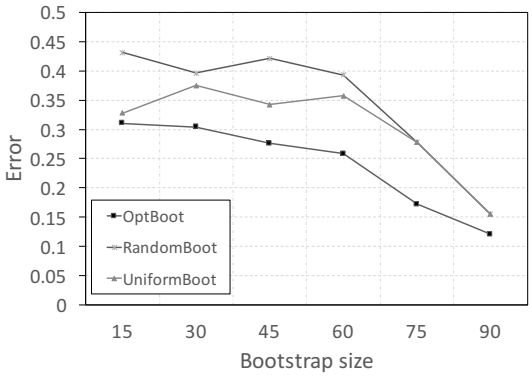


Fig. 7. Evaluation of bootstrapping algorithms

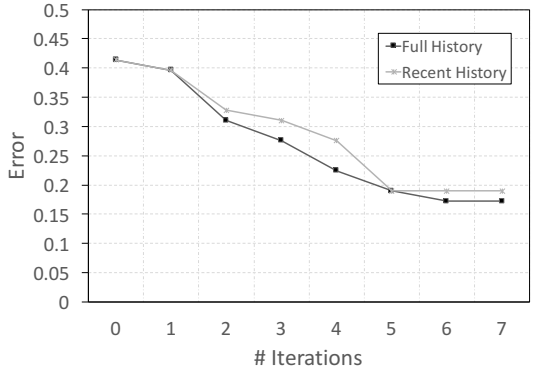


Fig. 6. Comparison between full and recent history

Worker no	Worker Keywords	Top-2 preference
1	dress,google street view, airlines, classification, wheelchair accessibility, scene	dress, scene
2	business, body parts, google street view, health, new year resolution, classification	classification, google street view
3	image, south Asia, disease, animals, text	image, text

TABLE II
WORKER KEYWORDS AND WORKER MODEL

E. Scalability Experiments

We conduct an in-depth scalability study of our solutions and their competitors. Unless otherwise stated, we always report running time in seconds.

Parameter Setting. Our dataset contains $165,168$ tasks and 100 task factors obtained from 58 workers. In this experiment, we vary the following parameters: # tasks, # task factors, k, and the bootstrapping budget b. Unless otherwise stated, all the numbers present the average running time of a single iteration over all the 58 workers. The default values are set as # tasks = $50,000$, # task factors = 50, $k = 3$, and $b = 25$. Unless otherwise stated, all four algorithms are compared with each other. For bootstrapping comparison, only the appropriate three methods are compared.

Figure 8 presents the results. Figure 8(a) presents the running time of the four algorithms with varying number of tasks. Of course, our proposed solution `k-ExFactor` makes a lot more computation to ensure optimization and hence has the highest running time. However, it is easy to notice that with an increasing number of tasks, it scales well and the running time is comparable to the other competing algorithms. A similar observation holds when we vary the number of task factors, as shown in Figure 8(b). `k-ExFactor` scales well and never takes more than 86 seconds. Figure 8(c) represents the running times by varying k, the number of task factors

chosen for preference elicitation. Here only k−ExFactor is compared with k−Random, as the other two algorithms do not rely on explicit preference elicitation. Unsurprisingly, k−Random is faster, but our proposed solution k−ExFactor scales well and has a comparable running time. Finally, in Figure 8(d), we vary the bootstrapping sample size and present the running time of OptBoot. For efficient implementation, we only randomly profile 10% of the branches of the bootstrap tree which makes the algorithm scale linearly. The other two baselines basically do not involve any computation and take negligible time to terminate.

Profiling k−ExFactor**.** We further profile the individual running time of k−ExFactor with the default settings; i.e., # tasks = $50,000$, # task factors= 50, $k = 3$. It takes 28 seconds to train the *Worker Model*, 35.85 seconds to solve **Question Selector** that finds the best k factors, and 29.1 seconds to run **Preference Aggregation** that updates the *Worker Model* with the added constraints. These results demonstrate that the individual components of the framework take comparable time.

V. Related Work

The related work can be classified into three categories: preference elicitation from the crowd, leveraging worker preferences in crowdsourcing processes, and worker models.

Preference Elicitation. In [33], [34], [35], the crowd was solicited to perform max/top-k and clustering operations with the assumption that workers may make errors. These papers study the relationship between the number of comparisons needed and error. Efficient algorithms are proposed with a guarantee to achieve correct results with high probability. A similar problem was addressed in [36] in the case of a skyline evaluation. In that setting, it is assumed that items can only be compared through noisy comparisons provided by the crowd and the goal is to minimize the number of comparisons. A recent work studies the problem of computing the all pair distance graph [37] by relying on noisy human workers. The authors addressed the challenge of how to aggregate those feedback and what additional feedback to solicit from the crowd to improve other estimated distances.

While we also rely on inputs from the crowd, the elicited input represents each worker's preference for different factors (as opposed to completing actual tasks), and is hence not assumed to be noisy or erroneous. However, as worker preferences evolve over time, we propose an iterative approach with the goal of improving task completion overall.

Leveraging Preferences. Worker preferences for task factors are heavily leveraged in all crowdsourcing processes. Very few of these efforts focused on leveraging them in *task completion* [6], [11], [38]. Authors of [39] investigated 13 worker *motivation* factors and found that workers were interested in *skill variety* or *task autonomy* as much as *task reward*. Chandler and Kapelner [6] empirically showed that workers *perceived meaningfulness* of a task improved throughput without degrading quality. Shaw et al. [38] assessed 14 incentives schemes and found that incentives based on *worker-to-worker comparisons* yield better crowd work quality. Hata et al. [10] studied worker *fatigue* and it affects how work quality over extended periods of time. Other efforts focused on gradually increasing pay during task completion to improve worker retention [40]. Lately, adaptive task assignment were studied with a particular focus on maximizing the quality of crowdwork [41], [16], [15], [14], [13] but primary for improved task assignment.

Existing work showed the importance of leveraging implicit worker preferences for task assignment. In contrast, we show explicit elicitation of worker preferences results in a more accurate model that leads to better task completion.

Worker Model. Matrix factorization [42], [43] is used to recommend tasks to workers, where both worker and task features are latent variables in a lower dimensional space. In our work, task factors are explicit and known since they are provided by the crowdsourcing platform and by requesters. Further complex models, such as Multi-Layered Networks as the ones used in deep learning, or Bayesian model are possible but are hard to scale - thus the two core computational problems in our framework that have to excessively use these models become prohibitively complex.

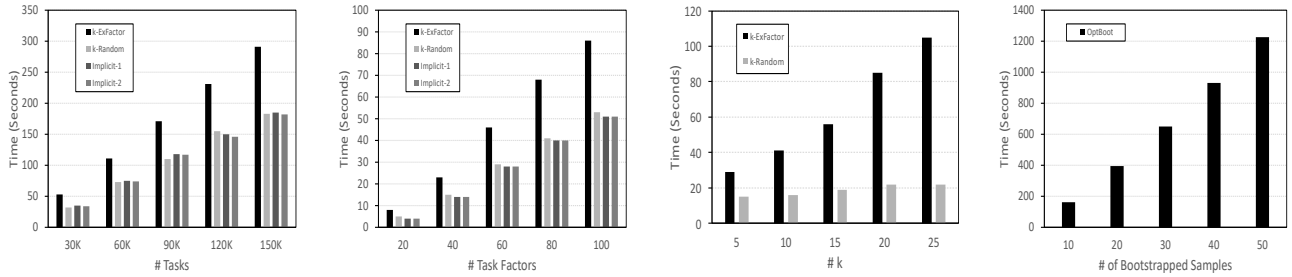
To the best of our knowledge, we present the first principled solution for explicit preference elicitation and rigorously study scalability.

VI. Conclusion and Future Work

We initiate the study of investigating explicit preference elicitation for improved task completion. Our proposed framework leverages a *Worker Model* that is personalized and learns from the past history of the worker and the task characteristics to predict task outcome. Two central problems that are part of our framework are **Question Selector** and **Preference Aggregator**. The former selects the best set of questions to elicit explicit preferences and the latter updates *Worker Model* with the obtained preferences. We present principled solutions and experimentally validate the effectiveness of explicit preference elicitation. Next, we discuss some interesting extensions.

Combining Explicit and Implicit Preference. An interesting open question is how to combine explicit preference elicitation with implicit preference computation. Our current understanding of the problem is, we only invoke explicit preference when certain event triggers it: such as, the error of the *Worker Model* obtained through implicit preference is too high, the worker is not undertaking enough tasks, the implicit preference solution is unable to discriminate worker's preference sufficiently. The challenge of this new problem is to design an optimization function that will guide when to seek explicit preference and when not to.

Handling Multiple Workers. A natural extension of our studied framework is to build and maintain a *Worker Model not per worker, but for a set of workers*. A simple approach would cluster workers based on their preference vectors and aggregate the individual *Worker Models* to build a *Virtual*



(a) varying # tasks (b) varying # task factors (c) varying k (d) varying b

Fig. 8. Scalability study

Worker Model. Such a model is likely to introduce more error (as the model is no longer personalized per worker) but is going to be more efficient to maintain. Such a model could further be used to profile workers in crowdsourcing platforms or improve crowdsourcing processes such as recruitment, completion or assignment.

Worker Models. We present a supervised approach to develop solutions for the *Worker Model*. A natural alternative is to study this problem in an unsupervised setting where the worker history is not available, using techniques such as Self Organizing Maps [44].A further interesting extension is in removing the assumption that there is an one-to-one correspondence between task factors and explicit questions. As long as the correspondence between the task factors and explicit questions is defined, our proposed optimization framework would be adapted.

REFERENCES

[1] B. B. Bederson and A. J. Quinn, "Web workers unite! addressing challenges of online laborers," in *CHI*, 2011, pp. 97–106.

[2] A. Kittur *et al.*, "The future of crowd work," in *CSCW*, 2013.

[3] S. B. Roy *et al.*, "Crowds, not drones: Modeling human factors in interactive crowdsourcing," in *DBCrowd*, 2013.

[4] S. Amer-Yahia and S. B. Roy, "Human factors in crowdsourcing," *Proceedings of the VLDB Endowment*, 2016.

[5] N. Kaufmann, T. Schulze, and D. Veit, "More than fun and money. worker motivation in crowdsourcing-a study on mechanical turk." in *AMCIS*, 2011.

[6] D. Chandler and A. Kapelner, "Breaking monotony with meaning: Motivation in crowdsourcing markets," *CoRR*, vol. abs/1210.0962, 2012.

[7] J. J. Horton and L. B. Chilton, "The labor economics of paid crowdsourcing," in *ACM EC*, 2010, pp. 209–218.

[8] D. B. Martin *et al.*, "Being a turker," in *CSCW*, 2014.

[9] J. Rogstadius *et al.*, "An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets," in *ICWSM*, 2011.

[10] K. Hata *et al.*, "A glimpse far into the future: Understanding long-term crowd worker quality," in *CSCW*, 2017.

[11] P. Dai *et al.*, "And now for something completely different: Improving crowdsourcing workflows with micro-diversions," in *ACM CSCW*, 2015.

[12] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[13] J. Pilourdault *et al.*, "Motivation-aware task assignment in crowdsourcing," in *EDBT*, 2017.

[14] Y. Zheng *et al.*, "QASCA: A quality-aware task assignment system for crowdsourcing applications," in *SIGMOD*, 2015.

[15] C. Ho and J. W. Vaughan, "Online task assignment in crowdsourcing markets," in *AAAI*, 2012.

[16] C. Ho *et al.*, "Adaptive task assignment for crowdsourced classification," in *ICML*, 2013.

[17] S. B. Roy *et al.*, "Task assignment optimization in knowledge-intensive crowdsourcing," *VLDB J.*, 2015.

[18] D. Margaritis and S. Thrun, "Bayesian network induction via local neighborhoods," in *Advances in neural information processing systems*, 2000.

[19] S. Biswas *et al.*, "Combating the cold start user problem in model based collaborative filtering," *CoRR*, vol. abs/1703.00397, 2017.

[20] A. Albert, *Regression and the Moore-Penrose pseudoinverse*. Elsevier, 1972.

[21] C. Dismuke *et al.*, "Ordinary least squares," *Methods and Designs for Outcomes Research*, 2006.

[22] S. E. Fienberg, "An iterative procedure for estimation in contingency tables," *The Annals of Mathematical Statistics*, 1970.

[23] S. E. Fienberg and M. M. Meyer, "Iterative proportional fitting," *Encyclopedia of Statistical Sciences*, 1983.

[24] D. Mottin *et al.*, "A probabilistic optimization framework for the empty-answer problem," *Proceedings of the VLDB Endowment*, 2013.

[25] M. R. Garey, "Optimal binary identification procedures," *SIAM Journal on Applied Mathematics*, vol. 23, no. 2, pp. 173–186, 1972.

[26] F. Pukelsheim, *Optimal design of experiments*. SIAM, 2006.

[27] F. De Hoog and R. Mattheij, "Subset selection for matrices," *Linear Algebra and its Applications*, 2007.

[28] H. Avron and C. Boutsidis, "Faster subset selection for matrices and applications," *SIAM Journal on Matrix Analysis and Applications*, 2013.

[29] R. S. Niculescu *et al.*, "Bayesian network learning with parameter constraints," *Journal of Machine Learning Research*, 2006.

[30] J. L. Mead and R. A. Renaut, "Least squares problems with inequality constraints as quadratic constraints," *Linear Algebra and its Applications*, vol. 432, no. 8, pp. 1936–1949, 2010.

[31] P. B. Stark and R. L. Parker, "Bounded-variable least-squares: an algorithm and applications," *Computational Statistics*, 1995.

[32] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1289–1305, 2003.

[33] S. Guo *et al.*, "So who won?: dynamic max discovery with the crowd," in *SIGMOD*, 2012.

[34] V. Polychronopoulos *et al.*, "Human-powered top-k lists," in *WebDB*, 2013, pp. 25–30.

[35] S. B. Davidson *et al.*, "Top-k and clustering with noisy comparisons," *ACM TODS*, 2014.

[36] B. Groz and T. Milo, "Skyline queries with noisy comparisons," in *PODS*, 2015, pp. 185–198.

[37] H. Rahman *et al.*, "A probabilistic framework for estimating pairwise distances through crowdsourcing." in *EDBT*, 2017.

[38] A. D. Shaw *et al.*, "Designing incentives for inexpert human raters," in *CSCW*, 2011.

[39] N. Kaufmann *et al.*, "More than fun and money. worker motivation in crowdsourcing - A study on mechanical turk," in *AMCIS*, 2011.

[40] Y. Gao *et al.*, "Finish them!: Pricing algorithms for human computation," *PVLDB*, 2014.

[41] J. Fan *et al.*, "icrowd: An adaptive crowdsourcing framework," in *SIGMOD*, 2015.

[42] C. H. Lin *et al.*, "Signals in the silence: Models of implicit feedback in a recommendation system for crowdsourcing." in *AAAI*, 2014.

[43] H. Rahman *et al.*, "Feature based task recommendation in crowdsourcing with implicit observations," *HCOMP*, 2016.

[44] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1, pp. 1–6, 1998.