Optimal Testing for Crowd Workers

Jonathan Bragg
University of Washington
Seattle, WA, USA
jbragg@cs.washington.edu

Mausam
Indian Institute of Technology
New Delhi, India
mausam@cse.iitd.ac.in

Daniel S. Weld
University of Washington
Seattle, WA, USA
weld@cs.washington.edu

ABSTRACT

Requesters on crowdsourcing platforms, such as Amazon Mechanical Turk, routinely insert gold questions to verify that a worker is diligent and is providing high-quality answers. However, there is no clear understanding of when and how many gold questions to insert. Typically, requesters mix a flat 10–30% of gold questions into the task stream of every worker. This static policy is arbitrary and wastes valuable budget — the exact percentage is often chosen with little experimentation, and, more importantly, it does not adapt to individual workers, the current mixture of spamming vs. diligent workers, or the number of tasks workers perform before quitting.

We formulate the problem of balancing between (1) testing workers to determine their accuracy and (2) actually getting work performed as a partially-observable Markov decision process (POMDP) and apply reinforcement learning to dynamically calculate the best policy. Evaluations on both synthetic data and with real Mechanical Turk workers show that our agent learns adaptive testing policies that produce up to 111% more reward than the non-adaptive policies used by most requesters. Furthermore, our method is fully automated, easy to apply, and runs mostly out of the box.

Keywords

Crowdsourcing; reinforcement learning

1. INTRODUCTION

The practice of crowdsourcing, or outsourcing tasks to a group of unknown people (“workers”) in an open call, has become extremely popular — used for solving a wide variety of tasks, from audio transcription to annotation of training data for machine learning. However, ensuring high-quality results for such tasks is challenging, because of the high variability in worker skill. Accordingly, many AI researchers have investigated quality control algorithms. Two prominent techniques have emerged for the problem [24]: (1) periodically inserting gold questions (those with known answers) for each worker, in order to estimate worker reliability, and firing workers who fall below an acceptable accuracy threshold, and (2) employing agreement-based approaches that do not use gold data, instead using expectation maximization (EM) or similar unsupervised methods to jointly estimate worker reliability and task output quality.

These two methods have complementary strengths and weaknesses. While the former is simpler and easy to understand, it puts the onus on the task designer to supply gold questions. Moreover, it is prone to ’bot attacks where, over time, workers identify all gold questions and create ’bots that answer those correctly while answering other questions randomly [18]. On the other hand, the unsupervised approach is technically more sophisticated, but requires significant amounts of data per worker and workers per task before low-quality workers can be reliably identified. Thus, it does not provide a quick filter for firing errant workers.

In practice, while unsupervised techniques have garnered significant research interest (e.g., [26, 25, 7]), the simpler technique of inserting gold questions is the norm in industry. CrowdFlower, a major crowdsourcing platform and consulting company, calls gold questions *“the best way to ensure high quality data from a job”* [2]. Furthermore, many research projects that use crowdsourcing to generate training data eschew EM-based quality control and simply insert gold questions [11, 6, 28, 3].

While insertion of gold questions is popular in practice, to the best of our knowledge, there is no formal model of when and how many gold questions to insert. CrowdFlower’s rule of thumb is to have 10–20% of data as gold, and to insert one gold question per page [2]. Such a policy is arbitrary and may waste valuable budget. Moreover, and more importantly, such a policy is static and does not adapt to individual workers or the current mix of spamming and diligent workers. The percentage of testing questions should only be high if a large percentage of the labor pool are poor quality workers or spammers. Similarly, an adaptive policy that tests less once it is certain that a worker is diligent will likely perform better than a static one.

We formulate the problem of balancing between (1) testing workers to ensure their accuracy and (2) getting work done as a Partially Observable Markov Decision Process (POMDP). Our worker model captures the possibility that worker performance may degrade over time and workers may leave our system after any question. Our model also takes as input a desired accuracy of the final output, and a base testing policy. We apply reinforcement learning over our POMDP to dynamically improve the given base policy with experience. Evaluations on both synthetic data and real data, from Amazon Mechanical Turk, show that our agent is robust to various parameter settings, and typically beats

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

the baseline policies used by most requesters, as well as the input base policy. Furthermore, our method is fully automated, easy to apply, and runs mostly out of the box. We release our software for further use by the research community.[1] Overall, we make the following contributions:

1. We use a POMDP model to formulate the problem of varying the number and placement of gold test questions in order to optimally balance the quality-cost tradeoff for crowdsourcing.

2. We present an adaptive reinforcement learning algorithm that simultaneously estimates worker reliability (by inserting gold questions) and assigns work to produce high-quality output.

3. We comprehensively test our model using simulation experiments and show that our algorithm is robust to variations in desired output accuracy, worker mix, and other parameters.

4. Additional experiments on three live data sets (collected using Amazon Mechanical Turk) show that our reinforcement learning agent produces up to 111% more reward than common policies found in the literature.

2. RELATED WORK

Most practitioners of crowdsourcing use some kind of testing regimen to ensure worker quality. However, the exact policy is usually ad hoc. For example, CrowdFlower has two settings, the number of test questions and total number of questions per page, with the default (best practice) settings as 1 and 5, respectively (i.e., 20% gold). It also allows requesters to specify a minimum acceptable running accuracy for a worker, whose default value is 80%. All these settings can be changed by a requester, but very little advice is offered on how to set their values [1].

Researchers also routinely insert gold questions in their training data collection. As an example, we survey some papers that used crowdsourcing to generate labeled data for the task of relation extraction from text. Zhang *et al.* [28] use a policy similar to CrowdFlower's — 20% gold questions and filtering workers under 80% accuracy. Gormley *et al.* [11] use three test questions every ten questions (30% gold) and filter workers under 85% accuracy. They also boot workers that answer three or more questions in less than three seconds each, and they show that the combination of these methods works better than either alone. Angeli *et al.* [3] insert two gold questions into a set of fifteen (13% gold) questions and filter the workers who have lower than 67% accuracy. They also filter the specific sets of fifteen questions on which a worker fails both test questions.

All these papers use a common strategy of gold question insertion and worker filtering, but choose very different parameter settings for the same task. Moreover, no paper describes a strategy for computing these parameter settings, which are usually chosen by gut instinct and, in rare cases, by limited A/B testing to assess cost-quality variations empirically for various parameter settings. More importantly, these settings are *static* and do not respond to any change in worker mix, or to individual worker characteristics. In this paper, we develop a POMDP-based formalism for mathematically modeling this problem, present an algorithm for automatically learning model parameters, and hence produce a suitable adaptive policy targeted to a given task and labor market.

Our work falls in line with the existing literature on the use of MDPs and POMDPs for crowdsourcing control. These methods have been previously used for deciding whether to hire another worker or to submit a crowdsourced binary answer [7, 14, 19]. POMDPs are also used for controlling more complex workflows such as iterative improvement [8], switching between multiple workflows for the same task [16], and selecting the right worker pool for a question [21]. Similarly, MDPs are used for optimal pricing for meeting a deadline of task completion [10].

In the literature exploring the aggregation-based, cost-quality tradeoff, POMDP decisions are taken based on the needs of the *question* at hand: "Should one take another vote or instead submit one's best estimate of the answer?" Here, the interactions with a worker are quite passive — following a *pull model*, where once a question is sent to the marketplace, any worker may answer it. In practice, a worker usually has a longer relationship with a requester (crowdsourcing control system), since she answers several questions in succession. This sequence of answers can be used for actively allocating test or work questions (*push model*). In our work, we use POMDPs to decide actively when to test a worker and whether to fire them, in case they are not performing to expectation.

Researchers have also explored the temporal dimension of worker behavior, for example Toomim *et al.*'s empirical analysis of worker survival curves [23]. Inspired by this work, our model uses a parameter to model the probability that a worker is going to leave the system at the next step. Worker retention can be improved by bonuses, making tasks more engaging, or other incentives, but we have not yet tried to optimize these factors. Some recent related work has used worker-centric MDPs for optimally placing an intermediate goal [15] or deciding whether to provide a bonus to encourage worker quality or retention [27].

Other researchers have developed more complex time-series models of worker reliability, e.g., using hidden Markov models [9, 27], or autoregressive models [12]. Since our focus is test question insertion, we use a simpler model for temporal reliability, including a parameter to account for the phenomenon that a worker's accuracy may decrease over time,[2] either due to fatigue, boredom, or deceit. Extensions to more complex models is a topic for future work.

Finally, we note that methods that induce lower-quality workers to leave of their own volition are complementary to our approach. Mao et al. [17] found that some workers self-police and stop working on tasks for which they are unsure. Carefully designed instruction and reputation systems may cause workers to recognize when they may be providing low-quality work and stop working.

3. WORKER CONTROL

We propose a controller that automatically decides at each time step whether a specific worker should answer a test

[1] `https://crowdlab.cs.washington.edu/optimal-training-and-testing-for-crowd-workers`

[2] We do not model increase in accuracy, since most fielded crowdsourcing workflows do not provide continued feedback (after an initial tutorial) that would cause workers to improve over time.

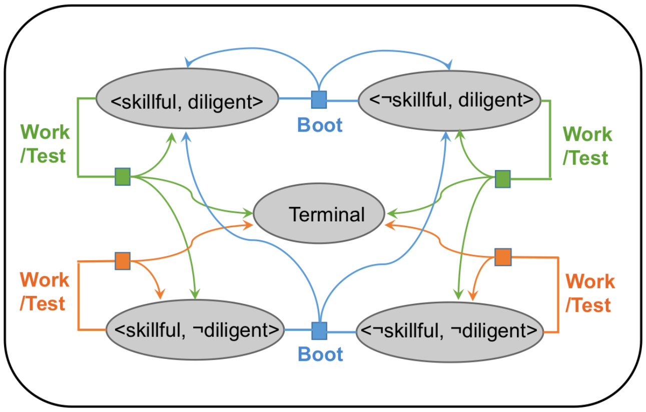


Figure 1: Transition dynamics of the Worker Controller POMDP. Note that the reward and observation model for Work and Test actions are different, but their transition models are the same, as shown.

question, a work question, or whether this worker's performance is not up to the mark, and so this worker should be fired and a replacement worker hired. The actuators that actually perform these actions are part of a web application, which selects new questions from a database of questions and interacts with a crowdsourcing platform to recruit new workers. Our controller should be adaptive, i.e., it should make decisions for each worker, based on their past behavior, instead of following some pre-defined static policy. The global objective of the controller is to get workers to answer as many questions as possible while obtaining a minimum target accuracy supplied by the requester. We first describe key design decisions in formulating this controller.

At a high level, this is clearly a sequential control problem, where the controller gets successive observations about a worker from test questions, and needs to take successive actions based on this history. This problem is *partially* observable because a worker's accuracy isn't known exactly and can change over time. However, the controller can maintain a probability distribution over each worker's accuracy based on its observations. This suggests a controller based on a Partially Observable Markov Decision Process (POMDP).

A POMDP [13] is a popular AI formalism for modeling sequential decision making problems under uncertainty. In a POMDP the state is not fully observed, although observations from information-gathering actions help the agent to maintain a *belief*, which is a probability distribution over the possible world states. A POMDP's objective is to maximize the long-term expected reward minus cost. The solution of a POMDP is a policy that maps a belief into an action.

One natural modeling choice for our worker controller POMDP would be to include the exact accuracy of the worker, a number in $[0, 1]$, in the world state. This accuracy could then be estimated over time based on the test questions. Unfortunately, solving POMDPs over a continuous state space is known to be notoriously hard [20]. Discretizing the accuracy space might be a workable alternative, though it may still be unnecessarily complex since many accuracy bins may result in the same policy.

In response, we adopt a *two-class*[3] worker model, assuming that a worker is randomly drawn from one of the two classes: skillful and unskillful. Both classes of workers have their own (initially unknown) accuracy distributions with mean accuracy of the skillful class higher than that of the unskillful class. This abstraction is beneficial for three reasons. First, for each individual worker the information gathering is limited to ascertaining which class they belong to, and no estimation of individual accuracy or accuracy bin is needed. Second, the worker population mix is abstracted into class means, which can be estimated using an initial exploration phase; these parameters are global, and once estimated apply to the whole population and need not be re-estimated per worker. Third, we can estimate these latent variables in closed form without the need for computationally-expensive approximate inference subroutines within our reinforcement-learning algorithm.

Our model also includes a parameter for workers leaving the task at will. We treat this as a global parameter for the whole worker population. Finally, we also realize that a key reason for repeated testing is that workers can become complacent and their performance may lapse over time. We model this by adding a variable (D) in the state, which, when to set to 1, represents that the worker is diligent (not complacent), and answering to the best of their ability. We now describe the controller in detail.

3.1 The POMDP Model

Defining a POMDP requires specifying (or learning) a state space, action space, transition and observation probabilities, and a reward function. We first describe these aspects of our controller POMDP.

- The state space $\mathcal{S}$ can be factored as $\langle C, D \rangle$, where C is a variable indicating the worker class (skillful or unskillful) and D is a Boolean variable indicating whether the worker is diligent ($D = 1$). The state variable D captures behavior where workers may lapse and start answering without focusing on the problem (with a consequent drop in accuracy); it also models the possibility of spammers faking high-quality work until they think a requester has stopped testing. Additionally, we define a special terminal state denoting the end of process.

- The set of actions $\mathcal{A}$ available to our agent in any non-terminal state consists of *test* (administer a test using a gold question), *work* (ask an unknown question), and *boot* (terminate employment and hire a replacement worker). No action is available in the terminal state.

- The transition function $P(s' \mid s, a)$, depicted pictorially in Figure 1, specifies a probability distribution over new states s' given that the agent takes action a from state s. We assume that workers are diligent to start, so any new worker starts from a state with $D = 1$ and C unknown. The POMDP's belief is a probability distribution over C which is initialized with a prior distribution over classes. This prior is estimated by a single class mix parameter that learns what fraction of workers are a priori skillful and what fraction unskillful.

  When the controller agent takes *test* or *work* actions, the worker may decide to leave the task with (un-

[3] We performed preliminary experiments with larger state spaces and found the gains to be insignificant. Moreover, these more expressive models require significantly more data to learn, which reduces their empirical effectiveness.

known) probability p_{leave} and transition to the terminal state. Moreover, if the worker is in a state $\langle C, D = 1\rangle$, she may transition to a spamming state $\langle C, D = 0\rangle$ with probability $p_{\text{lapse}}(1-p_{\text{leave}})$ or remain in the same state with probability $(1-p_{\text{lapse}})(1-p_{\text{leave}})$. If the worker is already in spamming state $\langle C, D = 0\rangle$, she remains there with probability $(1 - p_{\text{leave}})$. The *boot* action fires this worker and hires a replacement worker with initial belief as described above.

- Observation function: The agent can only directly observe responses to *test* actions. If the worker is in a state with $D = 1$, we assume she answers correctly according to her unknown class accuracy μ_c. Workers with state variable $D = 0$ spam with accuracy 0.5, i.e., they resort to random guessing of a binary-valued question. Additionally, the agent directly observes when a worker leaves.

- Reward function: The agent incurs a cost c for each *test* and *work* action it takes (assuming the worker provides a response). Additionally for *work* actions, our reward model incurs a penalty PN for each incorrect answer and a reward RW for each correct answer it receives. Since *work* actions ask questions with unknown answers, the POMDP needs to compute an expected reward, as follows:

$$R = \mathrm{E}_{Y,X}\left[f(y,x)\right],$$

where

$$f(y,x) = \begin{cases} \text{RW, if } y = x \\ \text{PN, otherwise.} \end{cases}$$

Here Y and X are binary random variables for the latent true answer to the question and the worker response, respectively. This expectation can be computed using joint probabilities $P(Y, X) = P(Y)P(X \mid Y)$, where $P(X = y \mid Y = y)$ is the worker accuracy. This accuracy depends on the current state in the POMDP. We assume that each worker class c has its own latent mean accuracy, μ_c, but that when $D = 0$, workers generate random answers.

So that the requester may specify this reward function in an intuitive way, our experiments focus on the setting where RW = 1 and the value of PN will vary depending on the requester's desired accuracy. If the requester would like to gather answers from workers with accuracy greater than a^*, she may specify PN = $a^*/(a^*-1)$, which will induce positive reward[4] only for workers with accuracy greater than a^*.

As defined, the POMDP, in its pursuit to maximize its long-term expected reward, should learn to test and boot unskillful workers (if their class accuracy is less than a^*) in order to obtain a positive reward. It should also periodically test skillful workers in order to verify that their performance hasn't begun to degrade. The exact parameters of how often and how much to test depend on the unknown mix of the worker classes, target accuracy, model parameters p_{lapse} and p_{leave}, as well as the belief on the current worker at hand.

Our POMDP need only reason about distributions over five world-states (based on different assignments of C and D, and the terminal state), and three actions; thus, it can easily be solved using most modern POMDP algorithms.

3.2 Reinforcement learning

When our system is deployed in a new crowdsourcing environment, it must also learn the POMDP model parameters. This necessitates a reinforcement learning solution with an exploration-exploitation tradeoff.

Five parameters need to be learned: p_{leave}, p_{lapse}, the class mix in the worker population, and the mean accuracies μ_1 and μ_2 for the two classes. Learning these parameters accurately could be data intensive, which means that a typical controller starting from scratch may waste significant budget in learning the model. It may, as part of exploration, boot skillful workers, leading to worker dissatisfaction and subsequent bad requester rating by workers. Most requesters would not be able to afford this.

To alleviate this concern, we allow the requesters to specify a base policy, say, something similar to CrowdFlower's recommended best practice policy. Our reinforcement learner can start with this base policy instead of a random policy. It will gradually transition to following the POMDP policy once it has observed enough workers to estimate parameters. Common base policies that insert a fraction of test questions are desirable because (1) they are widely adopted and easy to implement and (2) the inserted test questions enable our agent to estimate parameters accurately.

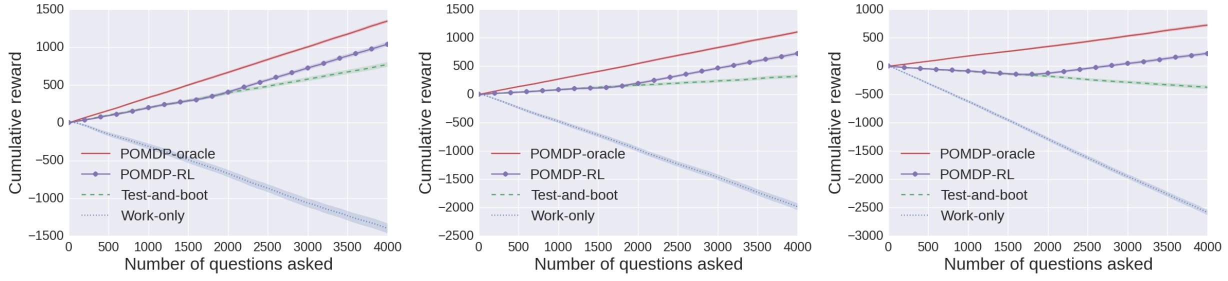
To implement this mixed off/on-policy learning strategy, when the controller agent hires a new worker, it follows the base policy with probability $q(b, B)$ and the POMDP policy with probability $(1 - q(b, B))$, where b is the budget spent so far and B is the total budget. When the agent decides to follow the POMDP policy, it reestimates model parameters and replans prior to hiring a new worker. We define our particular choice of function q in the next section.

To estimate parameters, we treat the sequential data as an input-output hidden Markov model [4] and use the Baum-Welch (EM) algorithm initialized with parameters from the previous episode and one random restart. We use default uninformed priors of Beta(1, 1) on parameters, but use a prior of Beta(5, 2) for class accuracy and a prior of Beta(2, 20) for p_{lapse}. Beta(5, 2) encodes the fact that the accuracy should be at least 0.5 for Boolean questions, and Beta(2, 20) is an optimistic prior that workers don't tend to become spammers very frequently.[5] We assume that each class has its own accuracy but use parameter tying to estimate values for p_{leave} and p_{lapse} that are shared between classes.

We also experimented with an estimation process that is identical, except that it does not estimate the class accuracies. Instead, it defines two worker accuracy bins, one on $[a^*, 1.0]$ for skillful workers and one on $[0.5, a^*)$ for unskillful workers and fixes $\mu_1 = (a^* + 1.0)/2$, $\mu_2 = (0.5 + a^*)/2$. While these accuracy means may differ from the maximum likelihood estimates, they still allow the agent to distinguish between workers expected to give work of sufficient quality (positive reward) and workers the agent should boot. This variant has two primary benefits. First, estimating only the class ratios requires much less data than estimating both the class ratios and the accuracy means. Second, it is pos-

[4] This formula comes from setting the expected reward (aRW + $(1 - a)$PN) to 0 when RW = 1.

[5] In preliminary experiments, we found an optimistic prior was important to enable our system to distinguish between skillful workers who may drop in accuracy and unskillful workers who had low accuracy from the start.



(a) 80:20 class ratio.

(b) 50:50 class ratio.

(c) 20:80 class ratio.

Figure 2: Our controller agent achieves significantly higher rewards than baseline policies under a variety of ratios of skillful to unskillful workers. As can be seen by the slopes of the curves, after learning, the RL policies start performing about as well as the optimal POMDP policy, which was given true parameters. These plots (and all subsequent reward plots) show mean performance over 200 runs with shaded 95% confidence intervals.

sible that the maximum likelihood estimates for accuracy means will produce means $\mu_1, \mu_2 < a^*$. Fixing an accuracy bin above a^* ensures that the agent will be able to identify high-quality workers even if the mean accuracy μ_1 for skillful workers is below the desired accuracy a^*.

4. EXPERIMENTS

In experiments, we first test our agent with simulated workers under a variety of settings; the goal is to assess the robustness of our method to differing task settings and its ability to adapt to attacks by different populations of spammers. Later, we demonstrate performance on three NLP datasets using real workers from Amazon Mechanical Turk.

We implement a reinforcement learning agent (POMDP-RL) that improves upon a base policy of inserting 20% gold questions and firing workers if their accuracy is less than the desired accuracy (Test-and-boot). We compare the RL agent with this base policy on its own, as well as a baseline policy that inserts no gold questions (Work-only). The Test-and-boot base policy inserts a batch of 4 test questions in every set of 20 questions. In simulation experiments, we are also able to compare performance with an agent that has access to the true model parameters (POMDP-oracle).

POMDP-RL learns the class ratio, class mean accuracies (μ_1, μ_2), p_{leave}, and p_{lapse} using the priors specified in the previous section. We use the Test-and-boot policy in each experiment as our base exploration policy. We used the ZMDP POMDP package,[6] which implements the Focused Real Time Dynamic Programming (FRTDP) algorithm for solving POMDPs [22]. We ran the solver with default configuration settings, maximum solve time of 1 minute,[7] and discount factor of 0.99. In order to speed up experiment runtimes, the agents recomputed the POMDP policy at most once every 10 workers and at most 10 times total.

4.1 Agent Robustness

In our simulation experiments, we considered two classes of workers, one of high accuracy ($\mu_1 = 0.9$) and one of low accuracy ($\mu_2 = 0.6$). Worker accuracy in each class varies according to a truncated normal distribution parameterized by the class mean and $\sigma = 0.1$ and bounded on $[0.5, 1]$. Unless otherwise noted, we consider a 50:50 mixture of these two classes, where workers degrade with probability $p_{\text{lapse}} = 0.01$. Only the simulator and POMDP-oracle had access to the true worker parameters; POMDP-RL estimated parameters based only on observations.

Our experiments are from the point of view of a requester who wants to ensure data quality above $a^* = 0.75$. Our penalty function gives us $\text{PN} = -3$, which provides positive rewards for data above this accuracy. The default budget size is $B = 4000$ questions. In this set of experiments, we used the sigmoid exploration function $q(b, B) = 1 - 1/(1 + \exp(40(b/B - 0.4))$, which approximately changes from 1 to 0 in the range of 25% to 50% of the budget.

RQ1. Is our agent robust to the ratio of skillful to unskillful workers in the labor pool?

In these experiments, we varied the mixture of skillful workers to unskillful workers, while keeping the worker distribution properties fixed. Figure 2 shows that as this ratio decreases from 80:20 to 20:80, policies achieve lower reward, as we would expect with just a few skillful workers. The relative benefit of the POMDP agents, however, remains significant. In all settings, POMDP-RL learns a policy that gains roughly as much reward per question as the POMDP-oracle. This can be seen by the fact that in the last half of the budget, POMDP-oracle and POMDP-RL have similar reward slopes.

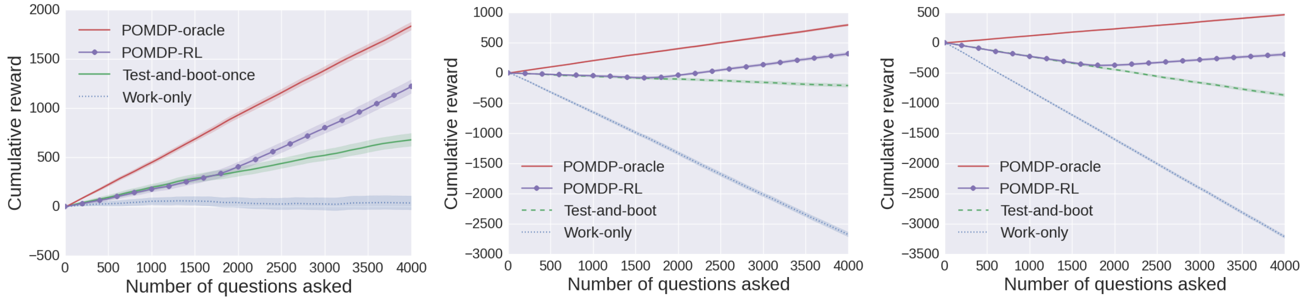
Both POMDP-oracle and POMDP-RL earn significantly higher total cumulative reward after spending the budget than the best baseline policy (Test-and-boot), according to two-tailed T tests ($p \ll 0.001$ for all; $t = 32.5$, $t = 13.2$ for 80:20; $t = 30.9$, $t = 13.8$ for 50:50; $t = 55.3$, $t = 27.7$ for 20:80). The differences tend to become more significant for settings with fewer skillful workers, since careful testing becomes more important there.

RQ2. Is our agent robust to the fraction of workers who stop being diligent (eventually answering randomly)?

In order to test how our agent responds to workers who may begin to spam with probabilities other than 0.01, we experimented with $p_{\text{lapse}} \in \{0, 0.2, 0.4\}$, as shown in Figure 3. Note that for $p_{\text{lapse}} = 0$ (Figure 3a), POMDP-RL uses the base policy of Test-and-boot-once, which tests 4 times *only at the start* (and fires the worker if they answer more than one question incorrectly). For this setting, we

[6] `https://github.com/trey0/zmdp`

[7] Solver ran on 2.3 GHz Intel Xeon E7-8850-v2 processors.

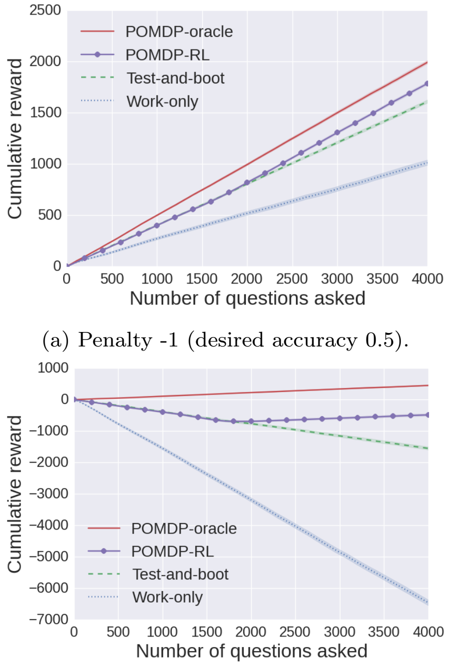


(a) $p_{\text{lapse}} = 0$.

(b) $p_{\text{lapse}} = 0.02$.

(c) $p_{\text{lapse}} = 0.04$.

Figure 3: Our RL controller is also robust to the number of deceptive spam workers, beating the baseline exploration policy, and eventually matching the performance of the POMDP-oracle model (which has knowledge of the true parameters). When $p_{\text{lapse}} = 0.04$, 4% of previously diligent workers start answering randomly after each question.



(a) Penalty -1 (desired accuracy 0.5).

(b) Penalty -6 (desired accuracy 0.86).

Figure 4: Our agents perform robustly when supplied different utility functions and produce higher relative rewards for utility functions corresponding to higher desired accuracies.

don't need a policy that tests at intervals; it is optimal to do all testing at the beginning if worker performance does not drop. This set of experiments used synthetic data from the default equally-sized worker classes. In all experiments, the POMDP-oracle and POMDP-RL agents again obtain significantly higher reward than the baselines.

RQ3. Is the RL controller robust to changes in the requester's utility function (i.e., desired accuracy)?

The previous experiments used PN $= -3$ (desired minimum accuracy of 0.75). In this set of experiments, we varied the utility function by setting PN $= -1$ and PN $= -6$, corresponding to desired accuracies of 0.5 and approximately 0.86, respectively. As shown in Figure 4, the relative gains of adaptive testing increase as the desired accuracy (and corresponding magnitude of penalty) increase. Note that for PN $= -6$ (Figure 4b), only the POMDP-oracle agent has final positive cumulative reward. After having spent some budget, POMDP-RL also learns a policy that improves cumulative reward (neither baseline policy is able to improve reward). In both experiments, POMDP-oracle and POMDP-RL produce significantly higher reward than the best baseline policy (Test-and-boot).

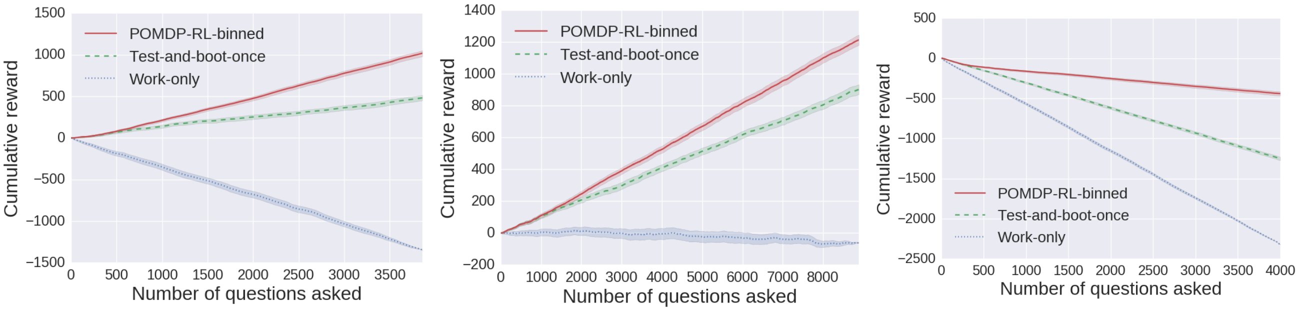
RQ4. How good is the learned RL policy compared to the POMDP with known parameters?

Figures 2 through 4 show that POMDP-RL is able to learn policies with reward slopes similar to those of POMDP-oracle. This research question evaluates the policy learned by POMDP-RL isolated in a pure exploitation phase on the last 10% of an experiment with $B = 2000$ and the default worker configuration (50:50 class ratio, $p_{\text{lapse}} = 0.01$). Figure 6 shows that the reward obtained by POMDP-RL is on par with POMDP-oracle (there is no statistically significant difference in cumulative reward at the end of the budget), suggesting that the combination of base exploration policy and exploration-exploitation tradeoff is learning effectively.

Examination of worker traces shows that POMDP-RL and POMDP-oracle take similar actions. Both have an initial testing phase to establish worker quality, and then periodically insert a test question in every batch of approximately 6 questions. If a worker answers the single test question incorrectly, the agent will administer additional test questions adaptively.

4.2 Testing on Real Workers & Tasks

To answer the question of how well our agent performs on real datasets, we conducted experiments using three datasets gathered on Amazon Mechanical Turk. The worker completes an Entity Linking task in which a sentence and a mention (a portion of the sentence) is shown, and the worker is asked to match the mention to the correct Wikipedia entry. Two of our datasets, *LinWiki* and *LinTag*, were supplied by Lin *et al.* [16], who had Mechanical Turk workers answer questions using two different styles of questions, which they called WikiFlow and TagFlow. These datasets consist of 110 questions. 135 workers supplied 3,857 answers in LinWiki, and 149 workers supplied 3,999 answers in LinTag.



(a) LinWiki dataset.

(b) Rajpal dataset.

(c) LinTag dataset.

Figure 5: Our RL controller (POMDP-RL) significantly outperforms baseline policies on all three datasets with real workers. Figures show mean performance over 200 runs (with shaded 95% confidence intervals).

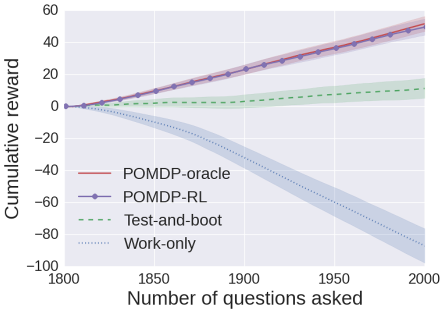


Figure 6: Zooming into the agent's behavior during the last 10% of the budget (i.e., after most learning), there is no statistically significant different between our RL controller and the agent that is given the true model of workers by an oracle. This figure uses the same worker distribution as Figure 2b (50:50 class mixture and $p_{\text{lapse}} = 0.01$).

Our third dataset, *Rajpal*, consists of 150 questions from the same task. Rajpal *et al.* [21] recruited workers with Mechanical Turk Masters qualifications (35 workers, 3,015 answers) as well as without those qualifications (108 workers and 5,870 answers). We combined these two sets of responses into a single dataset (143 workers, 8,885 answers) for this experiment.

When performing experiments, we set the budget B equal to the total number of answers. The Work-only policy used every answer from every worker exactly once upon completing this budget. Since the other policies may boot workers and therefore require more workers than exist in the original dataset while consuming the budget, we recycle workers as needed for those policies. Our simulator randomizes the order of workers and the order of worker answers because the datasets do not contain metadata (e.g., timestamps) that would let us determine the order in which answers were received.

Since answers from a worker arrive in random order, expected worker quality should not change over time; we fix $p_{\text{lapse}} = 0$. Thus, these experiments use the Test-and-boot-once variant of the base policy, which performs one block of testing at the start only (and boots if the accuracy in that block is below the desired accuracy).

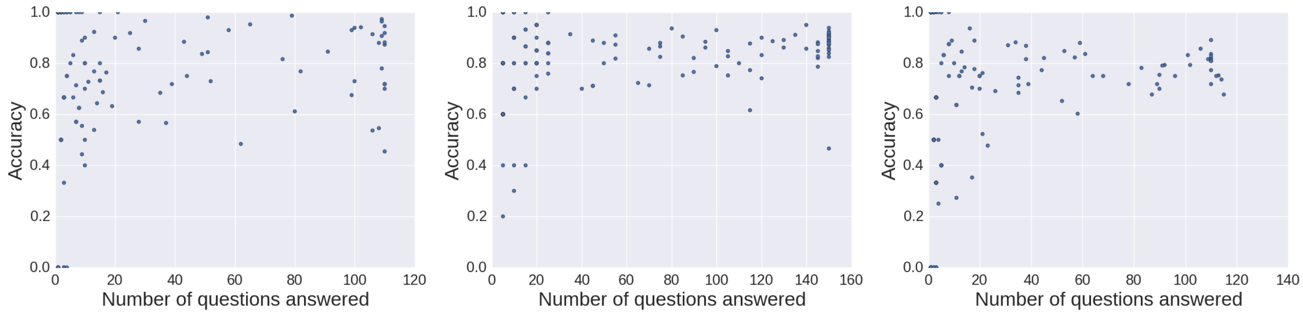
We set the desired accuracy to 0.85, a value close to the upper bound of what is reasonable, since only a small fraction of answers in the LinTag dataset come from workers above this accuracy. To give the base policy enough granularity when determining worker accuracy, the base policy tests 7 times (and boots if more than one answer is incorrect). We fix class mean accuracies μ_1, μ_2 for POMDP-RL using the binning method to ensure that our method can identify workers above the desired accuracy even if the maximum likelihood class means are below that value. Since the agent does not need to estimate these parameters, we explore only for the first 20 workers.

Note that workers cannot distinguish a gold question from a non-gold question; they have exactly the same effect on a worker. Since we know the correct answers, we can (post-hoc) treat any question as gold, and evaluate any possible policy.

Performance on these three datasets is summarized in Figure 5 and Table 1. Note that we are only able to run the POMDP-RL agent (not POMDP-oracle), since we do not know the true worker parameters. After consuming the budget, POMDP-RL generated 111% more cumulative reward than the best baseline for the LinWiki dataset (1018.2 vs. 481.7) and 35% more reward for the Rajpal dataset (1214.5 vs. 902.5). On the LinTag dataset, all methods produced negative reward, but POMDP-RL produced only 35% as much negative reward as the best baseline. The best baseline in each case is Test-and-boot-once. Running a two-tailed T test on these rewards determines that the differences are significant for all datasets ($t = 21.5$, 13.6, and 46.7 for the LinWiki, Rajpal, and LinTag datasets, respectively; $p \ll 0.001$).

Inspecting the distribution of worker accuracies and number of questions answered by each worker helps to explain these results. As shown in Figure 7, some low-accuracy workers answer a large number of questions in the LinWiki dataset; POMDP-RL produces large gains by adaptively testing and filtering these workers. In contrast, the lower quality workers in the Rajpal and LinTag datasets tend to leave on their own accord after a small number of questions, reducing the possible benefit of testing them in order to fire poor performers. The LinTag dataset has a small fraction of workers above the desired accuracy, and none of the methods were able to produce a dataset with the desired accuracy.

Examining the action traces and overall statistics on the number of times the POMDP-RL agent took test, work, and boot actions gives some insight into how the agent is able to improve on the static Test-and-boot baseline. As the



(a) LinWiki dataset.

(b) Rajpal dataset.

(c) LinTag dataset.

Figure 7: Scatter plots of worker accuracy vs. number of questions answered show that low accuracy workers leave on their own accord after a small number of questions in the Rajpal and LinTag datasets — hence all forms of worker testing have limited benefit in these scenarios. Adaptive testing has the most potential benefit in the LinWiki dataset.

Dataset	Policy	Reward	Labels	Acc
LinWiki	POMDP-RL	__*1018.2__	2656	**90.8**
LinWiki	Test-and-boot-once	481.7	2895	87.5
LinWiki	Work-only	-1346.6	3857	79.8
Rajpal	POMDP-RL	__*1214.5__	6867	**87.7**
Rajpal	Test-and-boot-once	902.5	7629	86.8
Rajpal	Work-only	-62.0	8885	84.9
LinTag	POMDP-RL	__*-439.5__	1253	**79.6**
LinTag	Test-and-boot-once	-1251.3	2842	78.4
LinTag	Work-only	-2320.2	3999	76.3

Table 1: Our reinforcement learning agent captures higher rewards than the baseline policies on all three live datasets. Asterisks indicate significantly higher rewards. Our agent produces datasets of higher accuracy (Acc) at the expense of gathering fewer labels.

POMDP-RL agent transitions from the base (static) exploration policy to the learned adaptive policy, the mean number of test actions per worker stays constant or decreases slightly, but the mean number of work actions per worker decreases and the agent boots more frequently. This suggests that the agent becomes more conservative with its work actions, thus increasing accuracy.

5. CONCLUSIONS

In order to distinguish between high-quality and error-prone workers, requesters on crowdsourcing platforms, such as Amazon Mechanical Turk, routinely reserve 10–30% of their workload for "gold" test questions, whose answers are known, and then dismiss workers who fail a disproportionate percentage. This type of static policy is arbitrary and wastes valuable budget. The exact percentage of test questions and dismissal rate is often chosen with little experimentation, and, more importantly, it does not adapt to individual workers, the current mixture of skillful vs. unskillful workers, nor the number of tasks workers perform before quitting. Intuitively, the percentage of test questions should be high if a large percentage of the workers are spammers. Furthermore, once one is very certain that a worker is diligent, one can likely decrease the testing frequency.

To develop a principled solution to the problem of balancing between (1) testing workers to determine their accuracy, and (2) actually getting work performed by good workers, we formulate it as a partially-observable Markov decision process (POMDP). Our worker model captures the possibility that worker performance may degrade over time (whether due to fatigue, boredom, or deceit) and workers may leave our system after any question. Our model also takes as input a minimum desired accuracy of the final output, and a base testing policy. We apply reinforcement learning over the POMDP to dynamically improve the given base policy with experience. We comprehensively test our model using simulation experiments and show that our algorithm is robust to variations in desired output accuracy, worker mix, and other parameters. Additional experiments on three live data sets (collected using Amazon Mechanical Turk) show that our agent performs up to 111% better than common policies found in the literature. Importantly, our software is fully automated, easy to apply, runs mostly out of the box, and is made available for further use by the research community.

We note that our method has several limitations. For example, gold test questions are only applicable for crowd work with objective answers. Neither our testing approach nor methods based on expectation maximization will work for subjective questions or descriptive tasks where correct answers can be highly variable. However, we think our POMDP model could be extended to schedule *separate* validation jobs from an independent set of crowd workers. Another exciting direction for future work would be to extend our model to handle instruction in addition to testing [5]. Such a model would choose between (1) teaching a worker to improve proficiency, (2) testing to measure worker accuracy, and (3) getting work done, all before the worker quits the system.

ACKNOWLEDGEMENTS

We thank Chris Lin and the anonymous reviewers for helpful comments. This work was supported by NSF grant IIS-1420667, ONR grant N00014-15-1-2774, the WRF/Cable Professorship, and a gift from Google to Dan Weld and gifts from Google and Bloomberg to Mausam.

REFERENCES

[1] Crowdflower Inc. job launch checklist. `https://success.crowdflower.com/hc/en-us/articles/202703195-Job-Launch-Checklist`.

[2] Crowdflower Inc. test question best practices. `https://success.crowdflower.com/hc/en-us/articles/202703105-Test-Question-Best-Practices`.

[3] G. Angeli, J. Tibshirani, J. Wu, and C. D. Manning. Combining distant and partial supervision for relation extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

[4] Y. Bengio and P. Frasconi. An input output HMM architecture. In *Advances in Neural Information Processing Systems (NIPS)*, 1995.

[5] J. Bragg, Mausam, and D. S. Weld. Learning on the job: Optimal instruction for crowdsourcing. In *ICML Workshop on Crowdsourcing and Machine Learning*, 2015.

[6] C. Callison-Burch and M. Dredze. Creating speech and language data with amazon's mechanical turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010.

[7] P. Dai, C. H. Lin, Mausam, and D. S. Weld. POMDP-based control of workflows for crowdsourcing. *Artif. Intell.*, 202:52–85, 2013.

[8] P. Dai, Mausam, and D. S. Weld. Decision-theoretic control of crowd-sourced workflows. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, 2010.

[9] P. Donmez, J. G. Carbonell, and J. G. Schneider. A probabilistic framework to learn from multiple annotators with time-varying accuracy. In *Proceedings of the SIAM International Conference on Data Mining (SDM 2010)*, 2010.

[10] Y. Gao and A. G. Parameswaran. Finish them!: Pricing algorithms for human computation. *Proceedings of the VLDB Endowment (PVLDB)*, 7(14):1965–1976, 2014.

[11] M. R. Gormley, A. Gerber, M. Harper, and M. Dredze. Non-expert correction of automatically generated relation annotations. In *NAACL Workshop on Creating Speech and Language Data With Amazon's Mechanical Turk*, 2010.

[12] H. J. Jung, Y. Park, and M. Lease. Predicting next label quality: A time-series model of crowdwork. In *Proceedings of the Second AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2014)*, 2014.

[13] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.

[14] E. Kamar, S. Hacker, and E. Horvitz. Combining human and machine intelligence in large-scale crowdsourcing. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, 2012.

[15] A. Kobren, C. H. Tan, P. G. Ipeirotis, and E. Gabrilovich. Getting more for less: Optimized crowdsourcing with dynamic tasks and goals. In *Proceedings of the 24th International Conference on World Wide Web (WWW 2015)*, 2015.

[16] C. H. Lin, Mausam, and D. S. Weld. Dynamically switching between synergistic workflows for crowdsourcing. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2012)*, 2012.

[17] A. Mao, Y. Chen, E. Horvitz, M. E. Schwamb, C. J. Lintott, and A. M. Smith. Volunteering Versus Work for Pay: Incentives and Tradeoffs in Crowdsourcing. In *Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2013)*, 2013.

[18] D. Oleson, A. Sorokin, G. P. Laughlin, V. Hester, J. Le, and L. Biewald. Programmatic gold: Targeted and scalable quality assurance in crowdsourcing. In *Human Computation Workshop*, page 11, 2011.

[19] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: Algorithms for filtering data with humans. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2012)*, 2012.

[20] J. M. Porta, N. Vlassis, M. T. Spaan, and P. Poupart. Point-based value iteration for continuous POMDPs. *J. Mach. Learn. Res.*, 7:2329–2367, Dec. 2006.

[21] S. Rajpal, K. Goel, and Mausam. POMDP-based worker pool selection for crowdsourcing. In *ICML Workshop on Crowdsourcing and Machine Learning*, 2015.

[22] T. Smith and R. Simmons. Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*, 2006.

[23] M. Toomim, T. Kriplean, C. Pörtner, and J. A. Landay. Utility of human-computer interactions: toward a science of preference measurement. In *Proceedings of the International Conference on Human Factors in Computing Systems (CHI 2011)*, 2011.

[24] D. Weld, Mausam, C. Lin, and J. Bragg. Artificial intelligence and collective intelligence. In T. Malone and M. Bernstein, editors, *The Collective Intelligence Handbook*. MIT Press, 2015.

[25] P. Welinder, S. Branson, S. Belongie, and P. Perona. The multidimensional wisdom of crowds. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[26] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. R. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[27] M. Yin and Y. Chen. Bonus or not? learn to reward in crowdsourcing. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 2015.

[28] C. Zhang, F. Niu, C. Ré, and J. W. Shavlik. Big data versus the crowd: Looking for relationships in all the right places. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*, 2012.