Artificial Intelligence for Artificial Artificial Intelligence

Peng Dai Mausam Daniel S. Weld
Dept of Computer Science and Engineering
University of Washington
Seattle, WA-98195
{daipeng,mausam,weld}@cs.washington.edu

Abstract

Crowdsourcing platforms such as Amazon Mechanical Turk have become popular for a wide variety of human intelligence tasks; however, quality control continues to be a significant challenge. Recently, we propose TURKONTROL, a theoretical model based on POMDPs to optimize iterative, crowd-sourced workflows. However, they neither describe how to *learn* the model parameters, nor show its effectiveness in a *real* crowd-sourced setting. Learning is challenging due to the scale of the model and noisy data: there are hundreds of thousands of workers with high-variance abilities.

This paper presents an end-to-end system that first learns TURKONTROL's POMDP parameters from real Mechanical Turk data, and then applies the model to dynamically optimize live tasks. We validate the model and use it to control a successive-improvement process on Mechanical Turk. By modeling worker accuracy and voting patterns, our system produces significantly superior artifacts compared to those generated through nonadaptive workflows using the same amount of money.

Introduction

Within just a few years of their introduction, crowdsourcing marketplaces, such as Amazon Mechanical Turk[1], have become an integral component in the arsenal of an online application designer. These have spawned several new companies such as CrowdFlower, CastingWords, and led to creative applications, *e.g.*, helping blind people shop or localize in a new environment [1]. The availability of hundreds of thousands of workers allows a steady stream of output. Unfortunately, the workers also come with hugely varied skill sets and motivation levels. Thus, quality control of the worker output continues to be a serious challenge.

To work around the variability in worker accuracy, people design workflows, flowcharts connecting sequences of primitive steps, where a step may be performed by multiple workers, thus improving overall quality. For example, CastingWords employs a proprietary workflow for the task of audio transcription. Recently, Little *et al.* [8] achieve impressive results using a workflow of *iterative improvement* for several tasks such as handwriting recognition and writing a text description for an image. In this workflow (see

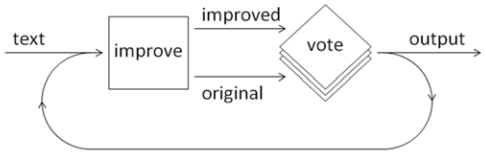

Figure 1: Flowchart for the iterative text improvement task, reprinted from [8].

Figure 1), the work by one worker goes through several improvement iterations; each iteration comprising an improvement phase (where previous work is improved by a worker) and an evaluation phase (where voters choosing between the improved work and the previous work through ballots – the 'vote' step in Figure 1). In essence, these workflows embody a novel type of collaboration between workers; one that generates high quality work.

These workflows typically have several decision points, *e.g.*, for iterative improvement one must decide how many evaluation votes to obtain and whether to repeat the improvement loop. From an AI perspective, this offers a new, exciting and impactful application area for intelligent control. Recently we [3] propose a POMDP formulation of the workflow control problem and show that TURKONTROL, the decision-theoretic controller, obtains higher quality outputs in a simulated environment. However, our previous work is primarily theoretical and provides no methods to learn the several distributions in the POMDP model. Nor does it provide strong evidence that the approach actually works in a real crowdsourced environment.

In this paper, we implement an end-to-end system that first learns parameters for TURKONTROL using real data from Mechanical Turk. This learning is challenging because of a large number of parameters and sparse and noisy training data. To make our problem feasible we choose specific parametric distributions to reduce the parameter space, and learn the improvement and ballot parameters independently. We validate the learned parameters in a simple voting task and observe that the model needs only half the votes compared to the commonly used majority baseline. This suggests the effectiveness of the model and our parameters.

We then employ TURKONTROL with our learned parameters to control the iterative improvement workflow for the image description task on Mechanical Turk. TURKONTROL can exploit knowledge of individual worker accuracies; however, it does not need such information and incremen-

[1] Amazon uses the tagline "Artificial Artificial Intelligence"

tally updates its model of each worker as she completes each job. We compare our AI-controlled, dynamic workflows with a nonadaptive workflow that spends the same amount of money. The results demonstrate that our system obtains an 11% improvement in the average quality of image descriptions. Our results are statistically significant with $p < 0.01$. More interestingly, to achieve the same quality, a nonadaptive workflow spends 28.7% more money, as quality improvement is not linear in the amount of cost.

Background

Iterative Improvement Workflow. Little *et al.* [8] design the iterative improvement workflow to get high-quality results from the crowd. As shown in Figure 1, the work created by the first worker goes through several improvement iterations; each iteration comprising an improvement and a ballot phase. In the improvement phase, an *improvement job*, solicits α', an improvement of the current artifact α (*e.g.*, the current image description). In the ballot phase, several workers respond to a *ballot job*, in which they vote on the better of the two artifacts (the current one and its improvement). Based on majority vote, the better one is chosen as the current artifact for next iteration. This process repeats until the total cost allocated to the particular task is exhausted.

POMDP. A *partially-observable Markov decision process* (POMDP) [7] is a widely-used formulation to represent sequential decision problems under partial information. An agent, the decision maker, tracks the world state and faces the decision task of picking an action. Performing the action transitions the world to a new state. The transitions between states are probabilistic and Markovian, *i.e.*, the next state only depends on the current state and action. The state information is unknown to the agent, but she can infer a *belief*, the probability distribution of possible states, from observing the world.

Controlling A Crowd-Sourced Workflow. There are various decision points in executing an iterative improvement process, such as which artifact to select, when to start a new improvement iteration, when to terminate the job, *etc.* We recently [3] introduce TURKONTROL, a POMDP based agent that controls the workflow, *i.e.*, makes these decisions automatically. The world state includes the quality of the current artifact, $q \in [0, 1]$, and q' of the improved artifact; true q and q' are hidden and the controller can only track a belief about them. Intuitively, the extreme value of 0 (or 1) represents the idealized condition that all (or no) diligent workers will be able to improve the artifact. We use Q and Q' to denote the random variables that generate q and q'.

Different workers may have different skills in improving an artifact. A conditional distribution function, $f_{Q'_x|q}$, expresses the probability density of the quality of a new artifact when an artifact of quality q is improved by worker x. The worker-independent distribution function, $f_{Q'|q}$, acts as a prior in cases where a previously unseen worker is encountered. The ballot job compares the two artifacts; intuitively, if the two artifacts have qualities close to each other then the ballot job is harder. We define intrinsic difficulty of a ballot job as $d = 1 - |q - q'|^M$, where M is a trained constant.

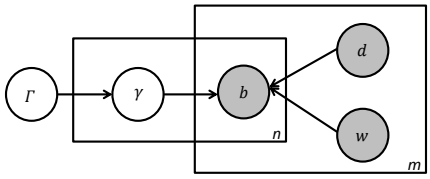

Figure 2: A plate model of ballot jobs; b represents the ballot outcome; γ, a worker's individual error parameter; d, the difficulty of the job and w, truth value of the job. Γ is the prior on workers' errors. Shaded nodes represent observed variables.

Given the difficulty d, ballots of two workers are conditionally independent to each other. We assume that the accuracy of worker x follows $a(d, \gamma_x) = \frac{1}{2}[1 + (1 - d)^{\gamma_x}]$, where γ_x is x's error parameter; higher γ_x signifies that x makes more errors.

Previously we discuss several POMDP algorithms to control the workflow including limited look-ahead, UCT, *etc.* While simulation results suggest benefits of our model, we do not discuss any approaches to learn these complex distributions, nor implement our techniques on a real platform to prove that the simplifying assumptions, formulae, and simulated gains hold in practice.

Model Learning

In order to estimate TURKONTROL's POMDP model, one must learn two probabilistic transition functions. The first function is the probability of a worker x answering a ballot question correctly, which is controlled by the error parameter γ_x of the worker. The second function estimates the quality of an improvement result, the new artifact returned by a worker.

Learning the Ballot Model

Figure 2 presents our generative model of ballot jobs; shaded variables are observed. We seek to learn the error parameters $\vec{\gamma}$ where γ_x is parameter for the xth worker and use the mean $\bar{\gamma}$ as an estimate for future, unseen workers. To generate training data for our task we select m pairs of artifacts and post n copies of a ballot job which asks the workers to choose between these pairs. We use $b_{i,x}$ to denote x^{th} worker's ballot on the ith question. Let $w_i = true(false)$ if the first artifact of the ith pair is (not) better than the second, and d_i denote the difficulty of answering such a question.

We assume the error parameters are generated by a random variable Γ. The ballot answer of each worker directly depends on her error parameter, as well as the difficulty of the job, d, and its real truth value, w. For our learning problem, we collect w and d for the m ballot questions from the consensus of three human experts and treat these values as observed. In our experiments we assume a uniform prior of Γ, though our model can incorporate more informed priors[2]. Our aim is to estimate γ_x parameters – we use the standard maximum likelihood approach. We use vector notation with

[2]We also tried priors that penalize extreme values but that did not help in our experiments.

$b_{i,x}$ denoting x^{th} worker's ballot on the ith question and $\vec{\vec{b}}$ denotes all ballots.

$$P(\vec{\gamma}|\vec{\vec{b}},\vec{w},\vec{d}) \propto P(\vec{\gamma})P(\vec{\vec{b}}|\vec{\gamma},\vec{w},\vec{d}) \quad (1)$$

Under the uniform prior of Γ and conditional independence of different workers given difficulty and truth value of the task, Equation 1 can be simplified to

$$\begin{aligned} P(\vec{\gamma}|\vec{\vec{b}},\vec{w},\vec{d}) &\propto P(\vec{\vec{b}}|\vec{\gamma},\vec{w},\vec{d}) \\ &= \Pi_{i=1}^{m}\Pi_{x=1}^{n}P(b_{i,x}|\gamma_x,d_i,w_i). \quad (2) \end{aligned}$$

Taking the log, the Maximum likelihood problem is:

$$\begin{aligned} \text{Constants}: &\quad d_1,\ldots,d_m,w_1,\ldots,w_m,b_{11},\ldots,b_{m,n} \\ \text{Variables}: &\quad \gamma_1,\ldots,\gamma_n \\ \text{Maximize}: &\quad \textstyle\sum_{i=1}^{m}\sum_{x=1}^{n} log[P(b_{i,x}|\gamma_x,d_i,w_i))] \\ \text{Subject to}: &\quad \emptyset \end{aligned}$$

Experiments on Ballot Model. We evaluate the effectiveness of our learning procedure on the image description task. We select 20 pairs of images ($m = 20$) and collect sets of ballots from 50 workers. We detect spammers and drop them ($n = 45$). We spend \$4.50 to collect this data. We solve the optimization problem using the NLopt package.[3]

Once the error parameters are learned they can be evaluated in a five-fold cross-validation experiment as follows: take 4/5th of the images and learn error parameters over them; use these parameters to estimate the true ballot answer ($\tilde{w}_i$) for the images in the fifth fold. Our cross-validation experiment obtains an accuracy of 80.01%, which is barely different from a simple majority baseline (with 80% accuracy). Indeed, we doublecheck that the four ballots frequently missed by the models are those in which the mass opinion differs from our expert labels.

We also compare the confidence, degree of belief in the correctness of an answer, for the two approaches. For the majority vote, we calculate the confidence by taking the ratio of the votes with the correct answer and the total number of votes. For our model, we use the average posterior probability of the correct answer. The average confidence values of using our ballot model is much higher than the majority vote (82.2% against 63.6%). This shows that even though the two approaches achieve the same accuracy on all 45 votes, the ballot model has superior belief in its answer.

While the confidence values are different the ballot model seems to offer no distinct advantage over the simple majority baseline. In hindsight, this is not surprising, since we are using a large number of workers. In other work researchers have shown that a simple average of a large number of non-experts often beats even the expert opinion [10].

However, one will rarely have the resources to doublecheck each question by 45 voters, so we study this further by varying the number of available voters. For each image pair, we randomly sample 50,000 sets of 3-11 ballots and compute the average accuracies of the two approaches. Figure 3 shows that our model consistently outperforms the majority vote baseline. With just 11 votes, it is able to achieve an accuracy of 79.3%, which is very close to that using all 45 votes. Also, the ballot model with only 5 votes achieves similar accuracy as a majority vote with 11. This shows the value of the ballot model – it significantly reduces the amount of votes needed for the same desired accuracy.

[3] *http://ab-initio.mit.edu/wiki/index.php/NLopt*

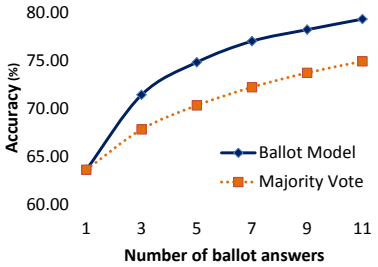

Figure 3: Accuracies of using our ballot model and majority vote on random voting sets with different size, averaged over 10,000 random sample sets for each size. Our ballot model achieves significantly higher accuracy than the majority vote ($p < 0.01$).

Estimating Artifact Quality

In order to learn the effect of a worker trying to improve an artifact (next section), we need labeled training data, and this means determining the quality of an arbitrary artifact. While it may seem a somewhat amorphous and subjective term, Following [3], we define *quality* of an artifact to be the probability that an average diligent worker fails to improve it. Thus, an artifact with quality 0.5 is just as likely to be hurt by an improvement attempt as actually enhanced. Since quality is a partially-observable statistical measure, we consider three ways to approximate it: simulating the definition, direct expert estimation, and averaged worker estimation.

Our first technique simply *simulates the definition*. We ask k workers to improve an artifact α and as before use multiple ballots, say l, to judge each improvement. We define quality of α to be 1 minus the fraction of workers that are able to improve it. Unfortunately, this method requires $k + kl$ jobs in order to estimate the quality of a single artifact; thus, it is both slow and expensive in practice. As an alternative, *direct expert estimation* is less complex. We teach a statistically-sophisticated computer scientist the definition of quality and ask her to estimate the quality to the nearest decile.[4] Our final method, *averaged worker estimation*, is similar, but averages the judgments from several Mechanical Turk workers via *scoring jobs*. These scoring jobs provide a definition of quality along with a few example; the workers are then asked to score several more artifacts.

Experimental Observations. We collect data on 10 images from the Web and use Mechanical Turk to generate multiple descriptions for each. We then select one description for each image, carefully ensuring that the chosen descriptions span a wide range of detail and language fluency. We also modified a description to obtain one that, we felt, was very hard to improve, thereby accounting for the high quality region. When simulating the definition, we average over

[4] The consistency of this type of subjective rating has been carefully evaluated in the literature; see *e.g.* [2].

$k = 22$ workers.[5] We use a single expert for direct expert estimation and an average of 10 worker scores for averaged worker estimation.

Our hope, following [10], was that averaged worker estimation, definitely the cheapest method, would prove comparable to expert estimates and especially to the simulated definition. Indeed, we find that all three methods produce similar results. They agree on the two best and worst artifacts, and on average both expert and worker estimates are within 0.1 of the score produced by simulating the definition. We conclude that averaged worker estimation is equally effective and additionally easier and more economical (1 cent per scoring job); so we adopt this method to assess qualities in subsequent experiments.

Learning the Improvement Model

Finally, we describe our approach for learning a model for the improvement phase. Our objective is to estimate the quality q' of a new artifact, α', when worker x improves artifact α of quality q. We represent this using a conditional probability density function $f_{Q'_x|q}$. Moreover, we also learn a prior distribution, $f_{Q'|q}$, to model work by a previously unseen worker.

There are two main challenges in learning this model: first, these functions are over a two-dimensional continuous space, and second, the training data is scant and noisy. To alleviate the difficulties, we break the task into two learning steps: (1) learn a mean value for quality using regression, and (2) fit a conditional density function given the mean. We make the second learning task tractable by choosing parametric representations for these functions. Our full solution follows the following steps:

1. Generate an improvement job that contains u original artifacts $\alpha_1, \ldots, \alpha_u$.
2. Crowd-source v workers to improve each artifact to generate uv new artifacts.
3. Estimate the qualities q_i and $q'_{i,x}$ for all artifacts in the set (see previous section). q_i is the quality of α_i and $q'_{i,x}$ denotes the quality of the new artifact produced by worker x. These act as our training data.
4. Learn a worker-dependent distribution $f_{Q'_x|q}$ for every participating worker x.
5. Learn a worker-independent distribution $f_{Q'|q}$ to act as a prior on unseen workers.

We now describe the last two steps in detail: the learning algorithms. We first estimate the mean of worker x's improvement distribution, denoted by $\mu_{Q'_x}(q)$.

We assume that $\mu_{Q'_x}$ is a linear function of the quality of the original artifact, *i.e.*, the mean quality of the new artifact linearly increases with the quality of the original one.[6] By introducing $\mu_{Q'_x}$, we separate the variance in a worker's ability in improving all artifacts of the same quality from the variance in our training data, which is due to her starting out from artifacts of different qualities. To learn this we perform linear regression on the training data $(q_i, q'_{i,x})$. This yields $q'_x = a_x q + b_x$ as the line of regression with standard error e_x, which we truncate for values outside $[0, 1]$.

To model a worker's variance when improving artifacts with the same quality, we consider three parametric representations for $f_{Q'_x|q}$: Triangular, Beta, and Truncated Normal. While clearly making an approximation, restricting attention to these distributions significantly reduces the parameter space and makes our learning problem tractable. Note that we assume the mean, $\hat{\mu}_{Q'_x}(q)$, of each of these distributions is given by the line of regression, $a_x q + b_x$. We consider each distribution in turn.

Triangular: The triangular-shaped probability density function has two fixed vertices $(0, 0)$ and $(1, 0)$. We set the third vertex to $\hat{\mu}_{Q'_x}(q)$, yielding the following density function:

$$f_{Q'_x|q}(q'_x) = \begin{cases} \frac{2q'_x}{\hat{\mu}_{Q'_x}(q)} & \text{if } q'_x < \hat{\mu}_{Q'_x}(q) \\ \frac{2(1-q'_x)}{1-\hat{\mu}_{Q'_x}(q)} & \text{if } q'_x \geq \hat{\mu}_{Q'_x}(q). \end{cases} \tag{3}$$

Beta: We wish the Beta distribution's mean to be $\hat{\mu}_{Q'_x}$ and its standard deviation to be proportional to e_x. Therefore, we train a constant, c_1, using gradient descent that maximizes the log-likelihood of observing the training data for worker x.[7] This results in $f_{Q'_x|q} = Beta(\frac{c_1}{e_x} \times \hat{\mu}_{Q'_x}(q), \frac{c_1}{e_x} \times (1 - \hat{\mu}_{Q'_x}(q)))$. The error e_x appears in the denominator because the two parameters for the Beta distribution are approximately inversely related to its standard deviation.

Truncated Normal: As before we set the mean to $\hat{\mu}_{Q'_x}$ and the standard deviation to be $c_2 \times e_x$ where c_2 is a constant, trained to maximize the log likelihood of the training data. This yields $f_{Q'_x|q} = Truncated\ Normal(\hat{\mu}_{Q'_x}(q), c_2^2 e_x^2)$ where the truncated interval is $[0, 1]$.

We use similar approaches to learn the worker-independent model $f_{Q'|q}$, except that training data is of the form $(q_i, \bar{q}'_i)$ where $\bar{q}'_i$ is the *average* improved quality for i^{th} artifact, *i.e.*, the mean of $q'_{i,x}$ (over all workers). The standard deviation of this set is $\sigma_{Q'_i|q_i}$. As before, we start with linear regression, $q' = aq + b$. The Triangular distribution is defined exactly as before. For the other two distributions, we have their standard deviations depend on the conditional standard deviations, $\sigma_{Q'_i|q_i}$. We assume that the conditional standard deviation $\sigma_{Q'|q}$ is quadratic in q, therefore an unknown conditional standard deviation given any quality $q \in [0, 1]$ can be inferred from existing ones $\sigma_{Q'_1|q_1}, \ldots, \sigma_{Q'_v|q_v}$ using quadratic regression. As before, we use gradient descent to train variables c_3 and c_4 for Beta and Truncated Normal respectively.

[5] We collected 24 sets of improvements, but two workers improved less than 3 artifacts, so they were tagged as spammers and dropped from analysis.

[6] While this is obviously an approximation, we find it is surprisingly close; $R^2 = 0.82$ for the worker-independent model.

[7] We use Newton's method with 1000 random restarts. Initial values are chosen uniformly from the real interval $(0, 100.0)$.

Experimental Observations. We seek to determine which of the three distributions best models the data, and we employ leave-one-out cross validation. We set the number of original artifacts and number of workers to be ten each. It costs a total of \$16.50 for this data collection. The algorithm iteratively trains on nine training examples, *e.g.* $\{(q_i, \bar{q}'_i)\}$ for the worker-independent case, and measures the probability density of observing the tenth. We score a model by summing the ten log probability densities.

Our results show that Beta distribution with $c_1 = 3.76$ is the best conditional distribution for worker-dependent models. For the worker-independent model, Truncated Normal with $c_4 = 1.00$ performs the best. We suspect this is the case because most workers have average performance and Truncated Normal has a thinner tail than the Beta. In all cases, the Triangular distribution performs worst. This is probably because Triangular assumes a linear probability density, whereas, in reality, workers tend to provide reasonably consistent results, which translates to higher probabilities around the conditional mean. We use these best performing distributions in all subsequent experiments.

TurKontrol on Mechanical Turk

Having learned the POMDP parameters, our final evaluation assesses the benefits of the dynamic workflow controlled by TURKONTROL versus a nonadaptive workflow (as originally used in TurKit [8]) under similar settings, specifically using the same monetary consumption. We aim at answering the following questions: (1) Is there a significant quality difference between artifacts produced using TURKONTROL and TurKit? (2) What are the qualitative differences between the two workflows?

As before, we evaluate on the image description task, in particular, we use 40 fresh pictures from the Web and employ iterative improvement to generate descriptions for these. For each picture, we restrict a worker to take part in at most one iteration in each setting (*i.e.*, nonadaptive or dynamic). We set the user interfaces to be identical for both settings and randomize the order in which the two conditions are presented to workers in order to eliminate human learning effects. Altogether there are 655 participating workers, of which 57 take part in both settings.

We devise automated rules to detect spammers. We reject an improvement job if the new artifact is identical to the original. We reject ballot and scoring jobs if they are returned so quickly that the worker could not have made a reasonable judgment.

Note that our system does not need to learn a model for a new worker before assigning them jobs; instead, it uses the worker-independent parameters $\bar{\gamma}$ and $f_{Q'|q}$ as a prior. These parameters get incrementally updated as TURKONTROL obtains more information about their accuracy.

Recall that TURKONTROL performs decision-theoretic control based on a user-defined utility function. We use $U(q) = \$25q$ for our experiments. We set the cost of an improvement job to be 5 cents and a ballot job to be 1 cent. We use the limited-lookahead algorithm from [3] for the controller, since that performed the best in their simulation. Under these parameters, TURKONTROL-workflows run an average of 6.25 iterations with an average of 2.32 ballots per iteration, costing about 46 cents per image description on average.

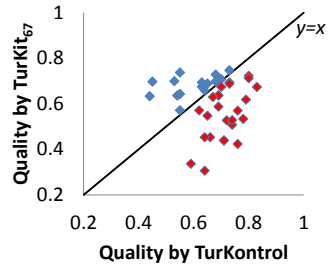

Figure 4: Average qualities of 40 descriptions generated by TURKONTROL and by TurKit respectively, under the same monetary consumption. TURKONTROL generates statistically-significant higher-quality descriptions than TurKit.

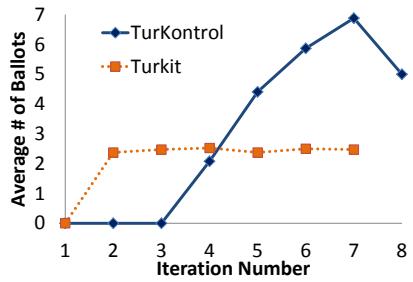

Figure 5: Average number of ballots for the nonadaptive and dynamic workflows. TURKONTROL makes and intelligent use of ballots.

We use TurKit's original fixed policy for ballots, which requests a third ballot if the first two voters disagree. We compute the number of iterations for TurKit so that the total money spent matches TURKONTROL's. Since this number comes to be 6.47 we compare against three cases: TurKit_6 with 6 iterations, TurKit_7 with 7 iterations and TurKit_{67} a weighted average of the two that equalizes monetary consumption.

For each final description we create a scoring job in which multiple workers score the descriptions. Figure 4 compares the artifact qualities generated by TURKONTROL and by TurKit_{67} for the 40 images. We note that most points are below the $y = x$ line, indicating that the dynamic workflow produces superior descriptions. Furthermore, the quality produced by TURKONTROL is greater on average than TurKit's, and the difference is statistically significant: $p < 0.01$ for TurKit_6, $p < 0.01$ for TurKit_{67} and $p < 0.05$ for TurKit_7, using the student's t-test.

Using our parameters, TURKONTROL generates some of the highest-quality descriptions with an average quality of 0.67. TurKit_{67}'s average quality is 0.60; furthermore, it generates the two worst descriptions with qualities below 0.3. Finally, the standard deviation for TURKONTROL is much lower (0.09) than TurKit's (0.12). These results demonstrate overall superior performance of decision-theoretic control on live, crowd-sourced workflows.

While the 11% average quality promotion brought by TURKONTROLis statistically significant, some wonder if it

is *material*. To better illustrate the importance of quality, we include another experiment. We run the nonadaptive, TurKit policy for additional improvement iterations, until it produces artifacts with an average quality equal to that produced by TURKONTROL. Fixing the quality threshold, the TurKit policy has to run an average of 8.76 improvements, compared to the 6.25 improvement iterations used by TURKONTROL. As a result the nonadaptive policy spends 28.7% more money than TURKONTROL to achieve the same quality results. Note that final artifact quality is neither linear in the number of iterations nor total cost. Intuitively, it is much easier to improve an artifact when its quality is low than when it is high.

We also qualitatively study TURKONTROL's behavior compared to TurKit's and find an interesting difference in the use of ballots. Figure 5 plots the average number of ballots per iteration number. Since TurKit's ballot policy is fixed, it always uses about 2.45 ballots per iteration. TURKONTROL, on the other hand, uses ballots much more intelligently. In the first two improvement iterations TURKONTROL does not bother with ballots because it expects that most workers will improve the artifact. As iterations increase, TURKONTROL increases its use of ballot jobs, because the artifacts are harder to improve in later iterations, and hence TURKONTROL needs more information before deciding which artifact to promote to the next iteration. The eighth iteration is an interesting exception; at this point improvements have become so rare that if even the first voter rates the new artifact as a loser, then TURKONTROL often believes the verdict.

Besides using ballots intelligently we believe that TURKONTROL adds two other kinds of reasoning. First, six of the seven pictures that TURKONTROL finished in 5 iterations have higher qualities than TurKit's. This suggests that its quality tracking is working well. Perhaps due to the agreement among various voters, TURKONTROL is able to infer that a description already has quality high enough to warrant termination. Secondly, TURKONTROL has the ability to track individual workers, and this also affects its posterior calculations. For example, in one instance TURKONTROL decided to trust the first vote because that worker had superior accuracy as reflected in a low error parameter. We expect that for repetitive tasks this will be an enormously valuable ability, since TURKONTROL will be able to construct more informed worker models and take superior decisions.

We present one image description example in Figure 6. It is interesting to note that both processes managed to find out the origin of the image. However, the TURKONTROL version is consistently better in language, factuality and level of detail. In retrospect, we find the nonadaptive workflow probably made a wrong ballot decision in the sixth iteration, where a decision was critical yet only three voters were consulted. TURKONTROL on the other hand, reached a decision after 6 unanimous votes at the same stage.

Related Work

Since crowd-sourcing is a recent development, few have tried using AI or machine learning to control such platforms.

Figure 6: An image description example.
It took TURKONTROL 6 improvement HITs and 14 ballot HITs to reach the final version: "*This is Gene Hackman, in a scene from his film "The Conversation," in which he plays a man paid to secretly record people's private conversations. He is squatting in a bathroom gazing at tape recorder which he has concealed in a blue toolbox that is now placed on a hotel or motel commode (see paper strip on toilet seat). He is on the left side of the image in a gray jacket while the commode is on the right side of the picture. His fingertips rest on the lid of the commode. He is wearing a black court and a white shirt. He has put on glasses also.*"
It took the nonadaptive workflow 6 improvement HITs and 13 ballot HITs to reach a version: "*A thought about repairing : Image shows a person named Gene Hackman is thinking about how to repair the toilet of a hotel room. He has opened his tool box which contains plier, screw diver, wires etc. He looks seriously in his tool box & thinking which tool he willuse. WearING a grey coat he sits in front of the toilet seat resting gently on the toilet seat*".

Ipeirotis *et al.* [6] observe that workers tend to have bias on multiple-choice, annotation tasks. They learn a *confusion matrix* to model the error distribution of individual workers. However, their model assumes workers' errors are completely independent, whereas, our model handles situations where workers make correlated errors due to the intrinsic difficulty of the task.

Huang *et al.* [5] look at the problem of designing a task under budget and time constraints. They illustrate their approach on an image-tagging task. By wisely setting variables, such as reward per task and the number of labels requested per image, they increase the number of useful tags acquired. Donmez *et al.* [4] observe that workers' accuracy often changes over time (*e.g.*, due to fatigue, mood, task similarity, *etc.*). Based on their model, one can predict which worker is likely to be most accurate for task at any time. As these approaches are orthogonal to ours, we would like to integrate the methods in the future.

Shahaf and Horvitz [9] develop an HTN-planner style decomposition algorithm to find a coalition of workers, each with different skill sets, to solve a task. Some members of the coalition may be machines and others humans; different skills may command different prices. In contrast to our work, Shahaf and Horvitz do not consider methods for learning models of their workers.

Similar to our approach of deciding the truth value of a ballot question from ballot results, Whitehill *et al.* [12] study the problem of integrating image annotations of multiple workers. Their proposed model also integrates a worker

model and a difficulty measure. They use unsupervised EM for learning, whereas we use gold standard annotations. Their results align well with our findings that integrating labels through an intelligent model achieves higher accuracy than the majority vote. Welinder *et al.* [11] extend Whitehill *et al.*'s work by modeling tasks and workers as multidimensional entities. Their model is more general so can capture other useful factors, such as personal bias, error tolerance level *etc*. However, both of these use unsupervised learning, and the learned models can be erroneous if there are hard tasks in the mix (*e.g.*the four hard training tasks in our ballot model learning). Also, they do not use a decision-theoretic framework to balance cost-quality tradeoffs.

Conclusions

Complex, crowd-sourcing workflows are regularly employed to produce high-quality output. Our work conclusively demonstrates the benefits of AI, specifically decision-theoretic techniques, in controlling such workflows. We present an efficient and cheap mechanism to learn the parameters of a POMDP model that enables this control. Next, we validate the parameters independently and show that our learned model of worker accuracy significantly outperforms the popular majority-vote baseline when resources are constrained. Thirdly, we demonstrate the effectiveness of the decision-theoretic techniques, using an end-to-end system on a live crowd-sourcing platform for the task of writing image descriptions. With our learned models guiding the controller, the dynamic workflows are vastly superior than nonadaptive workflows utilizing the same amount of money. Our results are statistically significant. Finally, we investigate the qualitative behavior of our agent, illustrating interesting characteristics of iterative workflows on Mechanical Turk.

We believe that decision-theoretic control, when powered by our learned models, has the potential to impact the thousands of requesters who use crowd-sourcing today, making their processes significantly more efficient. We plan to release a toolkit implementing our techniques for wide use.

Acknowledgments

This work was supported by the WRF / TJ Cable Professorship , Office of Naval Research grant N00014-06-1-0147, and National Science Foundation grants IIS 1016713 and IIS 1016465. We thank Mimi Fung for technical help in setting up the web server for TURKONTROL. We thank Greg Little and Rob Miller for providing the TurKit code and answering many technical questions. Comments from Andrey Kolobov, Raphael Hoffman, Andrey Kolobov, and Micheal Toomim significantly improved the paper.

References

[1] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samuel White, and Tom Yeh. Vizwiz: nearly real-time answers to visual questions. In *UIST*, pages 333–342, 2010.

[2] Dan Cosley, Shyong K. Lam, Istvan Albert, Joseph A. Konstan, and John Riedl. Is seeing believing?: how recommender system interfaces affect users' opinions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 585–592, 2003.

[3] Peng Dai, Mausam, and Daniel S. Weld. Decision-theoretic control of crowd-sourced workflows. In *AAAI*, 2010.

[4] Pinar Donmez, Jaime G. Carbonell, and Jeff Schneider. A probabilistic framework to learn from multiple annotators with time-varying accuracy. In *SIAM International Conference on Data Mining (SDM)*, pages 826–837, 2010.

[5] Eric Huang, Haoqi Zhang, David C. Parkes, Krzysztof Z. Gajos, and Yiling Chen. Toward automatic task design: A progress report. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 77–85, 2010.

[6] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 64–67, 2010.

[7] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.

[8] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkit: tools for iterative tasks on mechanical turk. In *KDD Workshop on Human Computation*, pages 29–30, 2009.

[9] Dafna Shahaf and Eric Horvitz. Generlized markets for human and machine computation. In *AAAI*, 2010.

[10] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks. In *EMNLP*, pages 254–263, 2008.

[11] Peter Welinder, Steve Branson, Serge Belongie, and Pietro Perona. The multidimensional wisdom of crowds. In *In Proc. of NIPS*, pages 2424–2432, 2010.

[12] Jacob Whitehill, Paul Ruvolo, Tingfan Wu, Jacob Bergsma, and Javier Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *In Proc. of NIPS*, pages 2035–2043, 2009.