CrowdDQS: Dynamic Question Selection in Crowdsourcing Systems

Asif R. Khan
Stanford University
Stanford, CA, USA
asifk@stanford.edu

Hector Garcia-Molina
Stanford University
Stanford, CA, USA
hector@cs.stanford.edu

ABSTRACT

In this paper, we present CrowdDQS, a system that uses the most recent set of crowdsourced voting evidence to dynamically issue questions to workers on Amazon Mechanical Turk (AMT). CrowdDQS posts all questions to AMT in a single batch, but delays the decision of the exact question to issue a worker until the last moment, concentrating votes on uncertain questions to maximize accuracy. Unlike previous works, CrowdDQS also (1) optionally can also decide when it is more beneficial to issue gold standard questions with known answers than to solicit new votes (both can help us estimate worker accuracy, but gold standard questions provide a less noisy estimate of worker accuracy at the expense of not obtaining new votes), (2) estimates worker accuracies in real-time even with limited evidence (with or without gold standard questions), and (3) infers the distribution of worker skill levels to actively block poor workers. We deploy our system live on AMT to over 1000 crowdworkers, and find that using our techniques, CrowdDQS can accurately answer questions using up to **6x fewer votes** than standard approaches. We also find there are many non-obvious practical challenges involved in deploying such a system seamlessly to crowdworkers, and discuss techniques to overcome these challenges.

Keywords

Crowdsourcing; Task Assignment; Quality Control; Error Estimation

1. INTRODUCTION

In recent years, the development of crowdsourcing marketplaces such as Amazon Mechanical Turk (AMT) has allowed for computer scientists to scalably leverage the power of human insights into their computational workflows. Recent applications of the crowd include labeling training data for machine learning, improving entity resolution workflows, data filtering, image understanding, sentiment analysis, and many others [8,17,18,28,35,37,39,40,41]. In industry, there are reports of several organizations spending millions of dollars annually on crowdsourcing budgets [24].

The standard setting for these crowdsourced data processing tasks is often as follows: requesters have a pool of questions they want the crowd to answer, where each question has a predefined number of answer choices. To ensure high data quality, requesters add redundancy by allocating a fixed number of votes for each question (e.g. a requester may stipulate that each question be voted on by 5 distinct workers). The questions and number of requested votes are sent together as a *batch* to the crowdsourcing system, which then publishes all tasks simultaneously for workers to complete. Finally, the requesters aggregate these votes (e.g. with majority voting) to produce a final set of answers. Existing crowdsourcing marketplaces like AMT are optimized and designed around this workflow.

This standard workflow has several inherent weaknesses because it is *batch-based* and *static* — i.e. the allocation of needed votes per task is decided in advance, without an easy ability to react to worker responses. Crowdworkers have unknown and variable skill levels — some workers produce lower quality data than others, and some workers are spammers, i.e. they try to answer as many questions as possible without any consideration for accuracy. Due to their prolificness, spammers and low-quality workers can often have a disproportionate effect on the quality of the aggregated votes, even if they represent a small fraction of the worker pool. Another problem in the standard workflow is that allocating a fixed number of votes per question is inherently wasteful. Questions which have high worker agreement end up receiving unnecessary extra votes, while more difficult questions with high worker disagreement receive few votes.

There are several existing techniques to help mitigate some of these problems after a batch has been completed. Many techniques [11,13,22,25,31,34,39,42,46], such as expectation-maximization (EM) [7], have been proposed which estimate worker accuracy and then discount the votes of poor quality workers during aggregation. However, these algorithms are usually designed to be run after the batch of tasks has been completed, and do not actively prevent poor quality workers from voting on questions.

Another common approach to estimate worker accuracy is to use *gold standard* questions (GSQs), where requesters have some number of questions in their tasks with known ground truth [27,38]. However, how do we choose when to issue gold standard questions versus solicit more votes? For example, if we pay a worker for 10 different votes, and 3 votes are devoted to gold standard questions, 30% of our

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

budget does not go toward doing real work. This trade-off is not well-addressed in the literature.

The focus of our work is instead to consider a *dynamic* approach to question selection, where tasks are assigned to workers on-the-fly *within* a batch, based on the latest set of previous responses. A dynamic approach has the potential to detect and evict poor quality workers, and also to focus votes on the questions which cause the most uncertainty.

In this paper, we present **CrowdDQS** (Crowd Dynamic Question Selection), a system implemented on top of Amazon Mechanical Turk which we find can need up to **6 times fewer** votes in practice than standard workflows, while still maintaining the same accuracy. CrowdDQS is designed to achieve the following:

- We estimate worker accuracies after each response within a batch and use the *most recent* evidence to do so. This allows the system to automatically learn about the distribution of workers on a specific group of tasks, and to use this information when blocking workers.
- When a worker asks to work on a question in our batch, we dynamically choose which question they should vote on using the latest set of voting evidence that has been collected.
- We automatically issue gold standard questions when we determine it is more important to learn about a worker's accuracy than it is to receive new votes from the worker.
- We block low-quality workers and spammers from answering more questions as soon as they are detected, preventing them from contributing more poor-quality votes to our batch.
- We deploy our system to workers on Amazon Mechanical Turk seamlessly, maintaining the same interface that workers are accustomed to.
- We use incremental updating algorithms to make all of these decisions in real-time. Even though we decide which question to serve a worker just before it is presented to them, the worker does not notice any perceived delay.

The idea of dynamically issuing tasks to workers has been discussed before in various contexts in the literature [4, 10, 15, 23, 41, 46]. However, due to the practical difficulty of seamlessly deploying a dynamic question selection algorithm on existing crowdsourcing marketplaces, most work has been confined to experiments on synthetic data only [4, 12, 15, 16, 32, 33]. There is a small handful of very recent work (e.g. QASCA [46] and iCrowd [10]) where researchers have directly tested dynamic task selection algorithms on Amazon Mechanical Turk. We discuss previous work in this area in more detail in Section 13. At a high level, CrowdDQS makes the following contributions relative to these recent systems:

- Our system dynamically chooses when to inject gold standard questions versus solicit new votes. We propose a novel mechanism for quantifying the trade-off between learning more about the specific accuracy of a worker versus improving the accuracy of our aggregated worker responses.
- Our system actively blocks poor performing workers by automatically learning about their accuracy. This happens while the batch is still receiving votes, as opposed to other approaches which wait for a batch to finish.
- Our approach to estimating worker accuracies is different from what we have previously seen used in other crowdsourcing work. Compared to standard EM-based variants, our approach is specifically designed to be more robust at estimating worker accuracies when we have few votes for a worker. This property is crucial in a setting where we must estimate worker accuracies in an online manner.
- We discuss a novel way of incrementally updating our worker accuracy estimates in an online manner when new votes arrive.
- We provide thorough insights about the *practical* challenges that need to be considered in order to deploy such systems seamlessly to workers on Amazon Mechanical Turk. Most of these practical insights have not yet been discussed in the literature.

2. PAPER OUTLINE

We organize our paper as follows:

- We begin by providing a high-level overview of CrowdDQS in Section 3.
- We next describe the algorithms that are run when a worker submits a new vote — our approach to worker accuracy estimation (Section 4), vote aggregation (Section 5), and our criteria for blocking workers (Section 6).
- We next describe what happens when a worker *requests* a new task from CrowdDQS, and how we dynamically assign a task to the worker (Section 7).
- In Section 8, we describe how to extend this dynamic question selection to also decide when it is most beneficial to issue gold standard questions.
- In Section 9, we describe how to update our worker accuracy algorithm to handle incremental updates, allowing for real-time performance.
- In Section 10, we characterize our algorithms by running experiments on synthetic data. We evaluate our worker accuracy estimation approach and incremental updating algorithm, and we demonstrate that static GSQ strategies often reduce predicted answer accuracy.
- In Section 11, we deploy CrowdDQS live to over 1000 workers on Amazon Mechanical Turk and show that we obtain significant improvements over standard approaches. We find that our approach can reduce the number of questions needed to reach a given accuracy by a factor of up to 6x.
- In Section 12, we discuss how to overcome the practical challenges of deploying such a system live to Amazon Mechanical Turk.
- Finally, we describe related work in Section 13, and summarize our work in Section 14.

3. CROWDDQS — OVERVIEW

Our dynamic question selection system, CrowdDQS, takes as input:

- A question bank Q of N multiple-choice questions that should be answered by the crowd. Our system assigns these questions to crowdworkers as they request work. Specifically, we define $Q = \{q_i\}, 1 \leq i \leq N$, where q_i represents the i-th question in the question bank. We also take as input c_i, the number of choices for the i-th question q_i.

- A set of N_G gold standard questions $Q_G = \{q_i^*\}$, as well as their associated answers $A_G = \{a_i^*\}$, $1 \leq i \leq N_G$. q_i^* and a_i^* represent the i-th gold standard question and answer, respectively. We denote c_i^* as the number of choices for q_i^*, and thus each $a_i^* \in \{1, 2, \cdots, c_i\}$.
- A budget B, i.e. the total number of votes we are willing to solicit from crowd. Note that this budget can be spent on gold standard questions or votes on questions in Q.

Our system responds to two different types of events until the budget B is exhausted — a worker requesting a question to vote on, and a worker submitting a vote to a question we provide to them.

- When a worker requests a question, the system utilizes the current set of answers it has received in order to select a question to present to a worker — this question can be from either the question bank Q or the gold standard set Q_G. The system makes sure to not present the worker with the same question multiple times. Note that in crowdsourcing systems such as AMT, it is not known in advance which workers will choose to work on our tasks.
- When a worker submits a vote to a question, the system records the worker ID and response that is received. Using the set of previous answers collected and this new worker response, the system can then block workers that have answered all available questions or that are estimated to have too high an error rate.

Finally, when the system has exhausted its budget B, it aggregates the set of worker responses, producing a set of predicted answers $\hat{A} = \{\hat{a}_i\}, 1 \leq i \leq N, \hat{a}_i \in \{1, 2, \cdots, c_i\}$. Our goal is to use our question selection strategy to maximize the accuracy of this final set of answers.

3.1 Implementation Details

A common misconception about a dynamic question selection approach is that it requires us to issue questions to a crowdsourcing platform one by one. This would be extremely problematic — issuing questions one by one can add an unacceptable amount of latency in completing a set of tasks because workers cannot work in parallel.

Instead, we deploy our question selection algorithms via an **ExternalQuestion** on Amazon Mechanical Turk (AMT). This allows us to issue all of our questions in a single batch, but still delay choosing which question to display to workers until the last possible moment. When a worker requests to answer a question from our group of tasks, the worker ID is sent to a server that is hosting our **ExternalQuestion**. Our server dynamically chooses which question from Q or Q_G to present to the crowdworker based on the latest set of worker responses our server has observed.

We discuss other practical challenges of deploying CrowdDQS onto Amazon Mechanical Turk in Section 12. In Appendix D, we describe various extensions to CrowdDQS — e.g. handling batching multiple questions to workers, incorporating machine-learning based priors, and modeling individual question difficulty.

4. WORKER ACCURACY ESTIMATION

In CrowdDQS, we attempt to estimate worker accuracies in order to guide our vote aggregation and question selection strategies. While there are many techniques used in the literature to perform this accuracy estimation after having received all worker responses, we wish to perform this estimation in an online manner, before we have received all worker responses. In our setting, the number of questions answered by any worker may initially be very low, and many questions will have few votes.

4.1 Standard EM-based approaches

Though expectation-maximization [7] is commonly used as a technique to aggregate votes and estimate worker accuracies [11, 13, 22, 25, 31, 34, 39, 42, 46], it is usually employed after all responses are collected. A typical approach with EM would be to first produce a set of aggregated answers assuming all workers have equal accuracies (the expectation step). Then, given these aggregated votes, we would compute the maximum likelihood accuracies of workers (maximization step). We would repeat this process, using these new accuracies to recompute the aggregated answers until the aggregated answers converge.

This approach works well in practice when we have already collected many answers for each question. However, when there are few answers, EM is overzealous in its estimates for worker accuracies because the expectation step forces us to choose a "correct" answer for each question, even if the current set of votes makes us unsure of what the correct answer should be. This leads to extreme estimates for the worker accuracies when there are a few number of votes.

Consider a simple example where we have two workers, w_1 and w_2, who have each voted on two yes/no (Y/N) questions. The responses for q_1 are $\{w_1 : Y, w_2 : Y\}$ and for q_2 they are $\{w_1 : Y, w_2 : N\}$ — i.e. the workers agree on q_1 but disagree on q_2. EM could converge on a "yes" answer for q_1 and a "no" answer for q_2. However, this would imply the maximum likelihood accuracy for w_1 to be 100% and for w_2 to be 50%. If we were then to use these accuracies to estimate the probability each of our answer estimates were correct, we would assume both answers were correct with 100% probability (because we assume that anything w_1 is always correct), without capturing the fact that we should be less confident in our aggregated answer for q_2 than our aggregated answer for q_1.

4.2 Marginal Likelihood Estimation

In our approach, we avoid being so overzealous in our estimates by taking into account the uncertainty due to each question's votes. To do so, for each worker we compute the likelihood of observing their votes, without making any assumption as to what the correct answer is for each question — instead, each answer choice is weighted by the currently observed evidence. Essentially, we estimate the marginal likelihood distribution of the accuracy of each worker, given the available evidence. We then use this likelihood distribution to compute the expected value of a worker's accuracy. We explain this further below.

Let $\hat{p}_j$ be the estimated accuracy for worker w_j, where we interpret $\hat{p}_j$ to mean that the worker answers a question correctly with probability $\hat{p}_j$, and answers a question incorrectly with probability $1 - \hat{p}_j$, with each incorrect answer choice being equally likely. Let Q_j^W be the set of question indices corresponding to the questions that worker w_j has answered, let W_i^Q be the set of workers that have answered question q_i, and let a_{ij} be the response of worker w_j to question q_i. Let g_j^R be the number of gold standard questions correctly answered by worker j, and let g_j be the total number of GSQs answered by this worker.

For clarity, assume that w_j refers to the worker whose accuracy, p_j, we are currently computing. We fix all other worker accuracies to their currently estimated value, and we can then estimate the expected accuracy of w_j by temporarily assuming all other worker accuracies are fixed. We then update our estimate for w_j's accuracy and repeat this process for the next worker. We continue until worker accuracies have converged or we've reached a fixed number of iterations. In practice we find this seems to usually converge in just 1 or 2 iterations. We initially estimate the worker accuracies by computing the fraction of times each worker agrees with the majority vote for a question, but when new votes arrive, we can initialize our estimates using our previously computed accuracies. More concretely, our approach is to repeat the following until the worker accuracies do not change more than a small delta ($\delta = 0.01$) between iterations:

1. For each worker w_j, use the following rule to update w_j's estimated accuracy:
$$\hat{p}_j' = \frac{\int_{p_{\min}}^{1} pL_j(p)\,dp}{\int_{p_{\min}}^{1} L_j(p)\,dp},$$
where $L_j(p)$ is the likelihood of worker w_j's accuracy being p given the available evidence:
$$L_j(p) = p^{g_j^R}(1-p)^{g_j - g_j^R} \times \prod_{i \in Q_j^W} \sum_{c=1}^{c_i} \left(p\mathbf{1}_{a_{ij}=c} + \frac{1-p}{c_i - 1}\mathbf{1}_{a_{ij}\neq c} \right) \times \prod_{k \in W_i^Q, k \neq j} \hat{p}_k \mathbf{1}_{a_{ik}=c} + \frac{1-\hat{p}_k}{c_i - 1}\mathbf{1}_{a_{ik}\neq c}.$$
2. If $\max_j |\hat{p}_j' - \hat{p}_j| \geq \delta$ and we haven't exceeded the maximum number of iterations, go back to the first step — otherwise, return the newly updated worker accuracies.

Essentially, for each question answered by w_j, we compute the likelihood of observing the other workers' responses over all possible values for w_j's accuracy. We then integrate over this likelihood distribution to compute the expected value of w_j's accuracy. We assume a minimum accuracy $p_{\min}$ which represents our assumption that workers are better than random. For example, for binary questions we might assume $p_{\min}$ to be 50%.

In our previous example, our approach would yield estimated worker accuracies of approximately $w_1 = 74\%$ and $w_2 = 74\%$. It makes intuitive sense that our estimates for both workers accuracies should be the same, as there is an ambiguity in what the correct response should be to question 2. Using methods described later in Section 5, we would also estimate 89% confidence in a "yes" answer for q_1 and 50% confidence in a "no" answer for q_2, thus capturing the notion of more uncertainty in our aggregated answer for q_2.

Remarks. We compare this method against standard EM in our synthetic data experiments in Section 10.1, and find our approach significantly reduces the root mean squared error (RMSE) in predicting worker accuracies when there are few votes per worker. A drawback of our approach is that a naive implementation is slightly more computationally intensive than EM — however, in Section 9, we describe how to significantly optimize our approach by incrementally updating worker accuracies when we receive new votes.

5. VOTE AGGREGATION

Once we have estimated worker accuracies (described in Section 4), the maximum likelihood answer choice for a question can be computed in a straightforward Bayesian manner.

Given a set of evidence E, and true correct answer a_i for question i, the probability that a_i is a specific answer choice c^* can be computed using Bayes' rule:
$$\begin{aligned} P(a_i = c^* \mid E) &= \frac{P(E \mid a_i = c^*)P(a_i = c^*)}{P(E)} \\ &= \frac{P(E \mid a_i = c^*)P(a_i = c^*)}{\sum_c P(E \mid a_i = c)P(a_i = c)}. \end{aligned}$$
To simplify the discussion, we assume each answer choice is equally likely *a priori*, and thus $P(a_i = c) = \frac{1}{c_i}$. If instead the answer choices have a skewed distribution, our equations can be easily modified to account for this. The maximum likelihood answer to question q_i, which we denote as $\hat{a}_i$, can be easily computed by finding the answer choice which maximizes the numerator of our previous equation:
$$\hat{a}_i = \operatorname*{arg\,max}_{c \in \{1,2,\cdots,c_i\}} \prod_{j \in W_i^Q} \hat{p}_j \mathbf{1}_{a_{ij}=c} + \frac{1-\hat{p}_j}{c_i - 1}\mathbf{1}_{a_{ij}\neq c}.$$

The estimated probability of $\hat{a}_i$ being the true answer for q_i, $\hat{\pi}_i = P(\hat{a}_i = a_i)$, can also be given by normalizing the previous result. We can refer to this as the *confidence* of CrowdDQS that the correct answer to q_i is our estimate $\hat{a}_i$:
$$\hat{\pi}_i = \frac{\prod_{j \in W_i^Q} \hat{p}_j \mathbf{1}_{a_{ij}=\hat{a}_i} + \frac{1-\hat{p}_j}{c_i-1}\mathbf{1}_{a_{ij}\neq \hat{a}_i}}{\sum_{c=1}^{c_i} \prod_{j \in W_i^Q} \hat{p}_j \mathbf{1}_{a_{ij}=c} + \frac{1-\hat{p}_j}{c_i-1}\mathbf{1}_{a_{ij}\neq c}}.$$

As an example, suppose we have three workers with estimated worker accuracies of 90%, 80%, and 60%. Suppose that these workers vote on a yes/no question with the votes {Yes, No, No}, respectively. The probability of observing the evidence given that the correct answer is "yes" is $0.9 * (1 - 0.8) * (1 - 0.6) = 0.072$, and the probability of observing the evidence given that the correct answer is "no" is $(1 - 0.9) * 0.8 * 0.6 = 0.048$, and thus the maximum likelihood answer for the question would be computed as "yes." The estimated probability of "yes" being the true, correct answer would be $0.072/(0.072 + 0.048) = 60\%$. In this case, even though there are two votes for "no," because the source of these votes are from lower quality workers, they get outweighed by a single high quality "yes" vote.

6. BLOCKING

We can get further improvements to CrowdDQS by incorporating a strategy to dynamically block poor-performing workers and spammers. In our earlier estimate of worker accuracies, for each worker we estimated the marginal likelihood distribution of their accuracy. This leads to a natural criterion for blocking a worker: namely, we wish to block a worker when we think it is likely that a *replacement worker* would have a higher accuracy than the worker we are considering to block.

Because we compute the marginal likelihood distribution curve for each worker while estimating their accuracy, we can just integrate over this curve to find the probability that $P(p_j \leq \hat{p}_{avg})$, where $\hat{p}_{avg}$ is the estimated average worker accuracy for workers on our task. If we denote the number of workers we have observed as N_W, $\hat{p}_{avg}$ is just:

$$\hat{p}_{avg} = \frac{1}{N_W}\sum_j \hat{p}_j.$$

If the probability of the worker being worse than average is high enough (in our experiments, we used a blocking threshold, T_B, of 60% or more), we block the worker, i.e. we block if the following is true:

$$P(p_j \leq \hat{p}_{avg}) = \frac{\int_{p_{\min}}^{\hat{p}_{avg}} L_j(p)\,dp}{\int_{p_{\min}}^{1} L_j(p)\,dp} \geq T_B.$$

Remarks. In Section 12, we discuss how to practically block a worker without negatively affecting a requester's reputation, and how to block a worker so that they can still continue to work on different groups of tasks by the same requester. If needed, we can adjust T_B to trade-off precision and recall for our blocking strategy. In theory, T_B can be any value over 50%, but we set $T_B = 60\%$ because it works well in practice without overzealously blocking workers.

7. DYNAMIC QUESTION SELECTION

When a worker w_j requests to vote on a new question from CrowdDQS, how should we assign the question to ask this worker? We first briefly describe the default round-robin (RR) approach used on crowdsourcing systems such as Amazon Mechanical Turk, before describing our dynamic question selection (DQS) approach, which uses existing votes to dynamically attempt to assign the best question to any given worker. For now, we do not consider gold standard questions, instead revisiting them in Section 8.

7.1 Typical Approach — Round Robin

In the naive round robin (RR) approach, we allocate our budget B to be evenly distributed among the questions in Q, without issuing any gold standard questions. For example, if we have $B = 150$ and $N = 50$, then each question in Q would be assigned 3 votes. If we increased B to 175, we would assign the first 25 questions in Q to have 4 votes, and the last 25 questions to have 3 votes.

7.2 Our Approach — Maximize Potential Gain

In our approach, we take into account previous worker responses and select the question that has the most potential to increase the expected accuracy of our final set of aggregated votes. Our goal with CrowdDQS is to assign questions to workers in order to maximize the overall accuracy of the aggregated responses $\hat{A} = \{\hat{a}_i\}$. The current accuracy of our set of aggregated votes can be estimated as $\frac{1}{N_Q}\sum_i \hat{\pi}_i$, where $\hat{\pi}_i$ (defined in Section 5) refers to CrowdDQS's current estimated confidence for question i.

The intuition behind our question selection algorithm is that we greedily choose the question q_i^* in Q whose confidence $\hat{\pi}_i$ stands to increase the most if we receive another vote from worker w_j. This also corresponds to the question that would most increase our overall expected accuracy $\frac{1}{N_Q}\sum_i \hat{\pi}_i$. Of course, the confidence that CrowdDQS has for its aggregated response to q_i^* will depend on w_j's vote for that question, which is unknown to us when we assign a question to the worker.

However, the maximum potential gain in our confidence occurs if the worker responds with a vote which agrees with the current aggregated response for q_i^*, as a dissenting vote would make us less certain about our aggregated response. Thus, for any given q_i, we compute the maximum potential gain in accuracy for q_i by recomputing the confidence $\hat{\pi}_i$ if w_j were to respond with our current estimated true answer for q_i, i.e. $\hat{a}_i$. We denote this estimated gain as $G_i(\hat{p}_j)$:

$$G_i(\hat{p}_j) = \frac{\hat{p}_j I_{i,\hat{a}_i}}{\hat{p}_j I_{i,\hat{a}_i} + \sum_{c\neq\hat{a}_i} \frac{1-\hat{p}_j}{c_i-1} I_{i,c}} - \hat{\pi}_i,$$

where we define the auxiliary notation $I_{i,c}$ to refer to the likelihood product that a_i is c:

$$I_{i,c} = \prod_{k\in W_i^Q} \hat{p}_k \mathbf{1}_{a_{ik}=c} + \frac{1-\hat{p}_k}{c_i-1}\mathbf{1}_{a_{ik}\neq c}.$$

Thus, our question selection algorithm assigns to worker w_j the question which maximizes $G_i(w_j)$, i.e.:

$$q_i^* = \arg\max_i \; G_i(\hat{p}_j).$$

In practice, this is a lightweight computation because our incremental updating algorithms cache $I_{i,c}$ for each answer choice of each question (described in Section 9).

8. DYNAMIC GOLD STANDARD QUESTION SELECTION

In later experiments detailed in Section 10.3, we find that typical static gold standard question strategies (i.e. strategies where all workers are issued gold standard questions up front) actually *reduce* accuracy if the budget is kept constant. These results make it clear that the only way for us to realize any practical gains from a gold standard question approach is to issue them dynamically — i.e. we must be prudent when asking gold standard questions, only asking them when it appears likely that we can increase our accuracy. CrowdDQS allows us the flexibility to implement such dynamic approaches which would otherwise be difficult to deploy on a crowdsourcing platform.

Another key advantage of CrowdDQS is it allows for us to issue gold standard questions to workers without them realizing they are being asked a gold standard question, because GSQs can be asked at any time to a worker. The typical approach to issuing gold standard questions is to ask workers to successfully pass a qualification test before allowing them to work on any other tasks by a submitter. The problem with this approach is a spammer can do well on the qualification test but then start spamming once granted access to a submitter's tasks. In CrowdDQS spammers will be much less likely to know when they are being asked a gold standard question.

8.1 Dynamic Gold Standard Question Issuing Strategy

We improve upon static gold standard question strategies by taking a more conservative approach to issuing gold standard questions. We first compute the gains for each question using our DQS approach. Let the incoming worker be denoted as w_j. Let $S(N)$ refer to the top N potential questions in Q, ordered by decreasing gain from the calculations of our DQS strategy from Section 7. First, we only consider issuing a gold standard question if we determine that an incorrect answer to the GSQ would immediately lead to us blocking

w_j, i.e. before considering if a GSQ should be issued, first we check if:

$$\frac{\int_{p_{\min}}^{\hat{p}_{avg}} (1-p) L_j(p)\, dp}{\int_{p_{\min}}^{1} (1-p) L_j(p)\, dp} \geq T_B.$$

The left-hand side of the previous expression computes the probability of the worker being worse than average if we assume an incorrect GSQ response is added to their set of evidence (adapted from Section 6). If that is the case, we next consider the expected potential gain by asking a GSQ. To do so, we must estimate the number of future questions the worker will answer including the GSQ, denoted as F_j. We use the heuristic that a worker will likely answer as many questions as they have already contributed, but to remain conservative, we cap this estimate: $F_j = \max(|Q_j^W|, 4)$. However, by asking a GSQ, we must account for the fact we will only get to ask the worker $F_j - 1$ other questions. We will assume that those questions will be given by our DQS algorithm from before.

If the worker gets the GSQ question correct, we improve our confidence in the questions that the worker has already answered because we have an increased accuracy estimate for the worker. The new estimated accuracy of w_j after correctly answering a GSQ, denoted as $\hat{p}_j^R$, would be:

$$\hat{p}_j^R = \frac{\int_{p_{\min}}^{1} p^2 L_j(p)\, dp}{\int_{p_{\min}}^{1} p L_j(p)\, dp}.$$

By using this new estimate for w_j's accuracy, we can recalculate our confidence in each question w_j answered. Let $\hat{\pi}_i^R$ be the new confidence of question i when using $\hat{p}_j^R$ as the estimate for w_j's accuracy. Thus if the worker correctly answers the GSQ, we can then say the expected potential gain in accuracy on w_j's previously answered questions is $\sum_{i \in Q_j^W} \hat{p}_j (\hat{\pi}_i^R - \hat{\pi}_i)$. We next consider what happens with the $F_j - 1$ other questions w_j would answer. We can estimate that our DQS algorithm would issue the questions in $S(F_j - 1)$ to w_j. If the worker answers the GSQ incorrectly, we will block the worker and we can assume that the questions in $S(F_j - 1)$ will be answered by a replacement worker with accuracy $\hat{p}_{avg}$ (the estimated average worker accuracy). This gives us a total estimated potential gain for asking a GSQ:

$$G_{GSQ} = \sum_{i \in Q_j^W} \hat{p}_j (\hat{\pi}_i^R - \hat{\pi}_i) + \sum_{i \in S(F_j - 1)} \hat{p}_j G_i(\hat{p}_j^R) + (1 - \hat{p}_j) G_i(\hat{p}_{avg}).$$

If we do not issue a GSQ, the potential gain is just the sum of the DQS computed gains of the top F_j questions (as opposed to $F_j - 1$ questions we consider when asking a GSQ):

$$G_{NoGSQ} = \sum_{i \in S(F_j)} G_i(\hat{p}_j).$$

We then issue a GSQ if $G_{GSQ} > G_{NoGSQ}$ and if the worker has not already answered all available GSQs.

This approach is much better than naive static GSQ strategies because it only considers asking gold standard questions if there appears to be a short-term realizable gain in doing so. However, there are still several shortcomings with this approach. First, it only considers asking a GSQ if it immediately leads to a blocking decision — a better strategy might try to look further out into the future. The strategy also takes a conservative approach to estimating the amount of future work we get from a worker — better estimates of the amount of future work would allow our strategy to more accurately estimate the potential gain of issuing a GSQ.

Despite these shortcomings, in our later experiments (Section 11), we see that our dynamic GSQ approach yields significant benefits on real-world data in how quickly our system is able to obtain accurate results.

9. INCREMENTAL UPDATES FOR REAL-TIME PERFORMANCE

Because our system is choosing questions to present to workers on-the-fly, it is important for us to be aware of the computational performance characteristics of our algorithms. At a high level, each time we receive a new vote and issue a new question, we must (1) recalculate worker accuracies given the new vote evidence, and (2) compute the most beneficial next question to solicit a new vote for, given our new worker accuracy estimates. In practice, computation time is dominated by needing to recalculate worker accuracies in the presence of new vote evidence.

We can analyze the average-case performance requirements of a naive non-incremental approach to executing these steps. We will calculate the total computational complexity of executing our worker accuracy estimation algorithm for a budget of B questions.

For simplicity of this analysis, we will assume we are not selecting gold standard questions or blocking workers. Conceptually, in a naive non-incremental approach to recalculating worker accuracies, we repeat the following until convergence:

- For each worker j, compute the likelihood distribution of the worker's accuracy, assuming every other worker's accuracy is fixed. Use this probability distribution to compute the expected value of worker j's accuracy.
- The computation of this probability distribution requires us to sweep values of the worker accuracy estimate, $\hat{p}_j$, and compute the following for each value of $\hat{p}_j$ that we iterate over:

  $$\prod_{i \in Q_j^W} \sum_{c=1}^{c_i} \prod_{k \in W_i^Q} \hat{p}_k \mathbf{1}_{a_{ik}=c} + \frac{1-\hat{p}_k}{c_i - 1} \mathbf{1}_{a_{ik} \neq c}.$$

  This requires us to loop over all the questions answered by worker j, and then for each question q_i answered by j, we must also loop over all responses by all other workers for q_i.

Suppose that we are currently recomputing worker accuracies for the b-th vote, where $1 \leq b \leq B$. Then, given an average worker lifetime of L questions, on average we will have b/L workers, with $O(L)$ responses per worker, and b/N_Q workers answering each question. Thus the average case computational complexity of recalculating all worker accuracies is roughly $O(b^2/N_Q)$ for the b-th vote, and $O(B^3/N_Q)$ for all B votes. If we make the reasonable assumption that the total budget for N_Q questions is $B = O(N_Q)$, then the total computation time required of our dynamic algorithm is $O(N_Q^2)$ under a naive non-incremental approach.

However, by intelligently reusing intermediate results from previous computations, we can significantly improve upon the $O(N_Q^2)$ naive approach to an $O(N_Q)$ approach. We reduce the computation needed to process a new vote through two mechanisms which we term *incremental worker updating* and *incremental question updating*, described next.

9.1 Incremental Worker Updating

In the incremental worker updating approach, we exploit the fact that a worker j's marginal likelihood distribution only potentially changes under two scenarios: (1) if another worker votes on a question that j already answered, or (2) if we update the accuracy estimate of another worker who has answered a question j already has. Thus, in our iterative approach to worker accuracy estimation, we only need to recalculate worker accuracy probability distributions for workers who meet either of these two conditions.

To do so, we keep track of a graph G where the nodes represent different workers, and an edge between a node j and node k indicates that worker j and worker k have both voted on the same question at some point. Suppose that a worker j votes on question q_i. Then, instead of computing new worker accuracy estimates for all workers, the first iteration of our worker accuracy estimation only needs to recalculate worker estimates for the workers who answered q_i, which is $O(b/N_Q)$ workers on average. Next, if the first iteration of our algorithm causes us to update the accuracies of any workers, for the next iteration of our algorithm we only would calculate new accuracies for workers who shared an edge in G with the newly updated workers. On average, the number of edges for a worker is roughly $O(bL/N_Q)$. In practice, we usually only need one or two iterations for our worker accuracy estimates to converge, and thus instead of recomputing accuracy estimates for all $O(b/L)$ workers, we usually recompute accuracies for $O(b/N_Q)$ workers (first-order approximation).

9.2 Incremental Question Updating

In the incremental question updating approach, we accelerate the computation of the worker accuracy probability distribution curve. For a worker j whose accuracy probability distribution we are trying to estimate, instead of recomputing

$$\prod_{i \in Q_j^W} \sum_{c=1}^{c_i} \prod_{k \in W_i^Q} \hat{p}_k \mathbf{1}_{a_{ik}=c} + \frac{1-\hat{p}_k}{c_i - 1} \mathbf{1}_{a_{ik} \neq c}$$

for each value of $\hat{p}_j$ in our sweep, we instead keep track of the following intermediate calculation $I_{i,c}$ for choice c of question i (first defined in Section 7):

$$I_{i,c} = \prod_{k \in W_i^Q} \hat{p}_k \mathbf{1}_{a_{ik}=c} + \frac{1-\hat{p}_k}{c_i - 1} \mathbf{1}_{a_{ik} \neq c}$$

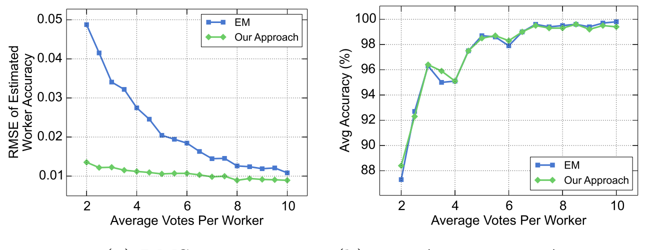
Instead of needing to iterate through all responses to q_i to recalculate $I_{i,c}$, we only need to divide out the contribution caused by the old accuracy estimate and then multiply in the new accuracy estimate's contribution. Thus when updating $\hat{p}_j$ to $\hat{p}'_j$ to estimate the worker accuracy's marginal likelihood, we only compute the following:

$$\prod_{i \in Q_j^W} \sum_{c=1}^{c_i} I_{i,c} \cdot \frac{\hat{p}'_j \mathbf{1}_{a_{ij}=c} + \frac{1-\hat{p}'_j}{c_i - 1} \mathbf{1}_{a_{ij} \neq c}}{\hat{p}_j \mathbf{1}_{a_{ij}=c} + \frac{1-\hat{p}_j}{c_i - 1} \mathbf{1}_{a_{ij} \neq c}},$$

reducing the computation required to estimate the likelihood distribution from $O(bL/N_Q)$ to $O(L)$ for the b-th vote.

Summary. In summary, both of the above approaches, taken together, cause us to reduce the number of workers we recompute accuracies for from $O(b/L)$ to $O(b/N_Q)$, and reduce the probability curve calculation from $O(L) \cdot O(b/N_Q)$ to $O(L)$. Thus the per vote computation time becomes $O(bL/N_Q)$, and the total computation becomes $O(B^2 L/N_Q)$ for B votes. If we again assume that $B = O(N_Q)$, then we have reduced our total computation from being $O(N_Q^2)$ to $O(N_Q)$.

In Section 9.1, we experimentally evaluate the performance gains of implementing these algorithms, and find it greatly reduces the computational requirements of our approach.



(a) RMSE. (b) Vote Aggregation Accuracy.

Figure 1: EM versus our marginal likelihood curve estimation method from Section 4.

10. SYNTHETIC DATA EXPERIMENTS

In Section 11, we will demonstrate that CrowdDQS is able to achieve significant cost reductions in practice on real-world datasets, deploying our algorithms to over 1000 workers on AMT. In this section, however, we supplement our understanding of CrowdDQS and our strategies through use of synthetic data experiments. Synthetic data experiments allow us to finely control and examine the effect that different variables like worker accuracy, worker distribution, and dataset size have on our system.

To carry out our synthetic data experiments we build a simulator to study the operation of CrowdDQS under various scenarios. Our simulator takes the following inputs:

- N — the number of questions in our question bank. For simplicity, we assume each question has 2 choices.
- B — The budget, i.e. the number of votes to simulate.
- L — The worker lifetime, i.e. the total number of votes a worker contributes before leaving our tasks. All workers are assumed to have the same L in our simulations.

Our simulator also takes as input the distribution of worker accuracies. Each simulated worker w_j is assigned a hidden accuracy p_j drawn randomly from this distribution, and when presented a question, w_j provides a correct answer with probability p_j and incorrect answer with probability $1 - p_j$. Workers arrive in a randomized order, are assumed to respond with answers instantly, and provide up to L votes. We allow for the question selection algorithms and worker accuracy estimation to be user-configurable. After all B votes have been simulated, our simulator evaluates the user-configured algorithms by computing the accuracy of the aggregated responses on the synthetic data.

10.1 Worker Accuracy Estimation vs. EM with Limited Votes

In Section 4, we proposed our marginal likelihood curve estimation as an alternative inference method to the com-

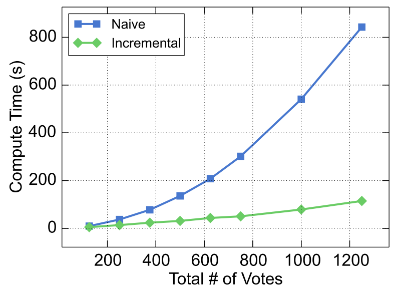


Figure 2: Total simulation computation time in seconds as we vary N, where total votes $B = 5N$, for both our incremental approach and the non-incremental naive approach. Our incremental approach significantly decreases computation time.

monly used expectation maximization (EM) approach to estimating worker accuracies, which we argued is less effective when there is limited worker voting evidence. In Figure 1, we demonstrate this effect on synthetic data. We simulate round-robin question issuing on a synthetic dataset of 10 questions with 10 workers whose accuracies are drawn uniformly at random between 80% and 95%, and we vary the worker lifetime L from 2 through 10 for each worker (i.e. we simulate $B = 10L$ for $L = 2, 3, \cdots, 10$, and each worker is randomly assigned questions). After each simulation, we estimate each worker's accuracy using EM and our marginal likelihood approach, and compute the root mean squared error (RMSE) of the estimates over 100 trials. We plot both the RMSE and average accuracy of aggregating votes using EM and using our approach.

We see in Figure 1a that compared to EM, our approach significantly decreases the RMSE of our estimates for worker accuracy, especially for low numbers of votes. Interestingly, we see in Figure 1b that the accuracy of the aggregated votes under both methods is approximately the same. However, precise worker accuracy estimation is critically important for our dynamic question selection algorithms, because our estimates guide our decision as to which questions most need additional votes. As the number of votes increase, we see that our approach and EM converge to similar RMSEs, but that our approach is much more informative when we have little voting evidence.

10.2 Incremental Updating Performance Gains

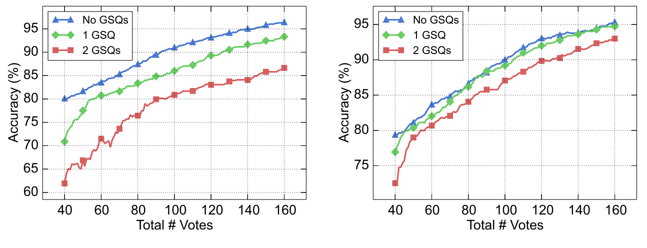
In Section 9, we described an incremental approach to greatly reduce our worker accuracy computational requirements to enable real-time performance. In Figure 2, we vary the number of questions N in our simulator, and measure the total computation time (over the entire run) to run our dynamic question selection algorithms and accuracy estimates for $B = 5N$ votes (we set $L = 10$). We see the full, non-incremental approach has quadratic time complexity whereas the incremental approach has linear time complexity, as our first order analysis indicated. Without these gains, our dynamic question approach would not scale as well when deployed as a real-time system.

10.3 Static Gold Standard Question Strategies

To illustrate some of the challenges with asking gold standard questions, we first consider static gold standard question strategies — i.e. strategies where each newly observed worker is given a set of gold standard questions to answer before they answer any questions from our question bank Q.

We find that issuing a *fixed* number of GSQs is almost always a bad idea, particularly when workers have short lifetimes. With more votes, we would still learn about worker accuracies through correlated voting patterns and have more voting evidence per question, but GSQs often end up wasting a large portion of our budget testing workers (more details in Appendix C). In Figures 3a and 3b, we show the effect asking 1 or 2 GSQs to each new worker has on the overall accuracy of our predicted answers. Even when worker lifetimes are long (which reduces the relative cost of asking a GSQ), as in Figure 3b where $L = 30$, non-dynamic approaches to asking GSQs can significantly decrease our accuracy. For shorter worker lifetimes as seen in Figure 3a ($L = 5$), we spend even more of our budget on GSQs and end up with even lower accuracy as a result of asking GSQs to each worker.

These results suggest that the common approach of asking gold standard questions to all workers is often inadequate, and thus we must be more clever about how to issue gold standard questions to workers, as we described in Section 8.



(a) Short lifetime ($L = 5$). (b) Long lifetime ($L = 30$).

Figure 3: Effect on accuracy of *aggregated votes* of asking each new worker one or more gold standard questions. Here we have $N = 40$ questions, with worker accuracies being drawn uniformly at random from 70% to 90% and DQS (dynamic question selection) + blocking enabled. We see that static GSQ strategies decrease overall accuracy.

11. CROWDDQS: REAL-WORLD SYSTEM

In order to determine the effectiveness of our algorithms in a real crowdsourcing marketplace, we built the CrowdDQS server in order to deploy our algorithms for dynamic question selection and blocking on Amazon Mechanical Turk. While AMT is arguably one of the most prominent marketplaces for microtasking work, it is not designed for dynamic question issuing. The standard idiom of use for AMT is for a requester to post a list of specific questions to Amazon Mechanical Turk in a batch, and then to also request that each question get a fixed number of votes from unique workers. Thus, we require some clever engineering in order to deploy our algorithms on AMT (for more details, see Section 12).

At a high-level, the CrowdDQS server works by allowing us to upload a question bank of multiple choice questions to be answered, a bank of available gold standard questions, and the total number of votes to ask workers. The server then creates a new batch of tasks on Mechanical Turk, with one task created for each vote in the budget. For example, if our budget was set at 250 votes on a question bank of 50 questions, our server would create 250 tasks on AMT. However, the question to display in each task is determined later and on-the-fly when a worker chooses to view the task.

As the system receives votes, it uses our algorithms to adjust which questions to display to workers, as well as determine when to block workers from completing more tasks on the system. During this time, the requester is able to see

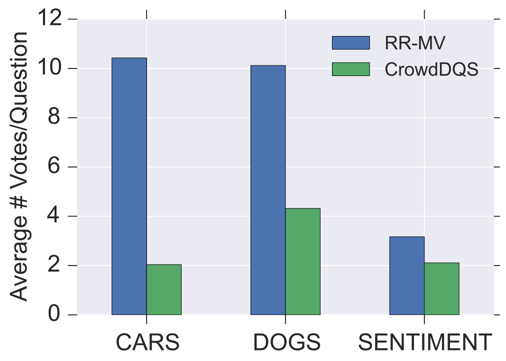


Figure 4: Number of votes required (per question) by our system to reach 100% accuracy, averaged over 5 trials. **RR-MV**: round-robin (RR) with majority voting, **CrowdDQS**: dynamic question selection with blocking and dynamic GSQs enabled. We see the greatest gains on the difficult CARS and DOGS datasets, with more moderate gains on the easy SENTIMENT dataset.

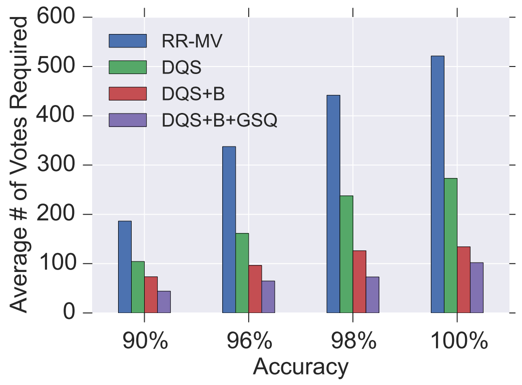


Figure 5: Number of votes required by our system to reach 90%, 96%, 98% and 100% accuracy on CARS, averaged over 5 trials (each run occurred during daylight hours PDT). The four strategies we test are **RR-MV**: round-robin (RR) with majority voting, **DQS**: our dynamic question selection (DQS) with marginal likelihood estimation of worker accuracies, **DQS+B**: DQS with blocking enabled, and **DQS+B+GSQ**: DQS with blocking and dynamic GSQ issuing enabled.

which questions are currently being voted on, blocked workers, estimated worker accuracy, and estimated answers for each question (along with the system's confidence in each answer). To ensure we are able to assign questions in real time to workers, the CrowdDQS server implements our incremental update techniques from Section 9 for recalculating worker accuracies after each vote.

11.1 Experiments

Our CrowdDQS server allows us to choose which approach we use for assigning questions to a worker, as well as for aggregating the worker's results. We test our algorithms on 3 different datasets[1], collecting over 12,000 responses from over 1000 workers:

(1) CARS. We use the CARS dataset of car images from [36], where our question bank Q consists of photos of 50 cars, and users are asked to identify the model of the car in the photo, given two choices. Our GSQ bank Q_G has 10 photos of cars. The options given to the user both have the correct *make* of the car, but only one choice has the correct *model*. Workers are paid \$0.01 per vote.

[1]Examples from each dataset are available in Appendix A.

(2) DOGS. In the DOGS dataset, we use photos of 3 closely related dog breeds from the Stanford Dogs dataset [19]: Australian terriers, Norfolk terriers, and Border terriers. Workers must choose the correct breed out of these 3 choices. Q has 50 photos and Q_G has 6 photos. Workers are paid \$0.02 per vote.

(3) SENTIMENT. In the SENTIMENT dataset, workers must correctly classify the sentiment (positive or negative) of Amazon product reviews (obtained from [26]). Q has 150 product reviews and for our GSQ bank, Q_G, we use 10 product reviews. Workers are paid \$0.01 per vote.

We test four strategies for question selection and vote aggregation. Because many factors can affect the performance of our strategies,[2] we run each strategy 5 times and run all strategies for a dataset at a similar time of day to reduce the variability of our results. For each strategy, we compute the number of votes required to first reach 90%, 94%, 98%, and 100% accuracy. We allow workers to continue voting on each run until we reach 100% accuracy on all tasks. The server is able to compute these statistics because we have labeled ground truth for all our datasets.

The four strategies we test are:

- **RR-MV**: Round-robin question assignment with majority voting (the standard workflow used in Amazon Mechanical Turk).
- **DQS**: Our dynamic question selection (DQS) with marginal likelihood estimation of worker accuracies.
- **DQS+B**: DQS with blocking enabled.
- **DQS+B+GSQ**: DQS with blocking and dynamic GSQ issuing enabled.

11.2 Overall Results

In Figure 4 we plot the average number of votes needed per question to reach 100% accuracy on all our datasets, comparing the standard RR-MV approach with our DQS+B+GSQ (dynamic question selection + blocking + GSQs) strategies from this paper. Any gold standard questions that were issued are also included in these vote totals. We find:

- On the CARS dataset, CrowdDQS uses **5.1x** fewer questions than the standard round-robin approach to accurately answer our questions.
- On the DOGS dataset, CrowdDQS uses **2.3x** fewer questions than the standard round-robin approach.
- On the SENTIMENT dataset, CrowdDQS uses **1.5x** fewer questions than the standard round-robin approach.

In general, we find that our approach with CrowdDQS can yield substantial cost savings while still maintaining accuracy. For example, for the CARS and DOGS datasets, CrowdDQS can obtain 100% accuracy in just 2-4 votes per question, even though 100% accuracy requires 10-11 votes for the round-robin approach. CrowddDQS provides less of an advantage over standard approaches when tasks are very easy like in the SENTIMENTS dataset, which only needs 3 votes per question on average to label accurately with RR. This occurs because for easy tasks, a single pass of votes leads to almost perfect accuracy. Nevertheless, we find CrowdDQS still provides cost savings in these scenarios.

[2]e.g. worker accuracy distribution, the number of spammers we attract, and the number of workers currently active in the marketplace.

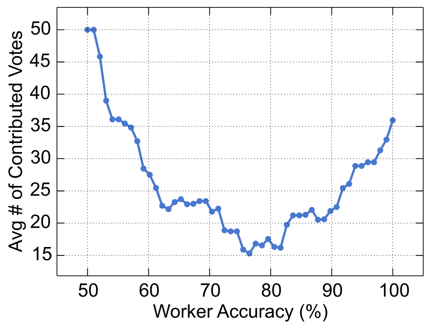


Figure 6: Demonstration of prolific nature of spammers and low-quality workers. We plot the average number of contributed votes versus the estimated worker accuracy for workers who have contributed at least 10 votes. Each data point is smoothed over a window of 5% (i.e. when reporting the average number of votes for 70% worker accuracy, we average the number of votes for workers with accuracy between 65% to 75%).

11.3 In-depth Results

We plot more in-depth results for the CARS dataset in Figure 5. We find that compared to the default round-robin question assignment scheme used on Mechanical Turk, using our techniques can reduce the average number of votes required to reach a given accuracy by up to **6x**. For example, to achieve 98% accuracy on our dataset, we found:

- We use **1.9x** fewer questions than round-robin by using our dynamic question selection (DQS) without using any blocking or gold standard questions.
- We use **3.5x** fewer questions than round-robin on average when we then add in the capability to dynamically block poor-performing workers (DQS+B).
- We use **6x** fewer questions than round-robin on average by opportunistically issuing gold standard questions to workers, in addition to blocking poor-performing workers and using our dynamic question selection (DQS+B+GSQ).

11.4 Discussion

Where do these cost reductions come from? The standard round-robin, majority vote approach cannot differentiate between questions that workers disagree on versus questions which have high worker agreement. This causes us to waste votes on questions that do little to improve the overall accuracy of our aggregated responses. In Appendix C, we further discuss the intuition behind why CrowdDQS is able to achieve these cost reductions, but in summary:

- DQS improves upon RR-MV by discounting votes of low-quality workers and by focusing votes on uncertain questions.
- DQS+B improves upon DQS by blocking low-quality workers from contributing more low-weighted votes, improving the overall accuracy of the pool of workers allowed to work on our task.
- DQS+B+GSQ improves upon DQS+B by using GSQs to detect and remove inaccurate workers and spammers early, instead of allowing them to fly under the radar until corroborating votes arrive.

11.5 General Insights

In previous sections, we specified mathematical criteria for question selection regarding both regular questions and gold standard questions, as well as for worker accuracy estimation. It is easy for a qualitative understanding of our system to be obscured by these equations and algorithms. Here we describe some high-level observations we made during our experiments which can help provide some intuition into how our system behaves.

(1) The initial strategy with DQS is to get votes for every question, as questions with no votes stand the most to gain in expected accuracy by receiving an initial vote.

(2) After this initial pass of votes, questions start to be assigned their 2nd vote. If the 2nd vote does not corroborate the first vote, our DQS algorithm will often immediately try to get a new vote to help resolve the ambiguity. If the second vote does corroborate the first vote, CrowdDQS will move on to soliciting extra votes on another question.

(3) In the process of issuing 2nd, 3rd, and later votes for questions, CrowdDQS may start to notice that some workers often disagree with accurate workers (or at least, workers that seem accurate). The accuracy estimate of these disagreeing workers are then lowered, which causes their votes to carry less weight during vote aggregation, and also potentially leads to blocking the workers.

(4) An interesting side effect of detecting inaccurate workers or spammers is that once CrowdDQS detects a low-quality worker, it will often then assign more votes to questions that were voted on by those workers. This is because CrowdDQS automatically learns that we should assign more uncertainty to questions that have a large proportion of their votes from low-quality workers.

(5) Earlier, we noted that spammers and low-quality workers often contribute a disproportionate number of votes relative to other workers. In Figure 6, we demonstrate this tendency by plotting the average number of votes per worker from the RR-MV CARS trials as a function of the estimated worker accuracy. For this graph only, we exclude data from workers who contribute less than 10 votes to ensure that the accuracy estimates are reasonable. We observe that low-quality workers answer the most questions. (Interestingly, the most accurate workers also contribute many votes.)

(6) One reason why our dynamic approaches are a significant improvement over round-robin is because RR becomes a very expensive strategy as we receive more votes. After a few rounds of votes, many questions will have been correctly resolved by majority voting, but there will still be a few troublesome questions that have high rates of worker disagreement. For example, say we have 120 questions, and after a few rounds of voting, only 3 questions remain which have a large number of disagreements. To get a single vote on these 3 difficult questions, the round-robin approach needs to ask for 120 votes — this is essentially equivalent to saying the votes on the last 3 difficult questions cost 40x more than they should!

(7) There are interesting secondary effects of increasing worker pay. With the typical round-robin approach, the first-order effect of increasing the price paid to workers is that more workers will be attracted to our tasks (thus completing our task more quickly). However, this often also attracts spammers and low-quality workers, which can then lower the quality of our results.

This changes under CrowdDQS. First, we are more robust to the presence of spammers and low quality workers because we block them from consuming our budget and we assign low weight to their votes. However, increasing the price also incentivizes workers to stay *longer* on our tasks, which allows us to get a more accurate estimate of the worker's accuracy. In addition, a higher price attracts more people working on our task *in parallel*. This allows us to get corroborated votes more quickly, giving us *faster* estimates of worker accuracy.

11.6 Insights on Gold Standard Questions

From observing the voting logs of our runs, we observed two scenarios where our system chose to issue GSQs, which we describe below. In Appendix B, we further discuss the intuition behind the trade-offs of asking GSQs.

(1) First, we saw that our system would issue GSQs to a worker if they had answered around 5-10 questions near the beginning of the job, with very few corroborating votes from other workers which would enable us to infer their accuracy. In this scenario, CrowdDQS sees that getting a better estimate of the worker's accuracy is better than the potential increase in predicted answer accuracy by getting a new vote (which provides less information about the worker's accuracy). While this better worker accuracy estimate would only have a small effect on the expected accuracy of each of the 5-10 questions, the cumulative effect on these questions outweighs the impact of a single new vote.

Note that in later stages of the job, we quickly get worker accuracy evidence from correlating voting patterns, and thus GSQs are less useful and rarely issued. This suggests that GSQs have even more practical value when we increase the number of questions in the question bank Q. In these scenarios, without GSQs, bad workers would have the opportunity to answer many questions without being detected.

(2) Second, we somewhat surprisingly also saw GSQs appear to relatively new workers at the *end* of a job. In these instances, there are only a few questions which the system is very uncertain about, and thus new workers are initially prioritized to vote on these uncertain questions. Because these are uncertain questions, this prevents the system from getting a good estimate of the worker's accuracy. In this scenario, it seems like CrowdDQS is more interested in breaking the ties of the uncertain questions by getting a better estimate of the new worker's accuracy rather than by trying to improve the confidence of questions that are already effectively resolved.

12. PRACTICAL CHALLENGES

In order to implement the CrowdDQS server live on Amazon Mechanical Turk, there were several practical challenges we had to overcome.

(1) Dynamic Question Selection. In order to dynamically display questions to workers, we make use of AMT's **ExternalQuestion** option for submitting tasks. A flawed solution for dynamic question selection would be to issue questions to AMT sequentially, where only one question is issued at a time. This is undesirable because (a) it precludes workers from working in parallel on our tasks, and (b) it would vastly increase latency because workers are often very unwilling to work on small batches of tasks. Workers prefer to know that they can work on several questions in a batch before moving on to a new batch.

With ExternalQuestions, a requester can specify an external URL where their question is hosted. Thus, we create tasks on AMT all with the same ExternalQuestion URL on our CrowdDQS server, and use our algorithms to dynamically choose what is displayed at this URL.

(2) Blocking. While AMT provides a method for blocking workers, this method negatively affects the worker's reputation on AMT and also blocks the worker for all of the requester's tasks, even for different types of questions posted by the requester. If a worker feels that a requester is unfair in their blocking decisions, they often voice their complaints on forums to other workers, which then diminishes the ability of the requester to find willing workers.

Instead, we use AMT's *qualification* system to prevent blocked workers from answering questions in our batch, while still allowing any unblocked worker to vote on our questions. The qualification system allows for requesters to assign arbitrary credentials to workers on the platform, and then specify that questions are only available to users that have either the presence or absence of some set of these credentials. For each new batch of questions uploaded to the CrowdDQS server, we create a new qualification on Amazon Mechanical Turk, and require that workers *do not* have this qualification assigned to them in order to work on tasks in the batch.

Thus, initially all workers are free to work on the requester's tasks, but if needed, workers are blocked by *assigning* them the newly-created qualification for a batch. This mechanism does not adversely affect a worker's reputation, and still allows them to work on other questions in different batches by the same requester.

(3) Real-time Decisions. Our algorithms rely on the current set of voting evidence to choose the best questions to ask workers. Thus, it is imperative that we are immediately notified when there is a new vote so that we can update our worker accuracy estimates and determine the next question to assign. CrowdDQS achieves this by interjecting code that first submits worker responses directly to our server before they are submitted to Amazon Mechanical Turk. This mechanism is invisible to the crowdworker, but it allows our server to immediately respond to new voting evidence. Once we receive a new vote, we use our incremental algorithms of Section 9 to efficiently update our accuracy estimates.

(4) Distinguishing Between Workers. AMT allows workers to preview a requester's task before deciding whether to accept it. During this preview period, AMT does not pass along any information to our server indicating the ID of the worker viewing our question. Once the worker decides to accept a task, only then does AMT pass along a worker ID to the server. This is problematic because our algorithms rely on being able to distinguish between workers to determine which question should be assigned next.

To work around this, we must set a cookie on the worker's machine to cache the worker's ID on Amazon Mechanical Turk after the first time they accept a task from our server. Thus, for future questions previewed by the worker, we are able to display the correct question by using the cached worker ID. The first time a worker previews one of our questions, we do not have an ID stored for them, so we just treat them as a new worker.

(5) Non-instantaneous Task Completion. Workers do not complete tasks instantly. Thus, it is possible for a worker to be in the midst of answering a question when a new worker comes to request to work on a task. To avoid assigning the same question over and over to different workers, we implement a reservation system in the server, where the server keeps tracks of questions that are currently being voted on by workers, and excludes reassigning those questions to new workers until after the questions have been voted on.

(6) Discarded Tasks. Unfortunately, Amazon Mechanical Turk does not notify our server if a worker *discards* a task. This is problematic — if we do not know which tasks have been discarded, the system can spend a long time waiting for a vote that never arrives. This can lead to that question being starved of votes, and ultimately greatly reducing accuracy. To counter this, we implement several mechanisms to detect when a worker has discarded a question.

The first mechanism we use is to implement a *heartbeat* signal into our question's JavaScript. Invisible to the worker, the question the worker is viewing pings the CrowdDQS server every 10 seconds to let our server know that the worker still has a tab open with the question that they accepted. If a worker closes their window, this heartbeat signal stops, and the system knows to allow that question to be reassigned to other workers. Another mechanism we implement attempts to send a message to our server when a worker closes their browser tab, directly notifying our server that a question has been discarded. This mechanism is not fully reliable (e.g. if a worker's Internet goes out or if their browser crashes, then this message will not have an opportunity to be sent), and thus we still need to use our heartbeat mechanism. Finally, if a user has been idle for 3 minutes without answering a question, we release the question back into the pool of assignable questions.

The trade-off we make here is that occasionally we may assign an extra vote or two to a question, but this is greatly preferable to starving a question of votes.

13. RELATED WORK

As mentioned in the introduction, the idea of dynamically issuing tasks to workers has been discussed before in the literature [4, 10, 15, 23, 41, 46]. Some prior work assume that the set of workers is fixed and attempt to match tasks optimally to these workers [1, 5, 15, 43, 44, 45], though this assumption is less applicable to prominent crowdsourcing systems like Amazon Mechanical Turk (which we focus on), where we do not know which workers will work on our tasks. Other work consider a worker setting similar to ours, but either do not deploy live to an existing crowdsourcing system [4, 12, 15, 16, 32, 33], or dynamically assign questions using multiple batches [23, 41], and thus do not respond immediately to new voting evidence, nor do they need to perform their computations in real-time. Some work [1, 12, 23] consider *early termination*, where questions are given more votes until a fixed confidence threshold is reached. This is different from our approach which focuses votes on the most beneficial questions that have not yet been terminated.

QASCA [46] and iCrowd [10], are the most closely related work that we are aware of. Like our work, they [10,46] make use of Amazon's ExternalQuestion mechanism to dynamically issue tasks to workers. QASCA's [46] task assignment criteria is similar to ours (QASCA uses expected accuracy gain instead of maximum potential accuracy gain), but they use EM [7] to estimate worker accuracies, which we argue (Section 4) is less robust when there is little evidence for a worker. iCrowd [10] attempts to match workers with tasks based on inferred topic-level expertise, and initially assigns GSQs to new workers to learn about their accuracy on a diverse set of topics (unlike our dynamic approach to GSQs).

However, unlike our work, neither of these systems actively blocks poor performing workers nor adaptively tries to use gold standard questions to improve the accuracy of their systems. We find, somewhat surprisingly, that GSQs are usually only useful when asked in very specific circumstances (see Section 10.3 and 11.6). Other work on GSQs [27, 38] show that GSQs can help estimate the accuracy of workers quickly and accurately, but unlike our work, they usually do not consider the practical trade-off made between asking gold standard questions versus using repeated voting to improve overall accuracy. We also provide incremental updating techniques for estimating worker accuracy that are not discussed in either QASCA or iCrowd. In addition, in Section 12 we discuss in detail many non-obvious practical challenges to deploying such a system that only receive limited treatment in [46] and [10]. For example, these challenges include dealing with parallel task assignment, real-time performance considerations, inferring worker IDs, detecting discarded tasks, and practical blocking mechanisms.

There has also been extensive work on worker accuracy estimation in the literature [2, 6, 11, 13, 14, 22, 30, 31, 34, 42]. However, the vast majority of this work has been in the context of computing worker accuracies in order to improve aggregating worker responses *after* a batch has completed, where we have ample computation time available and many votes per worker. Much of this work [11, 13, 22, 25, 31, 34, 39, 42,46] focuses on expectation maximization (EM) [7,9] based methods, which we show are less desirable for a dynamic approach where we have limited evidence for workers. There are other works [2, 5, 6, 8, 15, 34] which formulate the worker accuracy estimation problem in terms of probabilistic graphical models [21]. Our approach to marginal likelihood curve estimation most closely resembles the work in probabilistic graphical models, where we can consider our formulation of the problem to be similar to expressing a Markov Random Field [20]. Our accuracy estimation is similar to the methods of iterated conditional modes [3] and belief propagation [29] for inference, but we are able to exploit the knowledge of our specific domain to optimize our inference method to easily handle incremental updates, which allows us to practically deploy our algorithms in a real-time setting (Section 9).

14. CONCLUSION

In this paper, we presented CrowdDQS, a system interfacing seamlessly with Amazon Mechanical Turk to assign questions to crowdworkers on-the-fly and in real-time. Unlike other dynamic task assignment systems, CrowdDQS also automatically removes poor workers from the worker pool by (1) inferring worker accuracies through correlated voting patterns between workers, and (2) deciding to directly learn about worker accuracies by selectively issuing gold standard questions to workers. We show that static GSQ strategies are surprisingly ineffective at improving the accuracy of predicted answers, and instead develop an effective dynamic strategy for issuing GSQs. We present worker accuracy estimation algorithms and incremental updating algorithms which allow us to estimate worker accuracies in real-time even with limited evidence. We deploy CrowdDQS live to over 1000 workers on several real-world datasets, and find that CrowdDQS accurately answers questions using up to 6x fewer votes than standard approaches.

15. REFERENCES

[1] I. Abraham, O. Alonso, V. Kandylas, and A. Slivkins. Adaptive crowdsourcing algorithms for the bandit survey problem. In *COLT*, pages 882–910, 2013.

[2] Y. Bachrach, T. Graepel, T. Minka, and J. Guiver. How to grade a test without knowing the answers—a bayesian graphical model for adaptive crowdsourcing and aptitude testing. *arXiv preprint arXiv:1206.6386*, 2012.

[3] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302, 1986.

[4] R. Boim, O. Greenshpan, T. Milo, S. Novgorodov, N. Polyzotis, and W.-C. Tan. Asking the right questions in crowd data sourcing. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1261–1264. IEEE, 2012.

[5] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask?: jury selection for decision making tasks on micro-blog services. *Proceedings of the VLDB Endowment*, 5(11):1495–1506, 2012.

[6] P. Dai, C. H. Lin, D. S. Weld, et al. Pomdp-based control of workflows for crowdsourcing. *Artificial Intelligence*, 202:52–85, 2013.

[7] A. P. Dawid and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied statistics*, pages 20–28, 1979.

[8] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the 21st international conference on World Wide Web*, pages 469–478. ACM, 2012.

[9] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[10] J. Fan, G. Li, B. C. Ooi, K.-l. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1015–1030. ACM, 2015.

[11] J. Feng, G. Li, H. Wang, and J. Feng. Incremental quality inference in crowdsourcing. In *International Conference on Database Systems for Advanced Applications*, pages 453–467. Springer, 2014.

[12] P. G. Ipeirotis, F. Provost, V. S. Sheng, and J. Wang. Repeated labeling using multiple noisy labelers. *Data Mining and Knowledge Discovery*, 28(2):402–441, 2014.

[13] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 64–67. ACM, 2010.

[14] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Evaluating the crowd with confidence. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 686–694. ACM, 2013.

[15] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *Advances in neural information processing systems*, pages 1953–1961, 2011.

[16] D. R. Karger, S. Oh, and D. Shah. Budget-optimal task allocation for reliable crowdsourcing systems. *Operations Research*, 62(1):1–24, 2014.

[17] A. R. Khan and H. Garcia-Molina. Hybrid strategies for finding the max with the crowd: Technical report. 2014.

[18] A. R. Khan and H. Garcia-Molina. Attribute-based crowd entity resolution. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 549–558. ACM, 2016.

[19] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, volume 2, 2011.

[20] R. Kindermann and L. Snell. *Markov random fields and their applications*. 1980.

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

[22] A. Kurve, D. J. Miller, and G. Kesidis. Multicategory crowdsourcing accounting for variable task difficulty, worker skill, and worker intention. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):794–809, 2015.

[23] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. Cdas: a crowdsourcing data analytics system. *Proceedings of the VLDB Endowment*, 5(10):1040–1051, 2012.

[24] A. Marcus, A. Parameswaran, et al. Crowdsourced data management industry and academic perspectives. *Foundations and Trends in Databases*, 6(1-2):1–161, 2015.

[25] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *Proceedings of the VLDB Endowment*, 5(1):13–24, 2011.

[26] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM, 2013.

[27] D. Oleson, A. Sorokin, G. P. Laughlin, V. Hester, J. Le, and L. Biewald. Programmatic gold: Targeted and scalable quality assurance in crowdsourcing. *Human computation*, 11(11), 2011.

[28] 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 2012 ACM SIGMOD International Conference on Management of Data*, pages 361–372. ACM, 2012.

[29] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.

[30] A. Ramesh, A. Parameswaran, H. Garcia-Molina, and N. Polyzotis. Identifying reliable workers swiftly. 2012.

[31] V. C. Raykar, S. Yu, L. H. Zhao, A. Jerebko, C. Florin, G. H. Valadez, L. Bogoni, and L. Moy. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *Proceedings of the 26th Annual international conference on machine learning*, pages 889–896. ACM, 2009.

[32] V. S. Sheng, F. Provost, and P. G. Ipeirotis. Get

another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 614–622. ACM, 2008.

[33] J. W. Vaughan. Adaptive task assignment for crowdsourced classification. 2013.

[34] M. Venanzi, J. Guiver, G. Kazai, P. Kohli, and M. Shokouhi. Community-based bayesian aggregation models for crowdsourcing. In *Proceedings of the 23rd international conference on World wide web*, pages 155–164. ACM, 2014.

[35] V. Verroios and H. Garcia-Molina. Entity resolution with crowd errors. In *2015 IEEE 31st International Conference on Data Engineering*, pages 219–230. IEEE, 2015.

[36] V. Verroios, P. Lofgren, and H. Garcia-Molina. tdp: An optimal-latency budget allocation strategy for crowdsourced maximum operations. Technical report, Stanford University.

[37] N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. *Proceedings of the VLDB Endowment*, 7(12):1071–1082, 2014.

[38] J. Vuurens, A. P. de Vries, and C. Eickhoff. How much spam can you take? an analysis of crowdsourcing results to increase accuracy. In *Proc. ACM SIGIR Workshop on Crowdsourcing for Information Retrieval (CIR'11)*, pages 21–26, 2011.

[39] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494, 2012.

[40] P. Welinder, S. Branson, P. Perona, and S. J. Belongie. The multidimensional wisdom of crowds. In *Advances in neural information processing systems*, pages 2424–2432, 2010.

[41] P. Welinder and P. Perona. Online crowdsourcing: rating annotators and obtaining cost-effective labels. 2010.

[42] J. Whitehill, T.-f. Wu, J. Bergsma, J. R. Movellan, and P. L. Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in neural information processing systems*, pages 2035–2043, 2009.

[43] Z. Zhao, F. Wei, M. Zhou, W. Chen, and W. Ng. Crowd-selection query processing in crowdsourcing databases: A task-driven approach. In *EDBT*, pages 397–408, 2015.

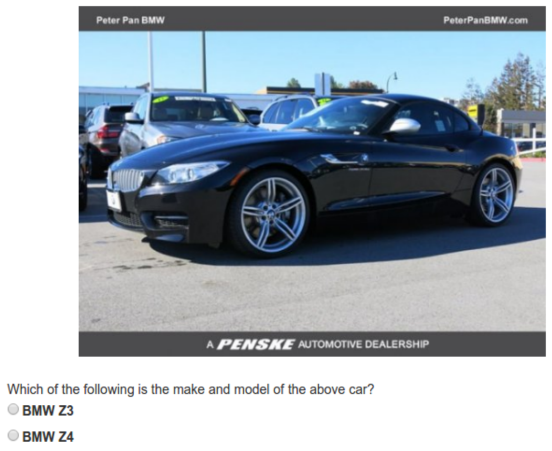
[44] Z. Zhao, D. Yan, W. Ng, and S. Gao. A transfer learning based framework of crowd-selection on twitter. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1514–1517. ACM, 2013.

[45] Y. Zheng, R. Cheng, S. Maniu, and L. Mo. On optimality of jury selection in crowdsourcing. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015*. OpenProceedings. org., 2015.

[46] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng. Qasca: a quality-aware task assignment system for crowdsourcing applications. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1031–1046. ACM, 2015.

APPENDIX

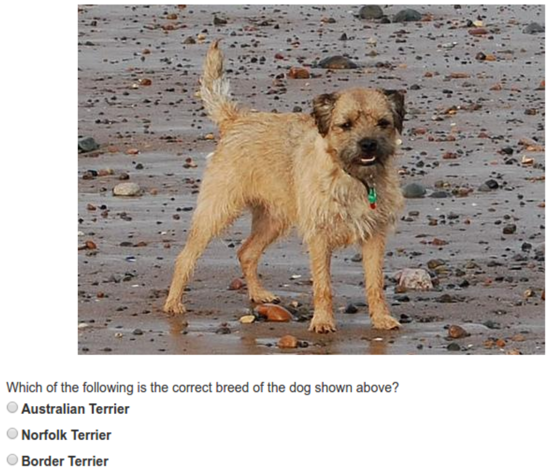
A. DATASET SAMPLE QUESTIONS



Which of the following is the make and model of the above car?

- **BMW Z3**
- **BMW Z4**

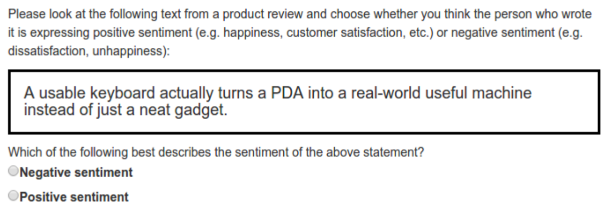
Figure 7: Sample question from the CARS dataset.



Which of the following is the correct breed of the dog shown above?

- **Australian Terrier**
- **Norfolk Terrier**
- **Border Terrier**

Figure 8: Sample question from the DOGS dataset.



Please look at the following text from a product review and choose whether you think the person who wrote it is expressing positive sentiment (e.g. happiness, customer satisfaction, etc.) or negative sentiment (e.g. dissatisfaction, unhappiness):

A usable keyboard actually turns a PDA into a real-world useful machine instead of just a neat gadget.

Which of the following best describes the sentiment of the above statement?

- **Negative sentiment**
- **Positive sentiment**

Figure 9: Sample question from the SENTIMENT dataset.

As mentioned in Section 11, we tested our system on several real-world datasets. We provide examples from each dataset in Figures 7, 8, and 9.

B. CROWDDQS INTUITION

In Section 11, we demonstrated that CrowdDQS can yield significant reductions in the budget required to answer a set of questions by the crowd. Where do the cost reductions of CrowdDQS come from? Here we provide an intuitive treatment of CrowdDQS's operation. The standard round-robin, majority vote approach cannot differentiate between questions that workers disagree on versus questions which have high worker agreement. This causes us to waste votes on questions that do little to improve the overall accuracy of our aggregated responses.

The DQS approach addresses the first problem with RR by focusing votes on questions that have the greatest potential gain in accuracy — in practice, this corresponds to selecting questions which CrowdDQS is most uncertain about. Thus voting efforts are concentrated on the most difficult questions, instead of being diffused over the entire question set.

The DQS approach also is able to mitigate the effect of spammers and inaccurate workers due to our marginal likelihood estimation approach. Spammers are inferred to have close to random accuracy, and thus their votes are weighted significantly less (or discounted entirely) when aggregating worker responses. By discounting the votes of low quality workers, DQS (unlike RR) does not suffer from low-quality workers being able to have a large effect on the aggregated set of votes.

Unfortunately, spammers can often be prolific. By introducing blocking, DQS+B can reduce the impact spammers have even further than the DQS-only approach does. However, to begin to estimate a worker's accuracy, we need for multiple workers to vote on the same questions. Initially, CrowdDQS prioritizes getting at least 1 vote for each question. Thus there is a period early in the life of a batch of questions where spammers can answer questions without being detected. By introducing gold standard questions, DQS+B+GSQ is able to catch these spammers early (see Section 11.6 for more details).

Summarizing again:

- DQS improves upon RR-MV by discounting votes of low-quality workers and by focusing votes on uncertain questions.
- DQS+B improves upon DQS by blocking low-quality workers from contributing more low-weighted votes, improving the overall accuracy of the pool of workers allowed to work on our task.
- DQS+B+GSQ improves upon DQS+B by using GSQs to detect and remove inaccurate workers and spammers early, instead of allowing them to fly under the radar until corroborating votes arrive.

C. GSQ INTUITION

One of our more surprising observations in Section 10.3 was that we must be very conservative when deciding to issue gold standard questions to workers if we wish to improve the accuracy of our predicted answers. There are several reasons why issuing GSQs are so tricky:

- Asking a gold standard question prevents us from asking for votes on questions in our question bank Q, and thus asking gold standard questions to all workers is incredibly wasteful (see Section 10.3).
- After each question in our question bank has received one or more votes, the questions themselves can act as "pseudo-gold standard questions." While we may not know the ground-truth responses for questions in our question bank with 100% certainty, we will have a high confidence in their predicted answers. This reduces the

relative information gain advantage of asking a gold standard question versus asking for a new vote on a question that already has a few votes.

- For a gold standard question to be useful, it must either provide a better accuracy estimate of the worker, or it must somehow lead to us blocking the worker faster. A better accuracy estimate is most useful for aggregating votes, especially when there are ties. However, our dynamic question selection approach already minimizes ties by asking for more votes on questions for which we are less certain of the answer. If we use GSQs in order to reduce the time needed to block a worker, we must also have some estimate of how many more questions the worker will answer for us, which may not be easy to predict.

Because of these difficulties, our approach in Section 8 is very conservative in deciding when to issue a gold standard question.

D. EXTENSIONS

As alluded to in Section 3, in this section we briefly discuss three possible extensions to CrowdDQS. We do not present details or experimental results for these extensions.

Batching Multiple Questions within a Task. In some cases, we may wish to issue multiple questions to a worker at once. The advantage of issuing multiple questions is it allows us to get a minimum number of votes per worker, which can help us better estimate worker accuracies. Batching questions can also help reduce the commission that must be paid to AMT when issuing tasks. Batching also allows us to post tasks with higher payment rewards for workers, which may influence worker psychology to be more amenable to accepting our tasks.

Implementing batching under CrowdDQS is straightforward. Suppose we want to batch K questions together within a task. Instead of needing to select from $\binom{N}{K}$ task assignments, our real-time algorithms allow us to make K different $\binom{N}{1}$ task assignments. When a worker requests a task from our server, we still use our algorithms to display only one question. However, when the user submits this question, our real-time algorithms can immediately display a 2nd question to the worker instead of concluding the task. Only once the K-th question has been submitted to our server does CrowdDQS need to conclude the task.

Machine-based Priors. We considered a crowd-only solution to answering the questions in Q in this paper. However, there are 2 ways we can incorporate machine-based priors into our system. If we have a model that is calibrated to output probabilities, then we can use these priors directly into our worker accuracy estimation and answer aggregation strategies as alluded to in Section 5. If we do not have a calibrated model, we can instead treat the machine-based labels as though they all come from a single worker. CrowdDQS would then use future crowd votes to learn the accuracy of the machine classifier. This learned accuracy can then be used to effectively integrate machine-based votes with crowdsourced votes.

Explicitly Modeling Question Difficulty. In our model for aggregating worker responses and estimating worker accuracies, our equations assume each question is of equal difficulty. It could also be possible for us to try to infer each question's difficulty by fitting a "question difficulty" parameter $D \in [0, 1]$ to each question that modulates each worker's accuracy for that question to be between $\frac{1}{c_i}$ (purely random) for high values of D to $\hat{p}_j$ for low values of D. We can then compute a maximum likelihood estimate for D for each question. However, we suspect this would only slightly improve our system. Even though we do not model question difficulty explicitly, our question selection implicitly targets votes on questions with high disagreement rates.