Counting with the Crowd

Adam Marcus David Karger Samuel Madden Robert Miller Sewoong Oh
MIT CSAIL
{marcua,karger,madden,rcm}@csail.mit.edu, swoh@illinois.edu

ABSTRACT

In this paper, we address the problem of selectivity estimation in a crowdsourced database. Specifically, we develop several techniques for using workers on a crowdsourcing platform like Amazon's Mechanical Turk to estimate the fraction of items in a dataset (e.g., a collection of photos) that satisfy some property or predicate (e.g., photos of trees). We do this without explicitly iterating through every item in the dataset. This is important in crowdsourced query optimization to support predicate ordering and in query evaluation, when performing a GROUP BY operation with a COUNT or AVG aggregate. We compare sampling item labels, a traditional approach, to showing workers a collection of items and asking them to estimate how many satisfy some predicate. Additionally, we develop techniques to eliminate spammers and colluding attackers trying to skew selectivity estimates when using this count estimation approach. We find that for images, counting can be much more effective than sampled labeling, reducing the amount of work necessary to arrive at an estimate that is within 1% of the true fraction by up to an order of magnitude, with lower worker latency. We also find that sampled labeling outperforms count estimation on a text processing task, presumably because people are better at quickly processing large batches of images than they are at reading strings of text. Our spammer detection technique, which is applicable to both the label- and count-based approaches, can improve accuracy by up to two orders of magnitude.

1. INTRODUCTION

Crowdsourcing platforms such as Amazon's Mechanical Turk (MTurk) (`mturk.com`) allow users to generate many small tasks for crowd workers (called Turkers) to complete in exchange for small amounts of money per task. For example, someone wishing to screen millions of images for offensive content might put each image on MTurk as a Human Intelligence Task (HIT), offering 1 cent for each image labeled "offensive" or "inoffensive." Platforms such as MTurk provide programmatic access to create HITs, and expose them to more than 100,000 crowd workers. Crowdsourcing applications range from human-assisted form processing (`captricity.com`) to human-powered grammar correction [4].

Proceedings of the VLDB Endowment, Vol. 6, No. 2

Building robust crowdsourced workflows with reliable, error-free answers is not easy. One has to consider how to design the user interface (an HTML form) the crowd worker sees, the price to pay for each task, how to weed out sloppy or intentionally incorrect answers, and how to deal with latency on the order of minutes to hours as workers generate responses. Several startups, such as CrowdFlower (`crowdflower.com`) and MobileWorks (`www.mobileworks.com`) aim to make crowdsourced workflow development easier by offering simplified and task-specific APIs.

The database community has made several contributions to the crowdsourcing space. Systems such as CrowdDB [10], Deco [19], Jabberwocky [1], and our own Qurk [16] make it simpler to write declarative queries that filter, join, and aggregate data. The majority of the research generated by the community has focused on building operators that are human-aware to perform tasks like finding the best photo of a restaurant (a human-powered MAX operator) [12], or performing entity resolution across images or text (a human-powered join) [17]. One important but untouched area of research is in building crowd-powered optimizers.

Our previous work looked at the problem of estimating the cost of sort and join operators in crowd databases [17]. Given these costs, one fundamental problem that remains to be addressed to allow optimization in a crowd-powered database is that of *selectivity estimation*. In selectivity estimation, we are given an expression or predicate, and we must estimate the number of results that will satisfy the expression. Given cost estimates for individual operators, input cardinalities, and selectivity estimates, standard cost-based optimization techniques can be used to estimate overall query cost.

In this paper, we study how to estimate the selectivity of a predicate with help from the crowd. Consider a crowdsourced workflow that filters photos of people to those of males with red hair. Crowd workers are shown pictures of people and provide either the gender or hair color they see. Suppose we could estimate that red hair is prevalent in only 2% of the photos, and that males constitute 50% of the photos. We could order the tasks to ask about red hair first and perform fewer HITs overall. Whereas traditional selectivity estimation saves database users time, optimizing operator ordering can save users money by reducing the number of HITs.

In addition to being used inside optimizers to estimate intermediate result size and cost, selectivity estimators can be used to estimate answers to COUNT, SUM, and AVERAGE aggregate queries with GROUP BY clauses. In a crowd context, such answers are fundamentally approximate, since humans may disagree on answers. For example, say we had a corpus of tweets and wanted to perform sentiment analysis on those tweets, identifying how many positive, negative, and neutral tweets were in our dataset. We could ask our database to provide us with a count of all tweets grouped by their crowd-identified sentiment.

The simplest way to perform selectivity estimation and count-/sum-based aggregation would be to iterate over every item in the database and ask the crowd to determine its label. Unfortunately, this requires work proportional to the number of items in the database, which could be prohibitive for large databases. A more efficient way to estimate selectivity used in traditional optimizers is to sample a subset of the rows. This concept naturally translates to the crowd: in our example, we can generate HITs on a subset of photos, asking workers to label either the gender or hair color. With enough samples, we could estimate the popularity of males or redheads in our dataset. This approach works, but does not take advantage of humans' natural ability to batch process multiple elements, especially for heavily visual items like images [26]. Our hypothesis is that people can estimate the frequency of objects' properties in a batch without having to explicitly label each item.

This observation leads to the first key contribution of this paper: we employ an interface and estimation algorithm that takes advantage of humans' batch processing capabilities. Instead of showing an image at a time, we can show 100 images (or 100 sentences) to a crowd worker, and ask them to tell us approximately how many people in the images are red-headed (or how many sentences have positive sentiment). By aggregating across several batches and multiple crowd workers, we can converge on the true fraction of each property. This "wisdom of the crowds" effect, where an individual may not accurately estimate an aggregate quantity, but the average of a number of individual estimates does approximate the quantity, has been well documented [20]. We show that this approach allows us to estimate aggregates across several image-based datasets using about an order of magnitude fewer HITs than sampling-based approaches with comparable accuracy. On textual datasets, the same effect does not apply: item labeling works better than batch estimation of an aggregate property of a collection of short sentences (Tweets, in our experiment.)

The fast convergence of our count-based approach is not without challenges. Workers are free to provide us with any estimate they wish, and we must design algorithms to detect and filter out "bad" workers such as spammers who answer quickly to receive payment without working in earnest. The algorithm we designed filters out spammers in both the count- and label-based interfaces by removing workers whose answer distribution does not match other workers'. Unlike previous techniques that require redundant answers to each label from multiple workers, here we ask each worker to process different random subsets of a dataset. We also identify a solution to the problem of a coordinated attack by multiple workers, or sybil attacks [7] from a single worker with multiple identities.

In summary, in this paper, our contributions are:

1. An interface and technique to estimate selectivity for predicates and GROUP BY expressions over categorical data. Specifically, for a dataset (e.g., a collection of images) the algorithm approximates the distribution of some property (e.g., gender) in the dataset. This has applications to query optimization in selectivity estimation, and to SELECT-COUNT-GROUP BY queries. On image-based datasets, our approach converges on a correct response with high confidence up to an order of magnitude faster than traditional sampling techniques. The counting interface also requires less time for workers to provide estimates than the labeling interface.
2. A method for identifying low-quality or spam responses to our batched interface. Prior work estimates worker quality by asking multiple crowd workers to label the same data. We instead have workers provide us with non-overlapping estimates of the number of items with a property in a given batch. Workers whose answers are consistently near the global worker mean are judged to provide quality work, while workers who consistently stray from the mean are judged to be low-quality. This approach improves our accuracy on real estimation problems by up to two orders of magnitude.
3. A technique to identify and avoid coordinated attacks from multiple workers, or one worker with multiple identities. If multiple workers agree to provide the same estimate (e.g., "let's report that any collection of 100 photos we see contains 80 males"), our spammer detection technique may be thrown off from the true value. To avoid this attack, we insert a random amount of verified gold standard data into each HIT, and show that this technique can weaken the strength of an attack in proportion to the amount of gold standard data used.

We show through experiments that our approaches generalize to several domains, including image estimation and text classification.

2. MOTIVATING EXAMPLES

In this section, we provide three example use-cases that use crowd-based counting and selectivity estimation. While our examples utilize features of our Qurk [17] crowd-powered workflow system, these optimization opportunities are available in all such systems.

2.1 Filtering Photos

Consider a table `photos(id, name, picture)` that contains names and references to pictures of people. Suppose we want to identify photos of red-headed males. In Qurk, we would write:

```
SELECT id, name
FROM photos
WHERE gender(picture) = 'male'
  AND hairColor(picture) = 'red';
```

In this case, `gender` and `hairColor` are crowd-powered user-defined functions (UDFs) that specify templates for HTML forms that crowd workers fill out. The `gender` UDF has the following definition in Qurk:

```
TASK gender(field) TYPE Generative:
    ItemPrompt: "<table><tr> \
                <td><img src='%s'> \
                <td>What is the gender of this person? \
              </table>", tuple[field]
    Response: Choice("Gender", ["male","female"])
    BatchPrompt: "There are %d people below.  \
                  Please identify the gender \
                  of each.", BATCHSIZE
    Combiner: MajorityVote
```

This UDF instantiates an `ItemPrompt` per tuple, with a choice/radio button that allows the worker to select *male* or *female* for that tuple. It is common for workers to answer multiple such `ItemPrompts` in a single HIT, and in such situations, the generated HIT is preceeded with a `BatchPrompt`, letting the worker know how many image labels are ahead of them. Qurk identifies the optimal batch size automatically. Multiple worker responses may be required to avoid a single incorrect worker from providing a wrong answer, and the `MajorityVote` combiner takes the most popular worker response per tuple. For details on the query and UDF format in Qurk, see [17].

As we describe below, a good crowd-powered optimizer will identify `gender(picture) = 'male'` as filtering out less records than `hairColor(picture) = 'red'` and first filter all tuples based on hair color to reduce the total number of HITs.

2.2 Counting Image Properties

Our second example involves grouping and aggregating data. Imagine a table `shapes(id, picture)` that contains several pictures of shapes that vary in fill color and shape. If one wanted to generate an interface to navigate this collection of images, it would help to summarize all colors of images and their frequency in the dataset, as in the following query:

```
SELECT fillColor(picture), COUNT(*)
FROM shapes
GROUP BY fillColor(picture);
```

This query would also provide a histogram of all image colors in the `shapes` table. Here, `fillColor` is a crowd-based UDF that asks a worker to specify the color of a shape from a drop-down list of possible colors. These colors would be predefined, or a crowd-based technique for listing potential colors [22, 3] could be used.

2.3 Coding Tweet Text

Our final example shows how our approaches apply to datatypes beyond images. It is common in social science applications to "code" datasets of user data. For example, in a study of Twitter usage habits by André et al. [2], the authors had crowd workers categorize tweets into categories such as "Question to Followers," or "Information Sharing." Given a table of tweet content such as `tweets(authorid, time, text)`, one might have crowd workers code those tweets in the following way:

```
SELECT category(text), COUNT(*)
FROM tweets
GROUP BY category(text);
```

Here `category` is a crowd-based UDF that presents workers with a form asking them to code each tweet. Having a fast way to provide a histogram over such textual data would be valuable to social scientists and other data analysts.

3. COUNTING APPROACH

Our fundamental problem comes down to estimating the number of items in a dataset that satisfy a predicate or belong to a group. These counts can be used to answer aggregate queries or estimate selectivities. We explore two methods for counting: a label-based approach and a count-based approach. The label-based approach is based on traditional sampling theory. We sample tuples and ask the crowd to label the category assigned to each tuple (e.g., whether a photo is of a male or a female) until we achieve a desired confidence interval around the frequency of each category. The count-based approach displays a collection of items to a worker and asks them to approximate how many of the items fall into a particular category (e.g., the number of images with males displayed).

Both approaches result in estimates of the true frequency of each category that, in the absence of faulty worker responses, converge on the true frequency. Because in practice all crowdsourced worker output has some uncertainty, all crowd-powered techniques result in approximate answers. We study the convergence rate and accuracy of various approaches in Section 6.

We assume that categories are known ahead of time (e.g., we know the domain of the GROUP BY attributes). One interesting line of research lies in how we can determine all of the distinct categories covered by our estimation technique. Various projects explore how this can be done through crowdsourcing [3, 22].

3.1 User Interfaces for Count Estimation

Crowd worker user interface design, or the specification of tasks and user interfaces that workers see when they perform a HIT, is an important component in acheiving good result quality. Asking questions in a way that does not bias workers to provide incorrect answers, and providing interfaces that make it as hard to provide an incorrect answer as it is to provide a correct one is crucial.

After some iterative design, we generated two interfaces that correspond to the two approaches above: a label-based interface prompts workers to provide a label for each item displayed, and a count-based interface shows workers a collection of items and asks them for an approximate count of items with a given property.

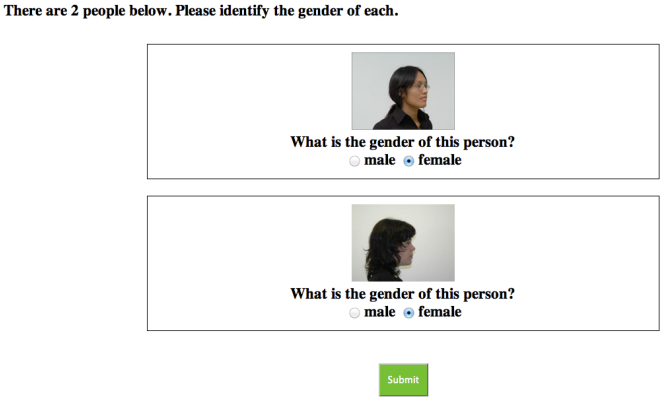

Figure 1: The label-based interface asks workers to label each item explicitly.

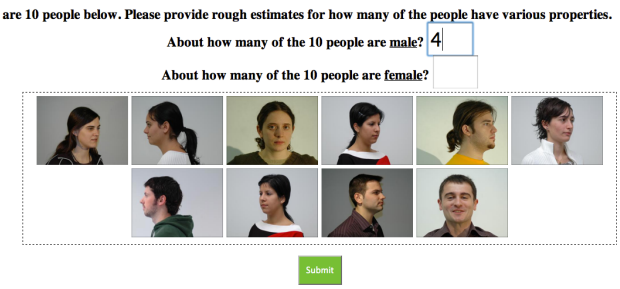

Figure 2: The count-based interface asks workers to estimate the number of items with a property.

Label-based interfaces are common on MTurk, as image labeling is a popular task. Our label-based interface is seen in Figure 1. We show several items, in this case images of people, in a single HIT. A prompt at the top of the HIT asks workers to select the best label for each image. Following each image, we see two radio buttons that the worker toggles between to identify the gender of the person in the image. After going through several images (in this case, the batch size of images per HIT is 2), the worker clicks the Submit button. We error-check the submission to ensure all of the radio button pairs have a selection, and upon successful submission of the HIT, offer the worker another set of 2 images to label.

Count-based interfaces are less common on MTurk, so this interface required more design. The interface we used for count-based experiments is in Figure 2. A prompt at the top of the HIT asks workers to identify how many images have a particular property. Below that general instruction, workers are prompted to enter the number of males and females in the collection of images below. To estimate the frequency of two categories such as *male* and *female*, we only need to ask about the number of items in one of those categories. We included both questions in the screenshot to illustrate the appearance of a multi-class interface. Fixed-width images are displayed below the questions in tabular format. With more images (e.g., 100) per page, the worker may have to scroll to reach the submit button. When a worker pushes submit, the interface alerts them if they have not filled in all of the requested counts, and performs bounds-checking on the inputs to ensure they are non-negative and add up to less than or equal the total number of images displayed.

Our count-based interface design has several alternatives. One interface would resize images so that, as more images are displayed on a page, the thumbnails shrink and no scrolling is required. We avoided this design so that we could study batch size (the number of images on a page) independent of task difficulty (identifying

properties of images as they shrink). Another design choice involves where to place the prompts (i.e., the questions asking how many males are displayed). We found that placing prompts at the top of the page made it easier for first-time workers to learn the task without scrolling to the bottom of pages with large batch sizes. Scrolling to the top of the page to fill in an answer after counting items did not seem to disturb veteran workers. Finally, there are several options for the wording of the prompts. We can ask workers to approximate the number of males, or prompt them for a precise measurement, and study the effect on result accuracy. We prompted workers to tell us "About how many" items have a given property, but allow them to be as precise as they desire. We leave a study of the effect of wording on performance for future work.

As we will show in Section 6, the count-based interface allows a batch size of about an order of magnitude more images than the label-based interface. Given the lack of prior work on count-based interfaces, it is likely that further user studies and design iterations could lead to more effective count-based interfaces. It is less likely that label-based interfaces, which are so popular and well-researched on MTurk, will see much improvement.

3.2 Modifying UDFs to Support Counting

The label-based approach to estimating the fraction of items with a property simply requires sampling the tuples in a table and applying a labeling UDF. Because we can apply the UDF directly, the user interface for the label-based approach can be derived directly from the UDF definition used in Qurk with no additional information. In the case of the count-based approach, however, new prompts are necessary to give the worker instructions on how to answer each query.

As an example, we modify the `gender` UDF from Section 2.1 to include prompts for counting the males and females in the dataset. Below is the original `gender` defintion with two additions in bold:

```
TASK gender(field) TYPE Generative:
    ItemPrompt: "<table><tr> \
              <td><img src='%s'> \
              <td>What is the gender of this person? \
            </table>", tuple[field]
    PropertyPrompt: "About how many of the %d \
       people are %s?", BATCHSIZE, PROPERTY
    Response: Choice("Gender", ["male","female"])
    BatchPrompt: "There are %d people below.  \
                  Please identify the gender \
                  of each.", BATCHSIZE
    CountPrompt: "There are %d people below. \
       Please provide rough estimates for how \
       many of the people have various \
       properties.", BATCHSIZE
    Combiner: MajorityVote
```

The first addition is `PropertyPrompt`, that specifies how to ask about each individual property. We also add `CountPrompt`, that specifies the overall instructions for the entire HIT. With these modifications, the Qurk optimizer can now prompt workers with the count-based interface for selectivity estimation and SELECT-COUNT-GROUP BY queries.

3.3 Estimating Overall Counts with Workers

The "wisdom of the crowds" [20] says one can get a reasonable estimate for the number of jelly beans in a jar by asking many people to estimate the number of beans in the jar and taking the average of their responses. Our problem is not quite this simple, because the population of items is large and the individual items are not compact enough to show every person all of the items at the same time. Instead, we show each worker a random sample of the items in our collection, and ask them how many items have some property. For example, we can take a sample of 100 images from a collection of several million, and ask a worker to approximate how many of the people in the images are male. There are three sources of error in measuring several of these samples across multiple workers:

Sampling error. In traditional sampling theory, we see errors in approximations derived from samples due to variations in the individual subsets that are selected.

Worker error. Workers provide incorrect responses due to human error, or because entering nonsensical or less-than-acurate results allows them to complete more tasks for money.

Dependent samples. Platforms such as MTurk allow workers to perform more than one of the available HITs, meaning that multiple responses can be received from each worker, and often spammers are the workers who complete the most tasks.

By the central limit theorem [9], taking the average of all worker estimates of the fraction of items with a given property reduces the sampling error. The average of N such samples has a standard deviation proportional to $\frac{1}{\sqrt{N}}$, implying that averaging more samples probabilistically increases accuracy. While the sampling error can be reduced by averaging, worker error and dependent samples make taking an average inaccurate. Furthermore, it is difficult to model error bounds for these sources of error because worker behavior is unpredictable.

Limiting each worker to a single response would reduce serious error due to spammers, but goes against the design of platforms such as MTurk, in which the majority of the work is often done by a small group of workers [10]. The majority of workers provide accurate results, and one should allow these workers the opportunity to do as much work as possible. Still, there is a danger in the spammer who sometimes provides the largest number of responses due to the speed at which they can generate inaccurate results. It is better to design algorithms that detect, discount, and block future results from these workers while letting other workers productively produce results. We explore a solution to this problem in Section 4.

3.4 Comparing the Approaches

The label- and count-based approaches see different rates of worker error, as they provide different interfaces and prompts to generate their estimates. They also incur different sampling errors because of the differing amount of items that workers are willing to process for a price in the different interfaces. We will study worker error in the experiments, and for now focus on how each approach results in different amounts of items sampled in a given number of HITs.

The approach to getting an item's label (e.g., identifying a picture of a flower as being *blue*) through multiple crowd workers is well-studied [14, 15]. At a high level, we retrieve R redundant labels for each item, usually around 3–7. The redundancy allows us to run majority vote-like algorithms amongst the multiple worker responses to boost the likelihood of a correct item label. A label-based interface supports batching B_L item labels per HIT, usually in the range of 5–20. For example, to label 500 pictures as male or female, we may display 10 images per HIT, and ask 5 workers to provide labels for each image. This would require $\frac{5*500}{10} = 250$ HIT assignments to complete the labeling task. More generally, with a budget of H HITs, each HIT containing B_L items to be labeled, and R redundant labels per HIT, the number of items we can label (N_L) is $N_L = B_L \left\lfloor \frac{H}{R} \right\rfloor$.

In the count-based approach, we are not labeling particular items precisely, and our spammer detection technique does not require redundant responses. We can batch B_C items per HIT, usually in the range of 50–150, since workers are performing less work per item labeled (they are counting or estimating images in chunks rather than explicitly working to label each image). In our image labeling example with 500 images of people and a batch rate of 75, we examine all images in 7 HITs, rather than the 250 it took to label

images precisely. More generally, with a budget of H HITs and each HIT containing B_C items to be counted, the number of items we can count (N_C) is $N_C = B_C H$.

Putting aside differences in worker error in each approach, we can compare how many items can be counted or labeled in our approach given a budget of H HITs. Taking the ratio of items counted to items labeled and assuming H is a multiple of R, we get

$$\frac{N_C}{N_L} = \frac{B_C R}{B_L}.$$

In our running example ($B_L = 10$, $R = 5$, $B_C = 75$), we end up with workers reviewing 37.5 times as many items counted as labeled for a given budget.

3.5 Estimating Confidence Intervals

In both the label- and count-based approach, a user wants to know that the estimate provided to them is within a certain margin of error from the true distribution of item properties. For example, in selectivity estimation, a user might be comfortable with a relatively wide margin of error between two filters, say 10%, as small errors will not harm query processing or cost too much. When performing an aggregate query, however, the user might prefer to attain high confidence in their results, demanding the confidence of the gender split in their dataset to be near 1%.

In the label-based approach, we can consider each item's label a Bernoulli random variable (a categorical random variable in the case of more than two property values) with some probability of each property value occurring. For example, when estimating the fraction of male photos in a collection after labeling around 100, the gender displayed in any photo is modeled as a Bernoulli variable with probability of being male p_{male}. If we have labeled 40 photos as male, we can estimate p_{male} as $\frac{40}{100} = 0.4$. Using sampling theory, we can either utilize the normal approximation of the confidence interval for binomial distributions or the more accurate Wilson score [24] to determine, with high probability, how close the actual answer is to the estimate.

The count-based approach works similarly. If we display B_C items (e.g., photos) per HIT and have workers provide us with a count (e.g., the number of males), we can model this count as a binomial random variable C that is distributed as $Binomial(B_C, p_{male})$. As we collect counts C_i from various HITs, the average of the fractions $\frac{C_i}{B_C}$ will approach p_{male} and we can again employ the Wilson score or normal approximation of the confidence interval.

In practice, some of the responses will come from spammers, and assuming we can detect them using techniques in Section 4, we can calculate a confidence interval on the remaining values. Additionally, instead of calculating confidence intervals, one might employ resampling techniques such as the bootstrap [8] to estimate these values on smaller sample sizes.

4. IDENTIFYING SPAMMERS

We now discuss how to identify spammers using count-based input from crowd workers. Spammer identification for label-based approaches is well-studied [14, 15], and so we focus on count-based spammer identification.

Unlike the label-based approach, the count-based one does not require redundant responses (each worker sees a random sample of the items in our dataset), and the responses are continuous/ordinal (workers provide us with sample property estimates which map to fractions in the range $[0, 1]$). While the spam detection techniques we discuss below apply to our estimation problem, it is more generally applicable for any worker response for data which is sampled from a continuous distributions concentrated around some mean (e.g., counts, movie ratings, best-frame selection [5]).

A useful side-effect of our algorithm working on non-redundant worker output is that, while we designed it for count-based inputs, it also works for label-based inputs where no redundant labels are collected. If a worker labels five sampled images as male and 20 as female, we can feed these numbers into the spammer detection algorithm below to determine how likely that worker is to be a spammer. We will show in Section 6 that collecting redundant labels requires excessive amount of worker resources for the label-based approach, so the applicability of our spammer detection technique to non-redundant label-based data is also a benefit of the approach.

We will first define some variables and terms used in our algorithmic descriptions. With these definitions, and some insights we got while engineering an effective algorithm, we will conclude with the algorithm that we will show works well in Section 6.

4.1 Definitions

Say we have N workers T_1, ..., T_N. Each worker T_i completes H_i HITs, and for each HIT provides us with counts C_{i1}, ..., C_{iH_i} of the number of items in that HIT with a particular property. If there are B_C randomly sampled items in each HIT, we can convert C_{ij} (count from HIT j reported by worker i) into a fraction $F_{ij} = \frac{C_{ij}}{B_C}$.

We wish to calculate $\hat{F}$, an approximation of F, the overall fraction of items with a given property in our population. If we know there are no spammers, we could simply average the worker responses to get $\hat{F} = \frac{\sum_{i,j} F_{ij}}{\sum_{i,j} 1}$. However, given the presence of spammers, we would like to discount workers with bad results. We will accomplish this by identifying workers as potential spammers and assigning each worker a weight $\theta_i \in [0, 1]$, with 0 representing a spammer and 1 representing a worker with high quality.

4.2 Spammer Detection Algorithm

Given these definitions, we develop an algorithm for identifying spammers. We estimate θ_i for all workers T_i. Our approach assumes that the majority of workers are not spammers, and that most workers will be able to reasonably estimate the correct answer to HITs. This means a spammer is a worker whose answers deviate substantially from the other workers' distribution of answers, and in particular from the mean of all responses. We propose an iterative algorithm that estimates the mean, uses that estimated mean to identify spammers, and then repeats, re-estimating the mean and finding spammers until a fixed point is reached.

Because spammers often answer many questions (often they are trying to maximize profit by doing many HITs cheaply), we limit the contribution of an individual worker to the overall estimate by first averaging that worker's estimate across all of the HITs they have submitted by calculating $F_i = \frac{\sum_j F_{ij}}{H_i}$.

We can now compute our first estimate of $\hat{F}$ as $\hat{F}_{\text{initial}} = \frac{\sum_i F_i}{N}$.

Here $\hat{F}$ is our current estimate, where each workers' contributions are limited by averaging over all of their answers. To summarize a worker's contribution, we also tried to use a worker's first response or to pick a random worker response, but we have found these approaches to be more fickle when a worker makes a single mistake that is not representative of their overall contribution.

We then compute θ_i, the weight of each worker, by computing the bias of their average response F_i relative to the current global estimate $\hat{F}$. We also threshold, disallowing workers who are too far from the mean estimate. More precisely, in each iteration:

$$\theta_i = \begin{cases} 1 - |F_i - \hat{F}|, & \text{if } |F_i - \hat{F}| < \lambda \\ 0, & \text{otherwise} \end{cases}.$$

Setting $\theta_i = 1 - |F_i - \hat{F}|$ when $|F_i - \hat{F}| < \lambda$ assumes an absolute loss function. In our experiments using a squared loss function did

not noticeably change results. For θ_i values to be accurate, we must calculate the global mean estimate $\hat{F}$ iteratively after every recalculation of θ values, as above, weighting by the θ values:

$$\hat{F} = \frac{\sum_i \theta_i \hat{F}_i}{\sum_i \theta_i}.$$

In our experiments in Section 6, we show that setting $\lambda = 0.14$ (approximately two standard deviations from the mean) tends to remove outlier workers while still including workers with moderate bias. This technique can be thought of as iterative outlier detection: workers whose performance is outside of a 95% confidence interval (two standard deviations) are removed. Our use of a λ cutoff at which θ_i becomes 0 comes from the literature on statistics that are robust to outliers, in particular trimmed means [23]. The biggest diversion from the traditional trimmed mean approach was to trim outlier workers rather than individual worker responses, and this observation gave us the biggest estimator accuracy improvement.

The algorithm iterates between estimating $\hat{F}$ and recalculating θ_i's until convergence. Once we have identified spammers, we no longer need to protect against a disproportionate number of responses from a single spammer, so we calculate our final $\hat{F}$ based on individual worker responses:

$$\hat{F}_{\text{final}} = \frac{\sum_{i,j} \theta_i F_{ij}}{\sum_{i,j} \theta_i}.$$

5. AVOIDING COORDINATED ATTACKS

In the previous section, we presented an algorithm that estimates the fraction of items with a given property (F) by preventing any one worker from skewing the estimate $\hat{F}$ toward their own by quickly responding to many HITs. This is because we calculate $\hat{F}$ as the average of each worker's average response, rather than the average of all worker responses. Each worker only contributes a single value toward the average regardless of how many HITs they performed.

The algorithm described in Section 4 is still vulnerable to a coordinated attack from multiple workers, however. While we have not seen a coordinated attack in our experiments, there are reports of sophisticated Turkers who deploy multiple accounts to subvert worker quality detection algorithms [13]. If several workers, or a single worker with multiple automated accounts, all submit estimates F_{ij} such that several values of F_i are similar to each other but far from the true mean, they will be able to skew their worker quality estimates in their own favor so that they are not blacklisted, or receive bonuses or additional pay for doing good work.

Note that an attack in which workers agree to submit random noise F_{ij} estimates is less effective than one in which workers coordinate on a single attack value. This is because the average-based algorithm in Section 4 is robust to random noise. Even an attack that is not random but has workers provide different values from one-another would result in less of an effect, as the algorithm would iteratively remove the most extreme worker F_i values.

Susceptibility to coordinated attacks is not limited to our algorithm. Any crowd-powered approach that rewards workers in the same range as other workers is susceptible. One such example is an approach by Bernstein et al. [5] to identify the best video frame by having crowd workers agree on the best scenes in the video.

A simple way to avoid this is to collect "gold standard" data with known labels on several items $\{G_1, ..., G_M\}$ [11]. One can then generate HITs over that gold standard data for workers to complete to ensure that they accurately count data from a known distribution. We will see in Section 6 that in a few HITs we can sample several thousand items, and so collecting a few tens or hundreds of gold standard items is not too resource-intensive.

The simple gold standard approach has several issues. First, sophisticated workers could learn to identify the page with the gold standard data and the correct answer to the estimates for that page. Second, because the distribution of the number of tasks completed by each worker is zipfian [10], many workers will only complete one or two tasks. This limitation means that the gold standard task would constitute the bulk of the contribution of many workers, which is not ideal.

We propose a different use of the gold standard data. Rather than generate an entire gold standard task, we randomly distribute the gold standard items throughout each task. To do this, we first vary the number of gold standard items with a given label in each task a worker completes. For example, if a worker is estimating the number of men and women in a task with 100 images, we will randomly place between 1 and 20 gold standard images with known labels into the 100. When a worker tells us the number of items with a property in the task they just completed (say, 25 males in 100 images), we subtract the number of gold standard images with that property (e.g., 13 gold standard male images) from their count when calculating the fraction F_{ij}. We cannot just check how the worker did on the gold standard items, because we are not asking the worker to label individual items, but to estimate the overall frequency of some property.

Using this approach, there is no one repeating gold standard HIT, as some gold standard data is mixed into every HIT. As a result, attackers can not identify and correctly solve the gold standard HIT. Additionally, we can continue to measure worker quality throughout the worker's contributions.

5.1 Correcting Worker Responses

Formally, we show a worker R items at a time, where G_{ij} gold standard items are introduced into the jth HIT shown to worker i. When a worker tells us there are C_{ij} items with a given property, we subtract the G_{ij} known items when calculating the fraction:

$$F_{ij} = \frac{C_{ij} - G_{ij}}{R - G_{ij}}.$$

In this approach, since each worker sees a different fraction of gold standard data, two or more workers who are colluding to skew F to some value should no longer agree once the gold standard data is removed. If these attackers coordinate in repeating the same value C_{ij} for each task, the estimate F_{ij} will still vary per task due to the randomness in G_{ij}. The larger the variance of G_{ij} is per hit, the larger the variance will be in the corrected F_{ij}. This variance will reduce the power of disruption of a coordinated attack, and increase the probability that attackers will be removed as outliers because their answer varies from the overall mean.

In contrast, non-colluding workers who estimate the true value of F should still agree once the gold standard data is removed.

A second benefit of our approach comes in cases where the counts of items with a given property that a worker reports are smaller than the number of gold standard items with that property in a HIT ($C_{ij} < G_{ij}$). In these cases, a worker's F_{ij} estimate will be negative, which is not possible. Aside from boundary situations where some user error results in negative F_{ij} estimates that are near 0, we can discard such workers as spammers. This benefit works against individual attackers and coordinated attacks by outright removing some workers with negative fraction estimates.

The second benefit is value-dependent, as attackers who submit low count values are more likely to be identified by the approach than attackers who submit high counts. Luckily, there is symmetry to the value-dependence: assuming we can determine early which property the attackers will say is least frequent, we can increase our likelihood of identifying attackers by providing a higher proportion of gold standard examples of that type, or by providing small amounts of gold standard data of the less frequent

types. For example, if workers estimate that 90% of pictures are of males, then they are implicitly claiming that 10% are of females. A small amount of gold standard female samples would allow us to spot spammers. Identifying the best classes of gold standard data to use becomes more challenging as the number of classes increases, since one has to identify classes that are both low-frequency and that attackers are coordinating to spam. We leave these challenges to future work.

Our approach is not without drawbacks. The most direct is that utilizing more gold data on average results in less information from each worker on average. If on average we use G gold standard items per task, and have R items per HIT total, then on average we require $\frac{1}{1-\frac{G}{R}}$ more HITs to cover the same amount of work. For example, if half of the data is gold, we require twice as many tasks to be completed to cover as much real data. Another drawback is that collecting gold standard data takes time (and possibly money), which in practice limits the maximum value of G.

5.2 Random Gold Standard Selection

For our gold standard technique to be effective, we need to introduce a different fraction of gold standard data into each HIT, while keeping the average amount of gold standard data used per HIT at some level. This allows users to control how much of their budget is spent on gold standard data.

Given a user-specified average number of gold standard items μ, we intersperse $G_{ij} = \mu + \mathcal{G}$ gold standard items into each task, where $\mathcal{G}$ is a random integer chosen uniformly from the range $[-g \ldots g]$. We require that the user pick $\mu > g$ and $R - \mu > g$ to prevent G_{ij} from falling below 0 or exceeding R. In our experiments we use $g = .2R$, though even doubling this value did not meaningfully change our findings.

6. EXPERIMENTS

The goal of our experiments is to understand how the convergence rates and accuracy of count-based and label-based counting/selectivity estimation compare on these different datasets. We show that the count-based approach has similar accuracy and much faster convergence than the label-based approaches on images, but does far worse on textual data (tweets). Finally, we simulate the effect of sybil attacks, and show scenarios in which our randomized gold standard approach averts the attacks.

6.1 Datasets

We utilize three datasets in our experiments, each of which is aligned with one of the three motivating examples in Section 2.

Face Gender. The dataset we use for most of our performance evaluation is the GTAV Face Database [21]. This dataset contains 1767 photos of 44 people. For each person, there are between 34 and 65 images of that person in different poses. We manually labeled each person in the dataset as male or female, so that we could verify our estimation techniques to determine the number of images in the dataset of a given gender. Some sample images from this dataset can be seen in Figures 1 and 2.

Shapes and Colors. To test the difficulty of perceptual tasks, we generated an image dataset of shapes. Our generated dataset consists of triangles, circles, squares, and diamonds. The fill color of each shape is either yellow, orange, red, green, blue, or pink. Additionally, each shape has an outline that is one of these colors. For variety, shapes are sized to fit various proportions inside a 100x100 pixel box. An example can be seen in Figure 3. We generated 1680 such sample shapes.

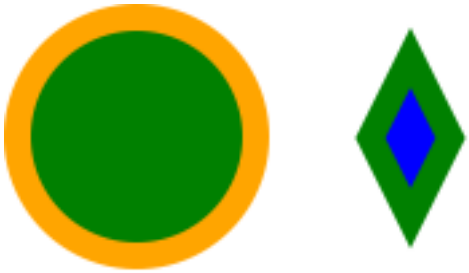

Figure 3: Example shapes in our Shapes and Colors dataset.

Tweet Categorization. To test our approaches on non-image data, we also ran our approximation techniques on tweet text. Our tweets come from an initial collection of approximately 4200 from André et al., who studied the value of tweet content to Twitter users' followers [2]. Tweets in this study were categorized using CrowdFlower into eight categories. The authors of that paper found several tweets to which it was hard to assign a single label, so we selected a subset of 2530 tweets in the three most stable categories: Information Sharing, where a user links to a piece of information (e.g., *Awesome article explaining the credit crisis:* `http://...`), Me Now, where a user says what they are doing now, (e.g., *Just finished cleaning the apartment!*), and Question to Followers, where a user poses a question (e.g., *What is your favorite headphone brand?*). We use our techniques to determine the fraction of tweets in these categories. "Coding" tasks, where a crowd is asked to categorize various items, are common in the social sciences, and we imagine our techniques may be useful in these situations.

For all datasets, since we have ground truth data on their labels, we can generate artificial datasets by sampling at different frequencies of labels. For example, we can generate a version of the face dataset with 10% males or 50% males by sampling more heavily from a subset of the data. For the faces and shapes dataset, all samples are of size 1000, meaning that regardless of the property distribution, we always select 1000 images on which to estimate properties. On the tweet dataset, our samples are of size 2500 tweets.

6.2 Estimating Counts

To quantify the error of our different algorithms, we tested several estimations on MTurk, varying parameters. For each parameter setting, we generated 1000 HITs. We tried the label-based and count-based worker interfaces, as shown in Figures 1 and 2. When counting, we used per-HIT batch sizes of: 5, 10, 25, 50, 75, 100, 125, and 150. For labeling, we used batch sizes of: 5, 10, 15, and 20. In label-based scenarios, we collected redundant labels (generating 200 HITs, with 5 workers completing each HIT) and no-redundancy labels (generating 1000 HITs, with 1 worker completing each HIT). We use Ipeirotis et al.'s approach [14] to combine redundant answers. We ran tests using the Faces, Shapes, and Tweet datasets. To understand the effect of varying selectivity, we sampled the Faces dataset such that it had 1%, 10%, 25%, 50%, 75%, 90%, and 99% male faces. To ensure repeatability, we ran each experiment at least twice during business hours in the Eastern US time zone. For all experiments, we paid workers $0.01 per HIT, avoiding higher pay increments because batching more than one item per HIT was possible even at this pay increment.

Overall Error. We start by studying the errors of various approaches on the Faces dataset. Figure 4 shows the error rates of the label- and count-based approaches. For the label-based approach, we report the errors with redundant labels (*LabelR*) and without (*LabelNoR*). For the count-based approach, we feed worker values into the thresholding-based spammer-detection algorithm described

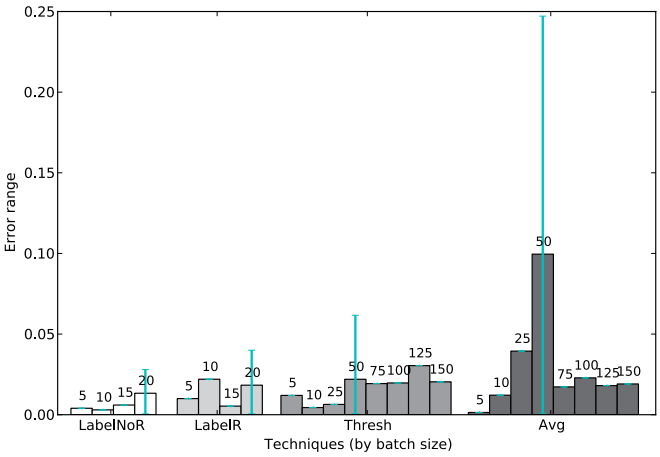

Figure 4: Chart of average error by approach, with error bars for min/max values. Numbers above each bar are batch size. *LabelNoR*: labels, no redundancy. *LabelR*: labels, 5-worker redundancy. *Thresh*: counts, spam detection. *Avg*: counts, no spam detection. *Thresh* and *Avg* have the same inputs.

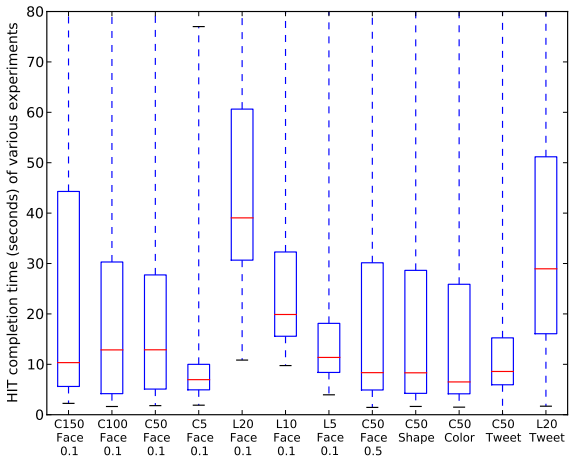

Figure 5: Box-and-whisker plots of the number of seconds workers took to complete tasks. The top of each box is the 75th percentile completion time and the bottom is the 25th percentile. Red lines indicate medians. Whiskers (error bars) represent minimum (bottom) and maximum (top) values.

in Section 4 (*Thresh*) and into a simple average of all worker-reported counts (*Avg*). We also tried the median and the average of the middle 5% and 10% of answers (not shown), but they were as vulnerable to spammers as the average. The bar charts displayed in this figure represent the average error across the experiments we ran, with the error bars representing the minimum and maximum error values. Error is calculated as the absolute difference between the estimated fraction and the ground-truth result of each experiment using all 1000 HITs in each experiment. We break the results down by the batch size of the various approaches.

We see that when using labeling, getting multiple labels per item does not appear to reduce the overall error. That is, we do not have evidence of heavy spamming of these interfaces, and we do have evidence that across several experiments, non-redundant labels have commensurate error rates to redundant labels. Because user error is low, sampling error dominates the error term in the labeling-based approach, and the $5\times$ increase in non-redundantly sampled items in *LabelNoR* reduces the overall error. For the count-based approaches, we see that taking a simple average across all worker-provided counts generates error rates up to 25%. Feeding that same count data into the spam-detecting thresholded approach generates error rates on par with the label-based approaches. In terms of error rates on 1000 HITs, it is possible to achieve similar results using label- and count-based approaches, although the count-based approach requires spam detection. The same count-based spam detection approach could be used on the non-redundant label-based approach should spammers strike.

Interface Latency. While the largest batch size we tried in the count-based interface was 150, we limited label-based interfaces to batching at most 20 items per HIT. Anecdotally, even at a batch size of 20, we would sometimes run into situations where workers would take a long time to pick up our tasks because they were bordering on undesirable. More importantly, as evidenced by Figure 5, workers took longer to label 20 images than they did to report counts on 150. In this figure, we show different experiments along the X-axis. Along the Y-axis, we display a box-and-whiskers plot of the number of seconds to complete each HIT. For each experiment, the edges of the box represent the 25th (bottom) and 75th (top) percentile, while the red line inside each box represents the median seconds to complete a HIT. The whiskers, or ends of each dotted line, represent the minimum and maximum seconds to complete a HIT in that experiment. Maximums are often not shown, because of pathological cases where workers would pick up a HIT but not submit it until a maximum of 5 minutes was up.

The different experiments are labeled on the X-axis. Experiments labeled C150, ..., C5 are count-based with batch size 150, ..., 5. Experiments labeled L20, ..., L5 are label-based with batch size 20, ..., 5. In this section, we focus only on experiments for Face 0.1 (Faces with males representing 10% of the dataset); the other experiments are described below.

First, we see that the count-based batch 150 approach takes workers less time, in general, than the label-based batch 20 approach. Decreasing batch size generally reduces the time it takes workers to complete each task. For any equivalent batch size, the count-based approach takes less time than the label-based one. The fact that we paid $0.01 per HIT regardless of batch size suggests MTurk is not a liquid market for tasks. Because label-based batches larger than 20 were not guaranteed to be picked up by workers in reasonable time, and because workers were spending longer on these tasks than on batch 150 count-based tasks, we decided to limit our batching at these values. The fact that counting is so much faster than labeling for images suggests that people are quickly scanning and estimating the quantities the HITs ask about, rather than manually iterating through each item and explicitly counting frequencies.

Another factor to consider is the end-to-end query time, or how long it takes to complete these HITs from start to finish. We omit a chart due to space limitations, but completing 1000 HITs in the count-based interface with batch size 150 takes about as long as 1000 HITs in the label-based interface with batch size 20 (30-40 minutes). For smaller batch sizes, 1000 HITs of counting 25 takes about as long as labeling 5 (15 minutes). Even though label-based tasks take workers longer to complete per task, we hypothesize that they are more recognizable to workers and see higher initial participation rates until enough workers warm up to the idea of trying a less familiar count-based interface. In our experience, these end-to-end times go down as an employer gets a reputation for reliably paying for work and for providing many similar work units.

Spam Detection Example. In Figure 6, we show an example where our spam detection algorithm reduces error. Here, the two

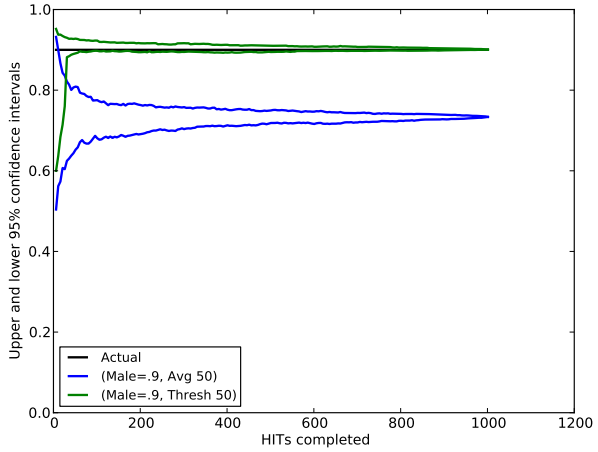

Figure 6: Given input data laden with spammer-provided values, the spammer-detection algorithm (*Thresh*) arrives at the correct estimate while a simple average of inputs (*Avg*) does not.

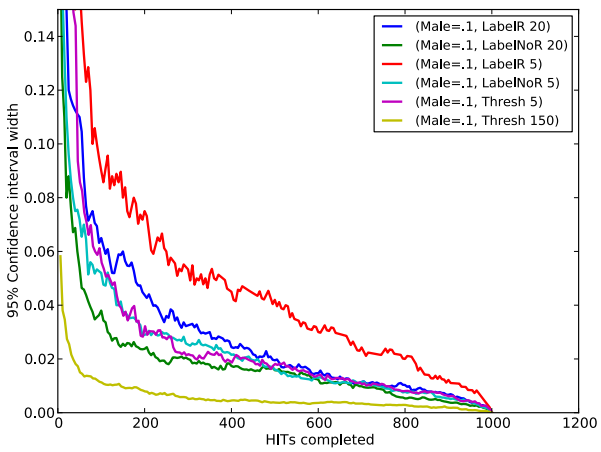

Figure 7: As we vary approaches and batch sizes, what 95% confidence interval width (Y-axis) do we achieve in a given number of HITs (X-axis)?

top workers finished 349 and 203 HITs of the 1000 HITs with error rates of about 46% and 26% respectively. In this figure we see that even after 1000 HITs, simply averaging across worker responses, more than half of which are spam, results in an inaccurate response. Applying our threshold-based spammer detection technique allows us to arrive at the actual fraction of items with high accuracy.

The chart also shows the convergence rate of the two approaches. Along the X-axis, we vary the number of HITs we sample from the 1000. For each X-axis value, we take 100 subsamples of that size from the 1000 HITs, and run *Avg* and *Thresh* on each of those subsamples. We plot the value of the 2nd smallest and 3rd largest estimates based on these subsamples, providing a bootstrapped measurement of the 95% confidence interval of the approach at each HIT size. Note that the gap between the lower and upper lines is smaller in the case of our threshold-based approach. From this, we can conclude that in addition to getting an accurate estimate with a large amount of HITs, we also converge on the correct value with a small number of them.

By using *Thresh*, the error in this example is reduced from 16.64% to 0.06% (a factor of about 265). The largest absolute improvement of the spam detection algorithm in our dataset brings an error from 26.74% to 1.77% (a 24.97% improvement).

Convergence Rates. As evidenced by the lack of a pattern in error rates amongst batch sizes in Figure 4, we did not find that batch size had a significant impact on the accuracy of our estimates at the limit (after 1000 HITs). As long as workers were willing to take on a count-based task, the error across batch sizes did not vary.

Since the label- and (spammer-eliminated) count-based approaches achieve similar error rates at the limit, what separates them is how quickly they converge on the true value. We explore the convergence rate of various approaches in Figure 7. In this chart, we measure the rate of convergence of various approaches when the fraction of males is 0.1. As we vary the sample size of the 1000 total HITs along the X-axis, we measure the 95% confidence interval width (e.g., the gap between the top and bottom lines in Figure 6), using the same subsampling method as the previous experiment.

We see that the fastest convergence rate is achieved with a count-based interface after spammer elimination with a batch size of 150 (the yellow line closest to the bottom left). The next fastest convergence rate is achieved by the label-based approach with no redundancy and a batch size of 20, followed by the batch size 5

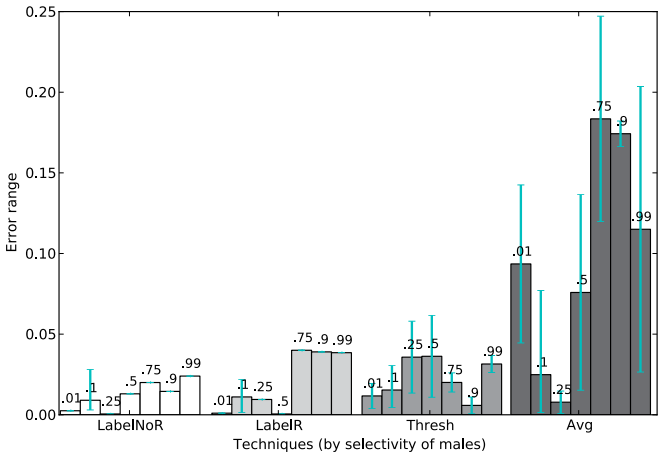

Figure 8: Bar chart of average error per approach, broken down by selectivity of males in the dataset. See Figure 4 for a description of the graph elements.

count-based and non-redundant label-based approaches. The two approaches with the slowest rates of convergence are the batch size 20 and batch size 5 redundant label-based approaches. Essentially, because they provide no additional protection against spammers, only 1 out of every 5 redundant HITs for each label provide novel information. Most importantly, we see that to be within 0.05 of the correct answer, the batch 150 count-based approach requires about 5 HITs, whereas the next fastest approach takes about 50. Similarly, we reach a confidence interval of .01 in less than 100 HITs with count-based batches of size 150, whereas the next fastest approach takes almost 600 HITs.

Varying Selectivity. In Figure 8, we show the error rates of different approaches as selectivity changes. While there is no clear trend for label-based interfaces, the spam-detected count-based technique (*Thresh*) sees a slight trend toward higher errors as selectivities approach 0.5. We see this trend again in Figure 9, where we show the convergence rate of the 95% confidence interval as we vary the frequency of males in the Face dataset, with slower convergence as selectivity approaches 0.5. The results are symmetric for selectivity .75, .9, and .99, and we omit them to make the graph clearer.

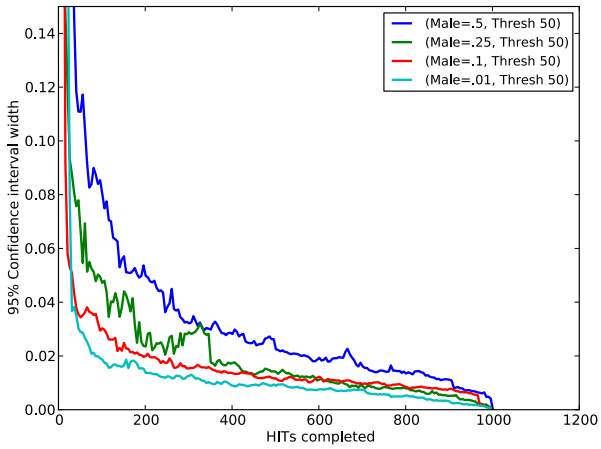

Figure 9: As we vary the selectivity of males, what 95% confidence interval width (Y-axis) do we achieve in a given number of HITs (X-axis)?

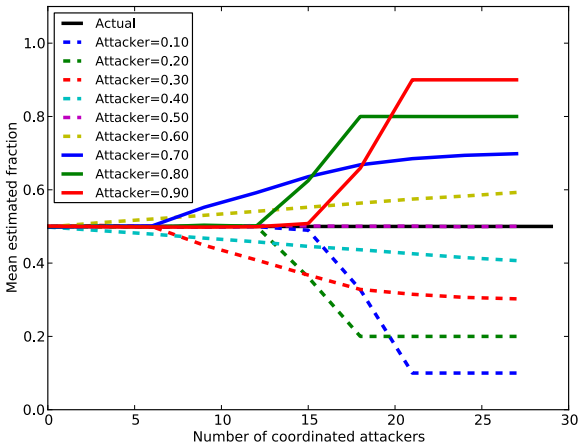

Figure 10: As we increase the number of coordinated attackers (X-axis), the spam detection algorithm eventually reports the attackers' values (Y-axis).

The increased error and reduced convergence rates for selectivities near 0.5 suggest that there are fewer errors in estimating the fraction of items with very frequent or very infrequent occurrences. We hypothesize that as the number of males approaches 50% of the data, workers have to perform more work to spot subtle differences in distribution. If the distribution is instead lopsided toward males or females, a quick glance can identify the outliers (less frequent) items in the dataset. A second observation also supports this hypothesis. In Figure 5, we see that the time workers spend in the count batch 50-based interface for male selectivities of 0.1 and 0.5 are about equal. If workers spend equivalent amounts of time on tasks that are easier to spot-check (e.g., identify 5 males of 50 pictures) than more nuanced tasks (e.g., are there 20 or 25 males out of 50?), the higher-selectivity tasks will see less accurate results.

A natural question arises: if the confidence interval becomes wider at higher frequencies, is there still a benefit to the count-based approach over the label-based one? The result is a rough draw at the highest level of batching for both techniques.

Generalizing Our Results. To see how well the results on the Face dataset generalize, we ran similar ones on the Shape and Tweet datasets. We were able to achieve similar conclusions on the Shape dataset, with very similar error and convergence rates, but our findings on the Tweet dataset revealed important differences.

On the Shape dataset, we tried two variations. Using a batch size of 50 for counting and 20 for labeling, we had workers approximate the border color and shape of two sampled distributions. For shape frequency estimation, we generated a dataset with 10% triangles, 30% circles, 30% squares, and 30% diamonds. For shape outline estimation, we generated a dataset with shape outline colors of 10% yellow, 30% orange, 30% red, and 30% green. These tasks are of different difficulty: given that the primary color of each shape is its fill color, identifying outline color is harder than determining its shape. Still, we found similar result and accuracy trends to the ones on the Face dataset. Additionally, as evidenced in Figure 5 (see C50 Shape and C50 Color), workers spent about as much time on these tasks as they did on batch 50 counts of faces at different selectivities. This further supports our hypothesis that across selectivities and visual task difficulties, workers tend to spend a constant amount of time per task, varying time on task only with batch size.

Results from the Tweet dataset were more surprising. Workers were asked to label and count tweets into one of the three categories described in Section 6.1. This task is harder than other estimation tasks described so far, since each tweet has to be read and analyzed, without any visual "pop-out" [26] effects.

Even though the task was harder, in Figure 5, we see that workers spent less time on Tweet counting (*C50 Tweet*) and labeling (*L20 Tweet*) tasks than they did on the images with equivalent batch sizes. Workers spent significantly less time on the count-based tasks than the equivalently sized count-based image counting tasks.

The error rates on the tweet datasets were also interesting. The label-based approach saw larger error rates than label-based image counting techniques, but still provided usable outcomes. The count-based approaches, on the other hand, saw error rates of up to 50%, signaling that workers were not accurately providing count estimates on tweets in our interface. These two error rate results are consistent with our timing results: as workers spent less time on a more difficult task, their error rates made the count-based interface less dependable. We consider the implications and future work suggested by these findings in Section 7.

6.3 Sybil Attacks

Coordinated attacks by a single worker with multiple identities (so-called "sybil attacks") or multiple workers attacking in concert make our count-based approach less effective at catching spammers, since such coordinated attacks can skew our estimate of the true mean. In Section 5 we described a technique for placing random amounts of gold standard data into each HIT to filter out such attackers, and we now quantify the benefit of this approach.

Because (to the best our knowledge) we did not experience any sybil attacks in our actual experiments on MTurk, we ran a series of simulations. We simulate a crowd of 30 workers estimating the fraction of items with a given property (the actual fraction is 0.5). Figure 10 shows the effect of this coordinated attack as we increase the number of attackers on the X-axis. The attackers do not count the items shown to them, instead agreeing to report the same fraction. The fraction they coordinate on (from 0.1 to 0.9) is shown on different lines. We use the spam-detection technique employed in the previous experiments and described in Section 4, without our gold-standard-based provisions to detect sybil attacks.

We see that as the number of coordinated attackers increases, the results become less and less accurate, eventually converging on the fraction that the attackers coordinate on. Larger attack fractions are initially farther from the overall mean, and so our spammer

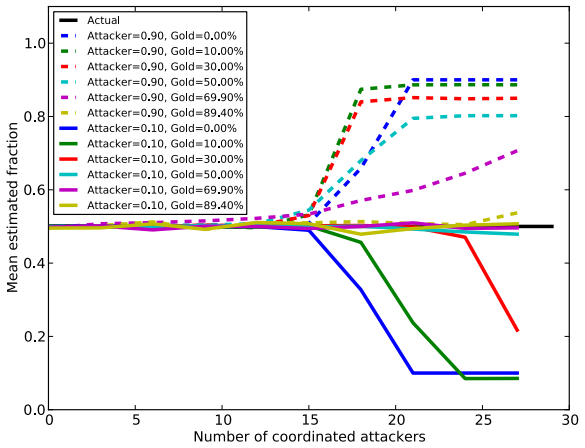

Figure 11: How do various amounts of gold standard data reduce the effectiveness of a coordinated attack?

elimination technique can withstand and filter more spammers with large attack fractions. With a large enough fraction of the attackers, however, the algorithm is always subverted.

In Figure 11, we see what happens when we add random amounts of gold standard data to each HIT, depicted in two scenarios. The dotted lines represent coordinated attackers providing response fractions equal to 0.9 pitted against HITs with various amounts of gold standard data ranging from 0% to 90% of each HIT on average. The solid lines represent coordinated attacks from attackers who provide a false value of 0.1, again pitted against HITs with varying average gold standard amounts.

We see that when attackers provide an incorrect value of 0.1 (solid lines), adding between 10% and 30% gold standard data nearly eliminates their effect. This shows the first benefit of gold standard data: outright identifying spammers whose corrected responses are negative. By the time workers see on average 30% gold standard data, all of the coordinating attackers are eliminated in all scenarios except for when there are 27 coordinating attackers. In that last case, one attacker was not eliminated because their assigned gold standard data was exactly 10%, which helped bring down the small sample that the remaining three accurate workers provided. More broadly, in expectation we eliminate 50% of attackers when the average amount of gold distributed equals the attack fraction, with linear increases in gold resulting in a linear amount of more attackers detected.

When attackers provide an incorrect value of 0.9 (dotted lines), it takes more gold standard data to neutralize their attack. This is because the benefit of outright spammer elimination is not available: inserting on average 20% males into tasks for which attackers claim 90% males will not result in a negative fraction after correction. Still, the other benefit of the gold standard approach kicks in with enough gold data (around 40%-70%): attackers who agreed on a single attack value are spread out by the random amounts of gold standard data on each HIT, reducing the coordination in their attack and making them easier to spot with spam elimination techniques.

As we discuss in Section 5, there are several ways to eliminate attackers that collude on a high value, and leave this to future work.

7. TAKEAWAYS AND DISCUSSION

Our results show that for estimation problems on datasets that exhibit visual "pop-out" effects such as images, count-based estimation is as accurate as label-based estimation, but arrives at an accurate result up to an order of magnitude faster. For datasets that require crowd workers to examine each item in detail, such as text classification, the label-based approach provides better accuracy. In our experience, the label-based approach works better with no redundant labels, instead using each additional HIT to sample more of the dataset and increasing the amount of information we learn.

Our spam detection algorithm successfully and significantly improved count-based estimation accuracy, in some cases by more than two orders of magnitude, over techniques such as taking the average or median worker result. While we did not experience label-based spammers, if the label-based approach is susceptible to spammers in the future, we can apply our count-based spammer elimination technique to the counts of non-redundant labels, thus eliminating spammers without requiring redundant labels.

In simulations, we showed how effective sybil attackers can be at subverting the spammer elimination technique. We then showed that as the number of coordinating attackers increases, we can increase the amount of gold standard data we add to every task on average to dampen the attack. In several cases, this approach eliminates the effect of the attackers. We also showed that it is important to use gold standard data from the *least frequent* class in our dataset, so that spammers attacking an infrequent property will be identified with less overall use of gold standard data.

To put our findings into context, we can get to within 5% of the fraction of items in a dataset using about 10 HITs. Labeling datasets that require 1000 HITs takes on the order of an hour at higher batch sizes. It would take less than five minutes to process 10 HITs on a system like MTurk. Thus, in addition to saving more than an order of magnitude of work in picking the correct operator to process first, we can also make the operator selection decision quickly, saving query processing time and money.

Given our ability to reproduce the low error and fast convergence of the count-based interface on several image datasets, and the poor performance of the count-based approach on the tweet dataset, we offer guidance for future usage. Text classification, which is harder to skim than high pop-out images, can not accurately be counted using the count-based interface. While more experiments are necessary, we do not believe that the limitations of the count-based approach are due to the specifics of our count-based interface as much as they are due to humans' inability to skim low pop-out data and estimate counts based on that skimming. The label-based interface is able to draw a worker's attention to each item and achieve better accuracy. One direction for future interfaces would be to combine the dense count-based interface's layout (a grid of items to be counted) with a lightweight marking mechanism for each data item (e.g., "click on tweets below that ask a question to followers").

One area for future development is in scaling up count- and label-based approaches to higher-cardinality groupings. Both interfaces are overwhelming to workers when more than five categories are available, and we are interested in interfaces that avoid this.

Finally, given that our spammer detection technique does not require redundant labels, it would be interesting to see how we can improve result quality in domains other than selectivity estimation. Using our technique, one could imagine an image labeling task that has high coverage but does not require redundantly labeling each image. Instead, once a worker is determined to be a spammer because their answer distribution differs from the crowd's, we could ask another trusted worker for an assessment.

8. RELATED WORK

In the databases field, several crowd-powered databases and query executors including CrowdDB [10], Deco [19], Jabberwocky [1], and our own Qurk [16] enable users to issue data-oriented tasks

to crowd workers. The majority of the work in the space has been around database operators, including filters [18], sorts and joins [17], and max-finding [12]. Recently, Trushkowsky et al. have shown how to both estimate the cardinality of a set, and identify the unique elements in that set with the help of the crowd [22]. This work is most related to our problem: particularly for calculating counts of large numbers of groups with the crowd, it is important to enumerate all members of that group. Our work, in addition to facilitating crowd-powered counts over such groups, is also the first to make use of crowds to power query optimizers by generating statistics for selectivity estimation. We show that it is both cost-effective and important to approximate how many items in a dataset have a given property to save time and money while avoiding extra crowd-powered operator invocations.

One of the key problems we solve is in crowd quality management. Ipeirotis et al. [14] show a practical algorithm for accurately labeling items in a dataset while identifying worker quality that is based on work by Dawid and Skene [6]. Karger et al. extend this work, identifying a theoretically better algorithm for quality labels [15]. We find that for count estimation, it is better to avoid the redundant labels required by these approaches, and instead increase diversity in samples to achieve faster convergence. Our spammer detection technique allows us to achieve even faster convergence.

Note that the use of the term "spammer" presents parallels to other forms of detection for uses such as email spam elimination. At a technical level, however, our work is different because both the signal (i.e., a count estimate) and mode of attack (i.e., providing a count along a spectrum) are different than in areas such as email spam detection, which tend to rely on more traditional unstructured text classification frameworks.

Human vision and psychology research has seen many studies of people's performance at visual search tasks. A related finding from Wolfe et al. suggests that for rare (low-selectivity) items, error rates will increase [25]. Our findings suggest a different pattern, though the scenario is different. In situations such as security scans or radiology, rare detection events (e.g., bombs or tumors) require a report, while frequent events are to go unreported. In our tasks, both rare and frequent events must be acted on by reporting a count or a label. Thus, more frequent items require counting more, resulting in higher fatigue and mistakes than low-frequency items. We thus found higher error rates for frequent/high-selectivity items than in rare items that pose less of a threat of being miscounted. Item frequency is not the only factor in human perception. The "pop-out" effect [26] also matters. Due to the effect, the relative prevalence of colors, shapes, location, and orientation of items affect reaction time and accuracy of finding a particular item.

While our algorithm and count-based approach achieves good estimates faster than the state of the art, it is susceptible to sybil attacks by workers with multiple identities or workers who act in concert to avoid being identified as spammers while finishing tasks faster than high-quality workers. Sybil attacks [7] are well-researched in systems design, but have not yet been introduced as research problems with practical solutions in the field of human computation. We propose a sybil-resistant solution to our estimation problem, and show that it works in simulations of such attacks.

9. CONCLUSION

We have shown that, for images, a count-based approach with a large batch size can achieve commensurate accuracy to a label-based approach using an order of magnitude less HITs. For text-based counts, we found that the label-based approach has better accuracy. Our spammer detection algorithm improves accuracy by up to two orders of magnitude without requiring redundant worker responses. In simulations, we show how randomized placement of gold standard data can reduce the effects of a coordinated attack. This is the first work, to our knowledge, to push crowdsourced computation into database optimizers. In doing so, we enable fast selectivity estimation decisions that reduce downstream monetary cost and latency, and facilitate accurate image-based count estimation with less input from the crowd. Such optimizations can potentially save crowd-powered database users significant time and money.

10. ACKNOWLEDGEMENTS

We are grateful to Eugene Wu and Panos Ipeirotis for their discussions on this topic.

11. REFERENCES

[1] S. Ahmad et al. The jabberwocky programming environment for structured social computing. In *UIST*, 2011.
[2] P. André, M. S. Bernstein, and K. Luther. Who gives a tweet?: evaluating microblog content value. In *CSCW*, pages 471–474, 2012.
[3] J. Attenberg et al. Beat the machine: Challenging workers to find the unknown unknowns. In *HCOMP*, 2011.
[4] M. S. Bernstein et al. Soylent: a word processor with a crowd inside. In *UIST*, 2010.
[5] M. S. Bernstein et al. Crowds in two seconds: enabling realtime crowd-powered interfaces. In *UIST*, 2011.
[6] A. Dawid and A. Skene. Max. Likelihood Estimation of Observer Error-Rates Using the EM Algorithm. *Royal Stat. Soc.*, 1979.
[7] J. R. Douceur. The sybil attack. In *IPTPS*, pages 251–260, 2002.
[8] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1993.
[9] W. Feller. *An Introduction to Probability Theory, Vol. 1*. Wiley, New York, NY, third edition, 1968.
[10] M. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: Answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.
[11] M. R. Gormley, A. Gerber, M. Harper, and M. Dredze. Non-expert correction of automatically generated relation annotations. In *HLT-NAACL*, 2010.
[12] S. Guo, A. Parameswaran, and H. Garcia-Molina. So who won? dynamic max discovery with the crowd. In *SIGMOD*, 2012.
[13] P. Ipeirotis. Identify Verification (and how to bypass it), March 2012. `http://www.behind-the-enemy-lines.com/2012/01/identify-verification-and-how-to-bypass.html`.
[14] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *HCOMP*, pages 64–67, 2010.
[15] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *NIPS*, pages 1953–1961, 2011.
[16] A. Marcus, E. Wu, et al. Crowdsourced databases: Query processing with people. In *CIDR*, 2011.
[17] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 2011.
[18] A. Parameswaran et al. Crowdscreen: Algorithms for filtering data with humans. 2011.
[19] A. Parameswaran and N. Polyzotis. Answering queries using databases, humans and algorithms. In *CIDR*, 2011.
[20] J. Surowiecki. *The Wisdom of Crowds*. Anchor Books, 2005.
[21] F. Tarrés and A. Rama. GTAV Face Database, March 2012.
[22] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Getting it all from the crowd. *CoRR*, abs/1202.2335, 2012.
[23] R. R. Wilcox and H. J. Keselman. Modern robust data analysis methods: measures of central tendency. *Psych. methods*, Sept. 2003.
[24] E. B. Wilson. Probable inference, the law of succession, and statistical inference. *American Statistical Assoc.*, 22(158), 1927.
[25] J. M. Wolfe et al. Low target prevalence is a stubborn source of errors in visual search tasks. In *Journal of Experimental Psychology*, 2007.
[26] J. M. Wolfe and H. Pashler (Editor). *Attention*, chapter Visual Search, pages 13–73. University College London Press, 1998.