VoxPL: Programming with the Wisdom of the Crowd

Daniel W. Barowy and Emery D. Berger
UMass Amherst
{dbarowy,emery}@cs.umass.edu

Daniel G. Goldstein and Siddharth Suri
Microsoft Research
{dgg,suri}@microsoft.com

ABSTRACT

Having a crowd estimate a numeric value is the original inspiration for the notion of "the wisdom of the crowd." Quality control for such estimated values is challenging because prior, consensus-based approaches for quality control in labeling tasks are not applicable in estimation tasks.

We present VOXPL, a high-level programming framework that automatically obtains high-quality crowdsourced estimates of values. The VOXPL domain-specific language lets programmers concisely specify complex estimation tasks with a desired level of confidence and budget. VOXPL's runtime system implements a novel quality control algorithm that automatically computes sample sizes and obtains high quality estimates from the crowd at low cost. To evaluate VOXPL, we implement four estimation applications, ranging from facial feature recognition to calorie counting. The resulting programs are concise—under 200 lines of code—and obtain high quality estimates from the crowd quickly and inexpensively.

ACM Classification Keywords

H.1.2 Information Systems: Human information processing; D.3.2 Language Classifications: Specialized application languages; G.3 Probability and Statistics: Probabilistic algorithms (including Monte Carlo)

Author Keywords

crowdsourcing; crowdprogramming; quality control; scalability; domain-specific languages; wisdom of the crowd.

INTRODUCTION

In this paper, we introduce VOXPL, a high-level programming framework and quality control algorithm for harnessing the wisdom of the crowd to obtain high-quality estimates of continuous quantities. For any estimation task, programmers provide their question together with a *target confidence interval* and a maximum *budget*. VOXPL incrementally increases the sample size, minimizing the cost, until either the estimate is sufficiently refined or the budget is exhausted. In the latter case, VOXPL returns the best estimate possible given the budget. VOXPL significantly extends the reach of crowdsourcing to encompass collective estimation tasks.

The earliest work using the crowd to estimate a quantity—the canonical example of the phrase "the wisdom of the crowd"—is Galton's 1906 paper describing the accurate estimation of the weight of an ox using a crowd's guesses [16, 44]. As Galton noted, many of the crowd's guesses were wrong. Nonetheless, by aggregating the guesses, an accurate estimate could be obtained. In Galton's case, the estimate was within an astonishing 0.8% of the true value. Galton argued that the median best represented the opinion of the crowd—the *vox populi*, as he put it—since "every other estimate [was] condemned as too low or too high by a majority of the voters". VOXPL builds on Galton's insight, applying it to crowdsourcing in order to obtain high-quality estimates.

Automatic quality control. Modern day programmers who rely on the crowd need an automated method for generating high quality estimates. Without explicit data quality guarantees, programmers are left with either manually inspecting the data for errors—an onerous process for tasks of any realistic scale—or writing their own error checking routines and potentially wasting resources or falling short of quality goals.

A naive quality control strategy recruits a redundant set of workers and aggregates their responses. Unfortunately, finding the right number of workers requires that programmers strike a delicate balance: recruit too few workers and the task falls short of quality goals; recruit too many workers and worker time and task budgets are wasted. Without careful attention to efficiency, these human factors become acute pain points for both programmers and workers. VOXPL helps crowd workers because it avoids asking for redundant data. VOXPL helps programmers because quality goals can be fully automated by way of a simple, high-level abstraction—the *target confidence interval*—that eliminates the need for manual checking.

Estimation vs labeling. Prior algorithmic and programmatic approaches to quality for crowdsourced work focus on *labeling* tasks, such as determining whether a picture is of a cat or a dog. However, these approaches do not extend to *estimation* tasks such as determining the number of calories in a plate of food, or guessing the weight of an ox. Estimation tasks are qualitatively and quantitatively different from labeling tasks. Labeling tasks ask workers to choose a single best option from a discrete (and low-cardinality) set of options. By contrast, estimation tasks ask workers to approximate a continuous value (or at least from a set with high cardinality).

CHI 2017, May 6-11, 2017, Denver, CO, USA.

http://dx.doi.org/10.1145/3025453.3026025

This difference means that algorithms that choose a label based on voter agreement do not work for estimation tasks: not only is there rarely a majority opinion, but worker responses are also not likely to agree at all on any single value.

This paper makes the following contributions:

- VOXPL automates the computation of high-confidence estimates from the crowd. By adjusting sample sizes on the fly, VOXPL does not waste crowd resources by asking for unnecessary input. In addition, VOXPL makes it possible for ordinary programmers to harness the wisdom of the crowd for estimation tasks in their applications. VOXPL eases the burden both on programmers and on crowd workers.
- The VOXPL algorithm, which is the first to systematically address quality for estimation tasks. VOXPL lets programmers trade cost for efficiency, producing estimates given a confidence level (e.g., 95%), a target confidence interval (e.g., ± 50), and a budget.
- The VOXPL framework, a runtime system that automatically generates the correct sampling procedure and statistical tests for requested estimates. VOXPL ensures that estimates are both efficient and statistically sound.
- Empirical validation that VOXPL is concise, expressive, and efficient with four case studies: (1) estimating the caloric content of meals; (2) facial feature recognition; (3) estimating the mass of an ox; and (4) identifying the location of objects in photos. In every case, VOXPL obtains accurate estimates from the crowd with few lines of code, quickly, and at low cost.

Technical Challenges

VOXPL is a declarative programming language for crowdsourcing estimates. Its central feature is an abstraction that lets programmers trade cost for quality. This abstraction lets VOXPL adjust sample sizes on the fly, efficiently utilizing human resources by asking for only necessary input. To make this possible, VOXPL must address the following challenges:

Guaranteeing quality for estimation tasks. As discussed above, estimation tasks pose unique quality control challenges that render prior approaches designed for labeling tasks inadequate. Algorithms based on voting do not work (or would require a prohibitive number of responses) when the range of possible answers is large. By contrast, the VOXPL algorithm uses a *non-parametric* statistical procedure that produces high-quality estimates without the need for voting and without any assumption about the distribution of responses.

Automatically computing the appropriate sample size. Programmers do not necessarily know the "right" sample size to choose for an estimation task. A sample needs to be large enough so that the estimated quantity is likely to be correct but small enough to be cost-effective. Computing the appropriate sample size *a priori* requires knowing the response distribution, the very thing that the programmer is attempting to measure. This limitation can be skirted by sampling incrementally, but this procedure risks introducing bias into results because there is a small probability that constraints will be satisfied by random chance [20]. The VOXPL runtime incrementally finds the minimum sample size for any possible estimation program, correcting for bias using the Bonferroni method, to ensure that estimates are valid [18].

Automatically guaranteeing the soundness of estimates. The true distribution of responses from workers is not usually known *a priori*. Typically, practitioners choose the right statistical procedure by making assumptions about the data (e.g., it is normally distributed). Unfortunately, if a programmer is allowed to specify any arbitrary estimation task, no distributional assumption is valid. Because the VOXPL algorithm is nonparametric, it performs the right sampling procedure automatically regardless of the distribution of responses.

Enabling flexible selection of quality requirements. In some applications, such as facial recognition and calorie counting, tight error bounds are important. In others, like estimating the construction cost of a house, rougher approximations are acceptable. Only the programmer knows how much uncertainty is tolerable in a given application. Unfortunately, minimizing uncertainty is in direct tension with minimizing cost, which makes it hard for programmers to balance these two goals correctly. The VOXPL domain-specific language lets programmers communicate their quality goals to the runtime simply and intuitively.

Supporting multidimensional estimates. While it can be straightforward to compute confidence for statistics over a single dimension (e.g., the median), there is in general no simple method to estimate confidence for arbitrary statistics over multiple dimensions. VOXPL lets programmers compute multidimensional estimates simply, by composing VOXPL functions (see Figure 1 for an example). The underlying runtime automatically computes the correct estimator and bounds, regardless of the number of dimensions and statistics used.

Supporting functions over estimates. Estimates can be used as inputs to calculations, such as computing the sum total caloric intake from a set of calorie estimates. For the error of the whole calculation to be valid, quality control methods need to consider the *entire computation*. Unfortunately, there are no known analytical solutions to compute consistent error bounds for arbitrary mathematical transformations of estimates. VOXPL instead employs an empirical method called the *bootstrap*—which can only be used at runtime—that guarantees consistency for arbitrary functions of estimates [13].

Automatically determining the wage to pay workers. While Galton's ox-weighing task benefitted from free labor in the form of contest participants, crowdworkers cannot (and should not) be expected to work for free or marginal wages. However, it is difficult to predict the cost of obtaining labor to do estimation tasks *a priori*. VOXPL dynamically and adaptively adjusts the amount of money to pay workers, while ensuring that they are always paid at the same effective hourly rate (by default, the US Federal minimum wage).

Ensuring fairness for crowdworkers. VOXPL is designed to respect the Worker's Bill of Rights to the greatest extent possible [22]: it never arbitrarily rejects work; it pays workers as soon as the work is completed; it pays workers the US

(a) (b)

Figure 1. Workers were asked to identify the center point of the nose in a set of images of people and animals. Red dots in these examples denote individual responses, the cross denotes the centroid, and blue ellipses denote the 95% confidence interval. The difference in variance between (a) and (b) illustrates that some tasks are inherently more difficult than others. Intuitively, VOXPL allows programmers to ensure minimal quality thresholds by limiting the maximum size of confidence ellipsoids, even in n dimensions.

minimum wage by default; and it automatically raises pay for tasks until enough workers agree to take them.[1]

DOMAIN-SPECIFIC LANGUAGE

VOXPL is a declarative domain-specific language (DSL) for crowdsourcing estimation tasks. VOXPL is embedded within Scala, a programming language that runs on the Java Virtual Machine. This embedding lets VOXPL functions work exactly like ordinary functions in Scala and Java, allowing users to freely intermix them with ordinary application code. Coordination of multiple crowdsourcing tasks (including interdependent tasks) is achieved by composing VOXPL functions.

VOXPL Components

The VOXPL DSL is built on top of a simple grammar that ensures that all user-defined expressions are well defined. VOXPL leverages the underlying type system such that only valid estimation programs compile. For those that do, VOXPL is always capable of inferring error bounds.

Programmers perform complex estimates with VOXPL using the *combinator pattern* [28]. Combinator-based languages have two kinds of functions: (1) those that *create* primitive values, and (2) those that *combine* primitives into more complex structures. Primitive values are strictly a function of their inputs and do not have side-effects (i.e., they are *purely functional*). In VOXPL, the key primitive value is an estimate: a random variable representing responses to a question.

Estimate creation. Programmers create estimates with the `Estimate` function and a small set of parameters. The most important function parameters are `text` (the question posed to the worker), `budget` (the maximum amount of money that VOXPL may spend estimating a given question), and `confidence_interval` (the maximum width of the estimate's confidence interval). Other parameters are automatically supplied

[1]To avoid worker disputes, we paid all workers regardless of work quality for all evaluations in this paper.

```
def numCalories(url: String) = Estimate (
  confidence_interval = SymmetricCI(25),
  text = "How many calories are in the food pictured?",
  image_url = url
)
```

Figure 2. A complete VOXPL definition for a function that estimates the number of calories for breakfast. Here, the programmer requests an estimate with a maximum tolerable error of ± 25 calories (for the default 95% confidence interval).

with sane defaults, but may be overridden by the programmer: `estimator` (the L_1 median by default), `confidence` (the confidence level, with a default of 95%), `title` (which displays a task title on MTurk, defaulting to the `text` field when not provided), and `image_url` (which optionally displays an image).

Figure 2 shows a VOXPL function that computes the caloric content of a plate of food. This VOXPL example contains the bare minimum of information required to perform the job. All other details, including task defaults, scheduling, pricing, and quality control are completely managed by the runtime. When this function is executed, it launches a set of *tasks* on the target crowdsourcing platform (*e.g.*, it posts a HIT on MTurk).

Estimate composition. Programmers combine estimates with arbitrary Scala functions using the `combineWith` combinator. Combinators take two estimates (e.g., `numCalories(lunch)` and `numCalories(dinner)`) and a combining function of the estimates. Figure 3 shows two combinators applied in sequence using +.

Estimation Question Types

VOXPL provides two estimation question types: `Estimate` and `MultiEstimate`. The former asks workers a single estimation question per form. The latter batches an arbitrary number of questions into a single HIT, enabling joint estimation of multiple quantities. For example, the x and y coordinates

```
val total = numCalories(breakfast) +
            numCalories(lunch) +
            numCalories(dinner)
```

Figure 3. A complete VOXPL definition for a combined calorie estimate for the whole day using the function defined in Figure 2. `breakfast`, `lunch`, and `dinner` are, respectively, image URLs for breakfast, lunch, and dinner. `+` is syntactic sugar for `combineWith` using the function `(a:Double) => (b:Double) => a + b`.

shown in Figure 1 were obtained with the `MultiEstimate` function shown in Figure 5.

Confidence Interval Constraints

VOXPL uses confidence intervals to tie together the statistical notion of *precision* with a programmer's intuitive notion of *tolerable error*. Given a statistic (*e.g.*, the median), confidence (*e.g.*, 95%), and sample data, an empirical confidence interval can always be inferred (*e.g.*, ± 25.8). By asking the programmer to declare a *target* confidence interval (*e.g.*, ± 25), VOXPL has sufficient information for a well defined statistical procedure: VOXPL gathers reponses from the crowd, repeatedly testing whether the inferred interval is at least as tight as the target interval, until either it is (with high probability), or the budget is exceeded. At this point, VOXPL returns the result.

There are three kinds of interval constraints:

1. *Symmetric* confidence intervals, where bounds are defined as symmetric low and high deltas around a single point. For example, `confidence_interval = SymmetricCI(50)` with `confidence = 0.95` (the default) means that the low and high bounds should be defined as the bounds of the 95% confidence interval about the point estimate.
2. *Asymmetric* confidence intervals, where bounds are defined as distinct low and high deltas around a single point. This constraint is primarily intended to support one-sided confidence intervals, although it is strictly more powerful. For example, `confidence_interval = AsymmetricCI(0,45)` along with the default `confidence = 0.95` defines the one-sided, 95% confidence interval above the point estimate. Using asymmetric confidence intervals lets programmers obtain results more efficiently when they do not care equally about both sides of the distribution.
3. *Unconstrained* confidence intervals, where the user supplies only a fixed sample size. This option disables VOXPL's dynamic sample size calculation. VOXPL simply returns the inferred symmetric confidence interval (at the desired confidence level) for the given sample size.

Combinator Logic

The VOXPL domain-specific language (DSL) is built around a small combinator logic that informs the runtime how to convert user-defined tasks into estimation and joint estimation procedures. While VOXPL uses the L_1 median as the default estimator, VOXPL can use any user-defined mathematical function (or combinations thereof) as estimators. Joint estimation combinators control the combined error of an arbitrary number of estimates. VOXPL automatically converts estimates into joint estimates when necessary.

For example, suppose a programmer wants to use the function defined in Figure 2 to estimate their caloric intake for the entire day as `numCalories(breakfast) + numCalories(lunch) + numCalories(dinner)`. The combination of these estimates, each with its own confidence interval, is also an estimate. However, the combined estimate's confidence interval is not in general a straightforward function of each component's confidence interval [13]. Nonetheless, programmers using VOXPL's DSL are guaranteed to compute estimates with correct error bounds, satisfying a property known as *estimator consistency*. This feature frees programmers from the fact that, in general, combined estimates entail complicated—and sometimes unknown—closed forms for composed error distributions. An application that combines estimates for breakfast, lunch, and dinner is shown in Figure 3.

Runtime

The VOXPL runtime generalizes and subsumes the AUTOMAN runtime system, which embeds labeling tasks into a programming language. VOXPL inherits AUTOMAN's automatic scheduling, pricing, and budgeting. For a detailed discussion of our modifications, see Related Work.

ALGORITHM

The VOXPL quality control algorithm produces valid estimates for all VOXPL programs that compile. Since statistical estimation theory can only provide limited analytical guidance for exact solutions to computing bounds, VOXPL skirts these limitations by solving for approximate solutions instead. To do this, VOXPL relies on the *basic bootstrap*, a *nonparametric* (i.e., distribution-free) Monte Carlo procedure that estimates error bounds for arbitrary functions of unknown multidimensional distributions [12, 13, 46]. The statistic in question can be simple (like the median) or complex (like the sum of L_1 medians). Armed with bootstrapped confidence bounds and the fact that uncertainty decreases as sample sizes increase, the VOXPL algorithm greedily samples from the crowd until either the programmer's constraints are met or the budget is exceeded. If the budget is exceeded, the best known estimate is returned. This section describes the algorithm formally.

Estimation

The goal of VOXPL is to estimate an unknown *parameter* θ of an unknown distribution F of crowd responses on space $\mathscr{X}$. Let $X = (x_1, \ldots, x_n)$ be a real-valued, i.i.d. sample of responses from F of size n.

Point estimates. Let $\hat{\theta}(X)$ be an arbitrary real-valued *statistic* (a function of X), the *point estimate of* θ.

Interval estimates. The fact that $\hat{\theta}$ is an arbitrary, user-defined function over an unknown distribution F complicates interval estimation since precise confidence bounds are only known for specific statistics (e.g. the mean) of known distributions (e.g. the normal distribution). Nonetheless, nonparametric methods, which relax parametric assumptions, can be used to estimate arbitrary $\hat{\theta}$ with high accuracy [49].

VOXPL uses the *basic bootstrap* procedure for estimation [46]. The bootstrap produces an estimate of parameter θ, denoted $\hat{\theta}^*$, by way of the statistic $\hat{\theta}$ and random "replicates" of X

we denote X^*. Let $\hat{F}$ be the empirical distribution such that each $x \in X$ contributes $1/n$ mass. Let B be the number of *bootstrap replications*. For each b from $1 \ldots B$, a bootstrap sample X_b^* is drawn from $\hat{F}$ randomly with replacement and used to compute the b^{th} *bootstrap replication* $\hat{\theta}^*(b)$.

VOXPL uses the *percentile method* to calculate the confidence interval [13]. Let $\hat{\theta}(\alpha) = \widehat{CDF}^{-1}(\alpha)$, a function that returns the *real value* corresponding to the $(1-\alpha)\cdot 100$th percentile of bootstrap replicates $\hat{\theta}^*$. Thus, $\theta \in [\hat{\theta}(\alpha), \hat{\theta}(1-\alpha)]$. As $n \to \infty$, $[\hat{\theta}(\alpha), \hat{\theta}(1-\alpha)]$ will include θ with probability $1-\alpha$.

Joint estimates. VOXPL is capable of estimating n parameters simultaneously, where n is an arbitrarily large number. Joint estimation is a straightforward generalization from the single estimation case. The joint significance threshold in this case is $\beta = \max 1-\alpha_1, \ldots, 1-\alpha_n$ (the maximum of all of the individual significance levels), and the n-dimensional confidence interval is defined as the region containing all of the estimates with probability β.

Sample Size Determination

VOXPL's initial sample size is 12 by default; this value is the knee of the curve for confidence intervals over a normally-distributed population [6, p. 20]. If the distribution of responses happens to be normal, this value minimizes latency.

There are two outcomes after sampling: (1) the confidence interval satisfies the user's width and confidence level, or (2) it does not. In case (1), VOXPL returns the estimate and confidence interval. Otherwise (2), it refines the bounds by obtaining another sample; the sample size doubles after each iteration. In the event that the crowd does not supply timely responses (a "timeout"), VOXPL first checks whether the estimate already meets the user's constraints before scheduling replacement tasks; if so, it returns that estimate.

Bounds estimated by VOXPL can accurately reflect the true variability of the population yet not meet (unrealistic) user constraints. The budget parameter serves as a limiting factor on the total sample size, ensuring that estimation always terminates with what the programmer believes to be a reasonable maximum cost.

Correcting for Multiple Comparisons

Statistical procedures that determine the sample size dynamically (i.e., that do not run with a fixed sample size) run the risk of terminating too early. The reason is that they repeatedly test whether they have enough evidence to terminate the study. Strictly speaking, each test constitutes a different hypothesis. The intuition behind the problem is simple: given a statistical test that is capable of correctly rejecting the null hypothesis with a 95% probability (i.e., with a p-value of 0.05), then in expectation, applying that test 100 times will incorrectly reject the null hypothesis 5% of the time.

To avoid this problem, VOXPL employs the Bonferroni correction, which keeps a count of the number of hypotheses and adjusts the test's confidence threshold accordingly [11]. More precisely, the Bonferroni correction controls what is called the *familywise error rate*, the probability of rejecting at least one true hypothesis.

Let $H_1, \ldots, H_m$ be a set of m hypotheses, let $p_i, \ldots, p_m$ be their associated p-values, and let $1-\alpha$ be the confidence level. The Bonferroni correction ensures that rejecting the null hypothesis for all $p_i \leq \frac{\alpha}{m}$ controls the familywise error rate. Since in VOXPL's case, all hypotheses test the same thing ("have user constraints been met?"), it is not useful to conclude that some tests be rejected while others are not. In other words, the only test that matters is H_m. Therefore, VOXPL only terminates its sampling procedure when $p \leq \frac{\alpha}{m}$.

Discussion

Parameter-free modeling. VOXPL uses the basic bootstrap, a non-parametric method, for handling arbitrary estimates of responses from an unknown distribution of crowd responses. The bootstrap trades the efficiencies gained by exact and/or parametric methods (such as the Gaussian distribution) for extra flexibility and robustness. This flexibility allows programmers to supply arbitrary combinations of arbitrary statistics. VOXPL produces estimates and confidence intervals numerically, rendering analytical solutions unnecessary.

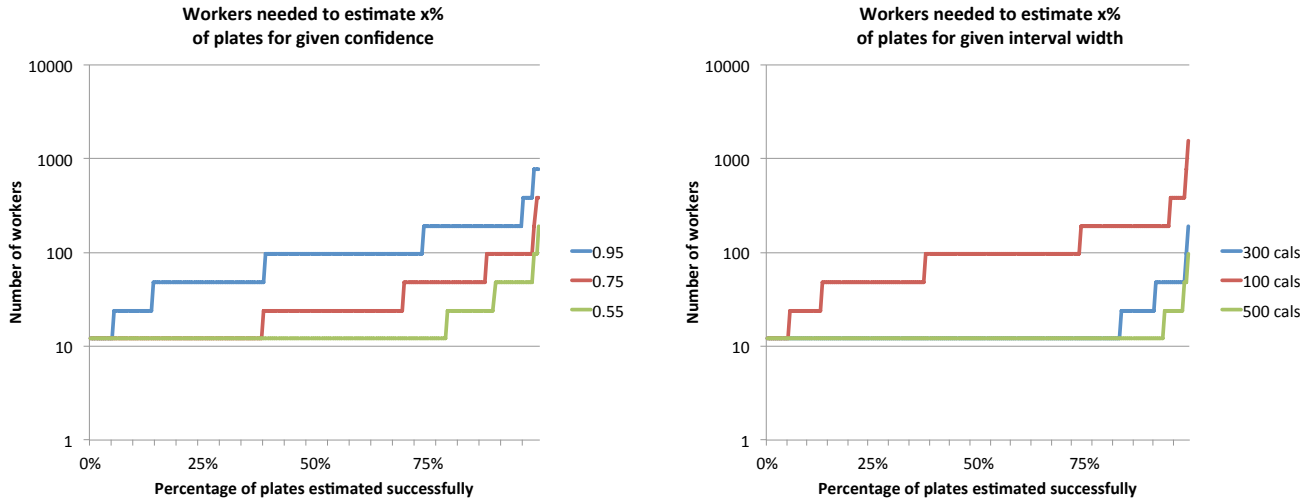
Although they do not have the statistical power of exact models, in practice, non-parametric methods are often quite efficient in terms of sample size. Furthermore, their computational requirements are modest by modern standards. The bulk of VOXPL's computation time is spent in bootstrapping code. For a typical estimate, VOXPL will recompute a statistic hundreds of times, which can usually be performed in microseconds on a modern computer. Note that VOXPL's throughput is dominated by *human* latency, so bootstrap costs are negligible.

Non-parametric methods are also more robust to model deviations. For instance, many otherwise-normal distributions are constrained by floor or ceiling effects (such as estimates of calories, which cannot be less than zero), which have important implications for estimates. Non-parametric procedures such as the bootstrap use empirical confidence bounds and are therefore less sensitive to asymmetry.

While the bootstrap is not guaranteed to produce a good estimate for every statistic, where exact solutions are known (e.g., for the Gaussian and other common distributions), the bootstrap's approximation error is known to be small (sometimes smaller than standard approximations) and estimates often converge as sample size increases [7, 14, 13].

Estimators and the breakdown point. Producing accurate estimates from noisy data is not a new problem. Practicing scientists commonly remove outliers from data by hand [37, Chapter 1.3.5.17]. This procedure is not feasible in the completely-automated scenario targeted by VOXPL. Nonetheless, the combination of the basic bootstrap and robust estimators means that VOXPL often produces good estimates in a wide variety of situations without user intervention.

The default estimator for VOXPL is the L_1 median, which estimates a distribution's central tendency. The L_1 median is a generalization of the ordinary median to more than one dimension. The Evaluation section demonstrates that the L_1 median



a b

Figure 4. These plots show how many workers are required to ensure that a given fraction of all tasks meet quality constraints for the calorie counting application (the y-axis is log scale). The right choice of sample size depends both on worker capabilities and the requested constraints, making *a priori* sample sizes either inefficient or ineffective. (a) shows the number of workers required for five different confidence level settings with the confidence interval fixed at 100 calories. At the 0.95 level, most tasks require around 100 workers. (b) shows the number of workers required for five different confidence interval width settings with the confidence level fixed at 0.95. Higher confidence settings or tighter confidence widths generally require more workers.

works well in both univariate and multivariate situations, as well as in outlier-prone scenarios. Here, we justify from a theoretical standpoint why the L_1 median is a good estimate when outliers are expected.

Informally, the *breakdown point* of an estimator is the proportion of data which must be moved to infinity so that the estimate will do the same [2]. While the arithmetic mean is widely used to capture central tendencies of distributions, it is exquisitely sensitive to outliers. This sensitivity is reflected in its breakdown point, $1/n$, meaning that a single large outlier is sufficient for the mean to fail to accurately estimate a distribution's central tendency. By contrast, both the ordinary median and the L_1 median—the point that minimizes the sum of the L_2 norms between itself and every other point—has a breakdown point of $1/2$, which is the theoretical maximum breakdown point.

EVALUATION

In this section, we evaluate VOXPL on a set of applications designed to explore its expressiveness, efficiency, and accuracy. Since we do not empirically evaluate ease of use, we note anecdotally that we find VOXPL easy to use. Many of the programs are concise (e.g., the calorie counter function is 10 lines of code) and took on the order of 10-20 minutes to write. The workflow consists of (1) adding a line to a project's build file to add a dependency to the VOXPL library, (2) writing an `import` statement to use the library in user code, (3) providing MTurk credentials, (4) defining a VOXPL function, and then (5) calling the function.

Photographic Calorie Counter

We wrote an application in VOXPL that uses the crowd to estimate the caloric content of a plate of food from a photograph. This application lets us explore VOXPL's efficiency-accuracy tradeoff across a range of confidence threshold and confidence interval width settings.

We perform this evaluation using a data set of 208 school lunch photos from elementary schools. For each photo, we have ground truth nutritional data that we obtained from school lunch menus, which public schools are required to report by Federal law. A sample lunch is shown in Figure 6. Each photo also includes a US quarter for scale.

Our evaluation proceeds as follows. First, we obtained large sample sizes for each plate (200 estimates) by having VOXPL post the estimation tasks on MTurk, but with a fixed sample size, rather than relying on VOXPL's quality control mechanisms. The accuracy of individual responses varies widely: the *mean absolute error* (MAE) for this dataset ranged from 1.56 to 305.61 calories for individual plates. Next, we run VOXPL with a variety of settings over these samples as if they were generated live by workers.

We conducted two different simulation experiments as described above, in both cases costs were generally linear with the number of workers ($0.06 per worker). First, we ran VOXPL with a fixed confidence interval width of 100 calories, varying the confidence parameter between 0.55 and 0.95, and measuring the number of responses required to satisfy user constraints (Figure 4(a)). When we varied confidence level, VOXPL needed an average of 111 responses for the highest confidence level (0.95) versus 15 for the lowest level (0.55).

Next, we ran VOXPL with a fixed confidence level (0.95), varying confidence interval widths between 100 and 500, and measured the number of responses required to satisfy user constraints (Figure 4(b)). Similarly, when we varied the confidence interval, VOXPL needed an average of 107 responses for the tightest confidence interval (width = 100) versus 9 for

```
def whereIsTheNose(imgUrl: String) = MultiEstimate (
  budget = 6.25,
  title = "Where is this person's nose?",
  text = "Locate the center of the tip of this " +
         "person's nose, click on it with your " +
         "mouse, and press the submit button.",
  image_url = imgUrl,
  dimensions = Array(
    Dim(id = 'x,
        confidence_interval = Symmetric(20)),
    Dim(id = 'y,
        confidence_interval = Symmetric(20))
  ),
  layout = layout,
  estimator = Mean
)
```

Figure 5. The function definition for a program that locates facial features (here, noses) from image URLs. The `layout` variable implements UI controls using Javascript and CSS to style the form automatically generated by VOXPL on MTurk. `Dim` fields describe constraints for dimensions x and y.

Figure 6. A sample of one of the 208 school lunch images we use to evaluate VOXPL's calorie-counting application (for which we have ground truth). Each picture is accompanied by a US quarter to provide a sense of scale.

the widest (width = 500). These experiments demonstrate that VOXPL automatically performs the appropriate cost-accuracy tradeoff. VOXPL automatically recruits more workers as constraints become more stringent.

The state of the art in machine learning-based caloric estimation is Google's IM2CALORIES [36]. IM2CALORIES does not rely on the crowd. Instead, it combines GPS location, vision, and information retrieval techniques in order to do calorie estimates. It first takes a GPS reading and then finds a likely set of menus from Google's database by searching by location. Next, it uses this information to find likely food matches from the image using hints from likely menus. Finally, likely foods are looked up in a nutritional database, portions estimated, and calories summed. By contrast, VOXPL uses only the food images themselves and queries only the crowd, without any additional information.

IM2CALORIES is not publicly available, so a direct comparison is not possible; however, VOXPL appears to produce better estimates. The IM2CALORIES paper reports that their best performing algorithm has a mean absolute error (MAE) of 152.95 kcal with a standard error (SE) of 15.61 kcal. VOXPL's best performing setting ($1-\alpha = 0.95$; confidence interval width = 100) has a MAE of 103.08 kcal with an SE of 6.00.

This result is somewhat surprising given the amount of information IM2CALORIES has available versus VOXPL's crowd, which only has access to images. VOXPL's calorie counter was trivial to write; the core function itself is only 10 lines of code, with the entire VOXPL program (including support code) consisting of 130 lines of code.

Facial Feature Locator

Our next application estimates the locations of facial features [19]. Finding facial features (e.g., the tip of the nose) is a component of face pose estimation, and important task in face recognition pipelines. Crowdsourcing is often utilized in machine learning specifically to build "ground truth" datasets for this kind of task; ensuring the high quality of training data is often important for the quality of the resulting model.

The application we built in VOXPL identifies the tip of a person's nose. This "nose finder" application asks workers to locate the "center of the tip of the nose" in 15 images of the faces of both people and animals taken from a Google Image search. Note that estimating the location of a nose is not a single estimation task; it is actually a *joint estimation* task since nose location combines both x and y information. The VOXPL code for this task is shown in Figure 5. Confidence intervals were set at ± 20 pixels, a tight bound.

Figure 1 presents sample photos and user responses with the 95% confidence region drawn as ellipses. In all 15 images, noses were correctly identified. As Figure 1(a) illustrates, confidence regions can be quite tight.

In response to the tightness of the requested confidence regions, VOXPL's quality control algorithm recruited a fairly large number of workers. VOXPL scheduled 1,356 tasks at an average cost of \$0.06 per task; the average cost to identify the nose per image was \$5.42. In all but two cases, the resulting confidence interval was tighter than ± 20 pixels (average x and y width of ± 17.01 and ± 17.47 pixels, respectively) with high, Bonferroni-adjusted confidence levels (average 98.7%). The two cases with looser bounds (x: ± 35.12, y: ± 36.6 pixels and x: ± 40.3, y: ± 43.96 pixels) exceeded the maximum budget of \$6.25. One latter case is the camel image shown in Figure 1(b); workers correctly identified its nose, but had some difficulty pinpointing the tip.

Our results demonstrate that VOXPL can be used to quickly collect high-quality ground truth data from images. This task took less than 50 minutes to run for 15 images, all of which were scheduled in parallel; the underlying runtime is capable of scaling to thousands of simultaneous tasks. We believe that this result compares favorably to the traditional method of generating ground-truth for such datasets: (1) either utilizing

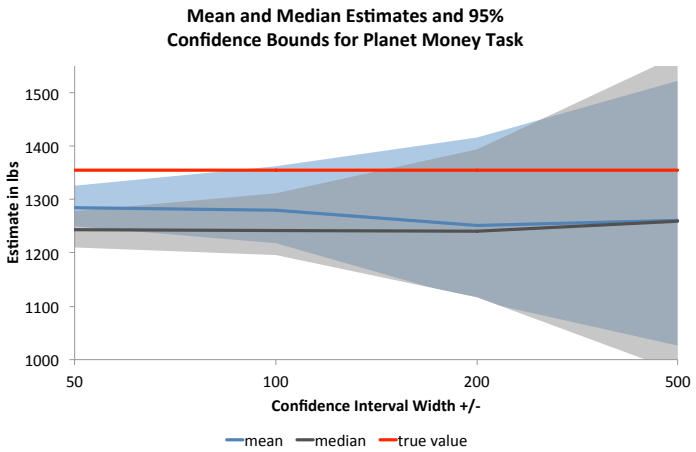


Figure 7. Observed confidence intervals at the 95% confidence level when using the mean or the median for estimating the mass of a cow. The shaded regions represent confidence intervals while the red line is the true value. When a user requests a tight constraint (x axis), VOXPL ensures that the resulting confidence interval is indeed tight (y axis). Results are the average of thirty simulations at each interval constraint.

workers to build them using *ad hoc* quality-control mechanisms, or (2) employing graduate students under a similar regime. VOXPL can build these datasets faster and more reliably (with user-defined error bounds); if desired, VOXPL can let programmers trade off looser error bounds for lower cost.

Estimating Mass

Our next application estimates the mass of an object using the crowd, automating Galton's canonical "wisdom of the crowd" study. Recall that Galton gathered the responses of participants in an ox-weighing contest; the median of the crowd's guesses was accurate within 0.8%. To Galton, this result suggested that expert knowledge could be acquired by aggregating the estimates of a large number of non-experts. Statisticians have since debated the cattle expertise of participants in that era. In his defense, Galton aptly observed that it was unlikely that all 800 participants were experts. Since their occupations were not recorded, the debate remains unsettled.

In August 2015, in a segment featuring New Yorker columnist James Surowiecki, the author of the book *The Wisdom of Crowds*, the National Public Radio program Planet Money sought to replicate the study themselves [24]. In contrast to the fair-goer of 1907, the typical Planet Money listener's life is likely far less agrarian: more than 80% reported that they had no special expertise in cattle. The show posted several photos of a cow, "Penelope," on their website and issued a call for participation. 17,184 people responded, with guesses ranging from 1 lb to 14,555 lbs. Penelope's true weight, obtained with a truck pull scale, was 1,355 lbs. The median of the Planet Money crowd, 1,245 lbs, was accurate to within 8.1%.

Planet Money supplied us with their data. Because of its size, the dataset allows us to explore a wide range of VOXPL confidence settings. Responses were provided by unpaid volunteers with little incentive to answer correctly, highlighting the value of quality control.

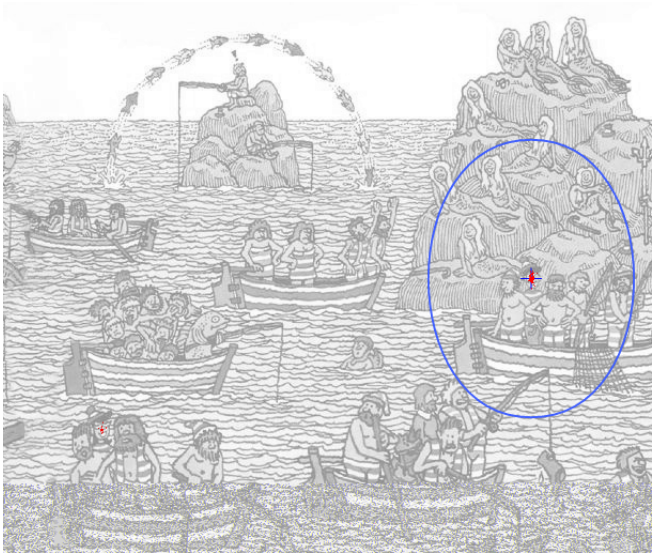


Figure 8. A difficult task for identifying objects in an image: a picture from the "Where's Waldo?" children's game. In every case, crowd workers found Waldo, although in some cases like the one shown here, the confidence region is large because a number of workers were fooled by the Waldo-lookalike on the left. In this case, VOXPL was able to provide a correct estimate, although the resulting confidence bounds did not satisfy the (tight) target constraint of ± 20 pixels.

We simulate Mechanical Turk responses by randomly sampling from the Planet Money data, without replacement. We compute equivalent MTurk workers costs (at a default rate of $7.25/hr, the U.S. Federal minimum wage) to explore cost-accuracy tradeoffs. Experiments explore four 95% confidence interval widths: ± 500, ± 200, ± 100, and ± 50. Each setting is repeated 30 times, and the results are averaged. While VOXPL's default estimator is the L_1 median, we also run these experiments with the arithmetic mean.

VOXPL achieves an effective cost-quality tradeoff for a variety of confidence interval widths. In all cases, estimates are remarkably good, especially given the wide variation in responses (see Figure 7). At its loosest confidence interval width (± 500), estimates using the mean had a mean absolute error (MAE) of 126.7 lbs and cost $0.72 on average; estimates using the median had a MAE of 118.2 lbs and cost $2.16 on average. At its tightest confidence interval width (± 50), estimates using the mean had a MAE of 71.0 lbs and cost $121.34 on average; estimates using the median had a MAE of 110.9 lbs and cost $89.09 on average. These results show that the choice of estimator can have a significant impact on cost depending on the stringency of the quality constraints; here, the median is more expensive for loose quality constraints, but cheaper for tighter constraints. However, this result depends on the distribution of responses, so this relationship is not predictable in advance.

Robust Location of Objects in an Image

Our final application asks workers to play the "Where's Waldo?" children's game (see Figure 8). Waldo is a character with a distinct set of features: a knit cap, eyeglasses, and a red and white striped shirt. The task is to locate Waldo in an image. The game can be surprisingly challenging because

images are often quite large, Waldo is usually lost among a sea of other people, and some of those other people are deliberately made to resemble him. While such a task is technically composed of two tasks—first *locating* characters and then *labeling* Waldo—this construction wastes resources because we do not need the locations of other characters. Instead, we ask workers to both *find* and *label* Waldo in a single step. In our experiment, we set a tight target constraint of ± 20 pixels (resulting in a circle around the point) and a confidence level of 95%; the maximum budget was $6.25.

As anticipated, workers performed poorly on this task; a number of workers were fooled by Waldo lookalikes. Nonetheless, VOXPL correctly located the real Waldo in every case. For all of the tasks, the resulting confidence regions are wider than the programmer's target constraint. All tasks cost $5.76. This result demonstrates that, despite the inherent difficulty of some tasks, VOXPL can trade off cost and quality to deliver the best estimate for the money while keeping the task under budget.

RELATED WORK

Our discussion first focuses on the most closely related work, AUTOMAN, since VOXPL builds on and significantly extends that system. We then discuss other related work, focusing on two of the key challenges identified by a recent position paper by Kittur et al. [25]: (1) how to ensure high quality, and (2) how to support complex workflows via programming frameworks. Finally, we discuss domain-specific work relating to the applications we use in our evaluation.

VOXPL modifications to AUTOMAN

AUTOMAN is a modular crowdprogramming framework with integrated automatic scheduling, payment, and quality control for any given multiple-choice question and confidence level [4, 5]. VOXPL generalizes and subsumes the AUTOMAN crowdprogramming framework, leveraging its automatic scheduling and pricing algorithms. VOXPL significantly extends AUTOMAN with a novel *combinator logic* and *estimation procedure*. The combinator logic guarantees that all VOXPL programs are statistically valid. The estimation procedure ensures that estimates meet quality constraints.

VOXPL estimation functions produce random variables representing arbitrary, empirical distributions. The L_1 median, confidence intervals, and the like are *inferences* about those distributions. When VOXPL estimates are composed (e.g, by adding them), the runtime transparently infers properties of the composed empirical distribution. AUTOMAN functions, by contrast, are not true random variables, and cannot be composed. Changing AUTOMAN to support VOXPL required a wholesale replacement of AUTOMAN's core algorithms, more than 5,000 lines of terse Scala code and one person-year of engineering effort. We describe the key modifications in the following sections.

Incorporating distributions for composability. The return type for AUTOMAN functions, a `Future[Answer]`, does not suffice for VOXPL. Critically, this return type does not represent a distribution. `Answer`s are the result of hidden aggregation operations. Using a single aggregated value as a response makes it impossible to compose the outcomes of two questions because the bootstrap requires access to both component distributions in order to compute confidence intervals.

To address this, we replaced AUTOMAN's `Future[Answer]` with an `Outcome`. An `Outcome` is essentially a probability monad, a datatype that represents probability distributions and ensures that all operations faithfully observe the laws of probability [41]. In addition, we hid concurrency-related `Future` datatypes from users as they complicate the use of the DSL, although we still rely on them extensively inside the runtime itself to manage task latencies.

Multiple comparisons. The core statistical procedure employed by AUTOMAN is subject to early termination bias. AUTOMAN optimistically schedules the minimum number of tasks that satisfy the user's constraints, but it schedules more when larger samples are needed. AUTOMAN should also update its stopping criteria, because incremental sampling implies a greater probability of stopping early by random chance. This phenomenon is known as the *multiple comparisons problem*. We incorporated the Bonferroni correction (see Correcting for Multiple Comparisons) into AUTOMAN's scheduler to control this probability.

Moving checking to the compiler. AUTOMAN relied on exceptions to handle events like exceeding the budget. Scala, AUTOMAN's host language, does not support *checked exceptions*, which means that failing to supply exception handlers is not a compile-time error. Unfortunately, this choice makes it easy for programmers to forget to handle these cases, leading to runtime failures. We ourselves found this to be a significant pain point while programming VOXPL tasks.

To address this problem, we completely replaced AUTOMAN's exception mechanism. Users now extract `Answer` values from `Outcome` distributions using functional pattern matching. All `Answer` subtypes (for estimation, etc.) are `sealed`, which means that they cannot be extended by the user [1]. `sealed` return types ensure that the compiler is aware of all possible VOXPL estimation return values. As a result, VOXPL leverages the compiler's *exhaustiveness check* to ensure that users handle exceptional conditions. Failing to so so produces a compilation error.

Other High-Level Programming Frameworks

Crowdsourcing program logic is easily obscured by the concerns introduced by mixing human and digital computation. TURKIT is an scripting system designed to make it easier to manage Mechanical Turk tasks [30]. CROWDFORGE provides language primitives for partitioning, mapping, and reducing tasks to form complex workflows [26]. CROWDDB and QURK integrate crowd calls with database queries [15, 33]. As with the quality control approaches mentioned above, these systems are limited to *labeling* or lack statistical guarantees. By contrast, VOXPL produces *estimates* with *error bounds*.

Quality Control

Labeling. Crowdsourcing has led to a resurgence in interest in quality control algorithms, but to date, research has focused primarily on producing accurate labels. Early on, researchers

noted that "inter-annotator agreement", also known as worker consensus, could improve the quality of crowdsourced labels. Many of the early approaches employ methods like majority vote that do not provide explicit quality guarantees [48, 43, 21, 34, 15, 40]. Later work takes a more statistical approach: many systems model latent variables like worker skill, worker speed, or task difficulty, or exploit correlations in the labels of similar inputs to predict accurate labels [9, 35, 29, 8, 10]. Several systems specifically address adversaries who may "game" the system by changing their behavior, treating the problem as a form of statistical noise rejection [4, 31, 27]. All of these approaches are designed with labeling in mind and do not provide quality guarantees for estimates.

Domain-specific estimation tasks. A number of systems address quality for specific instances of estimation problems. An algorithm based on the QURK platform provides a form of quality control for counting tasks [32]. Baba and Kashima use the crowd to bootstrap quality control for arbitrary tasks by asking the crowd itself for quality annotations [3]. While these systems do not provide quality guarantees, two alternative approaches do. Models borrowed from biology used to count animals can be repurposed to provide counts with confidence bounds [23]. The CROWD-MEDIAN algorithm, a nonparametric algorithm, can be used to find, e.g., "most representative" images, but is limited to computing centroids [17]. The Zooniverse suite of citizen science projects ask users to draw bounding boxes or mark locations on images, but they use domain-specific heuristics to handle quality.[2] VOXPL is designed to handle estimation generally, and can compute arbitrary statistics with quality guarantees.

Specific Applications

Calorie counting from images. PLATEMATE is an early approach that uses the crowd to analyze images [38]. While PLATEMATE can use the crowd to specifically estimate calories, it uses a fixed, ad hoc voting scheme that solicits five responses and then rejects outliers, returning the mean of the remaining responses. Machine learning based approaches include SRI's FIVR and Google's IM2CALORIES [39, 36]. FIVR requires multiple images, a comprehensive food database, a color calibration step, and speech annotations, and treats the problem as a form of volume estimation [39]. Google's IM2CALORIES represents the current state of the art, estimating calories from images by correlating them with items on restaurant menus [36].

Face recognition. EIGENFACES was the first practical automated face recoginition system, but cannot recognize faces when pose, illumation, and facial expressions are changed [47]. The current state of the art is Facebook's DEEPFACE system, which achieves near-human accuracy for *face verification* tasks (i.e., determining whether a photo is of a given person) [45]. Nonetheless, DEEPFACE needs a large training set (millions of labled images), and requires prior *face detection*. VOXPL lets programmers directly harness people's superior abilities at both face detection and verification [42].

[2]Private correspondence with the Zooniverse Principal Investigator, Chris Lintott.

CONCLUSION

Estimation is a natural task for crowd-powered systems. This paper presents VOXPL, a system that automatically generates statistically valid estimates from the crowd at low cost. VOXPL's declarative domain-specific language lets programmers concisely write applications based on these estimates. We show that VOXPL's quality control algorithms let programmers trade cost for accuracy, automatically generating the right statistical procedure to return crowd-generated results of the desired quality. We believe that VOXPL significantly extends the reach of crowdsourcing to encompass "wisdom of the crowd" estimation tasks, and that it has the potential to enable a new class of crowdsourcing-based applications.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CCF-1144520. The authors thank Joseph P. Price and his graduate students at Brigham Young University who gathered the photos and menus for the calorie dataset, and the NPR Planet Money team who gathered crowd guesses about Penelope the cow's weight. The authors also thank John Vilk for his assistance rendering confidence ellipsoids.

REFERENCES

1. 2016. *Scala Language Specification*. Technical Report Version 2.11. École Polytechnique Fédérale de Lausanne. https://www.scala-lang.org/files/archive/spec/2.11/

2. Greg Aloupis. 2001. *On Computing Geometric Estimators of Location (M.Sc Thesis)*. McGill University School of Computer Science.

3. Yukino Baba and Hisashi Kashima. 2013. Statistical Quality Estimation for General Crowdsourcing Tasks. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. ACM, New York, NY, USA, 554–562. DOI: http://dx.doi.org/10.1145/2487575.2487600

4. Daniel W. Barowy, Charlie Curtsinger, Emery D. Berger, and Andrew McGregor. 2012. AutoMan: A Platform for Integrating Human-based and Digital Computation. In *OOPSLA 2012*. 639–654. DOI: http://dx.doi.org/10.1145/2384616.2384663

5. Daniel W. Barowy, Charlie Curtsinger, Emery D. Berger, and Andrew McGregor. 2016. AutoMan: A Platform for Integrating Human-based and Digital Computation. *Commun. ACM* 59, 6 (May 2016), 102–109. DOI: http://dx.doi.org/10.1145/2927928

6. Gerald Van Belle. 2008. *Statistical Rules of Thumb* (second edition ed.). John Wiley & Sons, Inc., Hoboken, NJ, USA. DOI: http://dx.doi.org/10.1002/9780470377963.ch1

7. Peter J. Bickel and David A. Freedman. 1981. Some Asymptotic Theory for the Bootstrap. *The Annals of Statistics* 9, 6 (1981), 1196–1217. DOI: http://dx.doi.org/10.2307/2240410

8. Jonathan Bragg, Mausam, and Daniel S. Weld. 2013. Crowdsourcing Multi-Label Classification for Taxonomy Creation. In *Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2013, November 7-9, 2013, Palm Springs, CA, USA*. http://www.aaai.org/ocs/index.php/HCOMP/HCOMP13/paper/view/7560

9. Peng Dai, Christopher H. Lin, Mausam, and Daniel S. Weld. 2013. POMDP-based control of workflows for crowdsourcing. *Artificial Intelligence* 202 (Sept. 2013), 52–85. DOI: http://dx.doi.org/10.1016/j.artint.2013.06.002

10. Jia Deng, Olga Russakovsky, Jonathan Krause, Michael S. Bernstein, Alex Berg, and Li Fei-Fei. 2014. Scalable Multi-label Annotation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 3099–3102. DOI: http://dx.doi.org/10.1145/2556288.2557011

11. Olive Jean Dunn. 1961. Multiple Comparisons Among Means. *J. Amer. Statist. Assoc.* 56, 293 (1961), 52–64. DOI:http://dx.doi.org/10.1080/01621459.1961.10482090

12. B. Efron. 1979. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics* 7, 1 (1979), 1–26. DOI:http://dx.doi.org/10.2307/2958830

13. Bradley Efron. 1981. Nonparametric Standard Errors and Confidence Intervals. *Canadian Journal of Statistics* 9, 2 (1981), 139–158. DOI: http://dx.doi.org/10.2307/3314608

14. Bradley Efron. 1987. Better Bootstrap Confidence Intervals. *J. Amer. Statist. Assoc.* 82, 397 (1987), 171–185. DOI:http://dx.doi.org/10.1080/01621459.1987.10478410

15. Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. 2011. CrowdDB: Answering Queries with Crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*. ACM, New York, NY, USA, 61–72. DOI: http://dx.doi.org/10.1145/1989323.1989331

16. Francis Galton. 1907. Vox Populi. *Nature* 75, 1949 (March 1907), 450–451. http://www.nature.com/doifinder/10.1038/075450a0

17. Björn Hartman and Eric Horvitz (Eds.). 2013. *Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2013, November 7-9, 2013, Palm Springs, CA, USA*. AAAI. http://www.aaai.org/Library/HCOMP/hcomp13contents.php

18. Sture Holm. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics* 6, 2 (1979), 65–70. DOI: http://dx.doi.org/10.2307/4615733

19. Gary B. Huang, Manjunath Narayana, and Erik G. Learned-Miller. 2008. Towards unconstrained face recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2008, Anchorage, AK, USA, 23-28 June, 2008*. IEEE Computer Society, 1–8. DOI:http://dx.doi.org/10.1109/CVPRW.2008.4562973

20. John P. A. Ioannidis. 2005. Why Most Published Research Findings Are False. *PLoS Med* 2, 8 (08 2005), e124. DOI: http://dx.doi.org/10.1371/journal.pmed.0020124

21. Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. 2010. Quality Management on Amazon Mechanical Turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP '10)*. ACM, New York, NY, USA, 64–67. DOI: http://dx.doi.org/10.1145/1837885.1837906

22. Lilly C. Irani and M. Six Silberman. 2013. Turkopticon: Interrupting Worker Invisibility in Amazon Mechanical Turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 611–620. DOI: http://dx.doi.org/10.1145/2470654.2470742

23. Adam Kapelner. 2014. *Statistical Analysis and Design of Crowdsourcing Applications*. Ph.D. Dissertation. University of Pennsylvania. http://repository.upenn.edu/dissertations/AAI3622073

24. David Kestenbaum. 2015. Weighty Issue: Cow Guessing Game Helps To Explain The Stock Market. (15 Aug. 2015). http://www.npr.org/2015/08/20/432978431

25. Aniket Kittur, Jeffrey V. Nickerson, Michael Bernstein, Elizabeth Gerber, Aaron Shaw, John Zimmerman, Matt Lease, and John Horton. 2013. The Future of Crowd Work. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work (CSCW '13)*. ACM, New York, NY, USA, 1301–1318. DOI: http://dx.doi.org/10.1145/2441776.2441923

26. Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. 2011. CrowdForge: Crowdsourcing Complex Work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 43–52. DOI: http://dx.doi.org/10.1145/2047196.2047202

27. Ranjay A. Krishna, Kenji Hata, Stephanie Chen, Joshua Kravitz, David A. Shamma, Li Fei-Fei, and Michael S. Bernstein. 2016. Embracing Error to Enable Rapid Crowdsourcing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 3167–3179. DOI: http://dx.doi.org/10.1145/2858036.2858115

28. Daan Leijen and Erik Meijer. 1999. Domain Specific Embedded Compilers. In *Proceedings of the 2Nd Conference on Domain-specific Languages (DSL '99)*. ACM, New York, NY, USA, 109–122. DOI: http://dx.doi.org/10.1145/331960.331977

29. Christopher H. Lin, Mausam, and Daniel S. Weld. 2012. Crowdsourcing Control: Moving Beyond Multiple Choice. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, Nando de Freitas and Kevin P. Murphy (Eds.). AUAI Press, 491–500. https://dslpitt.org/papers/12/p491-lin.pdf

30. Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. 2010. TurKit: Human Computation Algorithms on Mechanical Turk. In *UIST 2010*. 57–66. DOI:http://dx.doi.org/10.1145/1866029.1866040

31. Benjamin Livshits and Todd Mytkowicz. 2014. Saving Money While Polling with InterPoll Using Power Analysis. In *Proceedings of the Seconf AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2014, November 2-4, 2014, Pittsburgh, Pennsylvania, USA*, Jeffrey P. Bigham and David C. Parkes (Eds.). AAAI. http://www.aaai.org/ocs/index.php/HCOMP/HCOMP14/paper/view/8979

32. Adam Marcus, David Karger, Samuel Madden, Robert Miller, and Sewoong Oh. 2012. Counting with the Crowd. *Proc. VLDB Endow.* 6, 2 (Dec. 2012), 109–120. DOI: http://dx.doi.org/10.14778/2535568.2448944

33. Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. 2011. Crowdsourced Databases: Query Processing with People. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*. www.cidrdb.org, 211–214. http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper29.pdf

34. Winter Mason and Siddharth Suri. 2012. Conducting behavioral research on Amazon's Mechanical Turk. *Behavior Research Methods* 44, 1 (2012), 1–23. DOI: http://dx.doi.org/10.3758/s13428-011-0124-6

35. Luyi Mo, Reynold Cheng, Ben Kao, Xuan S. Yang, Chenghui Ren, Siyu Lei, David W. Cheung, and Eric Lo. 2013. Optimizing plurality for human intelligence tasks. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management (CIKM '13)*. ACM, New York, NY, USA, 1929–1938. DOI:http://dx.doi.org/10.1145/2505515.2505755

36. Austin Myers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alexander N. Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin Murphy. 2015. Im2Calories: Towards an Automated Mobile Vision Food Diary. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 1233–1241. DOI: http://dx.doi.org/10.1109/ICCV.2015.146

37. National Institute of Standards and Technology / SEMATECH. 2013. *NIST/SEMATECH e-Handbook of Statistical Methods*. http://www.itl.nist.gov/div898/handbook

38. Jon Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z. Gajos. 2011. Platemate: Crowdsourcing Nutritional Analysis from Food Photographs. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 1–12. DOI:http://dx.doi.org/10.1145/2047196.2047198

39. M. Puri, Zhiwei Zhu, Q. Yu, A. Divakaran, and H. Sawhney. 2009. Recognition and Volume Estimation of Food Intake Using a Mobile Device. In *Applications of Computer Vision (WACV), 2009 Workshop on*. 1–8. DOI: http://dx.doi.org/10.1109/WACV.2009.5403087

40. Alexander J. Quinn and Benjamin B. Bederson. 2011. Human-Machine Hybrid Computation. In *CHI 2011 Workshop On Crowdsourcing And Human Computation*. http://crowdresearch.org/chi2011-workshop/papers/quinn.pdf

41. Norman Ramsey and Avi Pfeffer. 2002. Stochastic Lambda Calculus and Monads of Probability Distributions. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '02)*. ACM, New York, NY, USA, 154–165. DOI: http://dx.doi.org/10.1145/503272.503288

42. Pawan Sinha, Benjamin Balas, Yuri Ostrovsky, and Richard Russell. 2007. Face Recognition by Humans: Nineteen Results All Computer Vision Researchers Should Know About. *Proc. IEEE* 94, 11 (2007), 1948–1962. DOI: http://dx.doi.org/10.1109/JPROC.2006.884093

43. Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. 2008. Cheap and Fast - But is it Good? Evaluating Non-Expert Annotations for Natural Language Tasks. In *2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 254–263. http://www.aclweb.org/anthology/D08-1027

44. James Surowiecki. 2005. *The Wisdom of Crowds*. Anchor.

45. Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '14)*. IEEE Computer Society, Washington, DC, USA, 1701–1708. DOI:http://dx.doi.org/10.1109/CVPR.2014.220

46. Bradley Efron Thomas J. DiCiccio. 1996. Bootstrap Confidence Intervals. *Statist. Sci.* 11, 3 (1996), 189–212. DOI:http://dx.doi.org/10.1214/ss/1032280214

47. Matthew A. Turk and Alex Pentland. 1991. Face Recognition Using Eigenfaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 1991, 3-6 June, 1991, Lahaina, Maui, Hawaii, USA*. IEEE, 586–591. DOI: http://dx.doi.org/10.1109/CVPR.1991.139758

48. Luis von Ahn and Laura Dabbish. 2004. Labeling Images with a Computer Game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 319–326. DOI: http://dx.doi.org/10.1145/985692.985733

49. L. Wasserman. 2006. *All of Nonparametric Statistics*. Springer New York.