Active Learning for Crowd-Sourced Databases

Barzan Mozafari‡ Purnamrita Sarkar‡ Michael Franklin‡ Michael Jordan‡ Samuel Madden‡

† CSAIL MIT ‡ {barzan, maddan}@csail.mit.edu
‡ UC Berkeley ‡ {psarkar, franklin, jordan}@cs.berkeley.edu

ABSTRACT
Crowd-sourcing has become a popular means of acquiring labeled data for a wide variety of tasks where humans are more accurate than computers, e.g., labeling images, matching objects, or analyzing sentiment. However, relying solely on the crowd is often impractical even for datasets with thousands of items, due to time and cost constraints of acquiring human input. In this paper, we propose algorithms for integrating machine learning into crowd-sourced databases, with the goal of allowing crowd-sourcing applications to scale, i.e., to handle larger datasets at lower costs. The key observation is that, in many of the above tasks, humans and machine learning algorithms can be complementary, as humans are often more accurate but slow and expensive, while algorithms are usually less accurate, but faster and cheaper.

Based on this observation, we present two new active learning algorithms to combine humans and algorithms together in a crowd-sourced database. Our algorithms are based on the theory of non-parametric bootstrap, which makes our results applicable to a broad class of machine learning models. Our results, on three real-life datasets collected with Amazon’s Mechanical Turk, and on 15 well-known UCI data sets, show that our methods on average ask humans to label one to two orders of magnitude fewer items to achieve the same accuracy as the baseline (that randomly chooses which items to be labeled by the crowd), and two to eight times fewer questions than previous active learning schemes.

1. INTRODUCTION
Crowd-sourced market places like Amazon’s Mechanical Turk have made it easy to recruit a crowd of people for performing tasks that are difficult for computers, including producing entity resolution [17, 25, 22], audio transcription, image annotation [6], sentiment analysis [24], and document summarization and editing [5]. Many of these can be thought of as database problems, where each item is a row in a table with some missing attributes (labels) that need to be supplied by crowd workers, giving rise to a new generation of database systems, known as crowd-sourced databases [13] [17, 25, 22].

In this paper we focus on the particular problem of labeling, e.g., using a classifier to assign one of several discrete values to a data item, as opposed to the problem of producing missing data [22], or using regression to predict numeric values. Although humans are often better than machines at such labeling tasks, using humans for processing tens of thousands to tens of millions of documents or images is not cost-effective (or even feasible when the task occurs often enough, say for images uploaded each day). This is because acquiring a single label can cost several cents and take several minutes. In this paper, we advocate combining humans and machine learning algorithms to tackle this problem in two ways: (i) given a data set with data that needs to be labeled, identify data items (e.g., images or tweets) that are inherently difficult for the classifier, and ask for labels for those items from the crowd, processing other items with the algorithm and (ii) use humans to generate training data to supply to the classifier, which can then be used to complete the remaining tasks significantly faster and cheaper.

The primary challenge in such a setting is determining which items to label with the crowd (we call these “questions to ask the crowd” throughout the paper). Specifically, given a database of unlabeled data, and a classifier that, once trained sufficiently, can attach a label to each unlabeled item, what is needed is a way to determine which and how many questions we should ask the crowd to hand-label in order to achieve the best training data and overall accuracy. There are two aspects to this problem: (i) determining which items, if labeled, would be more beneficial for training, and (ii) determining which items are inherently hard for the classifier, no matter how much training data is provided. Our goal is to find an algorithm that can accurately and efficiently estimate these quantities, and then to use those estimates to optimally allocate a (time or dollar) budget to acquire labels from the crowd, in order to achieve the best overall cost or quality.

Our problem is similar to the classical problem of active learning in machine learning, where the goal is to select statistically optimal training data [8], and which has also been explored in the context of data integration [16] and cleaning [29]. Developing active learning algorithms for crowd-sourced databases introduces a number of new challenges that were generally not faced in traditional active learning literature, specifically:

- **Higher degree of noise.** Traditional active learning mostly deals with expert-provided labels that are assumed to be the ground-truth (notable exceptions are agnostic active learning approaches [3]). In contrast, crowd-provided labels are subject to a much higher degree of uncertainty, e.g., innocent errors due to typos or abbreviations, lack of enough domain knowledge, and even deliberate mistakes by spammers, none of which have been a great source of concern in active learning with expert-provided labels [29].

- **Generality.** The active learning literature has mostly dealt with learning problems where a specific task is addressed in a well-understood domain (e.g., for SVMs in text mining [31]). While there are active learning techniques that are designed for generic classifiers, this generality requirement is more stringent in a crowd-sourced database setting: users issue a broad range of queries, crowd workers exhibit a wide range of behaviors, and the system must be able to support arbitrary, user-supplied classification algorithms.

- **Scalability.** Many crowd-sourcing scenarios involve web-scale data such as making sense out of tens of millions of daily tweets, images, videos and blog posts. Here the two main aspects to scalability are: (i) human-scalability which involves limiting the number of tasks posed to the crowd, and (ii) machine-scalability which involves limiting the training overhead of the classifier, since training sophisticated classifiers can be quite slow.

- **Ease-of-use.** Crowd-enabled workflows and systems are used by a broad range of users. As a result, our integrated learning al-
algorithms need to be simple to use, require minimum supervision from the user, and must be able to automatically handle a number of important optimization questions, such as: When to stop the active learning process? What is the required redundancy in questions asked of the crowd when acquiring each label? What is the optimal batch size for meeting the time/quality/cost requirements of the user?

We address these challenges in two active-learning regimes: an upfront setting, where after an initial training phase, all of the items to be sent to the crowd are picked in one iteration, and the rest of the items are labeled by our classifier, and an iterative setting, where in each iteration we classify a few items at a time with the crowd, use those answers to improve the quality of our classifier, and repeat until we have exhausted our budget. We then label the remaining items with the classifier.

Our solution to these problems consists of two novel active learning algorithms, called MinExpError and Uncertainty. MinExpError combines the current quality (say, accuracy) of the classifier with an unbiased estimate of its uncertainty in a mathematically sound way, in order to choose the best question to ask next. We have also designed the Uncertainty algorithm, which only estimates the uncertainty of the classifier, and hence, has a lower computational overhead. However, Uncertainty has lower overall accuracy than MinExpError, especially in the upfront setting.

Both Uncertainty and MinExpError are based on (non-parametric) bootstrap [11], which produces unbiased estimates of uncertainty for an estimator (e.g., parameters learned by a classifier). Bootstrap repeatedly re-computes the given estimator on bootstrap replicates (samples drawn with replacement from the training data) and measures the variability of the estimator across these replicates. Based on these estimates of variability or confidence, our algorithms decide on which items to ask the crowd.

Once MinExpError or Uncertainty decide on which items to ask the crowd about, the next challenge is how to deal with the noise that is inherent in crowd-provided labels. A common practice is to use redundancy, i.e., to ask each question from more than one worker. However, instead of applying the same degree of redundancy to all items, we have developed another algorithm, called partitioning-based allocation or PBA, which partitions the unlabeled items based on their degree of difficulty for the crowd. PBA uses integer linear programming to decide on how many redundant labels to request for each item.

We have run a number of experiments to study the effectiveness of our algorithms, using 3 crowd-sourced and 15 well-known real-life datasets. The experiments show that, for the same quality requirements and compared to choosing data items from the pool of unlabeled data at random, our algorithms on average reduce the number of questions by $7 \times$ in the iterative scenario and more than $100 \times$ in the upfront scenario. Also, compared to previous state-of-the-art active learning techniques, our algorithms ask $2 \times$ to $12 \times$ fewer questions on average.

Before describing our algorithms and experiments in detail, we first describe our overall approach.

2. OVERVIEW OF APPROACH

The high level approach we take in this paper is as follows: the user provides (i) a pool of unlabeled items (possibly with some labeled items as initial training data), (ii) a classifier (or “learner”) that improves with more and better training data and a specification as to whether learning should be upfront or iterative, and (iii) a budget or goal, in terms of time, quality, or cost (and a pricing scheme for paying the crowd). For most of our experiments in this paper, we either use Amazon Mechanical Turk for crowd answers, or use a fully (hand) labeled data set where we hide the labels from the classifier until the learning algorithm chooses to observe them.

Our system uses the upfront or iterative active learning methods described in the next subsection to train the classifier and label items. At the core of these methods are a ranking and selection algorithm that choose the best items to ask the crowd to label. We have developed two algorithms for this purpose, called Uncertainty and MinExpError, described in Section 3.

Asking questions from the crowd raises a number of interesting questions, including how to best employ redundancy to minimize crowd error, and how many questions should be asked at one go from the crowd. We discuss these issues in Section 4.

We then compare the performance of our MinExpError and Uncertainty algorithms in the upfront and iterative settings on a number of real world data sets in Section 5.

2.1 Active Learning Regimes

As noted above, we consider two learning settings in this paper, the upfront setting, and the iterative setting. In this section we describe how learning in these two regimes works. Suppose we have a given budget (e.g., of money, time, or number of questions) $B$ for asking questions or a quality requirement $Q$ (e.g., required accuracy or F1-measure) for our classifier that we need to achieve. Given a set of unlabeled item, we use one of two basic strategies to acquire labels from the crowd: In the upfront strategy, we train our model on the available labeled data, and based on this select unlabeled items to ask the crowd to label, while the classifier labels the other items in parallel. The final result consists of the union of these two labeled sets. In the iterative approach, we use the crowd to label a few items, and add those labels to the existing training set, retrain, choose a new set of unlabeled items, and then iterate until we have exhausted our budget or met our quality goal (e.g., by using cross-validation on the training data).

The upfront scheme involves less training overhead as we do not have to repeatedly retrain our model and recompute the scores on the unlabeled items, and then wait for the crowd’s response. However, the iterative scheme can adaptively adjust its scores in each iteration, and thus achieve a smaller error than the upfront scheme for the same budget. In contrast, the upfront scheme has to choose all the items it wants to label at once, based only on an initial (limited) set of labeled data. Pseudocode for these two scenarios is given in Figures 1 and 2. As noted above, the budget $B$, set of data items $(L_0$ and $U_0)$, and learning regime (upfront vs iterative) are supplied by the user. In this paper, we compare the performance and behavior of different ranker strategies ($R$) Uncertainty and MinExpError with several different classification algorithms ($\theta$) such as SVM and decision trees. For ease of presentation, in the description of our approach, we assume binary classification, i.e., $\theta \in \{0, 1\}$. However, our algorithms work for general classifiers.

Note that the upfront scenario is not a special case of the iterative one because in the upfront case we do not include the crowd-sourced labels when training the classifier. This difference is important as some applications may prefer the upfront scenario over the iterative one. For instance, when early results are strictly preferred, the model has to be invoked as soon as possible to return labels for as much of the data as possible, without having to wait for the crowd’s answer. Another example is when stringent requirements are imposed as to only allow for gold standard data to be used.
Upfront Active Learning ($Q$ or $B, L_0, U, \theta, R, S$)

**Input:** $Q$ is the quality requirement (e.g., minimum accuracy, F1-measure), $B$ is the total budget (money, time, or # of questions), $L_0$ is the initial labeled data, $U$ is the unlabeled data, $\theta$ is a classification algorithm to (imperfectly) label the data, $R$ is a ranker that gives effectiveness scores to unlabeled instances, $S$ is a selection strategy (which labels need to come from crowd).

**Output:** $L$ is the labeled version of $U$.

1. $CL \leftarrow \emptyset$ // labeled data acquired from the crowd
2. $L \leftarrow \theta_0(U)$ // train $\theta$ on $L_0$ & invoke it to label $U$
3. $W \leftarrow R(U, \theta)$ // $w_i \in W$ is the effective score for $u_i \in U$
4. If under a certain budget $B$
5. Choose $U' \subseteq U$ based on selection strategy $S(U, W)$ and budget $B$
6. $CL \leftarrow$ Ask the crowd to label $U'$
7. $ML \leftarrow \theta_0(U - U')$ //train $\theta$ on $L_0$ to label remaining $U - U'$
8. Else, //under a certain quality requirement $Q$
9. Choose $U' \subseteq U$ based on $S(U, W)$ such that $\theta_0(U')$ satisfies $Q$
10. $ML \leftarrow \theta_0(U')$
11. $CL \leftarrow$ Ask the crowd to label the remaining $U - U'$
12. $L \leftarrow CL \cup ML$ // combine crowd and machine provided labels

**Figure 1:** The upfront scenario in active learning.

Iterative Active Learning ($Q$ or $B, L_0, U, \theta, R, S$)

**Input:** $Q$ is the quality requirement (minimum accuracy, F1-measure), $B$ is the total budget (money, time, or # of questions), $L_0$ is the initial labeled data, $U$ is the unlabeled data, $\theta$ is a classification algorithm to (imperfectly) label the data, $R$ is a ranker that gives effectiveness scores to unlabeled instances, $S$ is a selection strategy (which labels need to come from crowd).

**Output:** $L$ is the labeled version of $U$.

1. $CL \leftarrow \emptyset$ // labeled data acquired from the crowd
2. $L \leftarrow \theta_0(U)$ // train $\theta$ on $L_0$ & invoke it to label $U$
3. While $L$’s quality does not meet $Q$ and our budget $B$ is not exhausted:
4. $W \leftarrow R(U, \theta)$ // $w_i \in W$ is the effective score for $u_i \in U$
5. Choose $U' \subseteq U$ by applying the selection strategy $S$ on $w_i$ scores
6. $CL \leftarrow$ Ask the crowd to label $U'$
7. $CL \leftarrow CL \cup L'$, $U \leftarrow U - U'$ // remove crowd-sourced labeled from $U$
8. $L \leftarrow CL \cup \theta_0(L \cup CL)$ // train $\theta$ on $L_0 \cup CL$ to label remaining $U$

**Figure 2:** The iterative scenario in active learning.

for training (whereas crowd-sourced labels are often noisy).

### 3. Ranking Algorithms

In this section, we describe our ranking algorithms (which produce scores), called Uncertainty and MinExpError. Both methods are based on bootstrap, which they use to estimate the benefit (in terms of improving the model’s accuracy once the label is known) of having the crowd label an unlabeled data point in our input dataset. Either algorithm can be used as the ranker ($R$ in Figure 1 and 2). Once we have the scores from a ranker, we use biased sampling (that favors high scores) to choose a batch of unlabeled data-points to be sent to the crowd for labeling; here the probability of choosing each unlabeled instance is proportional to its score. Hence, our choice of selection strategy ($S$ in Figure 1 and 2) is weighted sampling.

The scores produced by both rankers are based on the nonparametric bootstrap [11], a powerful statistical technique traditionally used for estimating the uncertainty of estimators. The main idea is simple and is based on the “plug-in” principle: use the empirical distribution of the data (say $D$) as a proxy for the true underlying unknown distribution (denoted by $D$). If we observe $n$ i.i.d. sampled items $(x_1, \ldots, x_n)$ drawn from an unknown probability distribution $D$, then the empirical distribution $\hat{D}$ is defined as a discrete distribution which puts a probability mass of $1/n$ on each value of $x_i$, $i = 1, \ldots, n$. Say we want to estimate a parameter $\theta = \hat{t}(D)$. The “plug-in” principle simply estimates $\theta$ by $\hat{\theta} = \hat{t}(\hat{D})$. In particular, bootstrap can be used to compute the bias and the variance of $\hat{\theta}$. The beauty of bootstrap is that it computes these quantities automatically for a wide class of estimators $\hat{\theta}$. First we create $m$ bootstrapped datasets $\hat{x}^k$, $k = 1, \ldots, m$, where each $\hat{x}^k = \{\hat{x}^k_1, \ldots, \hat{x}^k_n\}$ is constructed by drawing $n$ i.i.d. items with replacement from the observed dataset (note that these are the same size as the observed dataset). For each sample, we compute $\hat{\theta}^k$. Now the variance of $\hat{\theta}$ is estimated by the sample variance of $\{\hat{\theta}^k, k = 1, \ldots, m\}$. The bias can also be computed by a simple plug-in principle.

The main advantage of bootstrap is two-fold: first, uncertainty can be estimated without making any simplifying assumption about the underlying distribution and second, the estimator can be arbitrary, as long as some smoothness assumptions are satisfied. In particular, bootstrap can be used for a large class of machine learning algorithms that can be framed as $M$-estimators, e.g., maximum likelihood estimators [20]. Therefore, by revisiting similar powerful theoretical results from classical nonparametric statistics, not only can we estimate the uncertainty of complex estimators, but we can also scale the computation up to big volumes of data for the following reasons. First, being able to estimate the uncertainty of an estimator (e.g., probability of correctly classifying a point) allows us to stop invoking the crowd, once we are confident enough. Second, each sample $\hat{x}^k$ can be processed independently, in parallel, allowing us to parallelize the computation across multiple processors or cores. With the exception of Provost et al. [27] (which works only for probabilistic classifiers), no attention has been given to exploiting the power of statistical bootstrapping in active learning, perhaps due to the computational overhead of bootstrap. But with recent advances in parallelizing bootstrap computation [18] as well as the rapid increase in RAM sizes and the number of cores on modern servers, bootstrap has now become a much more computationally viable approach, motivating our use of it in this paper.

#### 3.1 Uncertainty Algorithm

The goal of our Uncertainty algorithm is to estimate the confidence (or uncertainty) of a given classifier $\theta$ on each of the unlabelled items. Suppose, for the moment, that we have the underlying distribution $D$ over all the items and their true labels. Let the true label of item $u$ be $Y_u$. Having access to $D$, we could easily draw many datasets $\{L_i, i = 1, 2, \cdots \}$ from distribution $D$, each having the same size as our original training data; then train our classifier on each of these $L_i$’s and predict $u$’s label, denoted as $\theta(L_i, u)$. Now by definition, the true variance of this classifier on $u$ would be simply the variance of these labels. Unfortunately, in reality $D$ is not available to us. Hence, we use the empirical distribution instead. In other words, we bootstrap the training data multiple (say $m$) times to obtain $m$ different classifiers which are then invoked to generate the label of a given test datapoint. Since the bootstrap samples simply emulate the process of sampling data points from the original distribution, these bootstrapped labels can be thought of as labels given by classifiers learnt on different training datasets directly sampled from the true distribution. We can then compute the variance of these $m$ bootstrapped labels. By bootstrap theory [11], this is guaranteed to quickly converge to the true variance as we increase $m$. For example, in our experiments we found that $m = 10$ provides good estimates of the variance (we use $m = 10$ in all our experiments in Section 5).

From a classification point of view, the variance of the classifier’s
answer about a given item is important because the higher it is, the more likely the classifier is to mis-label the item. Therefore, once we compute the variance for each unlabeled item, we select items with larger variance (the probability of an item being chosen will be proportional to its variance) and ask the crowd to label them. The number of items selected in each iteration and the number of iterations depends on the budget, user requirements and whether we are operating in the upfront or iterative scenario. We return to the question of batch size in Sections 3.1 and 4.3

Formally, our notion of uncertainty is simply the variance of the classifier in its predictions, as follows.

Let \( L^*_k \) denote the \( k \)-th bootstrap, and \( \hat{l}_u \) be the prediction of our classifier for \( u \) when trained on this bootstrap. Also, let \( R(u) := \sum_{k=1}^m \hat{l}_u/m \). Since \( \hat{l}_u \in \{0, 1\} \), the uncertainty score for instance \( u \) is given by its variance, which is:

\[
\text{Uncertainty}(u) = R(u)(1 - R(u))
\]

Note that the power of using bootstrap is that we estimated \( R(u) \) and hence the variance without any assumptions on the classification algorithm, except that (due to the use of bootstrap) \( \theta \) must be relatively smooth (Hadamard differentiable), a property which holds for a large class of existing learning algorithms [20]. Thus, while many active learning proposals have tried to capture the uncertainty or variance of classifiers, they are either model-class specific (e.g., define the uncertainty of an SVM as the point’s distance from the separator margin [33]), assume probabilistic classifiers that produce highly accurate class probability estimates [34], or are simply heuristics that do not guarantee an unbiased estimate of the classifier’s variance. In contrast, our Uncertainty score applies to both probabilistic and non-probabilistic classifiers, and is guaranteed by bootstrap theory to provide an unbiased estimate of the variance.

The intuition behind why our Uncertainty algorithm can reduce the overall error with fewer labels is inspired by the results from Kohavi and Wolpert [19] where they show that the classification error for item \( u \) can be decomposed into the sum of three terms: (i) the classifier’s variance, \( \text{var}_{KW}(u) \), (ii) the classifier’s bias, \( \text{bias}_{KW}(u) \), and (iii) a noise term, \( \sigma^2(u) \), which is an error inherent to the data collection process, and thus cannot be reduced. Using our notation, \( \text{var}_{KW}(u) \) will be \( R(u)(1 - R(u)) \), and the squared bias is defined as \( [f(u) - R(u)]^2 \), where \( f(u) = E[l_u|u] \), i.e., the expected value of the true label given \( u \). Hence by asking for the label of \( u \) with large variance we are indirectly reducing the classification error. However, this is not all we can do. Bootstrap can also be used to reduce bias [11]. But, instead of estimating both bias and variance separately using bootstrap, we can directly estimate the classification error using bootstrap by simply comparing the bootstrapped labels \( \hat{l}_u \) to the label generated by a classifier trained on the original training data. This is used in our second algorithm, described next.

### 3.2 MinExpError Algorithm

The Uncertainty score identifies items that are most ambiguous or hardest for the classifier, and asks those from the crowd instead. However, it may also be a good idea to ask questions that the classifier is not uncertain about, which would have the largest impact on the classifier’s output. Hence we propose a new score which naturally combines both of these strategies.

The idea is that we want to get a label for an item which is most likely to minimize the overall training error. If we knew that the current model’s prediction \( \hat{l}_u \) was correct, then we could estimate the training error by adding \( \{u, \hat{l}_u\} \) to the existing training data \( L \) and retraining the model. Denote the error of this model by \( \epsilon_{\text{right}} \).

On the other hand, if we knew that the predicted label was incorrect, we could add \( \{u, 1 - \hat{l}_u\} \) to the training set and retrain the model. Denote the error of this model with \( \epsilon_{\text{wrong}} \). The problem of course is that (i) we do not know what the true label is, and (ii) we do not know the exact error of each model. Solving (ii) is relatively easy: we assume those labels and use cross validation on our already labeled data to estimate both errors, say \( \hat{\epsilon}_{\text{right}} \) and \( \hat{\epsilon}_{\text{wrong}} \). To solve the first problem, we can again bootstrap our current training data to estimate the probability that our current prediction \( \hat{l}_u \) is correct.

Following the same notation used in Section 3.1, denote \( \hat{l}_u \) := \( \theta(L^*_u, u) \), and let \( l_u \) be the true label. Thus, our goal is to estimate \( p(u) := Pr[l_u = l_u|u] \), as follows:

\[
\hat{p}(u) = \frac{\sum_{k=1}^m 1(\hat{l}_u = \hat{l}_u)}{m}
\]

Here, \( 1(c) \) is the decision function which evaluates to 1 when condition \( c \) holds and to zero otherwise. Intuitively, equation (2) says that the probability of the model’s prediction being correct can be estimated by the fraction of classifiers that agree on that prediction, if those classifiers are each trained on a bootstrap of the training set.

Thus \( \hat{p}(u) \) estimates the classification accuracy of the model for point \( u \). Now, we can compute the expected error of the model if we incorporated \( u \) in the training data (i.e., asked the crowd to label it for us) by averaging over its label choices.

\[
\text{MinExpError}(u) = \hat{p}(u)\epsilon_{\text{right}} + (1 - \hat{p}(u))\epsilon_{\text{wrong}}
\]

Since we want to sample such that \( u \) with a small \( \text{MinExpError} \) score is more likely to be picked, we define the score from which we will draw samples of \( u \) as:

\[
\text{ExpTrainingAccuracy}(u) := 1 - \text{MinExpError}(u)
\]

Without loss of generality, assume that the quantity \( \hat{\epsilon}_{\text{right}} - \hat{\epsilon}_{\text{wrong}} \) is non-negative. An analogous decomposition is possible when it is negative. We can break down the left hand side of equation (4), i.e. ExpTrainingAccuracy as:

\[
\hat{p}(u)(1 - \hat{\epsilon}_{\text{right}}) + (1 - \hat{p}(u))(1 - \hat{\epsilon}_{\text{wrong}}) = (1 - \hat{p}(u))(\hat{\epsilon}_{\text{right}} - \hat{\epsilon}_{\text{wrong}}) + (1 - \hat{\epsilon}_{\text{right}})
\]

Note that the \( 1 - \hat{\epsilon}_{\text{right}} \) term is non-negative, and hence \( (1 - \hat{p}(u))(\hat{\epsilon}_{\text{right}} - \hat{\epsilon}_{\text{wrong}}) \) is a lower bound on ExpTrainingAccuracy. Thus this naturally combines both the hardness of the question and how much knowing its answer can improve our classifier. In other words, equation (5) tells us that if the question is too hard (large \( 1 - \hat{p}(u) \)), even if its answer does not affect our ability in classifying other items, we may still choose to ask the crowd to label it to avoid a high risk of misclassification on \( u \). On the other hand, we may ask a question for which our model is fairly confident (small \( 1 - \hat{p}(u) \)), but having its true label can still make a big difference in classifying other items, namely, the value of \( \epsilon_{\text{wrong}} \) is much smaller than \( \hat{\epsilon}_{\text{right}} \). This means that, however unlikely, if our model happens to be wrong, we will have a higher overall accuracy and a lower overall error if we ask for the true label of \( u \).

Empirically, we found that a simple heuristic that adds some constant \( c > 0 \) to ExpTrainingAccuracy for all \( u \in U \) and renormalizes this vector, gives us a considerable edge over other methods. We have used \( c = 1 \) in all of our experiments in this paper. Intuitively, this is biasing the original distribution towards the uniform distribution while preserving the peaks in the original distribution. In Section 5 we show that our scoring algorithms deliver a significant boost in the quality of a passive learner, which picks unlabeled points uniformly at random.
3.3 Complexity and Scalability

Training machine learning algorithms is often a CPU intensive task. Training active learning algorithms is even more expensive, as some form of retraining is usually required after each new example is received. In this section, we describe how our choice of bootstrap for active learning allows us to achieve scalability, while preserving the generality criterion of our models. The primary benefit of bootstrap is that it is easily parallelizable. This means each bootstrap can be shipped to a different node or processor, which can perform training in parallel. Only at the end will the output (labels) of each instance be sent to a single node that can perform some light weight integration of results. For instance, in the case of our Uncertainty algorithm, the final node will only have to estimate the variance among individual labels and perform a weighted sampling to decide on the next batch of questions to ask from the crowd.

The time complexity of each iteration of our Uncertainty algorithm is $O(m \cdot T(|U|))$ where $|U|$ is the number of unlabeled data points in that iteration, $T(\cdot)$ is the training time of the classification algorithm (e.g., this is cubic in the input size for training SVMs), and $m$ is the number of bootstraps. Hence, in each iteration, we only need $m$ nodes to achieve the same run-time as training a single instance of the classifier. In Section 5.4 we study the effect of changing $m$ on the overall quality of our active learning algorithm.

Our other algorithm, MinExpError, is computationally more demanding as it involves a case analysis for each unlabeled point. The time complexity of each iteration of our MinExpError algorithm is $O((m + |U|) \cdot T(|U|))$. Here the algorithm is still parallelizable, since each unlabeled instance can be analyzed independently. However, MinExpError will involve a higher number of nodes, e.g., to achieve the same performance as Uncertainty, we need $O(|U|)$ nodes in the cluster. Note that this additional overhead is often justified since MinExpError is designed to deal with the upfront learning scenario where the job of the active learner is much more challenging. The challenge arises from the fact that it only gets to decide on which questions to ask based on the limited set of initial labeled data.

4. OPTIMIZING FOR THE CROWD

The previous sections described our two learning scenarios (up-front and iterative), and presented our learning algorithms. In this section, we describe several practical considerations that arise when asking questions with an actual crowd. Specifically, we address the following issues: How do we deal with the fact that the crowd does not always answer questions correctly (Section 4.1)? How do we know when our accuracy is “good enough” (Section 4.2)? Given that we can issue many queries in parallel to the crowd, what is the effect of batch size (number of simultaneous questions) on our learning performance (Section 4.3)? We deal with each of these questions in the next three subsections.

4.1 Handling Crowd Uncertainty

Optimizing Redundancy for Subgroups: Most previous active learning approaches have assumed that labels are provided by domain experts and hence, are perfectly correct (see Section 5). In contrast, in a crowd database, the crowd may produce answers that are incorrect, noisy and sometimes even adversarial. The conventional way to handle this is to use redundancy, e.g., to ask multiple workers to provide answers, and combine their answers to get the best overall result. Standard techniques, such as asking for multiple answers and using majority vote or the techniques of Dawid and Skene (DS) can improve answer quality when the crowd is mostly correct, but will not help much if users disagree more than they agree or when they converge to the right answers but too slowly. In our experience, for some classification datasets, crowd workers can be quite imprecise. For example, we took 1000 tweets with hand-labeled (“gold data”) sentiment (dataset details in Section 5.1.3), removed these labels and asked Amazon Mechanical Turk workers to label them again and measured the workers’ agreement. We used different redundancy ratios (1, 3, 5) and different voting schemes (majority and Dawid and Skene (DS)) and computed the ability of the crowd to agree with the hand-produced labels. The results are shown in Table 1.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Majority Vote</th>
<th>Dawid and Skene</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 worker per label</td>
<td>67%</td>
<td>51%</td>
</tr>
<tr>
<td>3 workers per label</td>
<td>70%</td>
<td>69%</td>
</tr>
<tr>
<td>5 workers per label</td>
<td>70%</td>
<td>70%</td>
</tr>
</tbody>
</table>

Table 1: The effect of redundancy (using both majority voting and Dawid and Skene voting) on the accuracy of crowd labels.

Note that adding labels from 3 to 5 does not significantly increase the crowd’s agreement in this case.

A second, perhaps more important, observation is that crowd accuracy often varies for different subgroups of the unlabeled data. For example, we asked Mechanical Turk workers to label facial expressions in the CMU Facial Expression dataset and measured agreement with hand-supplied labels. This dataset consists of 585 head-shots of faces of 20 users, each in 32 different combinations of different positions and facial expressions, where expressions are one of neutral, happy, sad or angry. We found that crowd performance was significantly worse when the faces were looking up versus in other positions:

<table>
<thead>
<tr>
<th>Facial orientation</th>
<th>Avg. accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>straight</td>
<td>0.6335%</td>
</tr>
<tr>
<td>left</td>
<td>0.6216%</td>
</tr>
<tr>
<td>right</td>
<td>0.6049%</td>
</tr>
<tr>
<td>up</td>
<td>0.4805%</td>
</tr>
</tbody>
</table>

Similar patterns appear in several other data sets, with the crowd performing significantly worse for certain subgroups. To take advantage of these two observations, we developed an algorithm that computes the optimal number of questions to ask about each subgroup, by estimating the probability $p_{g}$, with which the crowd correctly classifies items of a given subgroup $g$, and then solving an integer linear program (ILP) to choose the optimal number of questions (i.e., degree of redundancy) for labeling each item from that subgroup, given these probabilities.

First, observe that when combining answers using majority voting, with an odd number of votes, say $2v + 1$, for an unlabeled item $u$ with a true label $l$, the probability of the crowd’s combined answer $l^*$ being correct is the probability that at most $v$ or fewer workers get the answer wrong. Denoting this probability with $\Pr(l^* = l | 2v + 1$ votes $)$, we have:

$$
P_{g_{2v+1}} = \Pr(l^* = l | 2v + 1$ votes $) = 
\sum_{i=0}^{v} \binom{2v+1}{i} \cdot p_{g_{2v+1-i}} \cdot (1 - p_{g})^{i}.
$$

where $p_b$ is the probability that a crowd worker will correctly label an item in group $g$.

Next, we describe our algorithm, called **Partitioning Based Allocation (PBA)**, which partitions the items into subgroups and optimally allocates the budget to different subgroups, by computing the optimal number of votes per item, $V_g$, for each subgroup $g$. PBA consists of three steps:

1. Partition the dataset into $G$ subgroups. This can be done either by partitioning on some low-cardinality field that is already present in the dataset to be labeled (for example, in an image recognition dataset, we might partition by the user who took the picture, or the time when the picture was shot, e.g., at night or day), or by using an unsupervised clustering algorithm such as $k$-means. For instance, in the CMU facial expression dataset, we partitioned the images based on user IDs, leading to $G = 20$ subgroups each with roughly 32 images.

2. Randomly pick $n_0 > 1$ different data items from each subgroup, and obtain $v_0$ labels for each one of them; Estimate $p_b$ for each subgroup $g$, either by choosing data items for which the label is known and computing the fraction of labels that are correct, or by taking the majority vote for each of the $n_0$ items, assuming it is correct, and then computing the fraction of labels that agree with the majority vote. For example, for the CMU dataset, we asked for $v_0 = 9$ labels for $n_0 = 2$ random images from each subgroup, and hand-labeled those $n_0 \cdot G = 40$ images to estimate crowd’s $p_b$ for $g = 1, \ldots, 20$.

3. Solve an ILP to compute $V_g$ for all groups. Suppose we are given an overall budget $B$ of questions we can ask. We want to allocate that amongst our $G$ subgroups optimally. We use $b_g$ to denote the budget allocated to subgroup $g$, and create a binary indicator variable $x_{g,b}$ whose value is 1 iff subgroup $g$ is allocated a budget of $b$. Further suppose that our learner has chosen to try to label $f_g$ items from each subgroup $g$.

We can then formulate an ILP with the objective function that seeks to minimize:

$$
\sum_{g=1}^{G} \sum_{b=1}^{b_{max}} x_{g,b} \cdot (1 - P_{g,b}) \cdot f_g
$$

where $b_{max}$ represents the maximum number of votes that we are willing to ask per item. This goal function captures the expected weighted error of the crowd, i.e., it has a lower value when we allocate a larger budget ($x_{g,b} = 1$ for a large $b$ when $P_{g,b} > 0.5$) to subgroups whose questions are harder for the crowd ($P_{g,b}$ is small) or the learner has asked for more items from that group to be labeled ($f_g$ is large). This optimization is subject to the following constraints:

$$
\sum_{g=1}^{G} \sum_{b=1}^{b_{max}} x_{g,b} = 1
$$

$$
\sum_{g=1}^{G} \sum_{b=1}^{b_{max}} x_{g,b} \cdot b \cdot f_g \leq B
$$

Here the constraint (8) ensures that we pick exactly one $b$ value for each subgroup, and constraint (9) ensures that we stay within our overall item labeling budget.

We report the details of this PBA algorithm on the complete CMU dataset in Section 5.2

**Balancing Classes:** Our final observation about the crowd’s accuracy is that crowd workers perform better at classification tasks when the number of instances from each class is relatively balanced. For example, given a face labeling task where the goal is to tag each face as “man” or “woman”, people perform worse at labeling the rarer class when that class is very infrequent. For example, if only 0.1% of images are of men, then crowd workers will have a high error rate when labeling men (since they become conditioned to answer “woman”). This effect has been documented in other crowd-sourced settings as well [21].

Interestingly, both our Uncertainty and MinExpError algorithms naturally tend to increase the fraction of labels they obtain for rare classes. This is because they tend to have more uncertainty about items with rare labels (due to insufficient examples in their training set), and hence are more likely to ask users to label those items. Thus, our algorithms naturally improve the “balance” in the questions they ask about different classes, which in turn improves crowd labeling performance. We show this effect in Section 5.2 as well.

**4.2 When To Stop Asking**

Users may have a fixed budget, or may want to learn a model that achieves a specific error level on one or more fidelity metrics (e.g., accuracy or F1-measure). Given a fixed budget, it is straightforward to continue to ask questions until the budget is exhausted.

To achieve a specific error level, we need a way to estimate the current error of the learned model. The most straightforward way to do this is to learn a model based on the data collected so far, and measure its ability to accurately classify the gold data according to the desired metric. We can then continue to ask questions until a specific error level on gold data is achieved (or until the rate of improvement of error rates levels off.)

In the absence of (sufficient) gold data, we adopt the standard method of $k$-fold cross validation, where we randomly partition the crowd-labeled data into test and training data, and measure the ability of a model learned on training data to predict test values. We repeat this procedure $k$ times and average the error estimates to get an overall assessment of the model’s quality. This method, according to our experiments, provides more reliable estimates of the model’s current quality than estimating the training error on a small amount of gold data. We report on the performance of this idea in Section 5.2

**4.3 Effect of Batch Sizes**

At each iteration of the iterative scenario, we need to choose a batch of questions to ask from the crowd. In Section 4.3, we presented algorithms for ranking all the questions in terms of their potential benefit to the overall quality. Given such a ranked list, we still need to decide on the “batch size”, namely how many questions to submit to the crowd in the current iteration. Clearly, being able to ask a larger number of questions overall will yield better answers. Given an overall budget, however, asking the questions in smaller batch sizes yields better quality, because smaller batch sizes (i) allow for more rounds of feedback from the crowd into the learning process, and (ii) provide more chances to improve the classifier. Additionally, our ranking algorithms evaluate the quality of each question in isolation, and thus, they try to identify the single optimal question to submit to the crowd. Thus, larger batch sizes can also degrade the quality of the selected questions. In general, for a given batch size $\beta$, to find the optimal set of questions to ask the crowd, at least in principle, one has to compute the ranking score for all subsets of size $\beta$, namely $\binom{U}{\beta}$ subsets (here $U$ is the set of unlabeled items at each step). Since this is computationally infeasible, in practice, most active learners only calculate the ranking scores for single questions, and pick $\beta$ questions with highest scores. As a result, smaller the $\beta$, better the results, with the best results often being achieved with $\beta = 1$.

On the other hand, larger batch sizes reduce the overall run-time substantially. This is because submitting questions in larger batches
not only allows for more parallelism from the crowd (as several workers can label items at the same time), but also decreases the time complexity of our algorithms (by performing fewer iterations in the iterative case). We confirm this intuition in our experiments in Section 5.2.

Given these two observations, the proper batch-size can be chosen using the following strategies. When the goal is to minimize time under a minimum quality requirement, choose the largest possible batch-size, potentially asking all B questions at once, or even falling back to the upfront scenario. To ensure that the quality requirement is met, a reasonable choice is to start from smaller batch sizes and continuously increase it depending on the current quality (or the rate at which it improves), say by using cross-validation for estimating the quality. Fortunately, as we show in Section 5.2 the predictive performance of our active learning algorithms degrade gracefully, that is, the impact of increasing the batch size on quality is typically not as dramatic as its impact on the overall runtime. In fact, in most of our experiments with several classifiers and datasets, the F1-measure decays linearly with increasing batch size (with a very modest slope), while run-time decreases exponentially with increasing batch size.

When the goal is to maximize quality under a monetary constraint with no time constraint, a batch size of 1 is a clear choice. However, when a time constraint is also present, choosing the best batch-size becomes harder. In principle, one can submit questions in very small batches but risk not being able to spend the entire budget due to the time-constraint. This could be justified, only if the classifier would benefit from fewer but much more carefully selected labels (due to a smaller batch-size), than it would from more but less carefully selected set of questions. Otherwise, the best is to choose the smallest batch-size such that the algorithm can still spend all its budget within the time-constraint. As mentioned before, the quality gains of our active learning algorithms degrade gracefully with larger batch-sizes. However, determining the relationship between quality versus overall number of labels, is closely related to label complexity, which is a problem known to be hard and in general, depends on the classifier and the data distribution. Estimating the rate of improvement in quality while the algorithm progresses, could be used as a practical rule-of-thumb. Developing a robust algorithm to handle such cases represents an interesting future work. In all experiments of this paper, unless otherwise specified, we use 10% of the total budget as our fixed batch-size.

5. EXPERIMENTAL RESULTS

In this section, we present several experiments designed to evaluate the effectiveness of our active learning algorithms (Section 3) and crowd optimization techniques (Section 4). We consider different criteria that have been used for evaluating active learning techniques and compare our techniques against a random baseline (that chooses items uniformly at random, to be labeled by crowd) as well as several state-of-the-art algorithms. Our primary goal is to understand how much more quickly and/or accurately our active learning methods can label a dataset in comparison both the baseline and previous active learning techniques.

In Section 5.1, we report experiments that we ran on Amazon Mechanical Turk on entity resolution, vision, and sentiment analysis tasks on our Uncertainty and MinExpError methods. Section 5.2 shows the effectiveness of our crowd optimizations presented in Section 4. In Section 5.3, we provide a summary of results on 15 well-known datasets from the UCI KDD repository. Finally, we study the run-time and scalability of our algorithms in Section 5.4.

Overall, our experiments show the following: (i) our algorithms can ask several orders of magnitude fewer questions to achieve the same quality as the random baseline, (ii) our MinExpError algorithm works better than Uncertainty in the upfront setting, but the two are comparable in the iterative setting, (iii) Uncertainty has substantially lower computational overhead than MinExpError, and (iv) in the context of entity-resolution, our active learning algorithms are more general and perform comparably or better than an existing active learning algorithm specifically designed for entity resolution.

Experimental Setup. All the algorithms were tested on a Linux server with dual-quad core Intel Xeon 2.4 GHz processors and 24GB of RAM. Throughout this section, unless stated otherwise, we used the following parameters: we repeated each experiment 20 times and reported the average result, every task cost 1€, and the size of the initial training and the batch size were respectively 0.03% and 10% of the unlabeled set.

Methods Compared. We ran experiments on five different learning algorithms, in both the upfront and iterative scenarios.

3. MarginDistance: The method of Section 3.1, which is specifically designed for active learning with SVMs, and picks items that are closer to the margin (see Section 5).
4. Bootstrap-LV: Another bootstrap-based active learning technique proposed by [27]. This method can only be applied to probabilistic classifiers, e.g., we will not include this in experiments with SVMs (see Section 5).
5. Baseline: A passive learner that randomly selects unlabeled items to send to the crowd.

In the plots, we prepend the scenario name to the algorithm names, e.g. UpfrontMinExpError or IterativeBaseline. We have repeated our experiments with different classifiers as the underlying learner, including SVM, Naïve-Bayes classifier (NBC), neural networks, and decision trees. For lack of space, we only report the results of each experiment for one type of classifier. When not specified, we used linear SVM.

Evaluation Metrics. Active learning algorithms are typically evaluated based on their learning curve, which plots the quality measure of interest (e.g., accuracy or F1-measure) as a function of the number of data items that are labeled. To quantitatively compare different learning curves, the following metrics are often used:

1. Area under curve (AUC) of the learning curve.
2. AUCLOG, which is the AUC of the learning curve when the X-axis is in log-scale.
3. Cost reduction or the (average or maximum) horizontal distance between two learning curves.

The higher the AUC, the better the learner does at achieving higher quality for the same cost/number of questions. Due to the diminishing-return shape of learning curves, the average quality lift is usually in a $0–16\%$ range.

AUCLOG favors algorithms that improve the metric of interest early on (e.g., with few examples). Due to the logarithm, the improvement of this measure is typically in a $0–6\%$ range.

The cost reduction captured by the average horizontal distance between two curves indicates the average cost or number of questions that would be saved by one curve compared to the other, on average. We compute this measure by taking an average over all the quality values that are achievable by both curves. In some cases, when the two curves do not have a range in common and interpolation is not reasonable, this number is undefined. The number of questions saved by a competent active learner should be larger than $1\times$ (potentially $100\times$ or more), while a value $<1$ indicates a
performance worse than a random baseline.

5.1 Crowd-sourced Datasets

We experiment with several datasets labeled using Amazon Mechanical Turk in this paper. In this section, we report the performance of our algorithms on each of them.

5.1.1 Entity Resolution

Entity resolution is the task of finding different records that refer to the same entity. Entity resolution (ER) is an essential step in data integration and cleaning, especially when data comes from multiple sources. Crowd-sourcing is typically a more accurate means for ER than machine learning, but also more expensive and slower [35].

We used the Product2 dataset which contains product attributes (name, description, and price) of items listed on abt.com and buy.com websites. The task is to detect pairs of items on the two websites that even though listed under different names, descriptions or price, are in fact the same item (e.g., “iPhone White 16 GB” vs “Apple 16GB White iPhone 4†”). We used the same dataset that was used in [35], where authors asked the crowd to label 8,815 pairs of items as either identical or non-identical. This dataset consists of 12% identical pairs and 88% non-identical pairs. Given that the crowd’s accuracy can vary from one experiment to another, in order to have a fair comparison, we have obtained the same crowd labels used in [35] by contacting the authors. In their dataset, each pair has been labeled by 3 different workers, with an average accuracy of 89% and F1-measure of 56%. We also used the same classifier used in [35], namely a linear SVM where each pair of items is represented by the Levenshtein and Cosine similarities of their names and descriptions. When trained on 3% of the data, this classifier has an average accuracy of 80% and F1-measure of 40%. By using the same classifier and set of crowd-labels, we can isolate and compare the net effect of our crowd-sourcing strategies versus the technique proposed in [35], called CrowdER. The results are shown in Figure 3. Here, the baseline corresponds to the CrowdER algorithm. As expected, all methods improve with more questions (i.e., more labels). However, the overall F1-measures of these different strategies improve at different rates. MarginDistance, MinExpError, and CrowdER are all comparable, while Uncertainty improves much more quickly than the others. In other words, Uncertainty is able to identify the data items about which the model has the most uncertainty and get the crowd to label them earlier on. This result is highly encouraging, because even though CrowdER is itself a state-of-the-art active learning algorithm that is highly specialized to improve the F1-measure of entity resolution (see Section 6), our general-purpose active learning algorithms still achieve comparable results (in case of MinExpError) and even better results (in case of Uncertainty).

5.1.2 Image Detection

Vision-related problems are another area in which crowd-sourcing is heavily utilized, e.g., in tagging pictures, finding objects, and identifying bounding boxes [34]. In all of our vision experiments, we employed a relatively simple classifier where the PHOW features (a variant of dense SIFT descriptors commonly used in vision tasks [7]) of a set of images are first extracted as a bag of words, and then a linear SVM is used for their classification. While this is not the state-of-the-art image detection algorithm, we show that even with this simple classifier, our active learning algorithms can greatly reduce the cost of many challenging vision tasks.

Gender Detection. We used the faces from Caltech101 dataset [12] and manually labeled each image with its gender (266 males, 169 females) as our ground truth. We also gathered crowd labels by asking the gender of each image from 5 different workers. We started by training the model on a random set of 11% of the data. In Figure 4 we show the accuracy of the crowd, the accuracy of our machine learning model and also the overall accuracy of the model plus crowd data. For instance, when a fraction x of the labels were obtained from the crowd, the other 1 ‐ x labels were determined from the model, and thus, the overall accuracy was x × aC + (1 − x) × aM, where aC and aM are the crowd and model’s accuracy, respectively. Similar to our entity resolution experiments, our algorithms improve the quality of the labels that are provided by the crowd, i.e., by asking questions for which the crowd tends to be more reliable. In this case, though, the overall quality of the crowd is much higher than in the entity resolution case and therefore the crowd’s accuracy is improved only from 98.5% to 100%. Figure 4 shows that both MinExpError and Uncertainty perform well in the upfront scenario, respectively improving the accuracy of the baseline by 4% and 2% on average, and improving its AUCLOG by 2-3%. Here, due to the upfront scenario, MinExpError saves the most number of questions. The baseline has to ask 4.7× (3.7×) more questions than MinExpError (Uncertainty) to achieve the same accuracy. Again MarginDistance, although specifically designed for SVM, achieves little improvement over the baseline.

Object Containment. We again mixed 50 human faces and 50 background images from Caltech101 [12]. Because differentiating human faces from background clutter is easy for humans, we used the crowd labels as ground truth in this experiment. Figure 5 shows the upfront scenario with an initial set of 10 labeled images, where both Uncertainty and MinExpError save the most number of questions. The baseline has to ask 4.7× (3.7×) more questions than MinExpError (Uncertainty) to achieve the same accuracy. Again MarginDistance, although specifically designed for SVM, achieves little improvement over the baseline.

5.1.3 Sentiment Analysis

Popular microblogging web-sites such as Twitter have become rich sources of data for sentiment analysis (a.k.a. opinion mining) [24]. As an example, a analyst might ask questions such as “how many of the tweets that mention iPhone have a positive or negative sentiment?”. Training accurate classifiers requires a sufficient amount of accurately labeled data, and with over hundreds of million of daily tweets, asking the crowd to label all of them is impossible. Throughout our experiments, whenever improvement numbers are presented, they are measured from the log files and are not necessarily visible in the plots themselves.
feasible. In this experiment, we show that with our active learning algorithms, with as little as 1K-3K crowd-labeled tweets, we can achieve very high accuracy and F1-measure on a corpus of 10K-100K unlabeled tweets.

We randomly chose 100K tweets from an online corpus that provides ground truth labels for the tweets, with equal number of positive and negative-sentiment tweets. To obtain crowd labels, we obtained labels (positive, negative, neutral, or vague/unknown) for each tweet from 5 different workers. Figure 5 shows the results for using 3K initially labeled data points in the 100K dataset in the upfront setting. The results confirm that the upfront scenario is best handled by our MinExpError Algorithm. Here, the MinExpError, Uncertainty and MarginDistance algorithms improve the average F1-measure of the baseline model by 11%, 9% and 5%, respectively. Also, MinExpError increases baseline’s AUCLOG by 4%. All three active learning algorithms reduce the number of required questions to achieve a given accuracy or F1 dramatically. In comparison to the baseline, MinExpError, Uncertainty, and MarginDistance reduce the number of questions by factors of 46×, 32×, and 27×, respectively.

5.2 Experiments on Optimizing for the Crowd

In this section we present results for our crowd-specific optimizations described in Section 4.

PBA Algorithm: We first report experiments on the PBA algorithm for choosing how many items to ask from the crowd (see Section 4.1). Recall that this algorithm works by partitioning the items into subgroups and optimally allocating the budget to different subgroups. We report results on the CMU facial expressions dataset. We observed that the crowd had a particularly hard time telling the facial expression of some of the individual faces and had an easier time with others, so we created subgroups based on the user column of the data set, and asked the crowd to label the expression on each face. By choosing \( n_0 = 2 \), \( v_0 = 9 \), and \( b^{\text{max}} = 9 \) (as mentioned above), we ran PBA and compared its performance to a uniform budget allocation scheme where the same number of questions are asked about all items uniformly, as done in previous research (see 6). The results are shown in Figure 6. Here, the X axis shows the normalized budget, e.g. a value of 2 means the budget was twice the total number of unlabeled items. The Y axis shows the overall classification error of the model using majority voting under different allocations. Here, the solid lines show the actual error achieved under both strategies, while the blue and green dotted lines show our estimates of their performance before running the algorithms. From Figure 7 we see that even though our estimates of the actual error are not highly accurate, since we only use them to solve an ILP that would favor harder subgroups, our PBA algorithm (solid green) can still reduce the overall crowd error by about 10% (from 45% to 35%). We also show how PBA would perform if it had an oracle that provided access to exact values of \( P_{p,b} \) (red line).

Balancing Classes: Recall from Section 4.1 that our algorithms tend to ask more questions about rare classes than common classes. This can help to improve the overall performance of our algorithms. In Figure 8 we report the crowd’s F1-measure in an entity resolution task under different active learning algorithms. The dataset used in this experiment has many fewer positive instances (11%) than negative ones (89%). The main observation here is that although the crowd’s average F1-measure for the entire dataset is 56% (this is achieved by the baseline which randomly picks items to be labeled), our Uncertainty algorithm can lift this up to 62%, mainly because the questions posed to the crowd have a balanced mixture of positive and negative labels.

k-Fold Cross Validation for Estimating Accuracy: Figure 11 shows the F1-measure of an SVM model on the cancer UCI dataset,
and our estimates using the k-fold cross validation technique described in Section 4.2. Our estimates are reasonably close to the true F1 values, especially as more labels are obtained from the crowd. This suggests that this technique allows us to effectively estimate current model accuracy, and to stop acquiring more data once model accuracy has reached a reasonable level.

The Effect of Batch Size: We now present experiments showing the effect of batch size on result quality, based on the observations in Section 4.3. The effect is typically moderate (and often linear), as shown in Figure 9. Here we show that the F1-measure gains can be in the 8−10% range (as explained in Section 4.3). However, larger batch sizes reduce runtime substantially, as shown in Figure 10. Here, going from batch size 1 to 200 reduces the time to train a model, by about two orders of magnitude (from thousands of seconds to tens of seconds), which is significant.

5.3 UCI Classification Datasets

In Section 5.1 we validated our algorithms on crowd-sourced datasets. In this section, we also compare our algorithms on several datasets from the UCI KDD repository [1]. For these datasets, the labels are provided by experts; that is, the ground truth and the crowd labels are the same. Thus, by excluding the effect of an imperfect crowd, we can compare different active learning strategies in isolation. We have chosen 15 well-known datasets, as shown in Figures 12 and 13 (here, several of the algorithms could not complete on two of the datasets, so only 13 datasets are reported.)

To keep our results unbiased, we have avoided any dataset-specific tuning or preprocessing steps, and applied same classifier with the same settings to all datasets. In each case, we examined 10 different question asking budgets of 10%, 20%, …, 100%, each repeated 10 times and report the average. Also, to compute the F1-measure for datasets with more than 2 classes, we have either put all but the majority class in a single class, or have arbitrarily partitioned all the classes into two new ones (details in [23]).

Here, other than the random baseline, we compare Uncertainty and MinExpError against two other active learning techniques, namely MarginDistance and Bootstrap-LV. Bootstrap-LV is designed only for probabilistic classifiers i.e., when the classifier provides class probability estimates (CPE) as well as the predicted class label. Thus, for all learning techniques, we used MATLAB’s implementation of decision trees as our classifier. We used default parameters except for the following: no pruning, no leaf merging, and a ‘minparent’ of 1 (impure nodes with 1 or more observations can be split).

Figures 12 and 13 show the reduction in the number of questions on for both upfront and iterative settings for Bootstrap-LV, Uncertainty, and MinExpError, while Table 2 shows the average AUROC, F1, and reduction in the number of questions asked across all 15 datasets for the different methods. The two figures omit results for MarginDistance because it generally performs poorly (as indicated by the average results in Table 2). We report all the measures of different active learning algorithms in terms of their performance improvement relative to the baseline (so higher numbers are better). For instance, consider Figure 12. Here, on the cancer data set, MinExpError reduces the number of questions asked by 25×, while Bootstrap-LV and Uncertainty reduce the number by about 15×.

In summary, these results are consistent with those observed with crowd-sourced datasets. In the upfront setting, MinExpError is significantly beneficial and superior to other active learning techniques, with more than 104× savings in the total number of questions on average. MinExpError also improves the AUROC and average F1-measure of the baseline on average by 5% and 15%, respectively. After MinExpError, the Uncertainty and Bootstrap-LV are most effective with a comparable performance, i.e. 55-69× savings, improving the AUROC by 3%, and lifting the average F1-measure by 11-12%. Least effective in the upfront scenario is MarginDistance which still provides around 15× saving. Although Bootstrap-LV appears to perform well here, recall that it only works for probabilistic classifiers (e.g., decision tree classifiers).

For the iterative scenario, Uncertainty actually works better than MinExpError, with an average saving of 7× over the baseline in questions asked and an increase in AUROC and average F1-measure by 1% and 3%, respectively. Note that the reason why the savings are in general more modest than in the upfront case, is that in the iterative setting the baseline receives much more labeled data and therefore, its average performance is much higher than that in the upfront case, and hence there is less room for improvement. How-
ever, given the comparable (and even slightly better) performance of Uncertainty compared to MinExpError in the iterative scenario, it becomes a much more favorable choice for this scenario considering that Uncertainty incurs significantly less processing overhead than MinExpError (see Section 5.4).

Figure 12: Performance of our algorithms vs Bootstrap-LV in Upfront Scenario on UCI Datasets. Y Axis is log scale, reporting the ratio of questions asked by the random baseline to the questions asked by the method.

Figure 13: Performance of our algorithms vs Bootstrap-LV in Iterative Scenario on UCI Datasets

<table>
<thead>
<tr>
<th>Upfront</th>
<th>Method</th>
<th>AUCLOG(F1)</th>
<th>Avg(Q's Saved)</th>
<th>Avg(F1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Margin Distance</td>
<td>1.00x</td>
<td>12.37x</td>
<td>1.05x</td>
<td></td>
</tr>
<tr>
<td>Bootstrap-LV</td>
<td>1.03x</td>
<td>69.31x</td>
<td>1.12x</td>
<td></td>
</tr>
<tr>
<td>Uncertainty</td>
<td>1.03x</td>
<td>55.14x</td>
<td>1.11x</td>
<td></td>
</tr>
<tr>
<td>MinExp Error</td>
<td>1.05x</td>
<td>104.52x</td>
<td>1.15x</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Average improvement in AUCLOG, Questions Saved, and Average F1, across all 15 UCI datasets, in Upfront and Iterative setting.

6. RELATED WORK
Crowd-sourcing. Several groups have worked on integrating crowd-sourcing and human operators into data workflows and database systems [13, 17, 23, 22]. These crowd-enabled databases face limitations when dealing with large datasets. While these systems try to use DB techniques to reduce the amount of unnecessary work needed by humans (e.g., the number of pair-wise comparisons in a join query), in the end the crowd still has to provide at least as many labels as there are unlabeled items that are queried by the user. It is simply not feasible to label millions of data items this way. Our algorithms are motivated by large-scale datasets and aim to avoid obtaining crowd labels for a significant portion of the data by training and exploiting machine learning models.

Active Learning. There has been a large body of work on active learning in the machine learning literature (see [29] for a survey). However, this field has traditionally dealt with situations where a moderate number of datapoints in a specific domain (e.g., medical diagnosis [15]) need to be labeled and where labels are obtained from highly trained experts (e.g., doctors). As a result, most of these techniques are domain or model-class specific, computationally expensive, and often assume perfect labels. There have been many domain-specific active learning techniques, e.g. in vision [34], entity-resolution [4], and text classification [31]. Our algorithms however work for general classification tasks and do not use or require any domain knowledge.

Focusing on items for which the learner is most uncertain has proven to avoid obtaining crowd labels for a significant portion of the data by training and exploiting machine learning models.
bootstrap-based technique called Bootstrap-LV that uses the model’s class probability estimates to measure uncertainty. In Section 5, we showed that, in the iterative scenario, our Uncertainty algorithm performs better or comparably, while in the upfront scenario our MinExpError algorithm is significantly superior to Bootstrap-LV. Moreover, we do not assume probabilistic classifiers and use a simple notion of variance based on the classifier’s class prediction.

**Active learning in crowd-sourcing.** Recently, a few other papers have applied active learning in crowd-sourcing [2, 4, 35] for the specific problem of entity resolution. These solutions are typically only applicable to entity resolution while our algorithms are quite general. In Section 5.1, we applied Uncertainty and MinExpError to the same crowd-sourced dataset used in [35] and showed that our active learners can still improve the F1-measure by about 10% even for those pairs of items that would be sent to the crowd by [35].

Yan et al. [30] also looked at the problem of active learning from a group of workers, but focused on the problem of picking the best worker to answer each question. This is different from our scenario because in crowd-sourcing systems like Mechanical Turk, the crowd database has no control over which users answer a given item. Pujara et al. [26] also propose using active learning in a crowdsourced setting where they ask crowd workers to label the lowest confidence data items. Additionally, they simply use proximity to the decision boundary as a metric of confidence, which, as we showed in our experiments, does not perform much better than a random baseline. Filtering low-quality workers has been discussed in [10] while the effect of redundancy on accuracy has been studied in [30]. Our PBA algorithm improves on [30] by considering crowd error for different subgroups, while [30] assumes that the crowd error is independent of the item being classified.

**7. CONCLUSIONS**

In this paper, we proposed an approach to integrate new active learning algorithms into crowd-sourced databases. Specifically, we proposed two active learning algorithms designed for two different settings. In the **upfront** setting, we ask all questions from the crowd in one go. In the **iterative** setting, the questions to be asked from the crowd are adaptively picked, and added to the labeled pool, which is followed by retraining the model and iterating over the process. While this is more expensive because of the iterative retraining, it also has a higher chance of learning a better model. We design two algorithms Uncertainty and MinExpError based on the theory of non-parametric bootstrap, leading to wide applicability to a broad range of machine learning models. We also proposed algorithms for choosing the number of questions to ask from different crowd-workers, based on the characteristics of the data being labeled, and studied the effect of batching on the overall runtime and quality of our active learning algorithms. Our results, on three data sets collected with Amazon’s Mechanical Turk, and with 15 data sets from the UCI KDD archive, show that compared to choosing items to label uniformly at random, our algorithms make $8 \times$ fewer label requests than existing active learning techniques in the upfront scenario, and $6 \times$ fewer label requests in the iterative scenario. We believe that these algorithms would prove to be immensely useful in crowd-sourced database systems.

**8. REFERENCES**


