Crowdsourcing High Quality Labels with a Tight Budget

Qi Li[1], Fenglong Ma[1], Jing Gao[1], Lu Su[1], Christopher J. Quinn[2]
[1]SUNY Buffalo, Buffalo, NY USA
[2]Purdue University, West Lafayette, IN USA
{qli22,fenglong,jing,lusu}@buffalo.edu , cjquinn@purdue.edu

ABSTRACT

In the past decade, commercial crowdsourcing platforms have revolutionized the ways of classifying and annotating data, especially for large datasets. Obtaining labels for a single instance can be inexpensive, but for large datasets, it is important to allocate budgets wisely. With limited budgets, requesters must trade-off between the quantity of labeled instances and the quality of the final results. Existing budget allocation methods can achieve good quantity but cannot guarantee high quality of individual instances under a tight budget. However, in some scenarios, requesters may be willing to label fewer instances but of higher quality. Moreover, they may have different requirements on quality for different tasks. To address these challenges, we propose a flexible budget allocation framework called Requallo. Requallo allows requesters to set their specific requirements on the labeling quality and maximizes the number of labeled instances that achieve the quality requirement under a tight budget. The budget allocation problem is modeled as a Markov decision process and a sequential labeling policy is produced. The proposed policy greedily searches for the instance to query next as the one that can provide the maximum reward for the goal. The Requallo framework is further extended to consider worker reliability so that the budget can be better allocated. Experiments on two real-world crowdsourcing tasks as well as a simulated task demonstrate that when the budget is tight, the proposed Requallo framework outperforms existing state-of-the-art budget allocation methods from both quantity and quality aspects.

1. INTRODUCTION

In the Internet age, the thriving growth of crowdsourcing platforms unleashes the astonishing power of crowd wisdom, which makes it more convenient and efficient to obtain labels. On commercial crowdsourcing platforms such as Amazon Mechanical Turk[1] (mTurk) and CrowdFlower[2], requesters design and post their tasks (also known as human intelligence tasks, HITs). Each HIT may contain one or several instances (such as images) to be labeled. Workers can view available HITs and choose whichever they want to work on. After the workers finish their jobs, requesters adjudicate the final results and pay the workers. As individual workers may make mistakes, requesters can ask multiple workers for each instance. This "repeated labeling" is commonly used and recommended by mTurk[3] to get more accurate labels. Although crowdsourcing is known to be a cheap source for individual labels, with the repeated labeling and a big pool of instances, it is important to use the budget wisely, especially when the budget is tight comparing with the size of instance pool. For example, ten thousand dollars may still be tight to label all inappropriate videos on YouTube. With a tight budget, the requesters can only afford a limited amount of labels and they need to decide which instances should get labels and how many on them.

One of the major dilemma the requester faces is the trade-off between **quantity** of labeled instances and **quality** of the final results. Intuitively, if we want more instances to be labeled (high quantity), each instance will receive less labels, resulting in a lower quality of the final results; on the other hand, if we want high quality results, we may end up with fewer questions being labeled (low quantity) so each instance can get more labels. This trade-off is even more noticeable under a tight budget. The preference of this trade-off may vary for different tasks. For some tasks, quantity is more important, but for others, quality is preferred. Under a tight budget, most existing budget allocation work [4, 11, 19, 24, 25, 30, 31, 38], though being able to achieve good quantity, cannot guarantee high quality for individual instances. Consider a case that a requester has N instances and can only afford N labels. Then these budget allocation methods will give exactly one label to each instance, and the final results are inadequate to justify. However, in some scenarios, requesters may prefer quality over quantity. They may be willing to label less instances, i.e., sacrifice quantity to some extent, but require what they get is of high quality. In that case, the existing methods cannot meet the requesters' needs. Therefore, a method is badly needed if requesters have a preference of quality over quantity under a tight budget.

To achieve this, the major challenge is how a requester can measure the quality of the labels, a problem often overlooked in the crowdsourcing literature. Most crowdsourcing algorithms use accuracy rate to measure the quality of fi-

[1]https://www.mturk.com/mturk/
[2]http://www.crowdflower.com/

[3]mturkpublic.s3.amazonaws.com/docs/MTURK_BP.pdf

WSDM'16, February 22–25, 2016, San Francisco, CA, USA.

DOI: http://dx.doi.org/10.1145/2835776.2835797

nal results. It can be a nice measurement for an algorithm, but usually is impractical for requesters because the ground truths are often unknown, which is the reason requesters post the HITs in the first place. In this case, it is impossible to judge whether an instance is correctly labeled, and it is hard to tell how many labels are enough.

To address this challenge, we propose to use **confidence**, a user-specified measurement, to measure quality. Instead of judging a result as correct or incorrect, the requester can judge a result as confident or unconfident based on his "requirement," such as a minimum ratio between two classes' vote counts. If the results have high confidence under a strict requirement, it is likely that the results also have high accuracy. We use the following example to illustrate how confidence is decided under a requirement.

EXAMPLE 1. *Suppose a requester sets a requirement as a minimum ratio of* 4 *between two classes' vote counts. Then an instance with results of one vote for class* $+1$ *and two votes for class* -1 *is unconfident, while another instance with results of one versus four is confident. Comparing these two, the latter one is more likely to obtain the correct label.*

Using this measurement, the requesters can specify a concrete and detailed requirement to reflect their needs for quality. Another benefit of using confidence is that it can help requesters identify hard instances so that further actions can be taken. A challenging instance might remain unconfident after many rounds of labeling process. Then the requesters may not want to waste their precious budgets on that instance and should consider asking experts for labels.

In this paper, we develop a **Requ**irement based budget **allo**cation (Requallo) framework, which can adjust the allocation policy according to requesters' needs on quality in the form of a confidence requirement. No matter the size of a budget, the proposed Requallo framework can guarantee the requesters' desired quality while trying to maximize quantity (i.e., number of confident instances) and staying within the budget. Formally, the objective of the framework is stated as maximizing the overall expected completeness of the tasks subject to the budget, where completeness for an instance reflects how much confidence it gets under the requirement. To solve this optimization problem, we formulate it as a Markov decision process and use a one-step look-ahead greedy algorithm to search for the instance with the maximum reward in completeness to label next.

We further extend the basic model to incorporate workers' reliabilities. Instead of using vote counts when calculating expected completeness, we use the weighted vote counts derived via MLE estimation, where the weights are determined by the corresponding workers' reliability. Standard ways of inferring reliability for crowdsourcing include qualification, spammer detection, and EM based aggregation. Extensions on labeling in parallel and multi-class cases are also discussed.

In summary, we make the following contributions in this paper:

- We identify the trade-off between quantity and quality in crowdsourcing tasks under a tight budget. Requesters can specify a measure of quality.
- We build a flexible framework to allocate the budget for instances according to requesters' needs on quality. Examples of various quality requirements are discussed and demonstrated.
- We formulate the budget allocation problem as a Markov decision process and produce a sequential labeling policy. It maximizes the number of labeled instances that achieve the quality requirement while staying under the budget.
- We test the proposed framework on two real-world crowdsourcing tasks and a simulated task. The real-world tasks include one benchmark NLP task conducted on mTurk and one trivia game conducted via an app. The results clearly demonstrate the advantages of the framework when the budget is tight.

In the following section, we give a brief overview on the related work in crowdsourcing and budget allocation problem. Then in Section 3 we formulate the problem and derive the proposed Requallo framework. In Section 4, the behavior of different policies is examined and analyzed via experiments conducted on two real-world crowdsourcing tasks and a simulated task. Finally, we conclude the paper in Section 5.

2. RELATED WORK

Recent years have witnessed the growing importance and popularity of crowdsourcing applications in real life, and thus many research efforts have been contributed to the development of new algorithms and designs for crowdsourcing tasks. Quality is one of the biggest concern in crowdsourcing tasks [28]. Research topics for this concern include how to aggregate crowd labels [3, 5, 6, 21, 29, 33–35, 40], how to select which crowd workers to work for a instance/task [7, 9, 12, 37, 41], and how to better design the crowdsourcing questions/tasks [8, 22, 26, 32, 36, 39]. Either by better task design or targeting a small group of informative workers, fewer labels may be needed for each instance, so they can potentially save the costs. However, in those work, budget constraints are not explicitly considered.

Motivated by the label cost concern, the budget allocation problem in crowdsourcing has attracted considerable attention [4, 10, 11, 19, 24, 25, 30, 31, 38]. Methods in [25, 27, 30, 38] focus on pricing tasks given a budget. The platforms designed in [25, 27] use flexible prices for different workers. However, this work does not consider the quality of the results. Work in [10, 30, 38] takes quality into consideration. However, different from our problem setting, interdependent tasks are considered in [30] and negotiation between requesters and workers is required and modeled by game theory in [10, 38]. The method proposed in [11] provides a minimum number of labels in order to achieve target reliability ϵ. This work is built upon the assumption that instances have the same level of difficulty, which is different from ours. The work [31] further improves the theoretical guarantee from the PAC (probably approximately correct) perspective. All of these methods are all inapplicable when the budget is lower than supporting one label per instance.

Among the budget allocation work, [4, 19, 24] are the most related to ours. They all sequentially select instances to label. In [24], label uncertainty is estimated as the tail probability of a Beta distribution and the instances with the highest label uncertainties are selected for more labels. This work assumes that data quality is the same on all instances, whereas we assume the quality would be higher for easy instances. Both [4] and [19] model the sequential labeling problem as a Markov decision process, though they have different goals from ours: the goal in [4] is to maximize

overall accuracy, while the goal in [19] is to maximize a utility function with consideration of pull market (i.e., workers may not accept jobs from requesters). Under our problem setting, [19] can be considered as a special case of the proposed Requallo framework (see Section 4.1). Under a tight budget (such as affording N labels for N instances), methods in [4, 24] tend to allocate the budget uniformly among the instances since querying on an unlabeled instance would be optimal. As a result, though a large quantity of instances may be labeled, very few of them could achieve the quality requirement. To the best of our knowledge, we are the first to address the trade-off between quantity and quality under the tight budget. The proposed Requallo framework takes requesters' specific needs for quality into account and works nicely even with very tight budgets.

3. METHODOLOGY

In this section, we introduce the proposed **Requ**irement based budget **allo**cation (Requallo) framework. We start with the basic framework, and then extend it to incorporate workers' reliability. Finally, some practical issues are discussed.

3.1 Basic Requallo Framework

Problem Setting and Formulation

Suppose a requester has N instances. Each instance is a binary labeling problem and they are independent of each other. The true label for the i-th instance Z_i is either $+1$ or -1. Adopting the idea from probabilistic classification, we model Z_i as a random variable. $P(Z_i = +1)$, which can be interpreted as the relative frequency that $+1$ appears when the number of workers approaches infinity, indicates the difficulty level of the i-th instance. If $P(Z_i = +1)$ is close to 0.5, it implies that the i-th instance is difficult, because even when the requester asks a large number of workers, there is still no predominant answer. On the other hand, if $P(Z_i = +1)$ is close to 1 (or 0), the instance is relatively easy. Here we assume that instances are not misleading, that is, $P(Z_i = +1) \gg 0.5$ (or $P(Z_i = +1) \ll 0.5$) implies that the true label Z_i is $+1$ (or -1).

Here, we assume all the workers are noiseless and independent, i.e., $P(y_{ij} = +1) = P(Z_i = +1)$ where y_{ij} is worker j's label on the i-th instance. In other words, the labels provided by the workers only depend on the true label of the instance and its difficulty level.[4] Based on this assumption, all labels for the i-th instance are essentially i.i.d. samples drawn from a Bernoulli distribution with $P_i = P(Z_i = +1)$. Majority voting can be used when conducting the final results since individual labels are equivalent. The cases that workers have different levels of reliability will be discussed in Section 3.2.

The proposed framework has two inputs. The requester specifies a confidence requirement and the budget. As mentioned in the introduction, different tasks may have different requirements. Therefore, the proposed framework can incorporate a variety of requirements as part of input. In this paper, the cost of a single label is assumed to be the same over all instances and all workers, so the budget works as a constraint that the requesters can only afford a certain amount of labels. We use T to denote the maximum amount. Note that we do not have to exhaust the budget. If all instances achieve the requirement before the budget is exhausted, the labeling process will be stopped and the remaining budget will be saved.

The goal of the proposed Requallo framework is to have as many instances as possible achieve the requirement under the budget, i.e., to maximize quantity while maintaining desired quality. Since the quantity can only be an integer, this goal will inevitably lead to an integer programming, an NP-hard problem. Therefore, we relax this integer constraint by maximizing the overall completeness subject to the budget, where completeness of an instance is a ratio in $[0, 1]$. The concrete definition of completeness will be introduced later.

Table 1 summarizes the frequently used notations in this paper; some will be introduced later.

Table 1: Notations

Notation	Definition
$\{1, 2, \dots N\}$	index set of instances
$\{1, 2, \dots M\}$	index set of workers
Z_i	true label of the i-th instance
$P(Z_i = +1), P_i$	difficulty level of the i-th instance
$y_{i,j}$	worker j's label on the i-th instance
T	maximum number of labels under the budget
a_i	vote count of $+1$ labels
b_i	vote count of -1 labels
$V_i(a_i, b_i)$	expected completeness of the i-th instance
$R(S^t, i^t)$	reward for the i-th instance at stage t
θ_j	reliability of worker j

Requirement Examples

Before formally introducing the proposed framework, we want to provide some examples of the requirements that requesters may be interested in. We denote the vote count for $+1$ class as a_i, and vote count for -1 class as b_i for the i-th instance. The requester expresses his quality needs for the final results using a specific requirement. A strict requirement will ask for a big difference between a_i and b_i (i.e., big $|a_i - b_i|$). If a requester needs high confidence on the results, then he can set a strict requirement.

One example of requirements is to set a minimum ratio between two classes' vote counts, i.e., to label the i-th instance as $+1$ if $a_i : b_i \geqslant c$ and as -1 if $b_i : a_i \geqslant c$, where the constant $c > 1$. If we estimate $P(Z_i = +1)$ with $\frac{a_i}{a_i + b_i}$, then this requirement is equivalent to setting a threshold on entropy, which is a commonly used measurement for information uncertainty. A larger c means a stricter requirement on the confidence.

Another possible requirement is to conduct a hypothesis test and approve the result when the observed labels are statistically significant. For example, Fisher exact test can be used [1], where the null hypothesis is that the observed labels are from a random guess and the alternative hypothesis is that they are not randomly guessed. Based on the null hypothesis, we can calculate the p-value, the probability that the observed labels and more extreme vote counts happen given that the workers provide random labels. When the null hypothesis is rejected with significance level of α, i.e., p-value $< \alpha$, we approve the result and determine the final label by majority voting. The significance level α is often set as a small value, usually 0.05, 0.1, or 0.2. A smaller α

[4]Note that in our definition a worker being noiseless does not mean that he provides correct labels for every instance, which is different from some literature.

gives a stricter requirement. We use the following example to illustrate how the Fisher exact test can be conducted.

EXAMPLE 2. *Suppose we set $\alpha = 0.1$, and the observed vote counts are $a_i = 1$ and $b_i = 4$. To calculate the p-value, we find the more extreme case, with $a_i = 0$ and $b_i = 5$. Therefore, the p-value $= \binom{5}{1}0.5^1 0.5^4 + \binom{5}{0}0.5^0 0.5^5 = 0.1875$, which is greater than α. Then the null hypothesis cannot be rejected, that is, there is no strong evidence that the observed labels are not randomly guessed. For this observed labeling result, it does not achieve the requirement yet.*

The aforementioned requirements can also be combined with other simple requirements such as minimum count of labels per instance $a_i + b_i > \delta_{min}$ and maximum count of labels per instance $a_i + b_i < \delta_{max}$.

Calculating Completeness

An instance is completed once it is confident based on the requirement. We define the completeness of the i-th instance as the ratio between the observed total vote counts $a_i + b_i$ and the minimum count of labels it needs to achieve the requirement, where the latter is denoted as $r(a_i, b_i|Z_i)$. At the beginning of labeling process, an instance has no labels, so its completeness is 0. Then it gradually gets more and more labels. Once the instance achieves the requirement, then the ratio becomes 1, which means the labeling process on that instance is completed and it needs no more labels (therefore, $a_i + b_i$ will not increase any more). Note that the completeness depends on not only the current vote counts and the requirement, but also on Z_i. We use the following example to illustrate how to calculate completeness.

EXAMPLE 3. *Suppose we have $a_i = 3$ and $b_i = 1$, and the requirement is the minimum ratio of 4. In the case that $Z_i = +1$, we can achieve the requirement if $a_i = 4$ and $b_i = 1$. Therefore, we need at least five labels to pass the requirement, i.e., $r(3, 1|Z_i = +1) = 4 + 1 = 5$. The completeness in this case is 4/5. However, if $Z_i = -1$, then we need at least $b = 12$ to achieve the requirement, which means minimum count of labels it needs is $r(3, 1|Z_i = -1) = 3 + 12 = 15$ labels, and the completeness is 4/15.*

From Example 3, it is clear that the denominator in the completeness $r(a_i, b_i|Z_i)$ is calculated based on the vote count of the opposite class of Z_i. That is, given $Z_i = +1$, the completeness is only determined by the vote count for -1(i.e. b_i) and vice versa. Therefore, we abbreviate $r(a_i, b_i|Z_i = +1) = r(b_i)$ and $r(a_i, b_i|Z_i = -1) = r(a_i)$. Table 2 shows examples of function $r(\cdot)$ with different input x' for several aforementioned requirements. These examples present that with x votes for the wrong class, how many total vote counts the i-th instance needs to achieve the corresponding requirement.

Expected Completeness

Since the completeness of the i-th instance depends on Z_i, which is a random variable, from a decision theory perspective we want to take an action that maximizes the expected completeness defined as follows:

$$\begin{aligned} V_i(a_i, b_i) &= \mathbb{E}(Completeness_i|a_i, b_i) \\ &= P(Z_i = +1|a_i, b_i)\frac{a_i + b_i}{r(b_i)} \\ &\quad + P(Z_i = -1|a_i, b_i)\frac{a_i + b_i}{r(a_i)}. \end{aligned} \tag{1}$$

Table 2: Some examples of $r(x)$ with different x'.

Requirement	x = 1	2	3	4	5
minimum ratio $c = 4$	5	10	15	20	25
minimum ratio $c = 5$	6	12	18	24	30
Fisher $p = 0.2$	5	8	10	12	15
Fisher $p = 0.1$	7	9	12	14	17
Fisher $p = 0.05$	8	11	13	16	18

If $P(Z_i = +1|a_i, b_i)$ is estimated based on empirical relative frequency $\frac{a_i}{a_i+b_i}$ (which is also the MLE estimator for Bernoulli(P_i)), then $V_i = \frac{a_i}{r(b_i)} + \frac{b_i}{r(a_i)} \leqslant 1$ and "=" can be satisfied only when the requirement is absolute consensus. This conflicts with our intuition, however, as the expected completeness should fall into the range of $[0, 1]$, and reach 1 when the labeling process is considered to be completed, i.e., when it meets the requirement $a_i + b_i = r(b_i)$ or $a_i + b_i = r(a_i)$.

In order to solve this problem, we propose to modify the estimation of $P(Z_i = +1)$ as following:

$$P(Z_i = +1|a_i, b_i) = \begin{cases} \frac{a_i}{a_i+b_i} + \frac{b_i}{r(b_i)} & \text{if } a_i > b_i, \\ 0.5 & \text{if } a_i = b_i, \\ \frac{a_i}{a_i+b_i} - \frac{a_i}{r(a_i)} & \text{if } a_i < b_i. \end{cases} \tag{2}$$

When the i-th instance achieves the requirement and $a_i > b_i$, the requester is confident about the result, so he will stop collecting labels for this instance and use $+1$ as the final class label. In such a case, a_i is usually much bigger than b_i and Z_i is believed to be $+1$ with confidence, so assigning $P(Z_i = +1|a_i, b_i) = 1$ is reasonable. In the case that the i-th instance is far from completed, $P(Z_i = +1|a_i, b_i)$ would still be close to $\frac{a_i}{a_i+b_i}$.

With the expected completeness defined in Eq. (1), the goal of achieving the requirement for as many instances as possible, while staying within the budget, is modeled as the following optimization problem:

$$\begin{aligned} \sup_{\pi} \quad & \mathbb{E}_\pi(\textstyle\sum_{i=1}^N V_i(a_i, b_i)) \\ s.t. \quad & \textstyle\sum_{i=1}^N a_i + b_i \leqslant T, \end{aligned} \tag{3}$$

where π is the policy to choose instances for labeling and $\mathbb{E}_\pi$ means the expectation taken over all the potential outcomes of the policy.

Markov Decision Process

A Markov decision process (MDP) is a natural framework for the optimization problem in Eq. (3). A MDP contains a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, transition probabilities from one state to another $\mathcal{P}$ and a real valued reward function $\mathcal{R}(\mathcal{S}, \mathcal{A})$.

In our problem setting, the set of states $\mathcal{S}$ consists of all the possible labeling observations that can be achieved under the budget. Since the budget is finite, $\mathcal{S}$ is also a finite set. At time t, we get exactly t labels, so the state is then:

$$\begin{aligned} \mathcal{S}^t = \{(a_i^t, b_i^t)_{i=1}^N | \quad & a_i^t, b_i^t \geqslant 0, \textstyle\sum_{i=1}^N a_i^t + b_i^t = t, \\ & a_i^t + b_i^t \leqslant \min(r(a_i^t), r(b_i^t))\}, \end{aligned} \tag{4}$$

where a_i^t and b_i^t are the vote counts at stage t.

The set of actions consists of the potential instances to be queried next, that is, the instances that do not achieve

the requirement yet. If we only obtain one label on one instance at each time, the transition probability is essentially $P(Z_{i^t}|S^t, i^t)$, where i^t is the chosen instance to label at stage t. As the state of the i-th instance at time t is directly determined by (a_{i^t}, b_{i^t}), it is identical with $P(Z_{i^t}|a_{i^t}, b_{i^t})$ defined in Eq. (2).

For the reward function $R(S, i)$, we use the result from [4] and define the stage-wise expected reward as:

$$R(S^t, i^t) = \mathbb{E}(V_{i^t}(a_{i^t}^{t+1}, b_{i^t}^{t+1}) - V_{i^t}(a_{i^t}^t, b_{i^t}^t)|S^t, i^t), \quad (5)$$

and the optimization becomes:

$$\sup_{\pi} \mathbb{E}_{\pi}(\sum_{t=0}^{T-1} R(S^t, i^t)). \quad (6)$$

This stage-wise reward function is to evaluate how much difference an extra label can make prior to the decision. There are basically two cases to consider: the next label y_{i^t} is $+1$ or -1. Correspondingly, the differences it may make are:

$$R_{i^t}^{+1} = V_{i^t}(a_{i^t}^t + 1, b_{i^t}^t) - V_{i^t}(a_{i^t}^t, b_{i^t}^t),$$
$$R_{i^t}^{-1} = V_{i^t}(a_{i^t}^t, b_{i^t}^t + 1) - V_{i^t}(a_{i^t}^t, b_{i^t}^t).$$

Then $R(S^t, i^t)$ can be calculated as:

$$R(S^t, i^t) = P(y_{i^t} = +1)R_{i^t}^{+1} + P(y_{i^t} = -1)R_{i^t}^{-1}. \quad (7)$$

With the defined MDP model, there are well-developed algorithms to compute the optimal policy, such as dynamic programming [18] and linear programming [23]. However, the state space defined in Eq. (4) is exponential with respect to the budget. The computation complexity in such a case is intractable. Therefore, we use a one-step look-ahead greedy strategy instead to approximate the optimal policy. Motivated by [4], we also change the immediate reward function $R(S^t, i^t) = \max(R_{i^t}^{+1}, R_{i^t}^{-1})$ for better performance. The greedy strategy will provide a sequential policy and choose the instance that has the maximum immediate reward, the one with the most optimistic outcome for completeness in our setting. A similar strategy to build a policy for a MDP problem is also applied in [4, 19]. Using the greedy strategy, the framework will label instances with the highest priority to easy ones that are also nearly complete.

Summary of the Basic Requallo Framework

So far, we have introduced the basic Requallo framework for a simple problem setting. The requester inputs his budget (or the maximum number of labels he can afford T) and his specific quality requirement. Based on the inputs, a sequential policy is provided to maximize the number of instances that achieve the requester's quality requirement. The framework is summarized in Algorithm 1.

Algorithm 1 Basic Requallo framework

Input: instances, maximum number of labels T, requester's requirement.
Output: labeling result for instances.

1: **while** $t \leqslant T$ and $\max_i R(S^t, i^t) > 0$ **do**
2: Based on the requirement, select an instance to label: $i_t = \arg\max_{i \in \{1,\dots,N\}} R(S^t, i^t)$.
3: Receive label $y_{i^t} \in \{+1, -1\}$
4: Update $a_{i^t}^{t+1}$ and $b_{i^t}^{t+1}$ according to y_{i^t}
5: Update $a_i^{t+1} = a_i^t$ and $b_i^{t+1} = b_i^t$ for all $i \neq i^t$
6: $t = t + 1$
7: **end while**
8: **return** Labeling results for instances.

3.2 Model Workers' Reliability

In the basic Requallo framework, we consider noiseless workers, that is, their labels only depend on the instances' true labels and difficulty levels. However, the assumption may be too strong in real-world crowdsourcing scenarios. If workers' varying reliabilities are considered, we may further save the budget in the labeling process by eliminating labels from unreliable workers or adjusting labels' weights based on the workers' reliability.

Suppose there are M workers, and the labels they provide on the i-th instance are $\{y_{i1}, \dots, y_{iM}\}$. Each worker has a reliability degree denoted by θ_j, where j is the worker's index. θ_j is defined as $P(y_{ij} = Z_i|Z_i)$, i.e., the probability that worker j provides the same label as the one provided by the noiseless worker. If $\theta_j = 1$, it indicates worker j is noiseless. The label from a worker can be viewed as an output from two layers of Bernoulli sampling. First y_{ij} is drawn from Bernoulli($P_i = P(Z_i = +1)$), and then with probability of $1 - \theta_j$, y_{ij} flips its sign. As a result, we have:

$$P(y_{ij} = +1) = \theta_j P_i + (1 - \theta_j)(1 - P_i),$$
$$P(y_{ij} = -1) = \theta_j(1 - P_i) + (1 - \theta_j)P_i.$$

Here we assume that $\theta_j \geqslant 0.5$. Otherwise we can flip the sign of his labels so that $\theta_j \geqslant 0.5$.

In the following, we discuss how to incorporate workers' reliability under different types of requirements.

Fisher Exact Test Based Requirements

In the Fisher exact test, the p-value is calculated based on the worst case scenario: everyone is random guessing. Unfortunately, the hypothesis already assumes $P(y_{ij} = +1) = 0.5$, so workers' reliability cannot be explicitly applied. However, if we find anyone is indeed random guessing, that worker is considered as a spammer[5] and his labels can be removed. By doing so, the completeness of the task will increase with a high probability.

EXAMPLE 4. *Let's revisit Example 2. Suppose the worker who answers $+1$ is a spammer. After remove his label, we get a result of $a_i = 0$ and $b_i = 4$. Therefore, p-value $= \binom{4}{0}0.5^0 0.5^4 = 0.0625$, which is smaller than α. The null hypothesis is rejected and this instance achieves the requirement. Its completeness is 1, so it needs no more labels.*

A simple way to detect spammers is to test if $\theta \approx 0.5$ based on the labeled instances that achieve the requirement and his labels on those instances. Note that for those instances, $P(Z_i = +1|a_i, b_i)$ is either 0 or 1 based on Eq. (2) and Z_i is provided by majority voting. In order to estimate θ_j more accurately, we only consider worker j when he provides a certain amount of labels on the labeled instances. Then we can either conduct a Fisher exact test or calculate the confidence interval of θ_j. If the observed labels from him cannot reject the hypothesis or 0.5 falls into the confidence interval, this worker is considered as a spammer and will be blocked for

[5]It should be noted that in some literature spammer has a different meaning. In this work, a spammer refers to a worker whose $\theta = 0.5$.

further labeling process. Some existing techniques [11,20,34] can also be applied.

Minimum Ratio Requirements

When using the minimum ratio requirement for vote counts between two classes, the key idea in the basic Requallo framework can still be applied, but we can readjust labels' weights based on the workers' reliability. By doing so, the expected completeness is then computed according to weighted vote counts instead of observed vote counts. Similarly, weighted voting can be used to obtain the final labels.

Given the labels from workers and their reliabilities $\{\theta_j\}$, we can estimate $P_i = P(Z_i = +1)$ based on the likelihood function:

$$f(y_{i1}, \ldots, y_{iM}|P_i, \{\theta_j\}_{j=1}^M) = \prod_{j=1}^{M} \{(\theta_j P_i + (1-\theta_j)(1-P_i))^{\mathbb{1}(y_{ij}=+1)} \times (\theta_j(1-P_i) + (1-\theta_j)P_i)^{\mathbb{1}(y_{ij}=-1)}\}, \quad (8)$$

where $\mathbb{1}(y_{ij} = +1) = 1$ if $y_{ij} = +1$, and $\mathbb{1}(y_{ij} = +1) = 0$ if $y_{ij} = -1$. Then the MLE estimator for P_i is

$$P_i = \arg\max f(y_{i1}, \ldots, y_{iM}|P_i, \{\theta_j\}_{j=1}^M) \quad s.t. \quad P_i \in [0, 1]. \quad (9)$$

Although the closed-form solution is not easy to derive, there are many numerical ways to solve this optimization problem. Note that P_i may be 1 or 0 if some of the workers are unreliable. To prevent this, pseudo-labels can be added and more discussions can be found in Section 3.3.

Based on P_i, we can readjust the vote counts that satisfy:

$$\tilde{a}_i + \tilde{b}_i = a_i + b_i,$$
$$\tilde{a}_i : \tilde{b}_i = P_i : (1 - P_i).$$

Here $\tilde{a}_i$ and $\tilde{b}_i$ can be viewed as weighted vote counts, and they are derived as follows:

$$\tilde{a}_i = P_i(a_i + b_i),$$
$$\tilde{b}_i = (1 - P_i)(a_i + b_i).$$

The vote counts a_i and b_i in Eq. (1) then can be replaced by the $\tilde{a}_i$ and $\tilde{b}_i$. Similar to Eq. (2), we modify the MLE estimator as:

$$P(Z_i = +1|\tilde{a}_i, \tilde{b}_i) = \begin{cases} \frac{\tilde{a}_i}{\tilde{a}_i+\tilde{b}_i} + \frac{\tilde{b}_i}{r(\tilde{b}_i)} & \text{if } \tilde{a}_i > \tilde{b}_i, \\ 0.5 & \text{if } \tilde{a}_i = \tilde{b}_i, \\ \frac{\tilde{a}_i}{\tilde{a}_i+\tilde{b}_i} - \frac{\tilde{a}_i}{r(\tilde{a}_i)} & \text{if } \tilde{a}_i < \tilde{b}_i. \end{cases} \quad (10)$$

When calculating the reward function Eq. (5), due to the limitation of crowdsourcing platforms, we cannot assign instances to specific workers and cannot forecast who will label the next instance. The worst case is considered then, i.e., the worker who currently has the lowest reliability degree (i.e. the θ_j that is closest to 0.5) will label the next instance.

Note that in Eq. (9) we assume θ_j is known. If the requester uses qualification exams or control questions among the instances, it can be easily inferred by the workers' performance on those tasks [15]. In the case that θ_j is unknown a priori, there are some existing techniques that can be used to estimate it under different crowdsourcing scenarios [3, 6, 14, 16, 17, 33–35, 40]. Many of them are aggregation algorithms that model worker reliability and conduct weighted vote to improve label accuracy. In the Requallo framework, they are also helpful to estimate labels' weighted vote counts. If a worker is likely to be a spammer, i.e., $\theta_j \approx 0.5$, we can also eliminate/block him from the labeling process.

3.3 Discussion and Practice Issues

Here we discuss several techniques to make the proposed framework more general and practical.

Multi-class Case

Multi-class labeling problems can be converted into binary classification problems as follows. The class with the highest votes is converted to +1 class, and the class with the second highest votes is converted to −1 class. If the most popular and the second most popular classes get similar votes, it still implies that the instance is hard to classify. On the other hand, if the most popular class predominates the votes, then we can regard it as an easy instance. This conversion is straightforward and the proposed Requallo framework can be applied to the corresponding binary problem.

Adding Pseudo-labels

At the early stage of the labeling process, the estimation of P_i may be unreasonable when either $a_i = 0$ or $b_i = 0$. The easiest solution to this problem is to give each instance pseudo-labels. For the model that incorporates workers' reliability, the pseudo-labels are considered to be from noiseless workers. The pseudo-labels can also be interpreted as prior knowledge from Bayesian point of view. In the experiments, we use one pseudo-label for each class as the initial label.

Choosing a Proper Requirement

If the requirement is not set properly, it is possible that no instances can achieve the requirement when the budget is exhausted. The requester need to keep in mind that crowd workers are usually non-experts, and normally do not receive any special training. Therefore, the requirement should be reasonable for the task difficulty as well as the budget. If the requester does not have a good estimate of task difficulty, we suggest to start with a small sample of instances and set a low requirement. If the instances can easily achieve the requirement, then gradually raise the bar. When the requirement is settled, all the instances can be tested.

4. EXPERIMENTS

In this section, we test the proposed Requallo framework to solve the budget allocation problems on two real-world crowdsourcing tasks and one simulated task. The results clearly demonstrate the advantages of Requallo framework when the budget is tight. We also provide some guidelines to requesters with different size of budgets and different needs.

4.1 Experiment Setup

In the experiments, we assume the basic setting as described in Section 3.1. We use one unit of the budget for one label.

Compared Methods

There are existing methods that consider budgets as discussed in Section 2, but some of them are not applicable when the budget is tight and need major modifications to fit our settings. Thus, the following budget allocation policies are compared. All of them are sequential labeling strategies.

- *Random*. This policy will randomly choose an instance and ask for one label.
- *Looping*. This strategy will pick the instances sequentially and ask for one label each time. Once the last instances is reached, it returns back to the first.
- *Opt-KG*. This budget allocation policy is proposed in [4]. The optimization goal for this policy is to maximize the overall accuracy and an MDP is formulated. It greedily chooses the next instance which has the largest possible reward. The prior distribution is set as Beta$(1, 1)$ as suggested in the original paper.
- *LU*. This policy is proposed in [24]. As the full version of this method is designed for active learning, which is different from our problem setting, we only adopt the label uncertainty component (Eq. (4) in [24]) as the baseline and disregard the model uncertainty estimation. LU will choose the instance that has the highest label uncertainty in each iteration.
- *Requallo-c4*. This budget allocation policy is created by the proposed Requallo framework based on the minimum ratio of 4. We find that it is essentially the same policy as seq-lindley with a threshold of -0.73, which is proposed in [19][6]. That is, Requallo-c4 and seq-lindley will choose the same instance to label next. Although using different value functions, seq-lindley also adopts an MDP framework and a greedy algorithm to approximate the optimum policy. However, seq-lindley cannot take requirements other than the minimum ratio, while Requallo is more general. Thus, we can view seq-lindley as a special case of the proposed Requallo framework.
- *Requallo-p0.2*. This budget allocation policy is created by Requallo framework based on the requirement of Fisher exact test with a significant level of 0.2. In addition, we also add a maximum label count requirement as neither class can receive more than 14 labels.

As the proposed Requallo method can provide flexible budget allocation policies according to different requirements, we mainly use Requallo-p0.2 to compare with the baseline methods because the requirement is moderate and suitable for many tasks. Other Requallo policies are compared to provide a guideline for choosing proper requirements.

Performance Measures

We measure the performance from two aspects: quantity and quality. Due to the fact that baseline policies and the proposed Requallo policies have quite different goals, it is unfair to the baseline policies to measure quantity and quality only on the instances that achieve the requirements. It is also unfair to the Requallo policies to measure the performance without considering the requirements. Therefore, we choose a compromise and measure the performance with the minimum label count requirement, i.e., one instance needs to get at least three labels in order to be considered as labeled. The label count requirement is one of the most widely used requirements in crowdsourcing tasks. With the minimum requirement of 3, it is also one of the mildest in practice.

[6]Note that their problem setting is different from ours. Some modifications on the original method are necessary to fit our setting.

Under the minimum label count requirement, quantity is measured as the total number of labeled instances, i.e., the number of instances that have at least three labels. Quality is measured from two aspects: absolute count and accuracy. The former is defined as the number of correctly labeled instances. The latter is the percentage of correctly labeled instances in all labeled instances.

4.2 Real-World Crowdsourcing Tasks

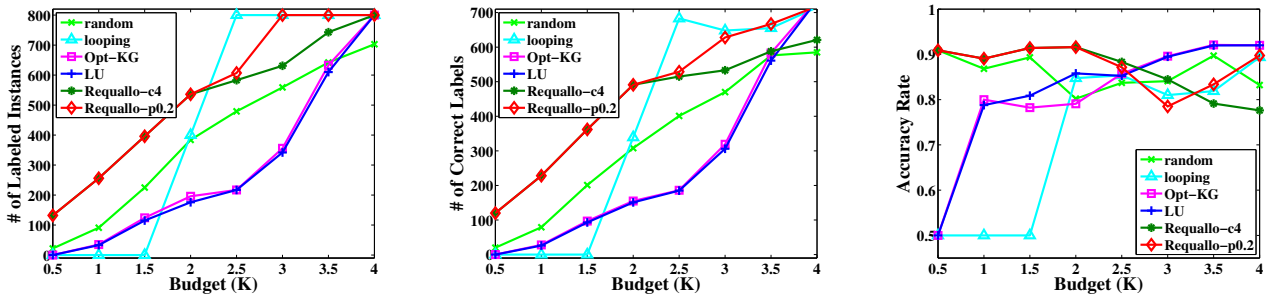
In this section, we show the experimental results on two real-world datasets: the RTE dataset and the Game dataset. Comparing with the baselines, the Requallo framework shows a better balance for the trade-off between quantity and quality under a tight budget.

RTE Dataset [28]. This crowdsourcing task is conducted on mTurk for recognizing textual entailment (RTE). The dataset is publicly available[7] and considered to be a benchmark for crowdsourcing tasks. It contains 800 instances, and each one is queried to 10 different workers for a binary label. Ground truth for every instance is also provided, and 700 out of 800 instances can get correct labels from majority voting. More detailed descriptions about this dataset can be found in the original paper [28].

Game Dataset [2, 13]. This crowdsourcing task is conducted using an Android app based on a TV game show "Who Wants to Be a Millionaire". When a question is displayed on the show, the Android app simultaneously sends the question and the four candidate answers to the app users, and then users send their answers back voluntarily. Ground truth information is also recorded as revealed by the anchor of the show. The original dataset contains 2169 questions and 221653 answers from 38196 users. Here, a question can be viewed as an instance and a user's answer can be viewed as a label from a worker. The following preprocessing steps are applied on this dataset: 1) Instances with fewer than 20 labels are removed. By doing so, the policies can fully demonstrate their characteristics when the budget increases. 2) Misleading instances are removed, i.e., the instances on which majority voting gives incorrect answers. Due to the nature of the game show, it contains some tricky questions that are intended to trap people, and most people would choose the wrong answer. This kind of design is highly discouraged in regular crowdsourcing tasks. Therefore, removing those instances is more reasonable for crowdsourcing tasks. 3) Since the original questions are multiple-choice, we convert them into binary problems using the technique discussed in Section 3.3. After the preprocessing, Game dataset contains 1709 instances and 158385 labels.

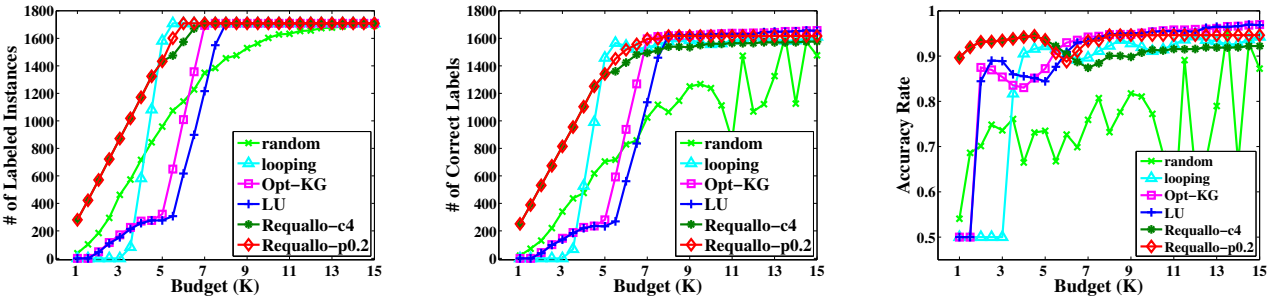
To test the behavior of different budget allocation policies, we start from a very tight budget which is not enough to label all instances once, and then gradually increase the budget. In the RTE dataset, each instance has only 10 labels, so we focus more on the tight budget and stop increasing the budget when it reaches 4000, i.e., each instance has five labels on average. In the Game data, since more labels are available for each instance, we explore the policies until the budget reaches 15000, i.e., each instance has about nine labels on average. Figures 1 and 2 demonstrate the performance of different budget allocation policies on the two real-world crowdsourcing tasks respectively.

[7]http://sites.google.com/site/nlpannotations/



(a) Quantity (b) Absolute Count (c) Accuracy

Figure 1: Comparison of Budget Allocation on RTE Dataset



(a) Quantity (b) Absolute Count (c) Accuracy

Figure 2: Comparison of Budget Allocation on Game Dataset

In Figures 1a and 2a, the quantity of labeled instances is compared. The looping policy has no labeled instance until the budget reaches $2N$ (N is the number of instances), when every instance has exactly two labels. Then the quantity linearly increases to N as the budget increases from $2N$ to $3N$, when every instance has exactly three labels. If the budget is $3N$, then looping provides the best quantity. Random can get some instances labeled under a tight budget, but needs a big budget to reach the maximum quantity. Note that Opt-KG and LU have no instances labeled until the budget is greater than N. This is because both Opt-KG and LU will query one label for each instance at the initial stage. The quantity increases fast when the budget is higher than $3N$. Comparing with baseline policies, the proposed Requallo-p0.2 and Requallo-c4 can provide a remarkable amount of labeled instances even with a budget tighter than $2N$. When the budget is around $3N$, only looping can provide a slightly higher quantity.

Absolute counts of correctly labeled instances are compared in Figures 1b and 2b. The pattern is similar to the ones presented in quantity comparison. When the budget is tight, Requallo-p0.2 and Requallo-c4 show a big advantage. When the budget is sufficient, Requallo-p0.2 still demonstrates a competitive performance, even though the quality requirement is not strict. Note that when the budget is small, Requallo-p0.2 and Requallo-c4 have the same performance. The reason is that due to the overlap of their corresponding requirements, these two policies will choose the same instance to label at next step when all instances have less than three labels. However, when some instances have more labels, these two policies will choose different instances because the requirements differ then. On the Game dataset, Requallo-p0.2 stops labeling process after spending a cost of 7715 because all instances achieve the requirement. If the budget is higher than 7715, the extras will be saved.

Figures 1c and 2c presents the change of accuracy. As looping, Opt-KG and LU have no labeled instance at the initial stage, we use 0.5 to indicate that the result is uninformative. The looping policy gives a wavy but slightly increasing accuracy rate once every instance is labeled. The accuracy rate of the random policy fluctuates widely. Opt-KG and LU perform similarly. Both policies improve the accuracy when the budget increases and can achieve good results when the budget is sufficient. However, when the budget is tight, their accuracy is unsatisfactory.

Requallo-p0.2 provides a high accuracy except a low ebb when the budget is around $3.5N$. This pattern is subtle in Figure 1c because the budget limit is low, but it is clear in Figure 2c. This is because when the budget is tight, Requallo-p0.2 gives easy instances priority so that the completeness can be maximized. It also leads to a high accuracy rate as shown in the figures. When the budget increases, easy instances are all labeled, and Requallo-p0.2 starts to work on harder instances. At that time, however, there is still not enough budget for sufficient labels on those hard questions. As the evaluation is not conducted with the original requirement, there are some labeled instances that do not achieve the requirement yet. Therefore, the increasing speed of quantity is faster than the increasing speed of correctly labeled instances. This leads to the low ebb on accuracy. As the budget keeps increasing, Requallo-p0.2 can allocate sufficient labels on hard questions, so the accuracy

raises back. A similar pattern happens to Requallo-c4. The low ebb occurs when the budget is around $4.5N$. The minimum ratio requirement is stricter than the Fisher exact test requirement on moderately hard questions, so once the easy instances are completed, Requallo-c4 intensively puts more effect on some hard questions. This slows down the increase of quantity, which causes the delay of low ebb on accuracy. The accuracy also raises back, but is slower.

In order to give some guidelines for requesters with different sizes of budgets and quality needs, other Requallo policies based on various requirements are also compared on Game dataset. Requallo-p0.2, Requallo-p0.1, and Requallo-p0.05 are the policies created based on Fisher exact test with different significance levels. The lower the p-value, the stricter the requirement. Requallo-c4 and Requallo-c5 are the policies created base on minimum ratio of 4 and 5 respectively. For these policies, we also adopt the same maximum label count requirements as mentioned in Section 4.1 as additional requirements. In addition, we show Requallo-m3, the policy with only minimum label count requirement, as a reference. Compared with the minimum ratio requirement, Fisher exact test based requirements are stricter at the initial stage, but looser when number of labels is relatively big. To better illustrate the effect of different requirements, we let the policies keep choosing instances until they are completed based on the requirement. The final labeling results are examined and Table 3 summarizes the performance.

Table 3: Comparison of Different Requallo Policies

Method	Cost	# Instances	# Correct	Accuracy
Requallo-p0.2	7715	1662	1587	0.9549
Requallo-p0.1	11191	1597	1558	0.9756
Requallo-p0.05	13878	1517	1493	0.9842
Requallo-c4	8689	1567	1518	0.9687
Requallo-c5	11266	1489	1464	0.9832
Requallo-m3	5127	1709	1580	0.9245

In Table 3, "Cost" is the total number of labels that each policy uses to finish labeling process. "# Instances" is the number of instances that achieve the corresponding requirement except for the maximum label count requirement. "# Correct" shows the number of correctly labeled instances among those who achieve the requirement. "Accuracy" is # Correct divided by # Instances. This table confirms our intuition of the trade-off between quantity and quality of the results, and the effect of different requirements. If a requester wants high quality results, he can set a strict requirement, but should expect a lower quantity of labeled instances or a higher cost. If the requester has a tight budget and no strong preference on quality, Requallo-p0.2 is a decent choice.

4.3 Simulated Crowdsourcing Task

As many crowdsourcing tasks are conducted to obtain training labels for machine learning problems, a simulation of such a scenario is created in this section, from crowdsourced labeling to training and finally testing. The Splice dataset from LIBSVM[8] is used as the target machine learning task. The original dataset contains 1000 training instances and 2175 testing instances. For each instance, there are 60 features and a binary label.

We first simulate the labels provided by crowd workers. A linear SVM classifier is learned on the training set and it provides a probability of being in class $+1$ for each training instance, which is used as $P(Z_i = +1)$. Random samples are drawn from Bernoulli($P_i = P(Z_i = +1)$) as the labels provided by noiseless workers for instance i. Under the same budget (from 500 to 5000), we then apply each policy and form a new training set by replacing the original labels with the labels given by the policy for all training instances. The same minimum label count requirement is used as in previous experiments. If a policy does not provide a label for an instance, it will be eliminated from the training set. Finally, new linear SVM classifiers are learned on the new training sets and applied to the test set.

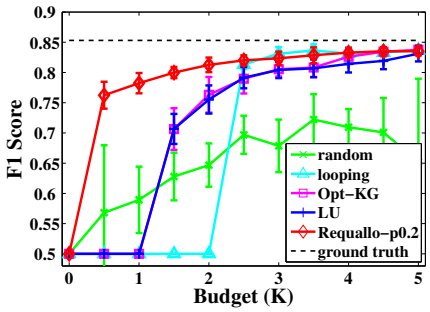


Figure 3: F1 Score on Test Data.

Due to the randomness in generating workers' labels, we run the simulation for 10 times and report the errorbar (standard deviation) of the results. The performance is evaluated as F1 score on testing data. A higher F1 score means a better performance. Figure 3 summarizes the results. Here we omit the results for Requallo-c4 because Requallo-p0.2 has almost the same performance. The black dash line represents the performance if the classifier is trained on the original training data, which is 0.853. Generally speaking, quantity and quality of training data are two of the most important factors for a good classifier. When the budget is tight, Requallo-p0.2 shows great advantages over baseline policies because of its excellent performance on both quantity and quality, which is also shown on the real-world crowdsourcing tasks. On the other hand, baseline policies need more labels to build up a good training set.

5. CONCLUSIONS

Crowdsourcing under a tight budget brings new unique challenges. Most existing budget allocation methods, which assume a sufficient budget is available, cannot achieve desired labeling quality if the budget is tight. As different requesters and tasks may have different needs on label quality, we proposed a framework that allows requesters specify their quality requirement on the results. The proposed framework can guarantee the requesters' desired quality while trying to maximize quantity under the budget. A Markov decision process is used to model the labeling process. To tackle the computational challenge, an approximate policy is built via a one-step look-ahead greedy algorithm. The policy sequentially chooses the next instance which can provide the highest reward on task completeness. The framework is further

[8]http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

extended to consider workers' reliability, so that unreliable workers can be eliminated from labeling process and reliable workers can have a higher weight in the vote counts. Experiments on both real-world and simulated crowdsourcing tasks demonstrate remarkable advantages of the proposed framework under a tight budget.

6. REFERENCES

[1] A. Agresti. A survey of exact inference for contingency tables. *Statistical Science*, pages 131–153, 1992.

[2] B. I. Aydin, Y. S. Yilmaz, Y. Li, Q. Li, J. Gao, and M. Demirbas. Crowdsourcing for multiple-choice question answering. In *Proc. of IAAI*, 2014.

[3] Y. Bachrach, T. Graepel, T. Minka, and J. Guiver. How to grade a test without knowing the answers—a Bayesian graphical model for adaptive crowdsourcing and aptitude testing. In *Proc. of ICML*, pages 1183–1190, 2012.

[4] X. Chen, Q. Lin, and D. Zhou. Optimistic knowledge gradient policy for optimal budget allocation in crowdsourcing. In *Proc. of ICML*, pages 64–72, 2013.

[5] N. Dalvi, A. Dasgupta, R. Kumar, and V. Rastogi. Aggregating crowdsourced binary ratings. In *Proc. of WWW*, pages 285–294, 2013.

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

[7] P. Donmez, J. G. Carbonell, and J. G. Schneider. A probabilistic framework to learn from multiple annotators with time-varying accuracy. In *Proc. of SDM*, pages 826–837, 2010.

[8] S. Faradani, B. Hartmann, and P. G. Ipeirotis. What's the right price? Pricing tasks for finishing on time. *Proc. of HCOMP*, 2011.

[9] P. G. Ipeirotis and E. Gabrilovich. Quizz: Targeted crowdsourcing with a billion (potential) users. In *Proc. of WWW*, pages 143–154, 2014.

[10] H. Jin, L. Su, D. Chen, K. Nahrstedt, and J. Xu. Quality of information aware incentive mechanisms for mobile crowd sensing systems. In *MobiHoc*, pages 167–176, 2015.

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

[12] H. Li, B. Zhao, and A. Fuxman. The wisdom of minority: Discovering and targeting the right group of workers for crowdsourcing. In *Proc. of WWW*, pages 165–176, 2014.

[13] Q. Li, Y. Li, J. Gao, L. Su, B. Zhao, M. Demirbas, W. Fan, and J. Han. A confidence-aware approach for truth discovery on long-tail data. *PVLDB*, 8(4), 2014.

[14] Q. Li, Y. Li, J. Gao, B. Zhao, W. Fan, and J. Han. Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation. In *Proc. of SIGMOD*, pages 1187–1198, 2014.

[15] Q. Liu, A. Ihler, and M. Steyvers. Scoring workers in crowdsourcing: How many control questions are enough? In *NIPS*, pages 1914–1922, 2013.

[16] F. Ma, Y. Li, Q. Li, M. Qiu, J. Gao, S. Zhi, L. Su, B. Zhao, H. Ji, and J. Han. Faitcrowd: Fine grained truth discovery for crowdsourced data aggregation. In *Proc. of KDD*, pages 745–754, 2015.

[17] C. Meng, W. Jiang, Y. Li, J. Gao, L. Su, H. Ding, and Y. Cheng. Truth discovery on crowd sensing of correlated entities. In *Proc. of SenSys*, pages 169–182, 2015.

[18] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[19] V. Raykar and P. Agrawal. Sequential crowdsourced labeling as an epsilon-greedy exploration in a markov decision process. In *Proc. of AISTATS*, pages 832–840, 2014.

[20] V. C. Raykar and S. Yu. Ranking annotators for crowdsourced labeling tasks. In *NIPS*, pages 1809–1817, 2011.

[21] V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *The Journal of Machine Learning Research*, 11:1297–1322, 2010.

[22] J. Redi and I. Povoa. Crowdsourcing for rating image aesthetic appeal: Better a paid or a volunteer crowd? In *Proc. of the International ACM Workshop on Crowdsourcing for Multimedia*, pages 25–30, 2014.

[23] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Egnlewood Cliffs, 1995.

[24] V. S. Sheng, F. Provost, and P. G. Ipeirotis. Get another label? Improving data quality and data mining using multiple, noisy labelers. In *Proc. of SIGKDD*, pages 614–622, 2008.

[25] Y. Singer and M. Mittal. Pricing mechanisms for crowdsourcing markets. In *Proc. of WWW*, pages 1157–1166, 2013.

[26] A. Singla, I. Bogunovic, G. Bartók, A. Karbasi, and A. Krause. Near-optimally teaching the crowd to classify. In *Proc. of ICML*, pages 154–162, 2014.

[27] A. Singla and A. Krause. Truthful incentives in crowdsourcing tasks using regret minimization mechanisms. In *Proc. of WWW*, pages 1167–1178, 2013.

[28] R. Snow, B. O'Connor, D. Jurafsky, and A. Y. Ng. Cheap and fast — but is it good? Evaluating non-expert annotations for natural language tasks. In *Proc. of EMNLP*, pages 254–263, 2008.

[29] Y. Tian and J. Zhu. Learning from crowds in the presence of schools of thought. In *Proc. of SIGKDD*, pages 226–234, 2012.

[30] L. Tran-Thanh, T. D. Huynh, A. Rosenfeld, S. D. Ramchurn, and N. R. Jennings. Budgetfix: Budget limited crowdsourcing for interdependent task allocation with quality guarantees. In *Proc. of AAMAS*, pages 477–484, 2014.

[31] L. Tran-Thanh, M. Venanzi, A. Rogers, and N. R. Jennings. Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks. In *Proc. of AAMAS*, pages 901–908, 2013.

[32] L. Von Ahn and L. Dabbish. Labeling images with a computer game. In *Proc. of CHI*, pages 319–326, 2004.

[33] P. Welinder, S. Branson, P. Perona, and S. J. Belongie. The multidimensional wisdom of crowds. In *NIPS*, pages 2424–2432, 2010.

[34] P. Welinder and P. Perona. Online crowdsourcing: Rating annotators and obtaining cost-effective labels. In *CVPRW*, pages 25–32, 2010.

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

[36] H. Wu, J. Corney, and M. Grant. Relationship between quality and payment in crowdsourced design. In *Proc. of CSCWD*, pages 499–504, 2014.

[37] Y. Xiao, Y. Zhang, and M. van der Schaar. Socially-optimal design of crowdsourcing platforms with reputation update errors. In *ICASSP*, pages 5263–5267, 2013.

[38] Q. Zhang, Y. Wen, X. Tian, X. Gan, and X. Wang. Incentivize crowd labeling under budget constraint. In *Proc. of IEEE Infocom*, 2015.

[39] H. Zheng, D. Li, and W. Hou. Task design, motivation, and participation in crowdsourcing contests. *International Journal of Electronic Commerce*, 15(4):57–88, 2011.

[40] D. Zhou, S. Basu, Y. Mao, and J. C. Platt. Learning from the wisdom of crowds by minimax entropy. In *NIPS*, pages 2195–2203, 2012.

[41] Y. Zhou, X. Chen, and J. Li. Optimal PAC multiple arm identification with applications to crowdsourcing. In *Proc. ICML*, pages 217–225, 2014.