Debiasing Crowdsourced Batches

Honglei Zhuang, Aditya Parameswaran, Dan Roth, Jiawei Han
Department of Computer Science
University of Illinois at Urbana-Champaign
{hzhuang3, adityagp, danr, hanj}@illinois.edu

ABSTRACT

Crowdsourcing is the de-facto standard for gathering annotated data. While, in theory, data annotation tasks are assumed to be attempted by workers independently, in practice, data annotation tasks are often grouped into batches to be presented and annotated by workers together, in order to save on the time or cost overhead of providing instructions or necessary background. Thus, even though independence is usually assumed between annotations on data items within the same batch, in most cases, a worker's judgment on a data item can still be affected by other data items within the batch, leading to additional errors in collected labels. In this paper, we study the data annotation bias when data items are presented as batches to be judged by workers simultaneously. We propose a novel worker model to characterize the annotating behavior on data batches, and present how to train the worker model on annotation data sets. We also present a debiasing technique to remove the effect of such annotation bias from adversely affecting the accuracy of labels obtained. Our experimental results on both synthetic data and real-world data demonstrate the effectiveness of our proposed method.

1. INTRODUCTION

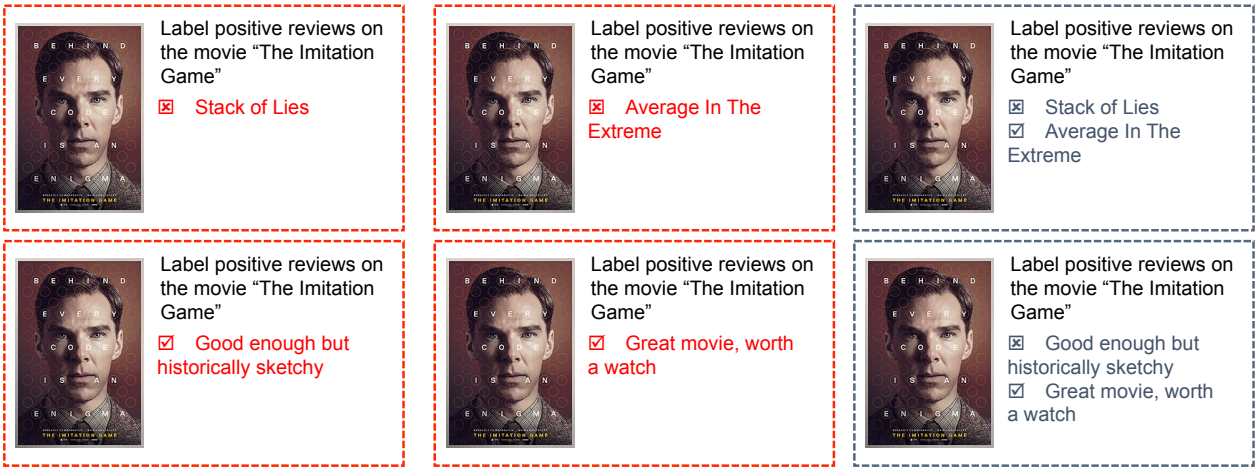
Crowdsourcing provides an efficient method to annotate data on a large scale for various machine learning tasks by employing a massive workforce drawn from global Internet users. Popular online crowdsourcing platforms include Amazon Mechanical Turk[1] and CrowdFlower[2]. However, while crowdsourcing is relatively cheap compared to employing experts, getting large quantities of training data annotated by crowds (say thousands, or millions of data items) can be rather expensive.

A key mechanism, often employed in practice for reducing costs, is *batching*, i.e., grouping multiple data items (to be annotated together) into one single task as a batch. Batching can save significant monetary costs, since the necessary instructions and background for completing the task needs to be provided just once for the entire batch. Thus, the worker will spend less time on reviewing these instructions, and more time on annotating data items, and therefore will be able to annotate more data items within the same time. For instance, consider a scenario where a worker has to judge whether a comment is relevant to a document. Here, making a judgment for each comment requires reading through the entire document. Instead, with batching, the worker only needs to read the entire document once, and then make a judgment for all the comments in the batch. In fact, even from the workers' point of view, it is also more attractive to label batches of data items as they can save time on switching between different tasks. Furthermore, once they start on a batch, they no longer have to "fight" for other high paying or attractive tasks.

However, even though batching is an attractive option in practice due to its cost and time savings, having workers annotate batches can lead to severe correlation between annotations within batches. For example, say we have a task of annotating whether a review of the movie "The Imitation Game" crawled from IMDb is positive. As illustrated in Figure 1(a), if we only show one review to be judged as part of each crowdsourcing unit task, workers will have to spend some time reading the instructions and possibly looking up the movie before they can make a single judgment on a review. Although judgments are likely to be independent, this way of assigning work is too costly to be practical. Instead, if we assemble multiple reviews of the same movie into a batch, as shown in Figure 1(b), workers can make multiple judgments after they read the instructions. Nevertheless, in this case, the annotation of different reviews might interfere with each other. For example, when the review "Average In The Extreme" does not seem like a positive review per se (Cf. top right in Figure 1(a)), when grouped with the review "Stack of Lies", it looks much more like a positive review (Cf. top in Figure 1(b)). Similarly, when the review "Good enough but historically sketchy" looks quite positive by itself (Cf. bottom left in Figure 1(a)), it does not look as positive as a strongly effusive review simply saying "Great movie", as shown in the bottom of Figure 1(b). Thus, overall these effects might be undesirable and misleading as it is inconsistent with the case when workers make independent judgments. Therefore, it is challenging to ascertain true labels of data items in batches.

So far, there has been little to no work in exploring the the possible annotation error introduced by grouping data items into batches. Although batching data items has been adopted in many crowdsourced tasks such as sorting [17], object recognition [32] or clustering [10], and anecdotally very widely used in practice, the assumption is often that the annotations are collected independently, which is not the case. While there is limited work on judging data items in sequence [18, 26, 27], it is not directly applicable to our setting where a batch of data items are presented and annotated in parallel. Our previous research [36] also noticed this specific

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



(a) Data items judged independently (b) Data items judged in a batch

Figure 1: Example of correlation between annotations on data items in the same batch. Workers are asked to label whether a review on the movie "The Imitation Game" crawled from IMDb is positive. Assign each review-movie pair to different workers separately can be costly, while assigning a batch of reviews together with a movie to workers might affect workers' judgments.

type of annotation bias, but instead of focusing on debiasing, we exploited the bias to develop an active learning algorithm aiming to improve a certain classifier performance. We defer the detailed discussion of the related work to Section 7.

There are several research challenges in solving this problem. First, how do we model workers' behavior when they make judgments in batches? Second, how do we leverage the model to debias the crowdsourced annotation of data batches? We make the following contributions in answering these questions:

1. *Proposing an interpretable worker annotation model on batches of data.* We propose a novel worker model for binary annotation behavior with data items presented as batches. The model incorporates independent judgments and batch judgments based on ranking. Different from the factor graph model in our previous work [36], we focus on obtaining the true labels of data items instead of improving classifier performance.

2. *Debiasing annotation data obtained as batches.* Based on our proposed worker model, we provide an algorithm to debias the inferred labels when they are collected from data items in batches.

3. *Conducting experiments on a real-world crowdsourcing platform.* We conduct experiments on both synthetic and real-world crowdsourcing data sets to verify the effectiveness of our proposed model and debiasing strategies. Experimental results show the effectiveness of our debiasing method over other baselines.

The rest of this paper is organized as follows: Section 2 introduces the basic concepts and formalizes the research problem; Section 3 proposes the worker model for annotating batches of data; Section 4 presents a strategy to debias batch annotations; Section 5 describes experimental results; Section 6 discusses extensions of our proposed method; Section 7 presents related work and Section 8 concludes.

2. PRELIMINARIES

In this section, we formally define the concepts and notations we use in this paper; we then formalize the problem of debiasing crowdsourced batches.

2.1 Basic Concepts and Terminology

First we need to formalize several basic concepts in a crowdsourcing platform. Suppose we are given a set of data items $X = \{x_i\}$, where $i = 1, \ldots, n$. Each data item is associated with a label $y_i \in \mathcal{Y}$, where $Y = \{y_i\}_{i=1}^n$. In following discussion, we focus on a binary classification task, where $\mathcal{Y} = \{0, 1\}$, but our framework generalizes to multi-class or rating cases seamlessly (Cf. Section 6). According to a standard formalization in learning theory for binary classification, we suppose each (x_i, y_i) is generated from a joint probability distribution $P_{\mathcal{X}\mathcal{Y}}$. We define an inherent score η_{x_i} to be the conditional probability $P(y_i = 1|x_i)$. For simplicity, we denote the inherent score as η_i.

In a job or task submitted to a crowdsourcing platform, we can assemble several data items into a batch. Each batch $\mathbf{b}_j$ is represented by a set of indices of data items in the batch, denoted as $\{b_{j1}, \ldots, b_{jk}\}$, where k is the size of a batch. To be strict, data items in the batch should be represented by $\mathbf{x}_j = \{x_{b_{j1}}, \ldots, x_{b_{jk}}\}$. However, for simplicity, we denote data items in the batch specified by $\mathbf{b}_j$ as $\{x_{j1}, \ldots, x_{jk}\}$. Similarly, we define $\mathbf{y}_j = \{y_{j1}, \ldots, y_{jk}\}$ to be true labels associated with data items in $\mathbf{x}_j$, where y_{jl} is the true label of x_{jl} according to Y for $\forall 1 \leq l \leq k$. In CrowdFlower language, a batch corresponds to a single "unit", where a worker has to judge the entire unit at the same time; in Mechanical Turk language, a batch corresponds to a single "HIT" (short for Human Intelligence Task). Usually, data items in the same batch might share the same context, background, or the same instruction, in order to reduce the overhead. For example, if one is asked to judge whether a review about a restaurant is positive or negative, it might save time for workers if reviews of the same restaurant are grouped into the same batch, as they only need to read the description of the restaurant once before they can make multiple judgments on different reviews.

As we assemble data items into batches, each worker has to judge the entire batch as a single judgment. Given a batch $\mathbf{b}_j$,

the judgment provided by a worker can be represented as $\mathbf{y}'_j = (y'_{j1}, \ldots, y'_{jk})$, where $y'_{jl} \in \mathcal{Y}$ is the annotation of data item corresponding to x_{jl}, provided by the worker. Noting that the worker annotation $\mathbf{y}'_j$ can be different from the true label $\mathbf{y}_j$. We refer to worker annotation as "annotation", while the ground-truth label is referred to as simply the "label".

In CrowdFlower, as a judgment can only be made based on a unit, workers are not allowed to submit partial results on a batch (as with Mechanical Turk). However, one can always add an "unknown" option for every data item, so that the workers can provide partial results on a batch. For simplicity, we consider no partial judgments in the rest of the paper.

Now, we are in a position to give a formal definition for a batch of data items:

DEFINITION 1 (BATCH). *Given a data set (X, Y), a batch of data items with size k extracted from the given data set can be represented as $(\mathbf{b}_j, \mathbf{x}_j, \mathbf{y}_j, \mathbf{y}'_j)$, where $\mathbf{b}_j = (b_{j1}, \ldots, b_{jk})$ is a set of indices for X and Y; $\mathbf{x}_j = \{x_{j1}, \ldots, x_{jk}\}$ is a set of all the data items, indexed by $\mathbf{b}_j$ (i.e. $x_{jl} = x_{b_{jl}}$); $\mathbf{y}_j = \{y_{j1}, \ldots, y_{jk}\}$ consists of the corresponding true labels of data items in $\mathbf{x}_j$, also indexed by $\mathbf{b}_j$; $\mathbf{y}'_j = \{y'_{j1}, \ldots, y'_{jk}\}$ is the worker annotation on the set of the batch.*

Additionally, a set of batches can be defined as:

DEFINITION 2. *Given a data set (X, Y), a set of batches extracted from the given data set is denoted as $A = (B, X_B, Y_B, Y'_B)$, where $B = \{\mathbf{b}_j\}_{j=1}^m$ consists of the indices of each batch; $X_B = \{\mathbf{x}_j\}_{j=1}^m$ is the set of data item batches, with their corresponding true labels $Y_B = \{\mathbf{y}_j\}_{j=1}^m$ and worker annotations $Y'_B = \{\mathbf{y}'_j\}_{j=1}^m$.*

Remarks. 1) Notice that a data item $x_i \in X$ may certainly appear in multiple batches in A. If data items in two different batches share the same index as indicated by corresponding item in $\mathbf{b}_j$, they refer to the same data item in X; 2) For the sake of fully utilizing the workforce of crowds, without loss of generality, we focus on the scenario when all batches have the identical size k. However, our model generalizes to the case when batches have different sizes; 3) In some real world crowdsourcing platforms, a batch can actually be judged by multiple workers, which means there could be multiple $\mathbf{y}'_j$'s associated to a single $(\mathbf{b}_j, \mathbf{x}_j, \mathbf{y}_j)$— for instance, this is referred to as multiple *assignments* on Mechanical Turk. However, for the purposes of debiasing, it is equivalent to regard a single batch as multiple identical batches, and associate each batch with a unique judgment made by different workers.

2.2 Problem Definition

Based on the concepts described thus far, we can formalize the problem of debiasing crowdsourced batches as the following:

PROBLEM 1 (DEBIASING CROWDSOURCED BATCHES). *Suppose we have a labeled data set (X_L, Y_L) with Y_L known, as well as its extracted batches and their crowdsourced annotation $(B_L, X_{B_L}, Y_{B_L}, Y'_{B_L})$. If we are then given another unlabeled data set X_U, as well as its extracted batches and crowdsourced annotation (B_U, X_{B_U}, Y'_{B_U}), the objective is to infer the true labels Y_U associated with X_U from the crowdsourced annotation.*

Notice that our problem formulation as described above requires as input labeled and annotated data items for training purposes. In practice, the labeled data for training can be collected from the "test questions" with ground-truth labels, inserted by the crowdsourcing platform for the purpose of quality control and monitoring of workers. The usage of test questions is standard practice: As an example, in CrowdFlower, all workers have to attempt a certain number of test questions with correct labels and need to achieve an accuracy over a certain threshold (*e.g.* 70%) before they can proceed to work on the regular task(s). Also, additional hidden data items with known labels can be inserted into the regular tasks to monitor the accuracy of workers. In our setting, worker behavior on these test questions or labeled data can additionally be used for training purposes.

Also notice that in this version of our problem formulation, we assume identical worker behavior. This is a more standard setting in crowdsourcing practice as there are usually not enough work done by each worker to ascertain individual behavior. Also, it is straightforward to extend our model when different workers have different behavior when working on tasks.

Table 1: Notation description.

Notation	Description
X	Set of all the data items $\{x_i\}_{i=1}^n$
Y	Set of all true labels associated with data items in X
$\mathbf{b}_j$	A set of data item indices $\{b_{jl}\}_{l=1}^k$
$\mathbf{x}_j$	A data item batch $\{x_{jl}\}_{l=1}^k$ where x_{jl} is extracted from the data item in X with index specified by b_{jl}
$\mathbf{y}_j$	A label batch consists of true labels $\{y_{jl}\}_{l=1}^k$ associated with data items in $\mathbf{x}_j$
$\mathbf{y}'_j$	Worker annotation collected from a crowdsourcing platform for data items in $\mathbf{x}_j$
B	Set of all the batches $\{\mathbf{b}_j\}_{i=1}^m$
X_B	Set of all the data item batches
Y_B	Set of all the true labels associated with data item batches in X_B
Y'_B	Set of all the worker annotation from crowds on data item batches in X_B

3. CROWDSOURCING WORKER ANNOTATION MODEL ON BATCHES

In this section, we first describe our model for workers' annotation behavior on a batch of data items; then we introduce how to train the model based on a training data set.

Our key intuition is the follows: when a worker judges a batch of data items, she can either: 1) choose to judge data items independently as if they are presented alone; or 2) to rank all the data items according to their relative inherent scores and annotate the top several items as positive, leaving the rest in the batch as negative.

Plackett-Luce model. Before we delve into our model, we first recap a probability model for generating rankings based on scores associated with items, namely the classical Plackett-Luce model [15, 21] introduced in the 70s. Without loss of generality, suppose we are given a set of items $x_1, \ldots, x_k$. Each item x_i is associated with a certain score $s(x_i) > 0$. Here the score $s(x_i)$ models the tendency of ranking x_i higher in a randomly generated ranking and can be viewed as a measure of the inherent "goodness" of the item. A ranking of these items can be represented as a bijection $\pi : \{i\}_{i=1}^k \mapsto \{x_i\}_{i=1}^k$, that maps the i to the data item at the i-th position in the ranking. The corresponding ranking list can be represented as $\pi(1) \succ \cdots \succ \pi(k)$. In Plackett-Luce model, the

probability of generating a ranking π is:

$$P(\pi) = \prod_{i=1}^{k} \frac{s(x_i)}{\sum_{r=i}^{k} s(x_r)} \quad (1)$$

The equation above can be interpreted as the following process: Initially, we have a pool A of all the data items. Each time one picks an item x_i from a pool A of data items with a probability proportional to its score, namely:

$$P(\text{picking } x_i \text{ from } A) = \frac{s(x_i)}{\sum_{x_r \in A} s(x_r)}$$

This item is then removed from the pool A and placed at the next position in the ranking. Repeat this operation until A becomes empty. The probability of generating a ranking list according to this process is equivalent to the probability described in the Plackett-Luce model.

Worker model. We now introduce our worker model for annotating batches of data items. Again, without loss of generality, suppose we are given a batch $\mathbf{x}_j$ where $x_{jl} = x_l$, namely the given data item batch can be denoted as $\mathbf{x}_j = \{x_1, \ldots, x_k\}$. Also, recall that for each data item x_i, we denote $P(y_i = 1|x_i)$ as its inherent score η_i, which is not explicitly known.

When a worker starts to work on a certain batch of data items, they may choose to use one of two strategies:

- *Independent judging.* If the worker is making judgments based on the absolute value of η_i for each data item, we suppose the worker judges each data item $x_i \in \mathbf{x}_j$ independently by drawing the annotation $y'_i = 1$ with probability η_i and $y'_i = 0$ with probability $(1 - \eta_i)$.

- *Relative judging.* If the worker is making judgments by comparing data items within the same batch without knowing absolute value of η_i, we suppose the worker chooses to first rank all the data items in the batch based on their inherent scores, then annotates the top-τ items in the ranking as positive, leaving the other items annotated as negative. To be precise, the worker generates a ranking π for k items in the batch according to the Plackett-Luce model, with the scoring function defined as $s(x_i) = \eta_i$. Then the worker draws an integer $0 \leq \tau \leq k$ from a certain distribution, where p_τ denotes the probability of drawing the integer τ. For data items ranked as top-τ in the ranking, denoted as $x_i \in \{\pi(1), \ldots, \pi(\tau)\}$ (could be empty if $\tau = 0$), the worker annotates them as $y'_i = 1$, while other data items not within the top-τ of the ranking π are annotated as $y'_i = 0$.

To combine these two different scenarios, we suppose the worker chooses to judge independently with a certain probability $0 < \lambda < 1$, while with probability $(1 - \lambda)$ the worker makes relative judgments.

The intuition of this model is to capture two behavior pattern of workers. In the independent judging scenario, workers can remain independent in judging different data items in the same batch, with each data item being judged based on its inherent score η_i. Nevertheless, sometimes workers might judge data items within a batch by comparison. In the relative judging scenario, workers simply judge the relative relationships between data items in the same batch, which is captured by the Plackett-Luce model for generating the ranking. In order to determine the labels of data items, they have an expectation of label distribution, which is reflected by the distribution of generating τ, as it characterizes the probability of having τ positives within k data items. For instance, if workers expect there to be few positive items, then the probability of τ being low is high, while if workers expect the batches to be balanced, then the probability of τ being close to $k/2$ is high comparing to other values of τ. However, this distribution does not necessarily reflect the correct label distribution. When they try to apply their expectation of the label distribution on the batch, bias might occur.

We summarize the process of generating annotation for a batch of data items in our proposed model as below:

1. Toss a coin $Z \sim Bernoulli(\lambda)$.
   If $Z = 1$, go to Step 2; otherwise go to Step 3.
2. For each x_i, generate $y'_i \sim Bernoulli(\eta_i)$.
   Output the results and exit.
3. Generate a ranking π based on Plackett-Luce model for data items x_i in the batch.
4. Draw $\tau \sim Mult(p_\tau)$.
5. For the top-τ items in ranking π, generate $y'_i = 1$;
   otherwise generate $y'_i = 0$.
   Output the results and exit.

Model learning. The parameters that need to be determined in this worker model include: the probability of making independent judgments λ, and the distribution of the number of positive annotation when making relative judgments, represented by $p_0, \ldots, p_k$, where $0 \leq p_\tau \leq 1$ and $\sum p_\tau = 1$. We assume these two parameters are fixed for each new application of our techniques. However, for different applications, these parameters might be different — for instance, these parameters for content moderation may be different from the same parameters for spam identification or sentiment analysis.

Suppose we are given a set of n_L items X_L with their true labels Y_L, or more ideally, their inherent scores $\{\eta_i\}_{x_i \in X_L}$. If the inherent score of a data item η_i is not given, but only the binary label y_i is known, we can estimate η_i by $\eta_i = (y_i + \epsilon)/(1 + 2\epsilon)$ where ϵ is a small constant, which is set to 10^{-3} in our experiments. Then, we form them into m_L batches B_L, send them to the crowds, and obtain their annotation from workers, denoted as Y'_{B_L}.

For each batch $\mathbf{b}_j \in B_L$, we denote the set of items annotated by workers as positive as $X_j^1 = \{x_{jt} | y'_{jt} = 1\}$, and the set of items annotated as negative as $X_j^0 = \{x_{jt} | y'_{jt} = 0\}$.

We train the model by maximum likelihood estimation. The likelihood of the obtained annotation can be written as:

$$L = \prod_{j=1}^{m_L} \Bigg[\lambda \underbrace{\prod_{t=1}^{k} \eta_{jt}^{y_{jt}} (1 - \eta_{jt})^{(1-y_{jt})}}_{\text{independent judging}} + (1 - \lambda) \underbrace{p_{\tau_j} P(X_j^1 \succ X_j^0)}_{\text{relative judging}} \Bigg] \quad (2)$$

where $\tau_j = |X_j^1|$ is the number of positive annotation in batch b_j; $P(X_j^1 \succ X_j^0)$ denotes the probability of generating any rankings π that rank items in X_j^1 higher than any items in X_j^0, namely:

$$P(X_j^1 \succ X_j^0) = \sum_{\pi \in R(X_j^1, X_j^0)} P(\pi)$$

where $R(X_1, X_0) = \{\pi | \pi^{-1}(x_0) > \pi^{-1}(x_1), \forall x_1 \in X_1, x_0 \in X_0\}$; and $P(\pi)$ is defined by the Plackett-Luce model, as presented in (1). Notice that the calculation of the exact value of $P(X_j^1 \succ X_j^0)$ is hard when k is large. In our experiments, k is small enough to enumerate entire set $R(X_j^1, X_j^0)$. If k is large, we can apply Monte Carlo method to estimate the value of $P(X_j^1 \succ X_j^0)$.

Applying an EM-algorithm, where at E-step, we can have

$$\hat{\lambda}_j = \frac{\hat{\lambda}\prod_{t=1}^{k}\eta_{jt}^{y_{jt}}(1-\eta_{jt})^{(1-y_{jt})}}{\hat{\lambda}\prod_{t=1}^{k}\eta_{jt}^{y_{jt}}(1-\eta_{jt})^{(1-y_{jt})} + (1-\hat{\lambda})\hat{p}_{\tau_j}P(X_j^1 \succ X_j^0)} \tag{3}$$

And at M-step, we update the parameters $\hat{\lambda}$ and $\hat{p}_\tau$ by

$$\hat{\lambda} = \frac{1}{m_L}\sum_{j=1}^{m_L}\hat{\lambda}_j, \qquad \hat{p}_\tau = \frac{1}{\hat{Z}}\sum_{j=1}^{m_L}(1-\hat{\lambda}_j)\mathbf{1}_{\{|X_j^1|=\tau\}} \tag{4}$$

where $\hat{Z} = \sum_{j=1}^{m_L}(1-\hat{\lambda}_j)$.

4. DEBIASING ANNOTATION

In this section, we introduce our method that debiases annotations collected for batches of data given the trained worker model. More precisely, given a set of n_U unlabeled data items X_U, assembled into m_U batches represented by B_U, as well as their annotations obtained from the crowds Y'_{B_U}, how do we infer their true labels Y_U?

The basic idea is, based on the given worker model, we infer η_i for each $x_i \in X_U$. Then, we simply apply the Bayes classifier to determine the inferred label, which yields $\hat{y}_i = 1$ if $\eta_i > 0.5$, or $\hat{y}_i = 0$ if $\eta_i \leq 0.5$.

We again adopt a maximum likelihood estimation techique. The log-likelihood of the obtained annotation is:

$$\log L(\eta) = \sum_{j=1}^{m_U}\log\Big[\hat{\lambda}\prod_{x_{jt_1}\in X_j^1}\eta_{jt_1}\prod_{x_{jt_0}\in X_j^0}(1-\eta_{jt_0}) + (1-\hat{\lambda})\hat{p}_{\tau_j}P(X_j^1 \succ X_j^0)\Big] \tag{5}$$

Notice that $\hat{\lambda}$ and $\hat{p}_{\tau_j}$ are parameters learned from Section 3, and $P(X_j^1 \succ X_j^0)$ is also a function of η_i's. Similar to the previous section, we apply an EM-algorithm here by first calculating $\hat{\lambda}_j$ for each batch at the E-step according to (3) but replacing λ and p_τ by the value we learned during the training step. Then we have:

$$\log L(\eta) \geq \sum_{j=1}^{m_U}\hat{\lambda}_j\Big[\sum_{x_{jt_1}\in X_j^1}\log\eta_{jt_1} + \sum_{x_{jt_0}\in X_j^0}\log(1-\eta_{jt_0})\Big] + \sum_{j=1}^{m_U}(1-\hat{\lambda}_j)\big[\log\hat{p}_{\tau_j} + \log P(X_j^1 \succ X_j^0)\big] \tag{6}$$

where the second term includes $\log P(X_j^1 \succ X_j^0)$, which is hard to optimize. We apply the idea of the EM-algorithm again here. We use notation R_j to represent $R(X_j^1, X_j^0)$. For each $\pi \in R_j$, we can calculate its conditional probability given $X_j^1 \succ X_j^0$, denoted as $\hat{q}_\pi$ by:

$$\hat{q}_\pi = P(\pi|X_j^1 \succ X_j^0;\hat{\eta}) = \frac{P(\pi;\hat{\eta})}{\sum_{\pi\in R_j}P(\pi;\hat{\eta})} \tag{7}$$

which is the E-step. According to Jensen's inequality we have:

$$\log P(X_1 \succ X_0) = \log\sum_{\pi\in R_j}P(\pi) \geq \sum_{\pi\in R_j}\hat{q}_\pi\log P(\pi) \tag{8}$$

where the last inequality yields the objective function we want to optimize. The correctness of EM-algorithm guarantees the convergence of optimizing this function.

Furthermore, according to the minorization-maximization (MM) algorithm used in [11], we obtain the lower bound for $\log P(\pi)$, which is defined by the Plackett-Luce model, by:

$$\log P(\pi) = \sum_{t=1}^{k-1}\Big[\log\eta_{\pi(t)} - \log\sum_{s=t}^{k}\eta_{\pi(s)}\Big] \geq \sum_{t=1}^{k-1}\Big[\log\eta_{\pi(t)} - \frac{\sum_{s=t}^{k}\eta_{\pi(s)}}{\sum_{s=t}^{k}\hat{\eta}_{\pi(s)}}\Big] \tag{9}$$

where $\hat{\eta}_i$ is the estimated parameter of last iteration.

By combining (6), (8) and (9), we obtain the objective function to optimize as:

$$Q(\eta) = \sum_{j=1}^{m_U}\hat{\lambda}_j\Big[\sum_{x_{jt_1}\in X_j^1}\log\eta_{jt_1} + \sum_{x_{jt_0}\in X_j^0}\log(1-\eta_{jt_0})\Big] + \sum_{j=1}^{m_U}(1-\hat{\lambda}_j)\sum_{\pi\in R_j}\hat{q}_\pi\sum_{t=1}^{k-1}\Big[\log\eta_{\pi(t)} - \frac{\sum_{s=t}^{k}\eta_{\pi(s)}}{\sum_{s=t}^{k}\hat{\eta}_{\pi(s)}}\Big] \tag{10}$$

Notice that $Q(\eta)$ is actually a lower-bound of the original log-likelihood function (5). Moreover, for two EM-step and one MM-step we apply in deriving $Q(\eta)$, it is proven that by improving $Q(\eta)$ from this iteration $Q(\hat{\eta})$, the improvement of the log-likelihood is no less than the improvement we achieve on the $Q(\eta)$. Therefore optimizing $Q(\eta)$ can also optimize the log-likelihood.

Take the derivative, we obtain

$$\frac{\partial Q(\eta)}{\partial\eta_i} = \sum_{j\in M_1(i)}\frac{1}{\eta_i} - \sum_{j'\in M_0(i)}\frac{\hat{\lambda}_{j'}}{1-\eta_i} - \sum_{j=1}^{m_U}(1-\hat{\lambda}_j)\sum_{\pi\in R_j}\hat{q}_\pi\Big[\sum_{t=1}^{|X_j^1|}\frac{\mathbf{1}_{\{\pi^{-1}(i)\geq t\}}}{\sum_{s=t}^{k}\hat{\eta}_{\pi(s)}}\Big] \tag{11}$$

where $M_1(i)$ and $M_0(i)$ are defined as $M_y(i) = \{j : x_i \in X_j^y\}$ for $y \in \{0,1\}$. The updating rule can be obtained by solving $\partial Q(\eta)/\partial\eta_i = 0$, namely

$$\hat{\eta}_i = \frac{T_1 + T_2 + T_3 - \sqrt{(T_1+T_2+T_3)^2 - 4T_1T_3}}{2T_3} \tag{12}$$

where

$$T_1 = |M_1(i)|, \qquad T_2 = \sum_{j'\in M_0(i)}\hat{\lambda}_{j'}$$

$$T_3 = \sum_{j=1}^{m_U}(1-\hat{\lambda}_j)\sum_{\pi\in R_j}\hat{q}_\pi\Big[\sum_{t=1}^{|X_j^1|}\frac{\mathbf{1}_{\{\pi^{-1}(i)\geq t\}}}{\sum_{s=t}^{k}\hat{\eta}_{\pi(s)}}\Big]$$

By iteratively updating the scores to optimize the likelihood of the annotation on test data, we can obtain the inferred $\hat{\eta}_i$ of each item. Based on this, we can determine the inferred binary label for each data item by assigning $y'_i = 1$ if $\hat{\eta}_i > 0.5$, or $y'_i = 0$ otherwise. Notice that we do not further tune the threshold in this step, as the scores we learned here are expected to be a reasonable estimate of the true η_i's. Therefore, if the inherent scores are known, learning theory guarantees us that by using Bayes classifier (namely to take 0.5 as threshold) is supposed to achieve the best expected performance in terms of square loss.

The entire process of training model and leverarging the model to debias the obtained annotations are summarized in Algorithm 1.

Input: Data batches $B_U = \{\mathbf{b}_j\}$, crowdsourced annotation $Y'_{B_U} = \{\mathbf{y}'_j\}$; training data batches B_L, crowdsourced annotation Y'_{B_L}, true labels Y_{B_L}.
Output: Inferred labels Y_U.

```
// Training work model;
```
Initialize $\hat{\lambda}$ and $\hat{p}_\tau$ by random values;
repeat
  Update $\hat{\lambda}_j$ bfor $\forall 1 \leq j \leq m_L$ by (3);
  Update $\hat{\lambda}$ and $\hat{p}_\tau$ by (4)
until L *converged*;
```
// Calculate debiased labels;
```
repeat
  Update $\hat{\lambda}_j$ bfor $\forall 1 \leq j \leq m_U$ by (3);
  Update $\hat{\eta}_i$ for $\forall 1 \leq i \leq n_U$ by (12);
until $Q(\eta)$ *converged*;
$y_i \leftarrow \mathbf{1}_{\{\hat{\eta}_i > 0.5\}}$ for $\forall 1 \leq i \leq n_U$;
Output Y_U.

Algorithm 1: Debiasing crowdsourced annotation on batches of data items.

5. EXPERIMENTAL RESULTS

In this section, we conduct experiments on a synthetic data set and a real data set to verify the effectiveness of our proposed worker model and debiasing technique.

5.1 Experimental Data Sets

We first introduce the data sets we used in this experiments. A summary of the data sets we use in our experiments is provided in Table 2.

Synthetic data set. We construct synthetic data sets following the worker annotation model we propose in Section 3. Suppose we have n items in X, we first generate their inherent scores η_i for each $x_i \in X$ from a Beta distribution $Beta(\alpha, \beta)$, then generate the true labels Y by drawing y_i from a Bernoulli distribution parameterized by η_i for each i. In our synthetic data set, we set $\alpha = 2$ and $\beta = 4$ to simulate the case when negative data items overwhelm positive data items.

Then, we generate m batches of size k by sampling without replacement for each batch. Notice that by the phrase "without replacement" we mean there are no identical data items within the same batch, while the same item can still appear in multiple batches as we do replace the items back into the pool after a batch is generated. Thereby we obtain the set of batches B. For each $\mathbf{b}_j$ in B, we generate the workers' annotation $\mathbf{y}'_j$ from our proposed batch annotation model. The probability of making independent judgments λ is set as 0.5. The distribution of determining number of positive annotations p_τ is also assigned to be:

$$p_\tau \propto (\tau + 1)^{-\rho}$$

where ρ is positive constant, set as 2 in our experiments.

Comments data set. We utilize a real world crowdsourcing data set for annotating comments, which is used in [36]. The original crowdsourcing task was to identify inappropriate comments on LinkedIn posts published by companies or LinkedIn influencers. Inappropriate comments are defined as comments containing promotional, profane, blatant soliciting, random greeting comments, as well as comments with *only* web links and contact information. In order to collect annotation of comments, for each post, k comments are sampled and sent to CrowdFlower as a batch (unit). Workers are also provided with a codebook (i.e., a sequence of instructions) explaining how to annotate the data items. Each comment is regarded as a data item and can be annotated as positive (inappropriate comment) or negative (acceptable comment). Each batch is annotated by 5 or more workers.

Table 2: Data set statistics.

Data set	k	n_L	m_L	n_U	m_U
Synthetic	5	1,000	10,000	5,000	50,000
Comments	5	110	1,099	651	5,267

In order to provide test questions and track the performance of each worker, some of the batches are annotated by 9 trained LinkedIn employees (experts) with the same codebook and interface as used for crowd workers. The average Cohen's kappa for all expert pairs is 0.7881. For this experiments, we only adopt the batches with all of their data items annotated by both crowds and experts as we can use the experts' annotation as ground truth (aggregated by majority voting). Out of these batches, the 1,099 batches that are annotated before a worker actually starts on the job are utilized as training data set B_L. while the other 5,267 batches are utilized as the test data set B_U to infer the 651 data items the 5,267 batches covered.

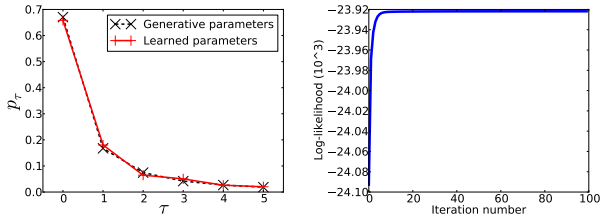
5.2 Experimental Setup

Methods evaluated. We compare the performance of our proposed method with several baselines:

- *Majority Voting (MV)*. For each data item in the test data set, simply determine its inferred label by its annotation given by the majority of workers. This aggregation strategy is often used in practice (*e.g.* [28]).
- *Majority Voting with Tuned Threshold (MVT)*. Instead of simply applying majority voting, we calculate the ratio of positive annotation on each item as a score, and tune the threshold for determining the binary inferred label. Based on a given training set of annotation and true labels, we find the threshold yielding the best F_1-score on training data set, and apply the same threshold on the test data set.
- *Plackett-Luce Model (PL)*. A strategy is to fit the Plackett-Luce model on the test data by inferring the scores $s(x_i)$ associated with each data items. We apply a Bayesian regularization on the inferred scores to confine it to be $0 \leq s(x_i) \leq 1$. We then infer a positive label to each data item with an inferred score $s(x_i) > 0.5$ and a negative label otherwise.
- * *Batch Annotation Model (BAM)*. The debiasing strategy proposed in Section 3 and 4.

Evaluation methodology. For baselines without training, we directly apply them on the test data set; for our proposed method as well as MVT, we first train the worker model on the training data set, then apply the debiasing strategy based on the trained worker model on the test data set. We compare the inferred labels to the ground-truth and evaluate the performance in terms of accuracy, precision, recall and F_1-score.

Trials and setup. For our proposed model, in training phase, we randomly initialize λ and p_τ; in debiasing phase, we initialize the all the inferred scores as 0.1. For training the worker model, we set a fixed number of iteration as 100. Our experimental results presented later show the model converges within a number of iterations much fewer than 100. For debiasing, we calculate the log-likelihood of the model and stop when the relative change of log-likelihood is within 10^{-5}.



(a) Parameter comparison of p_τ's (b) Convergence analysis

Figure 2: Learning worker model from the synthetic training data set.

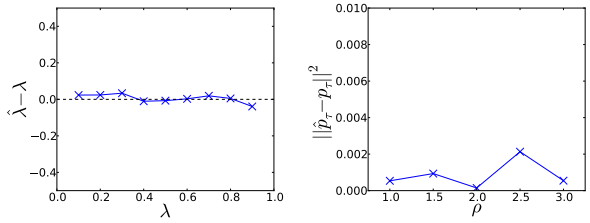
5.3 Experimental Results

Now we present the experimental results. We first verify the learning algorithm of our model on the synthetic data set, then present the learned model parameters on a real data set; we also evaluate the effectiveness of our debiasing strategy on both synthetic data set and real data set, which demonstrates an improvement in terms of F_1-score; finally we conduct a study on different configurations of experiments as a guideline for setting up a batched crowdsourcing task.

Worker model learning. We first verify the effectiveness of learning our proposed worker model. On our synthetic data set, the "true" value of probability of making independent judgments λ is set to 0.5. We learn the model from the synthetic training data and obtain the inferred $\hat{\lambda}$ as 0.4998, which reasonably recovers the original value. We also compare the original model parameters p_τ's to the inferred parameters in Figure 2(a). The black dashed line represents the original parameters used for generating synthetic annotation data, while the red solid line shows the inferred parameters of worker model, which seems as a precise fit of the original parameter. We also show the curve of log-likelihood of the training data set, which seems to converge within 20 iterations.

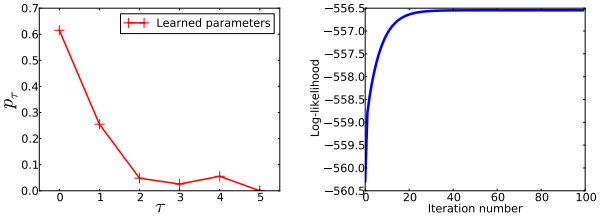
To further confirm the robustness of our learning method, we modify the configuration of synthetic data generation, and train the worker model on different data sets to check if they can recover the original parameters. We still take the same configuration of $n_L = 1,000$ and $m_L = 10,000$. The estimation error analysis is shown in Figure 3. Figure 3(a) shows the difference between the inferred parameter $\hat{\lambda}$ and the "true" parameter λ, given the annotation data generated by λ varying from 0.1 to 0.9. It can be observed that the error is reasonable small, basically within 0.1. Figure 3(b) shows the ℓ_2 norm of the difference between the estimated distribution $\hat{p}_\tau$ and the "true" distribution p_τ, when p_τ is generated with respect to different ρ varying from 1 to 3. In most of the settings, the error is below 2×10^{-3}, which is fairly low. Although we only instantiate p_τ's using a power-law distribution, as the learning method does not confine the learned distribution to be parametric, it can be directly applied to any other type of distributions.

Learned model on real data set. Given the effectiveness of our learning method verified, we apply the worker model trying to fit the data set of annotating inappropriate comments. The learned probability of a worker making independent judgments $\hat{\lambda}$ on comments data set is 0.7877. The learned distribution for determining the number of positive annotations in a batch is presented in Figure 4(a). It shows that a worker tends to annotate the entire batch as negative (*i.e.* acceptable comment) with a probability over 0.6, while picking only 1 of them as positive (*i.e.* inappropriate comment) also occurs with a relative high probability around 0.25. The workers seem to be reluctant to annotate more than 1 comments in a size-5 batch. This is coherent with most people's intuition



(a) Estimation error of λ (b) Estimation error of p_τ

Figure 3: Analysis of estimation error of parameters in the worker model under different configurations.



(a) Parameter analysis of p_τ's (b) Convergence analysis

Figure 4: Learning worker model from the comments training data set.

that inappropriate comments are rare comparing to the entire set of comments.

The convergence analysis is shown in Figure 4(b). The model converges within 50 iterations.

Performance comparison. We proceed to evaluate the performance of different aggregation strategies on both data sets. The overall performance results are shown in Table 3. In both data sets, our proposed debiasing strategy is a clear winner in terms of F_1-score, and also achieves the best accuracies.

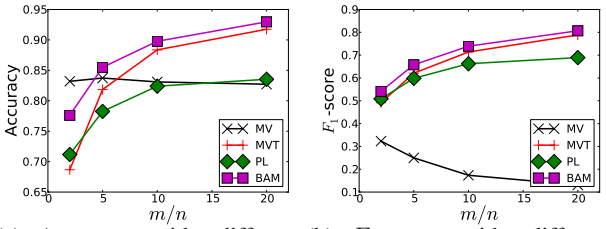
In synthetic data set, majority voting, without tuning the threshold (default set to 0.5), fails to identify most of the positive data items, and therefore achieves an extremely low recall. Only after the threshold is tuned on a training data set can it achieve a reasonable F_1-score of 71%. PL-model, in contrast, achieves a relatively low precision of 52%. Our proposed method is able to achieve the best overall performance in terms of F_1-score and accuracy, and the precision and recall achieved by our method are also relatively balanced. Notice that we do not directly apply any threshold tuning for our method and simply takes the threshold as 0.5.

In comments data set, the naïve majority voting strategy again obtains a poor recall below 80%. After tuning the threshold, its recall rises to around 85%, but still lower than our proposed method. The scores learned by PL-model yield a comparable recall to majority voting with tuned threshold, but fail to achieve a high precision. Our proposed method achieves a comparable precision of 93% and a higher recall of 87%, and therefore beat all the other baselines in terms of F_1-score (90%).

Batch number m vs. item number n. An interesting question to study is, for a certain number of items, how many (random) batches of data items does one need to label to obtain an aggregated result accurate enough. We study this question by generating synthetic data sets with different settings of number of batches m_L and m_U while number of data items n_L and n_U are fixed. In this experiments, we set $n_L = 1,000$ and $n_U = 5,000$, and generate synthetic data sets with $m_L/n_L = m_U/n_U = 2$, 5, 10, and 20. We then apply all the strategies on these data sets. To min-

Table 3: Performance comparison of Majority Voting (MV), Plackett-Luce Model (PL) and Batch Annotation Model (BAM). All results are shown as percents.

Data set	Method	Acc.	Prc.	Rcl.	F_1
Synthetic	MV	83.04	**98.91**	00.10	17.67
	MVT	88.06	64.69	80.06	71.56
	PL	82.74	52.25	**92.75**	66.85
	BAM	**89.88**	70.93	78.04	**74.31**
Comments	MV	95.55	**93.75**	79.65	86.12
	MVT	96.16	92.31	84.96	88.48
	PL	94.62	85.45	83.19	84.30
	BAM	**96.77**	93.40	**87.61**	**90.41**



(a) Accuracy with different m/n ratio (b) F_1-score with different m/n ratio

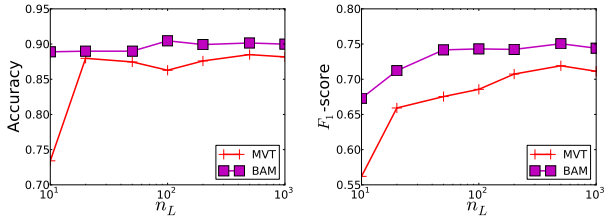
Figure 5: Performance of debiasing strategies on synthetic data sets generated by setting both m_L/n_L and m_U/n_U as 2, 5, 10, 20 respectively.

imize randomness, for each setting we repeat the data generation and application of debiasing strategies for 10 times, then report the average performance.

Results are shown in Figure 5. As we can observe, under all the different settings, the proposed method consistently outperforms other baselines, in terms of both accuracy and F_1-score. Majority voting with tuned threshold (MVT) is able to achieve comparable results to our proposed method when m/n are large enough (*e.g.* $m/n = 20$). However, when m/n is relatively small, our proposed method can achieve much better results than most of other baselines. When $m/n = 2$, it achieves an accuracy approximately 9% higher than MVT, and an F_1-score around 5% more than MVT. An exception is the naïve majority voting strategy that achieves the best accuracy when $m/n = 2$. This is due to the skewed distribution of data labels, and by simply labeling all the data items as negative can get an accuracy of approximately 80%. In comparison, the F_1-score of MV is only around 30%.

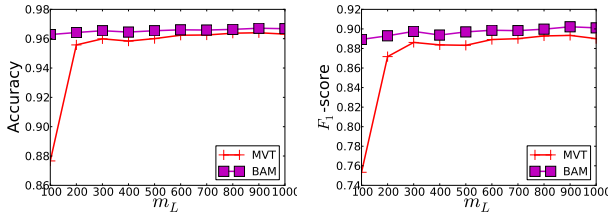
Another observation that we can make about Figure 5(b) is that the performance of majority voting drops as m/n increases. This result indicates when workers are biased and no debiasing techniques are applied, increasing the quantity of annotations collected does not help.

Size of training data set. As our method requires a small set of training data, there might be some concerns about how large a training data set is sufficient. We test the performance of two methods that rely on training data sets — MVT and our proposed method — on synthetic data sets and the comments data set. For the synthetic data set, we keep the size of test data set as $n_U = 5,000$ and $m_U = 50,000$, and vary the size of training data set by setting n_L as 10, 20, 50, 100, 200, 500, and 1,000, while setting m_L as $10n_L$. For each configuration, we generate synthetic data sets 10 times and utilize the average performance on these 10 data sets to evaluate the debiasing performance. For the comments data set,



(a) Accuracy with different sizes of training data set (b) F_1-score with different sizes of training data set

Figure 6: Performance of debiasing strategies on synthetic data sets generated by different size of training data set n_L ($m_L = 10n_L$), while the size of testing data set remains $n_U = 5,000$ and $m_U = 50,000$.



(a) Accuracy with different sizes of training data set (b) F_1-score with different sizes of training data set

Figure 7: Performance of debiasing strategies on comments data set where the training data set is randomly sampled from the original training set with different size of m_L, while the testing data set remains the same.

we randomly sample m_L batches from the training data set, where m_L is set to $100, 200, \ldots, 1000$. Again, for each configuration of training data size, we repeat the random sampling for 10 times and report the average performance.

The results of synthetic data set are shown in Figure 6. As observed, when training data set is extremely small (*e.g.* $n_L = 10$), the performance of MVT drops in terms of both accuracy and F_1-score (73% and 56% respectively). As the size of training data set increases, the performance of MVT becomes comparable to our proposed method. However, the performance of our proposed method is surprisingly stable, even when there are only 10 items and 100 batches as training data, which is as $1/500$ large as the data set used for testing. The results imply our proposed method can obtain very high performance with a small cost of labeling ground-truth data for collecting training data.

The results for the comments data set are shown in Figure 7. Again, when training data size is extremely small (*e.g.* $m_L = 100$), the performance of MVT drops substantially (89% in accuracy and 75% in F_1-score), while its performance gets more and more comparable to our method as the training data size increases. In contrast, our proposed method maintains a fairly stable performance (96% in accuracy and 90% in F_1-score) for different sizes of the training data set. This verifies again the ability of our proposed method to yield high-quality results with a sufficiently small training data set.

6. EXTENSIONS

In this section, we discuss two straightforward extensions of our proposed worker model and debiasing strategies, with respect to two useful applications other than binary classification: rating estimation and multi-class classification.

Rating estimation. In rating estimation, each data item x_i is no longer associated with a discrete label from a finite set of labels, but instead, a real value $y_i \in \mathbb{R}$. Suppose workers are asked to provide ratings on each data item in batches, and our goal is to identify the true (unknown) rating of each data item. A naïve way to aggregate their ratings on the same item is to take the average value of their ratings as the estimated rating. However, when multiple data items are grouped into batches, there can be bias similar to the one introduced in this paper.

Although we do not explicitly formalize our problem for a rating task, with some straightforward modifications, our techniques can still be applied. Without loss of generality, we can assume $0 < y_i < \infty$. If the actual rating can be negative, we can always apply a certain sigmoid function to normalize the scores to be positive values. For independent judging, we can design a well-regularized distribution with expectation of y_i for a worker to draw a rating, *e.g.* Gaussian distribution centered at y_i. For relative judging, we can still assume the worker to generate a ranking from Plackett-Luce model with parameters y_i's, and introduce distributions for generating rating of data items at different positions in the ranking, which are to be learned from the training data. For example, workers may tend to generate a rating from Gaussian distribution centered at $\mu_1 = 5.0$ for a top-ranked data item $\pi(1)$, but generate a rating from another Gaussian distribution centered at $\mu_5 = 1.0$ for a data item ranked as the fifth $\pi(5)$. If the training data is sparse and the ratings can take on any real values, it is probably preferable to employ some parametric distributions. Once the design of model is accomplished, it is straightforward to apply the same technique described in this paper to derive the debiasing strategy by maximizing the likelihood of observed annotations to estimate the underlying ratings for unrated data items.

Multi-class classification. In a multi-class classification problem, the label set $\mathcal{Y}$ may contain more than 2 possible labels. Workers are usually requested to assign data items with different labels. This is a natural extension from binary classification problem.

If the labels in $\mathcal{Y}$ have an order, for example, judging whether a review is "very helpful", "helpful" or "not helpful", the problem reduces to a rating estimation problem, where the possible value of rating are discrete values. We can simply apply the extended strategy described above.

If the labels in $\mathcal{Y}$ do not have an order, the problem can be reduces to several binary classification problems, which is straightforward to apply our strategy for debiasing workers' annotations.

7. RELATED WORK

In this section, we first introduce existing studies on annotation bias of crowds, when data items are presented either independently, or in a sequence or batches; we then introduce rank aggregation techniques and their application on crowdsourced ranking or rating.

Annotation bias in independent judgments. A number of studies have been conducted on verifying and quantifying annotation bias of crowd workers. Snow *et al.* [29] explore the performance of annotations by non-expert workers for several NLP tasks. Demeester *et al.* [8] discuss the disagreement between different users on assessment of web search results.

There are also extensive studies on modeling worker behaviors. Raykar *et al.* [23, 24, 25] study how to learn a model with noisy labeling. Specifically, they employ a logistic regression classifier, and insert hidden variables indicating whether a worker tells the truth. Karger *et al.* [12] propose an iterative algorithm to infer workers' reliability and aggregating their answers. Whitehill *et al.* [35] model the annotator ability, data item difficulty, and infer the true label from the crowds in a unified model. Most of these work also proposes various generative model to capture worker behavior. However, they assume judgments on different data items are independent, which is not necessarily true when data items are grouped into batches.

Venanzi *et al.* [33] propose a community-based label aggregation model to identify different types of workers, and correct their labels correspondingly. Das *et al.* [7] address the interactions of opinions between people connected by networks. They focus on another aspect of dependencies, which is the dependencies between workers, while in our studies, we are more concerned about dependencies between data items and their judgments.

Annotation bias in sequential and batch judgments. A few researchers also notice the correlation between judgments on different data items, but their work are mainly developed in the setting when data items are reviewed in a sequence. Scholer *et al.* [26, 27] study the annotation disagreements in a relevance assessment data set. They discover correlations between annotations of similar data items. They also explore "threshold priming" in annotation, where the annotators tend to make similar judgments or apply similar standard on consecutive data items they review. However, their work focuses on the scenario when data items are organized in a long sequence. It confines the dependencies to exist only between consecutive data items. Also, they focus more on qualitative conclusions, without a quantitative model to characterize and measure the discovered factors. Carterette *et al.* [4] provide several assessor models for the TREC data set. Mozer *et al.* [18] study the similar "relativity of judgments" phenomenon on sequential tasks instead of batches. Again, their focus is more on data items presented as a long sequence, while we focus more on data items presented in batches simultaneously.

Our recent work [36] also considers a similar setting when data items are organized in batches; we verify the existence of annotation bias caused by batching data items. Our focus in that paper was to design an active learning algorithm to smartly assemble batches, aiming to improve the performance of the classifier trained on this annotation batches. Our focus was not on improving the quality of labels collected, and we still used majority voting to obtain labels for data items. In this paper, we focus on debiasing the obtained labels directly, the higher label quality can benefit tasks including training and/or evaluating classifiers, with a broader range of applications.

Crowdsourced ranking and rating. In our model, we employ the Plackett-Luce model to capture worker behavior, and aggregate worker annotations on batches as rankings in order to infer true labels. There is a related thread of work on rank aggregation; however, to the best of our knowledge, we are the first to model crowds' annotating behavior on batches by ranking, and propose a debiasing strategy.

Studies on aggregating multiple rankings into a consistent ranking can be dated back to the seminal work of Arrow [2]. Negahban *et al.* [19] study how to aggregate pairwise comparisons into a ranking by utilizing the Bradley-Terry model [3], which is a simplified version of Plackett-Luce model utilized in this paper. Hunter *et al.* [11] propose the minorization-maximization (MM) algorithm to infer Plackett-Luce model from multiple partial orderings. Soufiani *et al.* [30] generalize Negahban *et al.*'s work and proposed a class of generalized method-of-moments (GMM) algorithm to infer parameters of Plackett-Luce model from multiple orderings, and compare the performance against MM-algorithm. They then further extend their algorithm to be applied to a more general class of ranking models called random utility models (RUMs) [31]. In ad-

dition, the technique for rank aggregation has also been studied in context of information retrieval [9, 13, 14, 22, 34]. These studies do not explicitly address the crowdsourcing settings to actually model the worker behavior. Directly applying their techniques (*e.g.* [11]) may not lead to better performance, as shown in our experiments.

There is related research on aggregating multiple rankings or leveraging crowds' power to obtain ranking of data items. Chen *et al.* [6] study aggregating crowdsourced annotation on pairwise comparison to obtain a ranking on data items. Mao *et al.* [16] show how aggregated results of noisy voting obtained from crowdsourcing platform may differ by using different aggregating strategies. However, their objective is just to obtain a ranking, while our model incorporates a ranking model but the ultimate goal is still to collect labels for data items.

Several papers also consider crowdsourced rating. Parameswaran *et al.* [20] focused on crowdsourced rating on items, and applied their system on a peer evaluation data set of a MOOC course. Crowdsourcing has also been utilized for rating multimedia content quality [5] and relevance assessment [1]. However, they do not explicitly study the scenario when data items are grouped into batches.

8. CONCLUSION

In this work we study a specific type of annotation bias in crowdsourcing, which occurs when data items are grouped into batches and submitted to workers to be judged simultaneously. We propose a novel worker model designed to capture this type of bias, and show how to train the worker model on annotation data. We also show how to debias the label obtained from crowds given a trained worker model. We conduct experiments on both synthetic data and real world data to verify the effectiveness of our methods.

The observation of batch annotation bias might exist in many scenarios other than crowdsourcing, and therefore the debiasing strategy can trigger a broad range of applications. For example, the conference paper review system where each reviewer is assigned a batch of papers can also be regarded as a batch annotation.

There are several interesting directions to extend this work. For example, one can extend the model to further incorporate the different behavior of each individual worker and adjust the debiasing strategy accordingly. Also, it would be interesting to see if it is possible to improve the efficiency of debiasing by actively assemble a batch of data items to collect the desired labels, instead of sending randomly formed batches to the crowds.

9. REFERENCES

[1] O. Alonso, D. E. Rose, and B. Stewart. Crowdsourcing for relevance evaluation. In *SIGIR Forum*, volume 42, pages 9–15. ACM, 2008.

[2] K. J. Arrow. *Social choice and individual values*. Yale university press, 1963.

[3] R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, pages 324–345, 1952.

[4] B. Carterette and I. Soboroff. The effect of assessor error on ir system evaluation. In *SIGIR*, pages 539–546. ACM, 2010.

[5] K.-T. Chen, C.-C. Wu, Y.-C. Chang, and C.-L. Lei. A crowdsourceable qoe evaluation framework for multimedia content. In *Multimedia*, pages 491–500. ACM, 2009.

[6] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *WSDM*, pages 193–202. ACM, 2013.

[7] A. Das, S. Gollapudi, R. Panigrahy, and M. Salek. Debiasing social wisdom. In *KDD*, pages 500–508. ACM, 2013.

[8] T. Demeester, R. Aly, D. Hiemstra, D. Nguyen, D. Trieschnigg, and C. Develder. Exploiting user disagreement for web search evaluation: an experimental approach. In *WSDM*, pages 33–42. ACM, 2014.

[9] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622. ACM, 2001.

[10] R. G. Gomes, P. Welinder, A. Krause, and P. Perona. Crowdclustering. In *NIPS*, pages 558–566, 2011.

[11] D. R. Hunter. Mm algorithms for generalized bradley-terry models. *Annals of Statistics*, pages 384–406, 2004.

[12] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *NIPS*, pages 1953–1961, 2011.

[13] A. Klementiev, D. Roth, and K. Small. Unsupervised rank aggregation with distance-based models. In *ICML*, pages 472–479. ACM, 2008.

[14] Y.-T. Liu, T.-Y. Liu, T. Qin, Z.-M. Ma, and H. Li. Supervised rank aggregation. In *WWW*, pages 481–490. ACM, 2007.

[15] R. D. Luce. *Individual choice behavior: A theoretical analysis*. Wiley, 1959.

[16] A. Mao, A. D. Procaccia, and Y. Chen. Better human computation through principled voting. In *AAAI*, 2013.

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

[18] M. C. Mozer, H. Pashler, M. Wilder, R. V. Lindsey, M. C. Jones, and M. N. Jones. Decontaminating human judgments by removing sequential dependencies. *NIPS*, 23, 2010.

[19] S. Negahban, S. Oh, and D. Shah. Iterative ranking from pair-wise comparisons. In *NIPS*, pages 2474–2482, 2012.

[20] A. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal crowd-powered rating and filtering algorithms. *VLDB*, 2014.

[21] R. L. Plackett. The analysis of permutations. *Applied Statistics*, pages 193–202, 1975.

[22] T. Qin, X. Geng, and T.-Y. Liu. A new probabilistic model for rank aggregation. In *NIPS*, pages 1948–1956, 2010.

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

[24] V. C. Raykar, S. Yu, L. H. Zhao, A. Jerebko, C. Florin, G. H. Valadez, L. Bogoni, and L. Moy. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *ICML*, pages 889–896. ACM, 2009.

[25] V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *JMLR*, 11:1297–1322, 2010.

[26] F. Scholer, D. Kelly, W.-C. Wu, H. S. Lee, and W. Webber. The effect of threshold priming and need for cognition on relevance calibration and assessment. In *SIGIR*, pages 623–632. ACM, 2013.

[27] F. Scholer, A. Turpin, and M. Sanderson. Quantifying test collection quality based on the consistency of relevance judgements. In *SIGIR*, pages 1063–1072. ACM, 2011.

[28] V. S. Sheng, F. Provost, and P. G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *KDD*, pages 614–622. ACM, 2008.

[29] 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 *EMNLP*, pages 254–263, 2008.

[30] H. A. Soufiani, W. Chen, D. C. Parkes, and L. Xia. Generalized method-of-moments for rank aggregation. In *NIPS*, pages 2706–2714, 2013.

[31] H. A. Soufiani, D. Parkes, and L. Xia. Computing parametric ranking models via rank-breaking. In *ICML*, pages 360–368, 2014.

[32] H. Su, J. Deng, and L. Fei-Fei. Crowdsourcing annotations for visual object detection. In *Human Computation Workshops at AAAI*, 2012.

[33] M. Venanzi, J. Guiver, G. Kazai, P. Kohli, and M. Shokouhi. Community-based bayesian aggregation models for crowdsourcing. In *WWW*, pages 155–164, 2014.

[34] M. N. Volkovs and R. S. Zemel. A flexible generative model for preference aggregation. In *WWW*, pages 479–488. ACM, 2012.

[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. Zhuang and J. Young. Leveraging in-batch annotation bias for crowdsourced active learning. In *WSDM*, pages 243–252. ACM, 2015.