Data Mining and Knowledge Discovery manuscript No.
(will be inserted by the editor)

Approximating the Crowd
SUPPLEMENTARY MATERIAL

Şeyda Ertekin · Cynthia Rudin · Haym Hirsh

the date of receipt and acceptance should be inserted later

A. Proposition 1

Proposition 1: *The score in (11) is always nonnegative.*

N_T = Total number of voters, which is an odd number so a majority always exists.
N_{voted} = number of voters that have voted already.
$N_{\text{current majority}}$ = number of voters that constitute the current majority vote. That is,

$$N_{\text{current majority}} > \frac{N_{\text{voted}}}{2} \tag{1}$$

N_{unvoted} = N_T - N_{voted} = number of labelers that have not yet voted.

To establish that the current majority vote is the true majority vote, we need

$$N_{\text{needed}} = \left\lceil \frac{N_T}{2} \right\rceil - N_{\text{current majority}}.$$

Using (1), $N_{\text{needed}} < \dfrac{N_T + 1}{2} - \dfrac{N_{\text{voted}}}{2} = \dfrac{N_{\text{unvoted}} + 1}{2}$

$$P_{\text{Bin}(N_{\text{unvoted}},0.5)}(X \geq N_{\text{needed}}) \geq P\left(X \geq \frac{N_{\text{unvoted}} + 1}{2}\right) \geq \frac{1}{2}.$$

Şeyda Ertekin · Cynthia Rudin
MIT CSAIL, Sloan School of Management, and Center for Collective Intelligence,
Massachusetts Institute of Technology, Cambridge, MA
E-mail: seyda@mit.edu, rudin@mit.edu

Haym Hirsh
Department of Computer Science and Information Science,
Cornell University, Ithaca NY, USA
E-mail: hirsh@cs.cornell.edu

To get the last inequality:

– If N_{unvoted} is odd, this probability is 1/2 with equality from the symmetry of the binomial distribution $\text{Bin}(N_{\text{unvoted}}, \frac{1}{2})$.
– If N_{unvoted} is even, this probability is greater than 1/2, since there is probability mass at $X = \dfrac{N_{\text{unvoted}} + 1}{2}$.

B. Flipping Labelers' Votes

Labelers who have a negative correlation with the majority vote can be "flipped" to consider the opposite of their votes. We will demonstrate how to do this using CrowdSense.Ind. When a labeler is flipped, observing a vote V_i with $P_i < 0.5$ is equivalent to observing $-V_i$ with probability $1 - P_i$. In the online iterations, if a labeler's quality estimate drops below 0.5, we flip the labeler's original vote and use this vote as the labeler's non-adversarial vote. For these labelers' quality estimates, their a coefficients are incremented when they *disagree* with the majority vote. In Algorithm 1, we present the pseudocode of CrowdSense.Ind with this heuristic. Figure 1 presents the tradeoff curves for CrowdSense.Ind with and without flipping the votes of the labelers with quality estimates less than 0.5, averaged over 100 runs. A remarkable difference between these two curves are observed only in experiments with ChemIR dataset. In experiments with all other datasets, both methods give similar results.

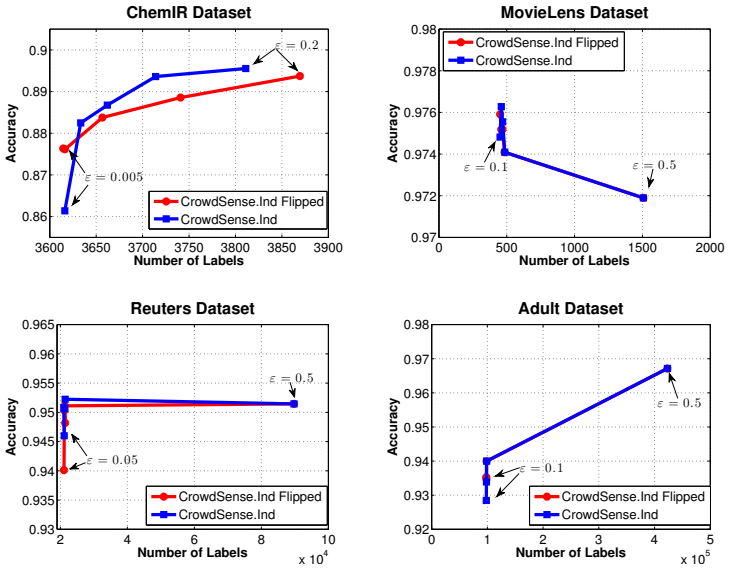

Fig. 1 Tradeoff curves for CrowdSense.Ind with and without the flipping heuristic.

Algorithm 1 Pseudocode for CrowdSense.Ind with flipping.

1. **Input:** Examples $\{x_1, x_2, \ldots, x_N\}$, Labelers $\{l_1, l_2, \ldots, l_M\}$, confidence threshold ε, smoothing parameter K.
2. **Define:** $L_Q = \{l^{(1)}, \ldots, l^{(M)}\}$, labeler id's in descending order of their quality estimates. P_+ is the prior probability of a $+1$ majority vote.
3. **Initialize:** $a_i \leftarrow 0,\ c_i \leftarrow 0$ for $i = 1, \ldots, M$.
4. **Loop for** $t = 1, ..., N$
   (a) $Q_{it} = \frac{a_i + K}{c_i + 2K}, \quad \forall i$
   (b) $\forall i$, **If** $Q_{it} < \frac{1}{2}$ **then** $Q_{it} \leftarrow 1 - Q_{it}, \quad F_i = 1$ **else** $F_i = 0$
   (c) Update L_Q with the new quality estimates.
   (d) Select 3 labelers and get their votes. $S_t = \{l^{(1)}, l^{(2)}, l^{(k)}\}$, where k is chosen uniformly at random from the set $\{3, \ldots, M\}$.
   (e) **if** $F_i = 1, V_{it} \leftarrow -V_{it}$ (flip labeler's vote)
   (f) $V_{it}^{\text{bin}} = \frac{(V_{it}+1)}{2}, \quad \forall i \in S_t$
   (g) $\psi_{it} = Q_{it}^{V_{it}^{\text{bin}}} (1 - Q_{it})^{1 - V_{it}^{\text{bin}}}, \quad \forall i \in S_t$
   (h) $\theta_{it} = (1 - Q_{it})^{V_{it}^{\text{bin}}} Q_{it}^{1 - V_{it}^{\text{bin}}}, \quad \forall i \in S_t$
   (i) **Loop for** candidate $= 3 \ldots M$, candidate $\neq k$
      i. $f(x_t|\text{votes}) = \dfrac{1}{1 + \dfrac{(\prod_{i \in S_t} \theta_{it})(1 - P_+)}{(\prod_{i \in S_t} \psi_{it}) P_+}}$
      ii. $\psi_{\text{candidate}} = 1 - Q_{\text{candidate},t}$
      iii. $\theta_{\text{candidate}} = Q_{\text{candidate},t}$
      iv. $f(x_t|\text{votes}, V_{\text{candidate},t}) = \dfrac{1}{1 + \dfrac{(\prod_{i \in S_t} \theta_{it}) \theta_{\text{candidate}} (1 - P_+)}{(\prod_{i \in S_t} \psi_{it}) \psi_{\text{candidate}} P_+}}$
      v. **If** $f(x_t|\text{votes}) \geq 0.5$ **and** $f(x_t|\text{votes}, V_{\text{candidate},t}) < 0.5 + \varepsilon$ ShouldBranch $= 1$
      vi. **If** $f(x_t|\text{votes}) < 0.5$ **and** $f(x_t|\text{votes}, V_{\text{candidate},t}) > 0.5 - \varepsilon$ ShouldBranch $= 1$
      vii. **If** ShouldBranch $= 1$ **then** $S_t = \{S_t \cup \text{candidate}\}$, get the candidate's vote **else** Don't need more labelers, break out of loop.
   (j) $\hat{y}_t = 2 \times \mathbb{1}_{[f(x_t|\text{votes}) > 0.5]} - 1$
   (k) $\forall i \in S_t$ where $V_{it} = \hat{y}_t, \quad a_i = a_i + (1 - F_i)$
   (l) $\forall i \in S_t$ where $V_{it} \neq \hat{y}_t, \quad a_i = a_i + F_i$
   (m) $c_i = c_i + 1, \quad \forall i \in S_t$
5. **End**

C. A Type of Baseline that Does Not Work Well – "Learning the Weights"

We experimented with algorithms where the weights λ_j are *learned*, or determined implicitly, using machine learning algorithms. To do this, we used the data and labels collected so far in order to fit the λ_j's and produce predictions. We describe one algorithm, which learns weights via AdaBoost, and uses CrowdSense's mechanism for adding labelers. AdaBoost constructs a classifier as a linear combination of "weak" classifiers. In the context of approximating the majority vote, each weak classifier (also known as a *feature*) corresponds to a labeler and their classifications are the observed labels from that labeler. In formal terms, let $\mathcal{F}$ denote the hypothesis space that is the class of convex combinations of features $\{h_j\}_{j=1...n}$, where $h_j : \mathcal{X} \to \{-1, 1\}$. The function

$f \in \mathcal{F}$ is then defined as a linear combination of the features:

$$f := f_{\boldsymbol{\lambda}} := \sum_j \lambda_j h_j,$$

where $\boldsymbol{\lambda} \in \mathbb{R}^n$ are the minimizers of the objective function. Consider we have already collected votes on t examples, and are selecting labelers for the next example. Following the initialization scheme of CrowdSense, we initialize with three labelers that follow the exploration/exploitation tradeoff and add new labelers on-demand based on exploitation qualities. We minimize AdaBoost's objective considering all the votes we have seen so far from the selected labelers and the next candidate labeler. Let λ_{S} denote the weights of the labelers already selected, and λ_c denote the candidate labeler's weight. We decide whether to add the candidate labeler to the set based on

$$\frac{|V_{t+1,S}\lambda_{\mathrm{S}}| - |\lambda_c|}{\left(\sum_{i\in \mathrm{S}} |\lambda_i|\right) + |\lambda_c|} < \varepsilon$$

where $V_{t+1,S}$ are the votes of the selected labelers. The normalization needs to depend on the λ's, since the scaling of λ's produced by AdaBoost can be extremely large. The criteria of adding the new labeler follows the same intuition as CrowdSense and determines whether the vote of the new labeler would bring us into the *regime of uncertainty*. Our experiments with this approach yielded poor results compared to CrowdSense.

The reason why this approach fails is because of overfitting in the early iterations. The amount of available data in early iterations is, by definition, small. This leads to problems with overfitting, which leads to poor predictions. This, in turn, leads to problems with the labels used in later iterations, which again leads to poor predictions in later iterations. The predictions become worse and worse, and the overall results were worse than many, if not all, of the baselines.

On the other hand, one could use a better method (like CrowdSense) to more accurately calculate the weights in early iterations, and then switch over to machine learning of the weights after a sufficiently large amount of data has been established.

D. Runtime Performance of CrowdSense and IEThresh

In Figure 2, we compare CrowdSense and IEThresh on MovieLens and Reuters datasets from several perspectives to compare their runtime performance and the breakdown of their labeler choices. For both datasets, we used $\varepsilon = 0.1$ for CrowdSense and $\varepsilon = 0.9$ for IEThresh, which allows IEThresh to use approximately twice the budget of CrowdSense. The average running accuracy (over 100 runs) in plots (Figures 2(a)(e)) indicate that CrowdSense achieves higher accuracy even though it‘ collects fewer number of labels (Figures 2(b)(f)) than IEThresh. Figures 2(c) and 2(g) present a bar plot of how many times each labeler was selected and Figures 2(d) and 2(h) show a bar

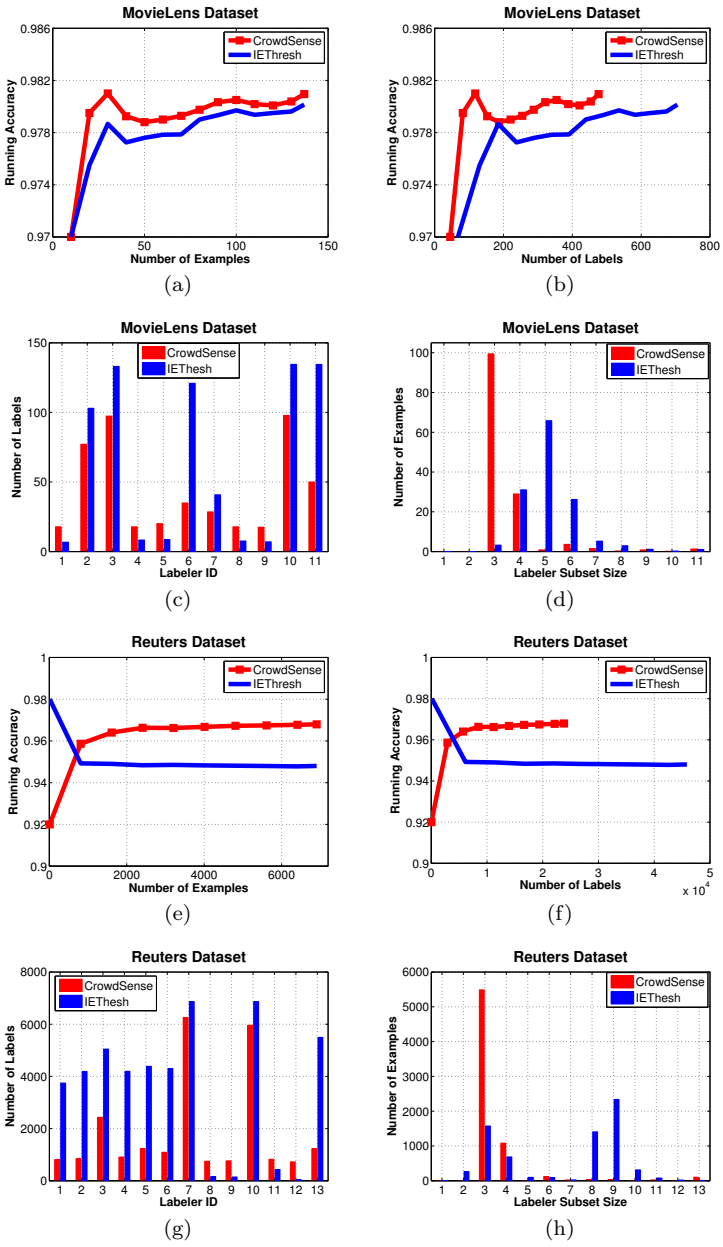

Fig. 2 Comparison of CrowdSense and IEThresh on MovieLens and Reuters datasets, averaged over 100 runs. The first two columns represent the accuracy as a function of the number of examples seen and labels collected, respectively. The third column presents the number of votes collected from each labeler, and the last column shows the breakdown of the number of labelers selected to vote on each example.

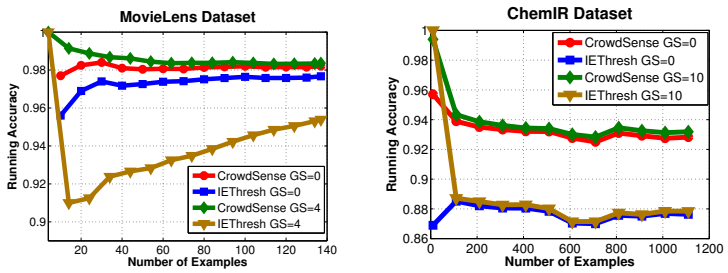

Fig. 3 Comparison of running accuracy with and without gold standard, averaged over 100 runs. For CrowdSense, we used $\varepsilon = 0.1$ and for IEThresh, we used $\varepsilon = 0.97$ to have comparable number of labelers.

plot of how many labelers were selected to vote on each round. For instance, for the MovieLens dataset, Figure 2(d) shows that CrowdSense most often chooses 3 labelers per examples whereas IEThresh rarely chooses 3 labelers per example and prefers 4,5,6, or 7 labelers per example. This also indicates that CrowdSense makes better use of the fewer number of votes it collects. Both algorithms collect labels mostly from the highest accuracy labelers (see Table 1 for labeler accuracies), but IEThresh is less effective in combing these labels. IEThresh and CrowdSense both select the highest quality labelers, where IEThresh combines them by using a simple majority vote, while CrowdSense uses a weighted majority vote; an interesting observation is that CrowdSense's weighted majority vote (of fewer labelers) is more effective for approximating the crowd's true simple majority vote than IEThresh's simple majority vote (of a larger number of labelers) – this is, in some sense, ironic.

E Initialization with Gold Standard

Gold standard examples are the "actual" opinion of the crowd gathered by taking the majority vote of all members. Even though collecting votes from the entire crowd for some of the examples initially increases the overall cost, it might help us make better estimate the true characteristics of the labelers earlier, and thus, achieve higher overall accuracy. In CrowdSense, initialization with gold standard refers to updating step 3 in the pseudocode by initializing c_{i1} with the number of gold standard examples and a_{i1} with the number of times labeler l_i agrees with the entire crowd for those gold standard examples.

In Figure 3, we present the effect of initializing with gold standard for CrowdSense and IEThresh. For the MovieLens dataset, the gold standard set contains four examples, where two of them were voted as +1 by the majority of the crowd, and the other two voted as -1. Since ChemIR is a larger dataset, we increase the gold standard set to 10 examples with an equal number of positives and negatives. The curves in Figure 3 start after observing the gold data.

As expected, gold standard data clearly improves CrowdSense's performance because the estimates are now initialized with perfect information for the first few examples. An interesting observation is that providing gold standard data to IEThresh can actually make its performance substantially worse. Consider a labeler who agrees with the crowd's vote on every gold standard example. In this case, the labeler will get a reward for every single vote, yielding $UI = 1$ (since $s(a) = 0$ and $m(a) = 1$). On the other hand, the labelers that disagree with the crowd on some examples will have standard error $s(a) > 0$ due to missing rewards, so these labelers will receive $UI > 1$ and therefore they will be preferred over the labelers that were in total agreement with the entire crowd's vote for each gold standard example. Examples of this phenomenon are shown in Figure 3, where IEThresh performs worse with gold standard than without. This illustrates why we believe that upper confidence intervals may not be the right way to handle this particular problem. In any case, we note that it may be possible to achieve better performance for IEThresh by using the upper confidence interval for a multivariate binomial distribution rather than the t-distribution, since the number of correct votes is approximately binomially distributed rather than normally distributed.