PEERANNOT: A FRAMEWORK FOR LABEL AGGREGATION IN CROWDSOURCED DATASETS

Axel Dubar[1] & Tanguy Lefort[2] & Joseph Salmon[3]

[1] *Univ. Montpellier, IMAG, France, axel.dubar@umontpellier.fr*
[2] *Univ. Montpellier, CNRS, IMAG, Inria, LIRMM, France, tanguy.lefort@umontpellier.fr*
[3] *Univ. Montpellier, CNRS, IMAG, IUF, France, joseph.salmon@umontpellier.fr*

Résumé. Ce travail présente `peerannot`, une librairie de classification de données dont les étiquettes sont générées par production participative. Elle est écrite en Python et permet d'établir une comparaison des méthodes de classification par agrégation avec d'autres librairies de référence.

Mots-clés. Statistique Computationnelle, Classification et modèles de mélange

Abstract. This work presents `peerannot`, an image data classification library of image data whose labels are generated by crowdsourcing. It is written in Python and allows a comparison of aggregation classification methods with other reference libraries.

Keywords. Computational statistics, Classification, Clustering, mixture models.

1 Introduction

Crowdsourcing is a way of building large datasets faster, cheaper, and more easily than relying on experts. It consists of leveraging multiple workers to annotate data samples, where each worker considers a subsample of the dataset. It is a methodology that has been gaining in importance for the last two decades. Nowadays, many datasets (ImageNet[1], LabelMe[2], ...) have been collected through crowdsourcing. Yet, the use of multiple and often untrained workers can lead to wrong answers and disagreement among the collected labels. This is why a cleaning step is needed to identify an estimate of the correct label (hard label case) or a probability vector of the correct label (soft label case). Crowdsourced datasets pose significant problems for which the `peerannot` [3] library proposes a framework to handle this. In this work, we focus on the following: How can we effectively aggregate multiple labels into a single label from crowdsourced tasks?

2 `peerannot`: a framework for crowdsourced datasets

We introduce `peerannot`, `https://github.com/peerannot/peerannot`, an open-source Python library developed to address crowdsourced dataset annotation problems. Its main objective

is to provide a standardized library to ensure reproducibility and accessibility. We focus here on the aggregation methods, but `peerannot` also provides identification methods that allow the user to detect ambiguous tasks and possibly prune them from the dataset before training a classifier. Deep learning strategies (such as the CoNAL [4] or CrowdLayer [5]) are also available in the toolbox and can be used to classify images directly from the crowdsourced dataset. Finally, it provides a simulation module to generate crowdsourced datasets for testing purposes.

2.1 Notation

Before presenting the aggregation methods, we define some notation. A dataset $\mathcal{D} = (x_i, y_i^\star)_{i=1}^{n_{\text{task}}}$ is composed of n_{task} tasks $x_i \in \mathcal{X}$ (the feature space) with unknown true labels $y_i^\star \in [K] = \{1, \dots, K\}$, with K the number of classes. The indicator function is denoted by $\mathbb{1}(.)$ We use the i index notation to range over the different tasks and the j index notation for the workers. As the true label of the task x_i is denoted by $y_i^\star$. The answer of the worker w_j to the task x_i is denoted by $y_i^{(j)}$ and the aggregated label from these answers is denoted by $\hat{y}_i$. We define the set of workers answering the task x_i:

$$\mathcal{A}(x_i) = \{j \in [n_{worker}] : w_j \text{ answered } x_i\} \ .$$

2.2 Aggregation methods

2.2.1 Peerannot methods

Focusing mainly on aggregation methods, we present the available strategies in `peerannot`.

- Majority vote (MV)
  This is the most basic model, it takes the label with the highest number of votes. It can be defined as:

$$\hat{y}_i^{\text{MV}} = \arg\max_{k \in [K]} \sum_{j \in \mathcal{A}(x_i)} \mathbb{1}_{\{y_i^{(j)} = k\}} \ .$$

- Naive Soft (NS)
  The Naive Soft model keeps the same intention as the Majority Vote but produces a soft label. By doing so it keeps track of the ambiguity among the workers for a task, a piece of information possibly discarded by the MV.

$$\hat{y}_i^{\text{NS}} = \left(\frac{1}{|\mathcal{A}(x_i)|} \sum_{j \in \mathcal{A}(x_i)} \mathbb{1}_{\{y_i^{(j)} = k\}} \right)_{j \in [K]} \ .$$

- Dawid and Skene (DS)
  The first two models do not take the worker's abilities into account. The Dawid and

Algorithm 1 DS (EM version)

Input: $\mathcal{D}$: crowdsourced dataset

Output: $(\hat{y}_i^{\mathrm{DS}})_{i\in[n_{\mathtt{task}}]} = (\hat{T}_{i,\cdot})_{i\in[n_{\mathtt{task}}]}$: estimated soft labels and $\{\hat{\pi}^{(j)}\}_{j\in[n_{\mathtt{worker}}]}$: estimated confusion matrices

1: **Initialization:** $\forall i \in [n_{\mathtt{task}}], \forall \ell \in [K],\ \hat{T}_{i,\ell} = \frac{1}{|\mathcal{A}(x_i)|}\sum_{j\in\mathcal{A}(x_i)} \mathbb{1}_{\{y_i^{(j)}=\ell\}}$

2: **while** Likelihood not converged **do**

3: Get $\hat{\pi}$ and $\hat{\rho}$ assuming $\hat{T}$s are known

4: $\forall (\ell,k) \in [K]^2,\ \hat{\pi}_{\ell,k}^{(j)} \leftarrow \frac{\sum_{i\in[n_{\mathtt{task}}]} \hat{T}_{i,\ell}\cdot\mathbb{1}_{\{y_i^{(j)}=k\}}}{\sum_{k'\in[K]}\sum_{i'\in[n_{\mathtt{task}}]} \hat{T}_{i',\ell}\cdot\mathbb{1}_{\{y_{i'}^{(j)}=k'\}}}$

5: $\forall \ell \in [K],\ \hat{\rho}_\ell \leftarrow \frac{1}{n_{\mathtt{task}}}\sum_{i\in[n_{\mathtt{task}}]} \hat{T}_{i,\ell}$

6: Estimate $\hat{T}$s knowing $\hat{\pi}$ and $\hat{\rho}$

7: $\forall (i,\ell), \in [n_{\mathtt{task}}]\times[K], \hat{T}_{i\ell} \leftarrow \frac{\prod_{j\in\mathcal{A}(x_i)}\prod_{k\in[K]} \hat{\rho}_\ell\cdot\left(\hat{\pi}_{\ell,k}^{(j)}\right)^{\mathbb{1}_{\{y_i^{(j)}=k\}}}}{\sum_{\ell'\in[K]}\prod_{j'\in\mathcal{A}(x_i)}\prod_{k'\in[K]} \hat{\rho}_{\ell'}\cdot\left(\hat{\pi}_{\ell'k'}^{(j')}\right)^{\mathbb{1}_{\{y_i^{(j')}=k'\}}}}$

8: **end while**

Skene's (DS) model [6], assumes each worker w_j answers independently from one another. It assigns to each worker a confusion matrix $\pi^{(j)} \in \mathbb{R}^{K\times K}$ that represents the ability of a worker to answer a task given its label denoted as $\pi_{k,\cdot}^{(j)}$. It represents the probability of the worker j answering a task labeled k correctly.
The associated likelihood can be written as:

$$\underset{\rho,\pi,T}{\arg\max} \prod_{i\in[n_{\mathtt{task}}]}\prod_{k\in[K]}\left[\rho_k \prod_{j\in[n_{\mathtt{worker}}]}\prod_{\ell\in[K]}\left(\pi_{k,\ell}^{(j)}\right)^{\mathbb{1}_{\{y_i^{(j)}=\ell\}}}\right]^{T_{i,k}}.$$

With $\rho_k = \mathbb{P}(y_i^\star = k)$ the probability of tasks labeled k to appear in the dataset and $T_{i,k} = \mathbb{1}_{\{y_i^\star=k\}}$ the vectors of label class indicators for each task the true label $y_i^\star$ is unknown. The estimation procedure associated relies on the Expectation-Maximization (EM) algorithm, as described in Algorithm 1.

- Variations of DS model (FDS and WDS)
  The DS model is one of the most studied models in the literature and variations have been proposed such as a Fast Dawid and Skene's (FDS) [7] or a Weighted Dawid and Skene's (WDS) model (described in [8]).

- Generative model of Labels, Abilities, and Difficulties (GLAD)
  Now that workers' abilities are taken into account, the GLAD model also takes the task difficulty into account in the model to recover the soft label $\hat{y}_i^{GLAD}$.
  Defining $\alpha_j \in \mathbb{R}$ as the worker ability and $\beta_i \in \mathbb{R}_*^+$ as the task's difficulty, the GLAD model can be defined as:

$$\forall k \in [K],\ \mathbb{P}(y_i^{(j)} = k | y_i^\star \neq k, \alpha_j, \beta_i) = \frac{1}{K-1}\left(1 - \frac{1}{1+\exp(-\alpha_j\beta_i)}\right) \quad .$$

Algorithm 2 GLAD (EM version)

Input: $\mathcal{D}$: crowdsourced dataset

Output:$\alpha = \{\alpha_j\}_{j\in[n_{\texttt{worker}}]}$: worker abilities, $\beta = \{\beta_i\}_{i\in[n_{\texttt{task}}]}$: task difficulties, aggregated labels

1: **while** Likelihood not converged **do**
2: Estimate probability of $y_i^\star$
3: $\forall i \in [n_{\texttt{task}}],\ \mathbb{P}(y_i^\star|\{y_i^{(j)}\}_i, \alpha, \beta_i) \propto \mathbb{P}(y_i^\star) \prod_j \mathbb{P}(y_i^{(j)}|y_i^\star, \alpha_j, \beta_i)$
4: Maximization step
5: Maximize auxiliary function $Q(\alpha, \beta)$ in Eq. 1 *w.r.t.* α and β
6: **end while**

The auxiliary function for the binary GLAD model is:

$$Q(\alpha, \beta) = \mathbb{E}[\log \mathbb{P}(\{y_i^{(j)}\}_{i,j}, \{y_i^\star\}_i)] = \sum_i \mathbb{E}[\log \mathbb{P}(y_i^\star)] + \sum_{i,j} \mathbb{E}[\log \mathbb{P}(y_i^{(j)}|y_i^\star, \alpha_j, \beta_i)] \ . \tag{1}$$

2.2.2 Other methods

Other competitors we compared with include the following:

- MMSR [9]: The Matrix Mean-Subsequence-Reduced strategy considers the reliability of all workers as a vector $s \in \mathbb{R}^{n_{\texttt{worker}}}$. Each entry s_j represents the reliability of the worker j. This strategy assumes that each worker answers independently. It also assumes that a worker is correct with probability $p_j \in [0, 1]$ and the worker's probability of being wrong is uniform across classes, *i.e.*:

$$\forall (i, j) \in [n_{\texttt{task}}] \times [n_{\texttt{worker}}],\ \begin{cases} \mathbb{P}(y_i^{(j)} = k) = p_j & \text{if } y_i^\star = k, \\ \mathbb{P}(y_i^{(j)} = k) = \frac{1-p_j}{K-1} & \text{if } y_i^\star \neq k \end{cases} .$$

  The reliability of a worker is linked to its probability of answering correctly: $s_j = \frac{K}{K-1} p_j - \frac{1}{K-1}$. This reliability can be estimated by solving a rank-one matrix completion problem defined as:

$$\mathbb{E}\left[\frac{K}{K-1} C - \frac{1}{K-1} \mathbb{1}\mathbb{1}^\top\right] = ss^\top \ ,$$

  where C is the covariance matrix of the workers' answers. More precisely, given two workers $j, j' \in [n_{\texttt{worker}}]$, the covariance between them is $C_{j,j'} = \frac{1}{N_{j,j'}} \sum_{i=1}^{n_{\texttt{task}}} \mathbb{1}(y_i^{(j)} = y_i^{(j')})$, with $N_{j,j'}$ the number of tasks in common: $N_{j,j'} = |\{i \in [n_{\texttt{task}}] | j, j' \in \mathcal{A}(x_i)\}|$. The final label is a weighted majority vote:

$$\hat{y}_i^{\text{M-MSR}} = \text{WMV}(i, W) \quad \text{with} \quad W_{j,k} = \log \frac{(K-1)p_j}{1-p_j} \ , \tag{2}$$

  where the form of the weights is derived from a maximum a posteriori formulation of the model, see [10, Corollary 9].

- WAWA [11]: This strategy, also known as the inter-rater agreement, weights each user by how much they agree with the MV labels on average. More formally, given a task i:

$$\hat{y}_i^{\text{WAWA}} = \text{WMV}\,(i, W)\,, \quad \text{with} \quad W_{j,:} = \left(\frac{1}{|\{y_{i'}^{(j)}\}_{i'}|} \sum_{i'=1}^{n_{\text{task}}} \mathbb{1}\left(y_{i'}^{(j)} = \hat{y}_{i'}^{\text{MV}}\right) \right) \mathbf{1}_K \ . \tag{3}$$

where $\forall i \in [n_{\texttt{task}}], \quad \hat{y}_i = \text{WMV}(i, W) := \arg\max_{k \in [K]} \sum_{j \in \mathcal{A}(x_i)} W_{j,k} \mathbb{1}(y_i^{(j)} = k)$, and $W \in \mathbb{R}^{n_{\texttt{worker}} \times K}$ is the matrix assigning the weight of worker j when answering class k. It allows us to instantiate a weight that can vary for each worker (but not per task) and it usually improves on the MV strategy.

3 `benchopt`: a benchmark framework for optimization

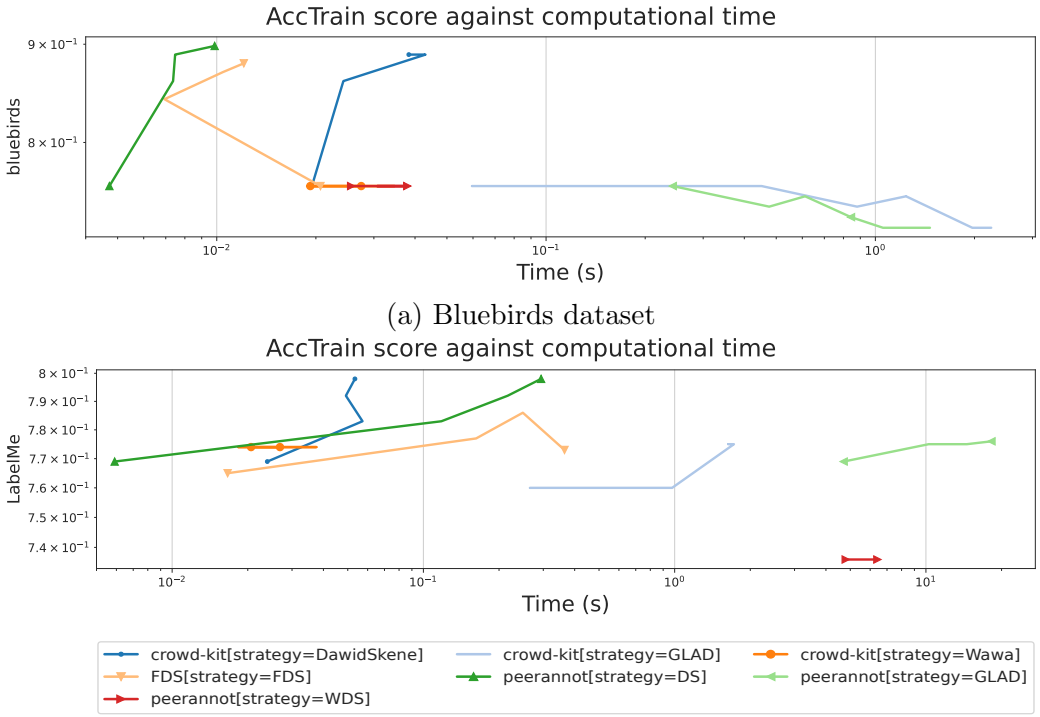
`benchopt` is an optimization benchmark library designed to facilitate comparing and reproducing optimization problems across different frameworks. It is an open-source library available at `https://github.com/benchopt/benchopt` that facilitates efficient and collaborative benchmarking by providing a standardized platform for researchers. With `benchopt`, users can easily explore, assess, and compare the performance of optimization algorithms on diverse problem sets. `benchopt` is designed for efficiency, enabling users to measure the performance of optimization algorithms by assessing the cumulated time taken to reach an optimum during the optimization steps. A list of already available implementation benchmarks such as OLS optimization or the LASSO optimization is available at `https://benchopt.github.io/available_benchmarks.html`.

4 Comparisons

To compare different strategies and different implementations across libraries, a crowdsourcing benchmark has been implemented with the `benchopt` library, this benchmark can be found at `https://github.com/benchopt/benchmark_crowdsourcing`. For each strategy, we measure the cumulated time taken to reach an optimum in the accuracy metric. Each strategy is run 5 times until convergence. We measure the accuracy with the AccTrain metric, a metric defined as:

$$\text{AccTrain}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \mathbb{1}_{\{y_i = \arg\max_{k \in [K]} (\hat{y}_i)_k\}}.$$

It computes the number of correct predictions assuming that the ground truth is known, a condition that is not always guaranteed with crowdsourced datasets. It is important to note that some strategies such as the MV and NS are computed and do not need optimization steps so their presence in the benchmark is to be viewed differently. Their representation is not shown in the figures below but if necessary they would need to be viewed as points and not dotted lines. Another library focused on crowdsourcing has been implemented named `crowd-Kit` which can be used to perform comparisons with `peerannot`.



(a) Bluebirds dataset

(b) LabelMe dataset

Figure 1: Comparison of computational time (in seconds) for all aggregation strategies

To perform the measure of the computational time and accuracy of the strategies we run a benchmark on two datasets. The first is the Bluebirds dataset. A small dataset composed of 39 workers, 108 tasks, and $K = 2$ classes. The second is the LabelMe dataset [2]. Bigger than Bluebirds with 77 workers, 1000 tasks, and $K = 8$. The benchmark can be reproduced with the following command from the crowdsourcing `benchopt` branch:

```
benchopt run ./benchmark_crowdsourcing
```

As we can see in Figure 1a, the DS implementation from `peerannot` is the first to reach its convergence followed by the FDS, then the DS from `crowd-Kit`. The fact that the DS from `peerannot` is faster than the DS from `crowd-Kit` is not guaranteed for every dataset as it is not the case in Figure 1b. `crowd-Kit` implementation is based on aggregations using vanilla pandas which can be slower. Slowness can also be explained by initialised priors which can lead to a faster convergence.
Strategies not based on DS reach lower performance as it is a binary classification problem with a low number of workers. It can be noted that using confusion matrices, a worker's answers that are consistently wrong are still informative.

5 Conclusion

We have presented a new library to address crowdsourced dataset annotation problems and an extension of `benchopt` to allow users to compare different aggregation strategies. `peerannot` and `crowd-kit` libraries can both handle classification crowdsourced datasets. We presented their aggregation methods, however, other modules allowing the identification of poorly performing workers or hardest tasks are also proposed. On these tasks, both libraries differ, with `peerannot` proposing more diverse strategies for identification.

Acknowledgments: This work was supported in part by the French National Research Agency (ANR) through the grant ANR- 20-CHIA-0001-01 (Chaire IA CaMeLOt).

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR*. 2009.

[2] F Rodrigues, F Pereira, and B Ribeiro. "Gaussian process classification and active learning with multiple annotators". In: *ICML*. PMLR. 2014, pp. 433–441.

[3] T. Lefort, B. Charlier, A. Joly, and J. Salmon. "Peerannot: classification for crowdsourced image datasets with Python". In: *Computo* (2024). URL: `https://tanglef.github.io/computo_2023/`.

[4] Z Chu, J Ma, and H Wang. "Learning from Crowds by Modeling Common Confusions." In: *AAAI*. 2021, pp. 5832–5840.

[5] F Rodrigues and F Pereira. "Deep learning from crowds". In: *AAAI*. Vol. 32. 2018.

[6] AP Dawid and AM Skene. "Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm". In: *J. R. Stat. Soc. Ser. C. Appl. Stat.* 28.1 (1979), pp. 20–28. (Visited on 10/10/2021).

[7] V B Sinha, S Rao, and V N Balasubramanian. "Fast Dawid-Skene: A fast vote aggregation scheme for sentiment classification". In: *arXiv preprint arXiv:1803.02781* (2018).

[8] T Lefort, B Charlier, A Joly, and J Salmon. "Identify ambiguous tasks combining crowdsourced labels by weighting Areas Under the Margin". In: *arXiv* (2022).

[9] Q Ma and A Olshevsky. "Adversarial crowdsourcing through robust rank-one matrix completion". In: *NeurIPS*. Vol. 33. 2020, pp. 21841–21852.

[10] Hongwei Li and Bin Yu. "Error rate bounds and iterative weighted majority voting for crowdsourcing". In: *arXiv preprint arXiv:1411.4086* (2014).

[11] Appen Limited. *Calculating Worker Agreement with Aggregate (Wawa)*. 2021. URL: `https://success.appen.com/hc/en-us/articles/202703205-Calculating-Worker-Agreement-with-Aggregate-Wawa-`.