

# Peerannot: classification for crowd-sourced image datasets with Python

Tanguy Lefort  IMAG, Univ Montpellier, CNRS, Inria, LIRMM  
Benjamin Charlier IMAG, Univ Montpellier, CNRS  
Alexis Joly  Inria, LIRMM, Univ Montpellier, CNRS  
Joseph Salmon  IMAG, Univ Montpellier, CNRS, IUF

Date published: 2023-11-23 Last modified: 2023-11-23

## Abstract

Crowdsourcing is a quick and easy way to collect labels for large datasets, involving many workers. However, workers often disagree with each other. Sources of error can arise from the workers' skills, but also from the intrinsic difficulty of the task. We present `peerannot`: a Python library for managing and learning from crowdsourced labels for classification. Our library allows users to aggregate labels from common noise models or train a deep learning-based classifier directly from crowdsourced labels. In addition, we provide an identification module to easily explore the task difficulty of datasets and worker capabilities.

*Keywords:* crowdsourcing, label noise, task difficulty, worker ability, classification

## 1 Contents

2	<b>1 Introduction: crowdsourcing in image classification</b>	2
3	<b>2 Notation and package structure</b>	3
4	2.1 Crowdsourcing notation . . . . .	3
5	2.2 Storing crowdsourced datasets in <code>peerannot</code> . . . . .	4
6	<b>3 Aggregation strategies in crowdsourcing</b>	6
7	3.1 Classical models . . . . .	7
8	3.1.1 Majority vote (MV) . . . . .	7
9	3.1.2 Naive soft (NS) . . . . .	7
10	3.1.3 Dawid and Skene (DS) . . . . .	7
11	3.1.4 Variations around the DS model . . . . .	8
12	3.1.5 Generative model of Labels, Abilities, and Difficulties (GLAD) . . . . .	9
13	3.1.6 Aggregation strategies in <code>peerannot</code> . . . . .	9
14	3.2 Experiments and evaluation of label aggregation strategies . . . . .	9
15	3.2.1 Simulated independent mistakes . . . . .	10
16	3.2.2 Simulated correlated mistakes . . . . .	14
17	3.3 More on confusion matrices in simulation settings . . . . .	16

<sup>1</sup>Corresponding author: [tanguy.lefort@umontpellier.fr](mailto:tanguy.lefort@umontpellier.fr)

18	<b>4 Learning from crowdsourced tasks</b>	<b>17</b>
19	4.1 Popular models . . . . .	17
20	4.1.1 CrowdLayer . . . . .	17
21	4.1.2 CoNAL . . . . .	18
22	4.2 Prediction error when learning from crowdsourced tasks . . . . .	18
23	4.3 Use case with peerannot on real datasets . . . . .	19
24	<b>5 Identifying tasks difficulty and worker abilities</b>	<b>21</b>
25	5.1 Exploring tasks' difficulty . . . . .	21
26	5.1.1 CIFAR-1OH dataset . . . . .	22
27	5.1.2 LabelMe dataset . . . . .	23
28	5.2 Identification of worker reliability and task difficulty . . . . .	24
29	5.2.1 CIFAR-10H . . . . .	24
30	5.2.2 LabelMe . . . . .	25
31	<b>6 Conclusion</b>	<b>28</b>
32	<b>7 Appendix</b>	<b>28</b>
33	7.1 Supplementary simulation: Simulated mistakes with discrete difficulty levels on tasks . . . . .	28
34	7.2 Comparison with other libraries . . . . .	30
35	7.3 Examples of images in CIFAR-10H and Labelme . . . . .	31

## 36 **1 Introduction: crowdsourcing in image classification**

37 Image datasets widely use crowdsourcing to collect labels, involving many workers who can annotate  
 38 images for a small cost (or even free for instance in citizen science) and faster than using expert  
 39 labeling. Many classical datasets considered in machine learning have been created with human  
 40 intervention to create labels, such as CIFAR-10, (Krizhevsky and Hinton 2009), ImageNet (Deng et  
 41 al. 2009) or Pl@ntnet (Garcin et al. 2021) in image classification, but also COCO (Lin et al. 2014),  
 42 solar photovoltaic arrays (Kasmi et al. 2023) or even macro litter (Chagneux et al. 2023) in image  
 43 segmentation and object counting.

44 Crowdsourced datasets induce at least three major challenges to which we contribute with `peerannot`:

- 45 1) **How to aggregate multiple labels into a single label from crowdsourced tasks?** This  
 occurs for example when dealing with a single dataset that has been labeled by multiple  
 workers with disagreements. This is also encountered with other scoring issues such as polls,  
 reviews, peer-grading, etc. In our framework this is treated with the `aggregate` command,  
 which given multiple labels, infers a label. From aggregated labels, a classifier can then be  
 trained using the `train` command.
- 51 2) **How to learn a classifier from crowdsourced datasets?** Where the first question is bound  
 by aggregating multiple labels into a single one, this considers the case where we do not need  
 a single label to train on, but instead train a classifier on the crowdsourced data, with the  
 motivation to perform well on a testing set. This end-to-end vision is common in machine  
 learning, however, it requires the actual tasks (the images, texts, videos, etc.) to train on – and in  
 crowdsourced datasets publicly available, they are not always available. This is treated with the  
`aggregate-deep` command that runs strategies where the aggregation has been transformed  
 into a deep learning optimization problem.
- 59 3) **How to identify good workers in the crowd and difficult tasks?** When multiple answers  
 are given to a single task, looking for who to trust for which type of task becomes necessary  
 to estimate the labels or later train a model with as few noise sources as possible. The module

62 identify uses different scoring metrics to create a worker and/or task evaluation. This is  
 63 particularly relevant considering the gamification of crowdsourcing experiments (Servajean et  
 64 al. 2016)

65 The library peerannot addresses these practical questions within a reproducible setting. Indeed,  
 66 the complexity of experiments often leads to a lack of transparency and reproducible results for  
 67 simulations and real datasets. We propose standard simulation settings with explicit implementation  
 68 parameters that can be shared. For real datasets, peerannot is compatible with standard neural net-  
 69 work architectures from the Torchvision (Marcel and Rodriguez 2010) library and Pytorch (Paszke  
 70 et al. 2019), allowing a flexible framework with easy-to-share scripts to reproduce experiments.

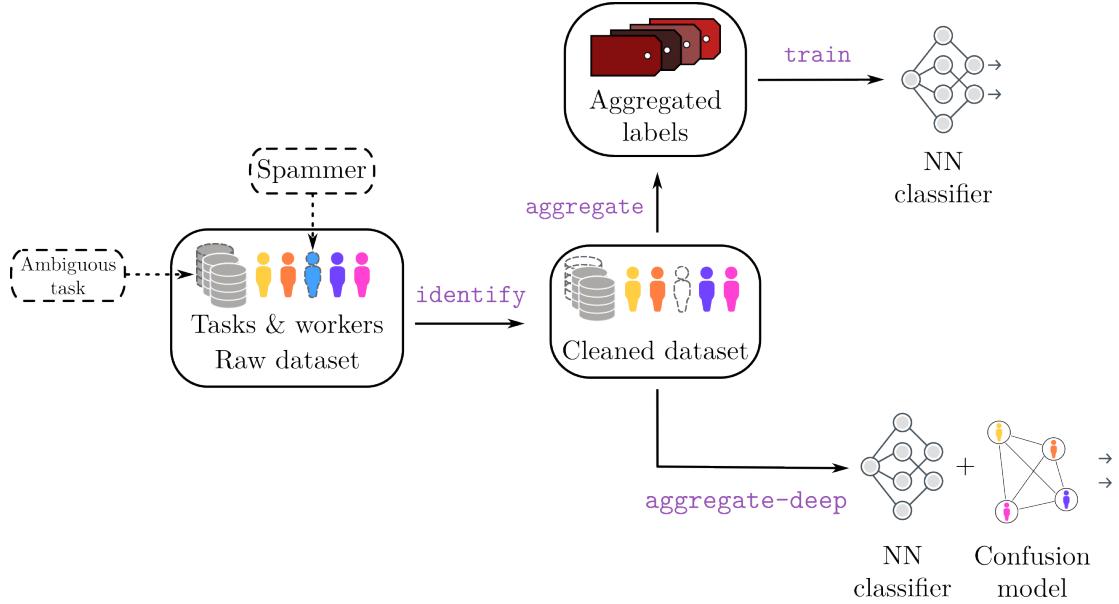


Figure 1: From crowdsourced labels to training a classifier neural network, the learning pipeline using the peerannot library. An optional preprocessing step using the identify command allows us to remove the worst-performing workers or images that can not be classified correctly (very bad quality for example). Then, from the cleaned dataset, the aggregate command may generate a single label per task from a prescribed strategy. From the aggregated labels we can train a neural network classifier with the train command. Otherwise, we can directly train a neural network classifier that takes into account the crowdsourcing setting in its architecture using aggregate-deep.

## 71 2 Notation and package structure

### 72 2.1 Crowdsourcing notation

73 Let us consider the classical supervised learning classification framework. A training set  $\mathcal{D} =$   
 74  $\{(x_i, y_i^*)\}_{i=1}^{n_{\text{task}}}$  is composed of  $n_{\text{task}}$  tasks  $x_i \in \mathcal{X}$  (the feature space) with (unknown) true label  $y_i^* \in$   
 75  $[K] = \{1, \dots, K\}$  one of the  $K$  possible classes. In the following, the tasks considered are generally  
 76 RGB images. We use the notation  $\sigma(\cdot)$  for the softmax function. In particular, given a classifier  $\mathcal{C}$   
 77 with logits outputs,  $\sigma(\mathcal{C}(x_i))_{[1]}$  represents the largest probability and we can sort the probabilities  
 78 as  $\sigma(\mathcal{C}(x_i))_{[1]} \geq \sigma(\mathcal{C}(x_i))_{[2]} \geq \dots \geq \sigma(\mathcal{C}(x_i))_{[K]}$ . The indicator function is denoted  $\mathbf{1}(\cdot)$ . We use  
 79 the  $i$  index notation to range over the different tasks and the  $j$  index notation for the workers  
 80 in the crowdsourcing experiment. Note that indices start at position 1 in the equation to follow  
 81 mathematical standard notation but it should be noted that, as this is a Python library, in the code  
 82 indices start at the 0 position.

83 With crowdsourced data the true label of a task  $x_i$ , denoted  $y_i^*$  is unknown, and there is no single  
 84 label that can be trusted as in standard supervised learning (even on the train set!). Instead, there is a  
 85 crowd of  $n_{\text{worker}}$  workers from which multiple workers  $(w_j)_j$  propose a label  $(y_i^{(j)})_j$ . These proposed  
 86 labels are used to estimate the true label. The set of workers answering the task  $x_i$  is denoted by

$$\mathcal{A}(x_i) = \{j \in [n_{\text{worker}}] : w_j \text{ answered } x_i\}. \quad (1)$$

87 The cardinal  $|\mathcal{A}(x_i)|$  is called the feedback effort on the task  $x_i$ . Note that the feedback effort can not  
 88 exceed the total number of workers  $n_{\text{worker}}$ . Similarly, one can adopt a worker point of view: the set  
 89 of tasks answered by a worker  $w_j$  is denoted

$$\mathcal{T}(w_j) = \{i \in [n_{\text{task}}] : w_j \text{ answered } x_i\}. \quad (2)$$

90 The cardinal  $|\mathcal{T}(w_j)|$  is called the workload of  $w_j$ . The final dataset can then be decomposed as:

$$\mathcal{D}_{\text{train}} := \bigcup_{i \in [n_{\text{task}}]} \{(x_i, (y_i^{(j)})) \text{ for } j \in \mathcal{A}(x_i)\} = \bigcup_{j \in [n_{\text{worker}}]} \{(x_i, (y_i^{(j)})) \text{ for } i \in \mathcal{T}(w_j)\}.$$

91 In this article, we do not address the setting where workers report their self-confidence (Yasmin et al.  
 92 2022), nor settings where workers are presented a trapping set – *i.e.*, a subset of tasks where the true  
 93 label is known to evaluate them with known labels (Khattak 2017).

## 94 2.2 Storing crowdsourced datasets in peerannot

95 Crowdsourced datasets come in various forms. To store [crowdsourcing datasets](#) efficiently and in a  
 96 standardized way, [peerannot](#) proposes the following structure, where each dataset corresponds to a  
 97 folder. Let us set up a toy dataset example to understand the data structure and how to store it.

---

**Listing 1** Dataset storage tree structure.

---

```
datasetname
    train
        ...
        images
        ...
    val
    test
    metadata.json
    answers.json
```

---

98 The `answers.json` file stores the different votes for each task as described in Figure 2. This `.json`  
 99 is the rosetta stone between the task ids and the images. It contains the tasks' id, the workers's  
 100 id and the proposed label for each given vote. Furthermore, storing labels in a dictionary is more  
 101 memory-friendly than having an array of size  $(n_{\text{task}}, n_{\text{worker}})$  and writing  $y_i^{(j)} = -1$  when the  
 102 worker  $w_j$  did not see the task  $x_i$  and  $y_i^{(j)} \in [K]$  otherwise.

103 In Figure 2, there are three tasks,  $n_{\text{worker}} = 4$  workers and  $K = 2$  classes. Any available task should  
 104 be stored in a single file whose name follows the convention described in Listing 1. These files are  
 105 spread into a `train`, `val` and `test` subdirectories as in [ImageFolder](#) datasets from [torchvision](#)

106 Finally, a `metadata.json` file includes relevant information related to the crowdsourcing experiment  
 107 such as the number of workers, the number of tasks, *etc*. For example, a minimal `metadata.json` file  
 108 for the toy dataset presented in Figure 2 is:

The diagram illustrates the mapping between the data stored in a peerannot file and the data collected from workers.

**Left (peerannot storage):**

	$\{ \textcolor{red}{\text{---}} : 1, \textcolor{teal}{\text{---}} : 0, \textcolor{purple}{\text{---}} : 1, \textcolor{yellow}{\text{---}} : 1 \}$
	$\{ \textcolor{red}{\text{---}} : 1, \textcolor{purple}{\text{---}} : 0, \textcolor{yellow}{\text{---}} : 0 \}$
	$\{ \textcolor{teal}{\text{---}} : 1, \textcolor{yellow}{\text{---}} : 1 \}$

**Right (Data Collected):**

$K = 2$ • 0: not smiling • 1: smiling				
	1	0	1	1
	1	X	0	0
	X	1	X	1

Figure 2: Data storage for the toy-data crowdsourced dataset, a binary classification problem ( $K = 2$ , smiling/not smiling) on recognizing smiling faces. (left: how data is stored in peerannot in a file `answers.json`, right: data collected)

```
{
    "name": "toy-data",
    "n_classes": 2,
    "n_workers": 4,
    "n_tasks": 3
}
```

109 The toy-data example dataset is available as an example [in the peerannot repository](#). Classical  
 110 datasets in crowdsourcing such as CIFAR-10H (Peterson et al. 2019) and LabelMe (Rodrigues, Pereira,  
 111 and Ribeiro 2014) can be installed directly using peerannot. To install them, run the `install`  
 112 command from peerannot:

```
! peerannot install ./datasets/labelme/labelme.py
! peerannot install ./datasets/cifar10H/cifar10h.py
```

113 For both CIFAR-10H and LabelMe, the dataset was originally released for standard supervised  
 114 learning (classification). Both datasets have been reannotated by a crowd of workers. These labels are  
 115 used as true labels in evaluations and visualizations. Examples of CIFAR-10H images are available in  
 116 Figure 16, and LabelMe examples in Figure 17 in Appendix. Crowdsourcing votes, however, bring  
 117 information about possible confusions (see Figure 3 for an example with CIFAR-10H and Figure 4  
 118 with LabelMe).

```
import torch
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from pathlib import Path
import json
import matplotlib.ticker as mtick
import pandas as pd
sns.set_style("whitegrid")
import utils as utx
utx.figure_5()
```

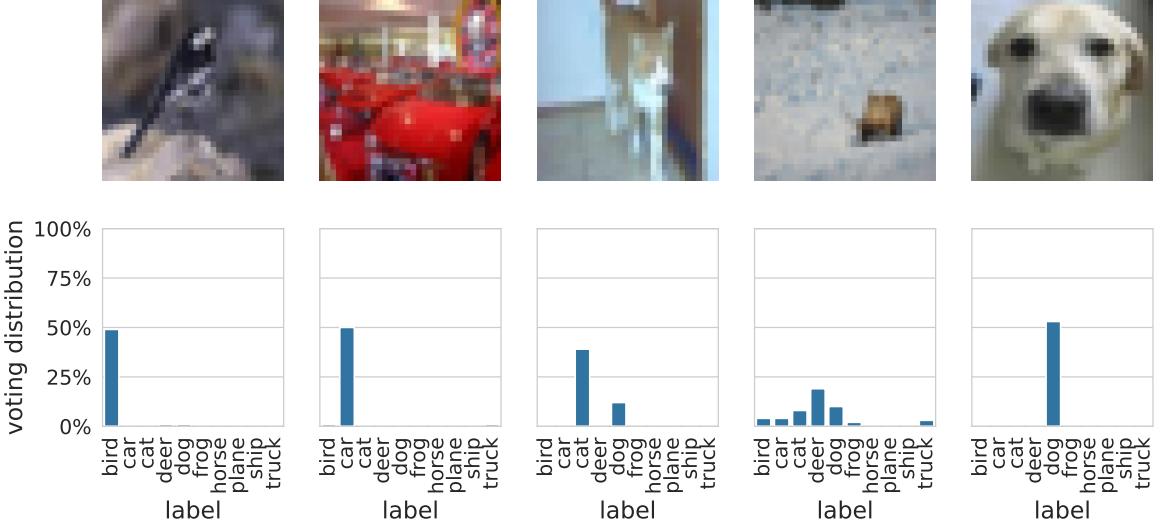


Figure 3: Example of crowdsourced images from CIFAR-10H. Each task has been labeled by multiple workers. We display the associated voting distribution over the possible classes.

```
utx.figure_5_labelmeversion()
```

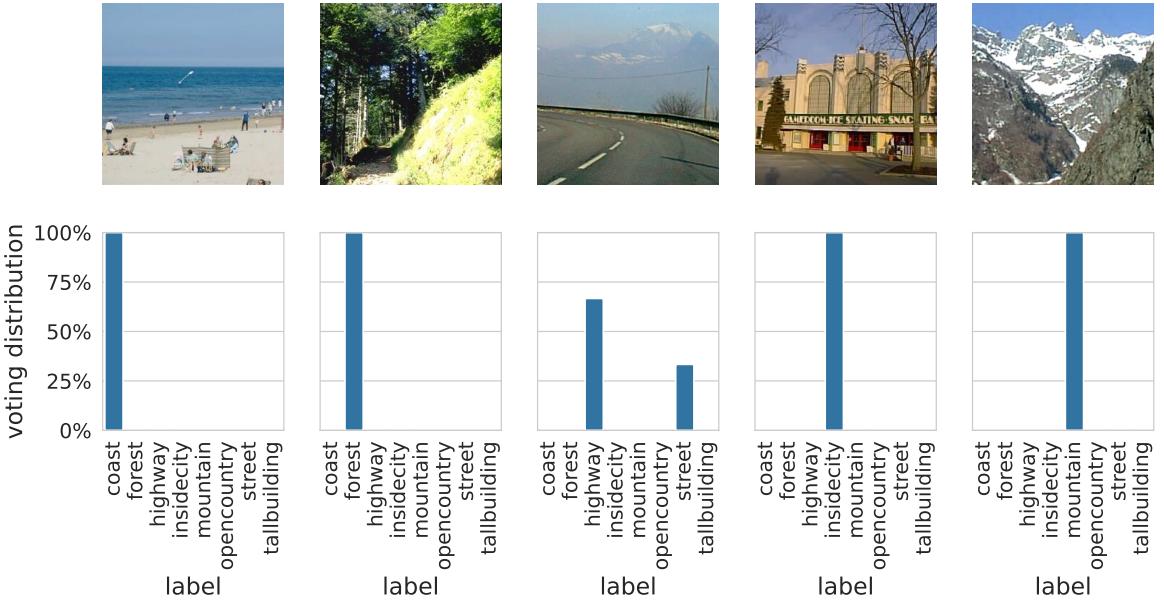


Figure 4: Example of crowdsourced images from LabelMe. Each task has been labeled by multiple workers. We display the associated voting distribution over the possible classes.

### 119 3 Aggregation strategies in crowdsourcing

120 The first question we address with peerannot is: *How to aggregate multiple labels into a single label  
121 from crowdsourced tasks?* The aggregation step can lead to two types of learnable labels  $\hat{y}_i \in \Delta_K$   
122 (where  $\Delta_K$  is the simplex of dimension  $K - 1$ :  $\Delta_K = \{p \in \mathbb{R}^K : \sum_{k=1}^K p_k = 1, p_k \geq 0\}$ ) depending on  
123 the use case for each task  $x_i, i = 1, \dots, n_{\text{task}}$ :

- 124 • a **hard** label:  $\hat{y}_i$  is a Dirac distribution, this can be encoded as a classical label in  $[K]$ ,

- 125 • a **soft** label:  $\hat{y}_i \in \Delta_K$  can represent any probability distribution on  $[K]$ . In that case, each  
 126 coordinate of the  $K$ -dimensional vector  $\hat{y}_i$  represents the probability of belonging to the given  
 127 class.

128 Learning from soft labels has been shown to improve learning performance and make the classifier  
 129 learn the task ambiguity (Zhang et al. 2018; Peterson et al. 2019; Park and Caragea 2022). However,  
 130 crowdsourcing is often used as a stepping stone to create a new dataset. We usually expect a  
 131 classification dataset to associate a task  $x_i$  to a single label and not a full probability distribution. In  
 132 this case, we recommend releasing the anonymous answered labels and the aggregation strategy used  
 133 to reach a consensus on a single label. With `peerannot`, both soft and hard labels can be produced.

134 Note that when a strategy produces a soft label, a hard label can be easily induced by taking the  
 135 mode, *i.e.*, the class achieving the maximum probability.

### 136 3.1 Classical models

137 We list below the most classical aggregation strategies used in crowdsourcing.

#### 138 3.1.1 Majority vote (MV)

139 The most intuitive way to create a label from multiple answers for any type of crowdsourced task is  
 140 to take the **majority vote** (MV). Yet, this strategy has many shortcomings (James 1998) – there is no  
 141 noise model, no worker reliability estimated, no task difficulty involved and especially no way to  
 142 remove poorly performing workers. This standard choice can be expressed as:

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

#### 143 3.1.2 Naive soft (NS)

144 One pitfall with MV is that the label produced is hard, hence the ambiguity is discarded by construction.  
 145 A simple remedy consists in using the **Naive Soft** (NS) labeling, *i.e.*, output the empirical distribution  
 146 as the task label:

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

147 With the NS label, we keep the ambiguity, but all workers and all tasks are put on the same level. In  
 148 practice, it is known that each worker comes with their abilities, thus modeling this knowledge can  
 149 produce better results.

#### 150 3.1.3 Dawid and Skene (DS)

151 Refining the aggregation, researchers have proposed a noise model to take into account the workers'  
 152 abilities. The **Dawid and Skene**'s (DS) model (Dawid and Skene 1979) is one of the most studied  
 153 (Gao and Zhou 2013) and applied (Servajean et al. 2017; Rodrigues and Pereira 2018). These types of  
 154 models are most often optimized using EM-based procedures. Assuming the workers are answering  
 155 tasks independently, this model boils down to model pairwise confusions between each possible  
 156 class. Each worker  $w_j$  is assigned a confusion matrix  $\pi^{(j)} \in \mathbb{R}^{K \times K}$  as described in Section 3. The  
 157 model assumes that for a task  $x_i$ , conditionally on the true label  $y_i^* = k$  the label distribution of the  
 158 worker's answer follows a multinomial distribution with probabilities  $\pi_{k,j}^{(j)}$  for each worker. Each  
 159 class has a prevalence  $\rho_k = \mathbb{P}(y_i^* = k)$  to appear in the dataset. Using the independence between

160 workers, we obtain the following likelihood to maximize, with latent variables  $\rho$ ,  $\pi = \{\pi^{(j)}\}_j$  and  
 161 unobserved variables  $(y_i^{(j)})_{i,j}$ :

$$\arg \max_{\rho, \pi} \prod_{i \in [n_{\text{task}}]} \prod_{k \in [K]} \left[ \rho_k \prod_{j \in [n_{\text{worker}}]} \prod_{\ell \in [K]} (\pi_{k,\ell}^{(j)})^{\mathbf{1}_{\{y_i^{(j)}=\ell\}}} \right].$$

162 When the true labels are not available, the data comes from a mixture of categorical distributions.  
 163 To retrieve ground truth labels and be able to estimate these parameters, Dawid and Skene (1979)  
 164 have proposed to consider the true labels as additional unknown parameters. In this case, denoting  
 165  $T_{i,k} = \mathbf{1}_{\{y_i^* = k\}}$  the vectors of label class indicators for each task, the likelihood with known true labels  
 166 is:

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

167 This framework allows to estimate  $\rho, \pi, T$  with an EM algorithm as follows:

- 168 • With the MV strategy, get an initial estimate of the true labels  $T$ .  
 169 • Estimate  $\rho$  and  $\pi$  knowing  $T$  using maximum likelihood estimators.  
 170 • Update  $T$  knowing  $\rho$  and  $\pi$  using Bayes formula.  
 171 • Repeat until convergence of the likelihood.

172 The final aggregated soft labels are  $\hat{y}_i^{\text{DS}} = T_{i,.}$  Note that DS also provides the estimated confusion  
 173 matrices  $\hat{\pi}^{(j)}$  for each worker  $w_j$ .

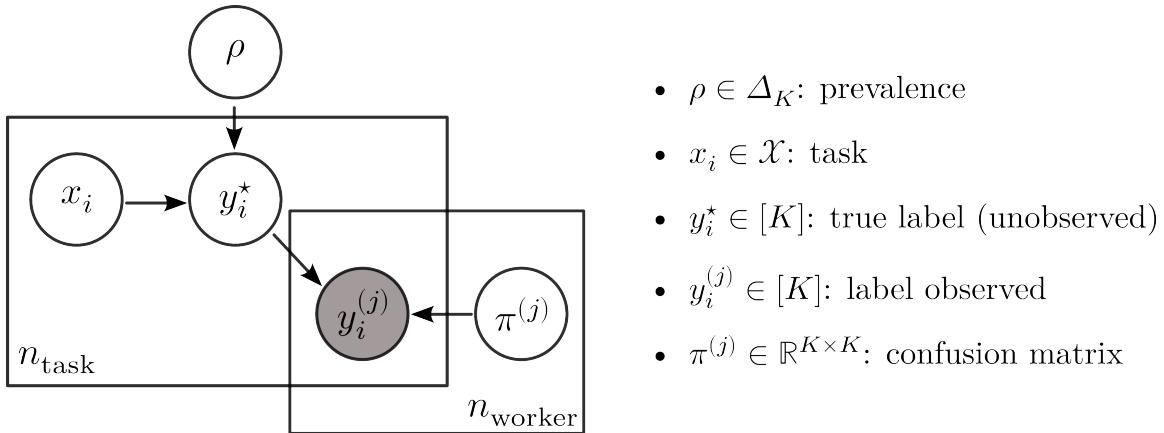


Figure 5: Bayesian plate notation for the DS model

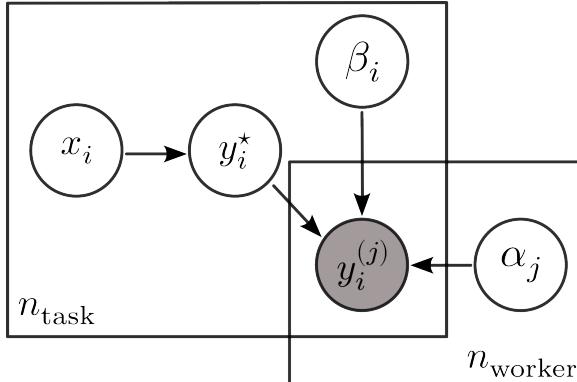
### 174 3.1.4 Variations around the DS model

175 Many variants of the DS model have been proposed in the literature, using Dirichlet priors on the  
 176 confusion matrices (Passonneau and Carpenter 2014), using  $1 \leq L \leq n_{\text{worker}}$  clusters of workers  
 177 (Imamura, Sato, and Sugiyama 2018) (DSWC) or even faster implementation that produces only hard  
 178 labels (Sinha, Rao, and Balasubramanian 2018).

179 In particular, the DSWC strategy (Dawid and Skene with Worker Clustering) highly reduces the  
 180 dimension of the parameters in the DS model. In the original model, there are  $K^2 \times n_{\text{worker}}$  parameters  
 181 to be estimated for the confusion matrices only. The DSWC model reduces them to  $K^2 \times L + L$   
 182 parameters. Indeed, there are  $L$  confusion matrices  $\Lambda = \{\Lambda_1, \dots, \Lambda_L\}$  and the confusion matrix of a  
 183 cluster is assumed drawn from a multinomial distribution with weights  $(\tau_1, \dots, \tau_L) \in \Delta_L$  over  $\Lambda$ , such  
 184 that  $\mathbb{P}(\pi^{(j)} = \Lambda_\ell) = \tau_\ell$ .

185 **3.1.5 Generative model of Labels, Abilities, and Difficulties (GLAD)**

186 Finally, we present the **GLAD** model (Whitehill et al. 2009) that not only takes into account the  
 187 worker’s ability, but also the task difficulty in the noise model. The likelihood is optimized using an  
 188 EM algorithm to recover the soft label  $\hat{y}_i^{\text{GLAD}}$ .



- $x_i \in \mathcal{X}$ : task
- $y_i^* \in [K]$ : true label (unobserved)
- $y_i^{(j)} \in [K]$ : label observed
- $\alpha_j \in \mathbb{R}$ : worker reliability
- $\beta_i \in \mathbb{R}_*^+$ : task difficulty

Figure 6: Bayesian [plate notation](#) for the GLAD model

189 Denoting  $\alpha_j \in \mathbb{R}$  the worker ability (the higher the better) and  $\beta_i \in \mathbb{R}_*^+$  the task’s difficulty (the higher  
 190 the easier), the model noise is:

$$\mathbb{P}(y_i^{(j)} = y_i^* | \alpha_j, \beta_i) = \frac{1}{1 + \exp(-\alpha_j \beta_i)} .$$

191 GLAD’s model also assumes that the errors are uniform across wrong labels, thus:

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

192 This results in estimating  $n_{\text{worker}} + n_{\text{task}}$  parameters.

193 **3.1.6 Aggregation strategies in peerannot**

194 All of these aggregation strategies – and more – are available in the `peerannot` library from [the](#)  
 195 [peerannot.models module](#). Each model is a class object in its own Python file. It inherits from the  
 196 `CrowdModel` template class and is defined with at least two methods:

- `run`: includes the optimization procedure to obtain needed weights (e.g., the EM algorithm for the DS model),
- `get_probs`: returns the soft labels output for each task.

200 **3.2 Experiments and evaluation of label aggregation strategies**

201 One way to evaluate the label aggregation strategies is to measure their accuracy. This means that  
 202 the underlying ground truth must be known – at least for a representative subset. This is the case in  
 203 simulation settings where the ground truth is available. As the set of  $n_{\text{task}}$  can be seen as a training  
 204 set for a future classifier, we denote this metric `AccTrain` on a dataset  $\mathcal{D}$  for some given aggregated  
 205 label  $(\hat{y}_i)_i$  as:

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

206 In the following, we write AccTrain for  $\text{AccTrain}(\mathcal{D}_{\text{train}})$  as we only consider the full training set  
 207 so there is no ambiguity. The AccTrain computes the number of correctly predicted labels by the  
 208 aggregation strategy knowing a ground truth. While this metric is useful, in practice there are a few  
 209 arguable issues:

- 210 • the AccTrain metric does not consider the ambiguity of the soft label, only the most probable  
 211 class, whereas in some contexts ambiguity can be informative,  
 212 • in supervised learning one objective is to identify difficult or mislabeled tasks (Pleiss et al.  
 213 2020; Lefort et al. 2022), pruning those tasks can easily artificially improve the AccTrain, but  
 214 there is no guarantee over the predictive performance of a model based on the newly pruned  
 215 dataset,  
 216 • in practice, true labels are unknown, thus this metric would not be computable.

217 We first consider classical simulation settings in the literature that can easily be created and repro-  
 218 duced using peerannot. For each dataset, we present the distribution of the number of workers per  
 219 task ( $|\mathcal{A}(x_i)|$ ) <sub>$i=1, \dots, n_{\text{task}}$</sub>  Equation 1 on the right and the distribution of the number of tasks per worker  
 220 ( $|\mathcal{T}(w_j)|$ ) <sub>$j=1, \dots, n_{\text{worker}}$</sub>  Equation 2 on the left.

### 221 3.2.1 Simulated independent mistakes

222 The independent mistakes setting considers that each worker  $w_j$  answers follows a multinomial  
 223 distribution with weights given at the row  $y_i^*$  of their confusion matrix  $\pi^{(j)} \in \mathbb{R}^{K \times K}$ . Each confusion  
 224 row in the confusion matrix is generated uniformly in the simplex. Then, we make the matrix  
 225 diagonally dominant (to represent non-adversarial workers) by switching the diagonal term with  
 226 the maximum value by row. Answers are independent of one another as each matrix is generated  
 227 independently and each worker answers independently of other workers. In this setting, the DS  
 228 model is expected to perform better with enough data as we are simulating data from its assumed  
 229 noise model.

230 We simulate  $n_{\text{task}} = 200$  tasks and  $n_{\text{worker}} = 30$  workers with  $K = 5$  possible classes. Each task  $x_i$   
 231 receives  $|\mathcal{A}(x_i)| = 10$  labels. With 200 tasks and 30 workers, asking for 10 leads to around  $\frac{200 \times 10}{30} \approx 67$   
 232 tasks per worker (with variations due to randomness in the affectations).

```
! peerannot simulate --n-worker=30 --n-task=200 --n-classes=5 \
    --strategy independent-confusion \
    --feedback=10 --seed 0 \
    --folder ./simus/independent

from peerannot.helpers.helpers_visu import feedback_effort, working_load
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
from pathlib import Path

votes_path = Path.cwd() / "simus" / "independent" / "answers.json"
metadata_path = Path.cwd() / "simus" / "independent" / "metadata.json"
efforts = feedback_effort(votes_path)
workload = working_load(votes_path, metadata_path)
feedback = feedback_effort(votes_path)
utx.figure_simulations(workload, feedback)
plt.show()
```

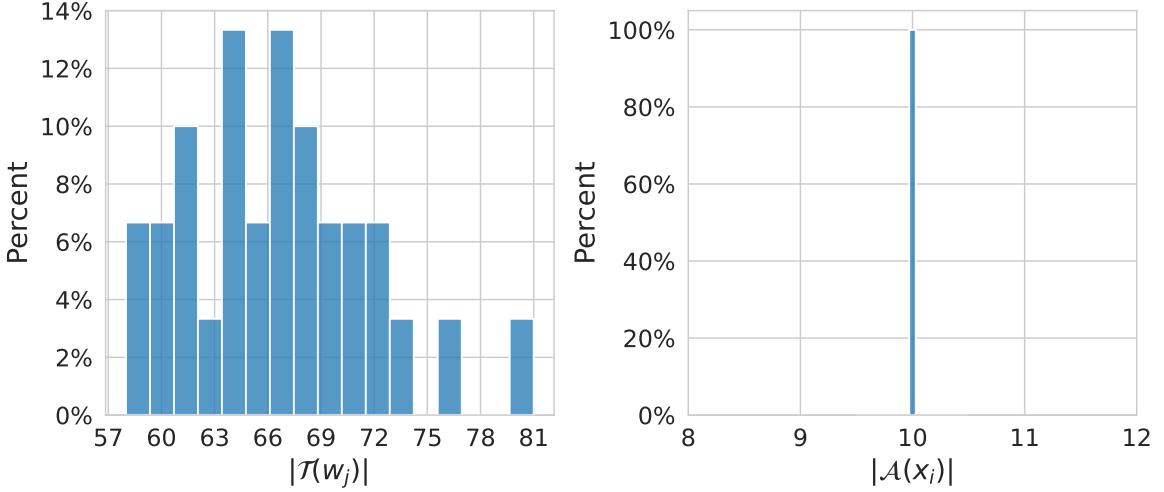


Figure 7: Distribution of number of tasks given per worker (left) and number of labels per task (right) in the independent mistakes setting.

233 With the obtained answers, we can look at the aforementioned aggregation strategies performance:

```

for strat in ["MV", "NaiveSoft", "DS", "GLAD", "DSWC[L=5]", "DSWC[L=10]"]:
    ! peerannot aggregate ./simus/independent/ -s {strat}

import pandas as pd
import numpy as np
from IPython.display import display
simu_indep = Path.cwd() / 'simus' / 'independent'
results = {
    "mv": [], "naivesoft": [], "glad": [],
    "ds": [], "dswc[l=5)": [], "dswc[l=10)": []
}
for strategy in results.keys():
    path_labels = simu_indep / "labels" / f"labels_independent-confusion_{strategy}.npy"
    ground_truth = np.load(simu_indep / "ground_truth.npy")
    labels = np.load(path_labels)
    acc = (
        np.mean(labels == ground_truth)
        if labels.ndim == 1
        else np.mean(
            np.argmax(labels, axis=1)
            == ground_truth
        )
    )
    results[strategy].append(acc)
results["NS"] = results["naivesoft"]
results.pop("naivesoft")
results = pd.DataFrame(results, index=['AccTrain'])
results.columns = map(str.upper, results.columns)
results = results.style.set_table_styles([
    dict(selector='th', props=[('text-align', 'center')])])
)

```

```

results.set_properties(**{'text-align': 'center'})
results = results.format(precision=3)
display(results)

```

Table 1: AccTrain metric on simulated independent mistakes considering classical feature-blind label aggregation strategies

Table 1

	MV	GLAD	DS	DSWC[L=5]	DSWC[L=10]	NS
AccTrain	0.755	0.775	0.890	0.775	0.770	0.760

<sup>234</sup> As expected by the simulation framework, Table 1 fits the DS model, thus leading to better accuracy  
<sup>235</sup> in retrieving the simulated labels for the DS strategy. The MV and NS aggregations do not consider  
<sup>236</sup> any worker-ability scoring or the task’s difficulty and perform the worst.

<sup>237</sup> **Remark:** peerannot can also simulate datasets with an imbalanced number of votes chosen uniformly  
<sup>238</sup> at random between 1 and the number of workers available. For example:

```

! peerannot simulate --n-worker=30 --n-task=200 --n-classes=5 \
    --strategy independent-confusion \
    --imbalance-votes \
    --seed 0 \
    --folder ./simus/independent-imbalanced/

sns.set_style("whitegrid")

votes_path = Path.cwd() / "simus" / "independent-imbalanced" / "answers.json"
metadata_path = Path.cwd() / "simus" / "independent-imbalanced" / "metadata.json"
efforts = feedback_effort(votes_path)
workload = working_load(votes_path, metadata_path)
feedback = feedback_effort(votes_path)
utx.figure_simulations(workload, feedback)
plt.show()

```

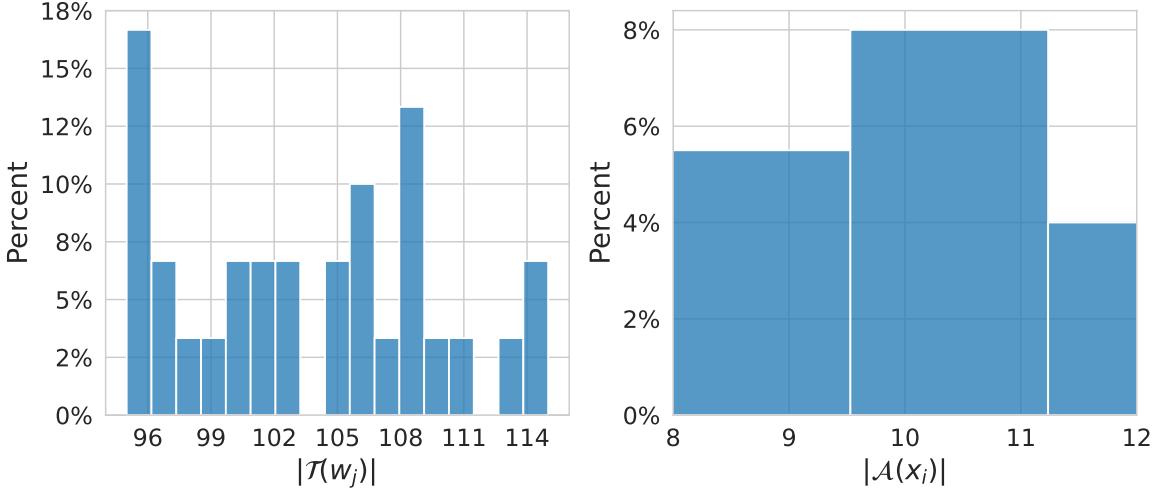


Figure 8: Distribution of the number of tasks given per worker (left) and of the number of labels per task (right) in the independent mistakes setting with voting imbalance enabled.

<sup>239</sup> With the obtained answers, we can look at the aforementioned aggregation strategies performance:

```

for strat in ["MV", "NaiveSoft", "DS", "GLAD", "DSWC[L=5]", "DSWC[L=10]"]:
    ! peerannot aggregate ./simus/independent-imbalanced/ -s {strat}

import pandas as pd
import numpy as np
from IPython.display import display
simu_indep = Path.cwd() / 'simus' / 'independent-imbalanced'
results = {
    "mv": [], "naivesoft": [], "glad": [],
    "ds": [], "dswc[l=5)": [], "dswc[l=10)": []
}
for strategy in results.keys():
    path_labels = simu_indep / "labels" / f"labels_independent-confusion_{strategy}.npy"
    ground_truth = np.load(simu_indep / "ground_truth.npy")
    labels = np.load(path_labels)
    acc = (
        np.mean(labels == ground_truth)
        if labels.ndim == 1
        else np.mean(
            np.argmax(labels, axis=1)
            == ground_truth
        )
    )
    results[strategy].append(acc)
results["NS"] = results["naivesoft"]
results.pop("naivesoft")
results = pd.DataFrame(results, index=['AccTrain'])
results.columns = map(str.upper, results.columns)
results = results.style.set_table_styles([dict(selector='th', props=[('text-align', 'center')])])
results.set_properties(**{'text-align': 'center'})
results = results.format(precision=3)

```

```
display(results)
```

Table 2: AccTrain metric on simulated independent mistakes with an imbalanced number of votes per task considering classical feature-blind label aggregation strategies

Table 2

	MV	GLAD	DS	DSWC[L=5]	DSWC[L=10]	NS
AccTrain	0.840	0.810	0.895	0.845	0.840	0.830

240 While more realistic, working with an imbalanced number of votes per task can lead to disrupting  
 241 orders of performance for some strategies (here GLAD is outperformed by other strategies).

### 242 3.2.2 Simulated correlated mistakes

243 The correlated mistakes are also known as the student-teacher or junior-expert setting (Cao et al.  
 244 (2019)). Consider that the crowd of workers is divided into two categories: teachers and students  
 245 (with  $n_{\text{teacher}} + n_{\text{student}} = n_{\text{worker}}$ ). Each student is randomly assigned to one teacher at the beginning  
 246 of the experiment. We generate the (diagonally dominant as in Section 3.2.1) confusion matrices of  
 247 each teacher and the students share the same confusion matrix as their associated teacher. Hence,  
 248 clustering strategies are expected to perform best in this context. Then, they all answer independently,  
 249 following a multinomial distribution with weights given at the row  $y_i^*$  of their confusion matrix  
 250  $\pi^{(j)} \in \mathbb{R}^{K \times K}$ .

251 We simulate  $n_{\text{task}} = 200$  tasks and  $n_{\text{worker}} = 30$  with 80% of students in the crowd. There are  $K = 5$   
 252 possible classes. Each task receives  $|\mathcal{A}(x_i)| = 10$  labels.

```
! peerannot simulate --n-worker=30 --n-task=200 --n-classes=5 \
--strategy student-teacher \
--ratio 0.8 \
--feedback=10 --seed 0 \
--folder ./simus/student_teacher

votes_path = Path.cwd() / "simus" / "student_teacher" / "answers.json"
metadata_path = Path.cwd() / "simus" / "student_teacher" / "metadata.json"
efforts = feedback_effort(votes_path)
workload = working_load(votes_path, metadata_path)
feedback = feedback_effort(votes_path)
utx.figure_simulations(workload, feedback)
plt.show()
```

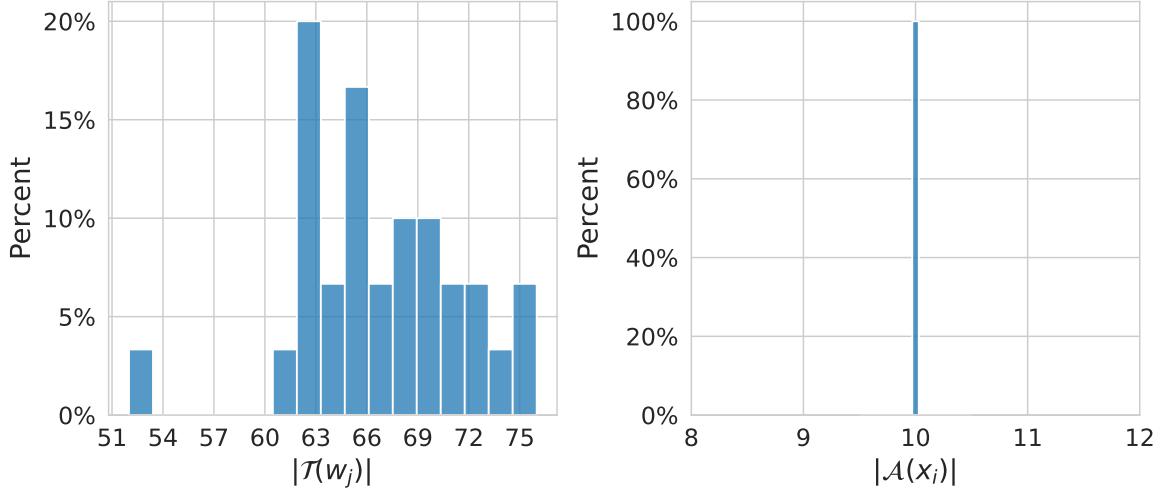


Figure 9: Distribution of number of tasks given per worker (left) and number of labels per task (right) in the correlated mistakes setting.

253 With the obtained answers, we can look at the aforementioned aggregation strategies performance:

```

for strat in ["MV", "NaiveSoft", "DS", "GLAD", "DSWC[L=5]", "DSWC[L=6]", "DSWC[L=10]"]:
    ! peerannot aggregate ./simus/student_teacher/ -s {strat}

simu_corr = Path.cwd() / 'simus' / "student_teacher"
results = {"mv": [], "naivesoft": [], "glad": [], "ds": [], "dswc[l=5)": [],
           "dswc[l=6)": [], "dswc[l=10)": []}
for strategy in results.keys():
    path_labels = simu_corr / "labels" / f"labels_student-teacher_{strategy}.npy"
    ground_truth = np.load(simu_corr / "ground_truth.npy")
    labels = np.load(path_labels)
    acc = (
        np.mean(labels == ground_truth)
        if labels.ndim == 1
        else np.mean(
            np.argmax(labels, axis=1)
            == ground_truth
        )
    )
    results[strategy].append(acc)
results["NS"] = results["naivesoft"]
results.pop("naivesoft")
results = pd.DataFrame(results, index=['AccTrain'])
results.columns = map(str.upper, results.columns)
results = results.style.set_table_styles(
    [dict(selector='th', props=[('text-align', 'center')])])
results.set_properties(**{'text-align': 'center'})
results = results.format(precision=3)
display(results)

```

Table 3: AccTrain metric on simulated correlated mistakes considering classical feature-blind label aggregation strategies

Table 3

	MV	GLAD	DS	DSWC[L=5]	DSWC[L=6]	DSWC[L=10]	NS
AccTrain	0.710	0.645	0.755	0.795	0.780	0.815	0.690

With Table 3, we see that with correlated data (24 students and 6 teachers), using 5 confusion matrices with DSWC[L=5] outperforms the vanilla DS strategy that does not consider the correlations. The best-performing method here estimates only 10 confusion matrices (instead of 30 for the vanilla DS model).

To summarize our simulations, we see that depending on workers answering strategies, different latent variable models perform best. However, these are unknown outside of a simulation framework, thus if we want to obtain labels from multiple responses, we need to investigate multiple models. This can be done easily with `peerannot` as we demonstrated using the `aggregate` module. However, one might not want to generate a label, simply learn a classifier to predict labels on unseen data. This leads us to another module part of `peerannot`.

### 3.3 More on confusion matrices in simulation settings

Moreover, the concept of confusion matrices has been commonly used to represent worker abilities. Let us remind that a confusion matrix  $\pi^{(j)} \in \mathbb{R}^{K \times K}$  of a worker  $w_j$  is defined such that  $\pi_{k,\ell}^{(j)} = \mathbb{P}(y_i^{(j)} = \ell | y_i^* = k)$ . These quantities need to be estimated since no true label is available in a crowd-sourced scenario. In practice, the confusion matrices of each worker is estimated via an aggregation strategy like Dawid and Skene's (Dawid and Skene 1979) presented in Section 3.1.

```
!peerannot simulate --n-worker=10 --n-task=100 --n-classes=5 \
--strategy hammer-spammer --feedback=5 --seed=0 \
--folder ./simus/hammer_spammer
!peerannot simulate --n-worker=10 --n-task=100 --n-classes=5 \
--strategy independent-confusion --feedback=5 --seed=0 \
--folder ./simus/hammer_spammer/confusion

mats = np.load("./simus/hammer_spammer/matrices.npy")
mats_confu = np.load("./simus/hammer_spammer/confusion/matrices.npy")

utx.figure_6(mats, mats_confu)
```

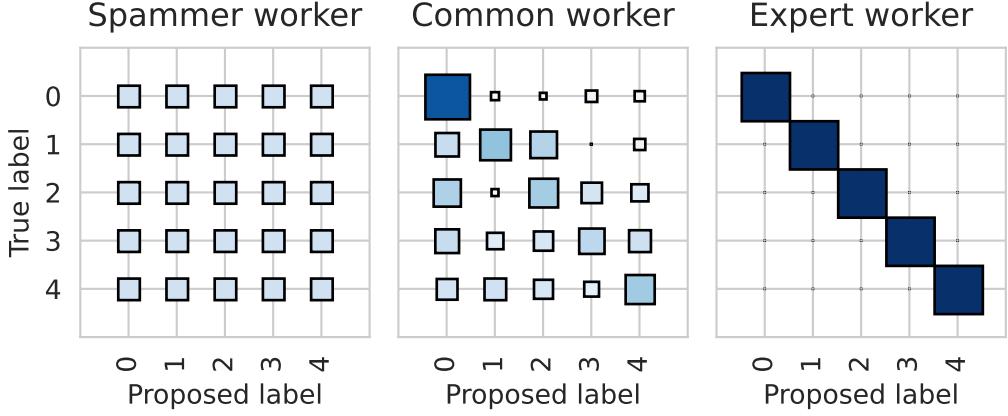


Figure 10: Three types of profiles of worker confusion matrices simulated with `peerannot`. The spammer answers independently of the true label. Expert workers identify classes without mistakes. In practice common workers are good for some classes but might confuse two (or more) labels. All workers are simulated using the `peerannot simulate` command presented in Section 3.2.

270 In Figure 10, we illustrate multiple workers' profile (as reflected by their confusion matrix) on  
 271 a simulate scenario where the ground truth is available. For that we generate toy datasets with  
 272 the `simulate` command from `peerannot`. In particular, we display a type of worker that can hurt  
 273 data quality: the spammer. Raykar and Yu (2011) defined a spammer as a worker that answers  
 274 independently of the true label:

$$\forall k \in [K], \mathbb{P}(y_i^{(j)} = k | y_i^* = k) = \mathbb{P}(y_i^{(j)} = k) . \quad (3)$$

275 Each row of the confusion matrix represents the label's probability distribution given a true label.  
 276 Hence, the spammer has a confusion matrix with near-identical rows. Apart from the spammer,  
 277 common mistakes often involve workers mixing up one or several classes. Expert workers have a  
 278 confusion matrix close to the identity matrix.

## 279 4 Learning from crowdsourced tasks

280 Commonly, tasks are crowdsourced to create a large annotated training set as modern machine  
 281 learning models require more and more data. The aggregation step then simply becomes the first step  
 282 in the complete learning pipeline. However, instead of aggregating labels, modern neural networks  
 283 are directly trained end-to-end from multiple noisy labels.

### 284 4.1 Popular models

285 In recent years, directly learning a classifier from noisy labels was introduced. Two of the most  
 286 used models: CrowdLayer (Rodrigues and Pereira 2018) and CoNAL (Chu, Ma, and Wang 2021), are  
 287 directly available in `peerannot`. These two learning strategies directly incorporate a DS-inspired  
 288 noise model in the neural network's architecture.

#### 289 4.1.1 CrowdLayer

290 `CrowdLayer` trains a classifier with noisy labels as follows. Let the scores (logits) output by a given  
 291 classifier neural network  $\mathcal{C}$  be  $z_i = \mathcal{C}(x_i)$ . Then CrowdLayer adds as a last layer  $\pi \in \mathbb{R}^{n_{\text{worker}} \times K \times K}$ , the

292 tensor of all  $\pi^{(j)}$ 's such that the crossentropy loss (CE) is adapted to the crowdsourcing setting into  
 293  $\mathcal{L}_{CE}^{\text{CrowdLayer}}$  and computed as:

$$\mathcal{L}_{CE}^{\text{CrowdLayer}}(x_i) = \sum_{j \in \mathcal{A}(x_i)} \text{CE}\left(\sigma\left(\pi^{(j)}\sigma(z_i)\right), y_i^{(j)}\right) ,$$

294 where the crossentropy loss for two distribution  $u, v \in \Delta_K$  is defined as  $\text{CE}(u, v) = \sum_{k \in [K]} v_k \log(u_k)$ .

295 Where DS modeled workers as confusion matrices, CrowdLayer adds a layer of  $\pi^{(j)}$ 's into the backbone  
 296 architecture as a new tensor layer to transform the output probabilities. The backbone classifier  
 297 predicts a distribution that is then corrupted through the added layer to learn the worker-specific  
 298 confusion. The weights in the tensor layer of  $\pi^{(j)}$ 's are learned during the optimization procedure.

#### 299 4.1.2 CoNAL

300 For some datasets, it was noticed that global confusion occurs between the proposed classes. It is the  
 301 case for example in the LabelMe dataset (Rodrigues et al. 2017) where classes overlap. In this case,  
 302 Chu, Ma, and Wang (2021) proposed to extend the CrowdLayer model by adding global confusion  
 303 matrix  $\pi^g \in \mathbb{R}^{K \times K}$  to the model on top of each worker's confusion.

304 Given the output  $z_i = \mathcal{C}(x_i) \in \mathbb{R}^K$  of a given classifier and task, CoNAL interpolates between the  
 305 prediction corrected by local confusions  $\pi^{(j)} z_i$  and the prediction corrected by a global confusion  
 306  $\pi^g z_i$ . The loss function is computed as follows:

$$\begin{aligned} \mathcal{L}_{CE}^{\text{CoNAL}}(x_i) &= \sum_{j \in \mathcal{A}(x_i)} \text{CE}(h_i^{(j)}, y_i^{(j)}) , \\ \text{with } h_i^{(j)} &= \sigma\left((\omega_i^{(j)}\pi^g + (1 - \omega_i^{(j)})\pi^{(j)})z_i\right) . \end{aligned}$$

307 The interpolation weight  $\omega_i^{(j)}$  is unobservable in practice. So, to compute  $h_i^{(j)}$ , the weight is obtained  
 308 through an auxiliary network. This network takes as input the image and worker information  
 309 and outputs a task-related vector  $v_i$  and a worker-related vector  $u_j$  of the same dimension. Finally,  
 310  $w_i^{(j)} = (1 + \exp(-u_j^\top v_i))^{-1}$ .

311 Both CrowdLayer and CoNAL model worker confusions directly in the classifier's weights to learn  
 312 from the noisy collected labels and are available in peerannot as we will see in the following.

## 313 4.2 Prediction error when learning from crowdsourced tasks

314 The AccTrain metric presented in Section 3.2 might no longer be of interest when training a classifier.  
 315 Classical error measurements involve a test dataset to estimate the generalization error. To do so, we  
 316 present hereafter two error metrics. Assuming we trained our classifier  $\mathcal{C}$  on a training set and that  
 317 there is a test set available with known true labels:

- 318 the test accuracy is computed as  $\frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \mathbf{1}_{\{y_i^* = \hat{y}_i\}}$ .
- 319 the expected calibration error (Guo et al. 2017) over  $M$  equally spaced bins  $I_1, \dots, I_M$  partitioning  
 320 the interval  $[0, 1]$ , is computed as:

$$\text{ECE} = \sum_{m=1}^M \frac{|B_m|}{n_{\text{task}}} |\text{acc}(B_m) - \text{conf}(B_m)| ,$$

321 with  $B_m = \{x_i | \mathcal{C}(x_i)[1] \in I_m\}$  the tasks with predicted probability in the  $m$ -th bin,  $\text{acc}(B_m)$   
 322 the accuracy of the network for the samples in  $B_m$  and  $\text{conf}(B_m)$  the associated empirical  
 323 confidence. More precisely:

$$\text{acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbf{1}(\hat{y}_i = y_i^*) \quad \text{and} \quad \text{conf}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \sigma(\mathcal{C}(x_i))_{[1]} .$$

324 The accuracy represents how well the classifier generalizes, and the expected calibration error (ECE)  
 325 quantifies the deviation between the accuracy and the confidence of the classifier. Modern neural  
 326 networks are known to often be overconfident in their predictions (Guo et al. 2017). However, it has  
 327 also been remarked that training on crowdsourced data, depending on the strategy, mitigates this  
 328 confidence issue. That is why we propose to compare them both in our coming experiments. Note  
 329 that the ECE error estimator is known to be biased (Gruber and Buettner 2022). Smaller training  
 330 sets are known to have a higher ECE estimation error. And in the crowdsourcing setting, openly  
 331 available datasets are often quite small.

### 332 4.3 Use case with peerannot on real datasets

333 Few real crowdsourcing experiments have been released publicly. Among the available ones,  
 334 CIFAR-10H (Peterson et al. 2019) is one of the largest with 10000 tasks labeled by workers (the  
 335 testing set of CIFAR-10). The main limitation of CIFAR-10H is that there are few disagreements  
 336 between workers and a simple majority voting already leads to a near-perfect AccTrain error. Hence,  
 337 comparing the impact of aggregation and end-to-end strategies might not be relevant (Peterson et al.  
 338 2019; Aitchison 2021), it is however a good benchmark for task difficulty identification and worker  
 339 evaluation scoring. Each of these dataset contains a test set, with known ground truth. Thus, we can  
 340 train a classifier from the crowdsourced data, and compare predictive performance on the test set.

341 The LabelMe dataset was extracted from crowdsourcing segmentation experiments and a subset of  
 342  $K = 8$  classes was released in Rodrigues et al. (2017).

343 Let us use peerannot to train a VGG-16 with two dense layers on the LabelMe dataset. Note that  
 344 this modification was introduced to reach state-of-the-art performance in (Chu, Ma, and Wang 2021).  
 345 Other models from the torchvision library can be used, such as Resnets, Alexnet etc.

```

for strat in ["MV", "NaiveSoft", "DS", "GLAD"]:
    ! peerannot aggregate ./labelme/ -s {strat}
    ! peerannot train ./labelme -o labelme_{strat} \
        -K 8 --labels=./labelme/labels/labels_labelme_{strat}.npy \
        --model modellabelme --n-epochs 500 -m 50 -M 150 -m 250 \
        --scheduler=multistep --lr=0.01 --num-workers=8 \
        --pretrained --data-augmentation --optimizer=adam \
        --batch-size=32 --img-size=224 --seed=1
for strat in ["CrowdLayer", "CoNAL[scale=0]", "CoNAL[scale=1e-4]"]:
    ! peerannot aggregate-deep ./labelme -o labelme_{strat} \
        --answers ./labelme/answers.json -s {strat} --model modellabelme \
        --img-size=224 --pretrained --n-classes=8 --n-epochs=500 --lr=0.001 \
        -m 300 -M 400 --scheduler=multistep --batch-size=228 --optimizer=adam \
        --num-workers=8 --data-augmentation --seed=1

# command to save separately a specific part of CoNAL model (memory intensive otherwise)
path_ = Path.cwd() / "datasets" / "labelme"
best_conal = torch.load(path_ / "best_models" / "labelme_conal[scale=1e-4].pth",

```

```

map_location="cpu")
torch.save(best_conal["noise_adaptation"]["local_confusion_matrices"],
path_ / "best_models" / "labelme_conal[scale=1e-4].local_confusion.pth")

def highlight_max(s, props=''):
    return np.where(s == np.nanmax(s.values), props, '')

def highlight_min(s, props=''):
    return np.where(s == np.nanmin(s.values), props, '')

import json
dir_results = Path().cwd() / 'datasets' / "labelme" / "results"
meth, accuracy, ece = [], [], []
for res in dir_results.glob("modellabelme/*"):
    filename = res.stem
    _, mm = filename.split("_")
    meth.append(mm)
    with open(res, "r") as f:
        dd = json.load(f)
        accuracy.append(dd["test_accuracy"])
        ece.append(dd["test_ece"])
results = pd.DataFrame(list(zip(meth, accuracy, ece)),
                       columns=["method", "AccTest", "ECE"])
results["method"] = [
    "NS", "CoNAL[scale=0]", "CrowdLayer", "CoNAL[scale=1e-4]", "MV", "DS", "GLAD"
]
results = results.sort_values(by="AccTest", ascending=True)
results.reset_index(drop=True, inplace=True)
results = results.style.set_table_styles([dict(selector='th', props=[
    ('text-align', 'center')])])
results.set_properties(**{'text-align': 'center'})
results = results.format(precision=3)
results.apply(highlight_max, props='background-color:#e6ffe6;',
             axis=0, subset=["AccTest"])
results.apply(highlight_min, props='background-color:#e6ffe6;',
             axis=0, subset=["ECE"])
display(results)

```

Table 4: Generalization performance on LabelMe dataset depending on the learning strategy from the crowdsourced labels. The network used is a VGG-16 with two dense layers for all methods.

Table 4

	method	AccTest	ECE
0	MV	81.061	0.189
1	CoNAL[scale=1e-4]	85.606	0.143
2	DS	86.448	0.136
3	CoNAL[scale=0]	87.205	0.117
4	NS	87.542	0.124

	method	AccTest	ECE
5	CrowdLayer	88.468	0.115
6	GLAD	88.889	0.112

346 As we can see, CoNAL strategy performs best. In this case, it is expected behavior as CoNAL  
 347 was created for the LabelMe dataset. However, using `peerannot` we can look into **why modeling**  
 348 **common confusion returns better results with this dataset**. To do so, we can explore the  
 349 datasets from two points of view: worker-wise or task-wise in Section 5.

## 350 5 Identifying tasks difficulty and worker abilities

351 If a dataset requires crowdsourcing to be labeled, it is because expert knowledge is long and costly to  
 352 obtain. In the era of big data, where datasets are built using web scraping (or using a platform like  
 353 [Amazon Mechanical Turk](#)), citizen science is popular as it is an easy way to produce many labels.

354 However, mistakes and confusions happen during these experiments. Sometimes involuntarily  
 355 (e.g., because the task is too hard or the worker is unable to differentiate between two classes) and  
 356 sometimes voluntarily (e.g., the worker is a spammer).

357 Underlying all the learning models and aggregation strategies, the cornerstone of crowdsourcing  
 358 is evaluating the trust we put in each worker depending on the presented task. And with the  
 359 gamification of crowdsourcing (Servajean et al. 2016; Tinati et al. 2017), it has become essential to  
 360 find scoring metrics both for workers and tasks to keep citizens in the loop so to speak. This is the  
 361 purpose of the identification module in `peerannot`.

362 Our test cases are both the CIFAR-10H dataset and the LabelMe dataset to compare the worker and  
 363 task evaluation depending on the number of votes collected. Indeed, the LabelMe dataset has only  
 364 up to three votes per task whereas CIFAR-10H accounts for nearly fifty votes per task.

### 365 5.1 Exploring tasks' difficulty

366 To explore the tasks' intrinsic difficulty, we propose to compare three scoring metrics:

- 367 • the entropy of the NS distribution: the entropy measures the inherent uncertainty of the  
 368 distribution to the possible outcomes. It is reliable with a big enough and not adversarial crowd.  
 369 More formally:

$$365 \quad \forall i \in [n_{\text{task}}], \text{Entropy}(\hat{y}_i^{NS}) = - \sum_{k \in [K]} (\hat{y}_i^{NS})_k \log ((\hat{y}_i^{NS})_k) .$$

- 370 • GLAD's scoring: by construction, Whitehill et al. (2009) introduced a scalar coefficient to score  
 371 the difficulty of a task.
- 372 • the Weighted Area Under the Margins (WAUM): introduced by Lefort et al. (2022), this weighted  
 373 area under the margins indicates how difficult it is for a classifier  $\mathcal{C}$  to learn a task's label. This  
 374 procedure is done with a budget of  $T > 0$  epochs. Given the crowdsourced labels and the trust  
 375 we have in each worker denoted  $s^{(j)}(x_i) > 0$ , the WAUM of a given task  $x_i \in \mathcal{X}$  and a set of  
 376 crowdsourced labels  $\{y_i^{(j)}\}_{j \in [K]} \in [K]^{\mathcal{A}(x_i)}$  is defined as:

$$377 \quad \text{WAUM}(x_i) := \frac{1}{|\mathcal{A}(x_i)|} \sum_{j \in \mathcal{A}(x_i)} s^{(j)}(x_i) \left\{ \frac{1}{T} \sum_{t=1}^T \sigma(\mathcal{C}(x_i))_{y_i^{(j)}} - \sigma(\mathcal{C}(x_i))_{[2]} \right\} ,$$

378 where we remind that  $\mathcal{C}(x_i)_{[2]}$  is the second largest probability output by the classifier  $\mathcal{C}$  for  
 the task  $x_i$ .

379 The weights  $s^{(j)}(x_i)$  are computed à la Servajean et al. (2017):

$$\forall j \in [n_{\text{worker}}], \forall i \in [n_{\text{task}}], s^{(j)}(x_i) = \langle \sigma(\mathcal{C}(x_i)), \text{diag}(\hat{\pi}^{(j)}) \rangle ,$$

380 where  $\hat{\pi}^{(j)}$  is the estimated confusion matrix of worker  $w_j$  (by default, the estimation provided by  
381 DS).

382 The WAUM is a generalization of the AUM by Pleiss et al. (2020) to the crowdsourcing setting. A  
383 high WAUM indicates a high trust in the task classification by the network given the crowd labels. A  
384 low WAUM indicates difficulty for the network to classify the task into the given classes (taking into  
385 consideration the trust we have in each worker for the task considered). Where other methods only  
386 consider the labels and not directly the tasks, the WAUM directly considers the learning trajectories  
387 to identify ambiguous tasks. One pitfall of the WAUM is that it is dependent on the architecture used.

388 Note that each of these statistics could prove useful in different contexts. The entropy is irrelevant in  
389 settings with few labels per task (small  $|\mathcal{A}(x_i)|$ ). For instance, it is uninformative for LabelMe dataset.  
390 The WAUM can handle any number of labels, but the larger the better. However, as it uses a deep  
391 learning classifier, the WAUM needs the tasks  $(x_i)_i$  in addition to the proposed labels while the other  
392 strategies are feature-blind.

### 393 5.1.1 CIFAR-10H dataset

394 First, let us consider a dataset with a large number of tasks, annotations and workers: the CIFAR-10H  
395 dataset by Peterson et al. (2019).

```
! peerannot identify ./datasets/cifar10H -s entropy -K 10 --labels ./datasets/cifar10H/answers.json
! peerannot aggregate ./datasets/cifar10H/ -s GLAD
! peerannot identify ./datasets/cifar10H/ -K 10 --method WAUM \
    --labels ./datasets/cifar10H/answers.json --model resnet34 \
    --n-epochs 100 --lr=0.01 --img-size=32 --maxiter-DS=50 \
    --pretrained

import plotly.graph_objects as go
from plotly.subplots import make_subplots
from PIL import Image
import itertools

classes = (
    "plane",
    "car",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
)
n_classes = 10
all_images = utx.load_data("cifar10H", n_classes, classes)
utx.generate_plot(n_classes, all_images, classes)
```

```

396 Unable to display output for mime type(s): text/html
397 Most difficult tasks sorted by class from MV aggregation identified depending on the strategy used
398 (entropy, GLAD or WAUM) using a Resnet34.
399 Unable to display output for mime type(s): text/html

400
401 The entropy, GLAD's difficulty, and WAUM's difficulty each show different images as exhibited in
402 the interactive Figure. While the entropy and GLAD output similar tasks, in this case, the WAUM
403 often differs. We can also observe an ambiguity induced by the labels in the truck category, with the
404 presence of a trailer that is technically a mixup between a car and a truck.

405 5.1.2 LabelMe dataset
406 As for the LabelMe dataset, one difficulty in evaluating tasks' intrinsic difficulty is that there is a
407 limited amount of votes available per task. Hence, the entropy in the distribution of the votes is no
408 longer a reliable metric, and we need to rely on other models.
409 Now, let us compare the tasks' difficulty distribution depending on the strategy considered using
410 peerannot.

! peerannot identify ./datasets/labelme -s entropy -K 8 \
--labels ./datasets/labelme/answers.json
! peerannot aggregate ./datasets/labelme/ -s GLAD
! peerannot identify ./datasets/labelme/ -K 8 --method WAUM \
--labels ./datasets/labelme/answers.json --model modellabelme --lr=0.01 \
--n-epochs 100 --maxiter-DS=100 --alpha=0.01 --pretrained --optimizer=sgd

classes = {
    0: "coast",
    1: "forest",
    2: "highway",
    3: "insidecity",
    4: "mountain",
    5: "opencountry",
    6: "street",
    7: "tallbuilding",
}
classes = list(classes.values())
n_classes = len(classes)
all_images = utx.load_data("labelme", n_classes, classes)
utx.generate_plot(n_classes, all_images, classes) # create interactive plot

411 Unable to display output for mime type(s): text/html
412 Most difficult tasks sorted by class from MV aggregation identified depending on the strategy used
413 (entropy, GLAD or WAUM) using a VGG-16 with two dense layers.

414
415 Note that in this experiment, because the number of labels given per task is in {1, 2, 3}, the entropy
416 only takes four values. In particular, tasks with only one label all have a null entropy, so not just
417 consensual tasks. The MV is also not suited in this case because of the low number of votes per task.

```

418 The underlying difficulty of these tasks mainly comes from the overlap in possible labels. For example,  
 419 tallbuildings are most often found insidecities, and so are streets. In the opencountry we  
 420 find forests, river-coasts and mountains.

## 421 5.2 Identification of worker reliability and task difficulty

422 From the labels, we can explore different worker evaluation scores. GLAD's strategy estimates a  
 423 reliability scalar coefficient  $\alpha_j$  per worker. With strategies looking to estimate confusion matrices,  
 424 we investigate two scoring rules for workers:

- 425 • The trace of the confusion matrix: the closer to  $K$  the better the worker.
- 426 • The closeness to spammer metric (Raykar and Yu 2011) (also called spammer score) that is the  
 427 Frobenius norm between the estimated confusion matrix  $\hat{\pi}^{(j)}$  and the closest rank-1 matrix.  
 428 The further to zero the better the worker. On the contrary, the closer to zero, the more likely it  
 429 is for the worker to be a spammer. This score separates spammers from common workers and  
 430 experts (with profiles as in Figure 10).

431 When the tasks are available, confusion-matrix-based deep learning models can also be used. We  
 432 thus add to the comparison the trace of the confusion matrices with CrowdLayer and CoNAL on  
 433 the LabelMe datasets. For CoNAL, we only consider the trace of the confusion matrix  $\pi^{(j)}$  in the  
 434 pairwise comparison. Moreover, for CrowdLayer and CoNAL we show in Figure 12 the weights  
 435 learned without the softmax operation by row to keep the comparison as simple as possible with the  
 436 actual outputs of the model.

437 Comparisons in Figure 11 and Figure 12 are plotted pairwise between the evaluated metrics. Each  
 438 point represents a worker. Each off-diagonal plot shows the joint distribution between the scores of  
 439 the y-axis row and the x-axis column. They allow us to visualize the relationship between these two  
 440 variables. The main diagonal represents the (smoothed) marginal distribution of the score of the  
 441 considered column.

### 442 5.2.1 CIFAR-10H

443 The CIFAR-10H dataset has few disagreements among workers. However, these strategies disagree  
 444 on the ranking of good against best workers as they do not measure the same properties.

```
! peerannot aggregate ./datasets/cifar10H/ -s GLAD
for method in ["trace_confusion", "spam_score"]:
    ! peerannot identify ./datasets/cifar10H/ --n-classes=10 \
        -s {method} --labels ./datasets/cifar10H/answers.json

path_ = Path.cwd() / "datasets" / "cifar10H"
results_identif = {"Trace DS": [], "spam_score": [], "glad": []}
results_identif["Trace DS"].extend(np.load(path_ / 'identification' / "traces_confusion.npy"))
results_identif["spam_score"].extend(np.load(path_ / 'identification' / "spam_score.npy"))
results_identif["glad"].extend(np.load(path_ / 'identification' / "glad" / "abilities.npy")[:, 1])
results_identif = pd.DataFrame(results_identif)
g = sns.pairplot(results_identif, corner=True, diag_kind="kde", plot_kws={'alpha':0.2})
plt.tight_layout()
plt.show()
```

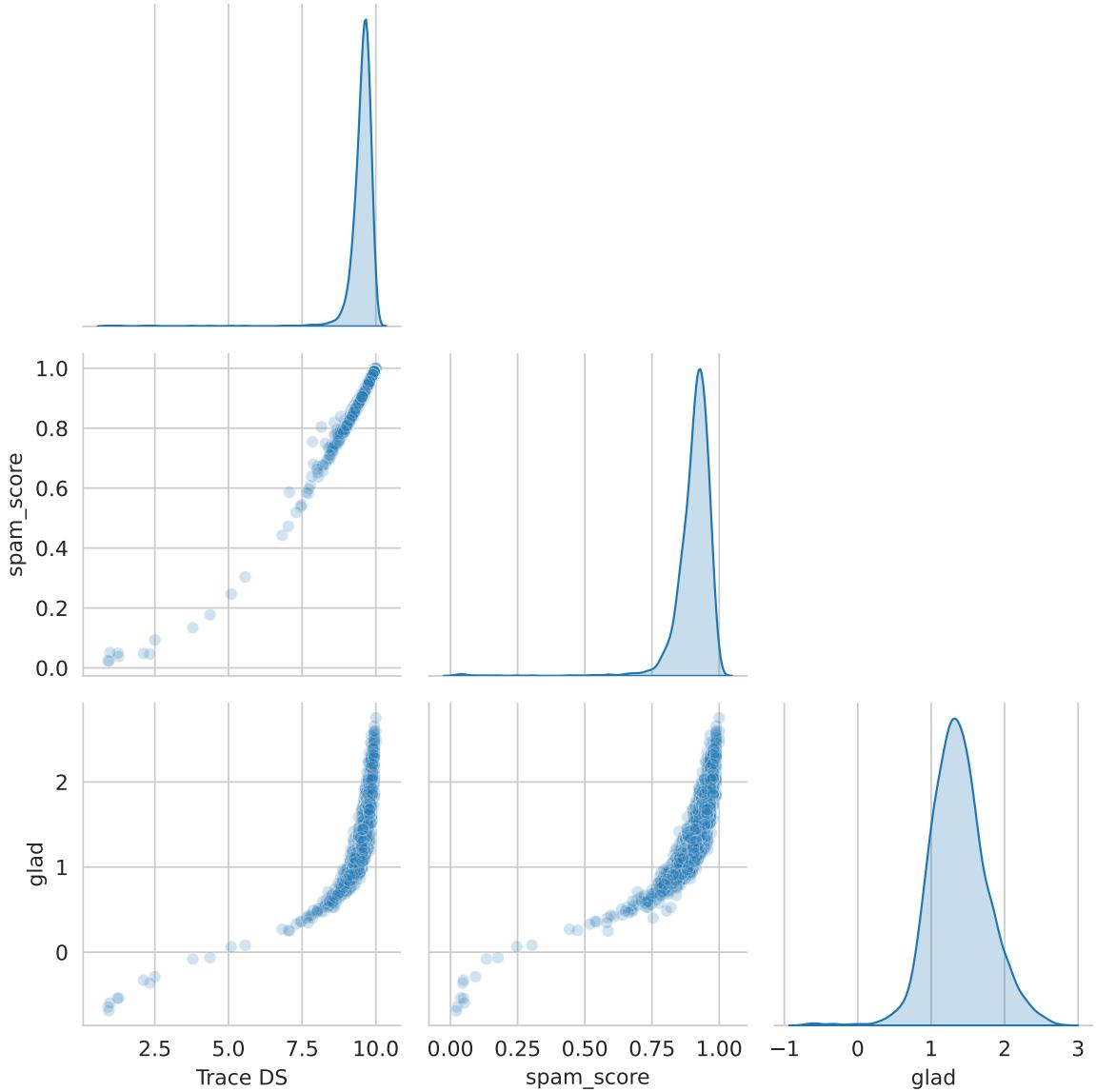


Figure 11: Comparison of ability scores by workers for the CIFAR-10H dataset. All metrics computed identify the same poorly performing workers. A mass of good and expert workers can be seen as the dataset presents few disagreements, thus few data to discriminate expert workers from the others.

From Figure 11, we can see that in this dataset, different methods easily separate the worst workers from the rest of the crowd (workers in the left tail of the distribution).

#### 5.2.2 LabelMe

Finally, let us evaluate workers for the LabelMe dataset. Because of the lack of data (up to 3 labels per task), ranking workers is more difficult than in the CIFAR-10H dataset.

```
! peerannot aggregate ./datasets/labelme/ -s GLAD
for method in ["trace_confusion", "spam_score"]:
    ! peerannot identify ./datasets/labelme/ --n-classes=8 \
        -s {method} --labels ./datasets/labelme/answers.json
# CoNAL and CrowdLayer were run in section 4
```

```

path_ = Path.cwd() / "datasets" / "labelme"
results_identif = {
    "Trace DS": [],
    "Spam score": [],
    "glad": [],
    "Trace CrowdLayer": [],
    "Trace CoNAL[scale=1e-4)": []
}
best_cl = torch.load(
    path_ / "best_models" / "labelme_crowdlayer.pth", map_location="cpu"
)
best_conal = torch.load(
    path_ / "best_models" / "labelme_conal[scale=1e-4]_local_confusion.pth",
    map_location="cpu",
)
pi_conal = best_conal
results_identif["Trace CoNAL[scale=1e-4]"].extend(
    [torch.trace(pi_conal[i]).item() for i in range(pi_conal.shape[0])]
)
results_identif["Trace CrowdLayer"].extend(
    [
        torch.trace(best_cl["confusion"][i]).item()
        for i in range(best_cl["confusion"].shape[0])
    ]
)
results_identif["Trace DS"].extend(
    np.load(path_ / "identification" / "traces_confusion.npy")
)
results_identif["Spam score"].extend(
    np.load(path_ / "identification" / "spam_score.npy")
)
results_identif["glad"].extend(
    np.load(path_ / "identification" / "glad" / "abilities.npy")[:, 1]
)
results_identif = pd.DataFrame(results_identif)
g = sns.pairplot(
    results_identif, corner=True, diag_kind="kde", plot_kws={"alpha": 0.2}
)
plt.tight_layout()
plt.show()

```

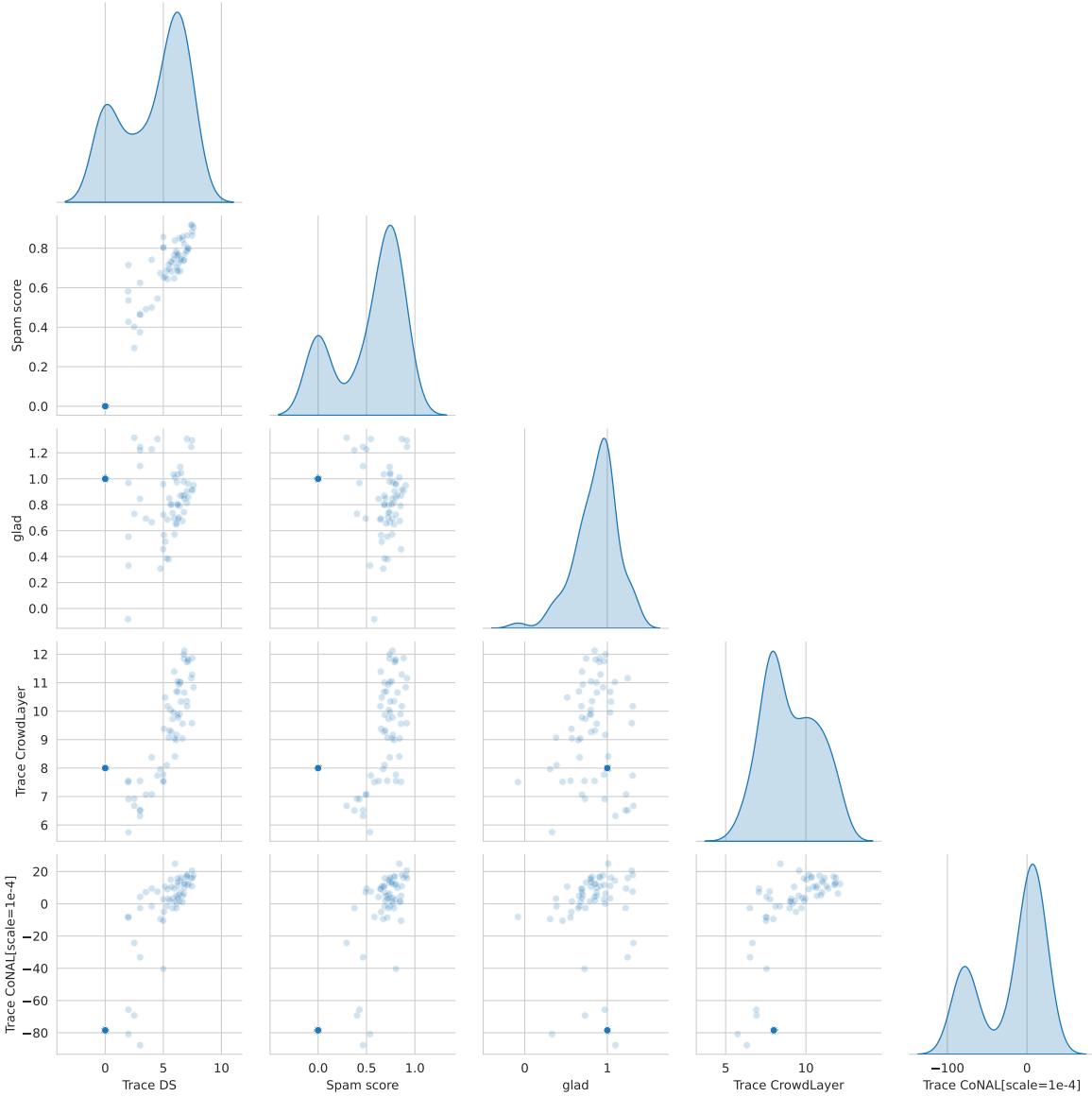


Figure 12: Comparison of ability scores by workers for the LabelMe dataset. With few labels per task, workers are more difficult to rank. It is more difficult to separate workers with their abilities in this crowd. Hence the importance of investigating the generalization performance of the methods presented in the previous section.

We can see in Figure 12 that the number of labels available by task highly impacts the worker evaluation scores. The spam score, DS model and CoNAL all show similar results in the distribution shape (bimodal distribution) whereas GLAD and CrowdLayer are more concentrated. However, this does not account for the ranking of a given worker by the methods considered. The exploration of the dataset lets us look at different scores, but generalization performance presented in Section 4.3 should also be considered in crowdsourcing. This difference in worker evaluation scores indeed further highlights the importance of using multiple test metrics to compare the model’s prediction performance in crowdsourcing. We have seen that the library `peerannot` allows users to explore the datasets, both in terms of tasks and workers, and easily compare predictive performance in this setting.

In practice, the data exploration step can be used to detect possible ambiguities in the dataset’s tasks,

461 but also remove answers from spammers to improve the data quality as shown in Figure 1. The easy  
 462 access to the different strategies allows the user to decide if, for their collected dataset, there is a  
 463 need for more recent deep-learning-based strategies to improve the results. This is the case for the  
 464 LabelMe dataset. Otherwise, the user can decide that standard aggregation-based crowdsourcing  
 465 strategies are sufficient and for example, if there are plenty of votes per task like in CIFAR-10H, that  
 466 the entropy of the vote distribution is a criterion that identified enough ambiguous tasks for their  
 467 case. As often, not a single strategy works best for all datasets, hence the need to perform easy  
 468 comparisons with peerannot.

## 469 6 Conclusion

470 We introduced `peerannot`, a library to handle crowdsourced datasets. This library enables both  
 471 easy label aggregation and direct training strategies with classical state-of-the-art classifiers. The  
 472 identification module of the library allows exploring the collected data from both the tasks and the  
 473 workers' point of view for better scorings and data cleaning procedures. Our library also comes  
 474 with templated datasets to better share crowdsourced datasets. Going beyond templating, it helps  
 475 the crowdsourcing community to have openly accessible strategies to test, compare and improve to  
 476 develop common strategies to analyze more and more common crowdsourced datasets.

477 We hope that this library helps reproducibility in the crowdsourcing community and also standardizes  
 478 training from crowdsourced datasets. New strategies can easily be incorporated into the open-source  
 479 code [available on GitHub](#). Finally, as `peerannot` is mostly directed to handle classification datasets,  
 480 one of our future works would be to consider other `peerannot` modules to handle crowdsourcing for  
 481 object detection, segmentation and even worker evaluation in other contexts like peer-grading.

## 482 7 Appendix

### 483 7.1 Supplementary simulation: Simulated mistakes with discrete difficulty levels 484 on tasks

485 For an additional simulation setting, we consider the so-called discrete difficulty presented in Whitehill  
 486 et al. (2009). Contrary to other simulations, we here consider that workers belong to two levels of  
 487 abilities: good or bad, and tasks have two levels of difficulty: easy or hard. The keyword `ratio-diff`  
 488 indicates the prevalence of each level of difficulty, it is defined as the ratio of easy tasks over hard  
 489 tasks:

$$490 \text{ratio-diff} = \frac{P(\text{easy})}{P(\text{hard})} \text{ with } P(\text{easy}) + P(\text{hard}) = 1 .$$

490 Difficulties are then drawn [at random](#). Tasks that are `easy` are answered correctly by every worker.  
 491 Tasks that are `hard` are answered following the confusion matrix assigned to each worker (as in  
 492 Section 3.2.1). Each worker then answers independently to the presented tasks.

493 We simulate  $n_{\text{task}} = 500$  tasks and  $n_{\text{worker}} = 100$  with 35% of good workers in the crowd and 50% of  
 494 easy tasks. There are  $K = 5$  possible classes. Each task receives  $|\mathcal{A}(x_i)| = 10$  labels.

```
! peerannot simulate --n-worker=100 --n-task=200 --n-classes=5 \
--strategy discrete-difficulty \
--ratio 0.35 --ratio-diff 1 \
--feedback 10 --seed 0 \
--folder ./simus/discrete_difficulty
```

```

votes_path = Path.cwd() / "simus" / "discrete_difficulty" / "answers.json"
metadata_path = Path.cwd() / "simus" / "discrete_difficulty" / "metadata.json"
efforts = feedback_effort(votes_path)
workload = working_load(votes_path, metadata_path)
feedback = feedback_effort(votes_path)
utx.figure_simulations(workload, feedback)
plt.show()

```

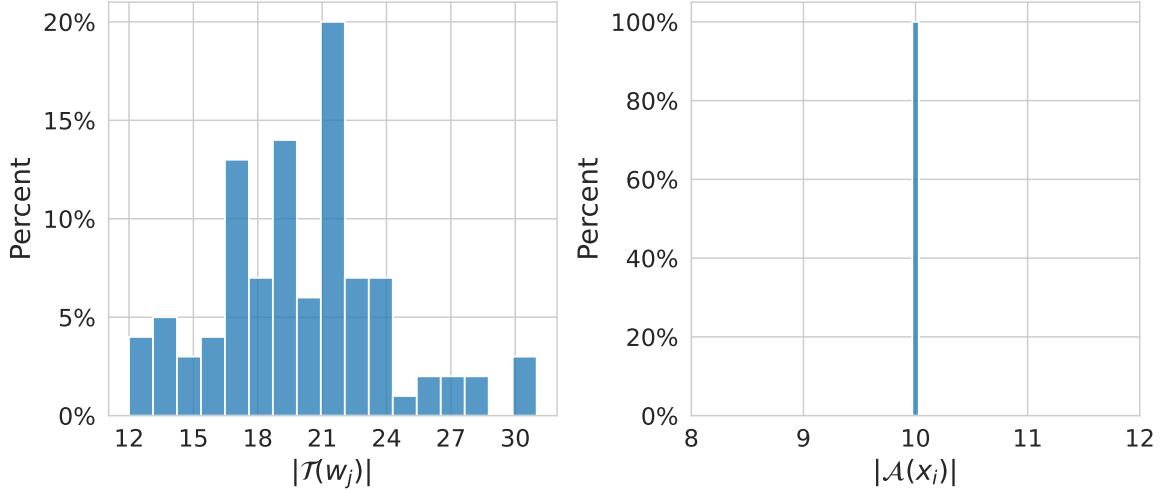


Figure 13: Distribution of the number of tasks given per worker (left) and of the number of labels per task (right) in the setting with simulated discrete difficulty levels.

495 With the obtained answers, we can look at the aforementioned aggregation strategies performance:

```

for strat in ["MV", "NaiveSoft", "DS", "GLAD", "DSWC[L=2]", "DSWC[L=5]"]:
    ! peerannot aggregate ./simus/discrete_difficulty/ -s {strat}

simu_corr = Path.cwd() / 'simus' / "discrete_difficulty"
results = {
    "mv": [], "naivesoft": [], "glad": [],
    "ds": [], "dswc[l=2)": [], "dswc[l=5)": []
}
for strategy in results.keys():
    path_labels = simu_corr / "labels" / f"labels_discrete-difficulty_{strategy}.npy"
    ground_truth = np.load(simu_corr / "ground_truth.npy")
    labels = np.load(path_labels)
    acc = (
        np.mean(labels == ground_truth)
        if labels.ndim == 1
        else np.mean(
            np.argmax(labels, axis=1)
            == ground_truth
        )
    )
    results[strategy].append(acc)
results["NS"] = results["naivesoft"]
results.pop("naivesoft")

```

```

results = pd.DataFrame(results, index=['AccTrain'])
results.columns = map(str.upper, results.columns)
results = results.style.set_table_styles([dict(selector='th', props=[('text-align', 'center')])])
results.set_properties(**{'text-align': 'center'})
results = results.format(precision=3)
display(results)

```

Table 5: AccTrain metric on simulated mistakes made when tasks are associated with a difficulty level considering classical feature-blind label aggregation strategies.

Table 5

	MV	GLAD	DS	DSWC[L=2]	DSWC[L=5]	NS
AccTrain	0.820	0.845	0.810	0.600	0.660	0.790

Finally, in this setting involving task difficulty coefficients, the only strategy that involves a latent variable for the task difficulty, knowing GLAD, outperforms the other strategies (see Table 5). Note that in this case, creating clusters of answers leads to worse decisions than an MV aggregation.

## 7.2 Comparison with other libraries

In this section, we provide several comparisons with the Ustalov, Pavlichenko, and Tseitlin (2023) library.

- Framework: `peerannot` focuses on image classification problems with categorical answers. `crowd-kit` also considers textual responses and image segmentation with three aggregation strategies for each field.
- Data storage: `peerannot` introduces this `.json` storage that can handle large datasets. `crowd-kit` stores the collected data in a `.csv` file with columns `task`, `worker`, `label`.
- Identification module: one of the major differences between the two libraries resides in the `identification` module of `peerannot`. This module allows us to explore the dataset and detect poorly performing workers / difficult tasks easily. `crowd-kit` only allows us to explore workers with the `accuracy_on_aggregation` metric that computes the accuracy of a worker given aggregated hard labels. `peerannot`, as demonstrated in Section 5, proposes several metrics such as the spam score, GLAD's worker ability coefficient and the trace of the confusion matrices. As for the task side, `peerannot` proposes the different popular metrics in `crowd-kit` accompanied with the WAUM (and also the AUMC) metrics from Lefort et al. (2022) and GLAD's difficulty coefficients.
- Training: `peerannot` lets users directly train a neural network architecture from the aggregated labels. This feature is not proposed by `crowd-kit`.
- Simulation: `peerannot` created a `simulate` module to check strategies on. This feature is also not in the `crowd-kit` library.

Finally, to compare different strategies across libraries, we implemented a [crowdsourcing benchmark](#) in the `Benchopt` (Moreau et al. (2022)) library. The `Benchopt` library allows users to easily compare and reproduce optimization problem benchmarks between multiple frameworks. After running each strategy, we measure the cumulated time taken to reach the optimum during the optimization steps. The metric measured on the y-axis is the AccTrain. Each strategy is run 5 times until convergence. The differences in results across iterations for the MV strategy come from the randomness in the choice in case of equalities. We provide a clone of the crowdsourcing benchmark and the results are obtained by running the following command:

```
benchopt run ./benchmark_crowdsourcing
```

528 First, let us see the performances on the **Bluebirds** dataset, a small dataset with 39 workers, 108 tasks  
 529 and  $K = 2$  classes.

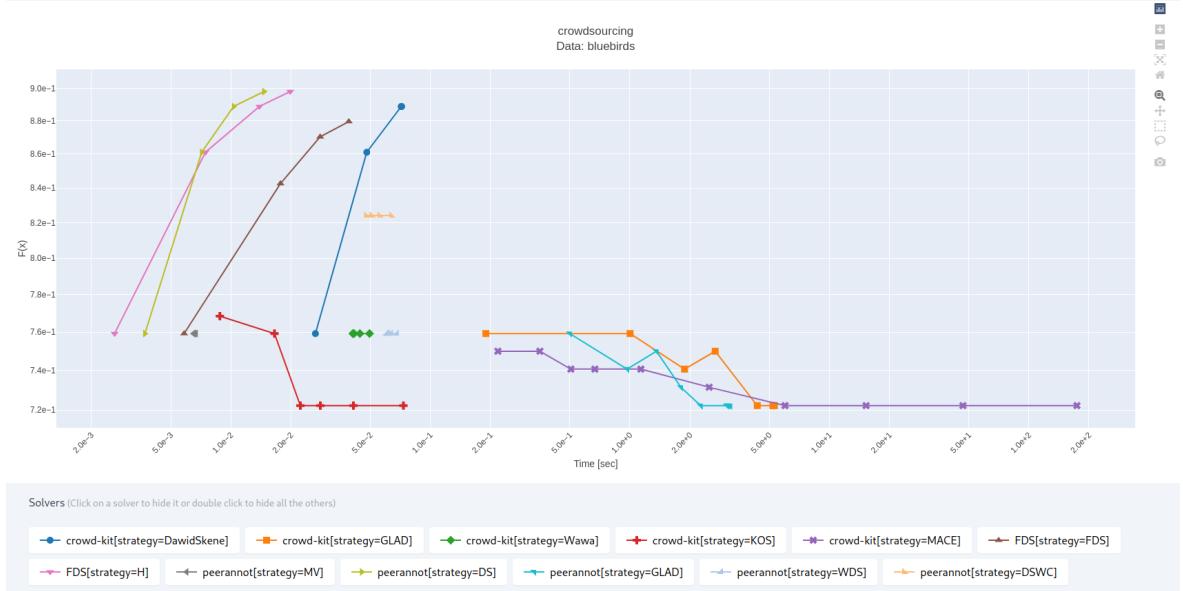


Figure 14: Aggregation strategies computational time during optimization procedure for the BlueBirds dataset with  $K=2$ .

530 We see in Figure 14 that the DS strategy from peerannot is the first to reach the optimum, followed  
 531 by the [Fast-DS strategy](#) and then crowd-kit DS. Other strategies do not lead to better accuracy on  
 532 this dataset and DS seems to be the best fitting strategy.

533 For the LabelMe dataset, DS strategy is also the best aggregation strategy, faster for crowd-kit. The  
 534 sensitivity of GLAD's method to the priors on  $\alpha$  and  $\beta$  parameters can lead to large performance  
 535 differences for real datasets as we see in Figure 15. Note that crowd-kit's KOS strategy is not  
 536 available for this dataset as it is only made for binary classification datasets.

### 537 7.3 Examples of images in CIFAR-10H and Labelme

538 In this section, we provide examples of images from the CIFAR-10H and LabelMe datasets. Both of  
 539 these datasets came with known true labels. For CIFAR-10H, the true labels were from the original  
 540 CIFAR-10 dataset. For LabelMe, the true labels were determined by the authors at release.

```
utx.figure_3()
```

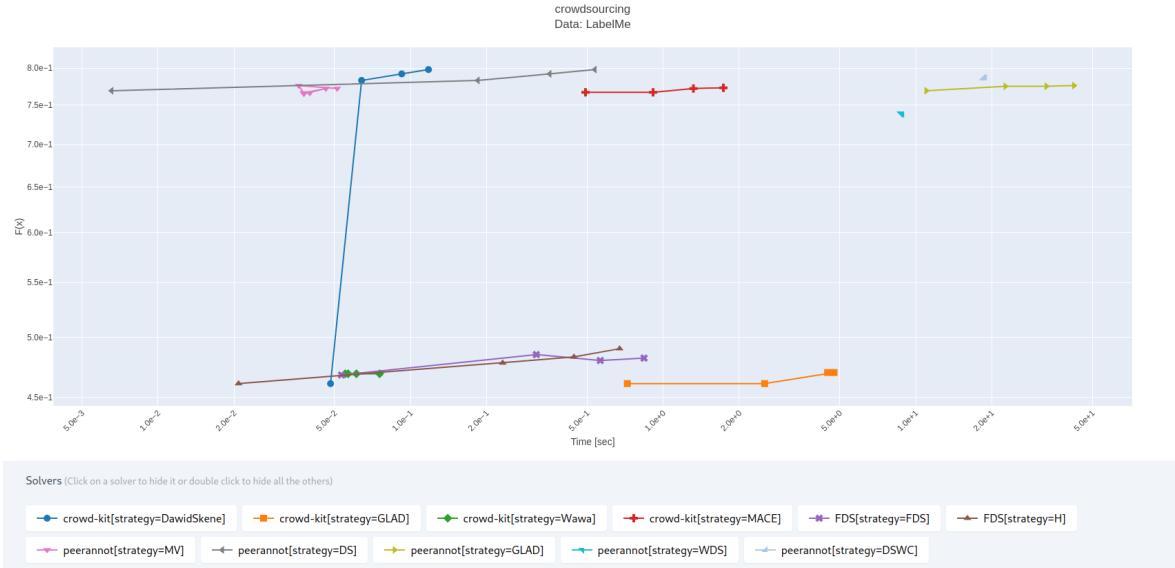


Figure 15: Aggregation strategies computational time during optimization procedure for the LabelMe dataset with  $K=8$

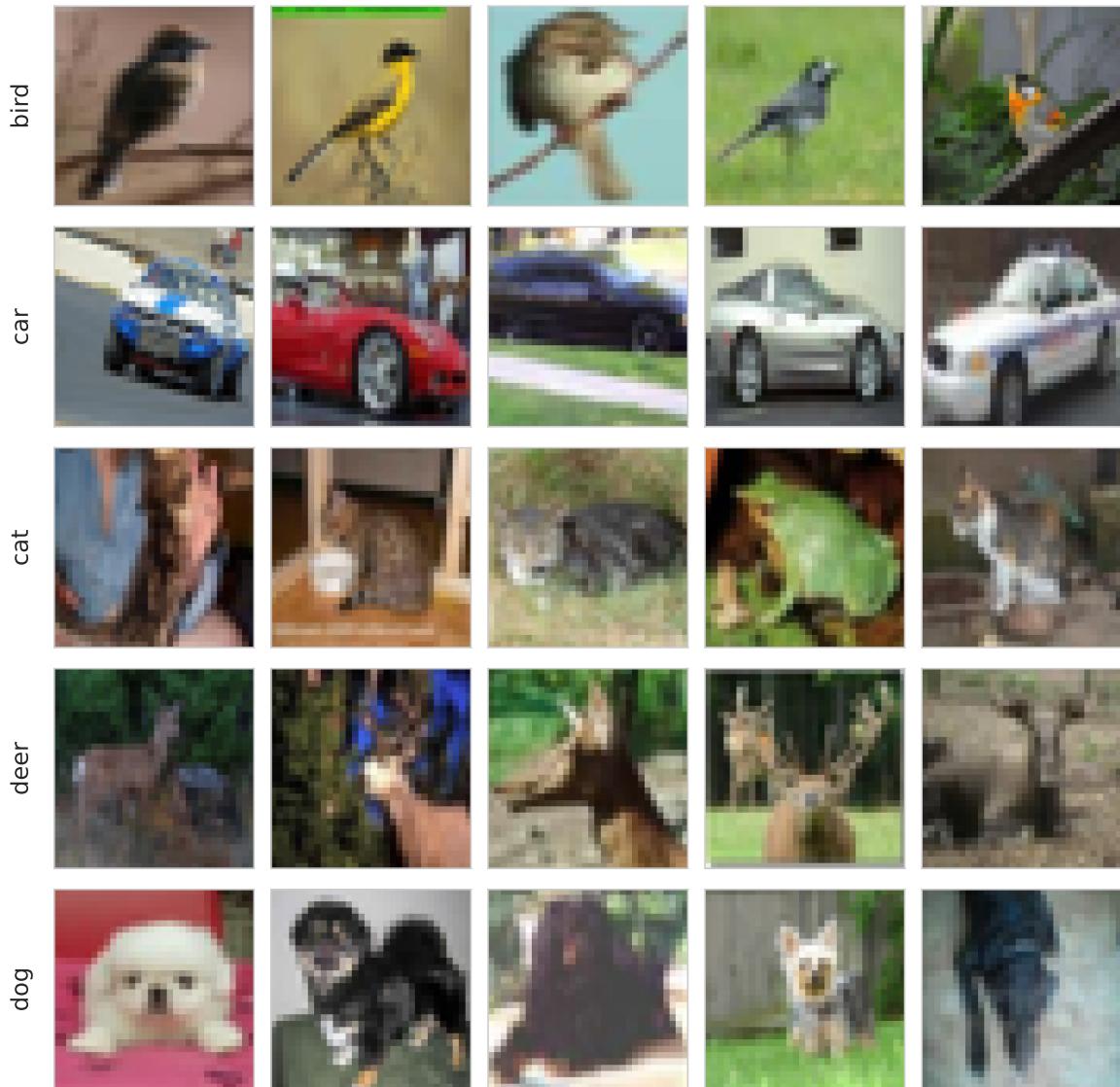


Figure 16: Example of images from CIFAR-10H. We display images row-wise according to the true label given initially in CIFAR-10.

utx.figure\_4()

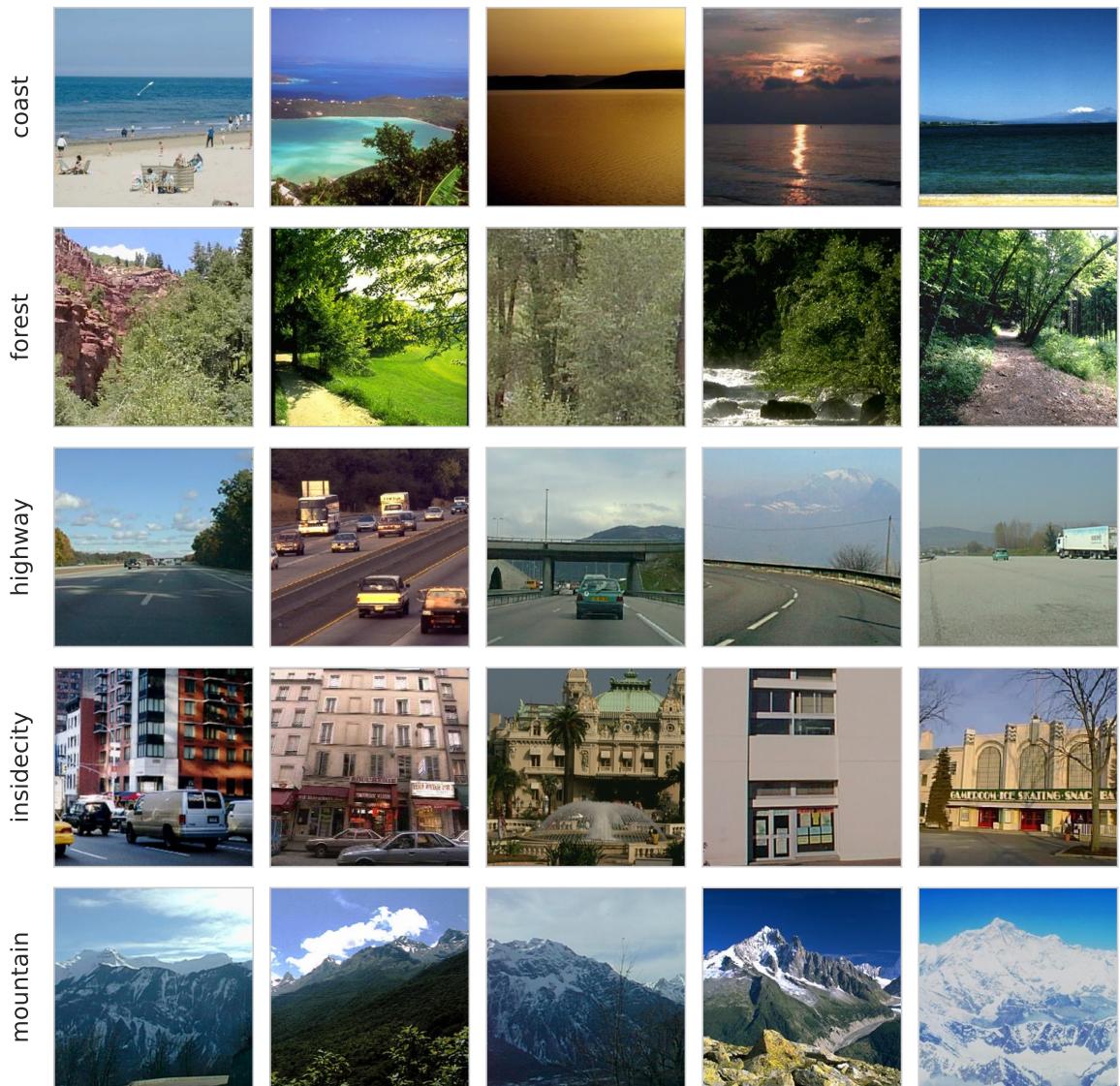


Figure 17: Example of images from LabelMe. We display images row-wise according to the true label given with the crowdsourced data.

- 541 Aitchison, L. 2021. “A Statistical Theory of Cold Posteriors in Deep Neural Networks.” In *ICLR*.  
 542 Cao, P, Y Xu, Y Kong, and Y Wang. 2019. “Max-MIG: An Information Theoretic Approach for Joint  
   543 Learning from Crowds.” In *ICLR*.  
 544 Chagneux, M, S LeCorff, P Gloaguen, C Ollion, O Lepâtre, and A Brûge. 2023. “Macrolitter Video  
   545 Counting on Riverbanks Using State Space Models and Moving Cameras.” *Computo*, February.  
   546 <https://computo.sfds.asso.fr/published-202301-chagneux-macrolitter>.  
 547 Chu, Z, J Ma, and H Wang. 2021. “Learning from Crowds by Modeling Common Confusions.” In  
   548 *AAAI*, 5832–40.  
 549 Dawid, AP, and AM Skene. 1979. “Maximum Likelihood Estimation of Observer Error-Rates Using  
   550 the EM Algorithm.” *J. R. Stat. Soc. Ser. C. Appl. Stat.* 28 (1): 20–28.  
 551 Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. “ImageNet: A Large-Scale Hierarchical  
   552 Image Database.” In *CVPR*.

- 553 Gao, G, and D Zhou. 2013. “Minimax Optimal Convergence Rates for Estimating Ground Truth from  
554 Crowdsourced Labels.” *arXiv Preprint arXiv:1310.5764*.
- 555 Garcin, C., A. Joly, P. Bonnet, A. Affouard, J.-C. Lombardo, M. Chouet, M. Servajean, T. Lorieul, and  
556 J. Salmon. 2021. “Pl@ntNet-300K: A Plant Image Dataset with High Label Ambiguity and a  
557 Long-Tailed Distribution.” In *Proceedings of the Neural Information Processing Systems Track on  
558 Datasets and Benchmarks*.
- 559 Gruber, S G, and F Buettner. 2022. “Better Uncertainty Calibration via Proper Scores for Classification  
560 and Beyond.” In *Advances in Neural Information Processing Systems*.
- 561 Guo, C, G Pleiss, Y Sun, and KQ Weinberger. 2017. “On Calibration of Modern Neural Networks.” In  
562 *ICML*, 1321.
- 563 Imamura, H, I Sato, and M Sugiyama. 2018. “Analysis of Minimax Error Rate for Crowdsourcing and  
564 Its Application to Worker Clustering Model.” In *ICML*, 2147–56.
- 565 James, GM. 1998. “Majority Vote Classifiers: Theory and Applications.” PhD thesis, Stanford  
566 University.
- 567 Kasmi, G, Y-M Saint-Drenan, D Trebosc, R Jolivet, J Leloux, B Sarr, and L Dubus. 2023. “A Crowd-  
568 sourced Dataset of Aerial Images with Annotated Solar Photovoltaic Arrays and Installation  
569 Metadata.” *Scientific Data* 10 (1): 59.
- 570 Khattak, FK. 2017. “Toward a Robust and Universal Crowd Labeling Framework.” PhD thesis,  
571 Columbia University.
- 572 Krizhevsky, A, and G Hinton. 2009. “Learning Multiple Layers of Features from Tiny Images.”  
573 University of Toronto.
- 574 Lefort, T, B Charlier, A Joly, and J Salmon. 2022. “Identify Ambiguous Tasks Combining Crowdsourced  
575 Labels by Weighting Areas Under the Margin.” *arXiv Preprint arXiv:2209.15380*.
- 576 Lin, Tsung-Yi, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays,  
577 Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. “Microsoft COCO:  
578 Common Objects in Context.” *CoRR* abs/1405.0312. <http://arxiv.org/abs/1405.0312>.
- 579 Marcel, S, and Y Rodriguez. 2010. “Torchvision the Machine-Vision Package of Torch.” In *Proceedings  
580 of the 18th ACM International Conference on Multimedia*, 1485–88. MM ’10. New York, NY, USA:  
581 Association for Computing Machinery.
- 582 Moreau, Thomas, Mathurin Massias, Alexandre Gramfort, Pierre Ablin, Pierre-Antoine Bannier,  
583 Benjamin Charlier, Mathieu Dagréou, et al. 2022. “BenchOpt: Reproducible, Efficient and Collab-  
584 orative Optimization Benchmarks.” In *NeurIPS*. <https://arxiv.org/abs/2206.13424>.
- 585 Park, Seo Yeon, and Cornelia Caragea. 2022. “On the Calibration of Pre-Trained Language Models  
586 Using Mixup Guided by Area Under the Margin and Saliency.” In *ACML*, 5364–74.
- 587 Passonneau, R J., and B Carpenter. 2014. “The Benefits of a Model of Annotation.” *Transactions of the  
588 Association for Computational Linguistics* 2: 311–26.
- 589 Paszke, A, S Gross, F Massa, A Lerer, J Bradbury, G Chanan, T Killeen, et al. 2019. “PyTorch: An  
590 Imperative Style, High-Performance Deep Learning Library.” In *NeurIPS*, 8024–35.
- 591 Peterson, J C., R M. Battleday, T L. Griffiths, and O Russakovsky. 2019. “Human Uncertainty Makes  
592 Classification More Robust.” In *ICCV*, 9617–26.
- 593 Pleiss, G, T Zhang, E R Elenberg, and K Q Weinberger. 2020. “Identifying Mislabeled Data Using the  
594 Area Under the Margin Ranking.” In *NeurIPS*.
- 595 Raykar, V C, and S Yu. 2011. “Ranking Annotators for Crowdsourced Labeling Tasks.” In *NeurIPS*,  
596 1809–17.
- 597 Rodrigues, F, M Lourenco, B Ribeiro, and F C Pereira. 2017. “Learning Supervised Topic Models for  
598 Classification and Regression from Crowds.” *IEEE Transactions on Pattern Analysis and Machine  
599 Intelligence* 39 (12): 2409–22.
- 600 Rodrigues, F, and F Pereira. 2018. “Deep Learning from Crowds.” In *AAAI*. Vol. 32.
- 601 Rodrigues, F, F Pereira, and B Ribeiro. 2014. “Gaussian Process Classification and Active Learning  
602 with Multiple Annotators.” In *ICML*, 433–41. PMLR.

- 603 Servajean, M, A Joly, D Shasha, J Champ, and E Pacitti. 2016. “ThePlantGame: Actively Training  
604 Human Annotators for Domain-Specific Crowdsourcing.” In *Proceedings of the 24th ACM In-*  
605 *ternational Conference on Multimedia*, 720–21. MM ’16. New York, NY, USA: Association for  
606 Computing Machinery.
- 607 ——. 2017. “Crowdsourcing Thousands of Specialized Labels: A Bayesian Active Training Approach.”  
608 *IEEE Transactions on Multimedia* 19 (6): 1376–91.
- 609 Sinha, V B, S Rao, and V N Balasubramanian. 2018. “Fast Dawid-Skene: A Fast Vote Aggregation  
610 Scheme for Sentiment Classification.” *arXiv Preprint arXiv:1803.02781*.
- 611 Tinati, R, M Luczak-Roesch, E Simperl, and W Hall. 2017. “An Investigation of Player Motivations in  
612 Eyewire, a Gamified Citizen Science Project.” *Computers in Human Behavior* 73: 527–40.
- 613 Ustalov, Dmitry, Nikita Pavlichenko, and Boris Tseitlin. 2023. “Learning from Crowds with Crowd-  
614 Kit.” arXiv. <https://arxiv.org/abs/2109.08584>.
- 615 Whitehill, J, T Wu, J Bergsma, J Movellan, and P Ruvolo. 2009. “Whose Vote Should Count More:  
616 Optimal Integration of Labels from Labelers of Unknown Expertise.” In *NeurIPS*. Vol. 22.
- 617 Yasmin, R, M Hassan, J T Grassel, H Bhogaraju, A R Escobedo, and O Fuentes. 2022. “Improving  
618 Crowdsourcing-Based Image Classification Through Expanded Input Elicitation and Machine  
619 Learning.” *Frontiers in Artificial Intelligence* 5: 848056.
- 620 Zhang, H, M Cissé, Y N. Dauphin, and D Lopez-Paz. 2018. “Mixup: Beyond Empirical Risk Minimiza-  
621 tion.” In *ICLR*.