

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: 2023-11-21 Last modified: 2023-11-21

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

¹Corresponding author: tanguy.lefort@umontpellier.fr

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

34 **1 Introduction: crowdsourcing in image classification**

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

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

- 43 1) **How to identify good workers in the crowd and difficult tasks?** When multiple answers
 44 are given to a single task, looking for who to trust for which type of task becomes necessary
 45 to estimate the labels or later train a model with as few noise sources as possible. The module
 46 `identify` uses different scoring metrics to create a worker and/or task evaluation. This is
 47 particularly relevant considering the gamification of crowdsourcing experiments (Servajean et
 48 al. 2016)
- 49 2) **How to aggregate multiple labels into a single label from crowdsourced tasks?** This
 50 occurs for example when dealing with a single dataset that has been labeled by multiple
 51 workers with disagreements. This is also encountered with other scoring issues such as polls,
 52 reviews, peer-grading, etc. In our framework this is treated with the `aggregate` command, that
 53 given multiple labels, infers a label. From aggregated labels, a classifier can then be trained
 54 using the `train` command.
- 55 3) **How to learn a classifier from crowdsourced datasets?** Where the second question is
 56 bound by aggregating multiple labels into a single one, this considers the case where we do
 57 not need a single label to train on, but instead train a classifier on the crowdsourced data,
 58 with the motivation to perform well on a testing set. This end-to-end vision, is common in
 59 machine learning, however, it requires the actual tasks (the images, texts, videos, etc.) to train
 60 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.

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

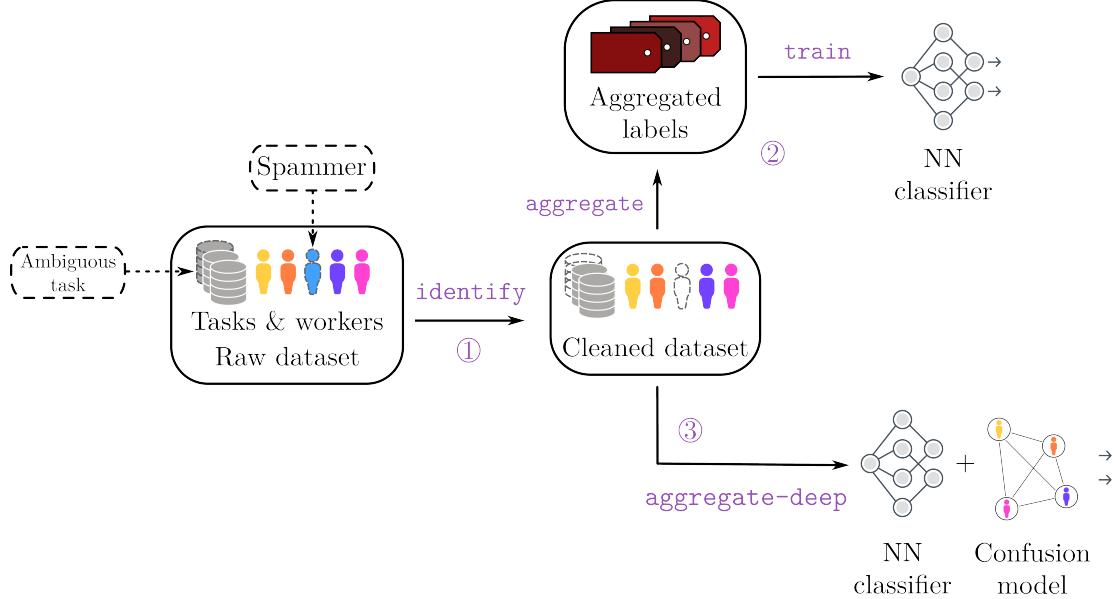


Figure 1: From crowdsourced labels to training a classifier neural network, the learning pipeline using the peerannot library. An optional preprocessing step using the `identify` command allows us to remove worse performing workers or images that can not be classified correctly (very bad quality for 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`.

2 Notation and package structure

2.1 Crowdsourcing notation

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

With crowdsourced data the true label of a task x_i , denoted y_i^* is unknown, and there is no single

82 label that can be trusted as in standard supervised learning (even on the train set!). Instead, there is a
 83 crowd of n_{worker} workers from which multiple workers $(w_j)_j$ propose a label $(y_i^{(j)})_j$. These proposed
 84 labels are used to estimate a true label. The set of workers answering the task x_i is denoted by

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

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

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

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

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

89 In this article, we do not address the setting where workers report their self-confidence (Yasmin et al.
 90 2022), nor settings where workers are presented a trapping set – *i.e* a subset of tasks where the true
 91 label is known to evaluate them with known labels (Khattak 2017).

92 2.2 Storing crowdsourced datasets in peerannot

93 Crowdsourced datasets come in various forms. To store [crowdsourcing datasets](#) efficiently and in a
 94 standardized way, peerannot proposes the following structure, where each dataset corresponds to a
 95 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
    class1
      imagename-<key>.png
      ...
      anotherimagename-<anotherkey>.png
      ...
    classK
  val
  test
  metadata.json
  answers.json
```

96 The `answers.json` file stores the different votes for each task as described in Figure 2. Thus, for
 97 example for an image named `smiley_face-1`, the associated labels are stored in the `answers.json`
 98 at the key numbered 1. This key identification system allows us to track directly from the filename
 99 the crowdsourced labels without having to rely on multiple indexing files as can be traditionally
 100 proposed. Furthermore, storing labels in a dictionary is more memory-friendly than having an array
 101 of size $(n_{\text{task}}, n_{\text{worker}})$ and writing $y_i^{(j)} = -1$ when the worker w_j did not see the task x_i and
 102 $y_i^{(j)} \in [K]$ 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` subdir as in [ImageFolder datasets](#) from `torchvision`

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

Left (peerannot storage):

	$\{ \textcolor{red}{\text{not smiling}} : 1, \textcolor{teal}{\text{neutral}} : 0, \textcolor{purple}{\text{neutral}} : 1, \textcolor{yellow}{\text{smiling}} : 1 \}$
	$\{ \textcolor{red}{\text{not smiling}} : 1, \textcolor{purple}{\text{neutral}} : 0, \textcolor{yellow}{\text{smiling}} : 0 \}$
	$\{ \textcolor{teal}{\text{neutral}} : 1, \textcolor{yellow}{\text{smiling}} : 1 \}$

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)

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:

```
{
    "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 in classical supervised learning
 114 form (without crowdsourcing). These labels are used as true labels in evaluations and visualizations.
 115 However, we emphasize that crowdsourcing strategies do not rely on the true labels (only on the
 116 workers' answers).

```
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_3()
```

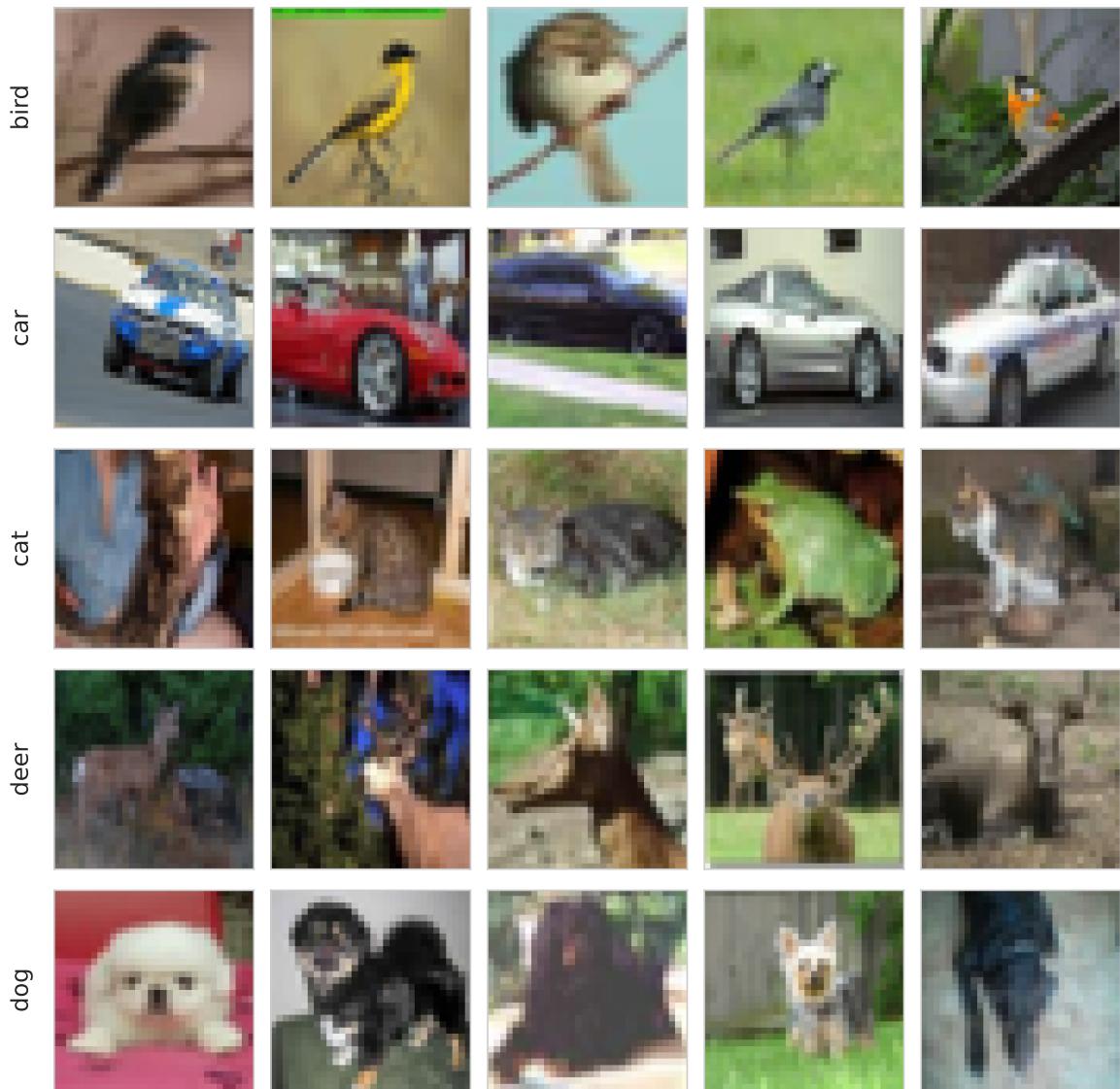


Figure 3: Example of images from CIFAR-10H. We display images rowise according to the true label given initially in CIFAR-10.

117 Examples of CIFAR-10H images are available in Figure 3, and LabelMe examples in Figure 4 here
118 below.

utx.figure_4()

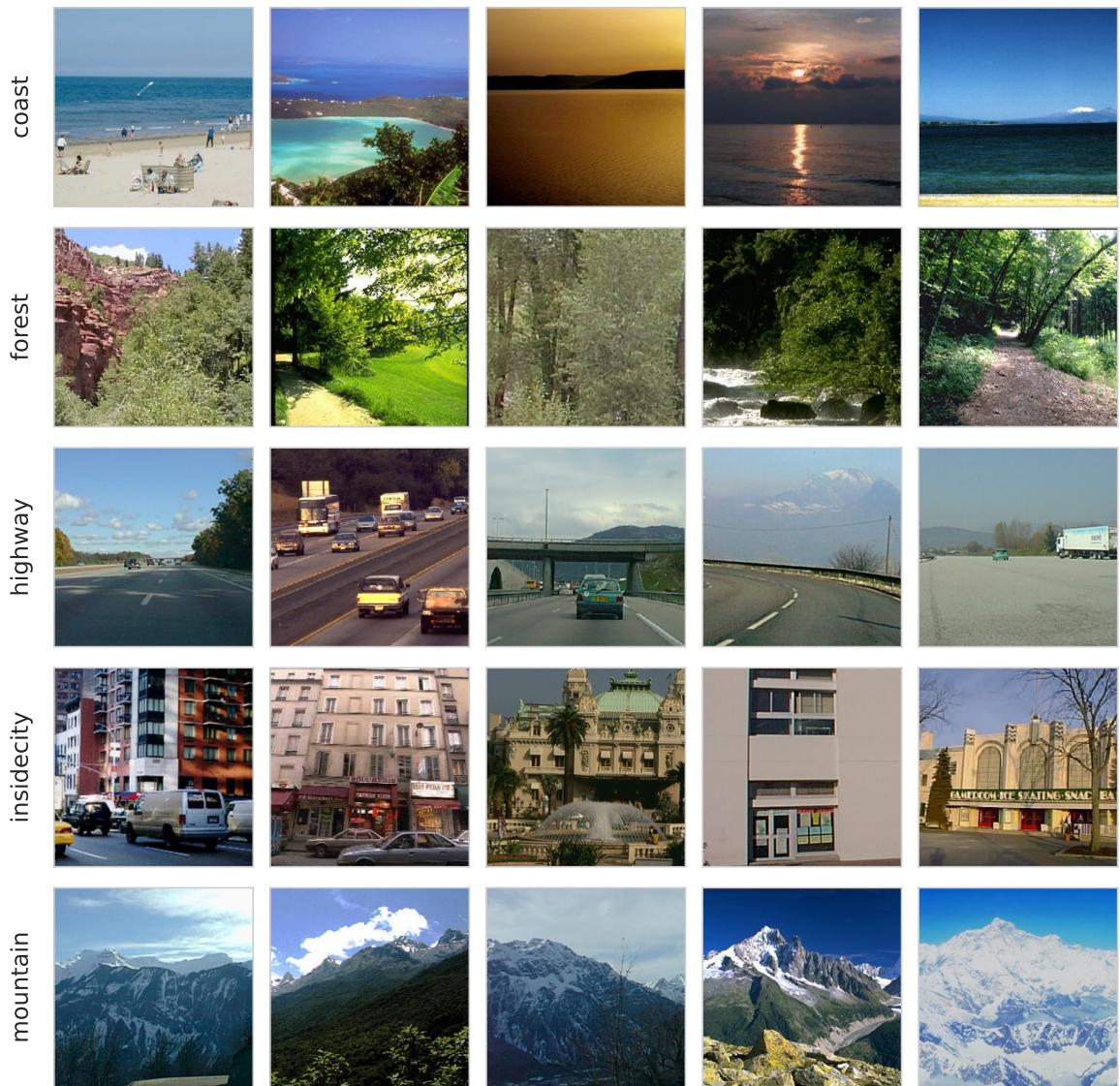


Figure 4: Example of images from LabelMe. We display images rowwise according to the true label given with the crowdsourced data.

¹¹⁹ Each of these tasks has been assigned a true label by the dataset's authors. Crowdsourcing votes
¹²⁰ however bring additional information about possible confusions (see Figure 5).

`utx.figure_5()`

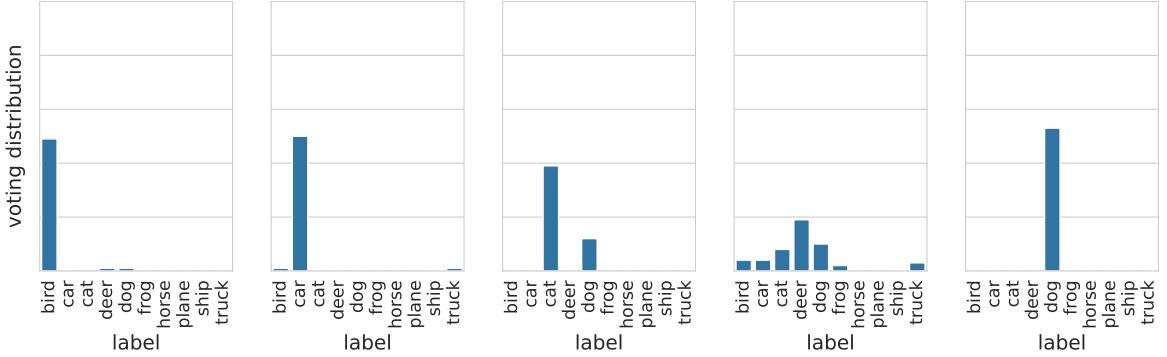
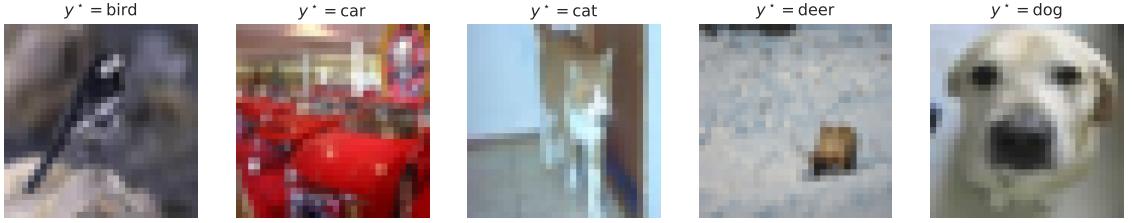


Figure 5: Example of crowdsourced images from CIFAR-10H. Each task has been labelled by multiple workers. We display the associated voting distribution over the possible classes. In addition, a true label is provided using the original CIFAR-10 dataset. This true label is only used for performance evaluation.

121 3 Aggregation strategies in crowdsourcing

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

- 126 • a **hard** label: \hat{y}_i is a Dirac distribution, this can be encoded as a classical label in $[K]$,
- 127 • a **soft** label: $\hat{y}_i \in \Delta_K$ can represent any probability distribution on $[K]$. In that case, each
 128 coordinate of the K -dimensional vector \hat{y}_i represents the probability to belong to the given
 129 class.

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

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

138 Moreover, the concept of confusion matrices has been commonly used to represent worker abilities.
 139 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
 140 quantities are not obtained using the true labels as they are unknown. In practice, the confusion
 141 matrices of each worker is estimated via an aggregation strategy like Dawid and Skene's (Dawid and
 142 Skene 1979) presented hereafter.

```

!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)

```

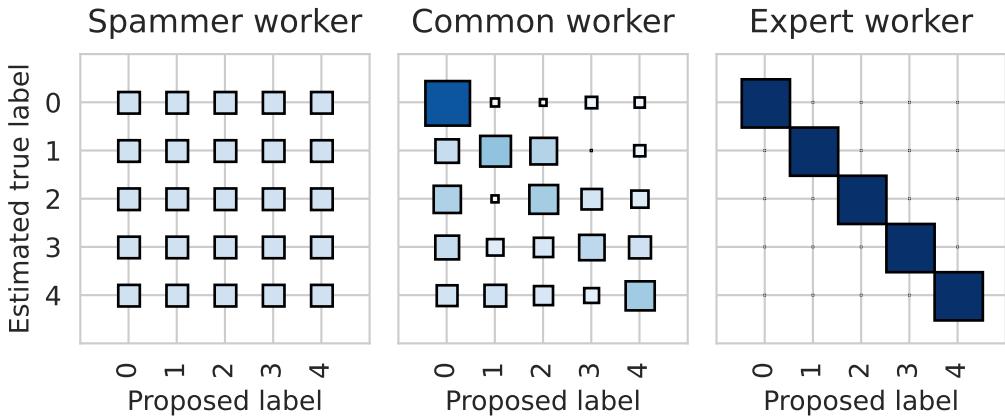


Figure 6: Three types of profiles of worker confusion matrices. The spammer answers independent from the true label. Expert workers identify classes without mistakes. In practice common workers are good for some classes but might confuse two (ore more) labels. All workers are simulated using the `peerannot simulate` command presented in Section 3.2.

143 In Figure 6, we illustrate multiple profiles of workers. In particular, one type of worker that can hurt
144 data quality is the spammer. Raykar and Yu (2011) defined a spammer as a worker that answers
145 randomly as:

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

146 As the probability distribution by row represent the confusion given an estimated true label, the
147 spammer has a confusion matrix with near-identical rows. Apart from the spammer, common
148 mistakes often involve workers ming-up one or several classes. Expert workers have a confusion
149 matrix near the identity matrix.

150 3.1 Classical models

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

152 3.1.1 Majority vote (MV)

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

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

157 3.1.2 Naive soft (NS)

158 One pitfall with MV is that the label produced is hard, hence the ambiguity is discarded by construction.
 159 A simple remedy consists in using the [Naive Soft \(NS\)](#) labeling, *i.e.* output the empirical distribution
 160 as the task label:

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

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

164 3.1.3 Dawid and Skene (DS)

165 Refining the aggregation, researchers have proposed a noise model to take into account the workers'
 166 abilities. The [Dawid and Skene](#)'s (DS) model (Dawid and Skene 1979) is one of the most studied
 167 (Gao and Zhou 2013) and applied (Servajean et al. 2017; Rodrigues and Pereira 2018). These types of
 168 models are most often optimized using EM-based procedures. Assuming the workers are answering
 169 tasks independently, this model boils down to model pairwise confusions between each possible
 170 class. Each worker w_j is assigned a confusion matrix $\pi^{(j)} \in \mathbb{R}^{K \times K}$ as described in Section 3. The
 171 model assumes that for a task x_i , conditionally on the true label $y_i^* = k$ the label distribution of the
 172 worker's answer follows a multinomial distribution with probabilities $\pi_{k,\ell}^{(j)}$ for each worker. Each
 173 class has a prevalence $\rho_k = \mathbb{P}(y_i^* = k)$ to appear in the dataset. Using the independence between
 174 workers, we obtain the following likelihood to maximize (with latent variables $\rho, \pi = \{\pi^{(j)}\}_j$ and
 175 unobserved variables $(y_i^{(j)})_{i,j}$):

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

176 When the true labels are not available, the data comes from a mixture of categorical distributions.
 177 In order to retrieve ground truth labels and be able to estimate these parameters, Dawid and Skene
 178 (1979) propose to consider the true labels as missing parameters. In this case, denoting $T_{i,k} = \mathbf{1}_{\{y_i^* = k\}}$
 179 the vectors of label class indicators for each task, the likelihood with known true labels is:

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

180 This framework allows to estimate ρ, π, T with an EM algorithm as follows:

- 181 • With the MV strategy, get an initial estimate of the true labels T .
 182 • Estimate ρ and π knowing T using maximum likelihood estimators.
 183 • Update T knowing ρ and π using Bayes formula.
 184 • Repeat until convergence of the likelihood.

185 The final aggregated soft label is $\hat{y}_i^{\text{DS}} = T_{i,.}$

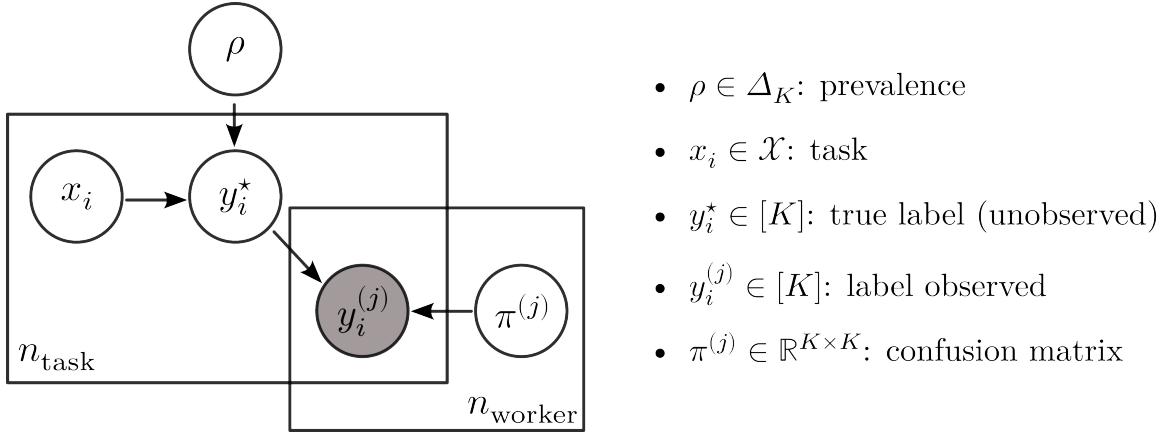


Figure 7: Bayesian plate notation for the DS model

186 3.1.4 Variations around the DS model

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

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

197 3.1.5 Generative model of Labels, Abilities, and Difficulties (GLAD)

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

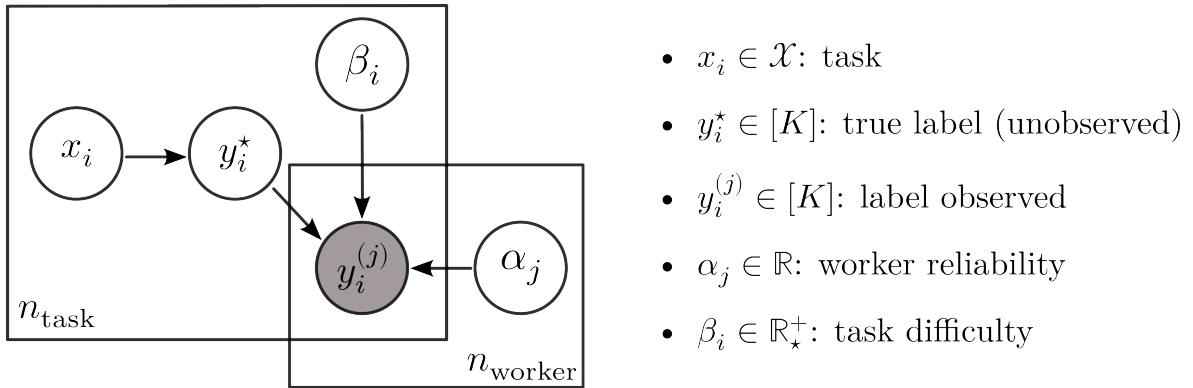


Figure 8: Bayesian plate notation for the GLAD model

201 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
 202 the easier), the model noise is:

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

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

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

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

205 3.1.6 Aggregation strategies in peerannot

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

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

212 3.2 Experiments and evaluation of label aggregation strategies

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

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

217 In the following, we write `AccTrain` for `AccTrain($\mathcal{D}_{\text{train}}$)` as we only consider the full training set so
 218 there is no ambiguity. While this metric is useful, in practice there are a few arguable issues:

- 219 • the `AccTrain` metric does not consider the ambiguity of the soft label, only the most probable
 class, whereas in some contexts ambiguity can be informative,
- 221 • in supervised learning one objective is to identify difficult or mislabeled tasks (Pleiss et al.
 222 2020; Lefort et al. 2022), pruning those tasks can easily artificially improve the `AccTrain`, but
 223 there is no guarantee over the predictive performance of a model based on the newly pruned
 224 dataset,
- 225 • in practice, true labels are unknown, thus this metric would not be computable.

226 We first consider classical simulation settings in the literature that can easily be created and repro-
 227 duced using `peerannot`. For each dataset, we present the distribution of the number of workers
 228 per task ($|\mathcal{A}(x_i)|_i$ [Equation 1](#) on the right and the distribution of the number of tasks per worker
 229 ($|\mathcal{T}(w_j)|_j$ [Equation 2](#) on the left).

230 3.2.1 Simulated independent mistakes

231 The independent mistakes setting considers that each worker w_j answers follows a multinomial
 232 distribution with weights given at the row y_i^* of their confusion matrix $\pi^{(j)} \in \mathbb{R}^{K \times K}$. Each confusion
 233 row in the confusion matrix is generated uniformly in the simplex. Then, we make the matrix
 234 diagonally dominant (to represent non-adversarial workers) by switching the diagonal term with
 235 the maximum value by row. Answers are independent of one another as each matrix is generated

236 independently and each worker answers independently of other workers. In this setting, the DS
 237 model is expected to perform better with enough data as we are simulating data from its assumed
 238 noise model.

239 We simulate $n_{\text{task}} = 200$ tasks and $n_{\text{worker}} = 30$ workers with $K = 5$ possible classes. Each task x_i
 240 receives $|\mathcal{A}(x_i)| = 10$ labels. With 200 tasks and 30 workers, asking for 10 leads to around $\frac{200 \times 10}{30} \approx 67$
 241 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)
workerload = working_load(votes_path, metadata_path)
feedback = feedback_effort(votes_path)
utx.figure_simulations(workerload, feedback)
plt.show()
```

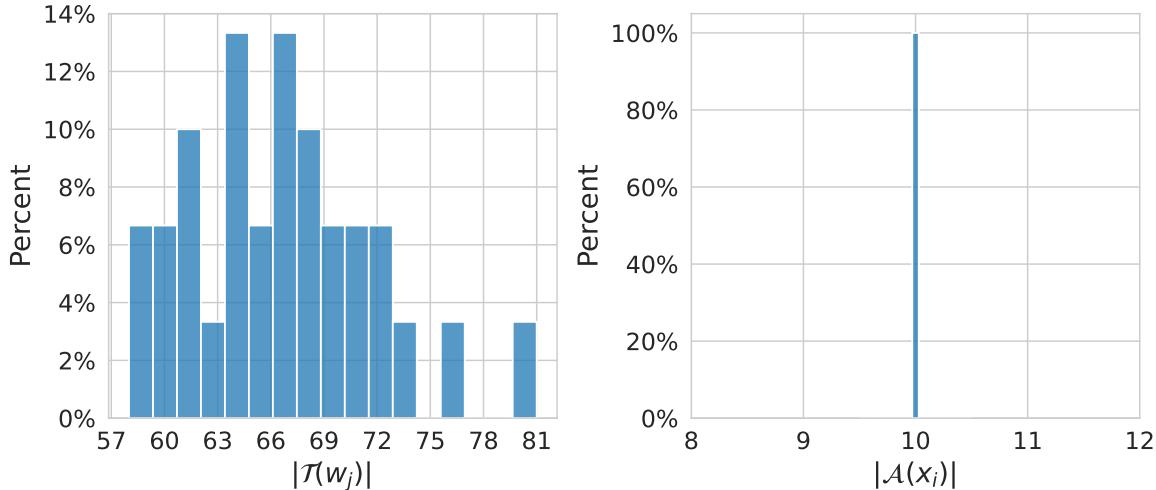


Figure 9: Distribution of number of tasks given per worker (left) and number of labels per task (right) in the independent mistakes setting.

242 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	0.760	0.775	0.890	0.775	0.770	0.760

243 As expected by the simulation framework, Table 1 fits the DS model, thus leading to better accuracy
 244 to retrieve the simulated labels for the DS strategy. The MV and NS aggregations do not consider
 245 any worker-ability scoring or the task's difficulty and performs the worse.

246 **Remark.** peerannot can also simulate datasets with an imbalanced number of votes chosen uniformly
 247 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)
workerload = working_load(votes_path, metadata_path)
feedback = feedback_effort(votes_path)
utx.figure_simulations(workerload, feedback)
plt.show()

```

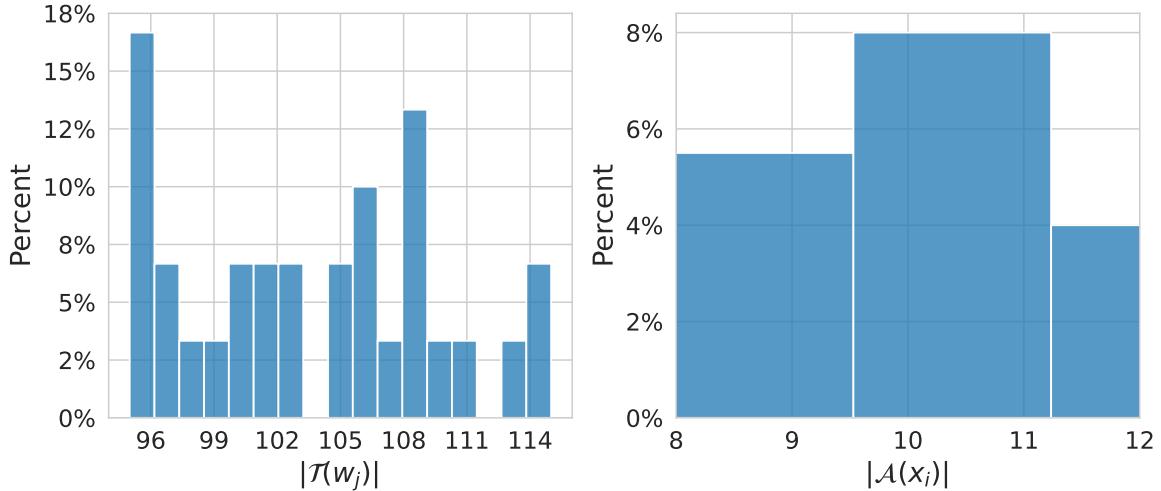


Figure 10: Distribution of number of tasks given per worker (left) and number of labels per task (right) in the independent mistakes setting with voting imbalance enabled.

²⁴⁸ 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[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	0.835	0.810	0.895	0.845	0.840	0.830

249 While more realistic, working with an imbalanced number of votes per task can lead to disrupting
 250 orders of performance for some strategies (here GLAD is downgraded with respect to other strategies).

251 3.2.2 Simulated correlated mistakes

252 The correlated mistakes are also known as the student-teacher or junior-expert setting (Cao et al.
 253 (2019)). Consider that the crowd of workers is divided into two categories: teachers and students
 254 (with $n_{\text{teacher}} + n_{\text{student}} = n_{\text{worker}}$). Each student is randomly assigned to one teacher at the beginning
 255 of the experiment. We generate the (diagonally dominant as in Section 3.2.1) confusion matrices of
 256 each teacher and the student share the same confusion matrix as their associated teacher. Hence,
 257 clustering strategies are expected to perform best in this context. Then, they all answer independently,
 258 following a multinomial distribution with weights given at the row y_i^* of their confusion matrix
 259 $\pi^{(j)} \in \mathbb{R}^{K \times K}$.

260 We simulate $n_{\text{task}} = 200$ tasks and $n_{\text{worker}} = 30$ with 80% of students in the crowd. There are $K = 5$
 261 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)
workerload = working_load(votes_path, metadata_path)
feedback = feedback_effort(votes_path)
utx.figure_simulations(workerload, feedback)
plt.show()

```

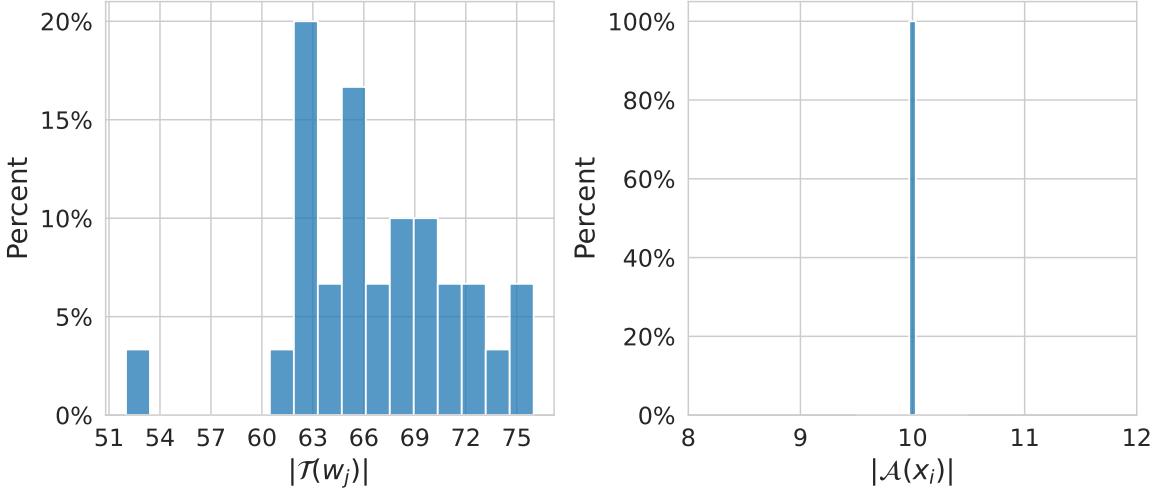


Figure 11: Distribution of number of tasks given per worker (left) and number of labels per task (right) in the correlated mistakes setting.

262 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	0.695	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. And 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`.

4 Learning from crowdsourced tasks

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

4.1 Popular models

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

4.1.1 CrowdLayer

`CrowdLayer` trains a classifier with noisy labels as follows. Let the scores (logits) output by a given 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 tensor of all $\pi^{(j)}$ ’s such that the crossentropy loss (CE) is adapted to the crowdsourcing setting into $\mathcal{L}_{CE}^{\text{CrowdLayer}}$ and computed as:

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

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

The confusion matrices of DS are taken into the network architecture as a new layer of weights to transform the output probabilities. The backbone classifier predicts a distribution that is then corrupted through the added layer to learn the worker-specific confusion.

292 **4.1.2 CoNAL**

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

297 Given the output $z_i = \mathcal{C}(x_i) \in \mathbb{R}^K$ of a given classifier and task, CoNAL interpolates between the
 298 prediction corrected by local confusions $\pi^{(j)} z_i$ and the prediction corrected by a global confusion
 299 $\pi^g z_i$. The loss function is computed as follows:

$$\mathcal{L}_{CE}^{\text{CoNAL}}(x_i) = \sum_{j \in \mathcal{A}(x_i)} \text{CE}(h_i^{(j)}, y_i^{(j)}) ,$$

with $h_i^{(j)} = \sigma((\omega_i^{(j)} \pi^g + (1 - \omega_i^{(j)}) \pi^{(j)}) z_i)$.

300 The interpolation weight $\omega_i^{(j)}$ is unobservable in practice. So, to compute $h_i^{(j)}$, the weight is obtained
 301 through an auxiliary network. This network takes as input the image and worker information
 302 and outputs a task-related vector v_i and a worker-related vector u_j of the same dimension. Finally,
 303 $w_i^{(j)} = (1 + \exp(-u_j^\top v_i))^{-1}$.

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

306 **4.2 Prediction error when learning from crowdsourced tasks**

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

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

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

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

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

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

325 **4.3 Use case with peerannot on real datasets**

326 Few real crowdsourcing experiments have been released publicly. Among the available ones,
327 CIFAR-10H (Peterson et al. 2019) is one of the largest with 10000 tasks labeled by workers (the
328 testing set of CIFAR-10). The main limitation of CIFAR-10H is that there are few disagreements
329 between workers and a simple majority voting already leads to a near-perfect AccTrain error. Hence,
330 comparing the impact of aggregation and end-to-end strategies might not be relevant (Peterson et al.
331 2019; Aitchison 2021), it is however a good benchmark for task difficulty identification and worker
332 evaluation scoring.

333 The LabelMe dataset was extracted from crowdsourcing segmentation experiments and a subset of
334 $K = 8$ classes was released in Rodrigues et al. (2017).

335 Let us use peerannot to train a VGG-16 with two dense layers on the LabelMe dataset. Note that
336 this modification was introduced to reach state-of-the-art performance in (Chu, Ma, and Wang 2021).
337 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
5	CrowdLayer	88.468	0.115
6	GLAD	88.889	0.112

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

342 5 Exploring crowdsourced datasets

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

346 However, mistakes and confusions happen during these experiments. Sometimes involuntarily
 347 (e.g. because the task is too hard or the worker is unable to differentiate between two classes) and

348 sometimes voluntarily (e.g. the worker is a spammer).

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

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

357 5.1 Exploring tasks' difficulty

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

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

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

- 362 • GLAD's scoring: by construction, Whitehill et al. (2009) introduced a scalar coefficient to score
363 the difficulty of a task.
364 • the Weighted Area Under the Margins (WAUM): introduced by Lefort et al. (2022), this weighted
365 area under the margins indicates how difficult it is for a classifier \mathcal{C} to learn a task's label. This
366 procedure is done with a budget of $T > 0$ epochs. Given the crowdsourced labels and the trust
367 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
368 crowdsourced labels $\{y_i^{(j)}\}_{j \in [K]} \in [K]^{\mathcal{A}(x_i)}$ is defined as:

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

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

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

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

377 Note that each of these statistics is useful in its context. The entropy can not be used in a setting
378 with small $|\mathcal{A}(x_i)|$ (few labels per task), in particular for the LabelMe dataset it is uninformative. The
379 WAUM can handle any number of labels, but the larger the better. However, as it uses a deep learning
380 classifier, the WAUM needs the tasks $(x_i)_i$ in addition to the proposed labels while the other strategies
381 are feature-blind.

382 **5.1.1 CIFAR-10H dataset**

383 First, let us consider a dataset with a large number of tasks, annotations and workers: the CIFAR-10H
384 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)

385 Unable to display output for mime type(s): text/html

386 Most difficult tasks identified depending on the strategy used (entropy, GLAD or WAUM) using a
387 Resnet34.

388 Unable to display output for mime type(s): text/html

389 The entropy, GLAD's difficulty, and WAUM's difficulty each show different images as exhibited in
390 the interactive Figure. We highlight that for the cat label, each strategy retrieves images that are
391 mislabeled in the true labeling. Indeed, the frog, dog and fox images are labeled as cat in CIFAR-10.
392 And while the entropy and GLAD output similar tasks, in this case the WAUM often differs. We can
393 also observe an ambiguity induced by the labels in the truck category, with the presence of a trailer
394 that is technically a mixup between a car and a truck.
```

395 **5.1.2 LabelMe dataset**

396 As for the LabelMe dataset, one difficulty in evaluating tasks' intrinsic difficulty is that there a limited
397 amount of votes available per task. Hence, the entropy in the distribution of the votes is no longer a
398 reliable metric, and we need to rely on other models.

399 Now, let us compare the tasks' difficulty distribution depending on the strategy considered using
400 `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

401 Unable to display output for mime type(s): text/html

402 Most difficult tasks identified depending on the strategy used (entropy, GLAD or WAUM) using a
403 VGG-16 with two dense layers.

404 Note that in this experiment, because the number of labels given per task is in {1, 2, 3}, the entropy
405 only takes four values. In particular, tasks with only one label all have a null entropy, so not just
406 consensual tasks.

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

410 5.2 Identification of worker reliability and task difficulty

411 From the labels we can explore different worker evaluation scores. GLAD's strategy estimates a
412 reliability scalar coefficient α_j per worker. With strategies looking to estimate confusion matrices,
413 we investigate two scoring rules for workers:

- 414 • The trace of the confusion matrix: the closer to K the better the worker.
- 415 • The closeness to spammer metric (Raykar and Yu 2011) (also called spammer score) that is the
416 Frobenius norm between the estimated confusion matrix $\hat{\pi}^{(j)}$ and the closest rank-1 matrix.
417 The further to zero the better the worker. On the contrary, the closer to zero, the more likely it
418 is the worker to be a spammer. This score separates spammers from common workers and
419 experts (with profiles as in Figure 6).

420 When the tasks are available, confusion-matrix-based deep learning models can also be used. We
421 thus add to the comparison the trace of the confusion matrices with CrowdLayer and CoNAL on
422 the LabelMe datasets. For CoNAL, we only consider the trace of the confusion matrix $\pi^{(j)}$ in the

423 pairwise comparison. Moreover, for CrowdLayer and CoNAL we show in Figure 13 the weights
424 learned without the softmax operation by row to keep the comparison as simple as possible with the
425 actual outputs of the model.

426 Comparisons in Figure 12 and Figure 13 are plotted pairwise between the evaluated metrics. Each
427 point represents a worker. Each off-diagonal plot shows the joint distribution between the scores of
428 the y-axis row and x-axis column. They allow us to visualize the relationship between these two
429 variables. The main diagonal represents the (smoothed) marginal distribution of the score of the
430 considered column.

431 5.2.1 CIFAR-10H

432 The CIFAR-10H dataset has few disagreements among workers. However, these strategies disagree
433 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()
```

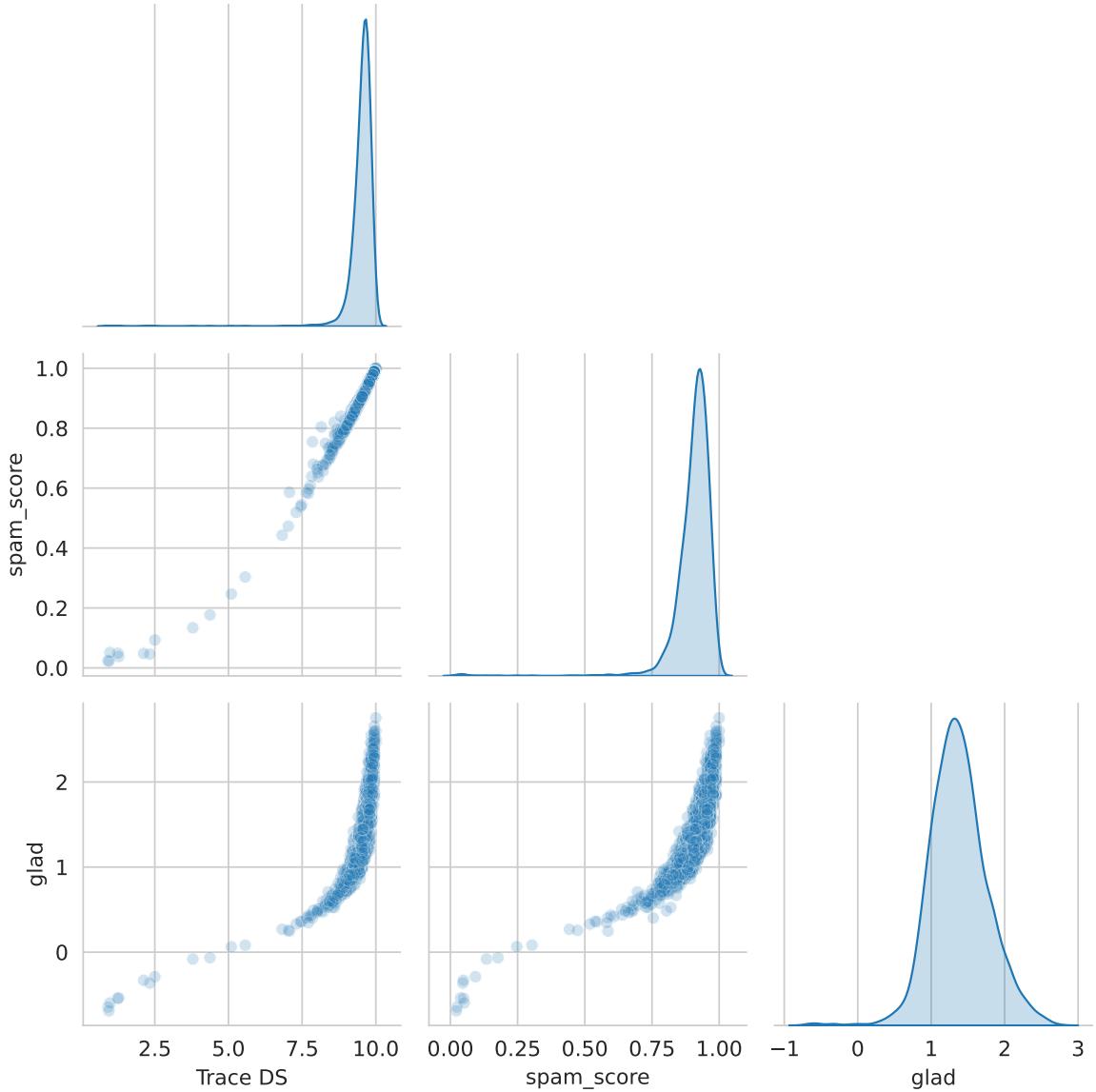


Figure 12: Comparison of ability scores by workers for the CIFAR-10H dataset. All metrics computed identify the same poorly performing workers. A mass of good and expert workers can be seen as the dataset presents few disagreements, thus few data to separate good from best workers.

434 From Figure 12, we can see that in this dataset, different methods easily separate the worse workers
 435 from the rest of the crowd (workers in the left tail of the distribution).

436 5.2.2 LabelMe

437 Finally, let us evaluate workers for the LabelMe dataset. Because of the lack of data (up to 3 labels
 438 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()

```

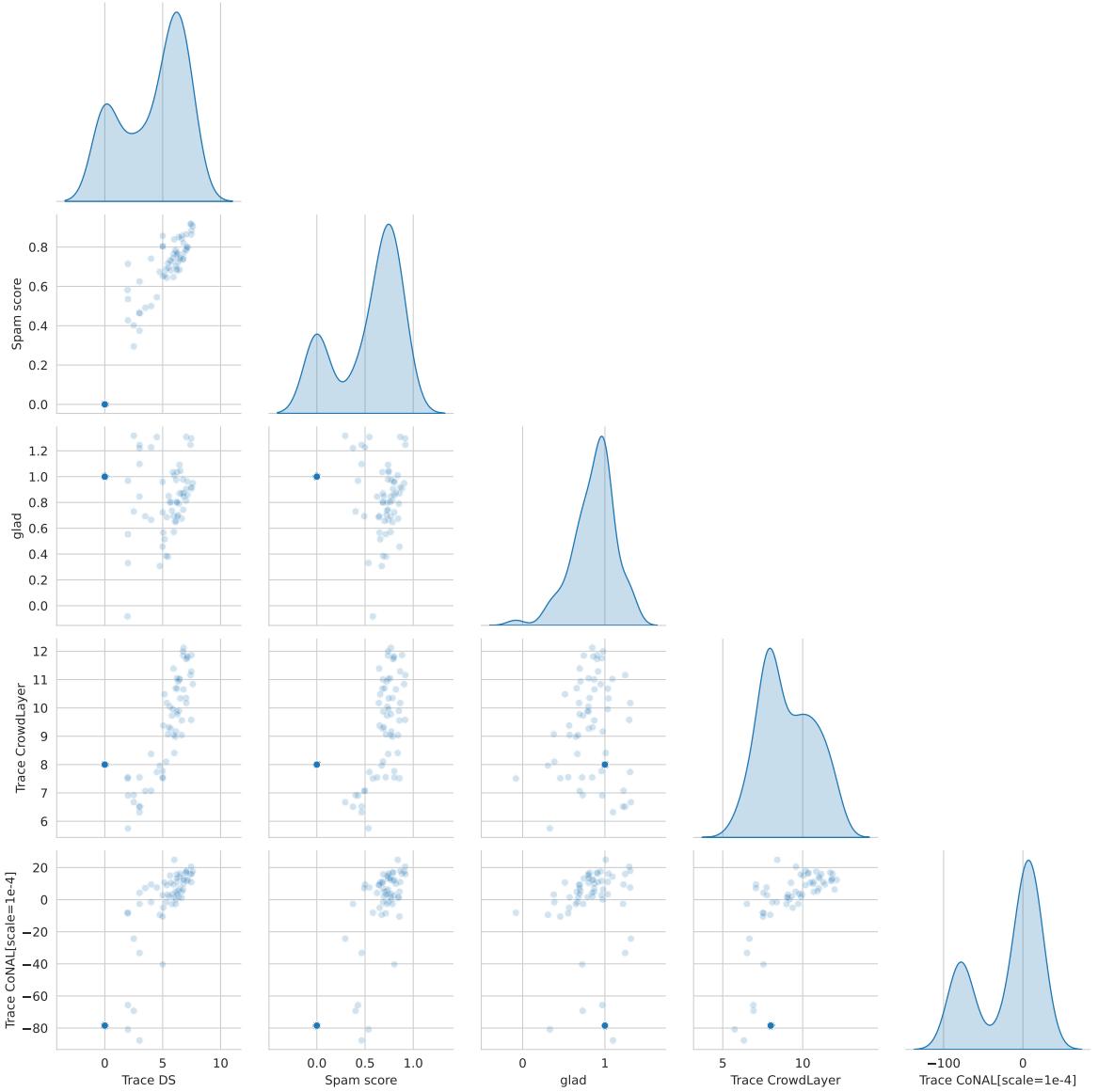


Figure 13: Comparison of ability scores by workers for the labelme dataset. With few labels per task, workers are more difficult to rank. It is more difficult to separate workers with their abilities in this crowd. Hence the importance of investigating the generalization performance of the methods presented in the previous section.

We can see in Figure 13 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 let 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 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, but also remove answers from spammers to improve the data quality as shown in Figure 1. The easy

450 access to the different strategies allows the user to decide if, for their collected dataset, there is a
 451 need for more recent deep-learning based strategies to improve the results. This is the case for the
 452 LabelMe dataset. Otherwise, the user can decide that standard aggregation-based crowdsourcing
 453 strategies are sufficient and for example, if there are plenty of votes per tasks like in CIFAR-10H,
 454 that the entropy in the votes distribution is a criterion that identified enough ambiguous tasks for
 455 their case. As often, not a single strategy works best for all datasets, hence the need to perform easy
 456 comparisons with peerannot.

457 6 Conclusion

458 We introduced peerannot, a library to handle crowdsourced datasets. This library enables both
 459 easy label aggregation and direct training strategies with classical state-of-the-art classifiers. The
 460 identification module of the library allows exploring the collected data from both the tasks and the
 461 workers' point of view for better scorings and data cleaning procedures. Our library also comes
 462 with templated datasets to better share crowdsourced datasets. Going beyond templating, it helps
 463 the crowdsourcing community to have openly accessible strategies to test, compare and improve in
 464 order to develop common strategies to analyse more and more common crowdsourced datasets.

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

470 7 Appendix

471 7.1 Supplementary simulation: Simulated mistakes with discrete difficulty levels 472 on tasks

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

$$478 \text{ratio-diff} = \frac{\mathbb{P}(\text{easy})}{\mathbb{P}(\text{hard})} \text{ with } \mathbb{P}(\text{easy}) + \mathbb{P}(\text{hard}) = 1 .$$

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

481 We simulate $n_{\text{task}} = 500$ tasks and $n_{\text{worker}} = 100$ with 35% of good workers in the crowd and 50% of
 482 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)
workerload = working_load(votes_path, metadata_path)
feedback = feedback_effort(votes_path)
utx.figure_simulations(workerload, feedback)
plt.show()

```

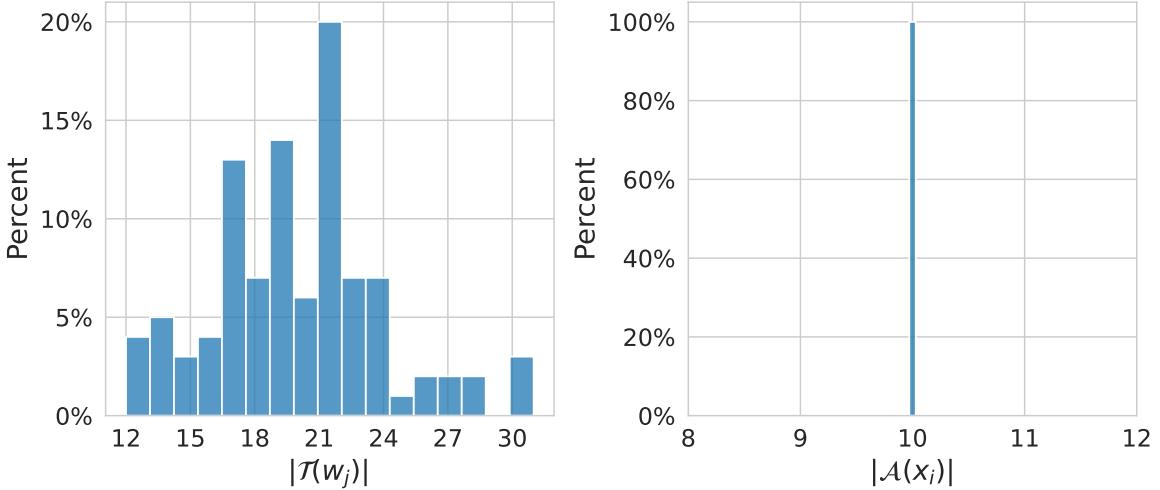


Figure 14: Distribution of number of tasks given per worker (left) and number of labels per task (right) in the setting with simulated discrete difficulty levels.

483 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 when tasks are associated a difficulty level considering classical feature-blind label aggregation strategies

Table 5

	MV	GLAD	DS	DSWC[L=2]	DSWC[L=5]	NS
AccTrain	0.810	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 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 problems benchmarks between multiple frameworks. Each strategy is run, we measure 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
```

516 First, let us see the performances on the [Bluebirds](#) dataset, a small dataset with 39 workers, 108 tasks
 517 and $K = 2$ classes.

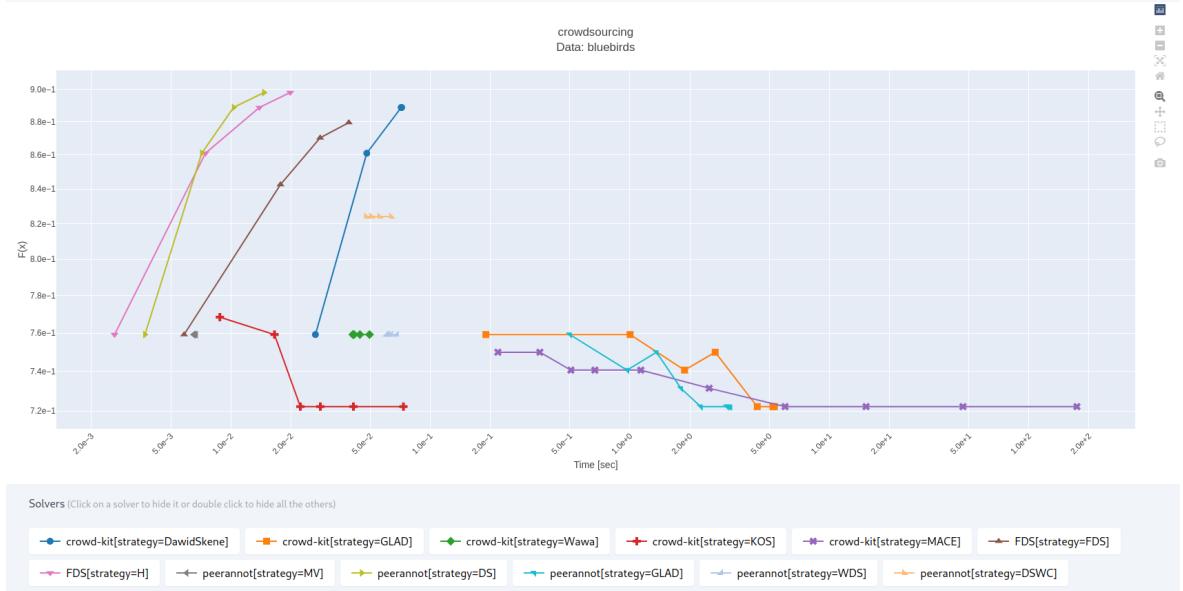


Figure 15: Aggregation strategies computational time during optimization procedure for the BlueBirds dataset with $K=2$.

518 We see in Figure 15 that the DS strategy from peerannot is the first to reach the optimum, followed
 519 by the [Fast-DS strategy](#) and then crowd-kit DS. Other strategies do not lead to better accuracy on
 520 this dataset, DS seem to be the best fitting strategy.

521 For the LabelMe dataset, DS strategy is also the best aggregation strategy, faster for crowd-kit. The
 522 sensitiviy of GLAD's method to the priors on α and β parameters can lead to large differences on real
 523 datasets performance as we see in Figure 16. Note that crowd-kit's KOS strategy is not available for
 524 this dataset as it is only made for binary classification datasets.

525 Aitchison, L. 2021. “A Statistical Theory of Cold Posteriors in Deep Neural Networks.” In *ICLR*.
 526 Cao, P, Y Xu, Y Kong, and Y Wang. 2019. “Max-MIG: An Information Theoretic Approach for Joint
 527 Learning from Crowds.” In *ICLR*.

528 Chagneux, M, S LeCorff, P Gloaguen, C Ollion, O Lepâtre, and A Brûge. 2023. “Macrolitter Video
 529 Counting on Riverbanks Using State Space Models and Moving Cameras.” *Computo*, February.
 530 <https://computo.sfds.asso.fr/published-202301-chagneux-macrolitter>.

531 Chu, Z, J Ma, and H Wang. 2021. “Learning from Crowds by Modeling Common Confusions.” In
 532 *AAAI*, 5832–40.

533 Dawid, AP, and AM Skene. 1979. “Maximum Likelihood Estimation of Observer Error-Rates Using
 534 the EM Algorithm.” *J. R. Stat. Soc. Ser. C. Appl. Stat.* 28 (1): 20–28.

535 Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. “ImageNet: A Large-Scale Hierarchical
 536 Image Database.” In *CVPR*.

537 Gao, G, and D Zhou. 2013. “Minimax Optimal Convergence Rates for Estimating Ground Truth from
 538 Crowdsourced Labels.” *arXiv Preprint arXiv:1310.5764*.

539 Garcin, C., A. Joly, P. Bonnet, A. Affouard, J.-C. Lombardo, M. Chouet, M. Servajean, T. Lorieul, and
 540 J. Salmon. 2021. “Pl@ntNet-300K: A Plant Image Dataset with High Label Ambiguity and a
 541 Long-Tailed Distribution.” In *Proceedings of the Neural Information Processing Systems Track on
 542 Datasets and Benchmarks*.

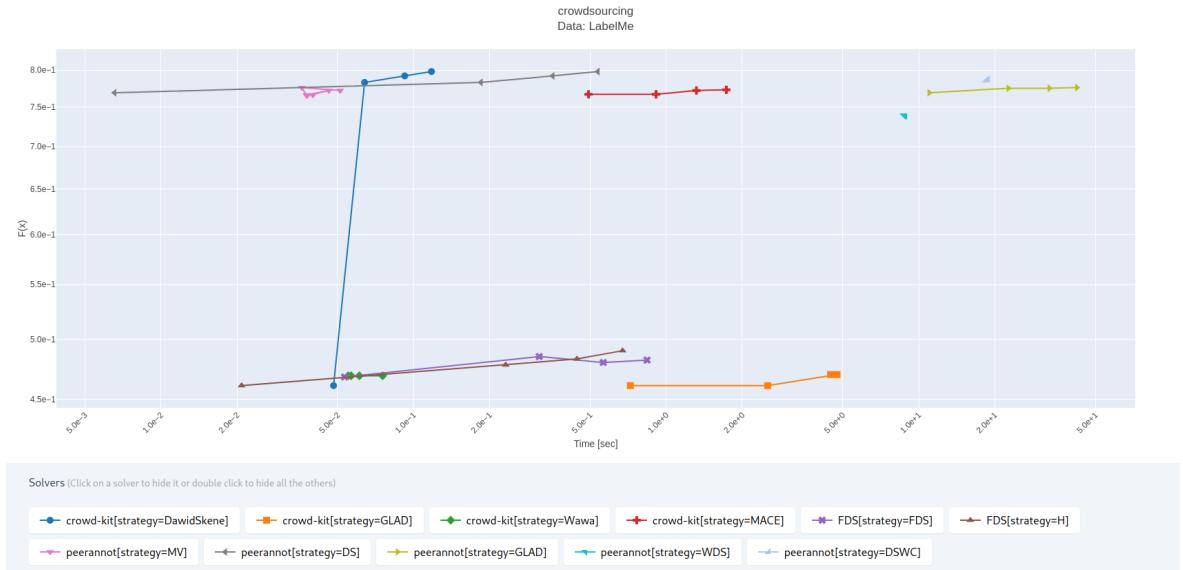


Figure 16: Aggregation strategies computational time during optimization procedure for the LabelMe dataset with K=8

- 543 Gruber, S G, and F Buettner. 2022. “Better Uncertainty Calibration via Proper Scores for Classification
544 and Beyond.” In *Advances in Neural Information Processing Systems*.
- 545 Guo, C, G Pleiss, Y Sun, and KQ Weinberger. 2017. “On Calibration of Modern Neural Networks.” In
546 *ICML*, 1321.
- 547 Immamura, H, I Sato, and M Sugiyama. 2018. “Analysis of Minimax Error Rate for Crowdsourcing and
548 Its Application to Worker Clustering Model.” In *ICML*, 2147–56.
- 549 James, GM. 1998. “Majority Vote Classifiers: Theory and Applications.” PhD thesis, Stanford
550 University.
- 551 Kasmi, G, Y-M Saint-Drenan, D Trebosc, R Jolivet, J Leloux, B Sarr, and L Dubus. 2023. “A Crowd-
552 sourced Dataset of Aerial Images with Annotated Solar Photovoltaic Arrays and Installation
553 Metadata.” *Scientific Data* 10 (1): 59.
- 554 Khattak, FK. 2017. “Toward a Robust and Universal Crowd Labeling Framework.” PhD thesis,
555 Columbia University.
- 556 Krizhevsky, A, and G Hinton. 2009. “Learning Multiple Layers of Features from Tiny Images.”
557 University of Toronto.
- 558 Lefort, T, B Charlier, A Joly, and J Salmon. 2022. “Identify Ambiguous Tasks Combining Crowdsourced
559 Labels by Weighting Areas Under the Margin.” *arXiv Preprint arXiv:2209.15380*.
- 560 Lin, Tsung-Yi, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays,
561 Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. “Microsoft COCO:
562 Common Objects in Context.” *CoRR* abs/1405.0312. <http://arxiv.org/abs/1405.0312>.
- 563 Marcel, S, and Y Rodriguez. 2010. “Torchvision the Machine-Vision Package of Torch.” In *Proceedings
564 of the 18th ACM International Conference on Multimedia*, 1485–88. MM ’10. New York, NY, USA:
565 Association for Computing Machinery.
- 566 Moreau, Thomas, Mathurin Massias, Alexandre Gramfort, Pierre Ablin, Pierre-Antoine Bannier,
567 Benjamin Charlier, Mathieu Dagréou, et al. 2022. “Benchopt: Reproducible, Efficient and Collab-
568 orative Optimization Benchmarks.” In *NeurIPS*. <https://arxiv.org/abs/2206.13424>.
- 569 Park, Seo Yeon, and Cornelia Caragea. 2022. “On the Calibration of Pre-Trained Language Models
570 Using Mixup Guided by Area Under the Margin and Saliency.” In *ACML*, 5364–74.
- 571 Passonneau, R J., and B Carpenter. 2014. “The Benefits of a Model of Annotation.” *Transactions of the*

- 572 Association for Computational Linguistics 2: 311–26.
- 573 Paszke, A, S Gross, F Massa, A Lerer, J Bradbury, G Chanan, T Killeen, et al. 2019. “PyTorch: An
574 Imperative Style, High-Performance Deep Learning Library.” In *NeurIPS*, 8024–35.
- 575 Peterson, J C., R M. Battleday, T L. Griffiths, and O Russakovsky. 2019. “Human Uncertainty Makes
576 Classification More Robust.” In *ICCV*, 9617–26.
- 577 Pleiss, G, T Zhang, E R Elenberg, and K Q Weinberger. 2020. “Identifying Mislabeled Data Using the
578 Area Under the Margin Ranking.” In *NeurIPS*.
- 579 Raykar, V C, and S Yu. 2011. “Ranking Annotators for Crowdsourced Labeling Tasks.” In *NeurIPS*,
580 1809–17.
- 581 Rodrigues, F, M Lourenco, B Ribeiro, and F C Pereira. 2017. “Learning Supervised Topic Models for
582 Classification and Regression from Crowds.” *IEEE Transactions on Pattern Analysis and Machine
583 Intelligence* 39 (12): 2409–22.
- 584 Rodrigues, F, and F Pereira. 2018. “Deep Learning from Crowds.” In *AAAI*. Vol. 32.
- 585 Rodrigues, F, F Pereira, and B Ribeiro. 2014. “Gaussian Process Classification and Active Learning
586 with Multiple Annotators.” In *ICML*, 433–41. PMLR.
- 587 Servajean, M, A Joly, D Shasha, J Champ, and E Pacitti. 2016. “ThePlantGame: Actively Training
588 Human Annotators for Domain-Specific Crowdsourcing.” In *Proceedings of the 24th ACM Inter-
589 national Conference on Multimedia*, 720–21. MM ’16. New York, NY, USA: Association for
590 Computing Machinery.
- 591 ———. 2017. “Crowdsourcing Thousands of Specialized Labels: A Bayesian Active Training Approach.”
592 *IEEE Transactions on Multimedia* 19 (6): 1376–91.
- 593 Sinha, V B, S Rao, and V N Balasubramanian. 2018. “Fast Dawid-Skene: A Fast Vote Aggregation
594 Scheme for Sentiment Classification.” *arXiv Preprint arXiv:1803.02781*.
- 595 Tinati, R, M Luczak-Roesch, E Simperl, and W Hall. 2017. “An Investigation of Player Motivations in
596 Eyewire, a Gamified Citizen Science Project.” *Computers in Human Behavior* 73: 527–40.
- 597 Ustalov, Dmitry, Nikita Pavlichenko, and Boris Tseitlin. 2023. “Learning from Crowds with Crowd-
598 Kit.” arXiv. <https://arxiv.org/abs/2109.08584>.
- 599 Whitehill, J, T Wu, J Bergsma, J Movellan, and P Ruvolo. 2009. “Whose Vote Should Count More:
600 Optimal Integration of Labels from Labelers of Unknown Expertise.” In *NeurIPS*. Vol. 22.
- 601 Yasmin, R, M Hassan, J T Grassel, H Bhogaraju, A R Escobedo, and O Fuentes. 2022. “Improving
602 Crowdsourcing-Based Image Classification Through Expanded Input Elicitation and Machine
603 Learning.” *Frontiers in Artificial Intelligence* 5: 848056.
- 604 Zhang, H, M Cissé, Y N. Dauphin, and D Lopez-Paz. 2018. “Mixup: Beyond Empirical Risk Minimiza-
605 tion.” In *ICLR*.