COMPUTO

ISSN 2824-7795

Peerannot: learning from crowdsourced image datasets with Python

Tanguy Lefort [1] 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: 2023-07-18 Last modified: 2023-07-18

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. 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

Contents

[1]Corresponding author: tanguy.lefort@umontpellier.fr

1 Introduction: crowdsourcing in image classification

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

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

1) *How to identify good workers in the crowd?* When multiple answers are given to a single task, looking for who to trust for which type of task becomes necessary to estimate the ground truth or later train a model with as few noise sources as possible. The module `identify` uses different scoring metrics to create a worker and/or task evaluation. This is particularly relevant considering the gamification of crowdsourcing experiments (Servajean et al. 2016)
2) *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, that given multiple labels, infers a ground truth label. From aggregated labels, a classifier can then be trained using the `train` command.
3) *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.

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.

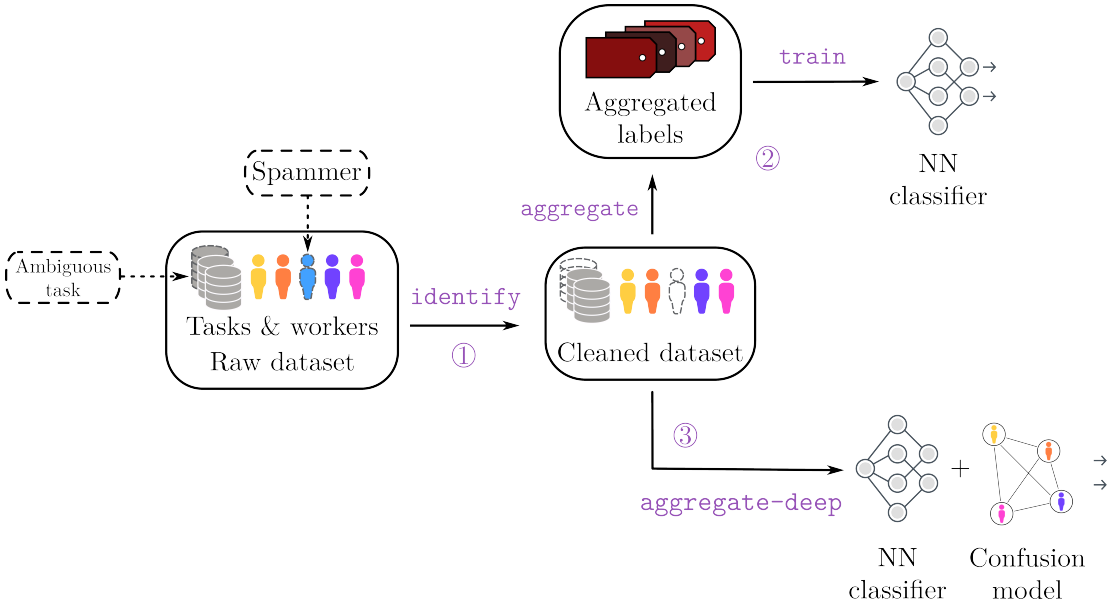


Figure 1: From crowdsourced labels to training a classifier neural network, the learning pipeline using the `peerannot` library. An optional preprocessing step using the `identify` command allows us to remove worse performing workers or images that can not be classified correctly (very bad quality for 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 $\mathscr{D} = \{(x_i, y_i^\star)\}_{i=1}^{n_{\text{task}}}$ is composed of n_{task} tasks $x_i \in \mathscr{X}$ (the feature space) with (unobserved) ground truth label $y_i^\star \in [K] = \{1, \dots, K\}$ 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. 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 addressed that, as this is a `Python` library, in the code indices start at the 0 position.

With crowdsourced data the ground truth of a task x_i, denoted $y_i^\star$ is unknown, and there is no single label that can be trusted as in standard supervised learning (even on the train set!). Instead, there is a crowd of n_{worker} workers from which multiple workers $(w_j)_j$ propose a label $(y_i^{(j)})_j$. The set of workers answering the task x_i is denoted by

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

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

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

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

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

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

2.2 Storing crowdsourced datasets in `peerannot`

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

```
{#lst-datasetconvention bash lst-cap="Dataset storage template"} datasetname
   train                class1                    imagename-<key>.png
   ...                    anotherimagename-<anotherkey>.png                ...
         classK          val          test          metadata.json          answers.json
```

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

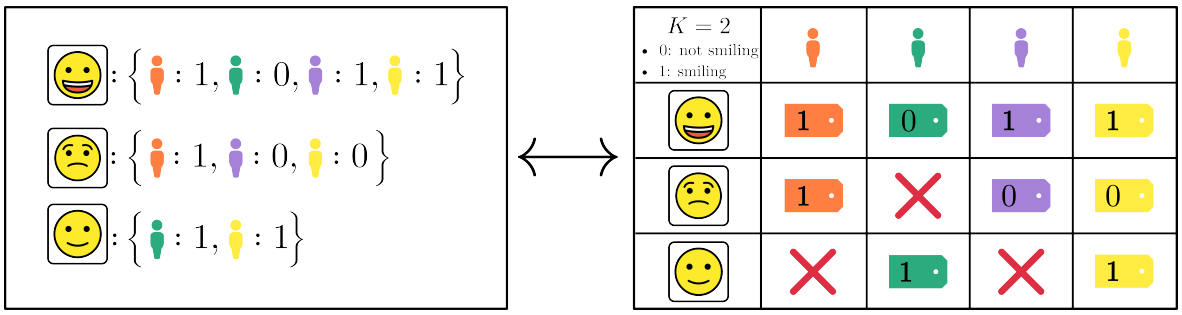


Figure 2: Data storage for a crowdsourced dataset for a binary classification problem ($K = 2$, smiling/smiling) on recognizing smiling faces. (left: how data is stored in `peerannot` in a file `answers.json`, right: data collected)

In Figure 2, there are three tasks, $n_{\text{worker}} = 4$ workers and $K = 2$ classes. Any available task should be stored in a single file whose name follows the convention described in **?@lst-datasetconvention**. These files are spread into a `train`, `val` and `test` subdir as in `ImageFolder` datasets from `torchvision`

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

```
{
    "name": "toy-data",
    "n_classes": 2,
```

```
    "n_workers": 4,
    "n_tasks": 3
}
```

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

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

For both `CIFAR-10H` and `LabelMe`, the dataset was originally released in classical supervised learning form (without crowdsourcing). These labels are used as ground truth in evaluations and visualizations. However, we emphasize that crowdsourcing strategies do not rely on the ground truth (only on the workers' answers).

```
import torch
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from pathlib import Path
nrow = 5
ncol = 5
fig, axs = plt.subplots(
        nrow,
        ncol,
        sharey="row",
        sharex="col",
        figsize=(12,8)
    )
match_ = {0: "bird", 1: "car", 2: "cat", 3: "deer", 4: "dog", 5: "frog", 6: "horse", 7: "plane",
path = Path.cwd() / "datasets" / "cifar10H" / "train"
for i in range(nrow):
  img_folder = path / f"{match_[i]}"
  all_imgs = list(img_folder.glob("*"))[:ncol]
  for j in range(ncol):
    image = np.asarray(Image.open(path / all_imgs[j]))
    axs[i,j].imshow(image, aspect="equal")
    axs[i,j].axis("off")
    axs[i,j].set_yticklabels([])
plt.subplots_adjust(wspace=-0.8, hspace=0.25)
plt.show()
```

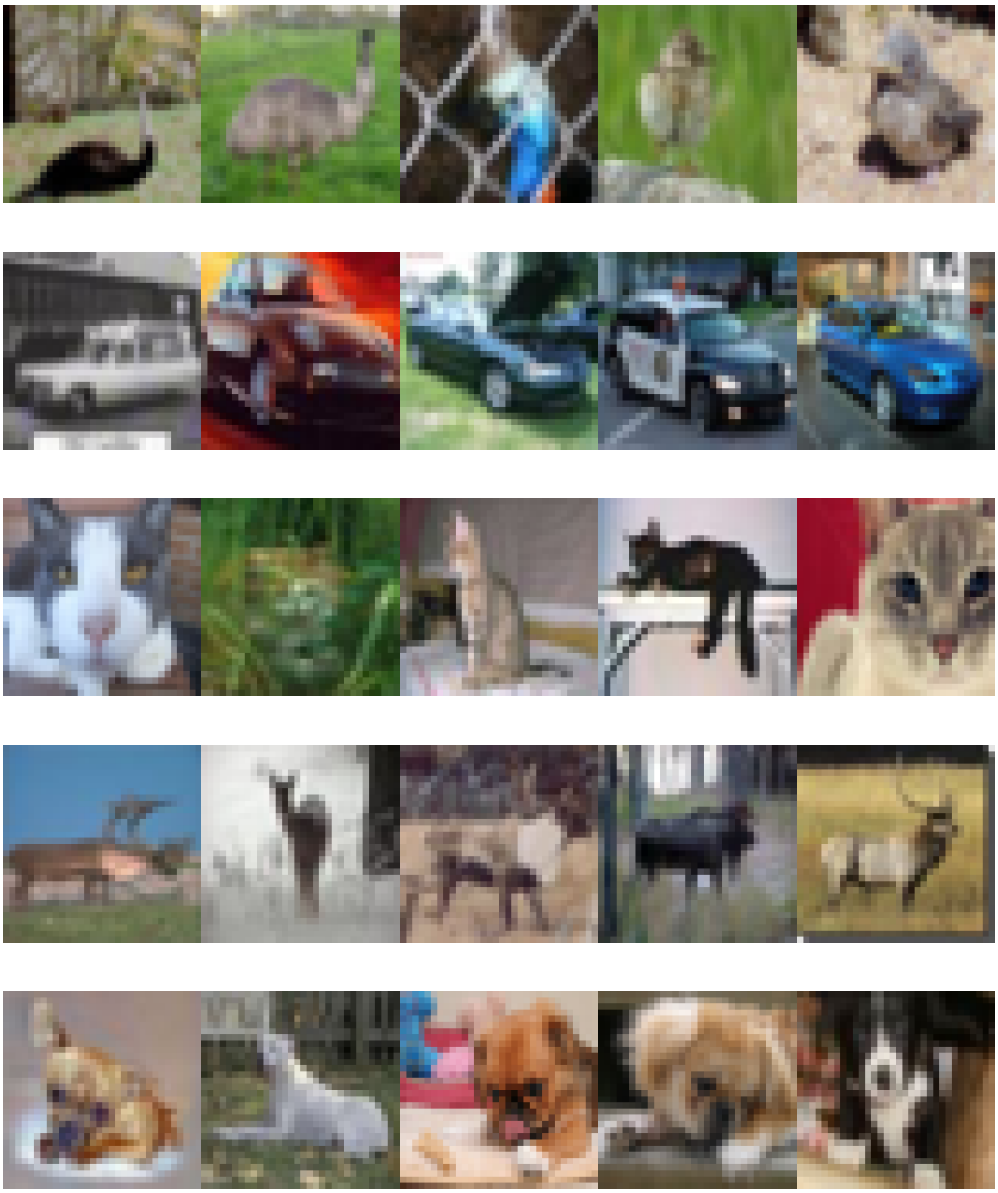


Figure 3: Example of images from CIFAR-10H. We display images rowise according to the ground truth label.

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

```
nrow = 5
ncol = 5
fig, axs = plt.subplots(
        nrow,
        ncol,
        sharey="row",
        sharex="col",
        figsize=(12,8)
```

```
    )
match_ = {0: "coast", 1: "forest", 2: "highway", 3: "insidecity", 4: "mountain", 5: "opencountry
path = Path.cwd() / "datasets" / "labelme" / "train"
for i in range(nrow):
  img_folder = path / f"{match_[i]}"
  all_imgs = list(img_folder.glob("*"))[:ncol]
  for j in range(ncol):
    image = np.asarray(Image.open(path / all_imgs[j]))
    axs[i,j].imshow(image, aspect="equal")
    axs[i,j].axis("off")
    axs[i,j].set_yticklabels([])
plt.subplots_adjust(wspace=-0.8, hspace=0.25)
plt.show()
```

Figure 4: Example of images from LabelMe. We display images rowise according to the ground truth label.

3 Aggregation strategies in crowdsourcing

The first question we address with `peerannot` is: *How to aggregate multiple labels into a single label from crowdsourced tasks?* The aggregation step can lead to two types of learnable labels $\hat{y}_i \in \Delta_K$ (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 the use case for each task $x_i, i = 1, \dots, n_{\text{task}}$:

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

Learning from soft labels has been shown to improve learning performance and make the classifier

learn the task ambiguity (Zhang et al. 2018; Peterson et al. 2019; Park and Caragea 2022). However, crowdsourcing is often used as a stepping stone to create a new dataset. We usually expect a classification dataset to associate a task x_i to a single label and not a full probability distribution. In this case, we recommend to release the anonymous answered labels and the aggregation strategy used to reach a consensus on a single label. With `peerannot`, both soft and hard labels can be produced.

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

3.1 Classical models

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

3.1.1 Majority vote (MV)

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

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

3.1.2 Naive soft (NS)

One pitfall with MV is that the label produced is hard, hence the ambiguity is discarded by construction. To remedy this, the Naive Soft (NS) labeling consists in using the empirical distribution as the task label:

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

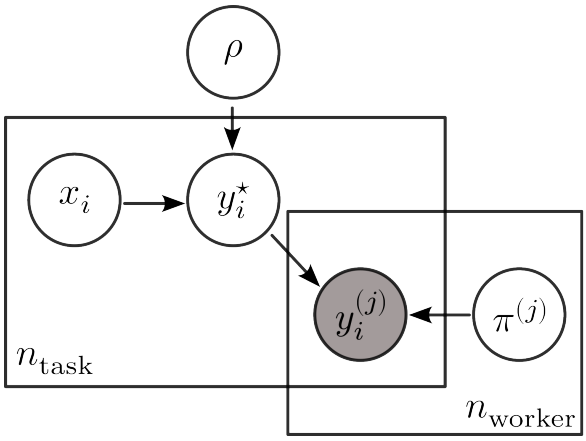
With the NS label, we keep the ambiguity, but all workers and all tasks are put on the same level. In practice, it is known that each worker comes with their abilities, thus modeling this knowledge can produce better results.

3.1.3 Dawid and Skene (DS)

Refining the aggregation, researchers began creating a noise model to take into account the workers' abilities. These types of models are most often EM-based and one of the most studied (Gao and Zhou 2013) and applied (Servajean et al. 2017; Rodrigues and Pereira 2018) is the Dawid and Skene's (DS) model (Dawid and Skene 1979). Assuming the workers are answering tasks independently, this model boils down to model pairwise confusions between each possible class. Each worker w_j is assigned a confusion matrix $\pi^{(j)} \in \mathbb{R}^{K\times K}$ such that $\pi^{(j)}_{k,\ell} = \mathbb{P}(y_i^{(j)} = \ell | y_i^\star = k)$. The model assumes that for a task x_i, conditionally on the true label $y_i^\star = k$ the label distribution of the worker's answer follows a multinomial distribution with probabilities $\pi^{(j)}_{k,\bullet}$ for each worker. Each class has a prevalence $\rho_k = \mathbb{P}(y_i^\star = k)$ to appear in the dataset. Using the independence between workers, we obtain the following likelihood to maximize (using the EM algorithm):

$$\prod_{i\in[n_{\texttt{task}}]} \prod_{k\in[K]} \left[\rho_k \prod_{j\in[n_{\texttt{worker}}]} \prod_{k\in[K]} \left(\pi^{(j)}_{k,k}\right)^{\mathbf{1}_{\{y_i^{(j)}=k\}}} \right]^{T_{ik}},$$

with $T_{i,k} = \mathbf{1}_{\{y_i^\star = k\}}$. The final aggregated soft label is $\hat{y}_i^{\mathrm{DS}} = T_{i,\cdot}$.



- $\rho \in \Delta_K$: prevalence
- $x_i \in \mathcal{X}$: task
- $y_i^\star \in [K]$: true label (unobserved)
- $y_i^{(j)} \in [K]$: label observed
- $\pi^{(j)} \in \mathbb{R}^{K \times K}$: confusion matrix

Figure 5: Bayesian plate notation for the DS model

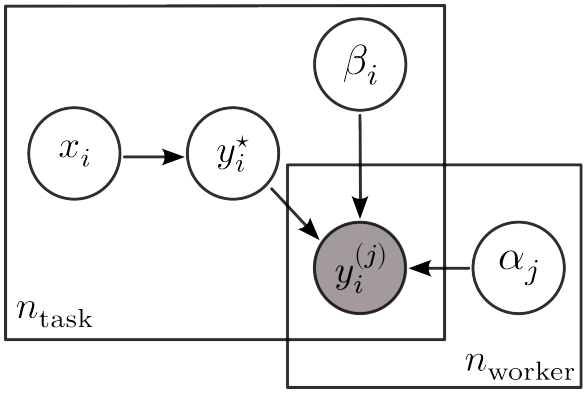
3.1.4 Variations around the DS model

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

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

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

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



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

Figure 6: Bayesian plate notation for the GLAD model

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

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

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

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

3.1.6 Aggregation strategies in `peerannot`

All of these aggregation strategies – and more – are available in the `peerannot` library from the `peerannot.models` module. Each model is a class object in its own `Python` file. It inherits from the `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_probas`: returns the soft labels output for each task.

3.2 Experiments and evaluation of label aggregation strategies

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

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

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

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

We first consider classical simulation settings in the literature that can easily be created and reproduced using `peerannot`. For each dataset, we present the distribution of the number of workers per task $(|\mathscr{A}(x_i)|)_i$ @Equation 1 on the right and the distribution of the number of tasks per worker $(|\mathscr{T}(w_j)|)_j$ @Equation 2 on the left.

3.2.1 Simulated independent mistakes simu-independent

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

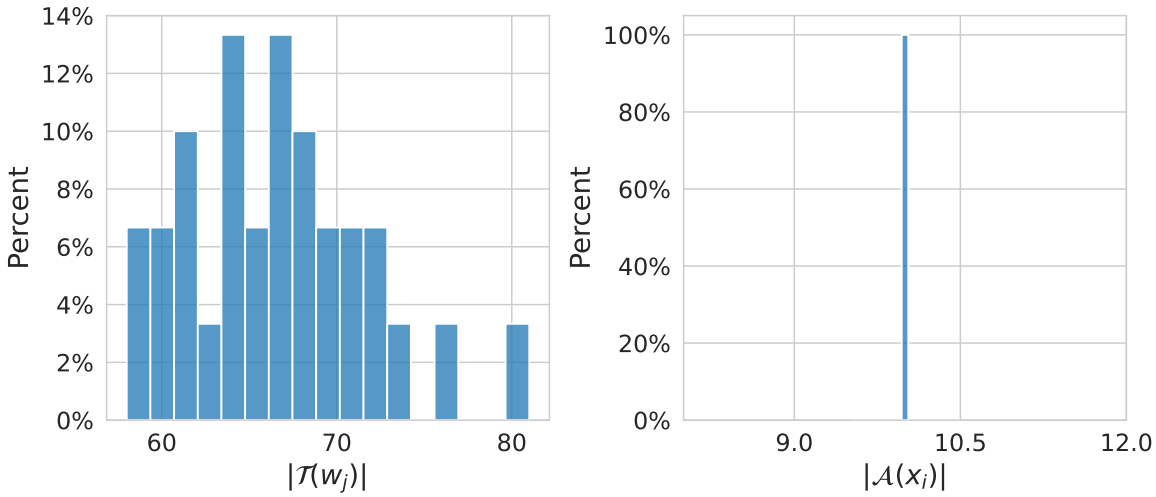
and each worker answers independently of other workers. In this setting, the DS model is expected to perform the best with enough data as we are simulating data from its assumed noise model.

We simulate $n_{\text{task}} = 200$ tasks and $n_{\text{worker}} = 30$ workers with $K = 5$ possible classes. Each task x_i receives $|\mathscr{A}(x_i)| = 10$ labels. With 200 tasks and 30 workers, asking for 10 leads to around $\frac{200\times10}{30} \simeq 67$ 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 seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
from pathlib import Path
import matplotlib.ticker as mtick
sns.set_style("whitegrid")

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)
nbins = 17
fig, ax = plt.subplots(1, 2, figsize=(9, 4))
sns.histplot(workerload, stat="percent", bins=nbins, shrink=1, ax=ax[0])
ax[0].yaxis.set_major_formatter(mtick.PercentFormatter(decimals=0))
ax[0].set_xlabel(r"$\vert\mathcal{T}(w_j)\vert$")
sns.histplot(feedback, stat="percent", bins=nbins, shrink=1, ax=ax[1])
ax[1].yaxis.set_major_formatter(mtick.PercentFormatter(decimals=0))
ax[1].set_xlabel(r"$\vert\mathcal{A}(x_i)\vert$")
ax[1].xaxis.set_major_locator(plt.MaxNLocator(3))
ax[1].set_xlim(8, 12)
ax[1].xaxis.set_major_locator(MaxNLocator(integer=True))
for i in range(2):
  ax[i].xaxis.set_major_locator(MaxNLocator(3))
  ax[i].xaxis.label.set_size(15)
  ax[i].yaxis.label.set_size(15)
  ax[i].xaxis.set_tick_params(labelsize=13)
  ax[i].yaxis.set_tick_params(labelsize=13)
  ax[i].title.set_size(18)
plt.tight_layout()
plt.show()
```



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 = 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

	MV	NAIVESOFT	GLAD	DS	DSWC[L=5]	DSWC[L=10]
AccTrain	0.770	0.760	0.780	0.890	0.775	0.770

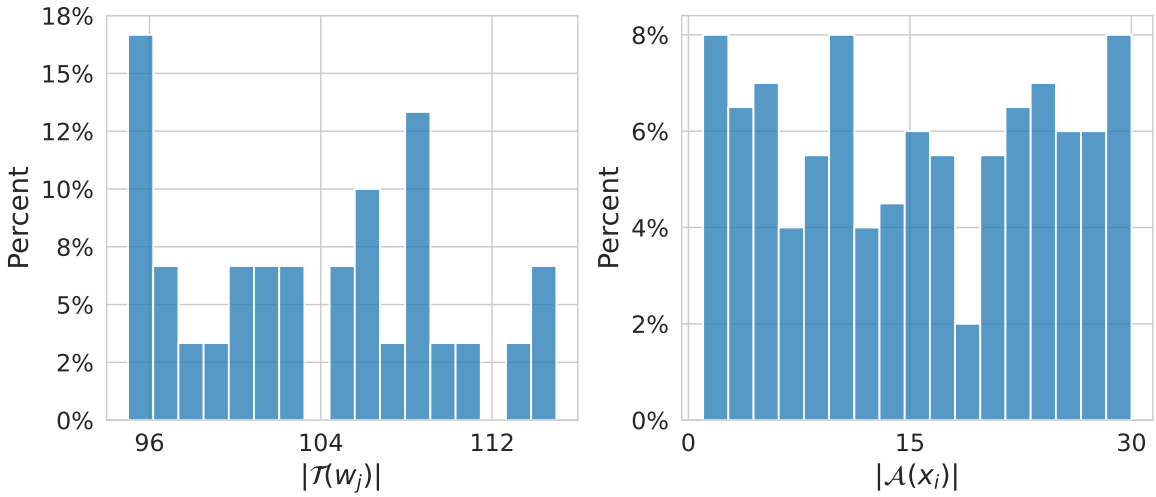
As expected by the simulation framework, Table 1 fits the DS model, thus leading to better accuracy to retrieve the simulated labels for the DS model. The MV aggregation does not consider any worker-ability scoring or the task's difficulty and performs the worse.

Remark. `peerannot` can also simulate datasets with an imbalanced number of votes chosen uniformly 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)
nbins = 17
fig, ax = plt.subplots(1, 2, figsize=(9, 4))
sns.histplot(workerload, stat="percent", bins=nbins, shrink=1, ax=ax[0])
ax[0].yaxis.set_major_formatter(mtick.PercentFormatter(decimals=0))
ax[0].set_xlabel(r"$\vert\mathcal{T}(w_j)\vert$")
sns.histplot(feedback, stat="percent", bins=nbins, shrink=1, ax=ax[1])
ax[1].yaxis.set_major_formatter(mtick.PercentFormatter(decimals=0))
ax[1].set_xlabel(r"$\vert\mathcal{A}(x_i)\vert$")
ax[1].xaxis.set_major_locator(plt.MaxNLocator(3))
ax[1].xaxis.set_major_locator(MaxNLocator(integer=True))
for i in range(2):
  ax[i].xaxis.set_major_locator(MaxNLocator(3))
  ax[i].xaxis.label.set_size(15)
  ax[i].yaxis.label.set_size(15)
  ax[i].xaxis.set_tick_params(labelsize=13)
  ax[i].yaxis.set_tick_params(labelsize=13)
  ax[i].title.set_size(18)
plt.tight_layout()
plt.show()
```



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 = 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

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

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

3.2.2 Simulated correlated mistakes

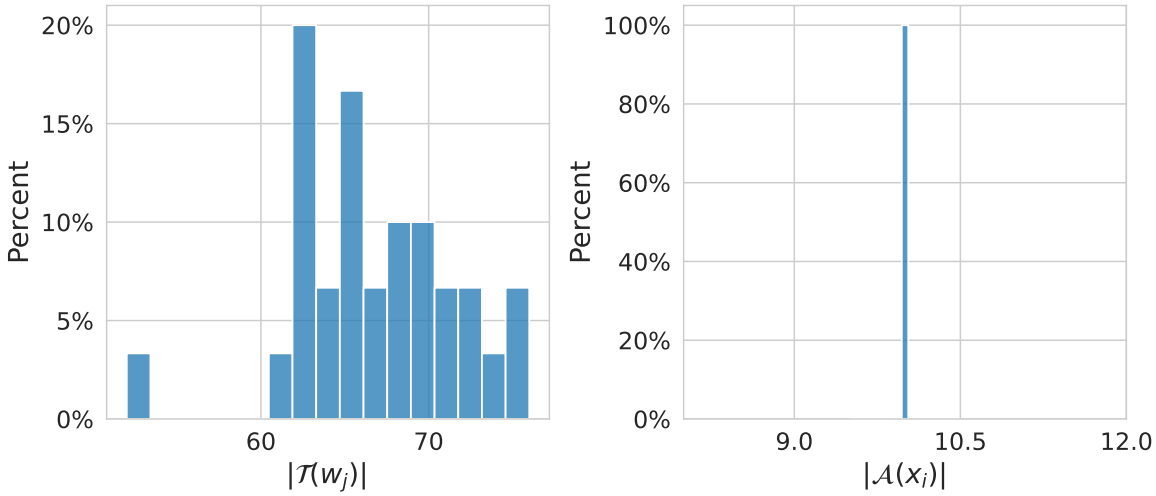
The correlated mistakes are also known as the student-teacher setting. Consider that the crowd of workers is divided into two categories: teachers and students (with $n_{\text{teacher}} + n_{\text{student}} = n_{\text{worker}}$). Each student is randomly assigned to one teacher at the beginning of the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

We simulate $n_{\text{task}} = 200$ tasks and $n_{\text{worker}} = 30$ with 80% of students in the crowd. There are $K = 5$ possible classes. Each task receives $|\mathscr{A}(x_i)| = 10$ 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)
nbins = 17
fig, ax = plt.subplots(1, 2, figsize=(9, 4))
sns.histplot(workerload, stat="percent", bins=nbins, shrink=1, ax=ax[0])
ax[0].yaxis.set_major_formatter(mtick.PercentFormatter(decimals=0))
ax[0].set_xlabel(r"$\vert\mathcal{T}(w_j)\vert$")
sns.histplot(feedback, stat="percent", bins=nbins, shrink=1, ax=ax[1])
ax[1].yaxis.set_major_formatter(mtick.PercentFormatter(decimals=0))
ax[1].set_xlabel(r"$\vert\mathcal{A}(x_i)\vert$")
ax[1].xaxis.set_major_locator(plt.MaxNLocator(3))
ax[1].set_xlim(8, 12)
ax[1].xaxis.set_major_locator(MaxNLocator(integer=True))
for i in range(2):
  ax[i].xaxis.set_major_locator(MaxNLocator(3))
  ax[i].xaxis.label.set_size(15)
  ax[i].yaxis.label.set_size(15)
  ax[i].xaxis.set_tick_params(labelsize=13)
```

```
  ax[i].yaxis.set_tick_params(labelsize=13)
  ax[i].title.set_size(18)
plt.tight_layout()
plt.show()
```



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/student_teacher/ -s {strat}
```

```
simu_corr = Path.cwd() / 'simus' / "student_teacher"
results = {"mv": [], "naivesoft": [], "glad": [], "ds": [], "dswc[l=5]": [], "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 = 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

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

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).

3.2.3 Simulated mistakes with discrete difficulty levels on tasks

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

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

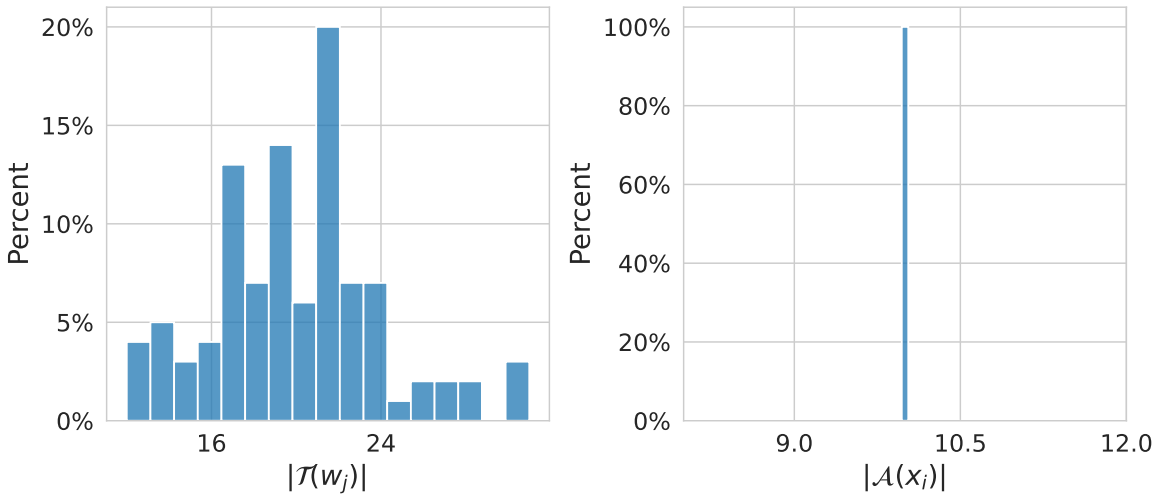
Difficulties are then drawn following at random. Tasks that are `easy` are answered correctly by every worker. Tasks that are `hard` are answered following the confusion matrix assigned to each worker. Each worker then answers independently to the presented tasks.

We simulate $n_{\text{task}} = 500$ tasks and $n_{\text{worker}} = 100$ with 35% of good workers in the crowd and 50% of easy tasks. There are $K = 5$ possible classes. Each task receives $|\mathscr{A}(x_i)| = 10$ 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)
nbins = 17
fig, ax = plt.subplots(1, 2, figsize=(9, 4))
sns.histplot(workerload, stat="percent", bins=nbins, shrink=1, ax=ax[0])
ax[0].yaxis.set_major_formatter(mtick.PercentFormatter(decimals=0))
ax[0].set_xlabel(r"$\vert\mathcal{T}(w_j)\vert$")
sns.histplot(feedback, stat="percent", bins=nbins, shrink=1, ax=ax[1])
ax[1].yaxis.set_major_formatter(mtick.PercentFormatter(decimals=0))
ax[1].set_xlabel(r"$\vert\mathcal{A}(x_i)\vert$")
ax[1].set_xlim(8, 12)
ax[1].xaxis.set_major_locator(MaxNLocator(integer=True))
```

```
for i in range(2):
  ax[i].xaxis.set_major_locator(MaxNLocator(3))
  ax[i].xaxis.label.set_size(15)
  ax[i].yaxis.label.set_size(15)
  ax[i].xaxis.set_tick_params(labelsize=13)
  ax[i].yaxis.set_tick_params(labelsize=13)
  ax[i].title.set_size(18)
plt.tight_layout()
plt.show()
```



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 = 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 4: AccTrain metric on simulated mistakes when tasks are associated a difficulty level considering classical feature-blind label aggregation strategies

	MV	NAIVESOFT	GLAD	DS	DSWC[L=2]	DSWC[L=5]
AccTrain	0.815	0.790	0.845	0.810	0.600	0.660

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 4). Note that in this case, creating clusters of answers leads to worse decisions than an MV aggregation.

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

Most often, tasks are crowdsourced to create a large 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-based 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 $\mathscr{C}$ be $z_i = \mathscr{C}(x_i)$. 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 $\mathscr{L}_{CE}^{\text{CrowdLayer}}$ and computed as:

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

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

The confusion matrices are taken into the network architecture as a new layer 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.

4.1.2 CoNAL

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

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

$$\mathscr{L}_{CE}^{\text{CoNAL}}(x_i) = \sum_{j\in\mathscr{A}(x_i)} \text{CE}(h_i^{(j)}, y_i^{(j)}) \ ,$$
$$\text{with } h_i^{(j)} = \sigma\Big(\big(\omega_i^{(j)}\pi^g + (1-\omega_i^{(j)})\pi^{(j)}\big) z_i\Big) \ .$$

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

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

4.2 Prediction error when learning from crowdsourced tasks

The AccTrain metric presented in Section 3.2 might no longer be of interest when training a classifier. Classical error measurements involve a test dataset to estimate the generalization error. To do so, we present hereafter two error metrics. Assuming we trained our classifier f_θ on a training set:

- the test accuracy is computed as $\frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}} \mathbf{1}_{\{y_i^\star = \widehat{f_\theta(x_i)}\}}$
- the expected calibration error (Guo et al. 2017) over M equally spaced bins $I_1, \dots, I_M$, computed as:

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

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

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

4.3 Use case with `peerannot` on real datasets

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

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

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

```
for strat in ["MV", "NaiveSoft", "DS", "GLAD"]:
  ! peerannot aggregate ./labalme/ -s {strat}
  ! peerannot train ./labelme -o labelme_${strat} -K 8 --labels=./labelme/labels/labels_labelme_
for strat in ["CrowdLayer", "CoNAL[scale=0]", "CoNAL[scale=1e-4]"]:
  ! peerannot aggregate-deep ./ -o labelme_${strat} --answers ./labelme/answers.json -s ${strat}


# 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="c
torch.save(best_conal["noise_adaptation"]["local_confusion_matrices"], path_ / "best_models"/ "l
```

```
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 = 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 5: 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.

	method	AccTest	ECE
0	ds	81.061	0.189
1	mv	85.606	0.143
2	naivesoft	86.448	0.136
3	crowdlayer	87.205	0.117
4	glad	87.542	0.124
5	conal[scale=0]	88.468	0.115
6	conal[scale=1e-4]	88.889	0.112

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

5 Exploring crowdsourced datasets

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

However, mistakes and confusions happen during these experiments. Sometimes involuntarily (*e.g.* because the task is too hard or the worker is unable to differentiate between two classes) and sometimes not (*e.g.* the worker is a spammer).

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

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

6 Conclusion

We introduced `peerannot`, a library to handle crowdsourced datasets. This library enables both easy label aggregation and direct training strategies with classical state-of-the-art classifiers. The identification module of the library allows exploring the collected data from both the tasks and the workers' point of view for better scorings and data cleaning procedures. Our library also comes with

templated datasets to better share crowdsourced datasets and have strategies more uniform to test on.

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

Aitchison, L. 2021. "A Statistical Theory of Cold Posteriors in Deep Neural Networks." In *ICLR*.

Chagneux, M, S LeCorff, P Gloaguen, C Ollion, O Lepâtre, and A Bruge. 2023. "Macrolitter Video Counting on Riverbanks Using State Space Models and Moving Cameras." *Computo*, February. https://doi.org/10.57750/845m-f805.

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

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

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

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

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

Gruber, S G, and F Buettner. 2022. "Better Uncertainty Calibration via Proper Scores for Classification and Beyond." In *Advances in Neural Information Processing Systems*.

Guo, C, G Pleiss, Y Sun, and KQ Weinberger. 2017. "On Calibration of Modern Neural Networks." In *ICML*, 1321.

Imamura, H, I Sato, and M Sugiyama. 2018. "Analysis of Minimax Error Rate for Crowdsourcing and Its Application to Worker Clustering Model." In *ICML*, 2147–56.

James, GM. 1998. "Majority Vote Classifiers: Theory and Applications." PhD thesis, Stanford University.

Kasmi, G, Y-M Saint-Drenan, D Trebosc, R Jolivet, J Leloux, B Sarr, and L Dubus. 2023. "A Crowdsourced Dataset of Aerial Images with Annotated Solar Photovoltaic Arrays and Installation Metadata." *Scientific Data* 10 (1): 59.

Khattak, FK. 2017. "Toward a Robust and Universal Crowd Labeling Framework." PhD thesis, Columbia University.

Krizhevsky, A, and G Hinton. 2009. "Learning Multiple Layers of Features from Tiny Images." University of Toronto.

Lefort, T, B Charlier, A Joly, and J Salmon. 2022. "Identify Ambiguous Tasks Combining Crowdsourced Labels by Weighting Areas Under the Margin." *arXiv Preprint arXiv:2209.15380*.

Lin, Tsung-Yi, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. 2014. "Microsoft COCO: Common Objects in Context." *CoRR* abs/1405.0312. http://arxiv.org/abs/1405.0312.

Marcel, S, and Y Rodriguez. 2010. "Torchvision the Machine-Vision Package of Torch." In *Proceedings of the 18th ACM International Conference on Multimedia*, 1485–88. MM '10. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1873951.1874254.

Park, Seo Yeon, and Cornelia Caragea. 2022. "On the Calibration of Pre-Trained Language Models Using Mixup Guided by Area Under the Margin and Saliency." In *ACML*, 5364–74.

Passonneau, R J., and B Carpenter. 2014. "The Benefits of a Model of Annotation." *Transactions of the*

Association for Computational Linguistics 2: 311–26.

Paszke, A, S Gross, F Massa, A Lerer, J Bradbury, G Chanan, T Killeen, et al. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In *NeurIPS*, 8024–35.

Peterson, J C., R M. Battleday, T L. Griffiths, and O Russakovsky. 2019. "Human Uncertainty Makes Classification More Robust." In *ICCV*, 9617–26.

Pleiss, G, T Zhang, E R Elenberg, and K Q Weinberger. 2020. "Identifying Mislabeled Data Using the Area Under the Margin Ranking." In *NeurIPS*.

Rodrigues, F, M Lourenco, B Ribeiro, and F C Pereira. 2017. "Learning Supervised Topic Models for Classification and Regression from Crowds." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (12): 2409–22.

Rodrigues, F, and F Pereira. 2018. "Deep Learning from Crowds." In *AAAI*. Vol. 32.

Rodrigues, F, F Pereira, and B Ribeiro. 2014. "Gaussian Process Classification and Active Learning with Multiple Annotators." In *ICML*, 433–41. PMLR.

Servajean, M, A Joly, D Shasha, J Champ, and E Pacitti. 2016. "ThePlantGame: Actively Training Human Annotators for Domain-Specific Crowdsourcing." In *Proceedings of the 24th ACM International Conference on Multimedia*, 720–21. MM '16. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2964284.2973820.

———. 2017. "Crowdsourcing Thousands of Specialized Labels: A Bayesian Active Training Approach." *IEEE Transactions on Multimedia* 19 (6): 1376–91.

Sinha, V B, S Rao, and V N Balasubramanian. 2018. "Fast Dawid-Skene: A Fast Vote Aggregation Scheme for Sentiment Classification." *arXiv Preprint arXiv:1803.02781*.

Tinati, R, M Luczak-Roesch, E Simperl, and W Hall. 2017. "An Investigation of Player Motivations in Eyewire, a Gamified Citizen Science Project." *Computers in Human Behavior* 73: 527–40.

Whitehill, J, T Wu, J Bergsma, J Movellan, and P Ruvolo. 2009. "Whose Vote Should Count More: Optimal Integration of Labels from Labelers of Unknown Expertise." In *NeurIPS*. Vol. 22.

Yasmin, R, M Hassan, J T Grassel, H Bhogaraju, A R Escobedo, and O Fuentes. 2022. "Improving Crowdsourcing-Based Image Classification Through Expanded Input Elicitation and Machine Learning." *Frontiers in Artificial Intelligence* 5: 848056.

Zhang, H, M Cissé, Y N. Dauphin, and D Lopez-Paz. 2018. "Mixup: Beyond Empirical Risk Minimization." In *ICLR*.