COMPUTO

ISSN 2824-7795

Peerannot: learning from crowd-sourced image datasets with Python

Tanguy Lefort[1] Statistics, Name of Affiliation one
Benjamin Charlier Computer Science, Name of Afficiliation two
Alexis Joly Computer Science, Name of Afficiliation two
Joseph Salmon Computer Science, Name of Afficiliation two

Date published: 2023-04-14 Last modified: 2023-04-14

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 freely 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 (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`:

- *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.
- *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 XXX to train on – and in crowdsourced datasets, they are not always available. This is treated with the `aggregate-deep` command.
- *How to score workers in a crowdsourcing experiment?* Beyond learning a classifier or inferring the true label, one might want to find a scoring or ranking between workers. For example, it is relevant for the gamification of crowdsourcing experiments (Servajean et al. 2016). This is treated with the `identify` 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.

2 Notation and package structure

2.1 Crowdsourcing notation

Let us consider the classical supervised learning classification framework. A training set $\mathcal{D} = \{(x_i, y_i^\star)\}_{i=1}^{n_{\text{task}}}$ is composed of n_{task} tasks $x_i \in \mathcal{X}$ (the feature space) with 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 σ for the softmax function.

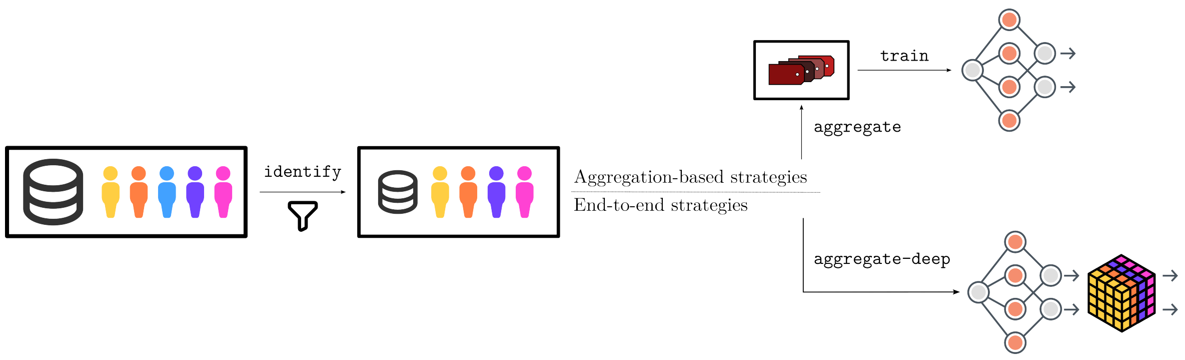


Figure 1: From crowdsourced labels to training a classifier neural network, the full pipeline using the `peerannot` library.

With crowdsourced data the ground truth $y_i^\star$ is unknown, and there is no single label that can be trusted as in standard supervised learning (even on the train set!). Instead, for a given task x_i, a worker w_j proposes a label $y_i^{(j)}$. The set of workers answering the task x_i is denoted by $\mathcal{A}(x_i) = \{j \in [n_{\texttt{worker}}] : w_j \text{ answered } x_i\}$. The cardinal $|\mathcal{A}(x_i)|$ is called the feedback effort on the task x_i. Note that the feedback effort can not exceed the total number of workers $n_{\texttt{worker}}$. Similarly, one can adopt a worker point of view: the set of tasks answered by a worker w_j is denoted $\mathcal{T}(w_j) = \{i \in [n_{\texttt{task}}] : w_j \text{ answered } x_i\}$. The cardinal $|\mathcal{T}(w_j)|$ is called the workerload of w_j. The final dataset can then be decomposed as:

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

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).

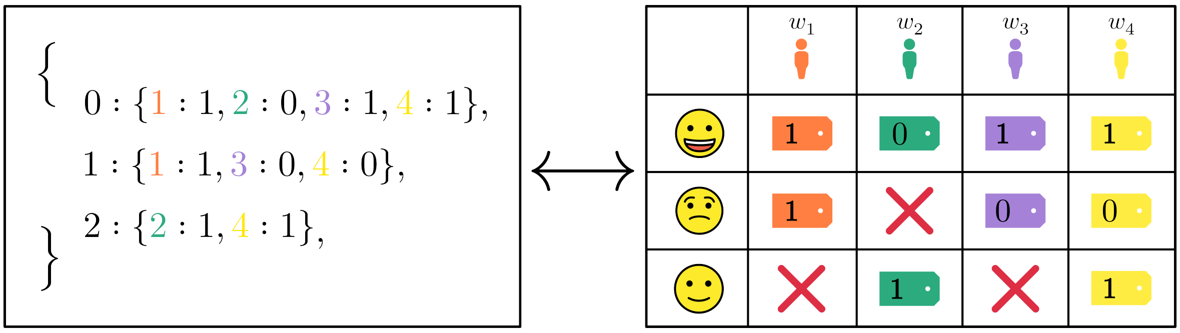
2.2 Storing crowdsourced datasets in `peerannot`

To store crowdsourcing datasets efficiently and in a standardized way, `peerannot` proposes the following structure, where each dataset equals a folder:

```
datasetname
        train
             class0
                  task0-<vote_index_0>.png
                  task1-<vote_index_1>.png
                  ...
                  taskn0-<vote_index_n0>.png
             class1
             ...
             classK
        val
        test
        metadata.json
        answers.json
```

If the tasks (images) are available, they must be stored as it is usual to store `ImageFolder` datasets with `pytorch` into a `train`, `val` and `test` folder. Each image can have its name followed by its index in the `answers.json` file.

The `answers.json` file stores the different votes for each task as described in **?@fig-answers**. Finally, a `metadata.json` file includes all relevant information related to the crowdsourcing experiment such as the number of workers, the number of tasks, *etc.* XXX (may be give an example for one iconic dataset)



$$\left\{ \begin{array}{l} 0: \{1:1, 2:0, 3:1, 4:1\}, \\ 1: \{1:1, 3:0, 4:0\}, \\ 2: \{2:1, 4:1\}, \end{array} \right\} \longleftrightarrow$$

	w_1	w_2	w_3	w_4
😀	1	0	1	1
😕	1	✗	0	0
🙂	✗	1	✗	1

(XXX put the number in the svg file to make it more readable with the left part)

In this example, there are three tasks, $n_{worker} = 4$ workers and $K = 2$ classes. For the first task, the feedback effort is $|\mathscr{A}(x_1)| = 4$. The workerload of w_2 is $|\mathscr{T}(w_2)| = 2$.

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 $\hat{y}_i \in \Delta_K$ defined on the simplex of dimension $K-1$ depending on the use case:

- 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 be a probability distribution other than Dirac distribution, in that case, each coefficient in $\hat{y}_i$ 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 creating a new dataset and 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 in practice releasing 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 induced by taking the class with the maximum probability.

3.1 Classical models

While the most intuitive way to create a label from multiple answers for any type of crowdsourced task would be to take the majority vote (MV), 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 baseline aggregation can be expressed as:

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

One pitfall with the 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 frequency distribution as the task label:

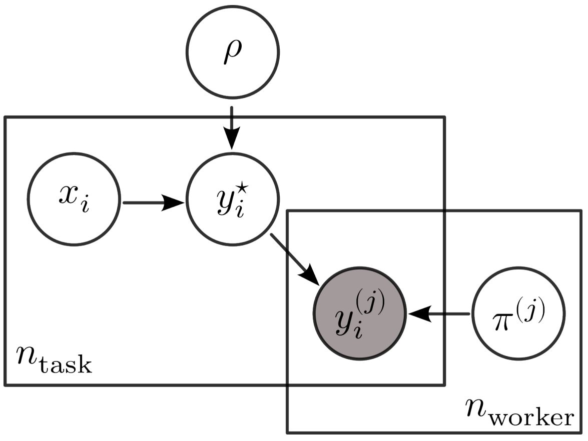
$$\hat{y}_i^{\mathrm{NS}} = \left(\frac{1}{|\mathscr{A}(x_i)|}\sum_{j\in\mathscr{A}(x_i)} 1_{\{y_i^{(j)}=k\}}\right)_{j\in[K]} .$$

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.

Going further into the aggregation, researchers began creating a noise model to take into account the workers' abilities in the aggregation. 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 the probability for a task x_i to have true label $y_i^\star = k$ 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):

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

with $T_{i,k} = 1_{\{y_i^\star = k\}}$. 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 2: Bayesian plate notation for 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 L clusters of workers (Imamura, Sato, and Sugiyama 2018) with $1 \leq L \leq n_{\text{worker}}$ (DSWC) or even faster implementation that produces only hard labels (Sinha, Rao, and Balasubramanian 2018).

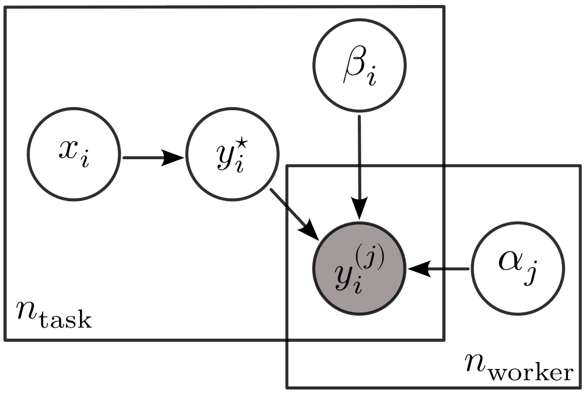
Finally, we present the GLAD model (Whitehill et al. 2009) that not only takes into account the worker's ability, but also the task difficulty in the noise model. Denoting $\alpha_j \in \mathbb{R}$ the worker ability (the higher the better) and $\beta_i \in \mathbb{R}_\star^+$ the task's difficulty (the 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:

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

The likelihood can then be 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 3: Bayesian plate notation for the GLAD model

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 $\mathcal{D}$ for a given aggregated label $(\hat{y}_i)_i$ as:

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

In the following, we write AccTrain for AccTrain($\mathcal{D}_{\mathrm{train}}$) 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.

We first consider classical simulation settings in the literature that can easily be created and reproduced using `peerannot`.

3.2.1 Simulated independent mistakes

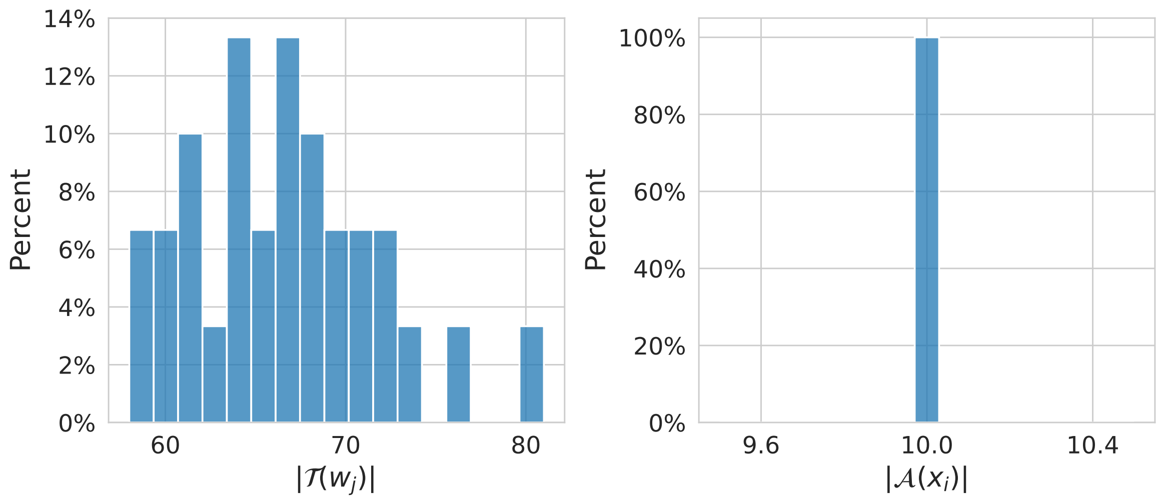
The independent mistakes consider that each worker w_j answers following a multinomial distribution with weights given at the row $y_i^\star$ of their confusion matrix $\pi^{(j)} \in \mathbb{R}^{K\times K}$. Each confusion matrix is generated diagonally dominant. Answers are independent of one another 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 receives $|\mathcal{A}(x_i)| = 10$ labels.

```
! 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))
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	0.740	0.760	0.775	0.890	0.775	0.770

As expected by the simulation framework, Table 1 fits the DS model, thus leading to a better accuracy on the retrieval of the simulated label for the DS model. The MV aggregation doesn't consider any worker-ability scoring or the task's difficulty and performs the worse.

3.2.2 Simulated correlated mistakes

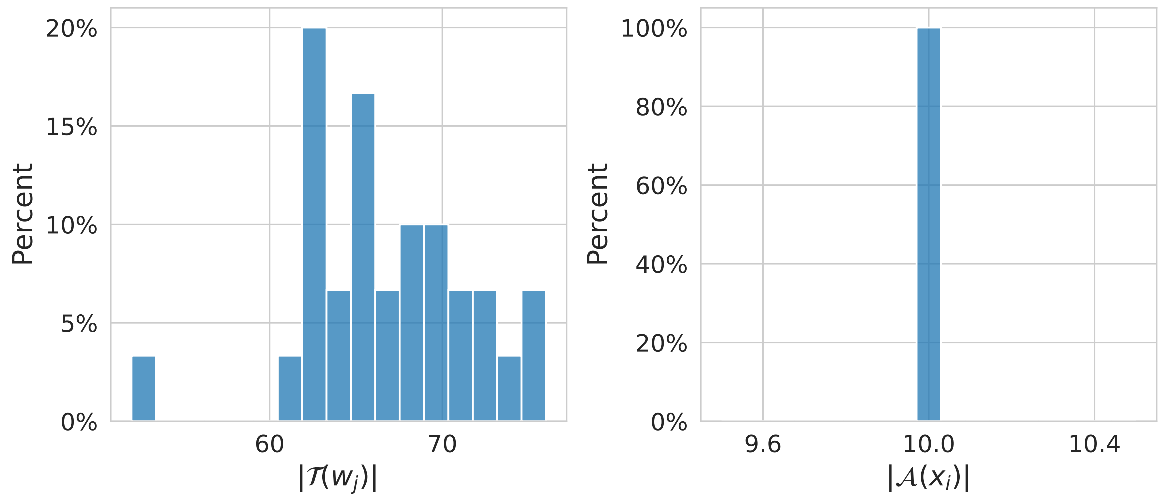
The correlated mistakes are also known as the student-teacher setting. Consider that the crowd of workers is divided into two categories: teachers and students such that $n_{\text{teacher}} + n_{\text{student}} = n_{\text{worker}}$. Each student is randomly assigned to one teacher at the beginning of the experiment. We generate the (diagonally dominant) confusion matrices of each teacher and the students are associated with their's teacher confusion matrix. 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 $|\mathcal{A}(x_i)| = 10$ 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)
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))
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 2: AccTrain metric on simulated correlated mistakes considering classical feature-blind label aggregation strategies

	MV	NAIVESOFT	GLAD	DS	DSWC[L=5]	DSWC[L=10]
AccTrain	0.725	0.690	0.645	0.755	0.795	0.815

With Table 2, 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 10 different confusion matrices (insted of 30 for the vanilla DS model).

3.2.3 Simulated mistakes with discrete difficulty levels on tasks

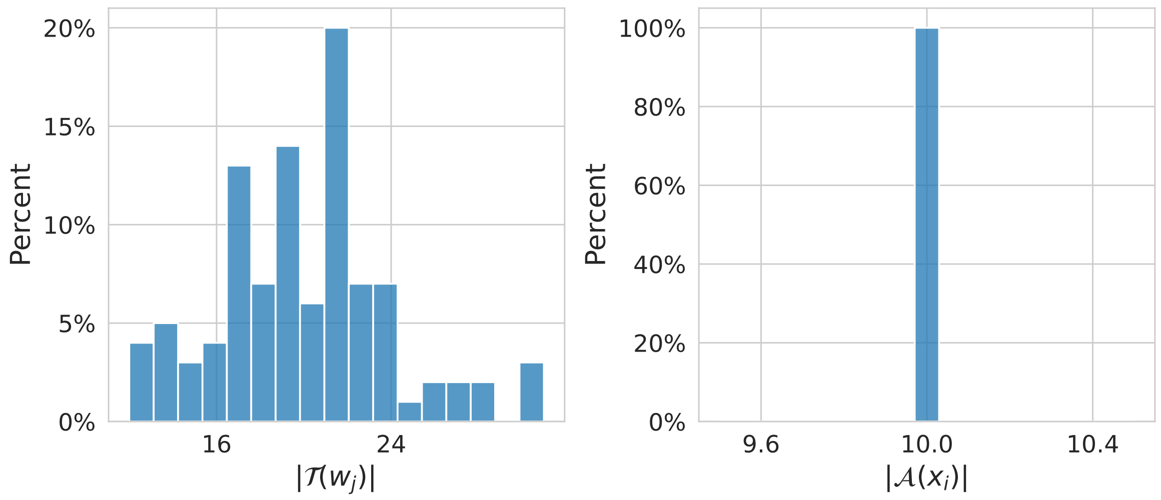
For the final simulation setting, we consider the discrete difficulty presented in Whitehill et al. (2009). Contrary to other simulations, we here consider that each worker is either good or bad and each task is either easy or hard. Easy tasks are answered without mistakes by involved workers. However, hard tasks are answered following the worker's confusion matrix. The confusion matrix $\pi^{(j)}$ is diagonally dominant for good workers while each row is drawn uniformly in the simplex Δ_K for bad workers. 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 $|\mathcal{A}(x_i)| = 10$ 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)
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"$\vert\mathcal{A}(x_i)\vert$")
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 3: 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 diffulty coefficients, the only strategy that involves a latent variable for the task difficulty, knowing GLAD, outperforms the other other strategies (see Table 3). Note that in this case, creating clusters of answers leads to worse decisions than a MV aggregation.

To summurize 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 let us directly train a classifier from multiple noisy labels.

4.1 Classical models

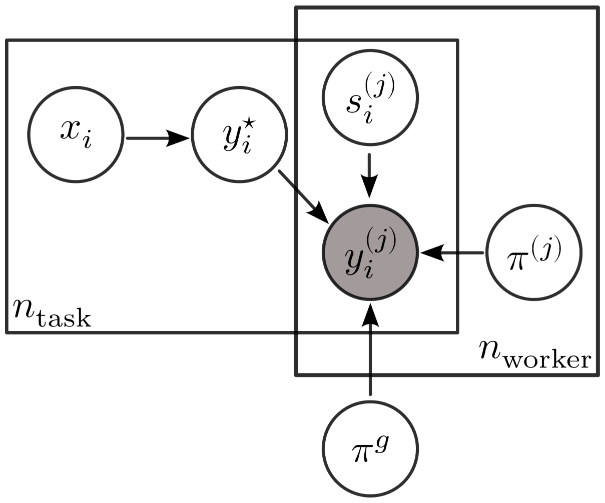
In recent years, directly learning a classifier from noisy labels was introduced. Two of the most used models: CrowdLayer (Rodrigues and Pereira 2018) and CoNAL (Chu, Ma, and Wang 2021), are directly available in `peerannot`. These two learning strategies directly incorporate a DS-based noise model in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

CrowdLayer trains a classifier with noisy labels as follows. Let the scores (logits) output of a given classifier neural network $\mathscr{C}$ be $z_i = \mathscr{C}(x_i)$. Then CrowdLayer adds a new 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:

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

The confusion matrices are incorporated as is into the network architecture as a new layer to transform the output probabilities to match each worker's answer. However, 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 not only modeling the worker confusion matrices; but also a global confusion matrix $\pi^g \in \mathbb{R}^{K\times K}$.

Given the output $z_i = \mathscr{C}(x_i) \in \mathbb{R}^K$ of a given classifier and task, CoNAL interpolates between the



- $x_i \in \mathcal{X}$: task
- $y_i^\star$: true label (unobserved)
- $y_i^{(j)} \in [K]$: label observed
- $\pi^{(j)} \in \mathbb{R}^{K\times K}$: confusion matrix
- $\pi^g \in \mathbb{R}^{K\times K}$: global confusion matrix
- $s_i^{(j)} \in \{0,1\}$: confusion source

Figure 4: Bayesian plate notation for CoNAL model. Each worker is assigned a confusion matrix $\pi^{(j)}$. A global confusion matrix π^g is shared between workers. A tradeoff between the global confusion and the local one is applied.

local confusion $\pi^{(j)}\sigma(z_i)$ and the global one $\pi^g\sigma(z_i)$. The loss function is computed as follows:

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

The interpolation weight is defined as $s_i^{(j)} \sim \mathcal{B}(w_i^{(j)})$ and is unobservable in practice. So, to compute $h_i^{(j)}$, the weight is obtained through an auxiliary network that projects the task x_i onto v_i and the worker w_j onto u_j, two vectors in an embedding space of dimension $d \geq 1$. 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 $\frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}} 1_{\{y_i^\star = \widehat{f_\theta(x_i)}\}}$
- 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 $B_m = \{x_i | \mathcal{C}(x_i)_{[1]} \in I_m\}$ 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, 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 issue with CIFAR-10H is that there are little to no disagreements and a simple majority voting leads to a near-perfect AccTrain error. While with this dataset, because of the lack of disagreements and data curation (Peterson et al. 2019; Aitchison 2021) comparing the impact of aggregation and end-to-end strategies might not be relevant, it is however a good benchmark for task difficulty identification and worker evaluation scoring

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

To install these datasets, we run the `install` command from `peerannot`:

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

Let us use `peerannot` to train a Resnet34 on the LabelMe dataset for:

- aggregation strategies: MV, NS, DS, GLAD,
- end-to-end strategies: CrowdLayer and CoNAL.

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`

5.1 Exploring tasks' difficulty

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

- the entropy of the NS distribution: reliable with a big enough and not adversarial crowd, the entropy measures the inherent uncertainty of the distribution to the possible outcomes.
- GLAD's scoring: by construction, Whitehill et al. (2009) introduced a scalar coefficient to score the difficulty of a task $\beta_i > 0$.
- the WAUM: introduced in (Lefort et al. 2022), this weighted area under the margins indicates how difficult it is for a model to classify the task given the crowdsourced labels and the trust we have in each worker.

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

5.1.1 CIFAR-10H dataset

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

```python
import torch
import matplotlib.pyplot as plt
from PIL import Image
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(left=0.05, bottom=0.05, right=0.95, top=0.95, wspace=0.05, hspace=0.05)
plt.show()
```

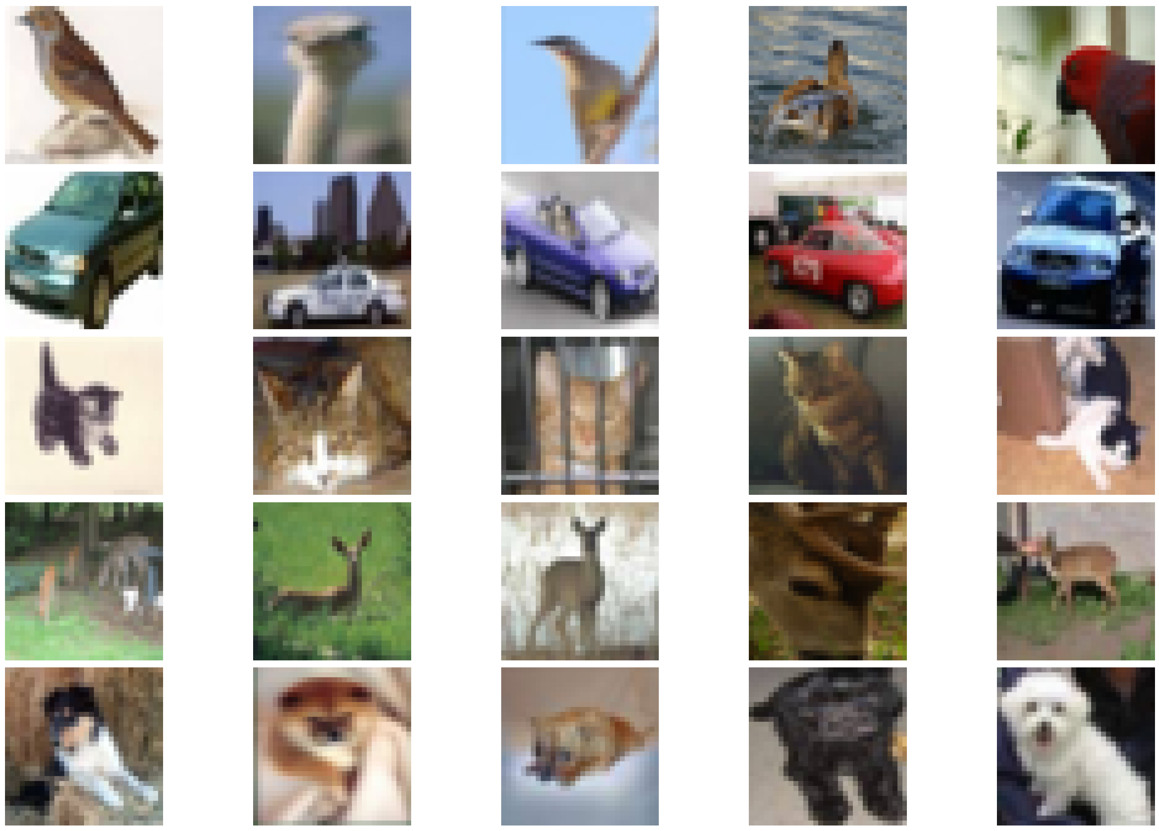


Figure 5: Example of images to label from the CIFAR-10H dataset with label `bird`, `car`, `cat`, `deer`and `dog` (top to bottom) by row.

```
! peerannot identify ./datasets/cifar10H -s entropy -K 10 --labels ./datasets/cifar10H/answers.j
! peerannot aggregate ./datasets/cifar10H/ -s GLAD
```

```
from scipy.stats import pearsonr
def corrfunc(x, y, ax=None, **kws):
    r, _ = pearsonr(x, y)
    ax = ax or plt.gca()
    ax.annotate(rf'$\rho$ = {r:.2f}', xy=(.1, .9), xycoords=ax.transAxes)
results = {'glad': [], "entropy": []}
path = Path.cwd() / "datasets" / "cifar10H" / "identification"
results["entropy"] = np.load(path / 'entropies.npy')
results["glad"] = np.exp(np.load(path / "glad" / "difficulties.npy")[:, 1])
# results["waum"] = pd.read_csv(path / "resnet34" / "waum_0.01_yang" / 'waum.csv')["waum"].value
results = pd.DataFrame(results)
g = sns.pairplot(results, corner=True, diag_kind="kde")
g.map_lower(corrfunc)

# axes = g.axes.flatten()
# for i, ax in enumerate(axes):
#   if i % len(results) == 0:
```

```
plt.show()
```

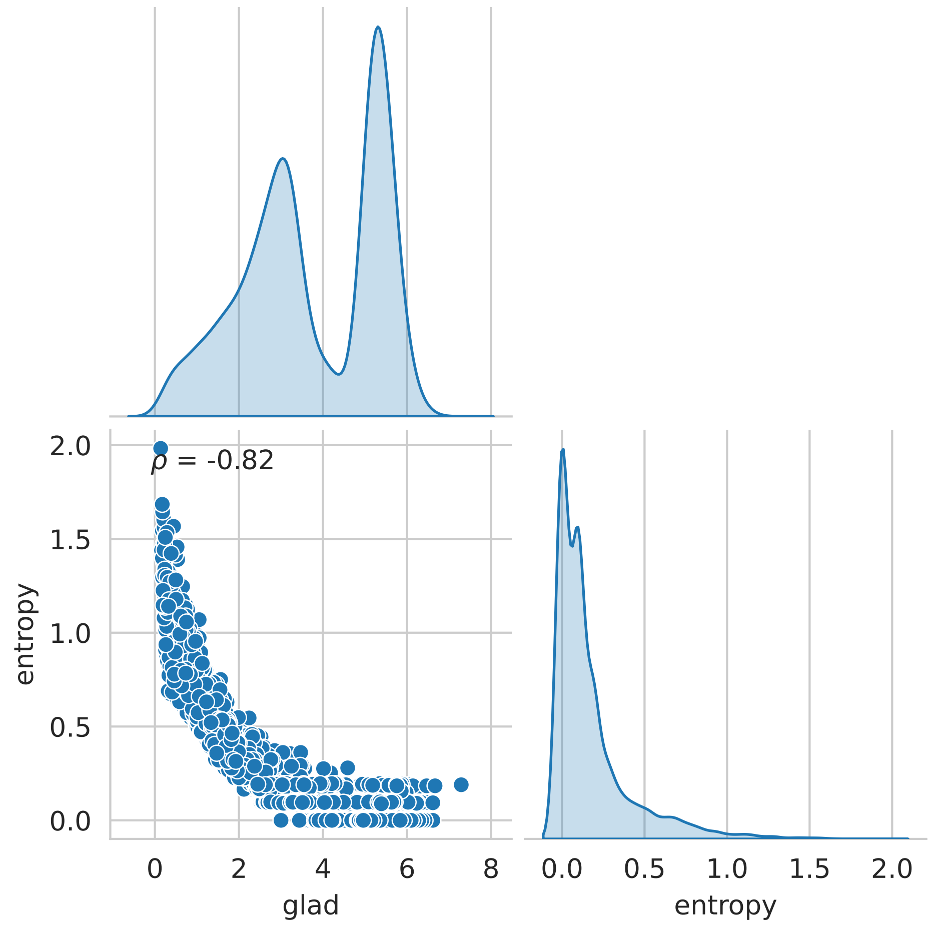


Figure 6: Comparison of metrics scoring the task's intrinsic difficulty.

5.1.2 LabelMe dataset

```
! peerannot identify ./datasets/labelme/ -s entropy -K 10 --labels ./datasets/labelme/answers.js
! peerannot aggregate ./datasets/labelme/ -s GLAD
```

```
from scipy.stats import pearsonr
def corrfunc(x, y, ax=None, **kws):
    r, _ = pearsonr(x, y)
    ax = ax or plt.gca()
    ax.annotate(rf'$\rho$ = {r:.2f}', xy=(.1, .9), xycoords=ax.transAxes)
results = {'glad': [], "entropy": []}
path = Path.cwd() / "datasets" / "labelme" / "identification"
results["entropy"] = np.load(path / 'entropies.npy')
results["glad"] = np.exp(np.load(path / "glad" / "difficulties.npy")[:, 1])
results = pd.DataFrame(results)
g = sns.pairplot(results, corner=True, diag_kind="kde")
g.map_lower(corrfunc)
```

```
plt.show()
```

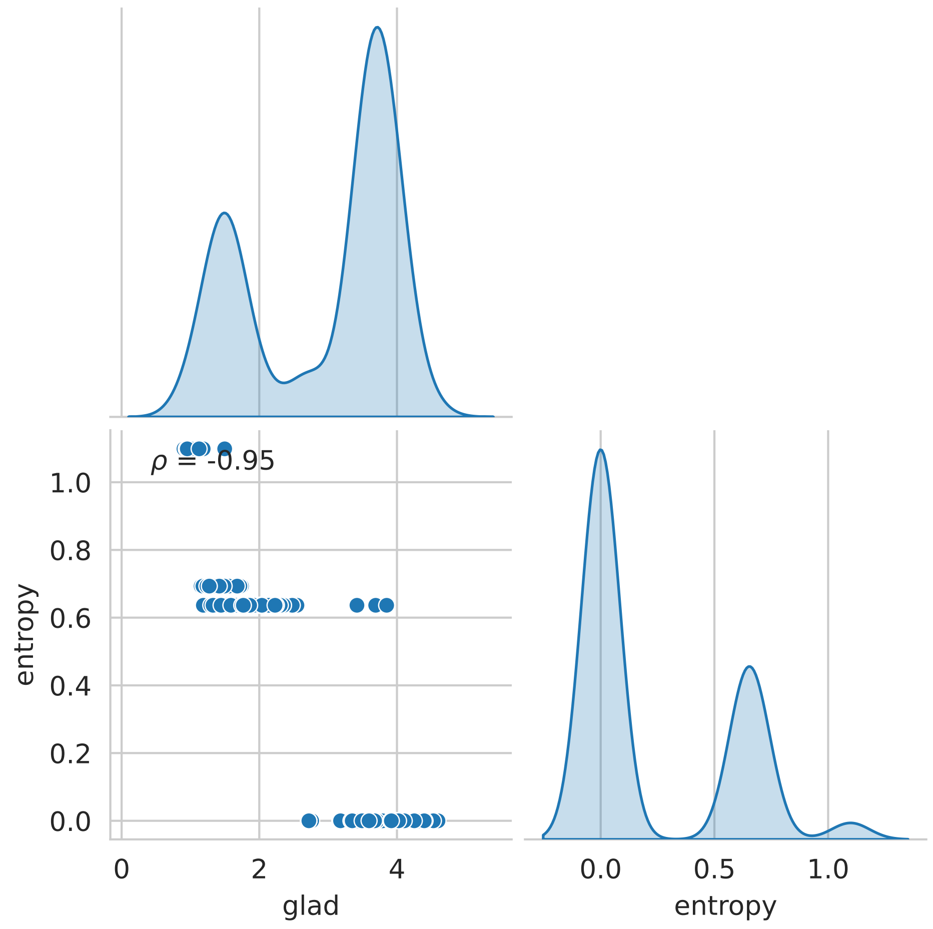


Figure 7: Comparison of metrics scoring the task's intrinsic difficulty.

5.2 Exploring workers' reliability

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

- the trace of the confusion matrix: the closer to K the better the worker,
- the spammer score (Raykar and Yu 2011) that is the Frobenius norm between the estimated confusion matrix $\hat{\pi}^{(j)}$ and the closest rank-1 matrix. The further to zero the better the worker.

When the tasks are available, confusion-matrix-based deep learning models can also be of use. We thus add to the comparison the trace of the confusion matrices with CrowdLayer and CoNAL on the datasets. For CoNAL, we only consider the trace of the confusion matrix $\pi^{(j)}$ in the pairwise comparison, and provide the common confusion matrix π^g as separate.

5.2.1 CIFAR-10H

The Cifar-10H dataset has few disagreements among workers. From **?@fig-abilities-cifar10H**, we can see that

```
! 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_confusion": [], "spam_score": [], "glad": []}
results_identif["trace_confusion"].extend(np.load(path_ / 'identification' / "traces_confusion.n
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")
g.map_lower(corrfunc)
plt.show()
```

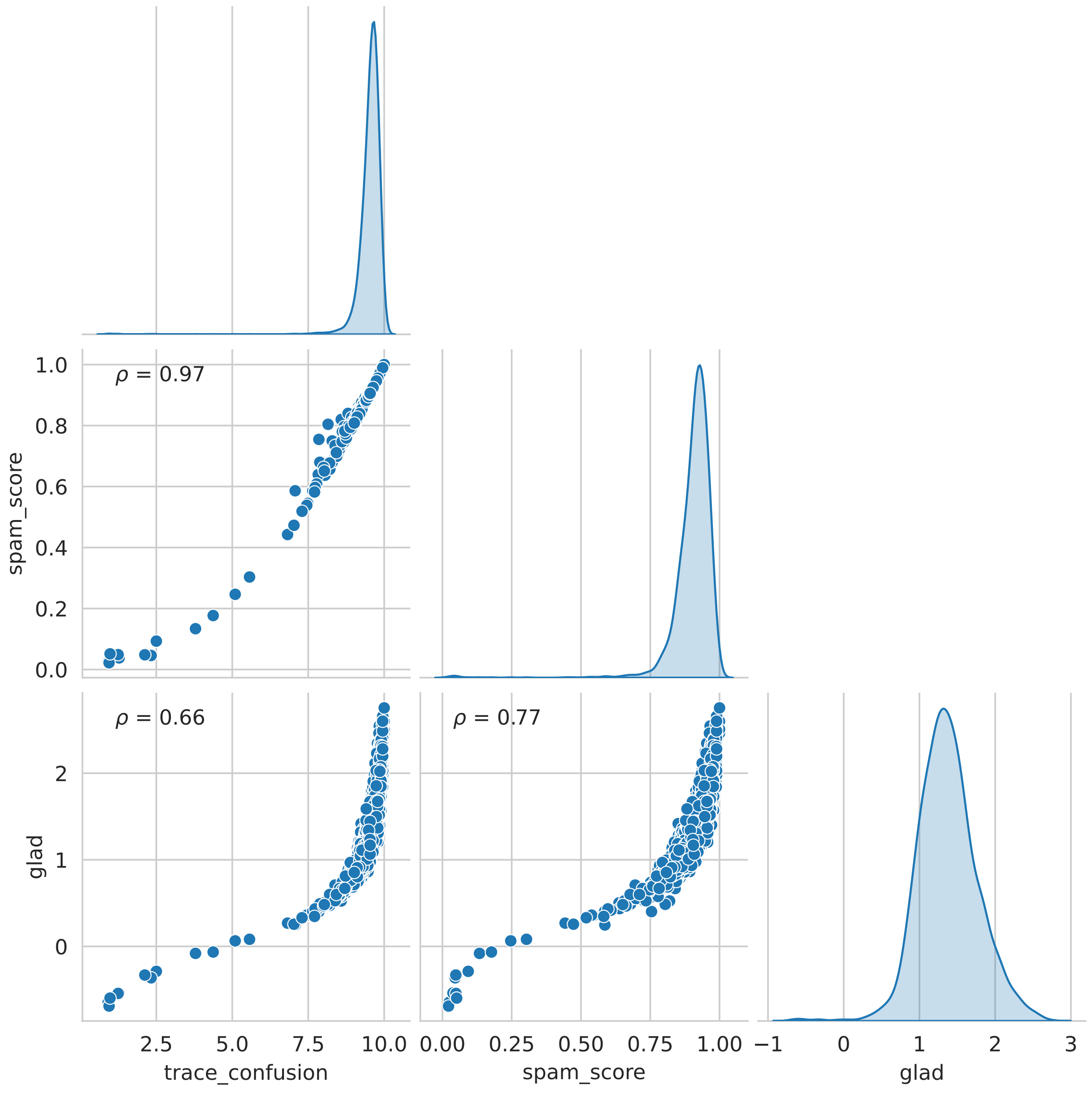


Figure 8: Comparison of ability scores by workers for the CIFAR-10H dataset.

5.2.2 LabelMe

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

```
path_ = Path.cwd() / "datasets" / "labelme"
results_identif = {"trace_confusion": [], "spam_score": [], "glad": []}
results_identif["trace_confusion"].extend(np.load(path_ / 'identification' / "traces_confusion.n
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")
g.map_lower(corrfunc)
plt.show()
```

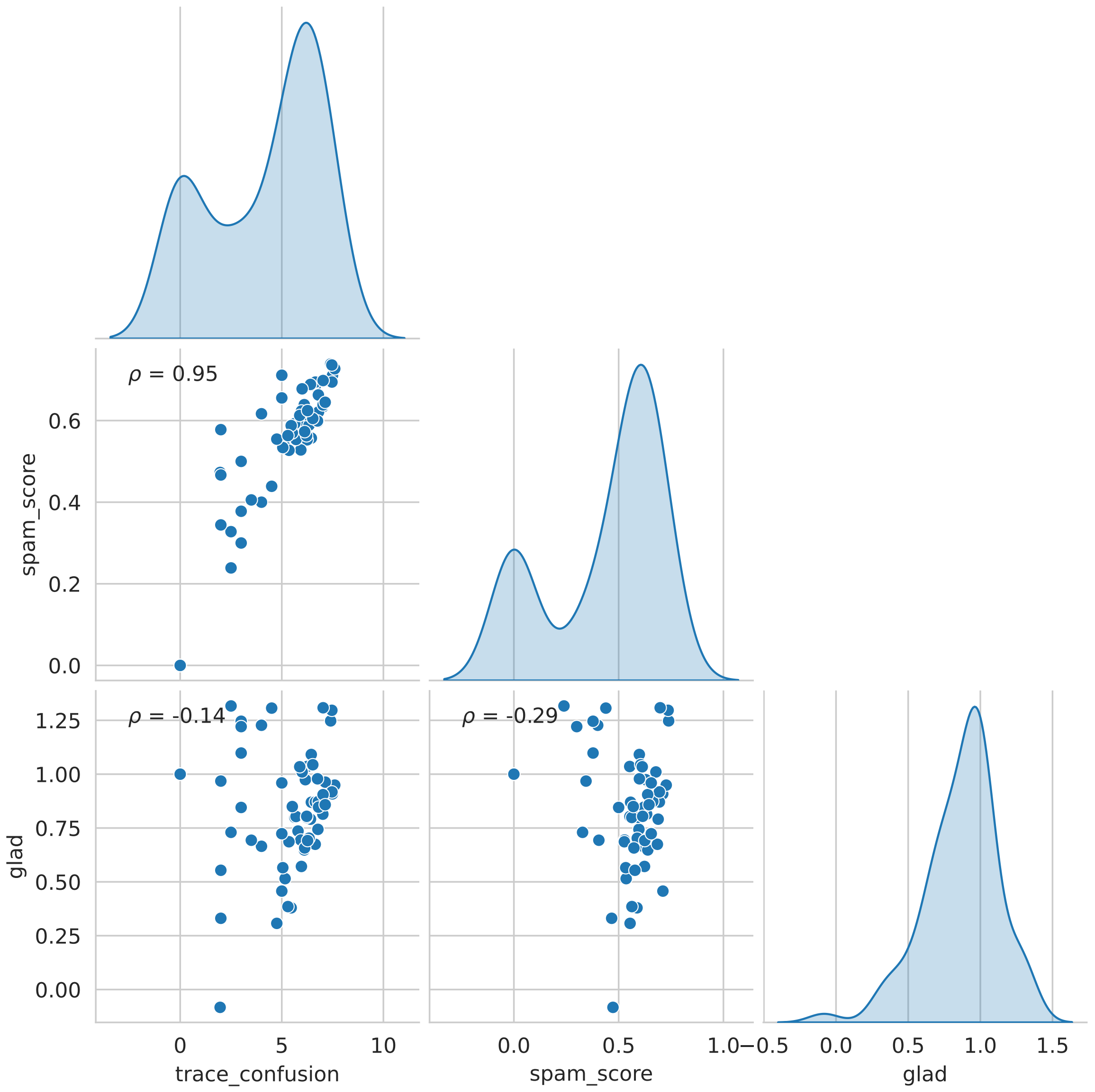


Figure 9: Comparison of ability scores by workers for the CIFAR-10H dataset.

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

Chagneux, Mathis, Sylvain LeCorff, Pierre Gloaguen, Charles Ollion, Océane Lepâtre, and Antoine Bruge. 2023. "Macrolitter Video Counting on Riverbanks Using State Space Models and Moving Cameras." *Computo*, February. https://doi.org/10.57750/845m-f805.

Chu, Zhendong, Jing Ma, and Hongning 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, Chao, and Dengyong 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, Sebastian Gregor, and Florian 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, Hideaki, Issei Sato, and Masashi Sugiyama. 2018. "Analysis of Minimax Error Rate for Crowdsourcing and Its Application to Worker Clustering Model." In *ICML*, 2147–56.

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

Kasmi, Gabriel, Yves-Marie Saint-Drenan, David Trebosc, Raphaël Jolivet, Jonathan Leloux, Babacar Sarr, and Laurent Dubus. 2023. "A Crowdsourced Dataset of Aerial Images with Annotated Solar Photovoltaic Arrays and Installation Metadata." *Scientific Data* 10 (1): 59.

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

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

Lefort, Tanguy, Benjamin Charlier, Alexis Joly, and Joseph 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, Sébastien, and Yann 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, Rebecca J., and Bob Carpenter. 2014. "The Benefits of a Model of Annotation." *Transactions of the Association for Computational Linguistics* 2: 311–26.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor

Killeen, et al. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In *NeurIPS*, 8024–35.

Peterson, Joshua C., Ruairidh M. Battleday, Thomas L. Griffiths, and Olga Russakovsky. 2019. "Human Uncertainty Makes Classification More Robust." In *ICCV*, 9617–26.

Pleiss, Geoff, Tianyi Zhang, Ethan R Elenberg, and Kilian Q Weinberger. 2020. "Identifying Mislabeled Data Using the Area Under the Margin Ranking." In *NeurIPS*.

Raykar, Vikas C, and Shipeng Yu. 2011. "Ranking Annotators for Crowdsourced Labeling Tasks." In *NeurIPS*, 1809–17.

Rodrigues, Filipe, Mariana Lourenco, Bernardete Ribeiro, and Francisco 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, Filipe, and Francisco Pereira. 2018. "Deep Learning from Crowds." In *AAAI*. Vol. 32.

Servajean, Maximilien, Alexis Joly, Dennis Shasha, Julien Champ, and Esther 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, Vaibhav B, Sukrut Rao, and Vineeth N Balasubramanian. 2018. "Fast Dawid-Skene: A Fast Vote Aggregation Scheme for Sentiment Classification." *arXiv Preprint arXiv:1803.02781*.

Tinati, Ramine, Markus Luczak-Roesch, Elena Simperl, and Wendy 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, Romena, Md Mahmudulla Hassan, Joshua T Grassel, Harika Bhogaraju, Adolfo R Escobedo, and Olac Fuentes. 2022. "Improving Crowdsourcing-Based Image Classification Through Expanded Input Elicitation and Machine Learning." *Frontiers in Artificial Intelligence* 5: 848056.

Zhang, Hongyi, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. 2018. "Mixup: Beyond Empirical Risk Minimization." In *ICLR*.