THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Biostatistiques

École doctorale Information, Structures, Systèmes

Unité de recherche Institut Montpelliérain Alexander Grothendieck

Label ambiguity in crowdsourcing for classification and expert feedback

Présentée par Tanguy LEFORT
en septembre 2024

Sous la direction de Joseph SALMON
et Benjamin CHARLIER et Alexis JOLY

Devant le jury composé de

Joseph Salmon, Professeur, Université de Montpellier	**Directeur de thèse**
Benjamin Charlier, Maître de conférence, Université de Montpellier	**Co-encadrant**
Alexis Joly, Directeur de recherche, INRIA Montpellier	**Co-directeur**
Jon Chamberlain, Senior Lecturer and researcher, University of Essex	**Rapporteur**
Sophie Donnet, Directrice de recherche, INRAE Paris-Saclay	**Rapportrice**
Odalric-Ambrym Maillard, Chargé de recherche HDR, INRIA Lille	**Examinateur**
Yolande Le Gall, Maître de conférence, Université de Rennes, IRISA	**Examinatrice**
Aurélien Bellet, Directeur de recherche, INRIA Montpellier	**Examinateur**

Benchamin

UNIVERSITÉ DE MONTPELLIER

Contents

1 Introduction

Key points – Crowdsourcing in todays spectrum

1. In the deep learning era we create bigger and bigger datasets, using labels that are not always correct. The prediction quality of a model follows the input data quality. Can we mitigate the impact of the noise induced by the data collection process?

2. Handling crowdsourcing data comes with two main strategies: either aggregate the collected labels and then use supervised learning from them. Or directly integrate a model that mitigates the crowd noise inside the neural network's architecture. However, most models deal with worker quality, few consider data difficulty.

3. Research has been made into trying to better learn the prediction uncertainty for a neural network by inducing noise in the label distribution. Using crowdsourced data, we have a direct access to the human uncertainty. However, this data is rarely openly available, making crowdsourcing experiments hard to reproduce and compare.

Contributions – Crowdsourcing: from the worker to the task and back

4. Following research on labeling noise in supervised datasets, we adapt the AUM into the WAUM to help detect highly ambiguous images in crowdsourced datasets.

5. We emphasize the need for reproducible results and shared algorithms with the creation of a crowdsourcing library: `peerannot`. This lets us easily compare scoring metrics for workers depending on the strategy used and their use case.

6. We discuss the need for more openly available crowdsourced datasets and the limitations of currently accessible ones.

Chapter 1 – Introduction:

1.1 Crowdsourcing in computer vision

Following the deep learning revolution, models with millions if not billions of parameters to optimize are trained nowadays. And to do so, we need to increase the size of our training datasets. The issue is, that in a training set in supervised classification (and we will mainly consider image classification for the rest of this work), the images need to come with a label to train from. While research has been conducted on inferring this label from the data itself, in the classical supervised learning datasets these labels have been collected and shared at some point during the creation of the dataset.

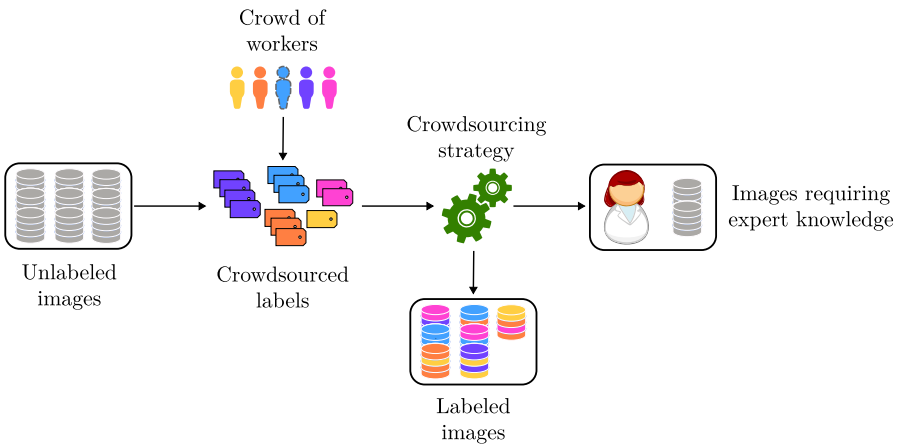


Figure 1.1 Crowdsourcing can be used to create datasets, but also relieves the tension experts receive in evaluating new data. By using a crowd of workers, many tasks can be labeled without needing expert evaluation.

The task of collecting data to create a dataset for image classification has been more and more linked with web-scrapping (Rhodes et al., 2015). However, it has been shown

numerous times that labeling mistakes do happen with this data collection strategy (Northcutt et al., 2021a; Vasudevan et al., 2022). In other fields of research, as healthcare providers, images might not come from the web but are collected between multiple locations – like hospital scanners. When trying to classify a scanner as *"presence or absence of tumors"*, it is often a field expert that provides the label. However such healthcare providers can't label thousands of images. In this case, the power of the crowd can be used to label the scans and provide some level of uncertainty. This is then to the data scientist area, in collaboration with experts, to determine which workers were useful in the experiment, and which were not, and then only ask experts to label the most ambiguous scans – which would represent only a small fraction of the original dataset (as illustrated in Figure 1.1).

1.1.1 Crowdsourcing is all over

This collaboration between citizens and scientists is not technically niche, only quite often it goes unnoticed. Using crowdsourcing to collect data has been used in medical science, video recommendation systems, and so many other fields of research and development. We wish to show a – non-exhaustive and not ordered – list of such projects relying on human interactions to collect data, not just for image classification:

- Pl@ntNet: plant species recognition application (Barthélémy et al., 2011),
- Eyewire: a game to map retina's neurons (Tinati et al., 2017),
- Tournesol: public interest YouTube video recommendation system (Hoang et al., 2021),
- ChatGPT: evaluate and correct prompt quality (OpenAI, 2023),
- Duolingo: traduction corrections and improvements[1],
- RTR: photovoltaic panels segmentation in images (Kasmi et al., 2023),
- Twitter/𝕏 via Birdwatch: identify misleading information (Wojcik et al., 2022),
- Spotify[2], TripAdvisor[3], SNCF[4],...

[1] https://www.theguardian.com/education/2014/aug/27/luis-von-ahn-ceo-duolingo-interview

[2] https://community.spotify.com/t5/Content-Questions/Shutting-down-Line-In/td-p/4557664

[3] https://www.kingigilbert.com/crowdsourcing-tripadvisor/

[4] https://www.usine-digitale.fr/article/tranquilien-quand-open-data-et-crowdsourcing-profitent-aux-voyageurs-franciliens.N200017

With this list, we showcase the large domains of applications of keeping humans – and citizens – in the loop, and thus the need for more research on crowdsourcing settings, strategies and implications. In this thesis, we mainly focus on the image classification setting.

1.1.2 The Pl@ntNet project

Created by Alexis Joly and Pierre Bonnet, Pl@ntNet (Barthélémy et al., 2011) is the meeting of two communities – botanists and computer science – to identify plant species from photos taken on the spot. The project's birth began in 2008, released its first web application in 2011, mobile application in 2013 and received in 2020 the price *Prix de l'innovation Inria – Académie des sciences – Dassault Systèmes*.

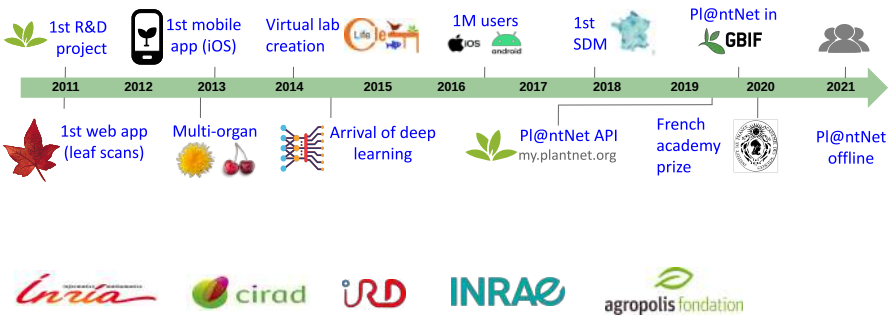


Figure 1.2 Timeline of the Pl@ntNet project

The Pl@ntNet system can be interpreted as follows. A user records an observation (an image or group of images from the same plant) and sends a query for identification. The current version of Pl@ntNet's computer vision model (referred to as Pl@ntNet's AI model and detailed in Chapter 4) makes a prediction and outputs the most probable species. The user can then agree with the AI model, or input another species. With the predicted probabilities, similar observations with indicated species are shown to the user to help them make their decision. The observation is then shared (based on the user's consent) with the community and can be voted on by other users.

Once the observation is registered, it can be revised at any time by other users. Currently, Pl@ntNet records over 50K species, 6M registered users and 21M observations spread across 77 floras. Over a billion image queries have been made, and the application has been downloaded over 10M times on Google AppStore. In total, more than 22M votes have been cast internationally. Each publicly shared observation presents, as in Figure 1.4 with an associated author, the current votes and the state of the current species determination. On the platform, users can vote for a malformed label, if the picture is about a leaf, a flower, a fruit, the bark, the habit or other. In this thesis, we only consider the species determination, not the other votes.

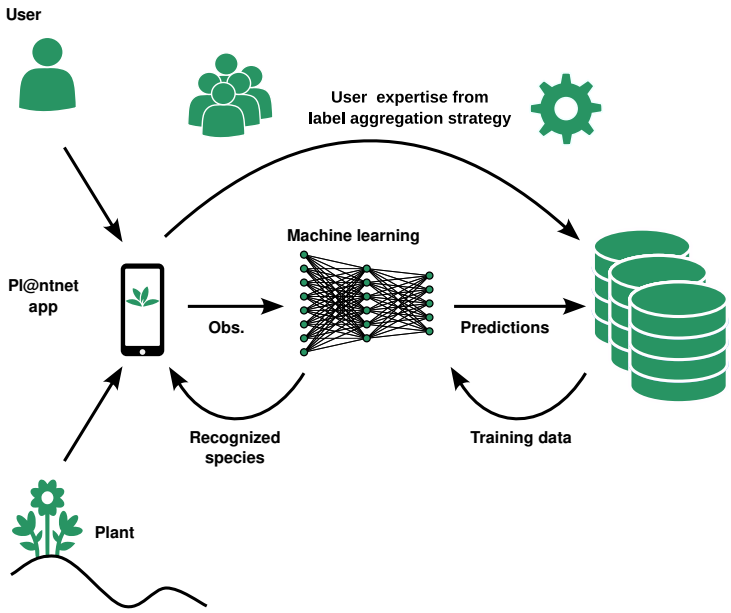
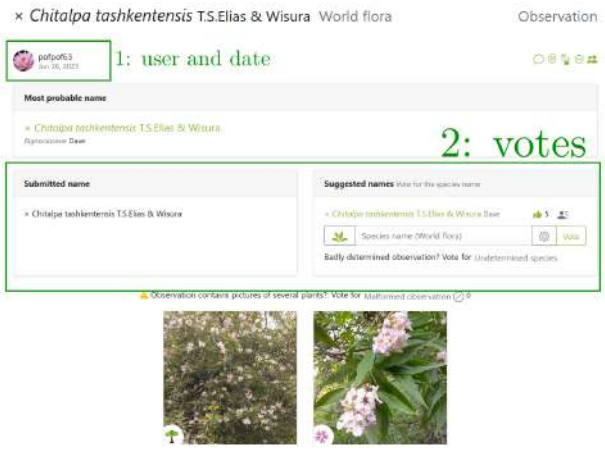
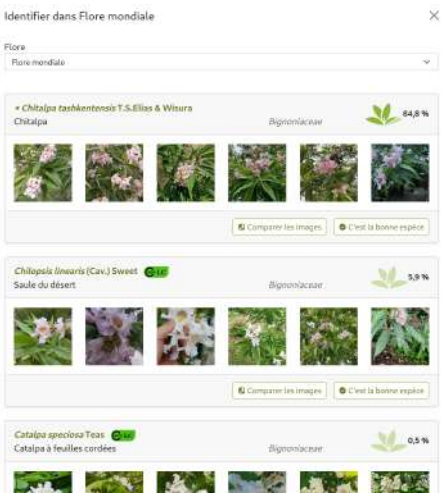


Figure 1.3 Pl@ntNet pipeline from the observation taken in the field to the active training of the computer vision model.



(a) Identification webpage. The observation is composed of two pictures, one of the organ (left) and the other of the flower (right).



(b) User vision of current model predictions. Examples for each species are proposed for visual comparison.

Figure 1.4 Example of observation from the Pl@ntNet online interface. The user shown is the author of the observation. We record the initial submitted species name, and the different votes (here 5 users agree on the label determination). If an observation's pictures do not contain the same plant, users can vote for a malformed label. By clicking on the icon near the identification field, users can see the current computer vision model prediction.

When voting, users are shown the current computer vision prediction on the observation – as shown in Figure 1.4b. This prediction influences users' votes, but also contains information. At the time of writing, Pl@ntNet's computer vision model is a DINOv2 (Oquab et al., 2024) transformer-based network. Using contrastive learning (Waida et al., 2023) *i.e.* representing similar images as close embedding to learn similar features for similar observations and then using supervised learning to finetune the model.

In Chapter 4, we present the Pl@ntNet label aggregation strategy, but also how we can incorporate model predictions into the label aggregation. Taking into account the network's vote is a delicate matter, as we don't want the AI to become overconfident in its prediction by relying mostly on its predictions and discarding human votes. Human expertise being the main goal of the platform, we want to keep humans in the loop, especially botanical experts to keep improving end-users plant identifications.

1.1.3 Regarding ethics in crowdsourcing

Crowdsourcing tasks are useful and have led to advances in research and applications. The issue is that not all crowdsourcing platforms have the same impact, and we know that there is still room for improvements to have more ethical crowdsourcing in general. We do not wish or pretend in any way to provide a fully extended analysis of ethics in this area of work. However, conducting a thesis around crowdsourced datasets without discussing worker considerations and working conditions would overlook the "where and how" the data was obtained which is not innocuous. We only offer a discussion around the ethics that should be continued until further improvements are implemented.

As stated in Schmidt and Computing (2013), crowdsourcing and worker exploitation are closely related in practice. Exploitation in itself is a loaded word, and we shall use the definition from Mayer (2007):

> *Exploiters, I will show, always inflict losses of a relative sort on disadvantaged parties. They do harm to their victims, even when their interactions are mutually advantageous, by failing to benefit the disadvantaged party as fairness requires.*

Mayer further classifies the exploitation into three subsets viewed from the fairness principle:

- exploiters do not benefit their victims at all,
- exploiters do not benefit their victims sufficiently,
- exploiters do not benefit their victims authentically.

In the case of part of the security system of chatgpt (OpenAI, 2023), OpenAI hired the company Sama to outsource examples of potential hate speech text, violence *etc.* and label them. This labeling was done by Kenyan workers in 2021, paid between 1$ 32

and 2$ an hour[5]. Here the exploiters do not benefit their victims sufficiently considering the wage[6].

Even more recently, Toloka[7] one of the largest crowdsourcing platforms, owned by Yandex, has been found collaborating with NTechLab and Tevian – *both sanctioned under the EU's human rights regime in July 2023 for contributing to the oppression and detention of protestors in Russia*[8]. Workers were not aware of the use of their annotations when drawing bounding boxes around people in images or labeling actions from surveillance videos. As the data was used to train facial recognition technology and employed for repression – monitoring and detention of journalists and activists in support of Alexei Navalny for example – the EU Council found Tevian and NTechLab to be *responsible for providing technical or material support for serious human rights violations in Russia, including arbitrary arrests or detentions, and violations or abuses of freedom of peaceful assembly and of association.*[9]. According to Annex I to Regulation (EU) 2020/1998, direct exchange of resources with these companies is thus prohibited and sanctioned. Further investigations are currently happening, however, the use of crowdsourcing data in this context is a clear example of possible malicious exploitations of knowledge and participation of workers.

The legal protection is also limited per the unusual working framework. At the European Union level, the Joint Research Centre (JRC) and the European Commission's Science and knowledge service published a report[10] to help policymakers in their decisions stating that *"current labour law framework is not aptly suited to govern new working patterns and should be revised, either on legislative or interpretative level"*. This report stresses that *"from a purely legal point of view, 'digital platform-enabled labour' does not even exist, in the sense that it is not a sort of watertight dimension of the economy and the labour market'"* as this is a very heterogeneous active field of work with difficult generalizations and conflicting perspectives between scholars, commentators and lawmakers.

Data privacy has also been of concern with the growing popularity of some platforms. In Lease et al. (2013), Personal Identity Identification (PII) exposure factors have been identified on the Amazon Mechanical Platform – that is considered by many as fully anonymous. On this platform, each worker is assigned a unique identification string of both letters and numbers. This string was provided in the final dataset and sometimes

[5] https://time.com/6247678/openai-chatgpt-kenya-workers/

[6] There is currently no universal minimum wage in Kenya, however in 2022 a cashier would earn in average 309.06Ksh/hour in Nairobi *i.e.* around $2.14/hour: https://africapay.org/kenya/salary/minimum-wages/2182-cities-nairobi-mombasa-and-kisumu

[7] https://toloka.ai/

[8] https://www.thebureauinvestigates.com/stories/2024-03-27/paid-pennies-to-train-tools-of-repression-the-humans-behind-moscows-state-surveillance/

[9] https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32023R1495

[10] https://publications.jrc.ec.europa.eu/repository/bitstream/JRC112243/jrc112243_legal_framework_digital_labour_platforms_final.pdf

released publicly – as a dataset or in figures as in Gao et al. (2015). However, this string added to a specific URL was[11] the access to the worker profile. Access to the profile meant access to names, ages, location, wish lists for purchases, and other demographic information.

Finally, we might wonder: if paid crowdsourcing brings such social and law-related issues, what about unpaid crowdsourcing – used with Pl@ntNet? And once again, we fall back to a case-by-case situation with the lack of regulations in this area. New questions about vigilantism have risen with crowdsourcing platforms like BlueServo[12] in Texas or EuropeanBorderWatch[13]. These platforms allow users to watch surveillance cameras near borders and monitor suspicious criminal activities – immigration related mainly. As stated in Schmidt and Computing (2013), *"this certainly brings up all kinds of ethical questions regarding the motives of those watching and reporting, the effect on those who should be paid doing these jobs and regarding those being named and shamed, potentially innocently, by a vigilant cyber mob"*. Unpaid crowdsourcing also often comes in the form of games to keep workers in the loop and gather more data. Rogl (2016) states that with this new form of labor, the risk is for workers not to be able to separate their work from these side leisure leading to problems in evaluating their working worth and compensation. Following the classification of exploitation, the question is: does this form of unpaid work fall into one of the three subsets, does it benefit the workers? In Nyström (2021), an evaluation experiment was designed to evaluate gamified tasks: the Darkness of Gamification Evaluation System. Previous work such as Pirkkalainen and Salo (2016) identified four darkness phenomena: information overload, technostress, IT anxiety, and IT addiction. Nyström (2021) concludes that gamification *"could contribute to all four, thus, the designing of gamification is crucial to avoid darkness that impacts the user negatively"*.

Within this thesis, we did not conduct any new crowdsourcing experiments. We used datasets from other research projects publicly available. The Pl@ntNet data collection process – explained in Chapter 4 – is a form of unpaid crowdsourcing where users know that their participation helps current research (known as explicit crowdsourcing, see Appendix A for more details).

1.2 Problematic and thesis organization

1.2.1 Organization of the thesis

In this thesis, we deal with image classification problems in crowdsourcing. The main setting we consider is: *How to learn from crowdsourced labels?* As this is a large problem

[11] currently this privacy seems to have been fixed, but it was identified in 2012 and was still active in 2019 `https://www.reddit.com/r/mturk/comments/bb16w8/separate_account_for_selling_on_amazon/`

[12] `https://www.blueservo.net/`

[13] `http://www.europeanborderwatch.org/`

we want to focus on the data quality we learn our classifiers from, especially in a context with few labels given per task. The literature has mainly considered modeling workers abilities and faults (Dawid and Skene, 1979; Sinha et al., 2018; Rodrigues and Pereira, 2018; Raykar and Yu, 2011), but very few models incorporate the tasks' difficulty (Whitehill et al., 2009; Chu et al., 2021). Moreover, when this difficulty is taken into account, it is often done without considering the actual images, only the answered labels (Whitehill et al., 2009). In this work, we first propose to identify ambiguous tasks in training datasets from the crowdsourced labels and the actual image with a metric called the WAUM introduced by Lefort et al. (2022). Defining what is a difficult task proves to be a challenge in itself. An informal definition of a difficult task is given by Angelova (2004):

> *Difficult examples are those which obstruct the learning process or mislead the learning algorithm or those which are impossible to reconcile with the rest of the examples. Defining difficult examples cannot be done without the learning model or the generating distribution.*

In this work, we focus on the tempering of the learning process. Our WAUM heavily relies on how easy it is for a model to learn the given task. Furthermore, identifying possible ambiguous tasks lets us better explore large crowdsourced datasets and consider strategies to improve learning performance. In addition, the work of Merler et al. (2004) and more recently Pleiss et al. (2020) show significant learning performance of models after pruning most unreliable tasks from the datasets. We show that similar improvements can be made in the crowdsourcing setting. In chapter 3, we present our open-source `python` library `peerannot` to ease and help standardize how to deal with crowdsourced datasets in image classification. We handle label aggregation strategies, learning strategies and identification processes of ambiguous tasks and/or poorly performing workers. We also propose a framework to evaluate multiple datasets and aggregation strategies with the `Benchopt` framework. Finally, in chapter 4 we compare label aggregation strategies on a large subset of Pl@ntNet's south-western European database we released. This subset is challenging as it contains more than 6 million tasks and over 800 thousand workers with a large number of classes. We investigate Pl@ntNet's current label aggregation strategy performance and how we could improve it using the current computer vision model predictions.

1.2.2 Notation

Classical supervised setting. Let us first reconsider classical supervised learning in a classification setting. A dataset $\mathcal{D} = \{(x_i, y_i^\star)\}_{i=1}^n$ is composed of n tasks $x_i \in \mathcal{X}$ and $y_i^\star \in \mathcal{Y}$. In image classification, $\mathcal{X}$ is classically the space of RGB images and $\mathcal{Y}$ is the set of possible labels. The possible labels are encoded as numbers, thus in a classification problem with K classes, $\mathcal{Y} = [K] = \{1, \dots, K\}$. The cardinal of the set $\mathcal{Y}$ is written as

$|\mathcal{Y}| = K$. To classify images, we need a classifier denoted $\mathcal{C}$. Then, given a task $x_i \in \mathcal{X}$, $\mathcal{C}(x_i) \in \mathbb{R}^K$ represents the vector of scores. To transform these scores into probabilities, we use the softmax function $\sigma(\cdot)$ defined as:

$$\sigma(z) = \left(\frac{e^{z_k}}{\sum_{\ell \in [K]} e^{z_\ell}} \right)_{k \in [K]} .$$

The probabilities predicted by the classifier sorted in decreasing order are

$$\sigma(\mathcal{C}(x_i))_{[1]} \geq \sigma(\mathcal{C}(x_i))_{[2]} \geq \cdots \geq \sigma(\mathcal{C}(x_i))_{[K]} \ ,$$

where $\sigma(\mathcal{C}(x_i))_{[1]}$ represents the predicted class for the task by the classifier. The indicator function is denoted $\mathbb{1}(\cdot)$. The simplex of dimension $K-1$ is written as $\Delta_K = \{p \in \mathbb{R}_+^K | \sum_{k=1}^K p_k = 1\}$. And finally, given a vector $z \in \mathbb{R}^K$, we denote the normalization operator $\mathbf{N}$ such that $\mathbf{N}(z) = z / \sum_{k \in [K]} z_k$.

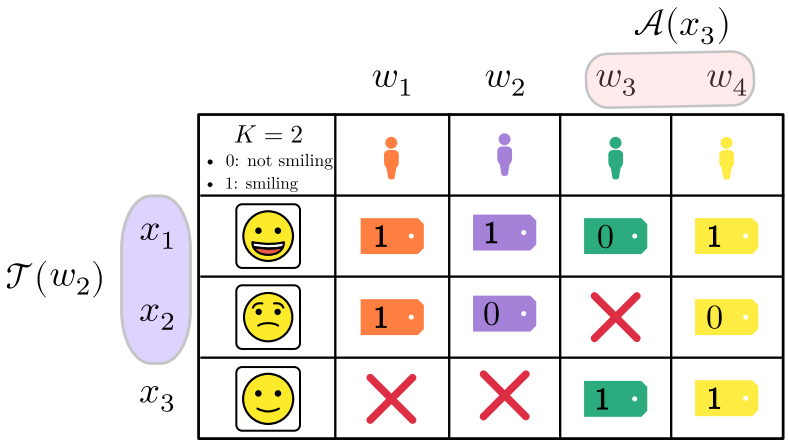


Figure 1.5 Crowdsourcing notation on a toy example. There are three tasks to classify into $K = 2$ groups. Only worker w_4 answers all tasks. Only the task x_1 is answered by all workers. The workerload of w_2 is $|\mathcal{T}(w_2)| = |\{1, 2\}| = 2$. The feedback effort on x_3 is $|\mathcal{A}(x_3)| = |\{3, 4\}| = 2$.

Crowdsourcing setting. With crowdsourced data, the ground truth $y_i^\star$ of a task x_i is unknown. Instead, there is a crowd of $n_{\text{worker}} \in \mathbb{N}^\star$ labeling the n_{task} training tasks – we assume access to a classical test and validation set. Each worker $w_j, j \in [n_{\text{worker}}]$ answers at least one task by giving a label denoted $y_i^{(j)} \in [K]$. The set of workers answering the task x_i is denoted by

$$\mathcal{A}(x_i) = \{j \in [n_{\text{worker}}] : w_j \text{ answered } x_i\} . \tag{1.1}$$

We call **feedback effort** the number of labels for a given task: $|\mathcal{A}(x_i)|$. 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

$$\mathcal{T}(w_j) = \{i \in [n_{\text{task}}] : w_j \text{ answered } x_i\} . \tag{1.2}$$

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

$$\mathcal{D}_{\text{train}} := \bigcup_{i \in [n_{\text{task}}]} \left\{ (x_i, (y_i^{(j)})) \text{ for } j \in \mathcal{A}(x_i) \right\} = \bigcup_{j \in [n_{\text{worker}}]} \left\{ (x_i, (y_i^{(j)})) \text{ for } i \in \mathcal{T}(w_j) \right\}. \tag{1.3}$$

Note that we use the index notation i for the tasks and j for the workers in the crowdsourcing experiments.

1.2.3 Existing label aggregation strategies

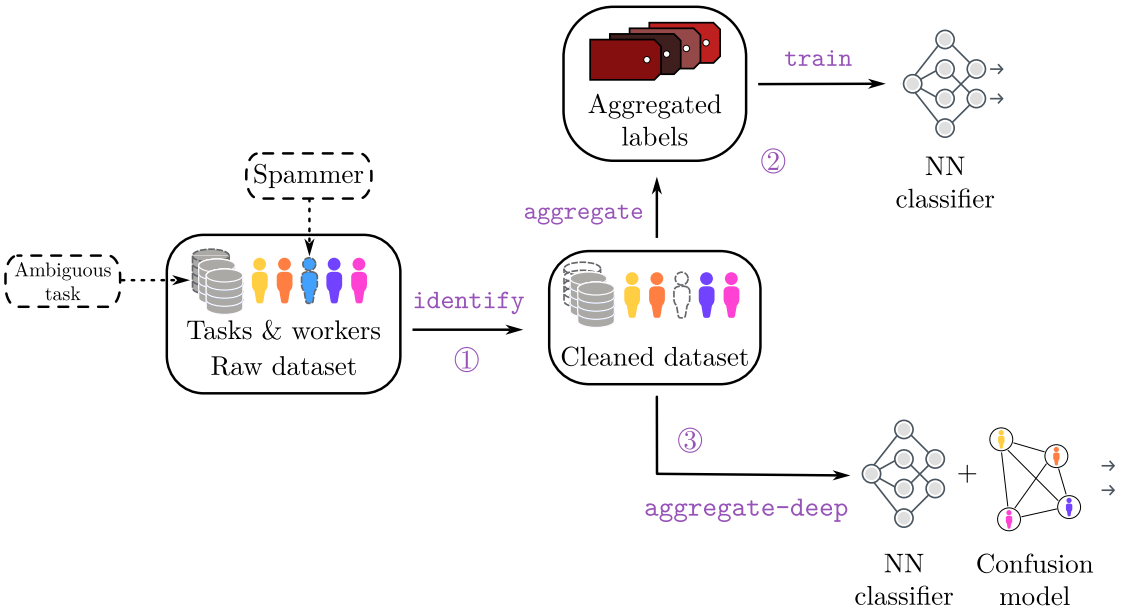


Figure 1.6 pipleine of crowdsourcing experiments. The data is collected. We identify poorly performing workers and/or ambiguous tasks. Then labels are aggregated to be used in a supervised learning setting. If the tasks are available and the goal is to learn a predictor, deep-learning-based strategies can be used to learn the confusion model and the classifier at the same time.

If the goal of the crowdsourcing experiment is to associate a label with a task in the training set, then we need to aggregate the collected labels into a single one. This aggregation step can lead to two types of labels:

- hard labels: the aggregated label $\hat{y}_i \in [K]$ represents a single class – it can be seen as a Dirac distribution over $\mathcal{Y}$.
- soft labels: the aggregated label $\hat{y}_i \in \Delta_K$ is a probability distribution over $\mathcal{Y}$ that is not necessarily a Dirac. This type of label keeps more of the uncertainty for the learning procedure.

Note that if the aggregation step produces a soft label, it is always possible to produce a hard label taking the mode of the distribution – *i.e.* using the argmax function.

In the following, we present classical aggregation strategies. This is not an exhaustive list, only models we mostly use for comparison. Assume that a dataset $\left(x_i, \{y_i^{(j)}\}_{j\in\mathcal{A}(x_i)}\right)_{i\in[n_{\text{task}}]}$ is available. We only need to consider the proposed labels – *i.e.* the tasks x_i are not used. These models – and more – are available in the library `peerannot` using the `aggregate` command shown in Figure 1.6.

Majority vote (MV)

The majority vote strategy is the simplest mechanism to create a label from multiple answers. It consists in collecting the votes and taking the most answered category. In the case of a draw, a random choice in the most answered choices is applied. More formally:

$$\forall i \in [n_{\text{task}}],\ \hat{y}_i^{\text{MV}} = \underset{k\in[K]}{\text{argmax}} \left(\sum_{j\in\mathcal{A}(x_i)} \mathbb{1}(y_i^{(j)} = k) \right)_{k\in[K]} . \tag{1.4}$$

The main drawback of the majority vote is the lack of consideration of workers' abilities. Intrinsically, this aggregation strategy assumes that all workers are equal. However, experiments have shown that this is not true (Snow et al., 2008; Vuurens et al., 2011). The MV strategy is one of the most theoretically studied thanks to its simplicity. Wang and Zhou (2015) found that the label error rate decreases exponentially with the number of workers selected for each task. This assumes that the worker quality is bounded and the workers are selected uniformly in the crowd. In the same work, they also provide similar results with worker quality characterized by their specificity – true negative rate – and sensitivity – true positive rate – with Gaussian-like assumptions instead of bounded qualities. Another issue with the MV strategy is that by taking the mode of the votes' distribution, the labeling uncertainty completely disappears (see Figure 1.7).

Furthermore, the majority vote is sensitive to a specific type of attack on the data: **spams**. A spammer is defined as a worker whose answers are given independently of the underlying ground truth (Raykar and Yu, 2011). More formally:

$$\forall i \in [n_{\text{task}}] \forall (k,c) \in [K]^2,\ \mathbb{P}(y_i^{(j)} = k \mid y_i^\star = c) = \mathbb{P}(y_i^{(j)} = k) \ . \tag{1.5}$$

Vuurens et al. (2011) discusses how critical it is to identify spammers to improve labeling quality. They show through several simulation settings that even with trapping systems, the majority vote is still susceptible to spam.

In practice, the MV strategy is most often used as a baseline strategy. Note that some taskmasters improved majority vote accuracy using prevention systems that are not considered throughout this work. Hoang et al. (2021) for example uses a vouching system where volunteer workers can be approved via specific institutional email domains.

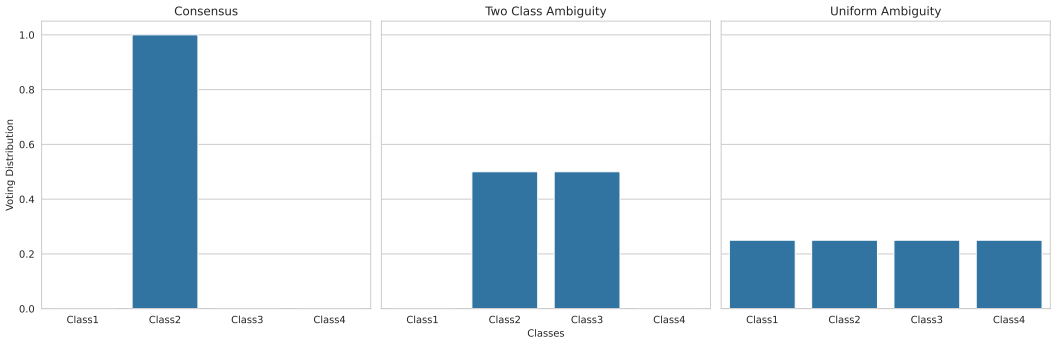


Figure 1.7 Three different configurations where the MV strategy can lead to the same label. When there is consensus there is no ambiguity to select the label. However, ambiguities can happen differently. There can be ambiguities between all classes or between a subset of classes.

Khattak (2017) and Peterson et al. (2019) use trapping sets – *i.e.* separate set of tasks where a ground truth is known. This set is considered to evaluate worker abilities and identify poorly performing workers, or potential spammers. Vuurens et al. (2011) showed that these sets lead to still sensitive decisions regarding other types of spammers.

Naive Soft (NS)

To compensate for the loss of uncertainty of the MV strategy, the simplest solution is not to consider the mode of the voting distribution but to keep the full distribution as a label. The Naive Soft (NS) strategy creates a simple soft label in the simplex from the frequency of votes for each class.

$$\forall i \in [n_{\text{task}}],\ \hat{y}_i^{\text{NS}} = \mathbf{N}\left(\sum_{j \in \mathcal{A}(x_i)} \mathbb{1}(y_i^{(j)} = k)\right)_{k \in [K]} . \tag{1.6}$$

Recent work (Collins et al., 2022) showed that with enough votes collected, naive soft labels do lead in general to better probability outputs in predictions (in terms of calibration error – see the ECE error in Section 1.2.5 for more details) compared to hard labels. However, this strategy still considers every worker equal in front of the tasks proposed like the MV, thus does not alleviate any ability score, for example not to be corrupted by spammers.

Dawid and Skene (DS)

The Dawid and Skene's (DS) strategy (Dawid and Skene, 1979) introduced a framework where each worker is represented by its confusion. This way, it is the first strategy to take into account worker abilities for each possible class. Through their model, they estimate simultaneously the worker confusion and the tasks' soft label. With its theoretical guarantees (Gao and Zhou, 2013) and its flexibility in the sense that it is easy to modify

it to take into account data specificities (Passonneau and Carpenter, 2014; Servajean et al., 2017; Sinha et al., 2018), the DS strategy remains a strong competitor in many cases.

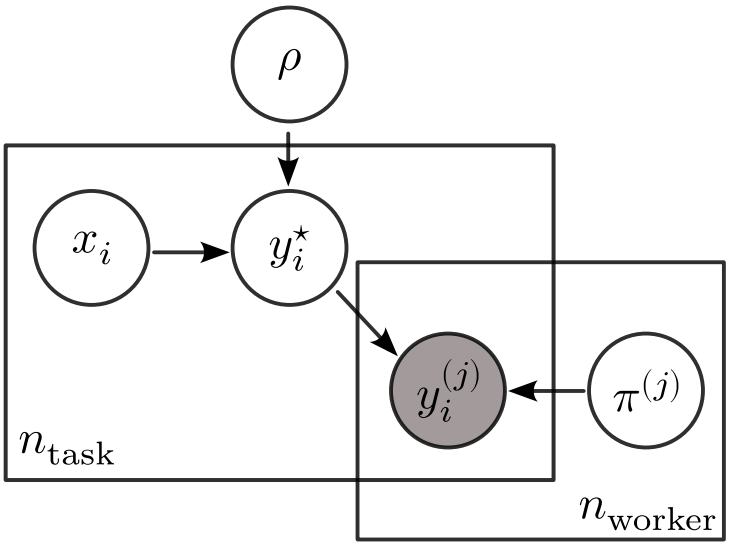


Figure 1.8 Bayesian plate diagram representation of the DS model. Only the labels $\{y_i^{(j)}\}_{i,j}$ are observed. Latent variables to estimate are the true labels $(y_i^\star)_i$, the prevalence ρ of each class and the confusion matrices $\{\pi^{(j)}\}_j$ for each worker. The underlying task features $(x_i)_i$ are not considered.

The model keystone of the DS model is the modeling of the workers' answers. Each worker is associated to its confusion matrix $\pi^{(j)} \in \mathbb{R}^{K\times K}$ such that the (k,ℓ)-entry represents the probability for worker w_j to answer $\ell \in [K]$ when the ground truth is $k \in [K]$. Then, we assume that each worker's answer is drawn from a multinomial distribution over the k^{th} row of its confusion matrix. More formally:

$$(y_i^{(j)} \ | y_i^\star = k) \sim \mathcal{M}\left(\pi_{k,\cdot}^{(j)}\right) \tag{1.7}$$

Given the probabilistic model, we wish to maximize the likelihood concerning the confusion matrices, the prevalence of each class in the dataset and the underlying true labels. Assume that the ground truths are sampled from a multinomial distribution parameterized by the class prevalence $\rho \in \Delta_K$ – *i.e.* for $k \in [K]$, $\mathbb{P}(y_i^\star = k) = \rho_k$, and that workers answer independently from one another. Denote $T_{i,k} = \mathbb{1}(y_i^\star = k)$ the indicator of the true label. Then the maximum likelihood estimation of the DS model writes:

$$\underset{\rho,\pi,T}{\arg\max} \prod_{i\in[n_{\texttt{task}}]} \prod_{k\in[K]} \left[\rho_k \prod_{j\in[n_{\texttt{worker}}]} \prod_{\ell\in[K]} \left(\pi_{k,\ell}^{(j)}\right)^{\mathbb{1}_{\{y_i^{(j)}=\ell\}}}\right]^{T_{i,k}} . \tag{1.8}$$

Construction of the DS likelihood

Given a single task i with true label $y_i^\star = k$ and one worker w_j, then responses of worker j follow a multinomial distribution by Equation (1.7) and the likelihood of the observed data $\{y_i^{(j)}\}_{i,j}$ is proportional to:

$$\prod_{\ell \in [K]} \left(\pi_{k,\ell}^{(j)}\right)^{\mathbb{1}_{\{y_i^{(j)}=\ell\}}} \quad .$$

Then, conditionally on the true label being k – *i.e.* $T_{i,k} = 1$ and for $k' \neq k$, $T_{i,k'} = 0$, we can use the workers' independence assumption to consider multiple workers:

$$\prod_{j \in [n_{\text{worker}}]} \prod_{\ell \in [K]} \left(\pi_{k,\ell}^{(j)}\right)^{\mathbb{1}_{\{y_i^{(j)}=\ell\}}} \quad .$$

Removing the conditional assumption on the true label, we take into account the class prior contained in the prevalence vector ρ:

$$\prod_{k \in [K]} \left[\rho_k \prod_{j \in [n_{\text{worker}}]} \prod_{\ell \in [K]} \left(\pi_{k,\ell}^{(j)}\right)^{\mathbb{1}_{\{y_i^{(j)}=\ell\}}} \right]^{T_{i,k}} \quad .$$

Finally, as all tasks are independent, we obtain Equation (1.8) by multiplying over the task index.

Note that the likelihood in Equation (1.8) can not be maximized directly. Using the expectation-maximization (EM) algorithm (Dempster et al., 1977), we can define a recursive procedure to estimate $\rho, \pi = \{\pi^{(j)}\}_j$ and $T = \{T_{i,k}\}_{i,k}$.

- Assuming that T is known, compute ρ and π.
- Assuming that ρ and π are known, compute T.

With T known: Denote $Y = \{y_i^{(j)}\}_{i,j}$ for readability. The logarithm of the likelihood function writes:

$$\log \text{Lik}(\pi, \rho | T, Y) = \sum_i \sum_k \left[T_{i,k} \log(\rho_k) \sum_j \sum_\ell \mathbb{1}_{\{y_i^{(j)}=\ell\}} \log(\pi_{k,\ell}^{(j)}) \right] \quad .$$

To compute π and ρ knowing T, we use the Lagrangian multipliers method. Let $\lambda = \{\lambda^{(j)}\}_j$ the Lagrange multipliers for the constraint $\sum_\ell \pi_{k,\ell}^{(j)} = 1$ – sum of probabilities for each row of confusion matrices equals 1 – and η the Lagrange multiplier for the constraint $\sum_k \rho_k = 1$ – the prevalence must sum to 1. Then the full Lagrangian $\mathcal{L}$ writes:

$$\mathcal{L}(\pi, \rho, \lambda, \eta) = \log \text{Lik}(\pi, \rho | T, Y) + \eta \sum_i \left[\sum_k T_{i,k} \log(\rho_k) - 1 \right] + \sum_j \lambda_j \sum_\ell \left[\pi_{k,\ell}^{(j)} - 1 \right] \quad .$$

- Differentiation with respect to $\pi_{k,\ell}^{(j)}, (k,\ell) \in [K]^2$:

$$\frac{\partial \mathcal{L}}{\partial \pi_{k,\ell}^{(j)}} = \sum_i \frac{T_{i,k}}{\pi_{k,\ell}^{(j)}} \mathbb{1}_{\{y_i^{(j)}=\ell\}} - \lambda_j \ ,$$

and setting it to zero leads to:

$$\pi_{k,\ell}^{(j)} = \frac{1}{\lambda_j} \sum_i T_{i,k} \mathbb{1}_{\{y_i^{(j)}=\ell\}} \ . \tag{1.9}$$

- Differentiation with respect to $\rho_k, k \in [K]$:

$$\frac{\partial \mathcal{L}}{\partial \rho_k} = \sum_i \frac{T_{i,k}}{\rho_k} - \eta \ ,$$

and setting the derivative to zero leads to:

$$\rho_k = \frac{1}{\eta} \sum_i T_{i,k} \ . \tag{1.10}$$

- Differentiating with respect to λ_j and η recovers the constraints conditions. Taking into account the constraints we finally obtain:

$$\lambda_j = \sum_{\ell'} \sum_i T_{i,k} \mathbb{1}_{\{y_i^{(j)}=\ell'\}} \quad \text{and} \quad \eta = n_{\text{task}} \tag{1.11}$$

Combining Equation (1.11) with Equation (1.9) and Equation (1.10), we obtain the update rule for π and ρ with T known.

Update rule for T: We now estimate T with π and ρ known using Bayes formula. Recall that $T_{i,k}$ represents the posterior probability of task i to belong in class k given the observed labels Y, the class prevalence ρ and the confusions π. Then:

$$\begin{aligned}
\mathbb{P}(T_{i,k} = 1 | Y, \pi, \rho) &\propto \mathbb{P}(Y, T_{i,k} = 1 | \pi, \rho) \\
&\propto \mathbb{P}(T_{i,k} = 1 | \rho) \mathbb{P}(\{y_i^{(j)}\}_j | \pi, T_{i,k} = 1) \\
&\propto \rho_k \prod_j \prod_\ell \left(\pi_{k,\ell}^{(j)}\right)^{\mathbb{1}_{\{y_i^{(j)}=\ell\}}} \ .
\end{aligned} \tag{1.12}$$

Using our updates rules, we can now write the full EM optimization for the DS model in Algorithm 1.

At this point, a few clarifications are needed for Algorithm 1. The initialization used for the estimated labels T is the NS strategy Section 1.2.3 Page 13, however, other initialization exist and are discussed in Dawid and Skene (1979). Numerically, as there is no single optimum, discrepancies in results might happen. Furthermore, let us discuss the stopping criterion. Given a precision $\epsilon > 0$, a possible criterion to stop is if, between two consecutive iterates, the likelihood did not go further than ϵ in absolute value. Other criteria could be applied, like relative error instead of absolute, but often do not impact results by much.

Algorithm 1 Expectation-Maximization (EM) for Dawid and Skene Model

Initialize T with NS strategy
while Not converged **do**
  // M-step: Maximization step
  for $k = 1$ to K **do**
    for $\ell = 1$ to K **do**
      Update $\pi_{k,\ell}$ using Equation (1.9)
    Update ρ_k using Equation (1.10)
  Normalize $P(T_{i,\cdot} = 1|Y, \pi, \rho)$
  // E-step: Expectation step
  for $i = 1$ to n_{task} **do**
    for $k = 1$ to K **do**
      Compute posterior $P(T_{ik} = 1|y, \pi, \rho)$ using Equation (1.12)

Weighted with Dawid and Skene (WDS)

The DS model, by design, often leads to soft labels $(T_{i,\cdot})_i$ with very low entropy distribution. Indeed, this variable represents the indicator function of belonging to a given class. To keep more uncertainty, one possibility is to shift our focus to only consider the diagonal of the confusion matrices estimated in the DS model. Each term on the diagonal represents the ability of the worker to answer correctly the underlying label. Using this DS diagonal, we obtain a weighted majority vote denoted WDS:

$$\hat{y}_i^{\text{WDS}} = \left(\sum_{j \in \mathcal{A}(x_i)} \pi_{k,k}^{(j)} \mathbb{1}_{\{y_i^{(j)}=k\}} \right)_{k \in [K]} . \tag{1.13}$$

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

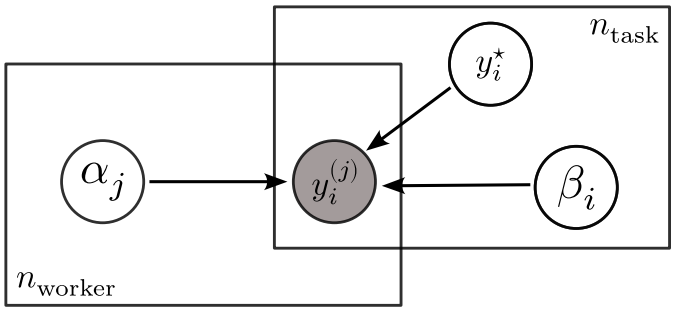


Figure 1.9 Bayesian plate diagram representation of the GLAD model. Only the labels $\{y_i^{(j)}\}_{i,j}$ are observed. Parameters to estimate are the true labels $(y_i^\star)_i$, the worker ability α_j and the task difficulty β_i. The underlying task features $(x_i)_i$ are not considered.

While DS-like models focused on evaluating workers' confusion, Whitehill et al. (2009) introduced GLAD to take into account two sources of error: the worker's ability and the task's difficulty. The worker ability is defined as a parameter $\alpha_j \in \mathbb{R}$. The larger the more reliable a worker is. When $\alpha_j = 0$, the worker answers randomly, whereas when $\alpha_j < 0$, the worker is adversarial and switches labels. The task difficulty is denoted using the parameter $\beta_i \in \mathbb{R}_+^\star$. If $\beta_i \simeq 0$, the task is almost impossible to classify correctly. When β_i is large, the task is easy to classify. GLAD's model is based on a logistic regression. Indeed, the model assumes that the probability for a worker to answer correctly the label $y_i^\star = k$ is generated by a sigmoid function:

$$\mathbb{P}(y_i^{(j)} = k | y_i^\star = k, \alpha_j, \beta_i) = \sigma(\alpha_j \beta_i) \ . \tag{1.14}$$

Equation (1.14) implies that the log-odds to label correctly a task is defined as $\alpha_j \beta_i$: a bilinear function of the worker's ability and the task difficulty. Figure 1.9 represents the plate diagram representation of the model and shows that there is no causal structure between the true labels $y_i^\star$ and the parameters α_j or β_i.

To estimate the worker's abilities and task difficulties, we use the maximum likelihood estimator. We also assume that errors are uniform across possible other labels. Then the likelihood is proportional to:

$$\prod_{i \in [n_{\text{task}}]} \prod_{k \in [K]} \mathbb{P}(y_i^\star = k) \prod_{j \in [n_{\text{worker}}]} \left(\frac{1}{K-1} \left(1 - \sigma(\alpha_j \beta_i)\right) \right)^{1 - \mathbb{1}_{\{y_i^{(j)} = k\}}} \sigma(\alpha_j \beta_i)^{\mathbb{1}_{\{y_i^{(j)} = k\}}} \ . \tag{1.15}$$

Construction of the GLAD likelihood

Given a single task i with true label $y_i^\star = k$ and one worker w_j, then responses of worker j follow a Bernoulli distribution to be correct by Equation (1.14) and the likelihood of the observed data $\{y_i^{(j)}\}_{i,j}$ is proportional to:

$$\sigma(\alpha_j \beta_i)^{\mathbb{1}_{\{y_i^{(j)} = k\}}} \left(\frac{1}{K-1} \left(1 - \sigma(\alpha_j \beta_i)\right) \right)^{1 - \mathbb{1}_{\{y_i^{(j)} = k\}}} \ .$$

Then, using the workers' independence assumption to consider multiple workers:

$$\prod_{j \in [n_{\text{worker}}]} \sigma(\alpha_j \beta_i)^{\mathbb{1}_{\{y_i^{(j)} = k\}}} \left(\frac{1}{K-1} \left(1 - \sigma(\alpha_j \beta_i)\right) \right)^{1 - \mathbb{1}_{\{y_i^{(j)} = k\}}} \ .$$

Removing the conditional assumption on the true label:

$$\prod_{k \in [K]} \mathbb{P}(y_i^\star = k) \prod_{j \in [n_{\text{worker}}]} \sigma(\alpha_j \beta_i)^{\mathbb{1}_{\{y_i^{(j)} = k\}}} \left(\frac{1}{K-1} \left(1 - \sigma(\alpha_j \beta_i)\right) \right)^{1 - \mathbb{1}_{\{y_i^{(j)} = k\}}} \ .$$

Finally, as all tasks are independent, we obtain Equation (1.15) by multiplying over the task index.

The likelihood is maximized using the EM algorithm. However, due to the lack of analytical solutions in the maximization step, we also compute the derivatives of the auxiliary function to maximize it using a first-order optimization method (like gradient descent). Let us compute the auxiliary function Q and its gradient for α and β. Denote $Y = \{y_i^{(j)}\}_{i,j}$, $Y^\star = \{y_i^\star\}_i$ and $p^k = \mathbb{P}(y_i^\star = k|Y, \alpha^{\text{old}}, \beta^{\text{old}})$ the latest estimation of the posterior distribution available, then:

$$\begin{aligned} Q(\alpha, \beta) &= \mathbb{E}\left[\log \mathbb{P}(Y, Y^\star|\alpha, \beta)\right] = \sum_i \mathbb{E}\left[\log \mathbb{P}(y_i^\star)\right] + \sum_{i,j} \mathbb{E}\left[\log \mathbb{P}(y_i^{(j)}|y_i^\star, \alpha_j, \beta_i)\right] \\ &= \sum_{i,k} p^k \log \mathbb{P}(y_i^\star = k) + \sum_{i,j,k} p^k \log \mathbb{P}(y_i^{(j)}|y_i^\star = k, \alpha_j, \beta_i) \ . \end{aligned} \tag{1.16}$$

Taking the derivative for α_j:

$$\begin{aligned} \frac{\partial Q}{\partial \alpha_j} &= \sum_{i,k} p^k \frac{\partial}{\partial \alpha_j} \left[\mathbb{1}_{\{y_i^{(j)}=k\}} \log \sigma(\alpha_j \beta_i) + \left(1 - \mathbb{1}_{\{y_i^{(j)}=k\}}\right) \left(\log \sigma(\alpha_j \beta_i) - \log(K-1)\right)\right] \\ &= \sum_{i,k} p^k \left[\mathbb{1}_{\{y_i^{(j)}=k\}} \beta_i \left(1 - \sigma(\alpha_j \beta_i)\right) - \left(1 - \mathbb{1}_{\{y_i^{(j)}=k\}}\right) \beta_i \sigma(\alpha_j \beta_i)\right] \\ &= \sum_{i,k} p^k \beta_i \left(\mathbb{1}_{\{y_i^{(j)}=k\}} - \sigma(\alpha_j \beta_i)\right) \ , \end{aligned}$$

using that for $x \in \mathbb{R}$, $\sigma(x) = 1 - \sigma(-x)$. Symmetrical reasoning leads to:

$$\frac{\partial Q}{\partial \beta_i} = \sum_{j,k} p^k \alpha_j \left(\mathbb{1}_{\{y_i^{(j)}=k\}} - \sigma(\alpha_j \beta_i)\right) \ .$$

The posterior probability for the ground truth labels can be computed directly using the plate diagram in Figure 1.9, Equation (1.14) and Bayes' theorem:

$$\begin{aligned} \mathbb{P}(y_i^\star|Y, \alpha, \beta) &\propto \mathbb{P}(y_i^\star|\alpha, \beta_i) \mathbb{P}(Y_j|y_i^\star, \alpha, \beta_i) \\ &\propto \mathbb{P}(y_i^\star) \prod_{j \in \mathcal{A}(x_i)} \mathbb{P}(y_i^{(j)}|y_i^\star, \alpha_j, \beta_i) \ . \end{aligned} \tag{1.17}$$

Aggregations that are not considered. In this work, we do not consider modifying the data collection model, only the aggregation strategy. However, it is important to note that the data collection model can be modified to improve the quality of the labels. For example, Khattak (2017) uses trapping sets to evaluate workers and reweight their involvement in the final aggregation step. Other work such as Chamberlain et al. (2018) considers a validation process to improve crowdsourced data quality to balance the need for more labels. In opposition to the **Annotation Mode** on their platform `Phrase Detectives`, users have access to a **Validation Mode** where they can have access to tasks, and agree or disagree with answers. Hoang et al. (2021) uses a vouching system where volunteer workers can be approved via specific institutional email domains. Aggregation strategies that rely on these data collection models are not considered in this work.

Algorithm 2 GLAD (EM version)

Input: $\mathcal{D}_{\text{train}}$: crowdsourced dataset

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

1: **while** Likelihood not converged **do**
2: Estimate probability of $y_i^\star$:
3: $\forall i \in [n_{\texttt{task}}],\ \mathbb{P}(y_i^\star|\{y_i^{(j)}\}_i, \alpha, \beta_i) \propto \mathbb{P}(y_i^\star)\prod_j \mathbb{P}(y_i^{(j)}|y_i^\star, \alpha_j, \beta_i)$
4: Maximization step:
5: Maximize auxiliary function $Q(\alpha, \beta)$ in Equation (1.16) w.r.t α and β

1.2.4 Integrating crowdsourcing in the neural network's architecture

In the era of deep learning, it is only expected to find new ways to introduce crowdsourcing into neural network architectures. Incorporating layers taking into account the crowd/tasks pairing in the architecture leads to a neural network able to be trained on crowdsourced data. The newly obtained classifier denoted $\mathcal{C}$ in the future, is used to predict labels of unseen images. However, contrary to models seen in Section 1.2.3, these strategies are only used for prediction and not for creating a new label for each task. We thus not only need the crowdsourced labels but also the associated tasks and a set of unseen data to validate/test the model on.

CrowdLayer and its matrix weights strategy (MW)-version

From (Rodrigues and Pereira, 2018), CrowdLayer is an end-to-end strategy in the crowdsourcing setting. From the output of a neural network, a new layer called *crowd layer* is added to take into account worker specificities. The main classifier thus becomes globally shared, and the new layer is the only worker-aware layer. As multiple variants of CrowdLayer can exist, we only considered in this paper the matrix weights (MW) strategy that is akin to the DS model. Denoting $z = f(x_i)$ the output of the neural network classifier f for a given task x_i labeled by a worker w_j, the added layer multiplies z by a matrix of weights $W^j \in \mathbb{R}^{K\times K}$. This matrix of weights per worker takes into account the local confusion of each worker. In practice, the forward pass F on a task x_i annotated by worker w_j using CrowdLayer computes $F(x_i, w_j) = W^j\sigma(f(x_i))$.

Also note that in practice, other CrowdLayer strategies are available (using a single scalar instead of a matrix for example) and that the MW strategy can be used with a regularization term to avoid overfitting on the workers' specificities (Tanno et al., 2019).

Common Noise Adaptation Layers (CoNAL)

CrowdLayer takes into account worker-specific confusion matrices. CoNAL (Chu et al., 2021) generalizes this setting by creating a global confusion matrix $W^g \in \mathbb{R}^{K\times K}$ in

addition to the local ones $W^j \in \mathbb{R}^{K\times K}$ for $j \in [n_{\texttt{worker}}]$ working all together with the classifier f. Given a worker w_j, the confusion is global with weight ω_i^j and local with weight $1-\omega_i^j$. The final distribution output used to compute the loss is given by:

$$p_{\text{out}}(x_i, w_j) = \omega_i^j W^g f(x_i) + (1-\omega_i^j) W^j f(x_i) \ .$$

As is, CoNAL local matrices tend to aggregate themselves onto the global matrix. To avoid this phenomenon, a regularization term in the loss can be added as leading to the final loss:

$$\mathcal{L}(W^g, \{W^j\}_{j\in[n_{\texttt{worker}}]}) = \frac{1}{n_{\texttt{task}}} \sum_{i\in[n_{\texttt{task}}]} \sum_{j\in[n_{\texttt{worker}}]} \mathrm{H}\Big(y_i^{(j)}, p_{\text{out}}(x_i, w_j)\Big) - \lambda \sum_{j\in[n_{\texttt{worker}}]} \|W^g - W^j\|_2 \ ,$$

with λ the regularization hyperparameter and H the crossentropy loss. The larger λ, the farther local confusion weights are from the shared confusion.

1.2.5 Evaluation metrics.

Throughout this thesis, we evaluate our results using different metrics depending on the data available.

- If a dataset $\mathcal{D}$ of n tasks is only composed of the answered labels – *i.e.* the images are not available: the crowdsourcing methods consist in estimating the label $y_i^\star$ with an aggregated label $\hat{y}_i$. To evaluate the aggregation method, we use the training recovery accuracy $\text{Acc}_{\text{train}}$ defined as:

$$\text{Acc}_{\text{train}}(\hat{y}, y^\star; \mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(\hat{y}_i = y_i^\star) \ . \tag{1.18}$$

- If the dataset is composed of both the answered labels and the associated images: we can learn the parameters of a classifier and then test the classifier performance on a test set. We thus record the test accuracy and the expected calibration error as in Guo et al. (2017). The test accuracy is defined as:

$$\text{Acc}_{\text{test}}(\hat{y}, y^\star, \mathcal{D}) = \frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}} \mathbb{1}(\hat{y}_i = y_i^\star) \ . \tag{1.19}$$

  The expected calibration error is a metric introduced in Naeini et al. (2015) that addresses the reliability of the predicted probabilities by a model. It is known that many machine learning models (Guo et al., 2017) are not well calibrated. For example in a binary classification problem with smiling and not smiling faces, out of all the tasks smiling that have an estimated probability $\hat{p} \in [0,1]$ to be classified

as smiling, there should be a portion $\hat{p}$ that are smiling. More formally, let us partition the $[0, 1]$ interval into M equally spaced bins $I_1, \ldots, I_M$. Let B_m be the set of tasks x_i with predicted probability falling in bin I_m for $m \in [M]$. Then, in each bin we can look at the accuracy of the model:

$$\text{acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{1}(\hat{y}_i = y_i^\star) \ , \tag{1.20}$$

and at the confidence of the model:

$$\text{conf}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \sigma(\mathcal{C}(x_i))_{[1]} \ . \tag{1.21}$$

The average discrepancy between these two metrics is the expected calibration error (ECE):

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n_{\text{test}}} |\text{acc}(B_m) - \text{conf}(B_m)| \in [0, 1] \ . \tag{1.22}$$

A model perfectly calibrated has an ECE null and the worse calibration happens with an ECE of 1. Note that in practice, we use $M = 15$ bins to partition the interval $[0, 1]$. Other calibration metrics exist (Kumar et al., 2019; Wongon et al., 2020) to alleviate some issues like the metric's bias (Gruber and Buettner, 2022).

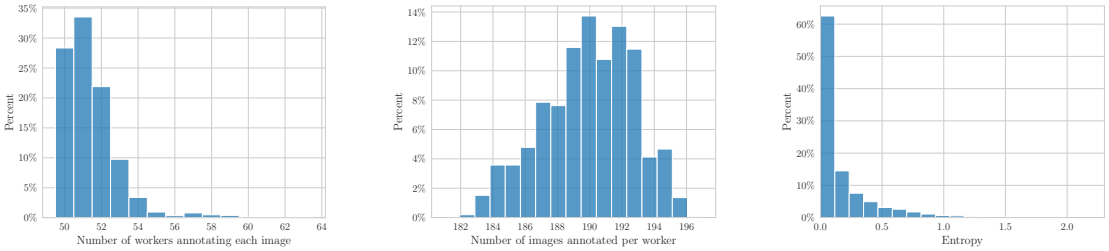
1.2.6 Some classical open access crowdsourcing datasets.

There are few crowdsourced datasets openly available online that released both the votes $\{y_i^{(j)}\}_{i,j}$ with the tasks $(x_i)_i$, and with a (partial) ground truth to evaluate from. We present here three of them that are often used in the literature. The `CIFAR-10H` (Peterson et al., 2019) dataset has been proposed to reflect human perceptual uncertainty in (a subpart of) the classical `CIFAR-10` dataset. Each worker has annotated a large number of (seemingly easy) tasks, thus leading to few disagreements. The `LabelMe` and `Music` datasets (Rodrigues et al., 2014, 2017) have very few votes per task, leading to more ambiguous vote distributions.

The `CIFAR-10H` dataset

Introduced by Peterson et al. (2019), the crowdsourced dataset `CIFAR-10H` attempts to recapture the human labeling noise present when creating the dataset. We have transformed this dataset, mainly by creating a validation set. Hence, the training set for our version of `CIFAR-10H` consists of the first 9500 test images from CIFAR-10, hence $|\mathcal{D}_{\text{train}}| = 9500$. The validation set is then composed of the last 500 images from the training set of `CIFAR-10` meaning $|\mathcal{D}_{\text{test}}| = 500$. The test set consists of the whole training set from CIFAR-10, so $|\mathcal{D}_{\text{test}}| = 50000$. The crowdsourcing experimentation involved $n_{\texttt{worker}} = 2571$ workers on Amazon Mechanical Turk. Workers had to choose

one label for each presented image among the $K = 10$ labels of CIFAR-10: `airplane`, `automobile`, `bird`, `cat`, `deer`, `dog`, `frog`, `horse`, `ship` and `truck`. Each worker labeled 200 tasks (and was paid \$1.50 for that): 20 for each original category. Answering time was also measured for each worker[14]. The `CIFAR-10H` annotating effort is balanced: each task has been labeled by 50 workers on average.



(a) Feedback effort per task distribution

(b) Load per worker distribution

(c) Naive soft labels, entropy distribution

Figure 1.10 `CIFAR-10H`: dataset visualization

From Figure 1.10, we can see that the creation of a validation set leads to some (small) imbalances in the load per worker and feedback effort distribution. The entropy distribution peaks around 0, meaning that the votes are not very ambiguous – most tasks are easy to classify and workers agree.

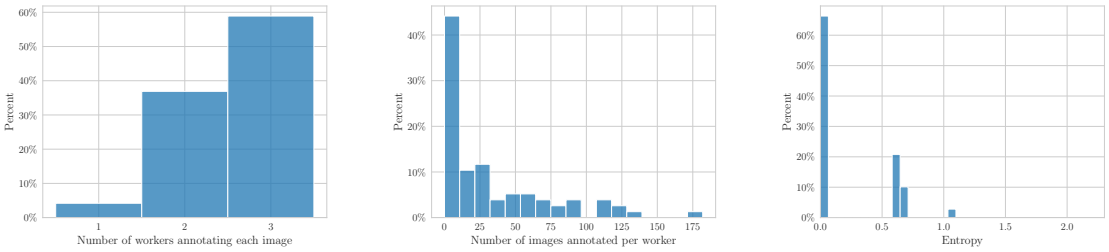
The `LabelMe` dataset

Another real dataset in the crowdsourced image classification field that can be used is the `LabelMe` crowdsourced dataset created by Rodrigues and Pereira (2018). This dataset consists of $n_{\text{task}} = 1000$ training images dispatched among $K = 8$ classes: `highway`, `insidecity`, `tallbuilding`, `street`, `forest`, `coast`, `mountain` or `open country`. The validation set has 500 images and the test set has 1188 images. The whole training tasks have been labeled by $n_{\texttt{worker}} = 59$ workers, each task having between one and three given (crowdsourced) labels. In particular, 42 tasks have been labeled only once, 369 tasks have been labeled twice and 589 received three labels. This is a way sparser labeling setting than the `CIFAR-10H` dataset.

Also, note that the `LabelMe` dataset has classes that overlap and thus lead to intrinsic ambiguities. This is the reason why the CoNAL strategy was introduced by Chu et al. (2021), see details in Section 1.2.4. For example, the classes `highway`, `insidecity`, `street` and `tallbuilding` are overlapping for some tasks: some cities have streets with tall buildings, leading to confusion.

From Figure 1.11, we can see that there are only up to three labels per task, leading to sparser votes per task. Most workers answered few tasks, leading to a peak in the load

[14]Note that attention checks occurred every 20 trial for each worker, for tasks whose labels were known. They have been removed from the dataset since the corresponding images are not available.



(a) Feedback effort per task

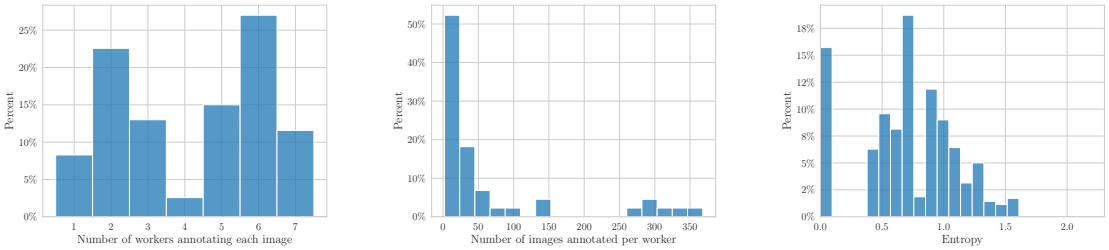
(b) Load per worker distribution

(c) Naive soft labels, entropy distribution

Figure 1.11 `LabelMe`: dataset visualization

per worker distribution. There is also a vast consensus on tasks with a high peak for the entropy distribution near 0. However, in this peak are also counted the tasks with only one label that the entropy can not separate.

The `Music` dataset



(a) Feedback effort per task distribution

(b) Load per worker distribution

(c) Naive soft labels, entropy distribution

Figure 1.12 Music: dataset visualization

Rodrigues et al. (2014) released a crowdsourced dataset of audio files. The goal of this classification task was to decide the genre of 30-second musical excerpts. Number of tasks is $n_{\texttt{task}} = 700$. The $n_{\texttt{worker}} = 44$ workers had $K = 10$ possible labels: `blues`, `classical`, `country`, `disco`, `hiphop`, `jazz`, `metal`, `pop` and `reggae`. Each audio file was labeled by between 1 and 7 workers. To test the results, a dataset of 299 labeled clips is used (originally 300, but one file is known to be corrupted). Instead of working with the original audio files, we have used Mel spectrograms, openly available[15], to rely on standard neural networks architecture for image classification.

From Figure 1.12 we observe that each task received up to 7 labels. Around 7.5% of those tasks were annotated by a single worker. Hence the entropy can not be used

[15] `https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification?datasetId=568973`

to detect the ambiguity for them. Most workers labeled few tasks, and some outliers labeled more than 300 music recordings.

1.2.7 List of publications

This thesis is supported by the following publications:

- Lefort, T., Charlier, B., Joly, A., & Salmon, J. (2022). Crowdsourcing label noise simulation on image classification tasks. 53èmes Journées de Statistique de la SFDS 2022.
- Moreau, T., Massias, M., Gramfort, A., Ablin, P., Bannier, P. A., Charlier, B., ... & Vaiter, S. (2022). Benchopt: Reproducible, efficient and collaborative optimization benchmarks. Advances in Neural Information Processing Systems, 35, 25404-25421.
- Lefort, T., Charlier, B., Joly, A., & Salmon, J. (2023) Weighting areas under the margin in crowdsourced datasets. 54èmes Journées de Statistique de la SFDS 2023.
- Lefort, T., Charlier, B., Joly, A., & Salmon, J. (2024). Peerannot: classification for crowdsourced image datasets with Python. Computo (SFDS)
- Lefort, T., Charlier, B., Joly, A., & Salmon, J. (2024). Improve learning combining crowdsourced labels by weighting Areas Under the Margin. TMLR.
- Lefort, T., Affouard, A., Charlier, B., Salmon, J., Bonnet, P. & Joly, A. (2024). Cooperative learning of Pl@ntNet's Artificial Intelligence algorithm using label aggregation. 55èmes Journées de Statistique de la SFDS 2024
- Dubar, A., Lefort, T., & Salmon, J. (2024). peerannot: A framework for label aggregation in crowdsourced datasets. 55èmes Journées de Statistique de la SFDS 2024
- Lefort, T., Charlier, B., Joly, A., & Salmon, J. (2024). Weighted majority vote using Shapley values in crowdsourcing. CAp 2024-Conférence sur l'Apprentissage Automatique.
- Article MEE

2 Identify ambiguous tasks

Key points – Identify ambiguous tasks

1. We know that datasets with better quality will lead to better models. It is famously a complicated task to explore big datasets and find ambiguous tasks in a classical supervised setting. What can we do for crowdsourced datasets?

2. While entropy or variance-based methods in distribution votes are useful to retrieve tasks that lead to very noisy decisions, they are not fit in settings with few votes per task. And even less fitted to settings where a task can be labeled by a single worker. Can we still exploit these tasks?

Contributions – Weighted Area Under the Margin

3. Following the literature on label noise, we adapt the AUM by Pleiss et al. (2020) into the AUMC and WAUM, two strategies to identify ambiguous tasks in datasets. The first is the baseline and directly falls back to the classical AUM using a majority vote. The WAUM introduces a trust score in the balance not to use poorly performing workers' answers.

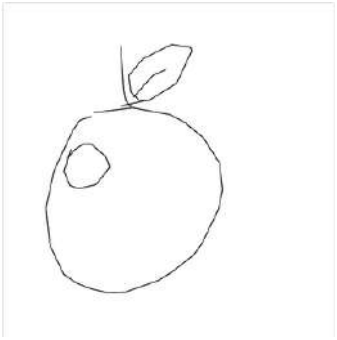
4. We provide a simple guideline: pruning most ambiguous tasks from the dataset, and reporting computer vision classifier performance with and without pruning on simulated and three real datasets: `CIFAR-10H`, `LabelMe` and `Music`.

Chapter 2 – Identify ambiguous tasks:

2.1 Do we know what is in our training sets?

While our datasets are getting larger and larger every year, one question naturally arises: *What is the quality of our training sets?* Indeed, small datasets can easily be looked at, but thousands of images – if not millions – represent herculean human work without assistance. Mistakes happen, and the quality of the data is not always perfect (Figure 2.1).

(a) $y^\star$ = `cat` in CIFAR-10 (Krizhevsky and Hinton, 2009)



(b) $y^\star$ = `tshirt` in Quickdraw

(c) $y^\star = 6$ in MNIST (Deng, 2012)

Figure 2.1 Three examples of labeling mistakes in classical classification datasets. The label can be wrong (CIFAR-10 and Quickdraw) or the task can be too ambiguous to classify (MNIST).

Data quality is linked with model performance (Budach et al., 2022), looking for outliers to prune or weight differently during the learning procedure is not new (Angelova, 2004). In this chapter, we first present the Area Under the Margin (AUM): a statistic from Pleiss et al. (2020) that uses model iterates score prediction to detect unreliable training data points in the classically supervised learning setting. Then, we propose to extend the AUM to the crowdsourcing setting with the Weighted Areas Under the Margin (WAUM). We show that the WAUM allows the identification of ambiguous tasks in crowdsourced datasets, and pruning a portion of those tasks leads to better test performance overall.

2.1.1 Detect labeling errors in classical datasets

In classically supervised learning settings, training sets, tasks are paired up with a single label: $\mathcal{D}_{\text{train}} := \{(x_i, y_i)\}_{i \in [n_{\text{task}}]}$. This label has been assigned either automatically or via human decision. Thus, the task might be mislabeled. We can see a mislabeled task as a task that is difficult to classify for the classifier given this wrong label. The AUM from Pleiss et al. (2020) allows us to identify tasks that are the most difficult to classify from the dataset.

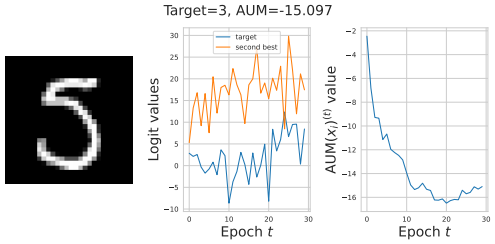
More formally, the AUM of a task (x_i, y_i) from a training set $\mathcal{D}_{\text{train}}$ given a classifier $\mathcal{C}$ and a number of training epochs $T > 0$ is defined as:

$$\text{AUM}\,(x, y; \mathcal{D}_{\texttt{train}}) = \frac{1}{T} \sum_{t=1}^{T} \left[\sigma_y^{(t)}(x) - \sigma_{[2]}^{(t)}(x)\right] \ . \tag{2.1}$$

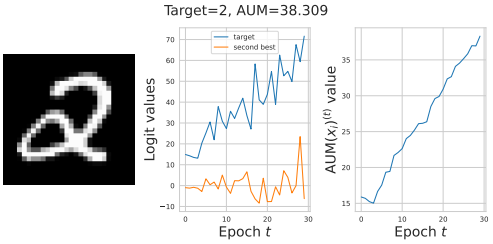
The AUM averages over time how far the score for the given class is from the most predicted other class by the classifier. This is an average over time of the prediction margin. It is classic to look at this margin in the literature to bound test error. For example, Bartlett et al. (1998) showed that test error is dependent on the margin's distribution over the training set, even with zero error reached during training. However, the AUM does not consider the margin given a trained classifier but looks at the early dynamics in the training procedure. Pleiss et al. (2020) use an average of margins over logit scores, while we rather consider the average of margin after a softmax step in Equation (2.1), to temper scaling issues, as advocated by Ju et al. (2018) in ensemble learning. Moreover, we consider the margin introduced by Yang and Koyejo (2020) since the corresponding hinge loss has better theoretical properties than the one used in the original AUM, especially in top-k settings[1] (Lapin et al., 2016; Yang and Koyejo, 2020; Garcin et al., 2022). However, one could easily consider the original margin with few differences in practice for top-1 classification. The algorithm to compute the AUM is detailed in Algorithm 3.

This early dynamic, and by association with the hyperparameter $T > 0$, is necessary as it is well known that modern neural networks classifiers can memorize the data and even classify correctly random tasks because of the hyperparametrization (Maennel et al., 2020). Memorization is a necessary phenomenon for training a neural network: we need the classifier to memorize patterns. But what we wish to consider the least in the AUM is the unintended memorization (Maennel et al., 2020) that can happen early and is difficult to temper – even with strategies like dropout or weight decay. And in Zhang et al. (2021), they show – among other results – that true labels are learned faster than random labels by neural networks classifier. This early training dynamic in the prediction logits can thus be used as a proxy to identify possible mislabeled data.

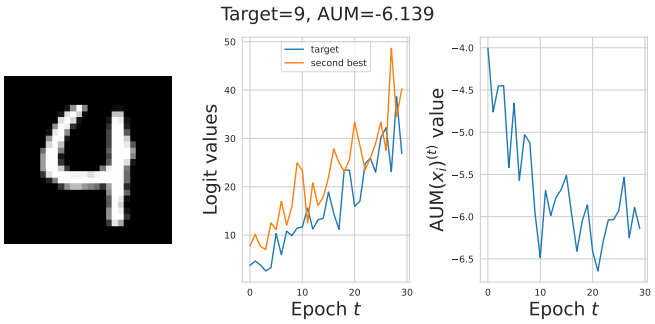
[1] For top-k, consider $\sigma_{[k+1]}^{(t)}(x)$ instead of $\sigma_{[2]}^{(t)}(x)$ in (2.1).



(a) Mislabeled image: AUM $\ll 0$



(b) Correct label: AUM $\gg 0$



(c) Confusion: the image is either a four or a nine cut-out with true label $y_i^\star = 9$, the AUM is closer to zero indicating high ambiguity in learning classes

Figure 2.2 Three examples from the MNIST dataset to illustrate the behavior of the AUM: if the sample is mislabeled the AUM is low as the classifier disagrees with the given label. Note that if $T > 0$ is too high the memorization kicks in and the AUM increases again. When the sample is correctly labeled the AUM increases over training. When the sample is ambiguous, the AUM is closer to 0. Results are from a 2-layer convolution network with max-pooling. Training is done in $T = 30$ epochs using Adam optimizer with a learning rate set to 0.01 and batches of 100 samples.

Note that there exist other algorithms to learn from mislabeled data, such as Confident Learning (Northcutt et al., 2021b) using by `CleanLab`[2]. Confident Learning does not correct the label nor does it re-weight the data. It estimates the joint distribution between the given and unknown latent labels with class-conditional noise and then prunes samples by class with a threshold of the average predicted probability for the samples in the given class. With the AUM, we can also correct the label during training – even though we will not consider this application for this work.

We recall in Algorithm 3 how to compute the AUM in practice for a given training set $\mathcal{D}_{\text{train}}$. This step is used within the WAUM (label aggregation step). Overall, considering a model, computing the AUM requires an additional cost: T training epochs are needed to record the margins' evolution for each task. This usually represents less than twice

[2] `https://cleanlab.ai/`

the original time budget. We recall that $\sigma^{(t)}(x_i)$ is the softmax output of the predicted scores for the task x_i at iteration t.

Algorithm 3 AUM algorithm

Input: $\mathcal{D}_{\text{train}} = (x_i, y_i)_{i \in [n_{\texttt{task}}]}$: training set with $n_{\texttt{task}}$ task/label couples, $T \in \mathbb{N}$: number of epochs
for $t = 1, \ldots, T$ **do**
  Train the neural network for the t^{th} epoch, using $\mathcal{D}_{\text{train}}$
  for $i \in [n_{\texttt{task}}]$ **do**
    Record softmax output $\sigma^{(t)}(x_i) \in \Delta_{K-1}$
    Compute margin $M^{(t)}(x_i, y_i) = \sigma^{(t)}_{y_i}(x_i) - \sigma^{(t)}_{[2]}(x_i)$
$\forall i \in [n_{\texttt{task}}],\ \text{AUM}(x_i, y_i; \mathcal{D}_{\text{train}}) = \frac{1}{T} \sum_{t \in [T]} M^{(t)}(x_i, y_i)$

2.2 The WAUM: extending the AUM to the crowdsourcing setting

In this work, we aim at identifying ambiguous tasks from their associated features, hence discarding hurtful tasks (such as the ones illustrated on Figure 2.4) following the pipeline presented in Figure 2.3. First, we collect the crowdsourced labels, then identify ambiguous tasks and prune them, and finally train a classifier on the pruned dataset.

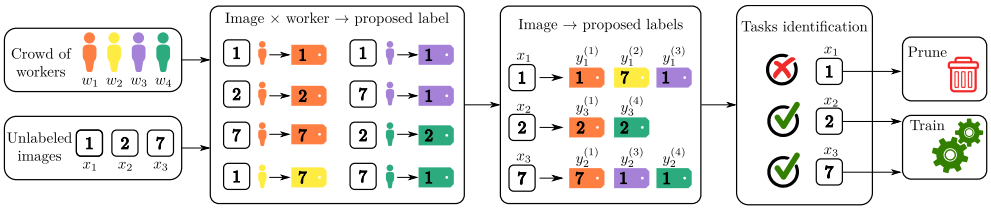
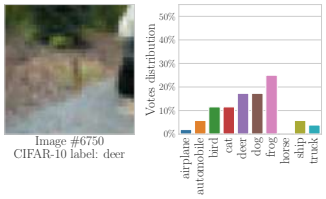


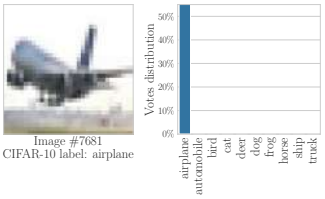
Figure 2.3 Learning with crowdsourcing labels: from label collection with a crowd to training on a pruned dataset. High ambiguity from either crowd workers or tasks intrinsic difficulty can lead to mislabeled data and harm generalization performance. To illustrate our notation, here the set of tasks annotated by worker w_3 is $\mathcal{T}(w_3) = \{1, 3\}$ while the set of workers annotating task x_3 is $\mathcal{A}(x_3) = \{1, 3, 4\}$.

Recent works on data-cleaning in supervised learning (Han et al., 2019; Pleiss et al., 2020; Northcutt et al., 2021b) have shown that some images might be too corrupted or too ambiguous to be labeled by humans. Hence, one should not consider these tasks for label aggregation or learning since they might reduce generalization power; see for instance (Pleiss et al., 2020). Throughout this work, we consider the ambiguity of a task with the informal definition proposed by Angelova (2004) that fit standard learning frameworks: *"Difficult examples are those which obstruct the learning process or mislead the learning algorithm or those which are impossible to reconcile with the rest of the examples"*. This definition links back to how Pleiss et al. (2020) detects corrupted samples using the

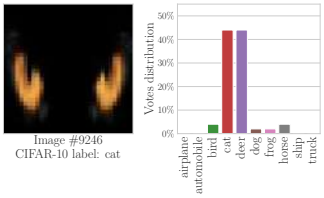
area under the margin (AUM) during the training steps of a machine learning classifier. However, it is important to notice that, in this context, the task ambiguity is inherent to the classifier architecture, and thus might not exactly overlap with human-level difficulty.



(a) Label `deer` is meaningless here, and workers are confused with all other labels.



(b) Label `airplane` is easy to identify (unanimity among workers).



(c) Label `cat` often confused with horns of a wild `deer`

Figure 2.4 Three images from `CIFAR-10H` dataset (Peterson et al., 2019), with the empirical distribution of workers' labels (soft labels): the `airplane` image (a) is easy, while the landscape (b) is ambiguous due to the image's poor quality. The last image (c) looks like a black cat face often perceived as the horns of a `deer`.

While the AUM shows results of identifying different types of sample difficulty by averaging the margin during $T > 0$ training epochs, there is no direct way to apply it to the crowdsourcing setting. Indeed, the AUM needs, by definition, a hard label y_i to consider the assigned-label score $\mathcal{C}(x_i)_{y_i}$ and also to consider the second biggest score for a class different than y_i. In the crowdsourcing setting however, there is no direct hard label $\hat{y}_i$ from the multiple answers $\{y_i^{(j)}\}_{j \in \mathcal{A}(x_i)}$ to a task x_i.

A naive transformation to apply the AUM in a crowdsourcing setting is to consider the majority voting aggregation. Equation (2.1) simply becomes:

$$\text{AUMC}\left(x_i, \left\{y_i^{(j)}\right\}_{j \in \mathcal{A}(x_i)}; \mathcal{D}_{\texttt{train}}\right) = \frac{1}{T}\sum_{t=1}^{T}\left[\sigma^{(t)}_{\hat{y}_i^{\text{MV}}}(x_i) - \sigma^{(t)}_{[2]}(x_i)\right] \quad . \tag{2.2}$$

The AUMC strategy lacks the worker abilities and task difficulty. There is no worker-specific margin, the MV aggregation strategy removed the worker-specific information from the AUM. In the following, we introduce the WAUM: a statistic that generalizes the AUM to the crowdsourcing setting by aggregating worker-specific margins into a weighted average to consider both worker abilities and task difficulty.

Out-of-Model-Scope and task difficulty.

Identifying ambiguous tasks in a training set seems close to the Out-Of-Distribution (OOD) detection problem – when we aim at detecting data that is not from the same distribution as our training set (Schorn and Gauerhof, 2020; Wang et al., 2022). Given a new data point, a monitoring system is built to detect if that point comes from the same distribution as the data used to train a model or whether it deviates significantly from

that distribution. Similarly, recent works argue that the Out-Of-Model-Scope (OMS) detection problem is a more general problem than OOD detection (Granese et al., 2021; Guérin et al., 2023). Given a model, we construct its scope as the set of data where the model is correct. A monitor is built so that, given a new task, we can detect if the model is suited or not to classify it. It is a model-dependent problem, and the scope is defined by the model's architecture and training data, contrary to the OOD problem which is only dependent on the training data.

However, both the OOD and the OMS detection problems are not the same as identifying ambiguous tasks in our setting. Indeed, OOD detection relies on testing whether the task is from the same distribution as the training set without a network. The OMS detection relies on testing whether the task is in the trained model scope at evaluation time. We consider with the AUM, and the WAUM, the task difficulty in the training set, given a new model, how to identify ambiguous tasks before the evaluation step.

2.2.1 Definition and construction of the WAUM

We combine task difficulty scores with worker abilities scores, but we measure the task difficulty by incorporating feature information. We thus introduce the Weighted Area Under the Margin (WAUM), a generalization to the crowdsourcing setting of the Area Under the Margin (AUM) by (Pleiss et al., 2020). The AUM is a confidence indicator in an assigned label defined for each training task. It is computed as an average of margins over scores obtained along the learning steps. The AUM reflects how a learning procedure struggles to classify a task to an assigned label. The AUM is well suited when training a neural network (where the steps are training epochs) or other iterative methods. For instance, it has led to better network calibration (Park and Caragea, 2022) using MixUp strategy (Zhang et al., 2018), *i.e.* mixing tasks identified as simple and difficult by the AUM. The WAUM, our extension of the AUM, aims at identifying ambiguous data points in crowdsourced datasets, so one can prune ambiguous tasks that degrade the generalization. It is a weighted average of workers AUM, where the weights reflect trust scores based on task difficulty and workers' ability.

Given several training epochs $T > 0$ and a classifier $\mathcal{C}$ for a crowdsourced training set $\mathcal{D}_{\text{train}}$, we write the WAUM as a weighted average of worker specific AUM. Let $s^{(j)}(x_i) \in [0, 1]$ be a trust factor in the answer of worker w_j for task x_i. The WAUM is then defined as:

$$\text{WAUM}(x_i) = \frac{\sum_{j \in \mathcal{A}(x_i)} s^{(j)}(x_i) \text{AUM}(x_i, y_i^{(j)})}{\sum_{j' \in \mathcal{A}(x_i)} s^{(j')}(x_i)} \quad . \tag{2.3}$$

It is a weighted average of AUMs over each worker's answer with a per task weighting score $s^{(j)}(x_i)$ based on workers' abilities. This score considers the impact of the AUM for

each answer since it is more informative if the AUM indicates uncertainty for an expert than for a non-expert.

The scores $s^{(j)}$ are obtained *à la* Servajean et al. (2017): each worker has an estimated confusion matrix $\hat{\pi}^{(j)} \in \mathbb{R}^{K\times K}$. Note that the vector $\text{diag}(\hat{\pi}^{(j)}) \in \mathbb{R}^K$ represents the probability for worker w_j to answer correctly to each label. With a neural network classifier, we estimate the probability for the input $x_i \in \mathcal{X}_{\texttt{train}}$ to belong in each category by $\sigma^{(T)}(x_i)$, *i.e.* the probability estimate at the last epoch. As a trust factor, we propose the inner product between the diagonal of the confusion matrix and the softmax vector:

$$s^{(j)}(x_i) = \langle \text{diag}(\hat{\pi}^{(j)}), \sigma^{(T)}(x_i) \rangle \in [0,1] \ . \tag{2.4}$$

The scores control the weight of each worker in Equation (2.3). This choice of weight is inspired by the bilinear scoring system of GLAD (Whitehill et al., 2009), as detailed hereafter. The closer to one, the more we trust the worker for the given task. The score $s^{(j)}(x_i)$ can be seen as a multidimensional version of GLAD's trust score. Indeed, in GLAD, the trust score is modeled as the product $\alpha_j\beta_i$, with $\alpha_j \in \mathbb{R}$ (resp. $\beta_i \in (0,+\infty)$) representing worker ability (resp. task difficulty). In Equation (2.4), the diagonal of the confusion matrix $\hat{\pi}^{(j)}$ represents the worker's ability and the softmax the task difficulty.

Construction of the WAUM

Given a single task x_i and a worker $w_j \in \mathcal{A}(x_i)$, a possible measure of label ambiguity is the AUM:

$$\text{AUM}(x_i, y_i^{(j)}) = \frac{1}{T}\sum_{t=1}^{T} \left[\sigma^{(t)}_{y_i^{(j)}}(x_i) - \sigma^{(t)}_{[2]}(x_i)\right] \ .$$

Extending to multiple workers $(w_j)_{j\in\mathcal{A}(x_i)}$, a naive extension (other than the AUMC) would be to average the AUM over workers:

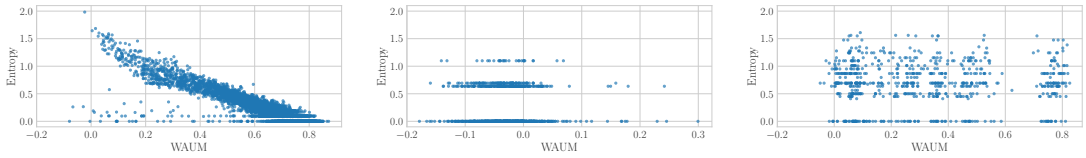
$$\widetilde{\text{WAUM}}(x_i, \{y_i^{(j)}\}_j) = \frac{1}{|\mathcal{A}(x_i)|}\sum_{j\in\mathcal{A}(x_i)} \left[\frac{1}{T}\sum_{t=1}^{T} \left[\sigma^{(t)}_{y_i^{(j)}}(x_i) - \sigma^{(t)}_{[2]}(x_i)\right]\right] \ .$$

However, by doing so the averaged ambiguity does not consider the worker performance. If the worker performs poorly, they should not contribute much to the overall task ambiguity. Hence the proposition is to use a weight $s^{(j)}(x_i)$ that considers both the worker and the task. Instead of the classical average, we thus obtain the weighted average defined in Equation (2.3).

Why variance or entropy-based identifications are not always suited.

The natural choice instead of the WAUM would be to use the entropy of the votes to detect ambiguous tasks. However, the entropy of the votes is not always suited for difficulty identification. Indeed, when large amounts of votes per task are available, the

entropy of the votes and the WAUM coincide well, as in Figure 2.5(a). Yet, when votes are scarce, as in Figure 2.5(b) and (c), entropy becomes irrelevant while our introduced WAUM remains useful.



(a) `CIFAR-10H` dataset. **(b)** `LabelMe` dataset. **(c)** `Music` dataset.

Figure 2.5 Entropy of votes vs. WAUM for `CIFAR-10H`, `LabelMe`, and `Music`, each point representing a task/image. When large amounts of votes per task are available, WAUM and entropy ranking coincide well, as in (a). Yet, when votes are scarce, as in (b) and (c), entropy becomes irrelevant while our introduced WAUM remains useful. Indeed, tasks with few votes can benefit from feedback obtained for a similar one. And for the `LabelMe` dataset in particular, there are only up to three votes available per task, thus only four different values of the entropy possible, making it irrelevant in such cases for modeling task difficulty.

Dataset Pruning and hyperparameter tuning.

Our procedure (Algorithm 4) proceeds as follows. We initialize our method by estimating the confusion matrices for all workers. For each worker w_j, the AUM is computed for its labeled tasks, and so is its worker-dependent trust scores $s^{(j)}(x_i)$ with Equation (2.4) during the training phase of a classifier. The WAUM in Equation (2.3) is then computed for each task. The most ambiguous tasks, the ones whose WAUM are below a threshold, are then discarded, and the associated pruned dataset $\mathcal{D}_{\text{pruned}}$ is output. We consider for the pruning threshold a quantile of order $\alpha \in [0, 1]$ of the WAUM scores. The hyperparameter α (proportion of training data points pruned) can be chosen on a validation set, yet choosing $\alpha \in \{0.1, 0.05, 0.01\}$ has led to satisfactory results in all our experiments. Note that the same pruning procedure can be applied to AUMC for comparison. Both the AUMC and WAUM inherit the hyperparameter $T > 0$ from the original AUM. Following the recommendations from Pleiss et al. (2020), we need T large enough for stability and T not too big to avoid overfitting the data. In practice, a guideline given is to train until the first learning rate scheduler drops to only keep the beginning of the scores trajectories without finetuning. The main assumptions to identify ambiguous tasks is thus not to over-train the neural network in the WAUM (or AUMC) step, and to be able to run a DS-like algorithm to recover the diagonal of the confusion matrix for Equation (2.4).

Algorithm 4 WAUM (Weighted Area Under the Margin).

Input: $\mathcal{D}_{\texttt{train}}$: tasks and crowdsourced labels, $\alpha \in [0,1]$: proportion of training points pruned, $T \in \mathbb{N}$: number of epochs, `Est`: Estimation procedure for the confusion matrices
Output: pruned dataset $\mathcal{D}_{\text{pruned}}$

Get confusion matrix $\{\hat{\pi}^{(j)}\}_{j\in[n_{\texttt{worker}}]}$ from `Est`
Train a classifier for T epochs on $\left(x_i, y_i^{(j)}\right)_{i,j}$
for $j \in [n_{\text{worker}}]$ **do**
    Get $\text{AUM}(x_i, y_i^{(j)}; \mathcal{D}_{\texttt{train}})$ using **??** for $i \in \mathcal{T}(w_j)$
    Get **trust scores** $s^{(j)}(x_i)$ using Equation (2.4) for $i \in \mathcal{T}(w_j)$
for each task $x \in \mathcal{X}_{\texttt{train}}$ **do**
    Compute $\text{WAUM}(x)$ using Equation (2.3)
Get q_α $(\text{WAUM}(x_i))_{i\in[n_{\texttt{task}}]}$, α-**quantile threshold**
$\mathcal{D}_{\text{pruned}} = \left\{ (x_i, (y_i^{(j)})_{j\in\mathcal{A}(x_i)}) : \text{WAUM}(x_i) \geq q_\alpha, x_i \in \mathcal{X}_{\texttt{train}} \right\}$

Refined initialization: estimating confusion matrices.

By default, we rely on the `Est`=DS algorithm to get workers' confusion matrices, but other estimates are possible: DS might suffer from the curse of dimensionality when the number K of classes is large (K^2 coefficients needed per worker).

Training on the pruned dataset.

Once a pruned dataset $\mathcal{D}_{\text{pruned}}$ has been obtained thanks to the WAUM, one can create soft labels through an aggregation step, and use them to train a classifier. Aggregated soft labels contain information regarding human uncertainty, and could often be less noisy than NS labels. They can help improve model calibration (Wen et al., 2021; Zhong et al., 2021), a property useful for interpretation (Jiang et al., 2012; Kumar et al., 2019). Concerning the classifier training, note that it can differ from the one used to compute the WAUM. We train a neural network whose architecture is adapted dataset per dataset and that can differ from the one used in Algorithm 4 (it is the case for instance for the `LabelMe` dataset). For an aggregation technique `agg`, we write the full training method on the pruned dataset created from the WAUM: `agg` + WAUM and instantiate several choices in our experiments. For comparison, we write `agg` + AUMC the training method on the pruned dataset created from the AUMC.

2.2.2 Evaluating the WAUM

Our first experiments focus on multi-class classification datasets with a large number of votes per task. We consider first a simulated dataset to investigate the WAUM and the pruning hyperparameter α. Then, with the real `CIFAR-10H` dataset from Peterson et al. (2019) we compare label aggregation-based procedures with and without pruning using

the AUMC or the WAUM. Finally, we run our experiments on the `LabelMe` dataset from Rodrigues and Pereira (2018) and `Music` dataset from Rodrigues et al. (2014), both real crowdsourced datasets with few labels answered per task. For each aggregation scheme considered, we train a neural network on the soft labels (or hard labels for MV) obtained after the aggregation step. We compare our WAUM scheme with several other strategies like GLAD (feature-blind) or CoNAL (feature-aware) with and without pruning from the AUMC identification step. For CoNAL, two regularization levels are considered: $\lambda = 0$ and $\lambda = 10^{-4}$ (λ controls the distance between the global and the individual confusion matrices). More simulations and an overview of the methods compared are available in Section 2.2.3.

Metrics investigated.

After training, we report two performance metrics on a test set $\mathcal{D}_{\texttt{test}}$: top-1 accuracy and expected calibration error (ECE) (with $M = 15$ bins as in Guo et al. (2017)). The ECE measures the discrepancy between the predicted probabilities and the probabilities of the underlying distribution. For ease of reporting results, we display the score $1 - \text{ECE}$ (hence, the higher the better, and the closer to 1, the better the calibration). Reported errors represent standard deviations over the repeated experiments (10 repetitions on simulated datasets and 5 for real datasets).

Implementation details.

For simulations, the training is performed with a three dense layers artificial neural network (an MLP with three layers) $(30, 20, 20)$ with batch size set to 64. Workers are simulated with `scikit-learn` (Pedregosa et al., 2011) classical classifiers. For `CIFAR-10H` the Resnet-18 (He et al., 2016) architecture is chosen with batch size set to 64. We minimize the cross-entropy loss and use when available a validation step to avoid overfitting. For optimization, we consider an `SGD` solver with 150 training epochs, an initial learning rate of 0.1, decreasing it by a factor 10 at epochs 50 and 100. The WAUM and AUMC are computed with the same parameters for $T = 50$ epochs. Other hyperparameters for `Pytorch`'s (Paszke et al., 2019) `SGD` are `momentum=0.9` and `weight_decay=5e-4`. For the `LabelMe` and `Music` datasets, we use the Adam optimizer with a learning rate set to 0.005 and default hyperparameters. On these two datasets, the WAUM and AUMC are computed using a more classical Resnet-50 for $T = 500$ epochs and the same optimization settings. The architecture used for train and test steps is a pre-trained VGG-16 combined with two dense layers as described in Rodrigues and Pereira (2018) to reproduce original experiments on the `LabelMe` dataset. This architecture differs from the one used to recover the pruned set. Indeed, contrary to the modified VGG-16, the Resnet-50 could be fully pre-trained. The general stability of pre-trained Resnets, thanks to the residuals connections, allows us to compute the WAUM and AUMC with way fewer epochs (each being also with a lower computational cost) compared to VGGs (He et al., 2016). As there

are few tasks, we use data augmentation with random flipping, shearing and dropout (0.5) for 1000 epochs. Experiments were executed with Nvidia RTX 2080 and Quadro T2000 GPUs. Chapter 3 presents more details on the code used with the `peerannot` library. Source codes are available at `https://github.com/peerannot/peerannot`. The WAUM and AUMC sources are available in the `identification` module.

2.2.3 Results on simulated datasets

To explore the behavior of the WAUM, we first two sets of simulated datasets. The first set represents a scenario where there is some ambiguity in the tasks and one worker performs poorly and adds more ambiguity. The WAUM is used to mitigate the impact of the poor worker and remove some tasks too ambiguous. The second set represents a scenario where the task's ambiguity is inherent to the data structure, and pruning should be avoided.

Three circles simulation

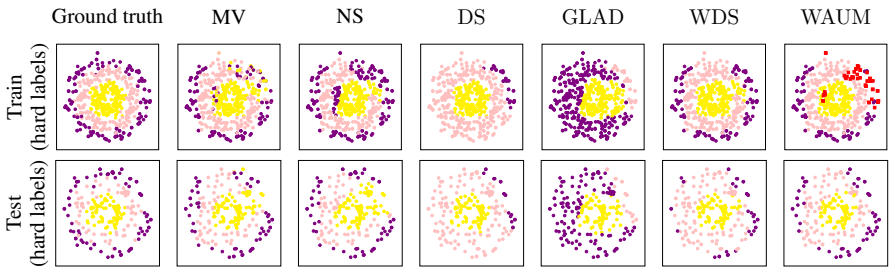


Figure 2.6 `three_circles`: One realization of Table 2.1 varying the aggregation strategy. Training labels are provided from Figure 2.8 and predictions on the test set are from three dense layers' artificial neural network $(30, 20, 20)$ trained on the aggregated soft labels. For ease of visualization, the color displayed for each task represents the most likely class. Red points are pruned from training by WAUM with threshold $\alpha = 0.1$. Here, we have $n_{\texttt{task}} = 525$. WAUM method as in Table 2.1 uses WDS labels.

We simulate three cloud points (to represent $K = 3$ classes) using `scikit-learn`'s function `two_circles`; see Figure 2.8. The $n_{\texttt{worker}} = 3$ workers are standard classifiers: w_1 is a linear Support Vector Machine Classifier (linear SVC), w_2 is an SVM with RBF kernel (SVC), and w_3 is a gradient boosted classifier (GBM). Data is split between train (70%) and test (30%) for a total of 750 points and each simulated worker votes for all tasks, *i.e.* for all $x \in \mathcal{X}_{\texttt{train}}$, $|\mathcal{A}(x)| = n_{\texttt{worker}} = 3$, leading to $n_{\texttt{task}} = 525$ tasks (points). The performance reported in Table 2.1 is averaged over 10 repetitions.

A disagreement area is identified in the northeast area of the dataset (see Figure 2.8). Table 2.1 also shows that pruning too little data (α small) or too much (α large) can mitigate the performance. In Figure 2.7, we show the impact of the pruning

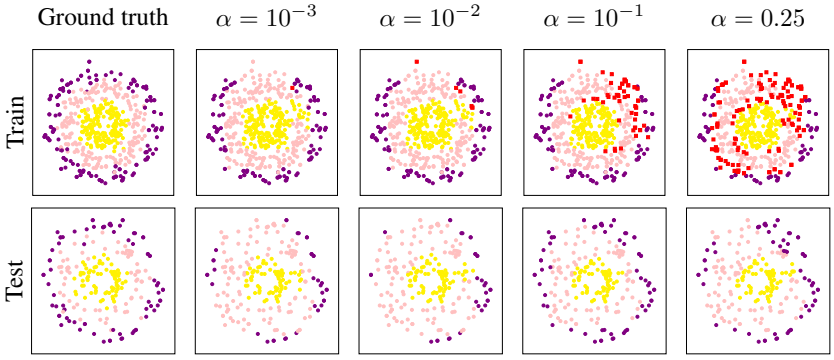


Figure 2.7 Influence of α on the pruning step. Red dots indicate data points pruned from the training set, at level q_α in the WAUM (see line 10 in Algorithm 4). We consider ($\alpha \in \{10^{-3}, 10^{-2}, 10^{-1}, 0.25\}$). The neural network used for predictions is three dense layers $(30, 20, 20)$, as for other simulated experiments. Training labels are from the WDS + WAUM strategy with performance reported in Table 2.1. The more we prune data, the worse the neural network can learn from the training dataset. However, removing the tasks with high disagreement noise helps to generalize.

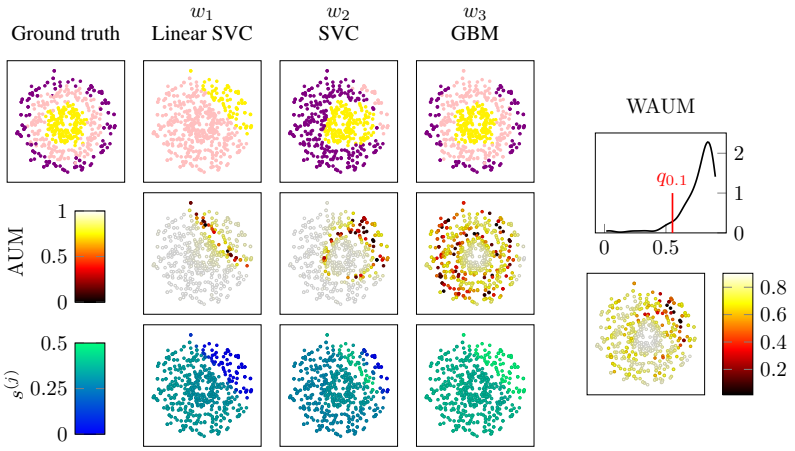


Figure 2.8 `three_circles`: one realization of simulated workers w_1, w_2, w_3, with their AUM, normalized trust scores $s^{(j)}$ (left) and WAUM distributions (right) for $\alpha = 0.1$. Worker w_1 has less impact on the final WAUM in the disagreement area. Note also that for worker w_1 (LinearSVC), the region with low AUM values recovers the usual classifier's margin around the decision boundary.

hyperparameter α. The closer α is to 1, the more training tasks are pruned from the training set (and the worse the performance).

Two moons simulation

This dataset is introduced as a case where pruning is not recommended, to illustrate the limitations of the worker-wise WAUM method. The `two_moons` simulation framework

Strategy	$\text{Acc}_{\texttt{test}}$	ECE
MV	0.73 ± 0.03	$\mathbf{0.13} \pm 0.03$
NS	0.70 ± 0.02	0.18 ± 0.02
DS	0.75 ± 0.07	0.22 ± 0.08
GLAD	0.58 ± 0.02	0.36 ± 0.02
WDS	0.81 ± 0.04	0.17 ± 0.03
WDS + AUMC($\alpha = 10^{-1}$)	0.81 ± 0.02	0.17 ± 0.01
WDS + WAUM($\alpha = 10^{-2}$)	0.80 ± 0.04	0.17 ± 0.01
WDS + WAUM($\alpha = 10^{-1}$)	$\mathbf{0.83} \pm 0.03$	0.19 ± 0.04
WDS + WAUM($\alpha = 0.25$)	0.69 ± 0.02	0.19 ± 0.02

Table 2.1 `three_circles`: Aggregation and learning performance presented in Figure 2.6 ($n_{\texttt{task}} = 525$ tasks, $|\mathcal{A}(x)| = n_{\texttt{worker}} = 3$, 10 repetitions). Errors represented are standard deviations. Note that the best worker, w_3, reaches 0.84 on test accuracy.

showcases the difference between relevant ambiguity in a dataset and an artificial one. This dataset is created using `make_moons` function from `scikit-learn`. We simulate $n_{\texttt{task}} = 500$ points, a noise $\varepsilon = 0.2$ and use a test split of 0.3.

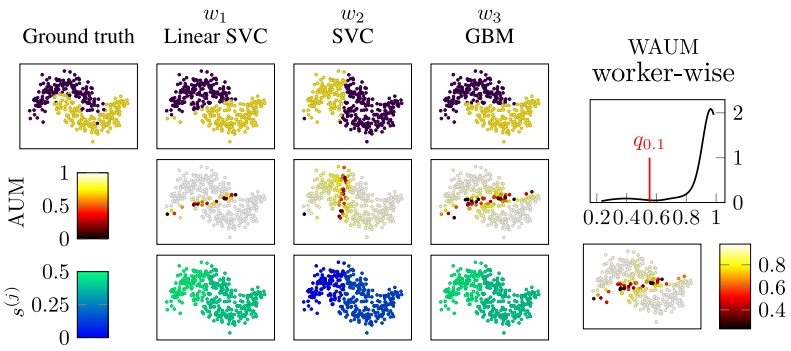


Figure 2.9 `two_moons` dataset: simulated workers with associated AUM and normalized trust scores. The hyperparameter α is set to 0.1 for the worker-wise WAUM. Notice that the SVC classifier is mostly wrong (since we only train for one epoch for this worker), inducing a lower trust score overall.

As can be observed with Figure 2.9 and Figure 2.10, the difficulty of this dataset comes from the two shapes leaning into one another. However, this intrinsic difficulty is not due to noise but is inherent to the data. In this case, removing the hardest tasks means removing points at the edges of the crescents, and those are important in the data's structure. From Table 2.2, we observe that learning on naive soft labeling leads to better performance than other aggregations. But with these workers, no aggregation produced labels capturing the shape of the data.

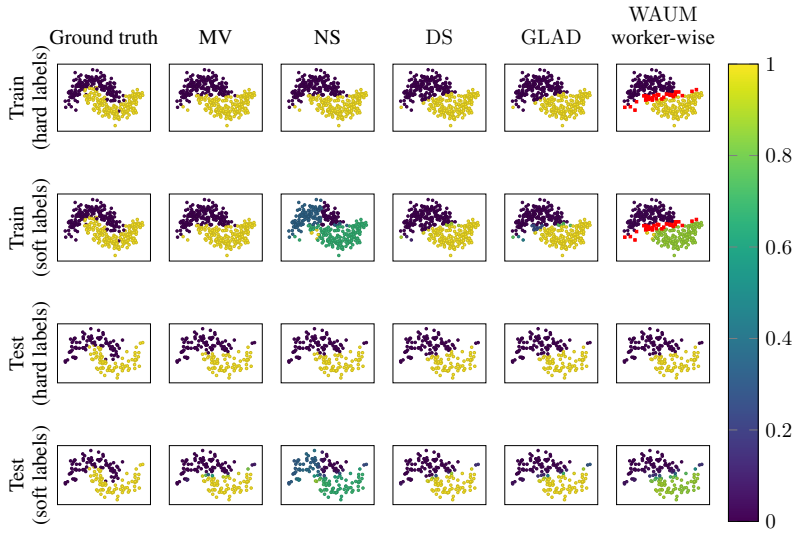


Figure 2.10 `two_moons` dataset: One realization of Table 2.2 varying the aggregation strategy. Label predictions on train/test sets provided by a three dense layers' artificial neural network $(30, 20, 20)$ trained on smooth labeled obtained after aggregating the crowdsourced labels (as in Figure 2.9). Points in red are pruned from the training set in the worker-wise WAUM aggregation. The α hyperparameter is set to 0.1. Each point represents a task x_i, and its color is the probability of belonging in class 1. One can visualize the ambiguity in the soft training aggregated labels, but also in the resulting predictions by the neural network. Errors represented are standard deviations.

2.2.4 Results on real datasets

In this section, we investigate three popular crowdsourced datasets: `CIFAR-10H`, `LabelMe` and `Music`. The first one, `CIFAR-10H` (Peterson et al., 2019), is a curated dataset with many votes per task while `LabelMe` (Rodrigues and Pereira, 2018) and `Music` (Rodrigues et al., 2014) datasets are more challenging, having fewer labels per task. This low number of votes per task, especially for `LabelMe` can lead to erroneous MV labels which then impact the quality of the AUMC. In this context, the label distribution's entropy is also a poor choice to identify hard tasks as can be seen in Figure 2.5. Indeed, with up to three labels, the entropy can only take four different values and thus is no help in ranking the difficulty of 1000 tasks.

To prune only a few tasks, we choose $\alpha = 1\%$ for `CIFAR-10H` and `LabelMe` datasets. For the `Music` dataset, $\alpha = 5\%$ leads to better generalization performance; considering the dataset size and complexity, picking $\alpha = 0.1$ would lead to worse performance. Ablation studies by architecture are performed on `CIFAR-10H` and `LabelMe` datasets in Figure 2.16 to show consistent improvement in performance by using the WAUM to prune ambiguous data.

Table 2.2 Training and test accuracy depending on the aggregation method used for the `two_moons`'s dataset with $n_{\texttt{task}} = 500$ points used for training a three dense layers' artificial neural network $(30, 20, 20)$. For reference, the best worker is w_3 with a training accuracy of 0.923 and a test accuracy of 0.900.

Aggregation	Acc_{test}	ECE
MV	$\mathbf{0.894 \pm 0.002}$	$\mathbf{0.098} \pm 0.004$
NS	0.887 ± 0.002	0.217 ± 0.010
DS	0.867 ± 0.000	0.126 ± 0.001
GLAD	0.872 ± 0.006	0.107 ± 0.004
WDS + WAUM($\alpha = 10^{-3}$)	0.875 ± 0.002	0.088 ± 0.012
WDS + WAUM($\alpha = 10^{-2}$)	0.874 ± 0.002	0.092 ± 0.011
WDS + WAUM($\alpha = 10^{-1}$)	0.870 ± 0.003	0.101 ± 0.020
WDS + WAUM($\alpha = 0.25$)	0.829 ± 0.006	0.135 ± 0.011

`CIFAR-10H` dataset.

The training part of `CIFAR-10H` consists of the 10 000 tasks extracted from the test set of the classical CIFAR-10 dataset (Krizhevsky and Hinton, 2009), and $K = 10$. A total of $n_{\texttt{worker}} = 2571$ workers participated on the Amazon Mechanical Turk platform, each labeling 200 images (20 from each original class), leading to approximately 50 answers per task. We have randomly extracted 500 tasks for a validation set (hence $n_{\texttt{train}} = 9500$). This dataset is notoriously more curated (Aitchison, 2021) than a common dataset in the field: most difficult tasks were identified and removed at the creation of the CIFAR-10 dataset, resulting in few ambiguities. Section 2.2.4 shows that in this simple setting, our data pruning strategy is still relevant, with the choice $\alpha = 0.01$. Images with worst WAUM for each class are presented in Figure 2.11.

Furthermore, the WAUM leads to better generalization performance than the vanilla DS model and the pruning with AUMC. Overall, we show that there is a gain in performance obtained by using a pruning preprocessing step compared to training the classifier on the aggregated labels for the full training set. There is consistently an improvement in using the WAUM pruning – which weights the margins by worker and tasks – over the naive AUMC which does not use reweighing.

`CIFAR-10H` is a relatively well-curated dataset, and we observe in Section 2.2.4 that in this case, simple aggregation methods already perform well, in particular NS. Over the 2571 workers, less than 20 are identified as spammers using Raykar and Yu (2011) but note that most difficult tasks were removed when creating the original CIFAR-10 dataset. We refer to the *"labeler instruction sheet"* of Krizhevsky and Hinton (2009, Appendix C) for more information about the directives given to workers.

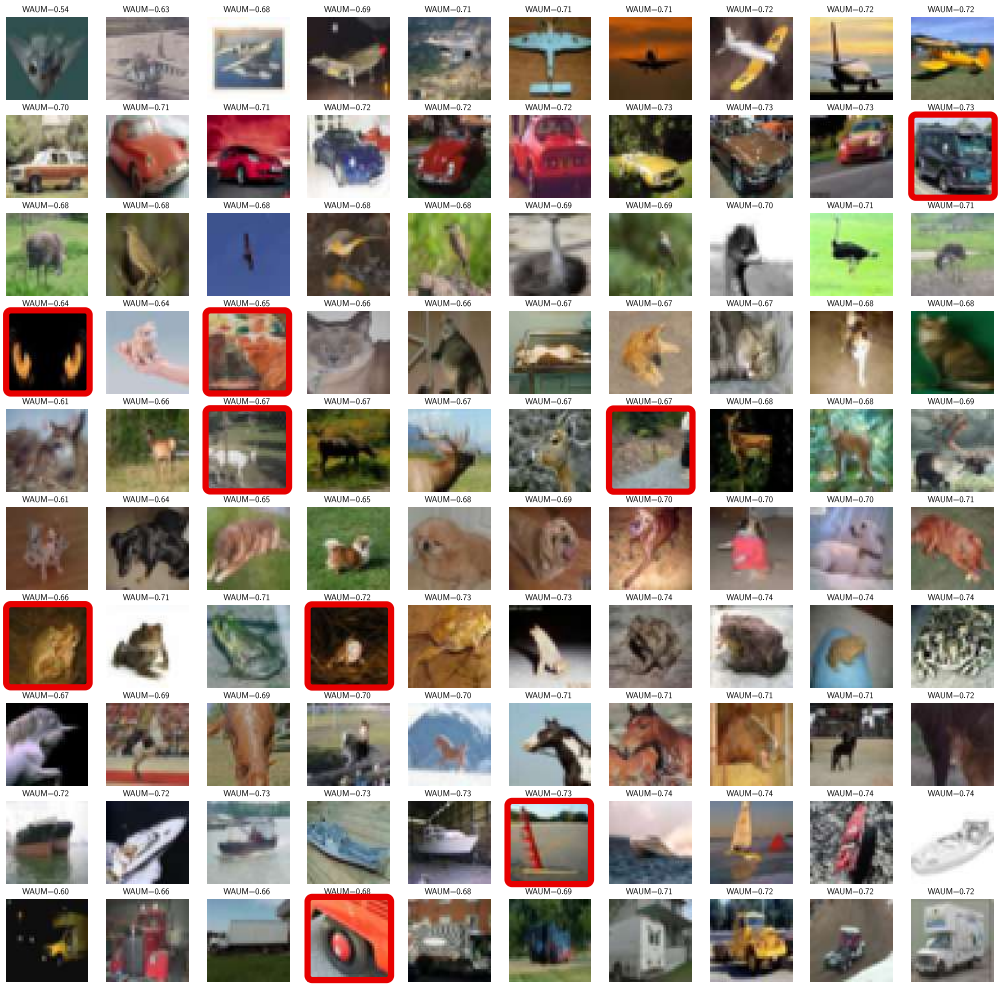


Figure 2.11 `CIFAR-10H`: 10 worst images for WAUM scores, by labels given in CIFAR-10. The rows represent the labels `airplane`, `automobile`, `bird`, `cat`, `deer`, `dog`, `frog`, `horse`, `ship`, and `truck`. Images in red can be particularly hard to classify as they are not typical examples of their label. Comparison with the AUMC and the AUM are available in Figure 2.17 Section 1.2.6.

`LabelMe` dataset.

This dataset consists in classifying 1000 images in $K = 8$ categories. In total 77 workers are reported in the dataset (though only 59 of them answered any task at all). Each task has between 1 and 3 labels. A validation set of 500 images and a test set of 1188 images are available.

We observe in Figure 2.12 that the WAUM improves the final test accuracy when combined with the CoNAL network with regularization. Note that the `LabelMe` dataset has classes that overlap and thus lead to intrinsic ambiguities. This is the reason why the CoNAL strategy was introduced by Chu et al. (2021): modeling common confusions helps the network's decision, so it was expected for the CoNAL to perform well. Combined with

Strategy	$\text{Acc}_{\texttt{test}}(\%)$	$1 - \text{ECE}$
MV	69.53 ± 0.84	0.825 ± 0.00
MV + AUMC	71.12 ± 1.12	$\mathbf{0.836 \pm 0.01}$
MV + WAUM	$\mathbf{72.34 \pm 1.01}$	0.814 ± 0.02
NS	72.14 ± 2.74	$\mathbf{0.868 \pm 0.03}$
NS + AUMC	71.80 ± 2.12	0.838 ± 0.00
NS + WAUM	$\mathbf{72.21 \pm 1.82}$	0.829 ± 0.00
DS	70.26 ± 0.93	0.827 ± 0.00
DS + AUMC	70.43 ± 1.10	$\mathbf{0.833 \pm 0.02}$
DS + WAUM	$\mathbf{72.71 \pm 0.98}$	0.814 ± 0.02
GLAD	70.28 ± 0.88	$\mathbf{0.838 \pm 0.01}$
GLAD + AUMC	70.42 ± 1.23	0.830 ± 0.01
GLAD + WAUM	$\mathbf{71.93 \pm 1.12}$	0.812 ± 0.02
WDS	72.49 ± 0.48	$\mathbf{0.868 \pm 0.00}$
WDS + AUMC	72.47 ± 0.45	0.866 ± 0.00
WDS + WAUM	$\mathbf{72.67 \pm 0.59}$	$\mathbf{0.868 \pm 0.00}$

Table 2.3 `CIFAR-10H`: performance of a `ResNet-18` by label-aggregation crowdsourcing strategy ($\alpha = 0.01$). Errors represented are standard deviations.

our WAUM, additional gains are obtained on both metrics. The vanilla strategy, either for aggregation or learning, can be improved using a pruning preprocessing step. However, between the AUMC and the WAUM, we show a consistent improvement in using the WAUM that considers weights for the workers individually. For example, the classes `highway`, `insidecity`, `street` and `tallbuilding` (in rows) are overlapping for some tasks: cities have streets with tall buildings, leading to confusion shown in Figure 2.14.

`Music` dataset.

This dataset differs from `LabelMe` and `CIFAR-10H` as it consists in classifying 1000 recordings of 30 seconds into $K = 10$ music genres. All the 44 workers involved voted for at least one music, resulting in up to 7 labels per task. Instead of classifying the original audio files, we use the associated Mel spectrograms following the methodology considered by Dong (2018) to retrieve an image classification setting. Though the benefits are not as striking as before on test accuracy, the ECE is slightly improved by combining our WAUM with CoNAL as can be seen in Figure 2.15. Moreover, we show constant improvement of the test generalization performance using the WAUM preprocessing either in accuracy or in calibration.

Among other interesting discoveries, the `WAUM` helped us detect that the music *Zydeco Honky Tonk* by Buckwheat Zydeco was labeled as `classical`, `country` or `pop` by the workers, though it is a `blues` standard. Another example is *Caught in the middle* by Dio classified (with the same number of votes) as `rock`, `jazz`, or `country` though it is a `metal` song. One last example detected: the music *Patches* by Clarence Carter is

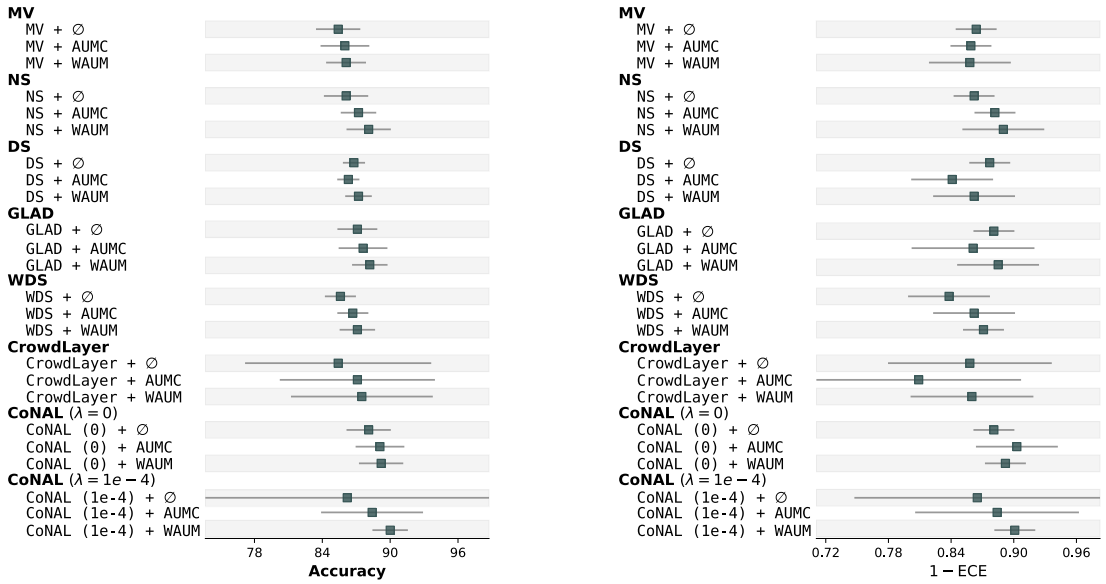
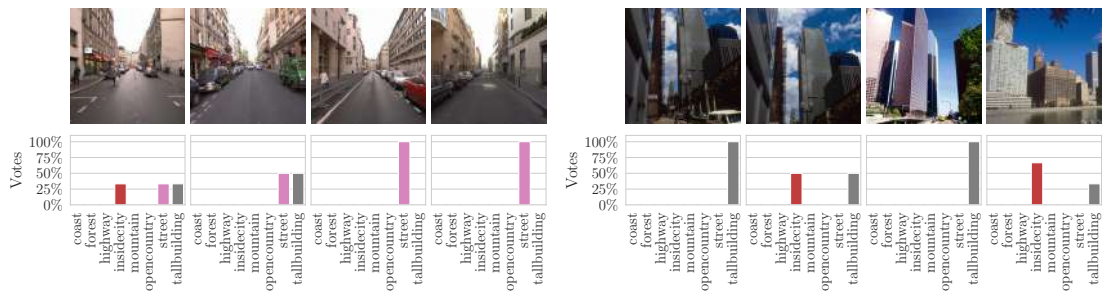


Figure 2.12 Ablation study on `LabelMe` using the VGG backbone: $\alpha = 0.01$. Errors are Gaussian confidence intervals at 95%.



(a) Label `street`. **(b)** Label `tallbuilding`.

Figure 2.13 `LabelMe` dataset: Worst WAUM for classes (top) and the associated voting distribution for each image (bottom). (a) Label `street` (b) Label `tallbuilding`. Even if the two tasks are very similar, because the workers are different the associated proposed labels can differ and add noise during training.

stored in the `disco00020.wav` file. The true label is supposed to be `disco`, while the workers have provided the following labels: two have chosen `rock`, two `blues`, one `pop` and another one proposed `country`. The actual genre of this music is `country`-soul, so both the true label and five out of six workers are incorrect.

Figure 2.14 `LabelMe`: top-10 worst images detected by the WAUM (with labels row-ordered from top to bottom: `highway`, `insidecity`, `street`, `tallbuilding`). Overlapping classes lead to labeling confusion and learning difficulties for both the workers and the neural network.

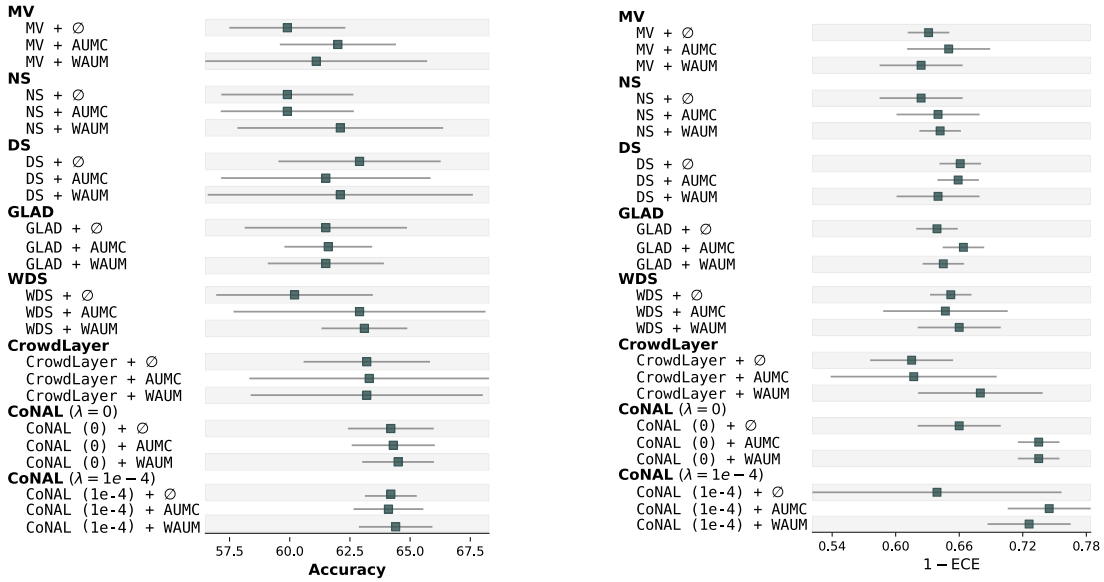


Figure 2.15 Ablation study on `Music` using the VGG backbone: $\alpha = 0.05$. Errors are Gaussian confidence intervals at 95%.

WAUM sensitivity to the neural network architecture.

In the following, we explore the architecture's impact on the generalization performance using the WAUM preprocessing. We compare three architectures, a VGG-16 with two dense layers added from Rodrigues and Pereira (2018), a Resnet-18 and a Resnet-34. We show in Figure 2.16 that depending on the network used, performance varies, but the WAUM step improves generalization performance in most cases (and does not worsen it).

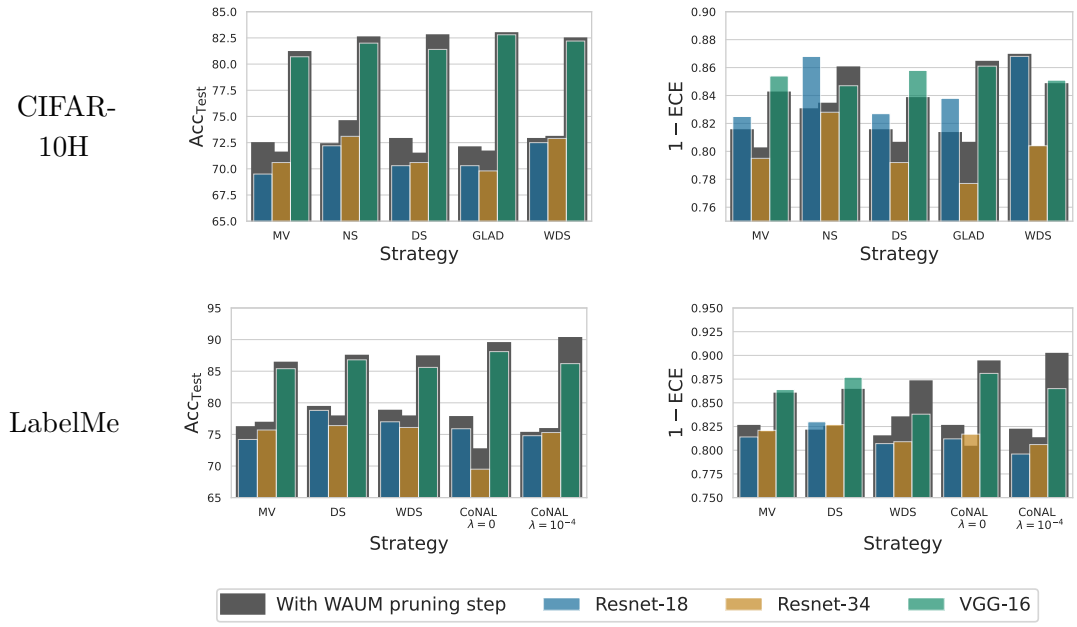


Figure 2.16 Performance obtained by training on the pruned dataset from the WAUM preprocessing step on `CIFAR-10H` and `LabelMe`. We consider multiple neural network architectures – ResNet-18, ResNet-34 or VGG-16 with batch normalization and two supplementary dense layers. We show that performance in accuracy is improved in most cases. Calibration performance in terms of ECE fluctuates depending on the architecture considered, especially for the `CIFAR-10H` dataset. Using the WAUM with CoNAL on the `LabelMe` dataset, we obtain the best performance both in accuracy and calibration.

Qualitative comparison between AUM, AUMC and WAUM

In Figure 2.17, we provide a qualitative view of the most ambiguous tasks detected using the classical AUM, the AUMC Equation (2.2) and the introduced WAUM Equation (2.3) on the CIFAR-10H dataset.

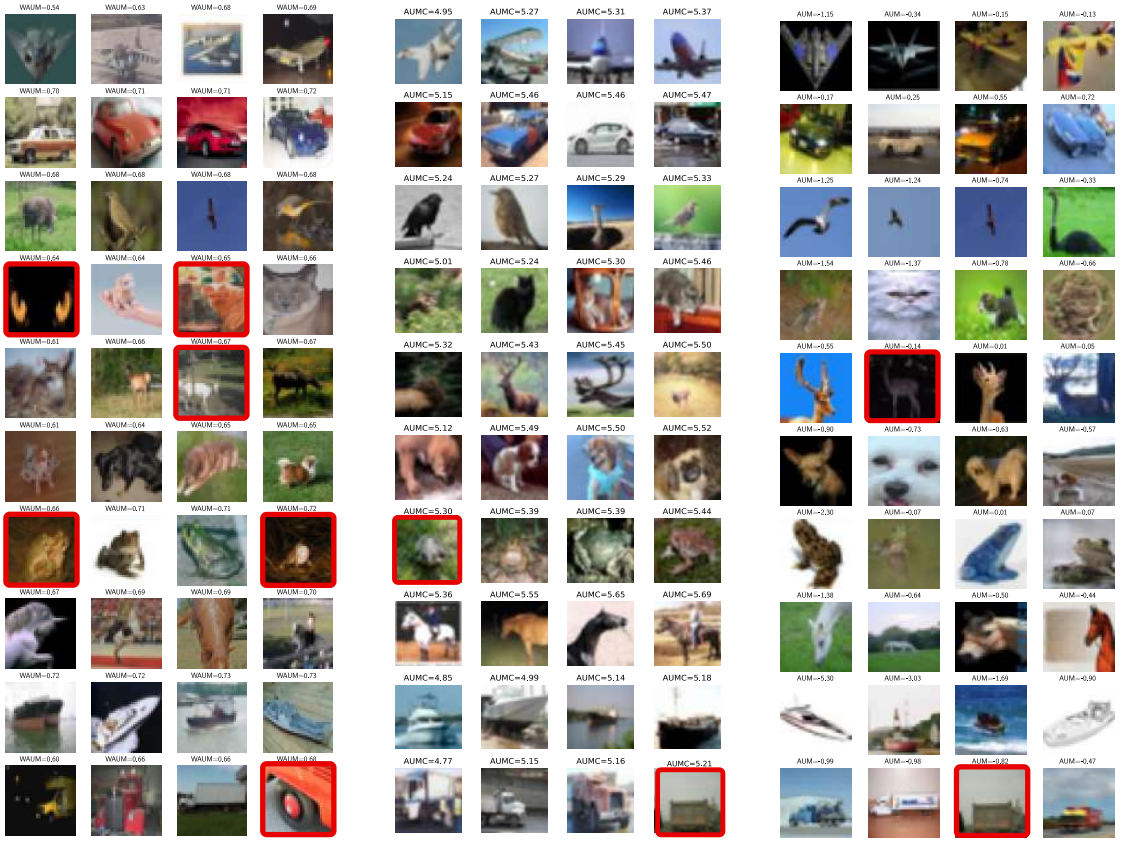
Margin comparison

In the AUM, AUMC and WAUM formulae, we consider a margin from Yang and Koyejo (2020) (denoted ψ_5 in the original article) that has better theoretical properties for top-k classification but that is not the margin proposed in Pleiss et al. (2020) (ψ_1). Indeed, our margin in the AUM is written as:

$$\sigma_{y_i}^{(t)}(x_i) - \sigma_{[2]}^{(t)}(x_i) \ ,$$

instead of

$$\sigma_{y_i}^{(t)}(x_i) - \max_{k \neq y_i} \sigma_k^{(t)}(x_i) \ .$$



(a) WAUM crowdsourced identification

(b) AUMC crowdsourced identification

(c) AUM ground truth identification

Figure 2.17 Comparison of the worse images detected by the WAUM, AUMC and classical AUM preprocessing step. Identification was computed with a ResNet-18 for 50 epochs using the parameters described in Section 1.2.5. Each row represents the class given by the unobserved ground truth label from the `CIFAR-10` dataset. Only the AUM uses the ground truth label, other methods are based on the crowdsourced labels only. Images framed in red can be hard to classify.

Using the `CIFAR-10H` dataset, we can compare the identified tasks using each margin. Note that in the library used (and described in Chapter 3) switching from the original margin to the top-k based margin is executed with the argument `use_pleiss=True` or `use_pleiss=False` with the WAUM, AUM and AUMC. A comparison of the images with the lowest AUM is provided in Figure 2.20. A similar visual comparison on the `CIFAR-10H` dataset is provided in Figure 2.23.

Furthermore, we provide an ablation study on top-2 accuracy scores using the WAUM with ψ_1 or ψ_5 margin on the `LabelMe` dataset in Table 2.4. We use the cross entropy loss during the training phase. With ψ_5, the top-2 margin writes $\sigma_{y_i}^{(t)}(x_i) - \sigma_{[3]}^{(t)}(x_i)$ as indicated in Section 2.2.1. We compute the top-2 accuracy *i.e.* the accuracy in recovering the true label as the first or second predicted label by the classifier. However, note

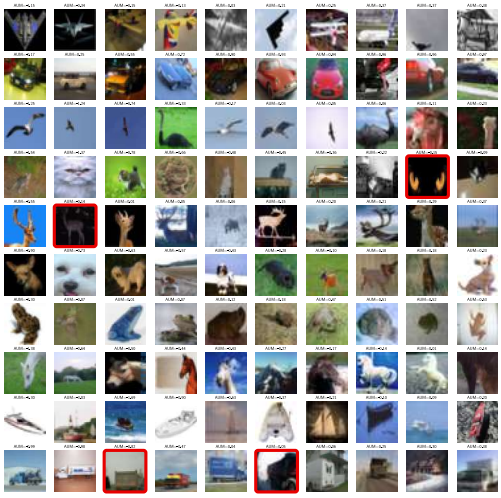


Figure 2.18 Lowest AUM with ψ_1 margin

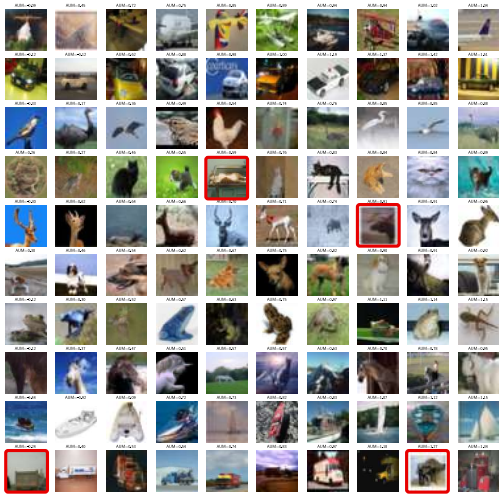


Figure 2.19 Lowest AUM with ψ_5 margin

Figure 2.20 Comparison of the images with lowest AUM in `CIFAR-10H` dataset using the margin from Pleiss et al. (2020) (ψ_1) or the margin for top-1 classification from Yang and Koyejo (2020) (ψ_5). Both margins yield similar results.

that this dataset has $K = 8$ classes, hence we do not report top-5 accuracy as all strategies perform similarly. We notice that performance on this dataset is similar for most strategies between the two margins used for pruning.

Table 2.4 Top-2 Accuracy comparison on the `LabelMe` dataset using the modified VGG-16 backbone and the same hyperparameters as in Section 2.2.2. Results are averaged over 5 repetitions, and errors are standard deviations.

Strategy	**Top-2 no pruning**	**Top-2 WAUM(ψ_1)**	**Top-2 WAUM(ψ_5)**
MV	91.25 ± 2.01	91.17 ± 2.12	92.02 ± 2.08
NS	90.92 ± 1.53	90.41 ± 2.77	89.91 ± 1.08
DS	89.98 ± 1.12	90.24 ± 0.92	91.41 ± 0.99
GLAD	90.78 ± 0.98	91.34 ± 1.59	90.49 ± 0.38
WDS	89.56 ± 1.76	90.82 ± 1.81	91.16 ± 2.78
CrowdLayer	87.45 ± 2.03	88.33 ± 1.49	88.57 ± 2.51
CoNAL($\lambda = 0$)	92.34 ± 0.74	89.49 ± 0.53	94.30 ± 1.32
CoNAL($\lambda = 10^{-4}$)	91.68 ± 1.01	94.10 ± 0.9	94.93 ± 0.76

Limitations and assumptions

First, concerning the weights $s_i^{(j)}$ (reflecting the trust in the image/worker interaction), we rely on confusion matrices $\{\hat{\pi}^{(j)}\}_{j \in [n_{\texttt{worker}}]}$. The DS model (Dawid and Skene, 1979) can be naturally used to estimate such matrices $\pi^{(j)} \in \mathbb{R}^{K \times K}$ for each worker w_j. Yet,

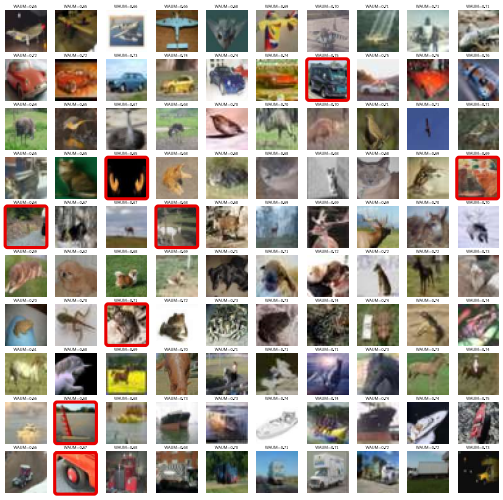


Figure 2.21 Lowest WAUM with ψ_1

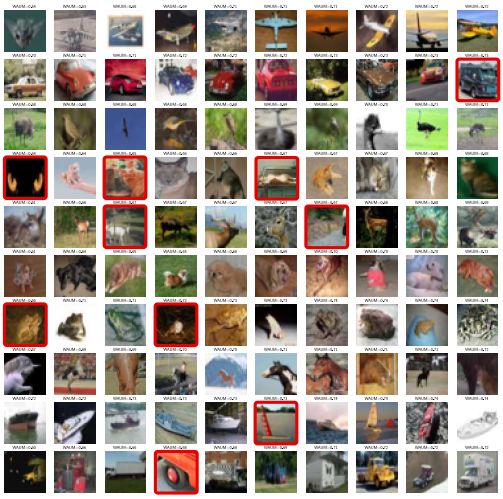


Figure 2.22 Lowest WAUM with ψ_5

Figure 2.23 Comparison of the images with lowest WAUM in `CIFAR-10H` dataset using the margin from Pleiss et al. (2020) (ψ_1) or the margin for top-1 classification from Yang and Koyejo (2020) (ψ_5). Both margins also lead to similar results.

the quadratic number of parameters (w.r.t. K) to be estimated for each worker can create convergence issues for the vanilla DS model when K is large. But as stated in Section 2.2.1, any model that can estimate confusion matrices can be considered for the WAUM's computation. We detail below some possible variants, that could help compute the confusion matrices used in the WAUM for the trust score computation.

- Sinha et al. (2018) accelerated the vanilla DS by constraining the estimated labels' distribution to be a Dirac mass. Hence, predicted labels are hard labels. This leads to worse calibration errors than vanilla DS but preserves the same accuracy.
- Passonneau and Carpenter (2014) introduced Dirichlet priors on the confusion matrices' rows and the prevalence ρ to incorporate previously known information on the workers in the model (from other experiments).
- Servajean et al. (2017) exploited the sparsity of the confusion matrices to cope with a large K.
- Imamura et al. (2018) estimated with variational inference $L \ll n_{\texttt{worker}}$ clusters of workers, constraining at most L different confusion matrices. This reduces the number of parameters required from $K^2 \times n_{\texttt{worker}}$ to $K^2 \times L$.

Pruning and *i.i.d* assumption. The pruning at preprocessing can induce a distortion in the training data distribution. A usual assumption made on learning problems is that the task/label pairs are *i.i.d.* However, by removing some of the hardest tasks, the new dataset $\mathcal{D}_{\text{pruned}}$ contains tasks that are not independent anymore. We should also keep

in mind that Ilyas et al. (2022) have shown that in the standard datasets, the data is not *i.i.d* to begin with.

2.3 Conclusion

In this chapter, we investigate crowdsourcing aggregation models and how judging systems may impact generalization performance. Most models consider the ambiguity from the workers' perspective (very few consider the difficulty of the task itself) and evaluate workers on hard tasks that might be too ambiguous to be relevant, leading to a performance drop. Using a popular model (DS), we develop the WAUM, a flexible feature-aware metric that can identify hard tasks and improves generalization performance over vanilla strategies and naive pruning AUMC that both extend the existing AUM to the crowdsourcing setting. It also yields a fair evaluation of workers' abilities and supports recent research on data pruning in supervised datasets. Independently of pruning, the WAUM allows identifying early the images that need extra labeling efforts or that are impossible to correctly label.

Extension of the WAUM to more general learning tasks (*e.g* top-k classification, Section 2.2.4) would be natural, including sequential label. Indeed, the WAUM could help to identify tasks requiring additional expertise and guide how to allocate more experts/workers for such identified tasks. However, there is a community need for openly available datasets with both tasks and votes for such evaluation.

Broader Impact Statement. As this work proposes a method to prune tasks from training datasets based on human-derived data, we remind that pruning based on learning difficulty can induce a learning bias for the model. To mitigate this, only pruning a small portion of the dataset can help avoid any class with a small number of representatives to be removed of the dataset. Also, in this paper, we only remove tasks that are difficult to classify, we do not remove workers from the dataset. In particular, there is no repercussion on their pay, and by only evaluating them on tasks that are not detected as ambiguous, we evaluate their abilities on fairer tasks.

3 Reproducible and open library

Key points – Community based data but what about codes...

1. Reproducibility has been a demand from the scientific community. With crowdsourcing, the coupling of the label-gathering step and the aggregation is key to creating a classical supervised learning dataset. Different label aggregation strategies can lead to widely different results. Releasing publicly available datasets' original version with collected labels would lead to better data quality.

2. More than the data itself, as the crowdsourcing community is made of researchers with very diverse backgrounds, new models arise quickly. In multiple coding languages (if any) and with personalized data formats. We need formatting propositions that can handle large datasets, are easily accessible and understandable.

3. Aggregation strategies are often EM-based with a two-step procedure repeated. While performance is an important decision factor in using one strategy over another, how much time it takes to run is essential. Especially with large datasets, we find memory scaling issues or a time complexity that forbids usage in applications.

Contributions – peerannot and Benchopt

4. We propose a new Python library `peerannot` fully documented. An `identify` module lets users identify ambiguous tasks from datasets using a wide range of strategies. The `aggregate` module performs label aggregation strategies. The `aggregate-deep` module uses learning strategies that are deep-learning based and have inserted the aggregation step inside the network's architecture. The `train` module allows to train classifiers from aggregated labels. Our library comes with data templates and examples available at `http://peerannot.github.io`

5. We created a crowdsourcing benchmark in the `BenchOpt` library to easily compare time performance on label aggregation strategies across libraries, on publicly available datasets: `https://github.com/benchopt/benchmark_crowdsourcing`.

Chapter 3 – Reproducible and open library:

3.1 `peerannot`: Open access for crowdsourcing strategies in python

The experiments ran in Chapter 2 made us realize key points in the field of crowdsourcing. The first one is that the data is often not released in a format that is easily usable – when released. The second is that most of the time, the code is not released – or partially released without functions and easy access to run new experiments, or also scattered with each their different programming language (python, R, stan, java,...). The third is that existing libraries to handle crowdsourcing data lack implemented strategies to identify poorly performing workers and/or ambiguous tasks. Thus, we created the `peerannot` library.

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

1. **How to aggregate multiple labels into a single label from crowdsourced tasks?** This occurs, for example, when dealing with a single dataset that has been labeled by multiple workers with disagreements. This is also encountered with other scoring issues such as polls, reviews, peer-grading, *etc.* In our framework, this is treated with the `aggregate` command, which given multiple labels, infers a label. From aggregated labels, a classifier can then be trained using the `train` command.

2. **How to learn a classifier from crowdsourced datasets?** Where the first question is bound by aggregating multiple labels into a single one, this considers the case where we do not need a single label to train on, but instead train a classifier on the crowdsourced data, with the motivation to perform well on a testing set. This end-to-end vision is common in machine learning; however, it requires the actual tasks (the images, texts, videos, *etc.*) to train on – and in crowdsourced

datasets publicly available, they are not always available. This is treated with the `aggregate-deep` command that runs strategies where the aggregation has been transformed into a deep learning optimization problem.

3. **How to identify good workers in the crowd and difficult tasks?** When multiple answers are given to a single task, looking for who to trust for which type of task becomes necessary to estimate the labels or later train a model with as few noise sources as possible. The module `identify` uses different scoring metrics to create a worker and/or task evaluation.

The library `peerannot` addresses these practical questions within a reproducible setting and an easy-to-follow pipeline presented in Figure 3.1. 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 network 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.

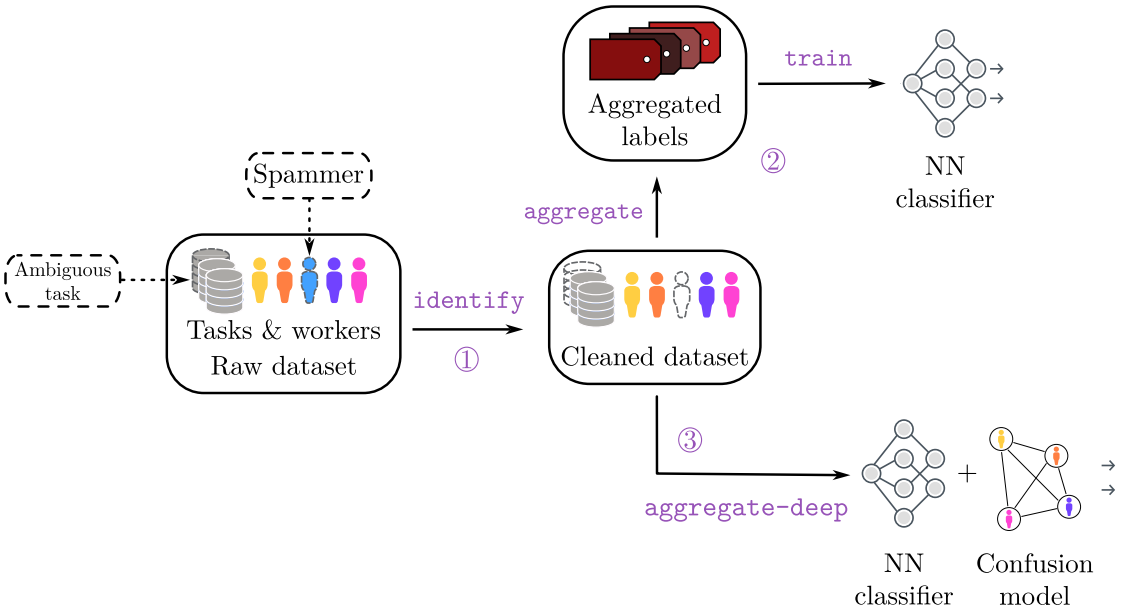


Figure 3.1 Pipeline on how to handle crowdsourced datasets with `peerannot`. After collecting the data, the `identify` module helps find poorly performing workers and/or ambiguous tasks. Those can be pruned to recover a *cleaned* set. Then, the `aggregate` module can be used to infer a label from multiple labels. The `aggregate-deep` module can be used to train a classifier from the crowdsourced labels without aggregation. Finally, the `train` module can be used to train a classifier from aggregated labels.

3.1.1 Presenting the peerannot library usage

The `peerannot` library is available on `https://peerannot.github.io/` and can be installed using `pip`:

```
$ pip install peerannot
```

When installed, it comes with both a `python` Application Programming Interface (API) and a Command Line Interface (CLI). Note that the `python` API is the main interface to use the library, and the CLI is a wrapper around the `python` API to make it easier to use for non-programmers. Moreover, the CLI can be used in a `python` program in interactive cells using the `!` character to run the shell commands indicated by the dollar sign `$`.

Dataset standardization

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.

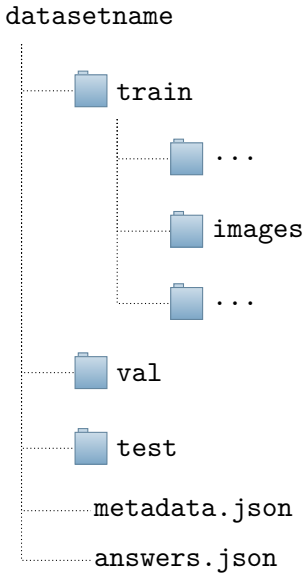


Figure 3.2 Template of a dataset folder in `peerannot`. Collected votes are in `answers.json`, all necessary information on the dataset are in `metadata.json`. Tasks are either in the `train`, `val` or `test` folders. `test` tasks are assumed to have an associated ground truth label.

The `answers.json` file stores the different votes for each task as described in Figure 3.3. This `.json` is the rosetta stone between the task IDs and the images. It contains the tasks' id, the workers's id and the proposed label for each given vote. 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.

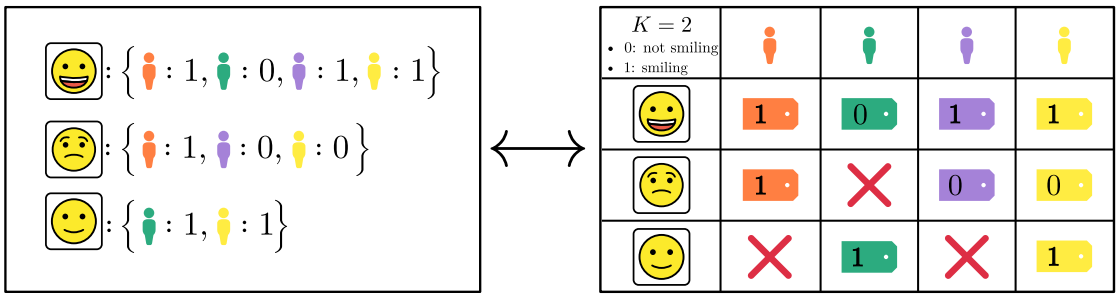


Figure 3.3 Data storage for the `toy-data` crowdsourced dataset, a binary classification problem ($K = 2$, smiling/not smiling) on recognizing smiling emoticons. On the left how `peerannot` stores the data, and on the right the raw data.

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 3.3 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 et al., 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 for standard supervised learning (classification). Both datasets have been reannotated by a crowd of workers.

Other popular formats

Other popular storage formats currently exist. For example, the `crowd-kit` library[1] uses a dataframe where each row specifies the 3-uplet (task, worker, label). This format is

[1] `https://github.com/Toloka/crowd-kit`

close to the `json` one, easily switchable between the two. However, it suffers from the redundancy of the task ID. More discussion on the `crowd-kit` library is available in Section 3.2.

The `LabelMe` dataset labels are stored in a dense matrix of size $(n_{\text{task}}, n_{\text{worker}})$, where each entry is the label given by the worker for the task. This format is not memory efficient for a large number of tasks or workers. Especially if workers do not get to label all tasks.

On a more practical note, the `json` format has the advantage of being easily readable and writable by humans and is also easily convertible to a data frame. It is also easy to use for `python`, `SQL` and `JavaScript`. As large crowdsourcing web platforms use requests in `JavaScript` to send and receive data, the `json` format is a motivating choice for applications.

3.1.2 Label agggregation with `peerannot`

In addition to the classical MV, NS, DS, GLAD aggregation strategies presented in Section 1.2.3, `peerannot` proposes a growing number of aggregation strategies to fit different needs. The full list is available by running the command:

```
$ peerannot agginfo
```

For example, the Worker Clustered DS model (DSWC) by Imamura et al. (2018) is based on the DS model. Each worker belongs to one of the $L \leq n_{\text{worker}}$ clusters. This strategy highly reduces the number of parameters. In the original DS strategy, there are $K^2 \times n_{\text{worker}}$ parameters to estimate for the confusion matrices. The DSWC strategy has $K^2 \times L + L$ parameters to estimate. Indeed, there are L confusion matrices $\Lambda = \{\Lambda_1, \dots, \Lambda_L\}$ of size $K \times K$ and the confusion matrix of a cluster is assumed drawn from a multinomial distribution with weights $(\tau_1, \dots, \tau_L) \in \Delta_L$ over Λ such that $\mathbb{P}(\pi^{(j)} = \Lambda_\ell) = \tau_\ell$ for $\ell \in [L]$.

Structure of a label aggregation strategy. All of the label aggregation strategies are stored in the `peerannot.models` module. Each strategy is a class object in its own `python` file. It inherits from the `CrowdModel` class template and is defined with at least three methods:

- `run`(): includes the optimization procedure to obtain needed weights (*e.g.* the EM algorithm for DS). It is only needed for optimization-based strategies.
- `get_probas`(): returns the soft labels output for each task after running the `run` method,
- `get_answers`(): returns the hard labels output for each task after running the `run` method.

Example of a label aggregation strategy. For example, let us consider minimal working examples (MWE) for the `NS` and the `DS` strategies. The first in Listing 1 is a non-parametric strategy without any optimization algorithm, and the second in Listing 11 in Appendix B is an EM-based parametric strategy.

```python
from ..template import CrowdModel
import numpy as np


class NaiveSoft(CrowdModel):
    def __init__(self, answers, n_classes=2, **kwargs):
        super().__init__(answers)
        self.n_classes = n_classes

    def get_probas(self):
        baseline = np.zeros((len(self.answers), self.n_classes))
        for task_id in list(self.answers.keys()):
            task = self.answers[task_id]
            for vote in list(task.values()):
                baseline[task_id, vote] += 1
        self.baseline = baseline
        return baseline / baseline.sum(axis=1).reshape(-1, 1)

    def get_answers(self):
        return np.vectorize(self.converter.inv_labels.get)(
            np.argmax(self.get_probas(), axis=1)
        )
```

Listing 1 MWE for the NS label aggregation in `peerannot`.

If a new user wants to add their strategy, they can follow the same structure and add it to the `peerannot` library. The strategy will then be available for all users to use through a pull request. Then, the `Benchopt` library can access it to provide comparisons with other strategies easily shared (see Section 3.2).

3.1.3 Compare label aggregation strategies with simulated datasets

Using the `peerannot` library, we can easily simulate crowdsourced answers for classification settings. Hereafter, we present two settings: one where workers answer independently, and another where mistakes are correlated. Another setting where the mistakes are dependent on the task's difficulty level is available in Appendix B.

Simulated independent mistakes

The independent mistakes setting considers 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 better with enough data as we are simulating data from its assumed noise model.

We simulate in Listing 2 $n_{\text{task}} = 200$ tasks and $n_{\text{worker}} = 30$ workers. The number of classes is $K = 5$. Each task x_i receives $|\mathcal{A}(x_i)| = 10$ labels. With 200 tasks and 30 workers, asking for 10 labels leads to around $\frac{200 \times 10}{30} \simeq 67$ tasks per worker (with variations due to randomness in the assignations as seen in Figure 3.4). Note that in practice achieving this result is not straightforward as workers can not label 67 images without specific motivation (games, money rewards, *etc.*).

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

Listing 2 Simulation of independent mistakes in `peerannot`.

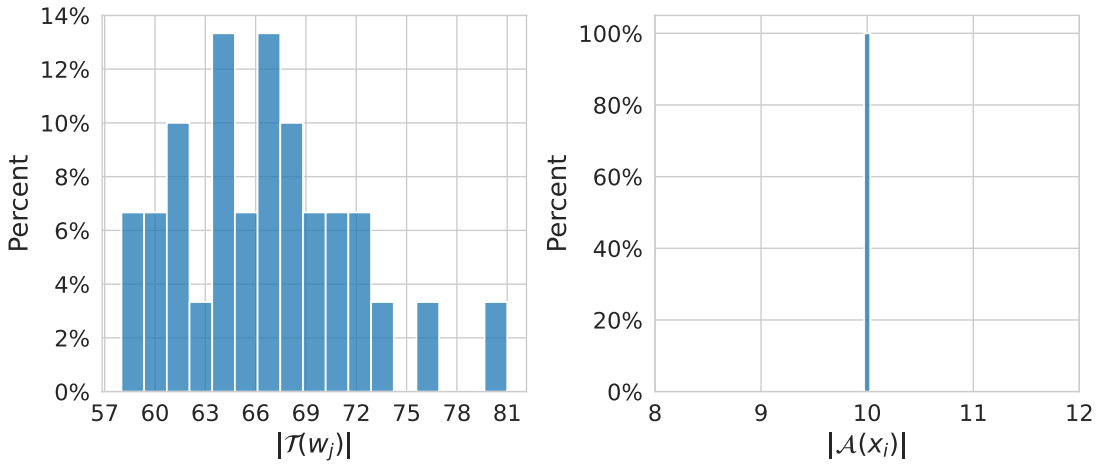


Figure 3.4 Distribution of the number of tasks given per worker (left) and number of labels per task (right) in the independent mistakes setting.

With the obtained answers, we can look at the aforementioned aggregation strategies' performance. The `peerannot aggregate` command takes as input the path to the data

folder and the aggregation strategy `--strategy/-s`. Other arguments are available and described in the `--help` description.

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

Listing 3 Running aggregation on the independent mistakes generated dataset.

Table 3.1 AccTrain metric on simulated independent mistakes considering classical feature-blind label aggregation strategies.

Method	MV	GLAD	DS	DSWC[L=5]	DSWC[L=10]	NS
AccTrain	0.765	0.775	0.890	0.775	0.770	0.760

As expected by the simulation framework, Table 3.2 fits the DS model, thus leading to better accuracy in retrieving the simulated labels for the DS strategy. The MV and NS aggregations

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/
```

Listing 4 Simulation of independent mistakes in `peerannot` with an imbalance in the number of votes per task.

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}
```

Listing 5 Running aggregation on the independent mistakes generated dataset.

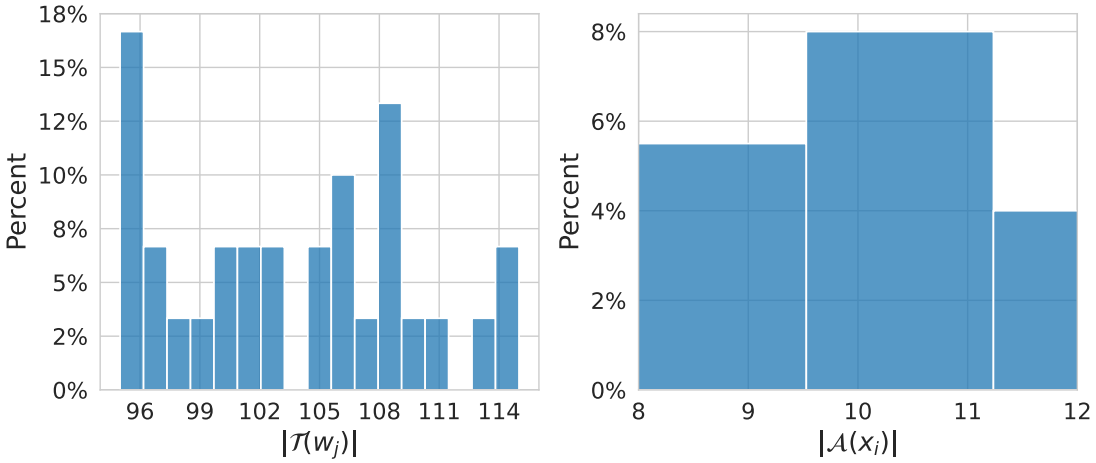


Figure 3.5 Distribution of the number of tasks given per worker (left) and the number of labels per task (right) in the independent mistakes setting with voting imbalance enabled.

Table 3.2 AccTrain metric on simulated independent mistakes, with votes imbalance, considering classical feature-blind label aggregation strategies.

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

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 outperformed by other strategies).

Simulated correlated mistakes

The correlated mistakes are also known as the student-teacher or junior-expert setting (Cao et al., 2019). 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 @sec-simu-independent) confusion matrices of each teacher and the students 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 $|\mathcal{A}(x_i)| = 10$ labels. And, with the obtained answers, we can look at the aforementioned aggregation strategies' performance:

```
$ 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
```

Listing 6 Simulation of independent mistakes in `peerannot` with an imbalance in the number of votes per task.

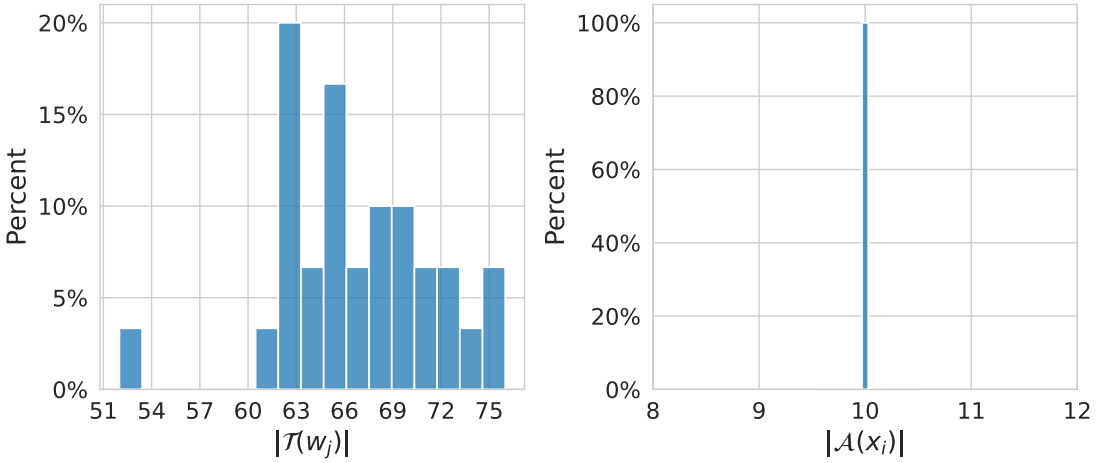


Figure 3.6 Distribution of the number of tasks given per worker (left) and the number of labels per task (right) in the correlated mistakes setting.

Table 3.3 AccTrain metric on simulated correlated mistakes considering classical feature-blind label aggregation strategies.

Method	MV	GLAD	DS	DSWC[L=5]	DSWC[L=10]	NS
AccTrain	0.705	0.645	0.755	0.795	0.815	0.690

With Table 3.3, we see that with correlated data (24 students and 6 teachers), using 5 confusion matrices with DSWC[L=5] outperforms the vanilla DS strategy that does not consider the correlations. The best-performing method here estimates only 10 confusion matrices (instead of 30 for the vanilla DS model).

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

More on confusion matrices in simulation settings

Moreover, the concept of confusion matrices has been commonly used to represent worker abilities. Let us remind that a confusion matrix $\pi^{(j)} \in \mathbb{R}^{K \times K}$ of a worker w_j is defined such that $\pi^{(j)}_{k,\ell} = \mathbb{P}(y_i^{(j)} = \ell | y_i^\star = k)$. These quantities need to be estimated since no true label is available in a crowd-sourced scenario. In practice, the confusion matrix of each worker is estimated via an aggregation strategy like Dawid and Skene's (Dawid and Skene, 1979) presented in Section 1.2.3.

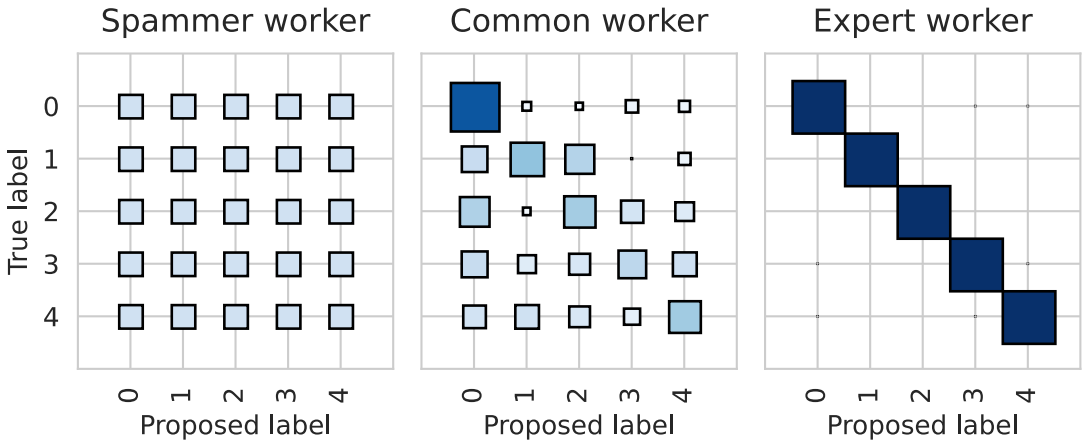


Figure 3.7 Three types of profiles of worker confusion matrices simulated with `peerannot`. The spammer answers independently of the true label. Expert workers identify classes without mistakes. In practice common workers are good for some classes but might confuse two (or more) labels. All workers are simulated using the `peerannot simulate` command.

In Figure 3.7, we illustrate multiple workers' profiles (as reflected by their confusion matrix) on a simulated scenario where the ground truth is available. For that, we generate toy datasets with the `simulate` command from `peerannot`. In particular, we display a type of worker that can hurt data quality: the spammer. Raykar and Yu (2011) defined a spammer as a worker that answers independently of the true label:

$$\forall k \in [K],\ \mathbb{P}(y_i^{(j)} = k | y_i^\star) = \mathbb{P}(y_i^{(j)} = k)\ . \tag{3.1}$$

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

3.1.4 Learning from crowdsourced tasks with `peerannot`

The `peerannot` library has also integrated end-to-end learning strategies in the `aggregate-deep` module. Such strategies include CrowdLayer and CoNAL presented in Section 1.2.4 and Section 1.2.4.

Let us use peerannot to train a VGG-16 with two dense layers on the LabelMe dataset. This model is called modellabelme in the peerannot library as this modification was introduced to reach state-of-the-art performance in Chu et al. (2021). Other models from the torchvision library can be used, such as Resnets, Alexnet *etc.* The aggregate-deep command takes as input the path to the data folder, --output-name/-o is the name for the output file, --n-classes/-K the number of classes, --strategy/-s the learning strategy to perform (*e.g.*, CrowdLayer or CoNAL), the backbone classifier in --model and then optimization hyperparameters for pytorch described with more details using the peerannot aggregate-deep --help command as shown in Listing 7.

```
for strat in ["MV", "NaiveSoft", "DS", "GLAD"]:
    !peerannot aggregate ./labelme/ -s {strat}
    !peerannot train ./labelme -o labelme_${strat} \
        -K 8 \
        --labels=./labelme/labels/labels_labelme_${strat}.npy \
        --model modellabelme \
        --n-epochs 500 \
        -m 50 -m 150 -m 250 --scheduler=multistep \
        --lr=0.01 --num-workers=8 \
        --pretrained \
        --data-augmentation \
        --optimizer=adam \
        --batch-size=32 --img-size=224 \
        --seed=1

for strat in ["CrowdLayer", "CoNAL[scale=0]", "CoNAL[scale=1e-4]"]:
    !peerannot aggregate-deep ./labelme \
        -o labelme_${strat} \
        --answers ./labelme/answers.json \
        -s ${strat} \
        --model modellabelme \
        --pretrained \
        --n-classes=8 \
        --n-epochs=500 \
        --lr=0.001 -m 300 -m 400 --scheduler=multistep \
        --batch-size=228 --img-size=224 \
        --optimizer=adam \
        --num-workers=8 \
        --data-augmentation \
        --seed=1
```

Listing 7 Command to learn from image classification tasks with crowdsourced labels using peerannot. Learning from tasks can be achieved by first aggregating labels, then, training a model. Or with end-to-end strategies calling the aggregate-deep command.

Method	AccTest	ECE
DS	81.061	0.189
MV	85.606	0.143
NS	86.448	0.136
CrowdLayer	87.205	0.117
GLAD	87.542	0.124
CoNAL[scale=0]	88.468	0.115
CoNAL[scale=1e-4]	88.889	0.112

Table 3.4 Generalization performance on LabelMe dataset depending on the learning strategy from the crowdsourced labels. The network used is a VGG-16 with two dense layers for all methods.

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 in Section 3.1.5.

3.1.5 Identifying tasks difficulty and worker abilities

If a dataset requires crowdsourcing 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[2]), citizen science is popular as it is an easy way to produce many labels.

However, mistakes and confusion 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 voluntarily (*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.

[2] `https://www.mturk.com/`

Exploring tasks' difficulty

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

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

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

- GLAD's scoring: by construction, Whitehill et al. (2009) introduced a scalar coefficient to score the difficulty of a task.

- the Weighted Area Under the Margins (WAUM): introduced by Lefort et al. (2022) and presented in Chapter 2, this weighted area under the margins indicates how difficult it is for a classifier $\mathcal{C}$ to learn a task's label. This procedure is done with a budget of $T > 0$ epochs. Given the crowdsourced labels and the trust we have in each worker denoted $s^{(j)}(x_i) > 0$, the WAUM of a given task $x_i \in \mathcal{X}$ and a set of crowdsourced labels $\{y_i^{(j)}\}_j \in [K]^{|\mathcal{A}(x_i)|}$ is defined as:

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

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

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

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

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

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

Note that each of these statistics could prove useful in different contexts. The entropy is irrelevant in settings with few labels per task (small $|\mathcal{A}(x_i)|$). For instance, it is uninformative for `LabelMe` dataset. The WAUM can handle 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.

Results on the `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).

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

```

Listing 8 Command to identify ambiguous tasks on the `CIFAR-10H` dataset using `peerannot`.

Figure 3.8 Most difficult tasks sorted by class from MV aggregation identified depending on the strategy used (entropy, GLAD or WAUM) using a Resnet34. We only display the `truck` class. All class results are available interactively in the main paper at `https://tanglef.github.io/computo_2023`.

The entropy, GLAD's difficulty, and WAUM's difficulty each show different images as exhibited in the interactive Figure. While the entropy and GLAD output similar tasks, in this case, the WAUM often differs. We can also observe an ambiguity induced by the labels in the `truck` category in Figure 3.8, with the presence of a trailer that is technically a mixup between a `car` and a `truck`.

Figure 3.9 Most difficult tasks sorted by class from MV aggregation identified depending on the strategy used (entropy, GLAD or WAUM) using a VGG-16 model with two dense layers. We only display the `opencountry` class. All class results are available interactively in the main paper at `https://tanglef.github.io/computo_2023`.

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

Now, let us compare the tasks' difficulty distribution depending on the strategy considered using `peerannot`. Note that in this experiment, because the number of labels given per task is in $\{1, 2, 3\}$, the entropy only takes four values. In particular, tasks with only one label all have a null entropy, so not just consensual tasks. The MV is also not suited in this case because of the low number of votes per task.

The underlying difficulty of these tasks mainly comes from the overlap in possible

labels. For example, `tallbuildings` are most often found `insidecities`, and so are `streets`. In the `opencountry` we find `forests`, river-`coasts` and `mountains`.

Identification of worker 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 closeness to spammer metric (Raykar and Yu, 2011) (also called spammer score) 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. On the contrary, the closer to zero, the more likely it is for the worker to be a spammer. This score separates spammers from common workers and experts (with profiles as presented in Figure 3.7).

When the tasks are available, confusion-matrix-based deep learning models can also be used. We thus add to the comparison the trace of the confusion matrices with CrowdLayer and CoNAL on the `LabelMe` datasets. For CoNAL, we only consider the trace of the confusion matrix $\pi^{(j)}$ in the pairwise comparison. Moreover, for CrowdLayer and CoNAL we show in Figure 3.11 the weights learned without the softmax operation by row to keep the comparison as simple as possible with the actual outputs of the model.

Comparisons in Figure 3.10 and Figure 3.11 are plotted pairwise between the evaluated metrics. Each point represents a worker. Each off-diagonal plot shows the joint distribution between the scores of the y-axis row and the x-axis column. They allow us to visualize the relationship between these two variables. The main diagonal represents the (smoothed) marginal distribution of the score of the considered column.

Results on CIFAR-10H workers. The `CIFAR-10H` dataset has few disagreements among workers. However, these strategies disagree on the ranking of good against best workers as they do not measure the same properties. We can use `peerannot` as shown in Listing 9 to identify worker reliability on the `CIFAR-10H` dataset with different strategies.

```
! 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
```

Listing 9 Command to identify worker reliability on the `CIFAR-10H` dataset using `peerannot`.

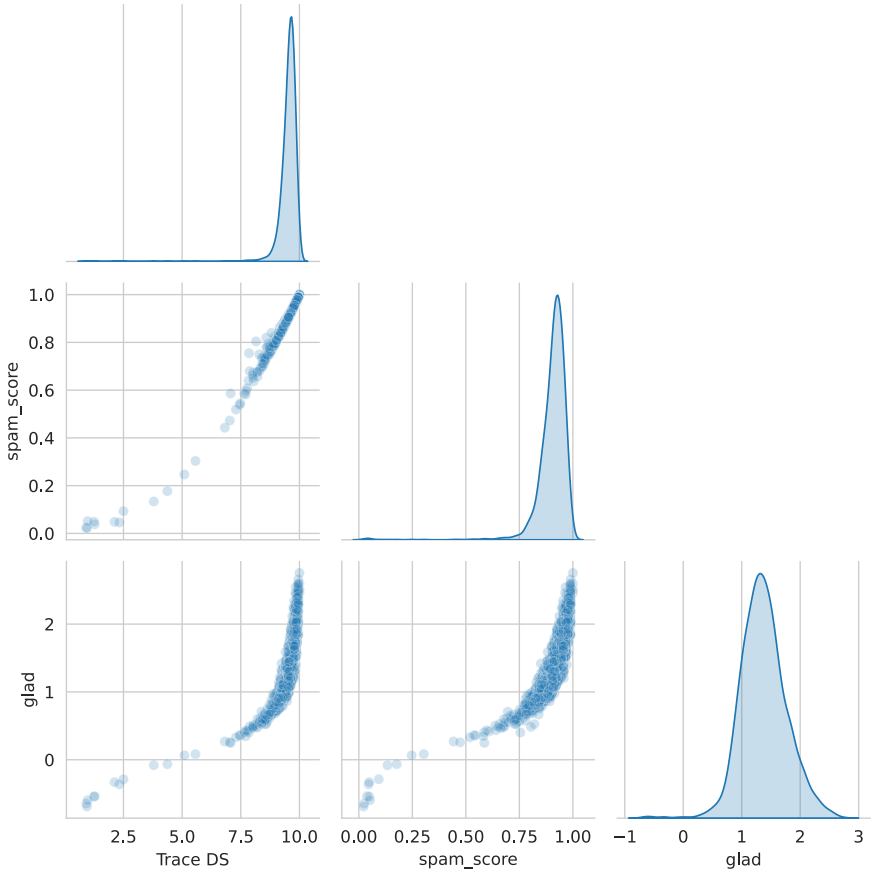


Figure 3.10 Comparison of ability scores by workers for the CIFAR-10H dataset. All metrics computed identify the same poorly performing workers. A mass of good and expert workers can be seen as the dataset presents few disagreements, thus few data to discriminate expert workers from the others.

From Figure 3.10, we can see that in this dataset, different methods easily separate the worst workers from the rest of the crowd (workers in the left tail of the distribution). Note that as different metrics investigate different properties, the best workers are not the same depending on the method used. However, overall all strategies agree on the worst workers in this case.

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

We can see in Figure 3.11 that the number of labels available by task highly impacts the worker evaluation scores. The spam score, DS model and CoNAL all show similar results in the distribution shape (bimodal distribution) whereas GLAD and CrowdLayer are more concentrated. However, this does not account for the ranking of a given worker by the methods considered. The exploration of the dataset lets us look at different scores,

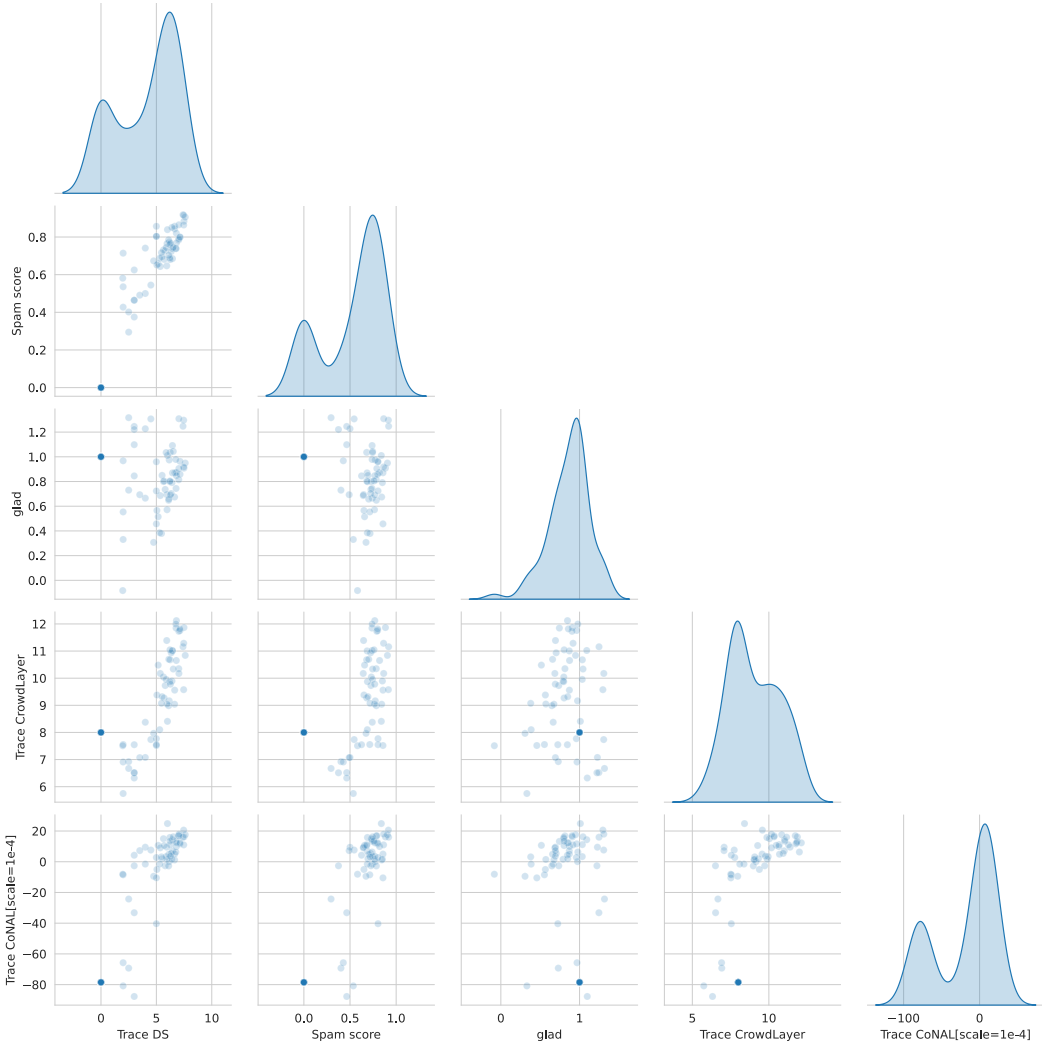


Figure 3.11 With few labels per task, workers are more difficult to rank. It is more difficult to separate workers with their abilities in this crowd. Hence the importance of investigating the generalization performance of the methods presented in the previous section.

but generalization performance presented in Section 3.1.4 should also be considered in crowdsourcing. This difference in worker evaluation scores indeed further highlights the importance of using multiple test metrics to compare the model's prediction performance in crowdsourcing. Poorly performing workers could be removed from the dataset with naive strategies like MV or NS. However, some label aggregation strategies like DS or GLAD can sometimes use adversarial votes as information – for example in binary classification, with a worker answering always the opposite label the confusion matrix retrieves the true label. We have seen that the library `peerannot` allows users to explore the datasets, both in terms of tasks and of workers, and easily compare predictive performance in this setting.

In practice, the data exploration step can be used to detect possible ambiguities in the dataset's tasks, but also remove answers from spammers to improve the data quality as shown in Figure 1.6. The easy access to the different strategies allows the user to decide if, for their collected dataset, there is a need for more recent deep-learning-based strategies to improve the results. This is the case for the `LabelMe` dataset. Otherwise, the user can decide that standard aggregation-based crowdsourcing strategies are sufficient and for example, if there are plenty of votes per task like in `CIFAR-10H`, that the entropy of the vote distribution is a criterion that identified enough ambiguous tasks for their case. As often, not a single strategy works best for all datasets, hence the need to perform easy comparisons with `peerannot`.

3.1.6 Case study with bird sound classification

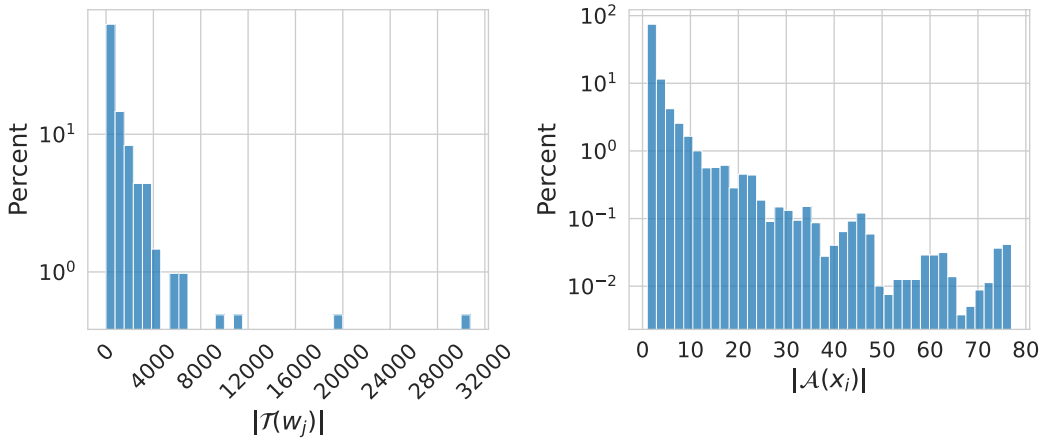


Figure 3.12 Distribution of the number of tasks given per worker (left) and of the number of labels per task (right) in the Audio Birds letters dataset.

We shared our results on the classical CIFAR-10H and LabelMe datasets. More recently, Lehikoinen et al. (2023) developed a platform for bird sound classification. They released the data for the following crowdsourcing experiment. Given the sample of the audio of a species (denoted as a letter on their web portal), users were presented with a new audio sample (the candidate). The question is as follows: *"Is the species vocalizing in the candidate the same as the species in the letter?"* The answer is a binary yes or no ($K = 2$). In total, $n_{\text{worker}} = 205$ workers labeled $n_{\text{task}} = 79\,592$ candidates. Each task received between 1 and 77 annotations. Workers answered between 1 and 30 759 tasks (only one worker achieved that record, and 23% of the workers answered 100 tasks). There is no test set available as is in the original dataset. However, to have an idea of the level of performance of the label aggregation strategies, we use the fact that workers reported their level of expertise between 1 and 4. The latter corresponds to *"I am a bird researcher or professional birdwatcher"*. This generates a test set of 13 041 tasks where

the expert label is used as the current truth. This test set is only used to compute the AccTrain metric. Note that we do not perform deep-learning methods as the tasks of comparing the birds from two audio files and designing specific architectures to match this framework are out of the scope of this work.

We then can run our aggregation strategies, and from @tbl-birds we see that strategies reach the same levels of label recovery, however naive they are. Indeed, most tasks have very few disagreements. Note that NS and MV performance difference comes from the random tie-breakers in case of equalities.

Table 3.5 AccTrain metric on simulated correlated mistakes considering classical feature-blind label aggregation strategies.

Method	**MV**	**DS**	**GLAD**	**NS**
AccTrain	0.954	0.946	0.950	0.960

We can explore what tasks lead to the most disagreements depending on the entropy criterion or GLAD's difficulty-estimated latent variable. Using the entropy criterion, the most difficult tasks (highest entropy) and GLAD's difficulty, we recover the index of the most ambiguous tasks. As we work with audio files, we can listen to the most ambiguous tasks and see if they are indeed difficult to classify. Audio records of such identified tasks are made available at `https://tanglef.github.io/computo_2023/#case-study-with-bird-sound-classification`.

- Entropy: we obtain the candidate `MRG18_20180514_000000_203.mp3` that was to be compared with the letter `HLO15_20180515_021439_31.mp3` (one worker agrees and another disagrees). And the candidate `MRG24_20180512_000000_437.mp3` that was to be compared with the letter `HLO12_20180511_150153_42.mp3` (one worker agrees and another disagrees)
- GLAD: we obtain the candidate `HLO04_20180511_034424_15.mp3` that was to be compared with the letter `MRG11_20180519_000000_506.mp3` (53 votes, 29 agreeing and 24 disagreeing). And the candidate `MRG27_20180512_000000_597.mp3` that was to be compared with the letter `HLO01_20180601_080126_30.mp3` (43 votes, 23 agreeing and 20 disagreeing).

In this dataset, a single task with two different votes has the highest entropy. GLAD's coefficient lets us explore tasks with multiple votes where workers were split.

We can also explore the dataset from a worker's point of view and visualize workers' performance and how many are identified as poorly performing. This gives us an idea of the level of noise in the answers.

From Figure 3.13, we notice that very few workers are identified as spammers and that different worker identification strategies seem to perform similarly. This is consistent with the high level of agreement between workers in this dataset.

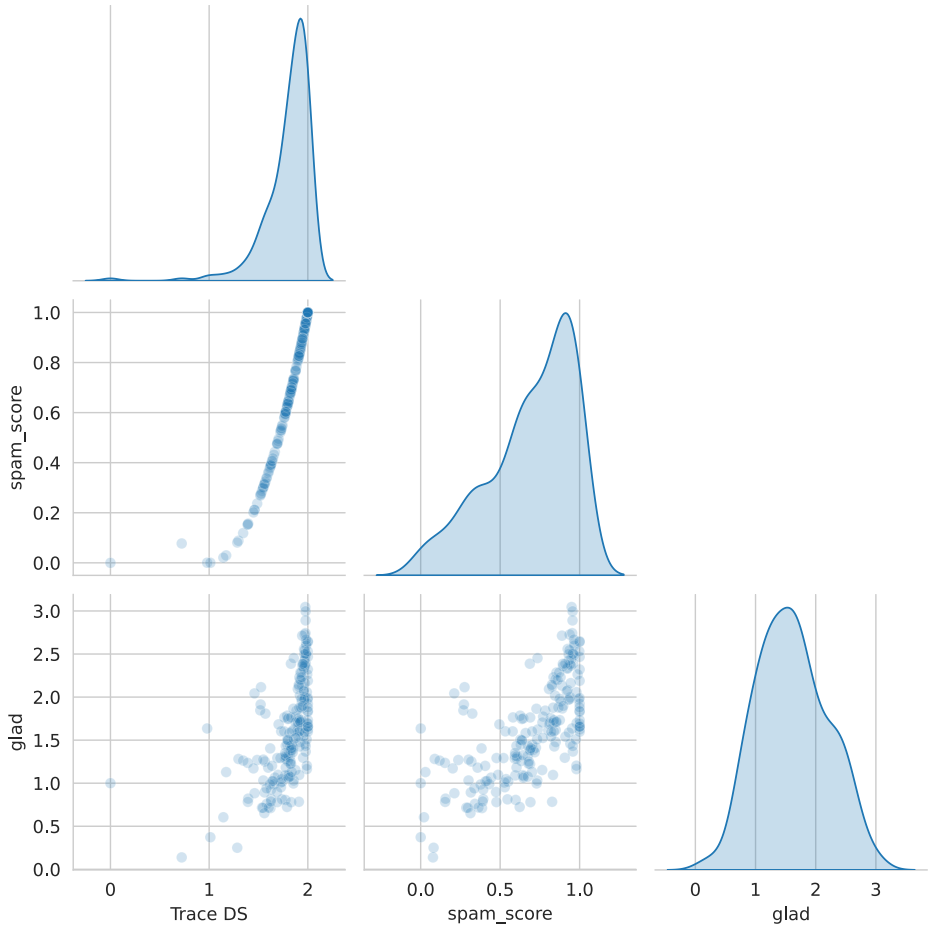


Figure 3.13 Comparison of ability scores by workers for the birds audio dataset. Most workers do seem to perform similarly, with very little noise voluntarily induced.

One of the closing statements of Lehikoinen et al. (2023) is *"we learned lessons for how to better implement similar citizen science projects in the future"*. On one hand, identifying the most ambiguous tasks can help by saving only these tasks to the most expert workers and acquiring better data. On the other hand, combining the task difficulty with the worker ability performance metrics could help to create personal feeds of tasks to label and generate more worker participation. Finally, the label aggregation step can lead to training classifiers with better labels. We hope that allowing easy access thanks to the `peerannot` library to each of those steps can indeed help to better implement citizen science projects and use the collected data.

3.1.7 Conclusion on `peerannot`

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. Going beyond templating, it helps the crowdsourcing community to have openly accessible strategies to test, compare and improve to develop common strategies to analyze more and more common crowdsourced datasets.

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[3]. 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.

3.2 Benchmarking aggregation strategies with `Benchopt`

The `Benchopt` library is an open-source benchmarking tool for optimization algorithms. It is designed to provide a fair comparison of optimization algorithms on a wide range of problems. The machine learning community has created platforms to release datasets (`OpenML` (Vanschoren et al., 2013) or `DataHub`[4]), reproducibility challenges [5] and journals (like Rescience[6] or Computo[7]) to counteract the reproducibility crisis (Baker, 2016). However, the optimization community has not yet developed a standard benchmarking tool to compare optimization algorithms. This is where `Benchopt` comes in.

3.2.1 How does it work?

Starting from an input dataset $\mathcal{D}$ and an objective function f, the `Benchopt` library considers problems of the form:

$$\arg\min_{\theta \in \Theta} f(\theta, \mathcal{D}, \Lambda) \ ,$$

where Λ is a set of hyperparameters and Θ is the feasible set for θ: the parameters in the objective. Following the iteration sequence $\{\theta_k\}_k$ generated by an optimization algorithm, the library computes the performance of the algorithm on the problem. Multiple objectives $f_1, f_2, \dots$ can be monitored, but only one is used in the optimization problem.

The library is designed to be modular and to allow easy integration of new optimization algorithms and new problems. It is not `scipy`-like an optimization module with a fixed

[3] `https://github.com/peerannot/peerannot`
[4] `https://datahub.io/`
[5] `https://reproml.org/`
[6] `https://rescience.github.io/`
[7] `https://computo.sfds.asso.fr`

set of methods, but rather a framework to compare them on a wide range of problems, add new ones and share them. Hardware components (number of CPUs, number of threads, GPU type, platform – Linux version – and libraries' version like `numpy`'s) are also stored for each benchmark.

Workflow

Each benchmark is defined by three objects. Each of them is defined as a class object in the `benchopt` library:

- **Objective**: the objective function to minimize (or maximize), its hyperparameters Λ and the set of possible parameters Θ. The objective defines the performance metrics to track along the optimization sequence.
- **Datasets**: the data $\mathcal{D}$ to be used in the objective function. The dataset can be a simple toy dataset or a real-world dataset. It is defined separately from the objective to be modulable. Datasets also define how to load and preprocess the data.
- **Solvers**: the optimization algorithms to compare. Each solver is defined by its hyperparameters and outputs a sequence of parameters $\{\theta_k\}_k$.

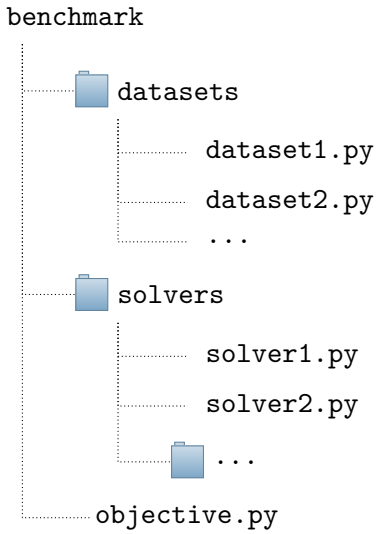


Figure 3.14 Template of a benchmark folder in `Benchopt`.

Datasets, solvers and objectives are compatible with hyperparameter settings. Solvers can have step sizes. Datasets can come with different versions or, in the case of simulations, different noise levels. Objectives can have different regularizations *etc.* Then, for each combination of dataset, solver and objective, the library computes the performance of

the solver on the objective using the dataset. These metrics are stored in a `parquet` file and then produce interactive figures using `plotly`. The full pipeline is described in Figure 3.15.

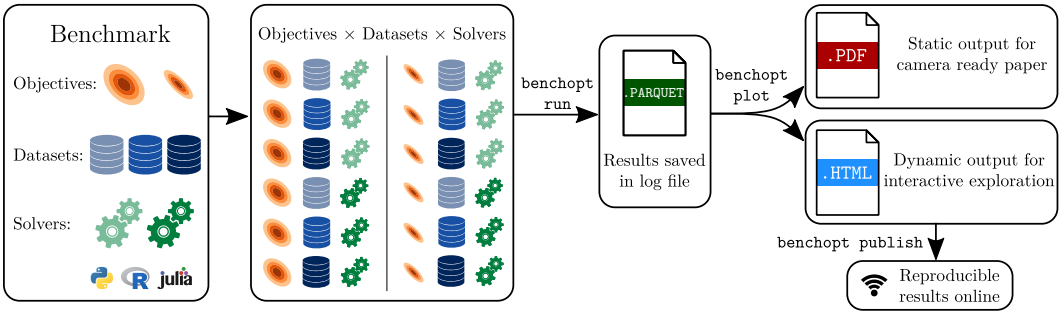


Figure 3.15 Example of a benchmarking result with `Benchopt`. The library allows to compare the performance of different solvers on a given objective function. Results are stored in a `parquet` file as visualized using interactive `plotly` webpage. They can also be shared with the community using the `benchopt publish` command. Solvers in `Python`, `Julia` and `R` are currently supported.

Benchmark structure and dependency relation. As the main goal of the library is to compare solvers on different objectives, the dependency relation between the three classes is crucial.

- Datasets are instantiated with a `.get_data()` method that returns the data $\mathcal{D}$.
- Objectives are instantiated with a `.get_objective()` method that returns all objects necessary for solvers to run. They also have a `.set_data()` method that takes as arguments the keys of the dictionary returned by `.get_data()`. It specifies how the data is used to compute the objective. Finally, the `.evaluate_result()` method computes the objective.
- Solvers are defined with a `.set_objective()` method that takes as arguments the keys of the dictionary returned by `.get_objective()`. It is the main communication between the objective and the solver. The `.run()` method computes the sequence of parameters $\{\theta_k\}_k$ following a sampling strategy and a stopping criterion (discussed hereafter). Finally, the `.get_result()` returns the estimated solution by the solver to the objective.

Iterations and stopping criteria

To evaluate each solver's performance, the library needs to define when to compute said performance during the iterative procedure. If done for each iterate, the cost of the benchmarking procedure can be high. Moreover, when to stop the optimization (using early stopping (Prechelt, 2002)) in the optimization procedure is a crucial question. Some

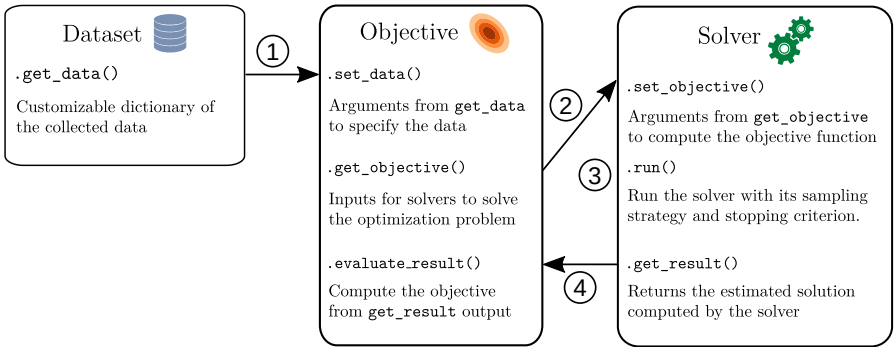


Figure 3.16 Dependency relation between datasets, solvers and objective objects. The three classes are the core of the `Benchopt` library and communicate to run the optimization problem and monitor the performance on multiple datasets. First, the data from Datasets are passed to the Objective class. Then, the objective communicates this data for the solver to be instantiated and run. Finally, the solver returns the sequence of parameters $\{\theta_k\}_k$ to the objective to compute the performance metrics.

solvers do not converge for intrinsic reasons on some datasets. The library needs to be able to handle these cases and not let a solver run indefinitely.

Iterates sampling. The sampling for which the performance is computed is defined by the `solver.sampling_strategy` attribute. It can either be `"iteration"` or `"tolerance"` depending on the way the solver itself is implemented (see Figure 3.17).

```python
def gd(grad, x0, lr, maxiter):
    x = x0
    for i in range(maxiter):
        # Compute gradient
        g = grad(x)
        # Update parameters
        theta -= lr * g
    return theta
```

Example of gradient descent implementation based on the number of iterations.

```python
def gd(grad, x0, lr, epsilon):
    x = x0
    norm_g = np.inf
    while norm_g > epsilon:
        # Compute gradient
        g = grad(x)
        # Update parameters
        theta -= lr * g
        # Update gradient norm
        norm_g = np.sum(g**2)
    return theta
```

Example of gradient descent implementation based on a threshold on the gradient's norm.

Figure 3.17 Two examples of gradient descent implementations. The first one is based on the number of iterations and the second one on a threshold on the gradient's norm. In practice, both criteria are often combined.

- iteration sampling: if the sampling strategy is set to `iteration`, the performance is computed for iterates of the solver following a geometrical growth sequence of

parameter $\rho = 1.5$. Starting from the first iterate, the next stopping iterate $\text{iter}_{\text{stop}}$ is computed as:

$$\text{iter}_{\text{stop}} = \max(\text{iter}_{\text{stop}} + 1, \texttt{int}(\rho \text{iter}_{\text{stop}})) \ .$$

- tolerance sampling: if the sampling strategy is set to `tolerance`, the performance is computed for iterates of the solver reaching a tolerance threshold. This threshold sequence follows a geometrical sequence of parameter $\rho = 1.5$. Starting from 1, the next stopping threshold $\text{thresh}_{\text{stop}}$ is computed when:

$$\text{thresh}_{\text{stop}} = \min\left(1, \max\left(\frac{\text{iter}_{\text{stop}}}{\rho}, 10^{-15}\right)\right) \ .$$

Note that if no difference is observed between two consecutive objective calls (the variation of the objective is zero), then ρ is increased by a factor of 1.2. This is to avoid computing the same objective multiple times and obtaining a flat curve. This growth can be manually changed if needed[8].

Stopping criterion. The stopping criterion is defined by the `solver.stopping_criterion` attribute. There are three possible stopping criteria and two fail-safes implemented in `Benchopt`:

- Sufficient Decrease Criterion: defined by a tolerance ε and a patience $p \in \mathbb{N}$. The solver is stopped when the relative decrease of the objective is less than ε for p evaluated samples.
- Sufficient Progress Criterion: defined by a tolerance ε and a patience $p \in \mathbb{N}$. The solver is stopped when the objective has not decreased by more than ε for p evaluated samples.
- Single Run Criterion: the performance is evaluated only once at a given value.

The fail-safes are a maximum number of solver runs and a maximum time to run the solver. They are set to 10 and 100 seconds by default, respectively.

3.2.2 Case study: benchmarking aggregation strategies in crowdsourcing

Let us run the `Benchopt` library to compare the performance of different label aggregation strategies on multiple datasets. We compare results from strategies within `peerannot`, `crowd-kit` and the Fast DS algorithms presented in Sinha et al. (2018). A description of datasets, new solvers and more points of comparison between `peerannot` and `crowd-kit` is provided hereafter.

[8] `https://benchopt.github.io/user_guide/performance_curves.html#changing-the-strategy-to-grow-the-computational-budget-stop-val`

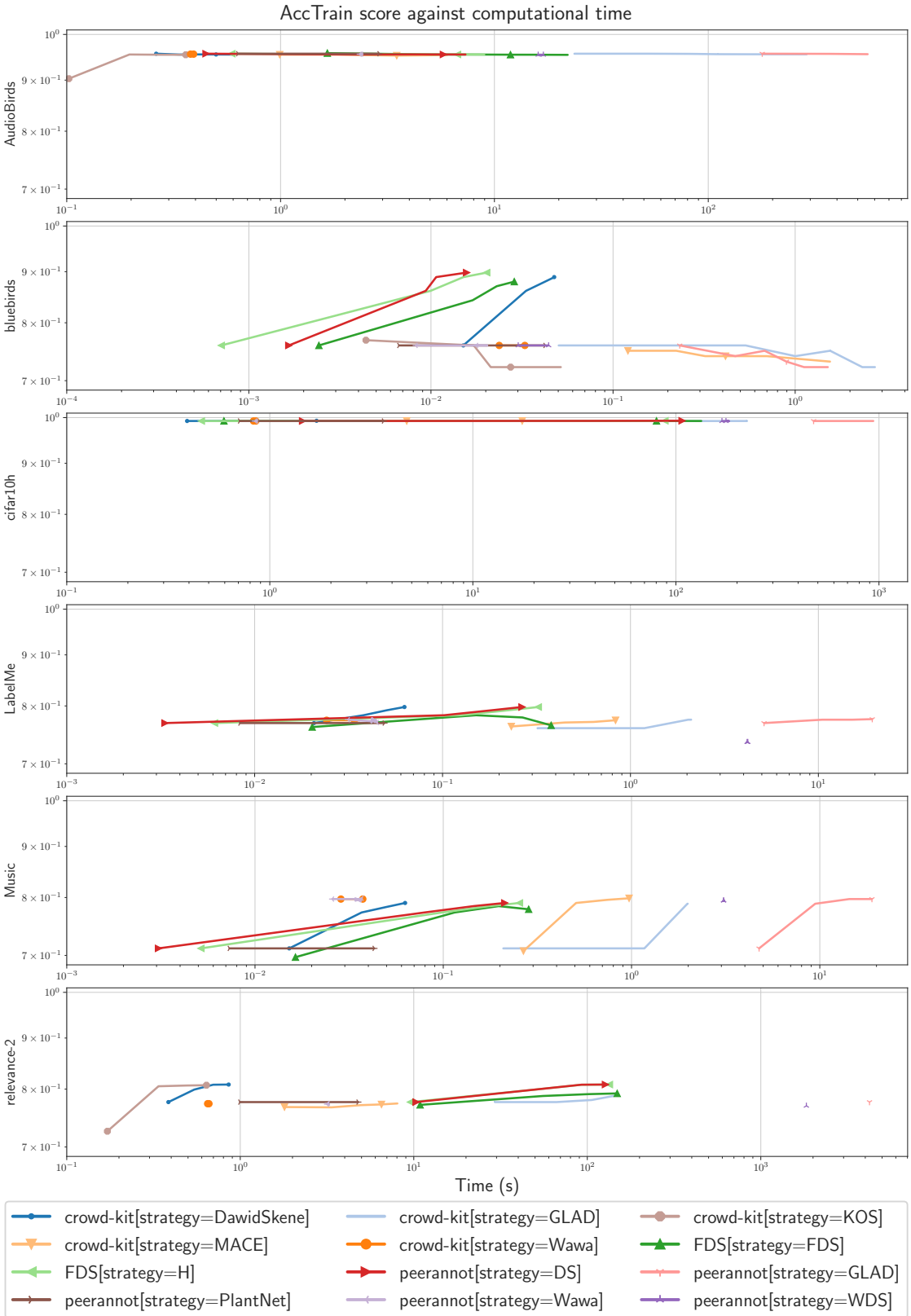


Figure 3.18 AccTrain score against computational time depending on the crowdsourced dataset and solver.

Datasets and solvers information

The list of all datasets compared in Figure 3.18 is available in Table 3.6. We use Krippendorff's $\alpha \in [0, 1]$ (Krippendorff, 1980) to measure the consistency in votes for each dataset. A value over 0.8 means a reliable dataset, between 0.6 and 0.8 the worker's answer includes inconsistencies. A low value indicates high levels of disagreement and possibly unreliable data without the proper strategies according to Krippendorff (2004).

More formally, denoting D_o the observed disagreement and D_e the expected disagreement, Krippendorff's α is defined as:

$$\alpha = 1 - \frac{D_o}{D_e} \ , \tag{3.2}$$

with D_o the observed disagreement and D_e the expected disagreement. In a classification setting, this formula writes:

$$\alpha = \frac{(n-1)\sum\limits_{c\in[K]} o_{c,c} - \sum\limits_{c\in[K]} n_c(n_c-1)}{n(n-1) - \sum\limits_{c\in[K]} n_c(n_c-1)} \ , \tag{3.3}$$

with $n = \sum_{c\in[K]} n_c$, and $n_c = \sum_k o_{c,k}$ for $k \in [K]$ and

$$o_{c,k} = \sum_{i\in[n_{\text{task}}]} \frac{1}{|\mathcal{A}(x_i)| - 1} |\{(c,k)\text{-pairs in task } i\}| \ .$$

Table 3.6 Dataset information

Dataset	n_{worker}	n_{task}	K	Krippendorff's α
AudioBirds (Lehikoinen et al., 2023)	205	79 592	2	0.810
bluebirds (Welinder et al., 2010)	39	108	2	0.126
relevance-2	7 138	99 319	2	0.262
CIFAR-10H (Peterson et al., 2019)	2 571	1000	10	0.910
LabelMe (Rodrigues et al., 2014)	77	1000	8	0.677
Music (Rodrigues et al., 2017)	44	700	10	0.301

In addition to presented solvers, we add in the comparison the Fast-DS solvers (Sinha et al., 2018), KOS (Karger et al., 2011), the Multi-Annotator Competence Estimation (MACE) (Hovy et al., 2013) and Worker Agreement with Aggregate (WAWA) [9].

[9] `https://success.appen.com/hc/en-us/articles/202703205-Calculating-Worker-Agreement-with-Aggregate-Wawa`

Computing Krippendorff's α

Let us consider a toy example with a dataset of $n_{\text{task}} = 5$ tasks and $n_{\text{worker}} = 3$ workers with $K = 3$ classes. Some workers do not get to answer some of the tasks. The votes are given as:

Worker j / Task i	**1**	**2**	**3**	**4**	**5**
1		1	1	2	1
2	0	1			0
3	0	2		2	0
4	0		1	2	2
$\lvert\mathcal{A}(x_i)\rvert$	3	3	2	3	4

The class-coincidence matrix associated to this dataset is:

Classes c	0	1	2	n_c
0	$o_{0,0}$	$o_{0,1}$	$o_{0,2}$	n_0
1	$o_{1,0}$	$o_{1,1}$	$o_{1,2}$	n_1
2	$o_{2,0}$	$o_{2,1}$	$o_{2,2}$	n_2

$\Longleftrightarrow$

Classes c	0	1	2	n_c
0	$\frac{11}{3}$	$\frac{2}{3}$	$\frac{2}{3}$	5
1	$\frac{2}{3}$	3	$\frac{4}{3}$	5
2	$\frac{2}{3}$	$\frac{4}{3}$	3	5

Let us detail two examples:

- $o_{0,0}$: In the first task there are 6 pairs of $(0,0)$ agreements over 3 answers. In the tasks $2,3,4$ there are no votes for class 0. And for task 5 there are 2 pairs of $(0,0)$ agreements over 4 answers. Hence $o_{0,0} = \frac{6}{3-1} + \frac{2}{4-1} = \frac{6}{2} + \frac{2}{3} = \frac{11}{3}$.
- $o_{2,1}$: In the tasks 1, 3 and 4 there are no $(2,1)$ disagreements. In task 2 there are 2 pairs of $(2,1)$ disagreements over 3 answers. And in task 5 there is 1 pair of $(2,1)$ disagreements over 4 answers. Hence $o_{2,1} = \frac{2}{3-1} + \frac{1}{4-1} = \frac{2}{2} + \frac{1}{3} = \frac{4}{3}$.

Finally, we use Equation (3.3) and obtain with $n = 5 + 5 + 5 = 15$:

$$\begin{aligned}\alpha &= \frac{(n-1)\left(\frac{11}{3} + 3 + 3\right) - (5(5-1) + 5(5-1) + 5(5-1))}{n(n-1) - (5(5-1) + 5(5-1) + 5(5-1))} \\ &\simeq 0.502 \ .\end{aligned}$$

Hence this data is quite unreliable.
Note that in settings other than classical classification (*e.g.* ranking) the coincidence matrix might not be symmetrical. See Krippendorff (1980) for forms of α expression instead of Equation (3.3).

Let us present the additional label aggregation strategies briefly. The PlantNet aggregation is presented in detail in Chapter 4.

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

$$\text{WAWA}(i, \mathcal{D}) = \arg\max_{k \in [K]} \sum_{j \in \mathcal{A}(x_i)} \beta_j \mathbb{1}(y_i^{(j)} = k)$$
$$\text{with } \beta_j = \frac{1}{|\{y_{i'}^{(j)}\}_{i'}|} \sum_{i'=1}^{n_{\text{task}}} \mathbb{1}\left(y_{i'}^{(j)} = \text{MV}(i', \{y_{i'}^{(j)}\}_j)\right) \quad .$$

Note that even if they are not iterative strategies WAWA and WDS can still be shown in `Benchopt`. However, the performance is computed at the end of the aggregation process, and only the first point of the curve should be considered. Indeed, the second point is an artifact from `Benchopt`'s stopping criterion (the patience) described in Section 3.2.1.

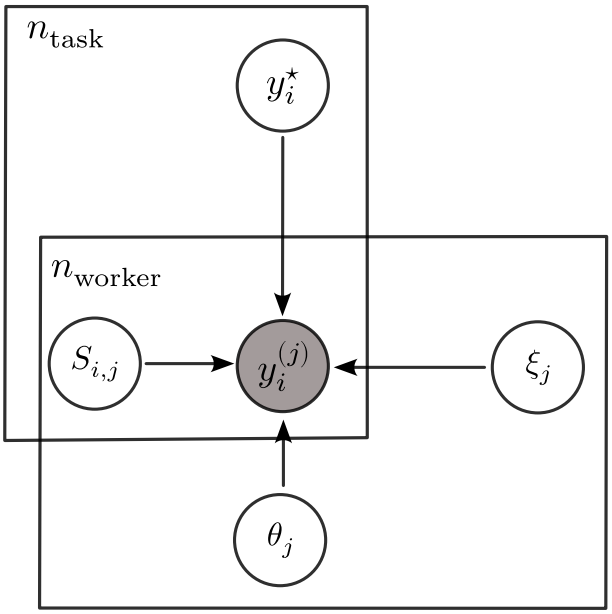


Figure 3.19 Bayesian plate diagram representation of the MACE model. Only the labels $\{y_i^{(j)}\}_{i,j}$ are observed. Latent variables to estimate are the true labels $(y_i^\star)$, the spamming state $(S_{i,j})_{i,j}$, the probability to spam $(\theta_j)_j$ and the worker behavior when spamming $(\xi_j)_j$.

MACE (Hovy et al., 2013). The MACE strategy has been designed to be more robust against spammers. Contrary to the definition of spammers by Raykar and Yu (2011), they here consider spam as a state of the worker.

Each worker can be spamming on each task. The probability for a worker to spam a task is given by a Bernoulli distribution with parameter $1 - \theta_j \in [0, 1]$. Thus, denoting $S_{i,j}$ the spamming state of worker j on task i, the probability of the worker to spam the task is drawn as $S_{i,j} \sim \mathcal{B}(1 - \theta_j)$. If a worker is not spamming ($S_{i,j} = 0$) then they answer the correct underlying label $y_i^\star$. If not, their answer is drawn at random from a multinomial distribution with parameter $\xi_j \in \Delta_K$ proper to each worker. For a given worker j_0, the distribution ξ_{j_0} indicates what is their behavior when they are spamming.

The associated likelihood – maximized using the EM algorithm as in DS – writes:

$$\mathbb{P}(\{y_i^{(j)}\}_{i,j}|\theta,\xi) = \sum_{y^\star\in[K]^{n_{\text{task}}}} \sum_{S\in\{0,1\}^{n_{\text{task}}\times n_{\text{worker}}}} \left[\prod_{i=1}^{n_{\text{task}}} \mathbb{P}(y_i^\star) \prod_{j=1}^{n_{\text{worker}}} \mathbb{P}(S_{i,j}|\theta_j)\mathbb{P}(y_i^{(j)}|S_{i,j},y_i^\star,\xi_j)\right] .$$

In Figure 3.19, we represent the associated bayesian plate diagram.

KOS (Karger et al., 2011). Only set for binary classification $y_i^\star \in \{\pm 1\}$, the KOS strategy from a graph-theory perspective. Denote $G([n_{\text{task}}] \cup [n_{\text{worker}}], E)$ the graph where edge (i,j) is connected if worker j has answered task i. The neighborhood of a task i is denoted ∂_i and the same with index j for workers. On each edge is stored the label answered by worker j for task i: $y_i^{(j)} \in [K]$.

The KOS strategy estimates how much a worker is reliable by how much their answers are consistent with the answers of their neighbors and the likelihood of a task having $y_i^\star = 1$. Both pieces of information are propagated into the graph as messages. The final label is then estimated by the sign of the sum of the messages received by the task weighted by the workers' reliability.

A task message denoted $x_{i\to j}$ is the log-likelihood of the task i having $y_i^\star = 1$. A worker message denoted $y_{j\to i}$ is the reliability of worker j. Both messages have scalar values. After random initialization, the messages are updated iteratively until convergence following the equations:

$$\begin{aligned} x_{i\to j} &\leftarrow \sum_{j'\in\partial_i\setminus\{j\}} y_i^{(j')} y_{j'\to i} \;\forall (i,j)\in E \\ y_{j\to i} &\leftarrow \sum_{i'\in\partial_j\setminus\{i\}} y_{i'}^{(j)} x_{i'\to j} \;\forall (i,j)\in E \ . \end{aligned}$$

Finally, the estimated label is computed as:

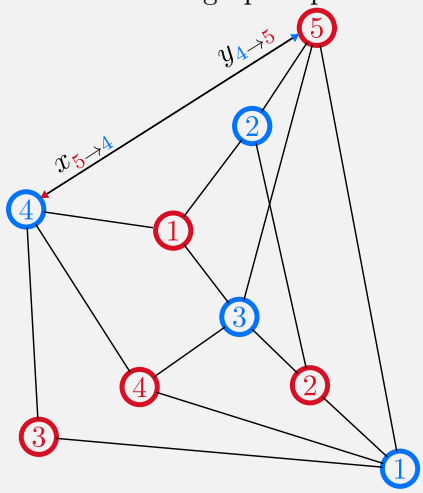
$$\forall i \in [n_{\text{task}}],\ \hat{y}_i^{\text{KOS}} = \text{sign}\left(\sum_{j\in\partial_i} y_i^{(j)} y_{j\to i}\right) .$$

Visualizing KOS.

Let us consider the binary answers:

Worker j / Task i	1	2	3	4	5
1		1	1	-1	1
2	-1	1			1
3	1	1		1	1
4	-1		1	-1	-1

The associated graph representing workers in blue and tasks in red is:



Each task sends a message $x_{i\to j}$ to each worker who answered them. Each worker sends a message $y_{j\to i}$ to each task they answered.
For example, task 5 sends messages to workers $4, 3, 2$ and 1
Thus $\partial_5 = \{1, 2, 3, 4\}$.
Worker 4 sends messages to tasks $1, 3, 4$ and 5.
Thus $\partial_4 = \{1, 3, 4, 5\}$.

At a given step, the reliability of worker 4 concerning their answer to task 5 is thus updated as follows:

$$y_{4\to5} = \sum_{i'\in\partial_4\setminus\{5\}} y_{i'}^{(4)} x_{i'\to4} = -1 \times x_{1\to4} + 1 \times x_{3\to4} + (-1) \times x_{4\to4} + (-1) \times x_{5\to4} \ .$$

Fast-DS (Sinha et al., 2018). As the DS strategy is known to have a high computational cost, Sinha et al. (2018) proposed a faster version of the DS strategy. The Fast-DS strategy is based on the same principle as DS, but the computation of the posterior distribution on the estimated ground truth is restricted as a Dirac distribution. Note that this does not alleviate the memory issue of the DS strategy – the $K^2 \times n_{\text{worker}}$ parameters to estimate for the confusion matrices. Under the assumption that there is only a single label per task – which is the case in this thesis – this *hard* version of the DS strategy claims to be faster than the original DS strategy while achieving similar performance in accuracy.

More formally, following the DS notation from Section 1.2.3 the Fast-DS method

adds the following step in the ground truth estimation:

$$T_{i,k} = \begin{cases} 1 & \text{if } k = \underset{k' \in [K]}{\arg\max}\, \mathbb{P}\left(y_i^\star = k' | \{y_i^{(j)}\}_{j \in \mathcal{A}(x_i)}\right) \\ 0 & \text{otherwise} \end{cases} .$$

The quantity $\mathbb{P}\left(y_i^\star = k' | \{y_i^{(j)}\}_{j \in \mathcal{A}(x_i)}\right)$ is estimated using Baye's theorem.

However, empirical results show that the Fast-DS strategy with this *hard* decision can reach lower accuracy. To alleviate this convergence issue, Sinha et al. (2018) proposed to use a *hybrid* version of the Fast-DS strategy. The hybrid strategy is defined as:

- Run the classical DS algorithm first to estimate the marginal class likelihood $\rho \in \Delta_K$ at each iteration.
- When the absolute difference between two consecutive iterates of ρ is lower than a given threshold, switch to the *hard* version of the DS strategy.

This hybrid strategy allows having a softer control on the label estimation at the beginning and better exploring the latent variable's space. Finishing with *hard* labels allows faster convergence. However, in practice this gain – as seen in Figure 3.18 – can be mitigated by the implementation of the DS strategy.

Results interpretation. Running `Benchopt` evaluation, we obtain Figure 3.18 showing the accuracy of the different label aggregation strategies against the computational time. Depending on the dataset, some solvers – *i.e.* label aggregation strategy – are more suited and less computationally expansive than others.

For example, on the `BlueBirds` dataset, `peerannot`'s DS strategy is the fastest to reach the best accuracy. It is followed by the Fast-DS and then `crowd-kit`'s DS aggregation. As the α value is very low ($\alpha = 0.126$), the dataset is unreliable. Hence, strategies that are specific to some workers' behaviors – KOS for spammers, GLAD for ambiguous tasks – are not suited and the DS strategy seems to be the best choice.

In the case of datasets with a high level of agreement ($\alpha > 0.8$) – in AudioBirds and CIFAR-10H – all strategies reach the level of performance. Notice that GLAD is again slower than most strategies (this conclusion can be seen in most cases). For those datasets, the simpler the strategy, the faster the same level of accuracy.

For the `Music` dataset, the best performance is achieved by the fastest method in this case: WAWA (with similar time computation between `peerannot` and `crowd-kit`). And finally, for the `LabelMe` dataset, the best performance is achieved by the DS-based strategies. WAWA aggregation underperforms: as shown in Figure 1.11, the tasks have very few labels and are ambiguous with the class overlap.

Overall, the Pl@ntNet strategy does not perform better than other label aggregation strategies. This is because the algorithm is not suited for such crowdsourced datasets. It is based on the assumption that the number of possible classes is high to evaluate the worker's reliability. This is not the case in the datasets we consider. We have at most

$K = 10$. Hence why in Chapter 4 we released a large subset of Pl@ntNet's database to evaluate this strategy on a more suitable dataset.

More comparison between `peerannot` and `crowd-kit`.

Both `peerannot` and `crowd-kit` are open-source libraries to handle crowdsourced datasets. While `crowd-kit` proposes segmentation and regression crowdsourcing strategies, `peerannot` is focused on classification tasks. The learning strategies (CrowdLayer and CoNAL) are available in both libraries, and each library has label aggregations strategies with some in common (MV, GLAD, DS, WAWA) and some library-specific (Pl@ntNet, WDS, KOS, MACE). However, the main difference between the two libraries is the identification step. So let us focus on this.

At the time of writing `crowd-kit` proposes four strategies to explore crowdsourced datasets:

- Krippendorff's α: to measure the consistency in votes for each dataset (also available in `peerannot`).
- The consistency: Averaged posterior distribution of a label determination from workers' estimated reliability.
- The uncertainty: the entropy of the distribution of the votes (also available in `peerannot`).
- The accuracy of aggregates: the accuracy of a subset of worker's answers from aggregated labels (if the subset of workers is all workers and the aggregated label is the MV label, this simplifies as the weight of users in WAWA).

In `peerannot`, we added:

- The AUMC: deep learning based ambiguity score on tasks via MV aggregation (see Chapter 2).
- The WAUM: deep learning based ambiguity score on tasks via weighted Areas Under the Margin (see Chapter 2).
- The spam score (Raykar and Yu, 2011): distance between the estimated confusion matrix and the closest rank-1 matrix for each worker. This allows identifying potential spammers.
- The trace confusion: sum of the diagonal terms of the DS confusion matrices. Indicates the level of reliability per worker for their answered label. The closer the trace to K, the more reliable the worker.

The `peerannot` also supports top-k classification metrics for deep-learning methods. We provide the `train` command to train deep learning models from `TorchVision` on the dataset and evaluate their accuracy and calibration if a test set is available. This allows users to have a fully end-to-end pipeline:

- identify potential ambiguous tasks / bad workers,
- aggregate the labels and train a computer vision network,
- or directly train a computer vision network with deep-learning-based strategies to handle crowdsourced tasks.

Keeping the same framework from beginning to end, based on PyTorch, allows for a more seamless integration of the crowdsourcing pipeline in existing projects.

As we will see in Chapter 4, the `peerannot` library is more suited for classification tasks with a high number of classes, tasks and/or workers. Indeed, for the dataset we will consider, none of the aggregation strategies from `crowd-kit` could be run due to memory issues that are directly linked to the data format choice.

3.3 Conclusion

We presented the `peerannot` library: an open-source Python library to handle crowdsourced datasets. This library is designed to be modular and to allow easy integration of new label aggregation strategies, learning strategies and identification strategies on new datasets.

To produce fair and reproducible comparisons of label aggregation strategies' performance, we added a new benchmark to the `Benchopt` library. This benchmark for crowdsourcing in classification problems allows us to compare the performance of different label aggregations depending on the ambiguity of a dataset. We showed that the `peerannot` library is a good candidate to handle crowdsourced datasets against standalone openly available methods (FDS) or other libraries like `crowd-kit`.

4 Crowdsourcing strategies in Pl@ntNet citizen based learning platform

Key points – Crowdsourcing plant species

1. Aiding botanists in plant species identification is a challenging task. Species are often visually close and their identification requires expert knowledge.
2. Pl@ntNet is a citizen science platform that allows users to upload images of plants and receive a list of possible species.
3. The collaborative aspect of Pl@ntNet allows users to vote on the species they think are present in the image and contribute to new labeled data, which is then used to train a computer vision and help new identifications.

Contributions – Exploration of Pl@ntNet label aggregation strategy

4. We release and evaluate the current Pl@ntNet label aggregation and compare it to other strategies.
5. We release a subset of Pl@ntNet with images url and collected labels in the South Western European Flora of more than 6 million observations and 800 thousand users in a large-scale classification setting.
6. We discuss how to integrate the current model's predictions in the votes aggregation. This is a challenging task due to the iterative aspect of Pl@ntNet: the current data helps train the next generation of models iteratively.

Chapter 4 – Pl@ntNet:

While crowdsourcing is advertised to collect a large number of data easily, with strategies to mitigate the noise, openly available datasets are still scarce and with a low number of classes ($K \leq 10$ in general). In this chapter, we focus on the Pl@ntNet platform, a citizen science platform for plant species recognition. One of the challenges in this classification setting is the large number of possible species ($K > 10^4$). After presenting the platform and voting system, we will investigate the current label aggregation strategy in Pl@ntNet. We compare it with other strategies that can handle this large number of classes, workers and tasks. Finally, we propose to improve the current algorithm by taking advantage of the Pl@ntNet pipeline.

Note that in previous chapters, the crowdsourcing experiments concerned workers paid to answer multiple tasks. In Pl@ntNet, contributions are by volunteers and the tasks are not paid. We thus slightly adapt the vocabulary used in this chapter to reflect this difference. People voting are called users. The tasks are observations of plants (detailed in Section 4.1.2) for which we wish to identify the species.

4.1 Crowdsourcing for plant species identification

Computer vision models are a great aid in plant species recognition in the field (Vidal et al., 2021; Borowiec et al., 2022; Mäder et al., 2021). However, to train them one needs large annotated datasets. These datasets are usually created thanks to crowdsourcing, but citizen science approaches, collecting both reliable and useful information (Brown

and Williams, 2019; Wright et al., 2021), tend to be more and more popular. Among existing plant recognition applications, the Pl@ntNet citizen science platform (Affouard et al., 2017) enables global data collection by allowing users to upload and annotate plant observations (Bonnet et al., 2020).

We first introduce the problem of labeling a plant observation and the complexity of plant taxonomy. Then, we present the Pl@ntNet platform and its voting interface. Finally, we discuss the current label aggregation strategy and propose ways to improve it.

4.1.1 Plant taxonomy generalities

First, we need to understand the complexity of plant taxonomy. Our goal here is to briefly present this taxonomy. Plants are divided following a hierarchy, from the most general to the most specific ranks of taxa: kingdom, division, class, order, family, genus, and species according to the International Code of Nomenclature for algae, fungi, and plants (ICN) (Turland et al., 2018). Each of these units of biological classification is called a taxon (taxa in plural). Further secondary ranks also exist (tribe, subspecies, variety, form) but we will focus on the main ones.

Roughly, an example of taxonomy levels is:

- Kingdom: separates plants from animals, fungi, and bacteria – *e.g* Plantae.
- Division: separates spore (*angiosperms*) or seed (*gymnosperms*) reproduction with specific characteristics. There are 14 plant divisions in total.
- Class: Angiosperms are divided into Monocotyledons(grasses, yuccas, etc.) and Dicotyledons (angiosperms with pair of leaves).
- Order: Group of families with common characteristics – *e.g Cucurbitales* (generally ends with *-ales*).
- Family: Plants with similar flower, fruit and seed structures – *e.g Begonaias and Allies*.
- Genus (genera in plural): First part of the plant's scientific name (capitalized and italicized) – *e.g Begonia*.
- Species: a group of organisms capable of producing fertile offspring – *e.g Begonia ferox*. If the species is unknown it is called *sp.* or *spp.* for plural.

In the case a plant is a hybrid – a cross between two species – it is written with an × as in *Begonia* × *semperflorens*. The position of the × indicates if it is the hybridization between two genera or species. For example, the ×*Agroelymus hajastanica* is the hybridization of the *Agropyron cristatum* and the *Elymus repens* – note that the genera are aggregating into a resulting genus. Similarly to the ×, a + is used to

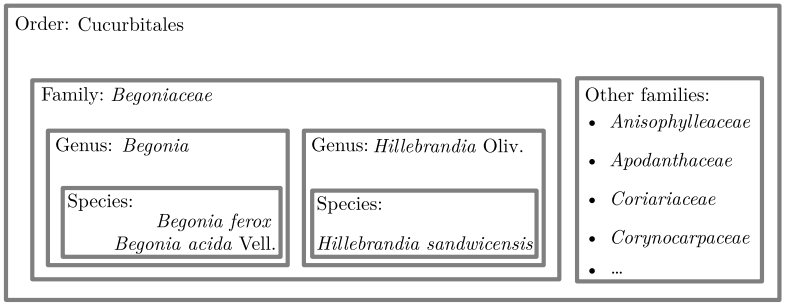


Figure 4.1 *Begonia ferox*: a species from the *Begonia* genus of the *Begoniaceae* family of the Cucurbitales order.

indicate a transplant chimera between two plants. For example, the + *Laburnocytisus adamii* results of the transplant of two individuals with different genera: the *Laburnum anagyroides* and the *Cytisus purpureus*.

Checklists and referentials

The plant name is generally composed of only two parts: the genus and the species. In this work, we do not consider vernacular names – the common names of plants – as they can vary from one region to another. Botanists have created checklists of accepted plant species.

One of the largest is the International Plant Names Index (IPNI) (IPN, 2024). The IPNI is a database of the published names and indicates which names are validly published. It does not take into account the taxonomic status of the names – the chronological changes and synonymity. The World Checklist of Vascular Plants (WCVP) (Govaerts et al., 2021) is an international collaborative database of taxa that provides the latest nomenclatural and taxonomic information on vascular plants – *clubmosses*, *horsetails*, *ferns*, *gymnosperms* (including conifers), and *angiosperms* (flowering plants). The data from WCVP is the backbone for Plants Of the World Online (POWO)[1] which is an interface to access more information on individual species as their distribution for example.

There are multiple other checklists and databases such as The Plant List (TPL) (Pla, 2013), the Catalogue of Life (CoL) (Cachuela-palacio, 2006), Leipzig Catalogue of Vascular Plants (LCVP) (Freiberg et al., 2020) *etc.* For a more exhaustive comparison of these databases we refer the reader to Schellenberger Costa et al. (2023). The Global Biodiversity Information Facility (GBIF) (Telenius, 2011) is an openly accessible network of shared knowledge about biodiversity on Earth. They currently regroup 105 different sources of taxonomy[2].

These checklists are the backbone of any botanical project as they provide a reference for the species names, their taxonomic status, and synonymic contributions. Synonyms

[1] `https://powo.science.kew.org`

[2] `https://www.gbif.org/fr/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c`

are not rare in plant taxonomy and can be due to different reasons such as the same species being described by different botanists, at different times, in different regions of the world. As an example, in WCVP there are 357 347 plant species and 565 200 synonyms for those species.

Moreover, note that checklists might not always cover all the species and synonyms from one another. For example, the *Pilosella officinarum* Vaill. is listed in POWO with 161 possible synonyms – 141 accepted synonyms and 20 illegitimate – while in the GBIF it is listed with 241 possible synonyms. In the Pl@ntNet database, synonyms are only considered at the species level, resulting in 22 possible accepted synonyms for this species.

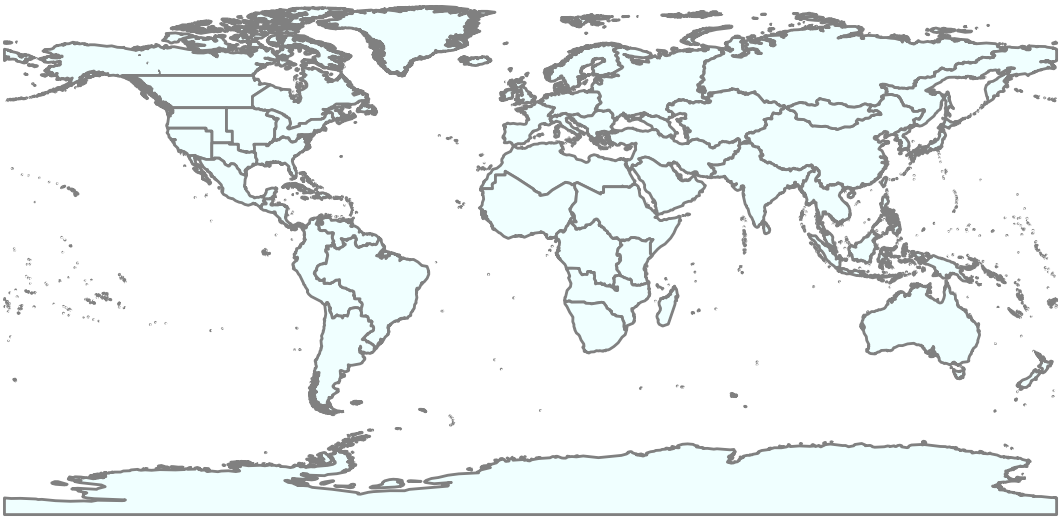


Figure 4.2 Regional areas at level 2: subcontinental botanical regions of the world.

The international system groups areas of the world at different levels to record plant distributions and have standardized comparisons. This system is called the World Geographical Scheme for Recording Plant Distributions (WGSRPD) (Brummitt et al., 2001). The first level represents the nine botanical continents – Europe, Africa, Asia-Temperate, Asia-Tropica, Australasia, Pacific, Northern America, Southern America and Antarctic. The second level is the subcontinental regions of the world – see Figure 4.2 for a representation. It divides each botanical continent between 2 and 10 subdivisions. For example, Europe is divided into Northern, Middle, Southwestern, Southeastern and Eastern Europe. Levels 3 and 4 include states, provinces or botanical areas. In Pl@ntNet, level 2 is used to filter the species that can be observed in a given area and adapt the models' predictions. In addition, users can specify to which flora – either a level 2 subdivision or a Pl@ntNet project like "Useful plants" or "World Flora" – they wish to associate their contribution.

Figure 4.3 Quadrats used in the field to estimate the abundance of species. They are often in a hard rectangle shape material like wood or soft with meters of ribbon to delimit the surface. (©Pierre Bonnet)

4.1.2 What is a plant observation?

Contrary to classical datasets, Pl@ntNet observations are not a single image but a set of images taken by a user in the field. A single image of a plant might not be enough to identify the species. More images of different parts of the plant – leaves, flowers, fruits, *etc.* – are needed to correctly identify the species and mimic a botanist's behavior while it is surely not the same and we can not replace such expertise.

Observation of a *Scabiosa columbaria* L. (©Llandrich anna)

Observation of a *Knautia arvensis* (L.) Coult. (©Francois Mansour)

Figure 4.4 Two observations of different species known to be difficult to identify from one another: a *Scabiosa columbaria* L. and a *Knautia arvensis* (L.) Coult. The flowers are visually close, but the leaves from the bottom view on the *Scabiosa* and the structural differences in the second photo allow a clear identification.

In the field, some recommendations include[3] but are not limited to:

- the type of plant: a shrub, an herb, a fern, *etc.*
- physical traits of the plant – height, color range, hairy (length, width), texture (velvety, rough, smooth), is it stingy, thorny? *etc.*

[3] `https://ibis.geog.ubc.ca/biodiversity/eflora/identification.html`

- position and number of leaves, flowers (symmetrical, shape), fruits or seeds (size, flesh), *etc.*
- blooming period
- the type of bark – smooth, rough, flaky, color, color changes depending on the season
- the roots – if safely available for both the identifier and the identified plant – to see rooted stems, rhizomes, bulbs or tubers. A horizontal expansion of the roots can be a sign of a rhizome.
- the aroma (minty, pungent)
- grown habits – bushy, sprawling around, erected, vine-like
- other species around, abundance of the species
- ...

Once these criteria are gathered, the botanist can identify possible species. To filter out the possible species, the botanist will also consider the habitat – sand, marshes, rocky fields – the elevation – sea level, mountain – and the region to narrow the confusion. The use of a determination key can also help the identification.

In practice, botanists use quadrats – plots of ground of a specific size – to count the number of species in a given area and estimate the abundance of each species. Such quadrats are shown in Figure 4.3.

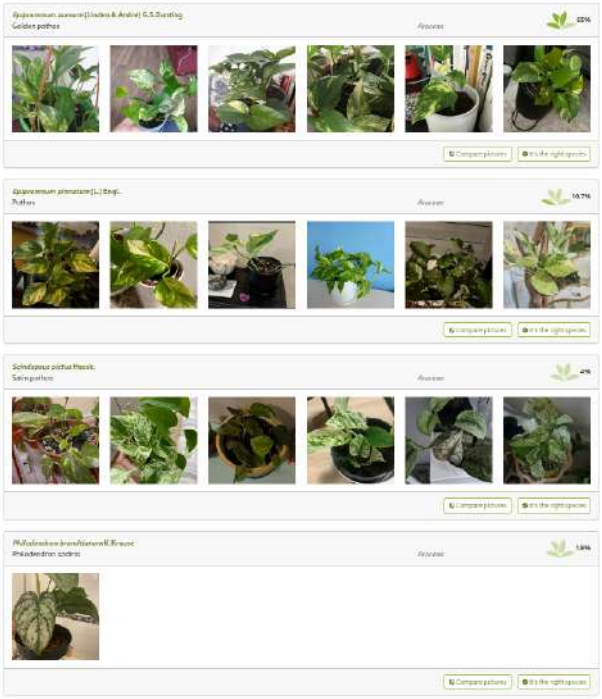
Most of these criteria (like the aroma, texture or habitat), can be difficult or even impossible yet to be gathered from multiple images, and much less from a single one. Hence, the use of multiple images is crucial to correctly identify the species or at least narrow down the possible species. Botanists will observe multiple organs (flower, fruit, leaf), from different angles (a flower from the bottom view can have specific traits not visible from the side). These characteristics can be transmitted via multiple images for the same observation. We give an example in Figure 4.4 of two species that are visually close but can be distinguished using multiple images in the same observation.

4.1.3 Presenting the voting interface

Author contribution.

In Pl@ntNet, an observation is a set of images taken by a user in the field. The author user can then upload these images to the platform. The platform will then propose a list of possible species for the observation with the associated predicted probabilities and other images that are close-looking to the picture taken. The author can then vote on the species they think are present in the observation as illustrated in Figure 4.5.

Observation of a plant which necessitates identification (©Tanguy Lefort). The image is sent to Pl@ntNet's computer vision model for identification. A similarity search is performed to present similar pictures for each proposed species.



Pl@ntNet identification results

Figure 4.5 From an image query, Pl@ntNet outputs possible species. For visual aid, close-looking images are also shown for each proposed species. The user can then click on the *compare picture* button to make their choice. Then, the user can vote on the species by clicking on the *It's the right species* button. Here the probability of the *Epipremnum aureum* (Linden & André) G.S.Bunting to be the right species is 65% – also known as *Pothos aureus* Linden & André.

The vote is guided by the computer vision model predictions (see Section 4.1.4) and other pictures selected via a similarity search algorithm. This similarity search is important as the same species can have different visual presentations at multiple stages of its life cycle. The similarity search is based on the features from the last layer of Pl@ntNet's network. These features are hashed using the Random Maximum Margin Hashing algorithm (Joly and Buisson, 2011) and then an approximate k-nearest neighbors is applied (Joly and Buisson, 2008).

If a user disagrees with the model's predictions, they can enter the name of the species freely in the input box as shown in Figure 4.6. The user input is guided by the name of possible species in the currently selected flora.

The author can also input any text freely – hence adding possible noisy votes. At this stage, there is no filter performed on the existence of the given species in the database. The author can also share the location of the observation, and add additional inputs in a specific reserved field – the habitat, the aroma or any other feature they would like to share. Some users unfortunately might mistake the different fields and input information

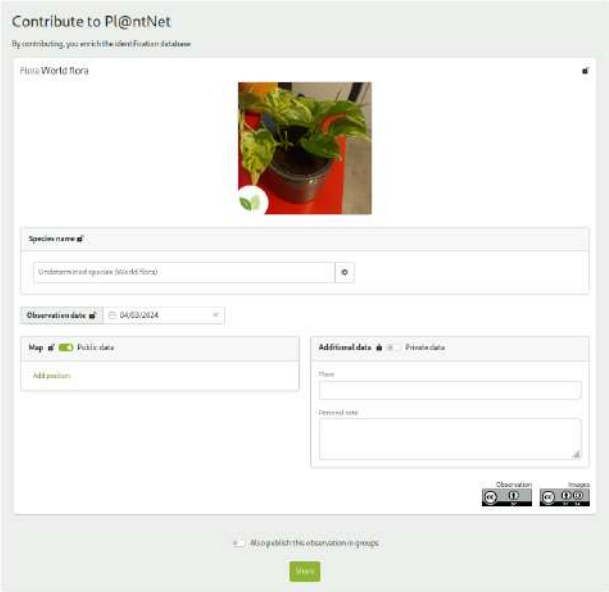


Figure 4.6 The author can select one of the model's predicted species or input freely the name of a species. The user input is guided by the name of possible species in the currently selected flora (here World Flora). Note that by clicking on the gear icon, a user can change the flora.

as the species name, a cleaning step is thus a necessary step in the platform.

Other users contributions.

Once the observation is shared online, it is time for others to enter the collaborative playground. Any user can visit the *identify* Pl@ntNet platform and vote on observations. The view is as presented in Figure 4.7.

The initial author determination – if existing – is shown, with the current prediction. In the *suggested name* box, all registered species votes are shown with the number of agreements. In Figure 4.7, 5 different users voted: 3 for the species *Vesalea grandifolia* Villarreal Hua Feng Wang & Landrein, 1 for the *Zabelia triflora* (R.Br. ex Wall.) Makino ex Hisauti & H.Hara and one user voted for *undertermined* – meaning they could not identify the species or there is not enough information provided to their knowledge to identify the species. There is a field for a new vote, where the user is guided to propose an existing species but can enter any text freely. By clicking on the Pl@ntNet icon next to it, the model's predictions are shown with the associated probabilities and close-looking examples to guide the identification as in Figure 4.5.

In addition to the species, users (as authors) can vote on the organ shown in the image(s). The organs can be a leaf, flower, fruit, bark, habit (form in which the plant grows) or other. Users can approve or disapprove the quality of the image – if the plant is not correctly visible, the image is blurry, or the organ is not visible. If there is no plant in the image, they can also flag it with the *no plant* button.

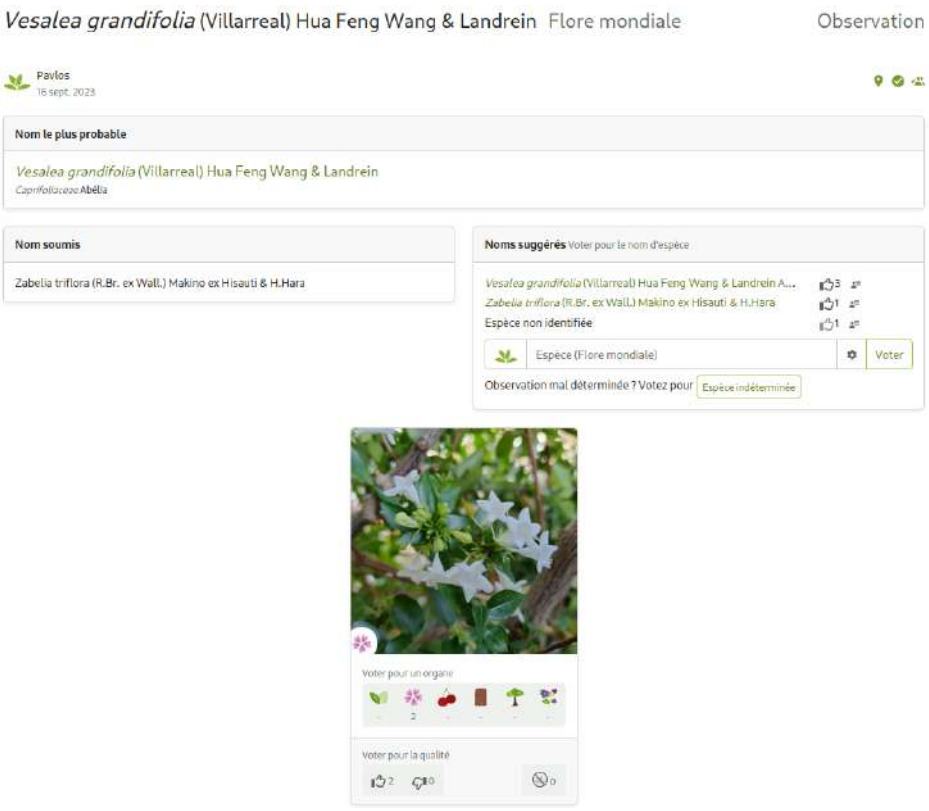


Figure 4.7 View of observations as a user (©Pavlos). The current prediction is shown, with the initial species proposed. Suggestions from other users are on the right, with a field to propose your thoughts or accept someone else's. Anyone can vote for the organ shown in the image – in this case, users agree on the flower. The quality is accepted by 2 users and there is no vote to reject the observation based on the fact there is no plant in the actual image.

In the upper-right section of the presented observation, the user can see if the observation was associated with any geolocalisation, if the observation is considered as **valid** by the Pl@ntNet algorithm and if it was reviewed by others. The *valid* status is given by the Pl@ntNet algorithm presented in Section 4.2.1 and is a proxy for the quality of the observation.

The iNaturalist platform uses a research grade status to indicate the quality of the observation[4]:

- there is a picture or recording of sound,
- the genus and/or species name of the organism are specified,
- the GPS coordinates are associated,
- the observation is timed and dated,

[4] `https://www.inaturalist.org/posts/39072-research-grade`

- the observatory is identified,
- there is at least a 2/3 agreement consensus by users on the identified species.

In Pl@ntNet, the data quality is monitored by this **valid** status that depends on the estimated reliability of a user and the agreement of the community on the observation. We explain in more detail the algorithm used to estimate the reliability of a user in Section 4.2.1.

What is not an observation.

Until now, we described what is an observation. So, let us show examples of what isn't an observation.

A drawing is not a plant (©Katyykk)

A fungus is not in the Plantae kingdom (©eugenio perez perez)

A video game tree of a birch tree) is not a real plant. (©PA0L0_D1_B3LL0)

(Minecraft version

Figure 4.8 Three examples of images that are not plant observations.

More generally, drawings – even realistic – of plants or photos of images of plants are not considered plant observations. The same goes for fungi, which are not part of the Plantae kingdom and thus are not identified by Pl@ntNet. Finally, images of video games – even if they represent plants – are not considered plant observations. We show examples of these in Figure 4.8. Photos of people, animals, pornographic images or any other object that is not a plant are also not considered plant observations. These images are flagged, removed from the platform and are not used to train the models.

Another possible issue is the presence of multiple species in the same observation. These observations are flagged as *malformed*, we provide an example in Figure 4.9. Users can cast their vote to indicate that the observation is malformed and should not be considered.

4.1.4 A step in a bigger pipeline

The gathering and aggregation of votes is only a part of the Pl@ntNet pipeline. Schematized in Figure 4.10, the Pl@ntNet system is a positive loop between curating the data

Figure 4.9 Malformed observation: each image is associated with a different species (©Leonardi Liberati).

and training a computer vision model. The label aggregation algorithm is presented in detail in Section 4.2. Let us briefly address the computer vision model part.

Model training.

At the time of writing, the model in use in Pl@ntNet is DINOv2 (Oquab et al., 2024) a transformer-based network, trained on Jean-Zay cluster generally every 2 to 3 months[5]. The training steps have evolved in the past years. In the next paragraphs, we present in a non-exhaustive way the pipeline (presented in Figure 4.11) for the current model training.

The first step is to gather the data from the Pl@ntNet platform and aggregate the labels as presented later in Section 4.2. Then, a first cleaning step is performed to remove the noisy labels – species not in any botanical checklist such as WCVP – and only keep **valid observations**. Roughly speaking, valid observations are observations that are considered reliable by the Pl@ntNet algorithm thanks to enough users' expertise and agreement in votes. In addition to species, the organ is also kept for training.

The data is then cleaned by collected species. In each species let us consider there are $n_k \in \mathbb{N}$ observations, the valid observations with the most user weight are selected, and then they randomly select $\min(n_k, n)$ observations, with $n \in \mathbb{N}_*$.

[5]or sooner if improvements are noticed by J-C Lombardo whom I'm most grateful for the details given about the Pl@ntNet training and testing pipeline.

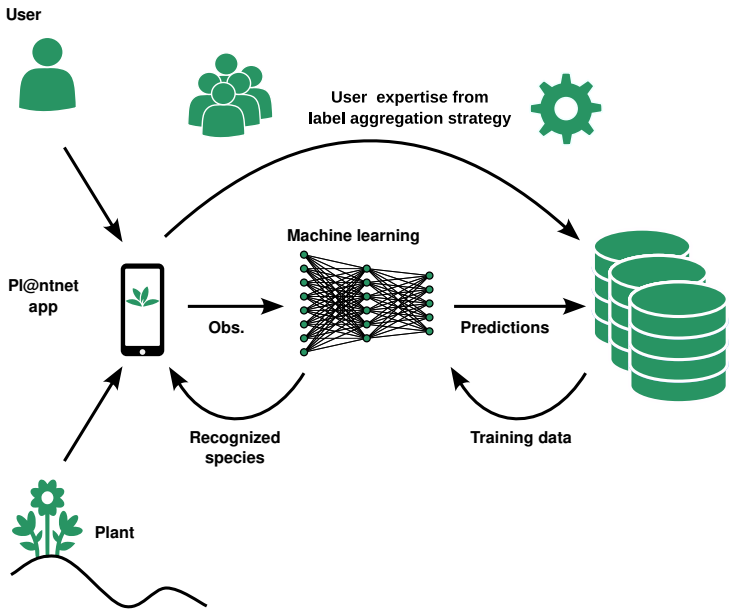


Figure 4.10 Pl@ntNet system of human-AI interaction for plant species recognition. Users take their plant observations in the Pl@ntNet application. A prediction is output by the AI model. Users can validate the prediction or propose another species. The whole votes collection is used to evaluate user expertise and actively revise observations identifications.

On the side, a neural network binary classifier is trained on another dataset to detect if the observation comes from a herbarium or not (binary classification). This classifier is then used to filter out the herbarium observations from the dataset. Herbarium observations are only used for rare species to populate classes with few observations. They are not used if there are *enough* in-the-wild observations as they might.

From the selected observations, the observations constitute a highly imbalanced dataset. This imbalance is necessary as plant species' presence is not balanced in the real world. Additional data is then collected from partners – botanic gardens, GBIF, *etc.* – to get information on organs, species, diseases and other relevant information. The additional data is not used to balance the dataset, only to provide more information and populate some species – but the imbalance is kept. All partner's observations are standardized (resize and compression).

As users can freely upload observations, the observations might not be plant observations (as in Figure 4.8). There are 20 rejection classes, including but not limited to *homosapiens* (presence of a hand, foot or face), *dogs*, *cats*, *fungi* (presence of a fungus), *no plant* (no plant in the image), *drawing* (presence of a drawing), *screenshot* (the image is not a photo taken by the user), *objects* (presence of an object), *genitalia*, *etc.* The rejection classes are used to filter out the observations that are not plant observations but also can provide feedback to the user. For example, if a face is present in the photo taken, the system will warn the user that they might want to retake the photo without

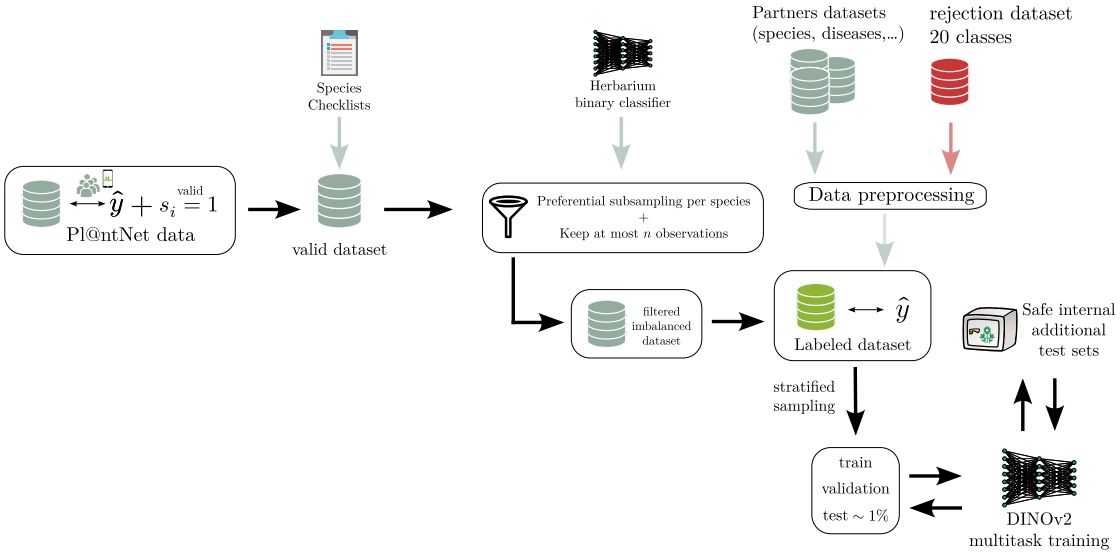


Figure 4.11 Pl@ntNet pipeline: the steps to train a neural network from the collaboratively annotated data. The algorithm presented in Section 4.2 is used to get labels and a validity indicator. Subsampling and data from partners are then combined to get a labeled dataset to train the current DINOv2 model. Evaluation is performed on the generated test set and additional private test sets.

the face. Note that not all rejections provide feedback to the user, some are used to filter out the observations (like the *genitalia* class) and train the Pl@ntNet model to detect images that should not be classified as plants.

The rejection classes are added to the species classes and so is the associated data. The data is then split into a training, validation and test set. The test set is about 1% of the filtered data. The split is stratified by species and organ. Then the model is trained.

Before the latest model version, the training was done in two steps. First, the model was trained on species recognition, then frozen, and then finetuned on organ recognition. Now, the current model is trained directly for multitask classification. In addition to species and organ recognition, diseases (on plants and crops), the genus, the family and some cultivars (sublevel of taxonomy) of interest for other Pl@ntNet projects are also predicted. Denoting $\mathcal{L}_\bullet$ the loss related to the classification task $\bullet$, the multitask loss $\mathcal{L}_{\text{multitask}}$ to minimize is of the form:

$$\mathcal{L}_{\text{multitask}} = \lambda_1 \mathcal{L}_{\text{species}} + \lambda_2 \mathcal{L}_{\text{organ}} + \lambda_3 \mathcal{L}_{\text{genus}} + \lambda_4 \mathcal{L}_{\text{family}} + \lambda_5 \mathcal{L}_{\text{disease}} + \lambda_5 \mathcal{L}_{\text{cultivars}} + \dots,$$

with $(\lambda_p)_p$ the importance weights in the loss, and $\lambda_1 > \lambda_{p'}$ for $p' > 1$ as the species determination is the main task for the model. Let us focus on the species determination as it is the main application of this thesis. To take into account the imbalance in the dataset, the species determination loss is weighted by the inverse of the class frequency. Note that other losses are also considered in recent work from the Pl@ntNet team (Garcin et al., 2022). Transformation on images such as rotation, cropping and flipping are

applied. Label smoothing is used as it is known to improve the uncertainty estimation of the model.

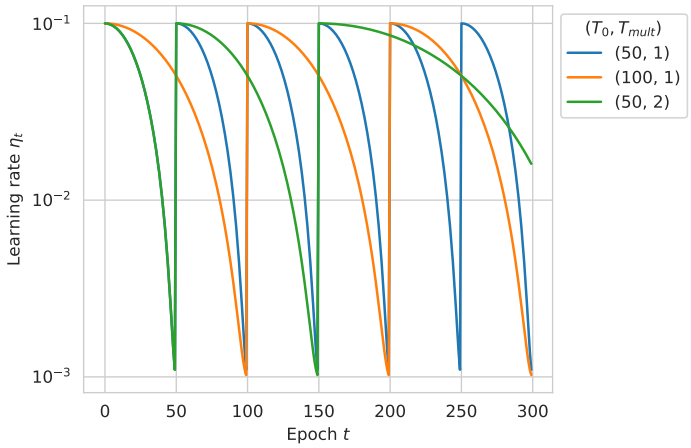


Figure 4.12 Learning rate schedule with a cosine annealing strategy with warm restarts. The learning rate at epoch t depends on the number of epochs in the initial cycle T_0 and the cycle length factor T_{mult}. The initial learning if $\eta_{\max} = 0.1$ and the minimal learning rate is $\eta_{\min} = 10^{-3}$.

Unlike in Chapter 2, the learning rate is scheduled with a cosine annealing strategy with warm restarts (Loshchilov and Hutter, 2016) and not a multistep strategy. Given an initial learning rate $\eta_0 = \eta_{\max} > 0$, a minimal learning rate $\eta_{\min} > 0$, the number of epochs in a cycle T_0 and the current number of epochs in the cycle T_{curr}, the learning rate is updated at each iteration t as:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left\{\frac{T_{curr}}{T_0}\pi\right\}\right) \quad .$$

In practice, the cycle size is increased by a factor of $T_{mult} > 0$ at each restart – see Figure 4.12 for a visualization of the learning rate scheduled with such a scheduler. Thus, denoting $N_{T_{curr}}$ the number of restarts before the current iteration T_{curr}, the scheduler becomes:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left\{\frac{T_{curr}}{T_{mult}^{N_{T_{curr}}} T_0}\pi\right\}\right) \quad .$$

This scheduler is known to help improve convergence and performance. Indeed, the smooth variation helps the optimization process to escape local minima and explore the parameter space more effectively. Moreover, thanks to the periodic restarts, the model is forced to explore different regions of the optimization landscape. This can prevent the model from getting stuck in suboptimal solutions and encourages better exploration of the parameter space.

Model evaluation.

The model is evaluated on multiple test sets. A first test set is inherited from the train-validation-test stratified split. It represents 1% of the data and is used to check the performance on Pl@ntNet and the partner's data. Other test sets are used to evaluate the model on specific tasks and species recognitions. Those safe sets contain unreleased and curated data that are only to be used for the evaluation of this model.

On the model characteristics.

The current Pl@ntNet model – DINOv2 (Oquab et al., 2024) – is a transformer-based network (Dosovitskiy et al., 2020). The former models used were a BEiT (Bao et al., 2021) – also transformer-based – and an InceptionV3 (Szegedy et al., 2015) – a convolutional neural network. Hereafter, we briefly describe the DINOv2 model characteristics.

In essence, the DINOv2 model is part of the DINO family: a family of models that combines self-supervised learning and knowledge distillation. More details about DINOv2 are available in the original paper (Oquab et al., 2024) and the work before (Caron et al., 2021).

On one hand, knowledge distillation is performed by training a student model to mimic a teacher model. Given an input image, both networks predict a feature embedding vector and the KL divergence loss between the two predictions is minimized. On the other hand, the self-supervised aspect allows the training of a network without labels first and then finetuning it on a supervised dataset. To achieve this, contrastive learning is used: the goal is for the model to minimize the distance between similar pairs of data and maximize the distance for dissimilar ones.

So how does DINOv2 combine both concepts? First, note that the contrastive learning step is hidden in the embedding representation of the images (Caron et al., 2020). Then, we can simplify DINOv2 into three main objectives:

- Image-level objective: cropped images are fed to a ViT-H/16 transformer – a vision transformer that divides images into 16 patches (4×4 grid) that is pretrained here on the ImageNet-22K dataset – to produce embeddings. The student network is trained on the embeddings to mimic the teacher network.

- Patch-level objective: The DINO teacher network only sees the image-level embeddings. The student network is trained to predict the teacher's embeddings from both the image and the patches' embeddings. This forces the student to learn helpful parts of the image to predict the teacher's embeddings.

- Additional objectives: several (non-exhaustively) other regularization terms are added to the objective such as KoLeo regularization of ℓ_2 normed embeddings from the same batch to spread them uniformly in the feature space, Sinkhorn-Knopp batch normalization (used but not necessary considering their ablation study) *etc.*

Note that, we consider embeddings, but a final softmax operator is used to consider distributions and compute cross-entropy loss between teacher and student predictions. However, as both networks do not have access to class labels yet, cross-entropy minimization is done to align the teacher and student embeddings. Note that additional tricks are performed to prevent the teacher and student networks from collapsing. There are four main tricks used:

- Untying weights: as the model head is much smaller than the patch-level architecture, the head can overfit the image-level data while the patches are underfitted. Untying the weights of the head and the patch-level architecture helps to prevent this.

- resolution adaptation during training to prevent the loss of small objects, this is especially important in Pl@ntNet as small details on plants can help differentiate species. Note that Pl@ntNet images use a 518×518 resolution.

- EMA (exponential moving average): to stabilize the teacher network and prevent collapsing predictions from the student, the teacher network is an exponentially weighted average of the student. Roughly, the update at epoch t of DINOv2's teacher weights is $\theta_t \leftarrow \lambda\theta_t + (1-\lambda)\theta_s$ with $\lambda > 0$ exponential weight (that follows a cosine schedule from 0.996 and 1 during training), θ_t the teacher weights and θ_s the student's.

- centering and sharpening: both tricks – while intuitively opposing – can help avoid collapsing to uniform or Dirac distributions in the feature space. Centering the teacher embeddings before the softmax operator helps to prevent one feature from dominating the others. Sharpening is done by scaling the teacher's embeddings in the softmax. Intuitively, the teacher has access to the full image and thus should be more confident than the student who mostly sees patches. Thus the teacher distribution should peak for the current (unknown) target class. By combining both, the teacher's distribution is non-trivial. This allows Pl@ntNet's network to keep some uncertainty in the predictions.

Finally, once this self-supervised learning is done, the model is finetuned on the Pl@ntNet dataset with aggregated labels and can identify plant species and more. Note that some patch-level operations were removed in Pl@ntNet in order not to classify an image based on a single patch – *e.g.* an image of a flower next to a genitalia should not be classified as a flower, but a rejection. The next step in this thesis is thus to detail the label aggregation strategy, compare it to existing (and scaling) ones, and propose ways to improve it.

4.2 Pl@ntNet's label aggregation strategy

As we discussed in Section 4.1, plant species identification is a task that requires skills to recognize morphological traits (shapes, measurements, environments and specific characteristics). A large number of users with diverse skills have participated in gathering plant observations and helped improve the training dataset of our computer vision model. Their participation is based on votes that they can cast on others' observations, or by the initial species determination of their observation. The quality of each vote is then processed by the algorithm presented in Section 4.2.1.

At the time of writing, this participatory approach has resulted in the collection of over 20 million observations, belonging to almost 46 000 species, by more than 6 million users worldwide. In total, more than 25 million of images are shared in these observations.

Other citizen science projects such as iNaturalist (Van Horn et al., 2018) or eBird (Sullivan et al., 2009) use a similar approach to collect data, but differ in their label aggregation strategy. The iNaturalist project, with more than 2.5 million users, records the votes at different taxonomic levels. The resulting label is the aggregation of at least two votes on a species-level identification (or coarser or finer taxonomic level). A taxon requires at least two-thirds agreements among identifiers and all users have the same weight in the decision-making. Over time, a taxon can be further refined by the community, debated or revoked. eBird handles taxon quality control by using a checklist in each region for observers. Quality control on the checklist is performed and, combined with user knowledge – number of species and checklist submitted, number of flagged observations, discussions among local experts – the species observation is accepted. The eBird project also showed that monitoring species accumulation from observers can help to sort their skills (Kelling et al., 2015). While they consider the species accumulation by hours spent on each collected observation, we present a strategy that takes into account the entire history of observations of the observer.

4.2.1 Presentation of the algorithm

Pl@ntNet label aggregation strategy relies on estimating the number of correctly identified species for each user. Similar to other strategies, we rely on an EM-based iterative procedure (Dempster et al., 1977) to estimate consecutively the users' skills and each observation's species. The detailed iterative algorithm is provided in Algorithm 5.

The label aggregation strategy generates a trust indicator (s_i) on the observation that can reveal whether an observation is valid or not. Notice that in Algorithm 5 we value 10 times more authored observations than voting on other's observations – if a user proposes a new observation with a label (species name) it is more useful than proposing a label by clicking. Indeed, being on the field leads to more information on the environment and a better determination of the species. Finally, note that an identified species is exclusively identified as an author – part of n_j^{author} in Algorithm 5 – or as click – part of n_j^{vote} – to

Algorithm 5 Pl@ntNet iterative weighted majority vote

Input: Votes as $(j, y_i^{(j)})_{i\in[n_{\texttt{task}}],j\in[n_{\text{user}}]}$ for each observation $x_i \in \mathcal{X}$ and user j answering the voted species $y_i^{(j)}$, accuracy threshold θ_{acc}, confidence threshold θ_{conf}, weight function f, initial weight $\gamma > 0$

Ouptput: Estimated labels $\hat{y}_i$ and validity indicator s_i for each observation i

Initialize $\hat{y}_i = \text{MV}\left(\{y_i^{(j)}\}_j\right)$ for each observation $i \in [n_{\texttt{task}}]$

Initialize user weights as $w_j = \gamma$ for each user $j \in [n_{\text{user}}]$

while not converged **do**

  for each observation $i \in [n_{\texttt{task}}]$ **do**

    Compute label confidence: $\text{conf}_i(\hat{y}_i) = \sum_{j\in\mathcal{A}(x_i)} w_j \mathbb{1}(y_i^{(j)} = \hat{y}_i)$

    Compute label accuracy: $\text{acc}_i(\hat{y}_i) = \text{conf}_i(\hat{y}_i) / \sum_{k\in[K]} \text{conf}_i(k)$

    Compute validity indicator: $s_i = \mathbb{1}(\text{acc}_i(\hat{y}_i) \geq \theta_{\text{acc}} \text{ and } \text{conf}_i(\hat{y}_i) \geq \theta_{\text{conf}})$

  for each user $j \in [n_{\text{user}}]$ **do**

    Compute the number of valid identified species for authoring observations:

$$n_j^{\text{author}} = |\{y_i^{(j)} \in [K] \,|\, y_i^{(j)} = \hat{y}_i, s_i = 1, \text{Author}(i) = j\}|$$

    Compute the number of identified species by voting on other's observations:

$$n_j^{\text{vote}} = |\{y_i^{(j)} \in [K] \,|\, y_i^{(j)} = \hat{y}_i, \text{Author}(i) \neq j\}|$$

    Compute the rounding number of identified species per user:

$$n_j = \text{Round}\left(n_j^{\text{author}} + \frac{1}{10} n_j^{\text{vote}}\right)$$

    Transform number of estimated species per user into trust score: $w_j = f(n_j)$

  Update estimated labels with a weighted majority vote

$$\forall i \in [n_{\texttt{task}}],\ \hat{y}_i = \underset{k\in[K]}{\arg\max} \sum_{u\in\mathcal{A}(x_i)} w_j \mathbb{1}(y_i^{(j)} = k)$$

avoid redundant skills. The final number of species identified by users is the aggregation of these two terms: $n_j = \text{Round}\left(n_j^{\text{author}} + \frac{1}{10} n_j^{\text{vote}}\right)$.

The weight function f shown in Figure 4.13 is a non-decreasing function that maps the number of identified species n_j to a trust score in the form of:

$$w_j = f(n_j) = n_j^{\alpha} - n_j^{\beta} + \gamma \ , \tag{4.1}$$

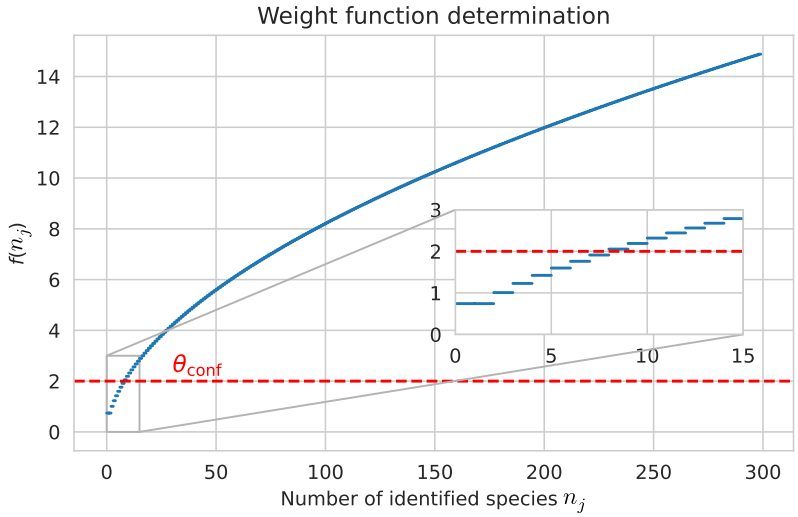


Figure 4.13 Weight function in Equation (4.1) used to map the number of identified species to a trust score in the Pl@ntNet label aggregation strategy. A new user starts with a weight of $f(0) = f(1) = \gamma \simeq 0.74$. The user confidence threshold $\theta_{\text{conf}} = 2$ requires a user to have identified at least $n_j = 8$ species to become **self-validating**. The parameters $\alpha = 0.5$, $\beta = 0.2$ and $\gamma \simeq 0.74$ are used in practice.

where $\alpha, \beta \in \mathbb{R}_+^\star$ are hyperparameters that were calibrated internally to fit prior knowledge and $\gamma > 0$ is the constant representing the initial weight of each user. In practice, we use $\alpha = 0.5$, $\beta = 0.2$ and $\gamma = \log(2.1) \simeq 0.74$ in the weight function. This function is sub-linear ($\mathcal{O}(\sqrt{n_j})$) but with two regimes. The goal of Equation (4.1) is to separate new users from experts and then help sort multiple experts. This is modeled by the two regimes of the weight function. In the first regime which corresponds to new users with low n_j, the term in the power of β decreases the weight. We chose an initial weight $w_j = \gamma$ such that a user has a weight equal to 1 (rounding to two decimals) with two different identifications. This separates the users who only come once to test the application from others. In the second regime with a higher number of identified species, the term to the power of β becomes negligible and we tend to the square root function. The sub-linear scale allows for reducing discrepancies between people who have identified a comparable number of species (and thus have presumably comparable expertise). As for the two thresholds that control the level of uncertainty accepted for a given label, they are set to $\theta_{\text{conf}} = 2$ to control the total weight on an observation and $\theta_{\text{acc}} = 0.7$ to control the agreement between users given their expertise.

Users are said **self-validating** when they are trusted enough so that their proposed label single-handedly makes an observation valid ($s_i = 1$). From Algorithm 5, we see that this is verified when their weight w_j is greater than the level θ_{conf}. Indeed, with a single label we obtain $\text{conf}_i(\hat{y}_i) = w_j > \theta_{\text{conf}}$ and $\text{acc}_i(\hat{y}_i) = 1 > \theta_{\text{acc}}$. In practice, this means

that an experienced user who has collected enough weight can validate any observation without any other user's vote. Note that this identification can later be invalidated by other users with enough weight thanks to the accuracy threshold θ_{conf}.

4.2.2 Introducing a subset of Pl@ntNet to evaluate the current strategy

To compare the different label aggregation strategies on large-scale datasets, we introduce a subset of the Pl@ntNet database focused on Southwestern European flora observations – Baleares, Corsica, France, Portugal, Sardegna and Spain – from 2017 to October 2023. In total, 9 005 108 votes are cast by $n_{\text{user}} = 823\,251$ users on 6 699 593 observations after two cleaning steps on the voted species. The first one is a filtering step. We only keep the votes with plant species belonging to the World Checklist of Vascular Plants (WCVP) (Govaerts, 2023). For the second step, according to Kew's Royal Botanical Garden, we matched synonyms to their backbone species if the species is part of the *k-southwestern-europe* checklist from Plants of the World Online (POWO, 2024) (POWO) system. Note that there are plant species listed in the accepted species from WCVP that are not in the *k-southwestern-europe* POWO checklist. As there is a possible taxon ambiguity in this case – multiple species possible for a given synonym depending on the referential – we leave the proposed label untouched. The dataset is available at `https://zenodo.org/records/10782465`.

In the following, denote K the number of species within the dataset. We will consider multiple subsets of one large released dataset. We index the observations by $i \in [n_\bullet] = \{1, \dots, n_\bullet\}$ where $\mathcal{D}_\bullet$ is the considered dataset composed of $n_\bullet$ observations and their associated votes. For a given subset we do not address the number of observations (previously called tasks) as $n_{\texttt{task}}$ but as $n_\bullet$. For example, the full south-western European flora dataset from Pl@ntNet of $n_{\text{SWE}} = 6\,699\,593$ observations is denoted $\mathcal{D}_{\text{SWE}}$. Other subsets are presented in Figure 4.14. Each user j has a unique identifier used as an index, and we denote $\mathcal{A}(x_i)$ the set of users that have voted on observation x_i. The vote of user j on observation i is denoted $y_i^{(j)} \in [K]$ is a free text field. Estimated labels are denoted $\hat{y}_i \in [K]$ – we only consider hard labels. Each observation i is created by an author j stored in Author(i).

Creation of an evaluation set in a crowdsourcing setting.

To evaluate the performance of a label aggregation strategy, it is necessary to know the ground truth on a subset of the data. However, in the context of crowdsourced data, there is no known truth for the observations. The sheer volume of data makes it impossible to ask botanical experts to create such ground truth for the whole database. Moreover, identifying species from images is much less accurate than identifying them in the field, due to the partial information contained in the image (Experts, 2018).

Instead of asking experts to label a subset of the data, we rather identify botanical experts in the Pl@ntNet user database and consider their determinations as ground truth.

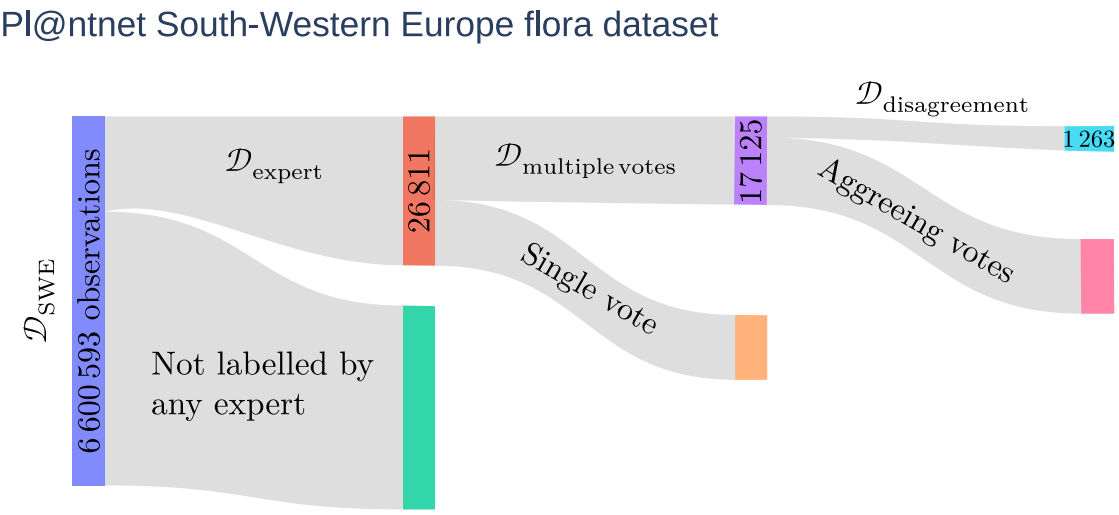
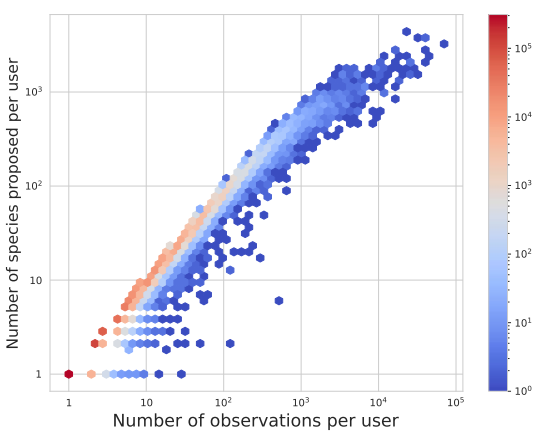


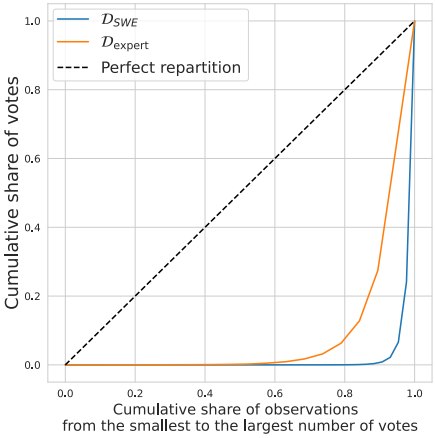
Figure 4.14 Log-scales distribution of the observations in the South-West European Flora subset from the Pl@ntNet database. Note that the (sub-)datasets introduced are nested: $\mathcal{D}_{\text{SWE}} \supset \mathcal{D}_{\text{expert}} \supset \mathcal{D}_{\text{multiple votes}} \supset \mathcal{D}_{\text{disagreement}}$. $\mathcal{D}_{\text{expert}}$ and the following subsets contain observations that received at least one vote from one of the experts.

We asked botanical-known experts to reference other experts who could have a Pl@ntNet account to create a list of expert users. To these we have added TelaBotanica (**?**) users with registered confirmed botanical experience from their account and that are also Pl@ntNet users that participated in the South-Western Europe flora subset. In total, 98 Pl@ntNet users were identified as botanical experts. Observations with at least one vote from one of these experts constitute our test set denoted $\mathcal{D}_{\text{expert}}$. The answers of these experts are considered ground truth labels and used to evaluate strategies' performance. Despite our selection process of supposedly 'indisputable' experts, a few observations in the test set still end up with contradictory labels (4 observations in total). As they represent a very small percentage, we simply removed them from $\mathcal{D}_{\text{expert}}$.

Our evaluation set $\mathcal{D}_{\text{expert}}$ is finally composed of 26 811 observations which received at least one vote from one of the experts. Despite the large number of users, not all observations obtain multiple annotations. Indeed, 310 564 users were single-time voters (meaning they interacted with the system only once). The lack of votes is a large component of difficulty in the Pl@ntNet database, as there is a high imbalance of the distribution of votes between observations as represented in Figure 4.15b. There is a high concentration of votes for a small percentage of the observations as shown in Figure 4.15a. Of these evaluation data, 17 125 received more than two identifications and are stored in $\mathcal{D}_{\text{multiple votes}}$. Then, 1 263 have more than two votes with at least one disagreement between users and are stored in $\mathcal{D}_{\text{disagreement}}$. Figure 4.14 shows the distribution of observations from $\mathcal{D}_{\text{SWE}}$ to the finer and more ambiguous $\mathcal{D}_{\text{disagreement}}$.



(a) Relationship between the number of observations per user and the variety of species proposed per user. Each point represents a concentration of users in the SWE flora subset. 310,564 users proposed a single vote.



(b) Lorenz curves representing the imbalance distribution of the number of votes in the South-West European Flora subset from the Pl@ntNet database. This imbalance is mitigated but kept in the created test set.

Figure 4.15 Pl@ntNet activity summary in the SWE flora subset. (A): The majority of users have proposed a small number of observations and species. However, some users have proposed a large number of observations and species. (B): In a perfectly balanced dataset, the Lorenz curve would be the diagonal – 50% of the votes would be for 50% of the observations. In practice, there is a high imbalance of the distribution of votes between observations – 80% of the observations are represented by 10% of votes. In total, 5.90% of the users are **self-validating**($50\,446 users in SWE flora subset$).

Evaluated aggregation strategies.

Plant species label aggregation is a challenging task due to the large number of species $K = 11\,425$. Hence, many classical strategies in the label aggregation literature such as Dawid and Skene's (Dawid and Skene, 1979) and other variations (Passonneau and Carpenter, 2014; Sinha et al., 2018) are not applicable as they require estimating a $K \times K$ confusion matrix for each user. For the considered dataset $\mathcal{D}_{\text{SWE}}$, this would result in $11\,425^2 \times 823\,251 \approx 10^{14}$ parameters to estimate. Similar issues occur for other label aggregation strategies (Whitehill et al., 2009; Hovy et al., 2013; Ma and Olshevsky, 2020). We do not consider deep-learning-based crowdsourcing strategies as Rodrigues and Pereira (2018); Chu et al. (2021) or Lefort et al. (2022) as they require training a neural network from crowdsourced labels, but do not output aggregated labels on the training set. In the Pl@ntNet application, we need to propose one or multiple species for each observation to users. To overcome these issues, we consider the following label aggregation strategies that can scale with K and the number of users:

- **Majority Vote (MV)**: it selects the most answered label and is the most common

aggregation strategy. More formally, given an observation i:

$$\mathrm{MV}(i, \{y_i^{(j)}\}_j) = \underset{k\in[K]}{\arg\max} \sum_{j\in\mathcal{A}(x_i)} \mathbb{1}(y_i^{(j)} = k) \ .$$

- **Worker Agreement With Aggregate (WAWA)**: this strategy, also known as the inter-rater agreement, weights each user by how much they agree with the MV labels on average. More formally, given an observation i:

$$\mathrm{WAWA}(i, \mathcal{D}_{\mathrm{SWE}}) = \underset{k\in[K]}{\arg\max} \sum_{j\in\mathcal{A}(x_i)} w_j \mathbb{1}(y_i^{(j)} = k)$$
$$\text{with } w_j = \frac{1}{|\{y_{i'}^{(j)}\}_{i'}|} \sum_{i'=1}^{n_{\mathrm{SWE}}} \mathbb{1}\left(y_{i'}^{(j)} = \mathrm{MV}(i', \{y_{i'}^{(j)}\}_j)\right) \ .$$

  As there is no observation filter for the MV and WAWA, we consider that for all observation i, $s_i = 1$ for these two strategies.

- **TwoThird**: The TwoThird aggregation generates a label for observations with at least two votes. The estimated label represents the one with at least two-thirds of the majority in agreement. Every user has the same weight in the aggregation. It is part of the iNaturalist's label aggregation system (Van Horn et al., 2018). More formally:

$$\mathrm{TwoThird}(i, \{y_i^{(j)}\}_j) = \begin{cases} \mathrm{MV}(i, \{y_i^{(j)}\}_j) & \text{if } s_i = 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$
$$\text{with } s_i = \mathbb{1}\left(\max_{k\in[K]} \frac{1}{|\mathcal{A}(x_i)|} \sum_{j\in\mathcal{A}(x_i)} \mathbb{1}(y_i^{(j)} = k) \geq \frac{2}{3}\right) \ .$$

Evaluation metrics.

To evaluate the label aggregation strategies, we use the following label recovery accuracy computed on the evaluation datasets:

$$\mathrm{Acc}(\hat{y}, y; \mathcal{D}_\bullet) = \frac{1}{n_\bullet} \sum_{i=1}^{n_\bullet} \mathbb{1}(\hat{y}_i = y_i)\mathbb{1}(s_i = 1) \ ,$$

with $\hat{y} = (\hat{y}_i)_i$ the estimated labels on $\mathcal{D}_\bullet \in \{\mathcal{D}_{\text{expert}}, \mathcal{D}_{\text{multiple votes}}, \mathcal{D}_{\text{disagreement}}\}$, $y = (y_i)_i$ the associated experts labels, considered as ground truth. When the aggregation strategy indicates the observation as invalid ($s_i = 0$ for Pl@ntNet and TwoThird), this metric considers the sample as wrongly classified. Precision and recall scores are also computed to respectively measure the correctness of the observations indicated as valid and the ability to recover the ground truth observations in the valid set. We take into account the species imbalance by using a macro-average for these metrics. This treats rare species as

equally important to common ones. Denoting respectively TP_k, FP_k and FN_k the true positives, false positives and false negatives related to the species k, the macro averaged precision and recall write

$$\mathrm{Precision}_{\mathrm{macro}} = \frac{1}{K}\sum_{k=1}^{K}\frac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FP}_k} \quad \text{and} \quad \mathrm{Recall}_{\mathrm{macro}} = \frac{1}{K}\sum_{k=1}^{K}\frac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FN}_k} \ .$$

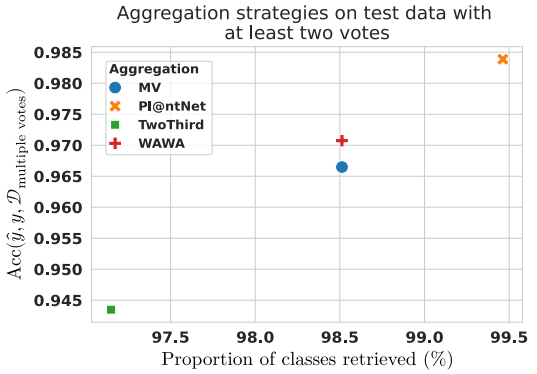
As both the Pl@ntNet and the TwoThird strategies can invalidate some of the observations, we also compute the proportion of observations removed from the whole dataset (whereas previous metrics are computed on the evaluation dataset). This complementary metric allows measuring the proportion of samples "lost" for the training of the AI model after the aggregation step. In practice, filtering data might remove some noisy samples from the dataset. Yet, in general, the more samples are filtered, the fewer ones to train the neural network training. Finally, we also consider the proportion of species retrieved by the aggregation strategies on $\mathcal{D}_{\text{expert}}, \mathcal{D}_{\text{multiple votes}}$ and $\mathcal{D}_{\text{disagreement}}$. This is a critical consideration because if a species identified by experts is absent from the aggregated data, the neural network trained on this data will be unable to make predictions for that very species.

We evaluate the label recovery Acc of each strategy on $\mathcal{D}_{\text{expert}}, \mathcal{D}_{\text{multiple votes}}$ and $\mathcal{D}_{\text{disagreement}}$ (see also Figure 4.14): the test set where experts have provided at least one vote ($\mathcal{D}_{\text{expert}}$), its subset of observations with at least 2 votes and one from an expert ($\mathcal{D}_{\text{multiple votes}}$) and its subset of observations with at least 2 votes, one from an expert, and one disagreement ($\mathcal{D}_{\text{disagreement}}$). The latter is the most challenging as it contains the observations with the most ambiguity. We selected these subsets to investigate the label aggregation strategies' performance depending on the ambiguity level.

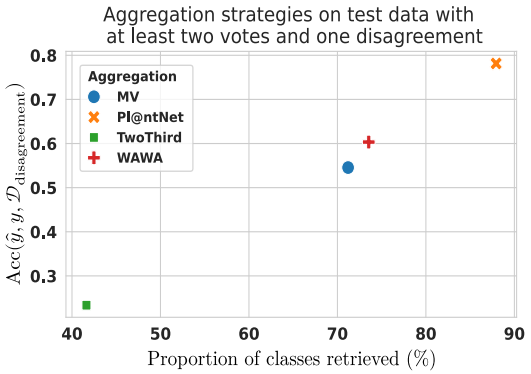
4.2.3 Results on label aggregations

Accuracy of the aggregation strategies.

We begin by evaluating the accuracy of the label aggregation strategies on the set of observations labeled by experts, $\mathcal{D}_{\text{expert}}$. Figure 4.16 shows how many predicted labels match the experts answers on $\mathcal{D}_{\text{multiple votes}}$ and $\mathcal{D}_{\text{disagreement}}$. More importantly, we compare this quantity with the proportion of species retrieved by the aggregation strategy. We observe that the data filtering from the TwoThird strategy – requiring at least two third of agreements – highly degrades performance with respect to other strategies. On $\mathcal{D}_{\text{expert}}$, MV reaches 97% of accuracy, WAWA 98%, TwoThird 60% and Pl@ntNet 99%. To differentiate between the best-performing strategies, we need to look at more ambiguous observations like those in $\mathcal{D}_{\text{multiple votes}}$ and $\mathcal{D}_{\text{disagreement}}$. In highly ambiguous frameworks, the WAWA strategy outperforms the MV one. However, overall the Pl@ntNet aggregation is more often in adequation with the experts and retrieves almost 90% of plant species identified by experts in highly ambiguous datasets against 73% for WAWA, 71% for MV and only 41% for TwoThird.



(a) Accuracy on $\mathcal{D}_{\text{multiple votes}}$ w.r.t. to the proportion of classes recovered



(b) Accuracy on $\mathcal{D}_{\text{disagreement}}$ w.r.t. the proportion of classes recovered

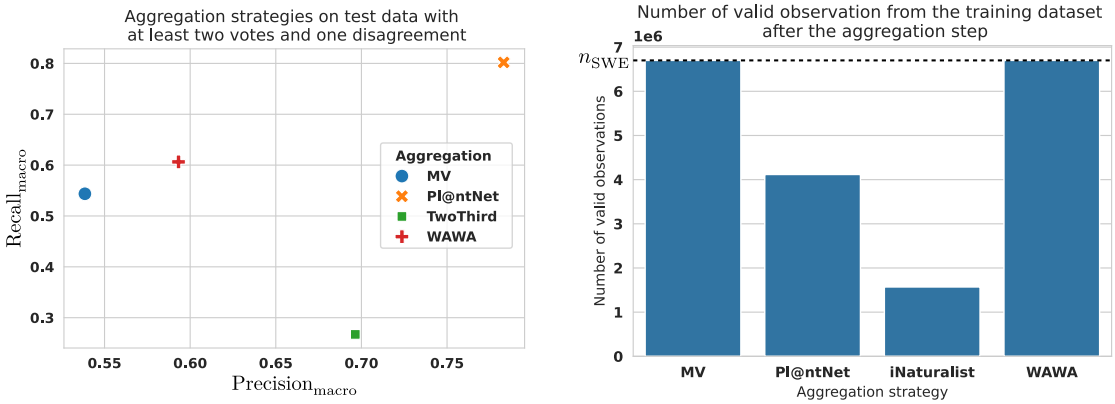
Figure 4.16 Accuracy of the aggregation strategies w.r.t. the proportion of classes (species) retrieved on subsets with at least two votes – either agreeing (A) or with at least one disagreeing vote (B). The Pl@ntNet aggregation is more accurate, especially in a highly ambiguous setting (B). The TwoThird data filter highly impacts how many classes are kept in the dataset and the overall accuracy in both settings. WAWA and MV perform similarly with a benefit for WAWA when skill evaluation is needed.

Precision and recall.

To better evaluate each aggregation strategy, we compute the macro precision and recall metrics for each species. Results are shown in Figure 4.17a. The observations filter ($s_i = 0$) for the TwoThird strategy highly impacts its ability to identify most of the positive observations for a given species. While this agreement threshold filter is created to keep as few noisy samples as possible in research-graded (data quality indicator for research database usage in TwoThird) observations, TwoThird obtains better precision than MV and WAWA but Pl@ntNet's precision shows significant improvement. WAWA strategy outperforms a naive MV aggregation showing that, indeed, weighing users can lead to better performance. Pl@ntNet strategy outperforms all others by several orders of magnitude. Weighing users based on their number of identified species is both interpretable and effective. The observation filter does not negatively impact the recall.

Volume of valid data.

The community labels are aggregated to generate training data for the AI model. The more data the better, however, we need to filter out observations with low visual quality or potentially mislabeled. This is the reason for the validity indicator s_i in the TwoThird and Pl@ntNet strategies. On $\mathcal{D}_{\text{SWE}}$, Figure 4.17b shows how much data is kept for later training. MV and WAWA keep all proposed observation for training – including potential noisy ones. TwoThird filters out most observations to keep nearly 1.5 million



(a) Precision and recall of label aggregation strategies on $\mathcal{D}_{\text{disagreement}}$.

(b) Number of observations in $\mathcal{D}_{\text{SWE}}$ indicated as valid for training ($s_i = 1$).

Figure 4.17 (A): TwoThird strategy has better precision than MV and WAWA strategies, with lower recall because of the heavy filter on the validity of observations. Pl@ntNet aggregation strategy obtains the best precision and recall and outperforms other strategies. (B): TwoThird performance drop can be explained in part by the high proportion of data considered invalid. Note that MV and WAWA strategies do not invalidate any observation, hence keeping potentially mislabeled or low-quality observations. Pl@ntNet achieves a balance between filtering out observations and achieving high performance.

(representing 23.43% of the total observations). Pl@ntNet finds an improved balance between filtering invalid observations and keeping enough data for training.

Figure 4.18 Examples of invalid ($s_i = 0$) and valid ($s_i = 1$) observations using the Pl@ntNet strategy described in Algorithm 5.

Qualitative results on Pl@ntNet observation filter

In this section, we show some examples of observations invalidated by the Pl@ntNet strategy (see Figure 4.18). Invalid observations often come from the lack of user participation with other's observations. Causes of disagreements from users can occur from a multitude of factors – blurriness, multiple species in the same observation, the distance from the plant does not allow precise identification, *etc.* Valid observations, as shown in the second row of Figure 4.18 are zoomed in on the plant's flower, leaf or organ to help the identification process.

4.3 On the choice of the weight function

The current weight determination function in Pl@ntNet is based on the number of species identified by the user:

$$w_j = f(n_j) = n_j^{0.5} - n_j^{0.2} + \gamma \ . \tag{4.2}$$

The confidence threshold is set such that with $n_j = 8$ a user becomes **self-validating**. One can argue that a simplification could be made of such a function to simplify both the computation and the interpretability. We propose to explore simple monotonous functions of the number of identified species. And for each weight function f, we adapt the θ_{conf} such that $\theta_{\text{conf}} \simeq f(8)$.

The simplification constraints are:

- *interpretability*: we need to understand easily the link between the weight and the number of identified species
- *closeness to the current weights*: the new weight functions should not modify the current general behavior in order not to impact the platform
- *performance*: the new weight functions should not degrade the performance of the label aggregation strategy
- *collaborative correction*: the new weight function should allow an expert to correct erroneous answers from multiple low-weight users. Similarly, multiple low-weight users' votes should be able to correct votes from a non-expert user.

Table 4.1 Weight functions tested, the associated confidence threshold θ_{conf} and the accuracy performance on $\mathcal{D}_{\text{disagreement}}$.

Function $f(n_j)$	$n_j^{0.5} - n_j^{0.2} + \gamma$ (current)	$n_j + \gamma$	$n_j^{0.5} + \gamma$	$n_j^{0.75} + \gamma$	$n_j^2 + \gamma$	$\log(n_j) + \gamma$
θ_{conf}	2	8.75	3.6	5.5	64.75	2.82
$\text{Acc}(\hat{y}, y, \mathcal{D}_{\text{disagreement}})$	0.781	0.779	0.782	0.779	0.777	0.721

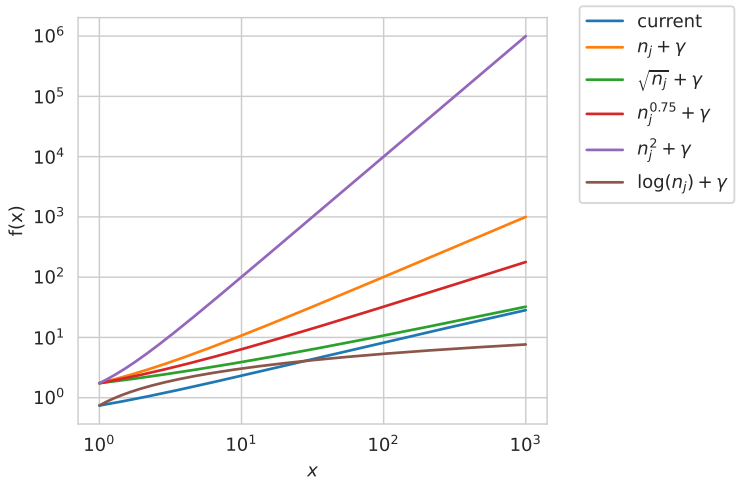


Figure 4.19 Visualisation of the different functions tested for the weight determination.

The first conclusion when running the label aggregation with the weight functions from Table 4.1 is that except for the log transformation, all weight functions lead to similar performance. For example, the label recovery accuracy Acc on $\mathcal{D}_{\text{disagreement}}$ is given in Table 4.1.

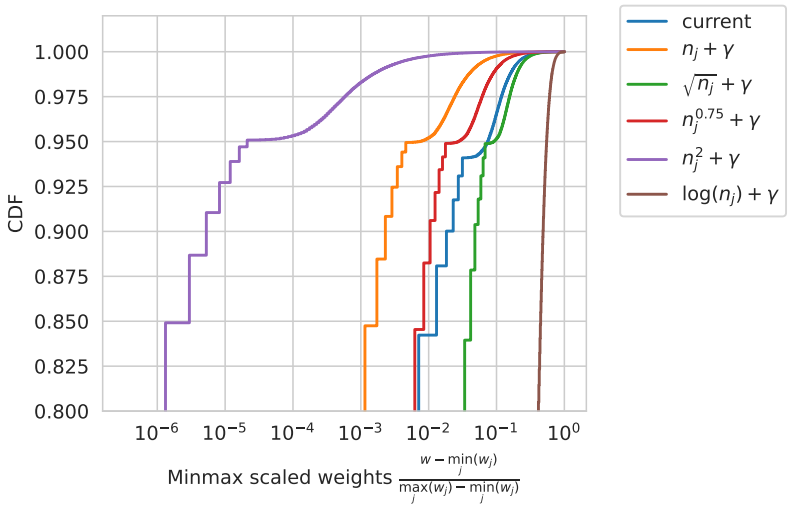


Figure 4.20 Zoomed empirical CDF of the weights obtained after running Pl@ntNet's label aggregation strategy with the weight functions tested in Table 4.1. The log transformation result is represented for the sake of completeness.

As performance is similar between the different weight functions other than the logarithmic one – similar conclusions are observed for the other evaluation subsets – let

us look at how the weight functions spread users' weights. To do so, we plot the empirical cumulative distributive function (CDF) of the weights per function tested. As the scales vary between the obtained weights, we use a minmax scaling to compare the CDFs with weights in $[0, 1]$. More formally, for each vector of weight obtained, we compute:

$$\mathrm{CDF}_w(t) = \frac{1}{n_{\texttt{user}}} \sum_{j=1}^{n_{\texttt{user}}} \mathbb{1}\left(\tilde{w}_j \leq t\right), \ \forall t \in [0,1]$$
$$\text{with, } \tilde{w} = \frac{w - \min_j(w_j)}{\max_j(w_j) - \min_j(w_j)} \ .$$

From Figure 4.20, we observe the following:

- The high concentration of low-weighted users creates the visible steps, then in the second phase there are a lot of users spread with weights close to each other leading to the smooth portion of the CDF.

- To mimic the current weight function, the most appropriate functions are $f(n_j) = n_j^{0.75} + \gamma$ for low weighted users and $f(n_j) = n_j^{0.5} + \gamma$ for the high weighted users. The asymptotic behavior is expected: $n_j^{0.5} - n_j^{0.2} + \gamma \underset{n_j \to +\infty}{\sim} n_j^{0.5}$.

- The square transformation spreads users the most, however, it might not be appropriate in practice as the superlinear behavior means that it is easy to become a high-weighted user and difficult for users to correct mistakes.

- On a side note, we can observe a reason why the log transformation is not suited. Users are close to one another – the weights are not spread – meaning that an expert with a high number of identified species is easily contradicted by a few users with few identified species.

In conclusion, the current weight function could be simplified as $f(n_j) = n_j^{0.5} + \gamma$ or $f(n_j) = n_j^{0.75} + \gamma$. Both yield similar results and spread of user weights. To keep the asymptotic behavior, the square root transformation is the most suited. Test deployments on the full database could help in practice differentiate between the two candidates.

An example of why log and square are not suited for collaborative correction?

Let us ignore the γ term for simplicity in the following examples.

- **Log transformation**: expert knowledge is not represented correctly as low-weight users can easily overweight an expert.

  Consider an expert with $n_{\text{expert}} = 80$ identified species and 3 users u_1, u_2 and u_3 with $n_1 = 3$, $n_2 = 5$ and $n_3 = 6$ identified species. Then:

$$\begin{aligned} w_{\text{expert}} &= \log(80) = \log(8) + \log(10) \simeq 4.38 \\ &< 4.50 \simeq \log(3) + \log(5) + \log(6) \\ &< w_{u_1} + w_{u_2} + w_{u_3}. \end{aligned}$$

- **Square transformation**: An expert aided by multiple self-validating users might not correct another expert with similar knowledge.

  Take two experts e_1 and e_2 such that $n_{e_1} = 102$ and $n_{e_2} = 100$. Those experts should have close weights are their number of identified species is close. If expert e_2 wants to correct e_1, they can not do so with 5 other users $u_1, \ldots, u_5$ that are self-validating (*i.e.* with at least 8 identified species):

$$w_{e_1} = 102^2 = 10404 > 10320 = 100^2 + 5 \times 8^2 = w_{e_2} + \sum_{j=1}^{5} w_{u_j}.$$

Hence, the square and log transformation should not be considered for such a task.

4.3.1 On the choice of the weight in the weight function

Currently, the weight of user j, denoted by $w_j > 0$ is computed from the estimated number of identified species n_j. This choice does not penalize users who either/both:

- might have correctly identified a species a one point and now have lost this ability,
- are **self-validating** – more than 8 identified species – and produce new wrong identifications.

This is a common issue in crowdsourcing systems where users can have a high reputation but still make mistakes. To mitigate this issue, classically in crowdsourcing, the votes of other users are used to correct the mistakes and a time-dependency can be introduced to the weights.

Penalize the mistakes

As we do not have a skill evolution metric to create a time dependency in the votes, let us consider ways to penalize the mistakes of users. We compare the following strategies:

- **Vanilla**: the number of identified species is not penalized.
- **Difference**: the number of identified species is penalized by the number of wrongly identified species. The weight writes:

$$w_j = f(\tilde{n}_j) = f(\min(n_j - n_{j,\text{invalid}}, 0))$$

  with $n_{j,\text{invalid}} = \text{Round}\left(n^{\text{author}}_{j,\text{invalid}} + \frac{1}{10} n^{\text{vote}}_{j,\text{invalid}}\right)$ the number of wrongly identified species by user j either by vote or new observation. The quantity $n^{\text{author}}_{j,\text{invalid}} = |\{y_i^{(j)} \in [K] \,|\, y_i^{(j)} \neq \hat{y}_i, s_1 = 1, \text{Author}(i) = j\}$ represents wrong identifications on authored valid observations. The same applies for $n^{\text{vote}}_{j,\text{invalid}}$ for votes on all observations. Conceptually, one's weight is great if they have more correct identifications than wrong ones.
- **Ratio**: the number of identified species is penalized by the number of different proposed species. The weight writes:

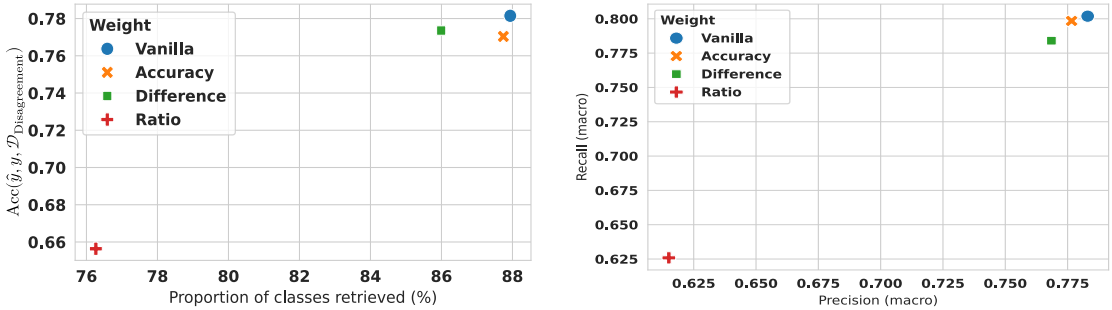
$$w_j = f(\tilde{n}_j) = f\left(\frac{n_j}{n_{\text{proposed}}} n_j\right),$$

  with $n_{\text{proposed}} = |\{y_i^{(j)} \in [K]\}|$ the number of different proposed species by user j. Morally, the ratio $\frac{n_j}{n_{\text{proposed}}} \in [0, 1]$ represents the proportion of identified species among the proposed ones. If a user proposes a lot of different species, they should be correct otherwise this diversity could be seen as spam votes.
- **Accuracy**: similar to the WAWA strategy, we also consider weighting n_j by the accuracy of worker j:

$$w_j = f(\tilde{n}_j) = f\left(\left[\frac{1}{|\{y_{i'}^{(j)}\}_{i'}|} \sum_{i'=1}^{n_{\text{SWE}}} \mathbb{1}\left(y_{i'}^{(j)} = \text{MV}(i', \{y_{i'}^{(j)}\}_j)\right)\right] n_j\right).$$

  Morally, a user-identified species can not be trusted if other users need to correct them too often.

In Figure 4.21, we observe that the new weight functions do not outperform the current system. The **Accuracy** reweighting is the closest to the **Vanilla** strategy. Hereafter, we discuss the reasons why the new weight functions do not outperform the current system and possible perspectives.



(a) Accuracy on $\mathcal{D}_{\text{disagreement}}$ w.r.t. to the proportion of classes retrieved

(b) Macro recall w.r.t the macro precision on $\mathcal{D}_{\text{disagreement}}$.

Figure 4.21 Accuracy, proportion of classes retrieved, precision and recall on $\mathcal{D}_{\text{disagreement}}$ depending on the new computation of the weights. The **Accuracy** reweighting is the closest to the **Vanilla** performance. Note that, no newly proposed method outperforms the **Vanilla** strategy.

Discussion.

First, consider that from a user perspective modifying the weights comes at a high cost: one might feel that their contributions are not valued if we penalize mistakes and participate less. But the goal of Pl@ntNet is also to get quality labels to output reliable predictions.

Several issues can be considered and possibly mitigated in the previously proposed methodology to modify the user's weight:

- **Data sparsity.** A user that is **self-validating** can validate their observation. If they propose a new species, their identification becomes valid and increases their weight even though there is no feedback on their knowledge of these new species. And if a **self-validating** user proposes multiple identification incorrect, only other votes can invalidate them. This is both a strength and weakness of crowdsourcing: we need to trust the users to some extent and interactions between users on the same observation to estimate their reliability. Observations with a single label are more likely to be mislabeled. In the SWE subset, 24.08% of the observations have only one label (1 613 354 observations). These votes are not reviewed by the community and can only be trusted to some extent. And 62.28% of the users only voted once (512 687 users). These users have a weight close to zero and their observations are not valid in a vast majority.

- **Corrections and penalizations.** First, mistakes from users that are not **self-validating** are less impactful in the system as they are easily fixed. Until someone agrees with the vote there is not enough weight to validate the observation (thanks to the θ_{conf} threshold). However, for **self-validating** users, the mistakes are more impactful as they can validate their observations and gain weight more easily. If

we penalize their participation by weights like WAWA's based on their accuracy we can think of at least three possibilities:

- *Classical accuracy*: but as most of their observation has only their vote (and are not corrected) they have a small penalty that does not influence performance (see Figure 4.21). And also they are not incentivized to vote on others' observations as they might decrease in weight with these interactions.
- *Accuracy on reviewed valid observations*: we could consider the accuracy of the user on observations currently valid ($s_i = 1$) and with at least two votes. Given a user j, denote their set of observations with at least two votes $\mathcal{D}^{(j)}_{\text{reviewed,valid}} = \{i\,;\, |\{y_i^{(j)}\}_i| \geq 2 \text{ and } s_i = 1\}$.
  The main issue is that if an observation is corrected with a disagreeing vote, for it to be valid the θ_{acc} threshold needs to be reached. This is hard to meet. For example, consider a vote by a user with $n_1 = 30$ identified species. If another user answers a contradicting vote, they should have at least 141 identified species to validate the observation with the θ_{acc} threshold. For two contradicting users, we can plot the minimal number of identified species for the users to validate the observation with the θ_{acc} threshold in Figure 4.22. Each user must have at least $n_2 = n_3 = 40$ identified species to validate their correction if $n_1 = 30$, or they can compensate each other knowledge (see Figure 4.22). This requirement is hardly met in practice, leading to a similar accuracy for users.

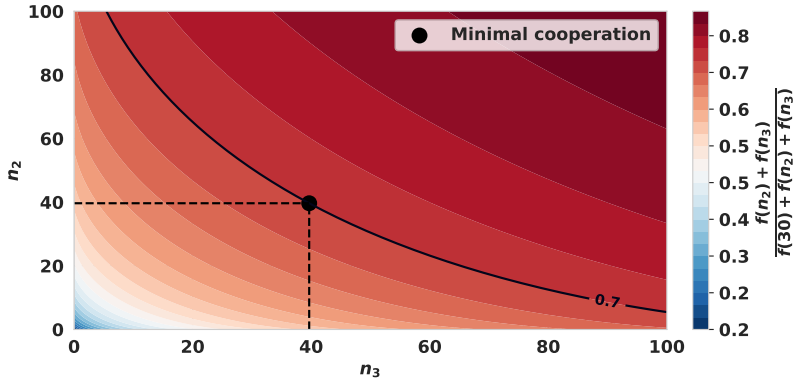


Figure 4.22 Minimal number of identified species n_2 and n_3 for two users to validate an observation with a common disagreeing vote if the author has $n_1 = 30$ identified species. The θ_{acc} threshold is $\frac{f(n_2)+f(n_3)}{f(n_1)+f(n_2)+f(n_3)}$ and must be greater than $\theta_{\text{acc}} = 0.7$.

In a more general setting, this can be seen as an optimization problem with constraints:

$$\underset{n\in\mathbb{R}_+^{d+1}}{\arg\min} \|n\|_2^2 \quad \text{s.t.} \quad \frac{\sum_{j\geq 1} n_j}{\sum_{j\geq 0} n_j} - \theta_{\text{acc}} \geq 0 \ ,$$

where $n \in \mathbb{R}_+^{d+1}$ is the vector of the number of identified species for $d+1$ users: n_0 for the author of the vote to correct and $(n_j)_{j\geq 1}$ for the d others trying to correct it. This problem can be solved numerically using `scipy.optimize` library for example for any dimension.

– *Accuracy on reviewed observations*: we can relax the latter condition and consider all observations and no condition on the validity. Given a user j, denote their set of observations with at least two votes $\mathcal{D}^{(j)}_{\text{reviewed}} = \{i\,;\, |\{y_i^{(j)}\}_i| \geq 2\}$. The reviewed accuracy of user j writes:

$$\text{Acc}_{\text{reviewed}}(\{y_i^{(j)}\}_j, y; \mathcal{D}_{\text{SWE}}) = \frac{1}{|\mathcal{D}^{(j)}_{\text{reviewed}}|} \sum_{i \in \mathcal{D}^{(j)}_{\text{reviewed}}} \mathbb{1}(y_i^{(j)} = y_i) \ .$$

However, in practice, this still does not outperform the Vanilla weight. This can be explained as there are few observations with multiple votes and a disagreement in $\mathcal{D}_{\text{SWE}}$ (67 962 observations, representing 1.01% of the dataset). Fewer of these observations have a ground truth label ($\mathcal{D}_{\text{disagreement}}$) so we might not see the impact of such weight even on a such large subset dataset of Pl@ntNet.

To conclude this discussion on the penalization of the mistakes, we can say that the current weight function is a good compromise between the number of identified species and the quality of the identifications. Penalizing the mistakes is not straightforward and the current system is already quite efficient. The main issue is the lack of interactions between users on the same observation. With more interactions in the future, we could reconsider these penalizations to improve the quality of the data – and the code will be able to do so.

So, if the issue is the lack of votes, we need to incentivize users to vote on others' observations. This could be done using a recommendation system for users. This recommendation system should take into account both *some sense of skills* (the currently identified species, or at least their family), the *current weight* (a vote from a new user is not very useful against an expert), and an *exploration* part to help users discover new species. In the meantime, the lack of votes can be mitigated by the current Pl@ntNet computer vision predictions'.

4.4 Integrating model predictions in the aggregation

While we restricted ourselves to the SWE subset, Pl@ntNet's data is collected internationally. The more correctly identified observations are added to the training set, the better the prediction of the trained model for end-users. This classifier is trained from valid observations and aggregated labels (see Figure 4.10 and Section 4.1.4). At the time of writing, the model in use in Pl@ntNet is DINOv2 (Oquab et al., 2024) a transformer-based network. However, note that some observations from $\mathcal{D}_{\text{SWE}}$ have been

processed by an earlier version of Pl@ntNet's AI: either an InceptionV3 (Szegedy et al., 2015) or a BEIT (Bao et al., 2021) classifier. We can use the classifiers to generate votes. For an observation i, the AI vote is denoted $y_i^{\text{AI}} \in [K]$. The probability output in the classifier's predicted species is denoted $\mathbb{P}(y_i^{\text{AI}})$.

4.4.1 Possible strategies

If we consider the trained model as any other user, denoted as **AI as user**, the previously presented label aggregations are available. However, with the Pl@ntNet aggregation algorithm, the AI weight increases drastically and overpowers human users. This would mean the next Pl@ntNet model is mostly trained on the predictions of the previous one. This defeats the purpose of a cooperative active learning system and the human-AI interaction. It would result in a dangerous feedback loop, and possible mode collapse. Thus, we explore alternative ways of integrating the AI votes in the aggregation algorithm:

- **AI as user**: This is the naive approach we just described. The AI is considered as any other user in the database. The total number of users is thus raised to $n_{\text{user}} + 1$.
- **Fixed weight AI**: Give a fixed weight $w_{\text{AI}} = 1.7 > 0$ to the AI. The weight is below the threshold θ_{conf} so that it can not self-validate its predictions. The confidence writes

$$\text{conf}_i(\hat{y}_i) = \sum_{j \in \mathcal{A}(x_i)} w_j \mathbb{1}(y_i^{(j)} = \hat{y}_i) + w_{\text{AI}} \mathbb{1}(y_i^{\text{AI}} = k) \ . \tag{4.3}$$

  The final estimated label becomes

$$\hat{y}_i = \underset{k \in [K]}{\arg\max} \sum_{j \in \mathcal{A}(x_i)} w_j \mathbb{1}(y_i^{(j)} = k) + w_{\text{AI}} \mathbb{1}(y_i^{\text{AI}} = k) \ . \tag{4.4}$$

- **Invalidating AI**: The AI is considered as a user with a fixed weight and can only participate in invalidating identifications *i.e.* have $s_i = 0$. This translates as the confidence updated as in Equation (4.3) but the final Weighted MV remains unchanged from Algorithm 5.
- **Confident AI**: The AI is considered a user with a fixed weight and can only participate if the confidence in its prediction $\mathbb{P}(y_i^{\text{AI}})$ is over a threshold $\theta_{\text{score}} \in [0, 1]$. The confidence writes

$$\text{conf}_i(\hat{y}_i) = \sum_{j \in \mathcal{A}(x_i)} w_j \mathbb{1}(y_i^{(j)} = \hat{y}_i) + w_{\text{AI}} \mathbb{1}(y_i^{(j)} = \hat{y}_i, \, \mathbb{P}(y_i^{\text{AI}}) \geq \theta_{\text{score}}) \ . \tag{4.5}$$

  The final estimated label becomes

$$\hat{y}_i = \underset{k \in [K]}{\arg\max} \sum_{j \in \mathcal{A}(x_i)} w_j \mathbb{1}(y_i^{(j)} = k) + w_{\text{AI}} \mathbb{1}(y_i^{\text{AI}} = k, \, \mathbb{P}(y_i^{\text{AI}}) \geq \theta_{\text{score}}) \ . \tag{4.6}$$

4.4.2 On the choice of the AI weight.

The AI has a fixed weight $w_{\text{AI}} > 0$ for the **Fixed weight AI**, the **Invalidating AI** and the **Confident AI** strategies. The choice of this weight must meet several constraints. First, we would like to avoid the AI votes to be **self-validating** as it would validate all the AI predictions on a large part of the database, thus we must have $w_{\text{AI}} < \theta_{\text{conf}}$ in Algorithm 5. We also want the AI votes to help clean the database by invalidating some observations from low-weight users (with weight $0< w_{\text{low}} \leq \theta_{\text{conf}}$). Thus $w_{\text{low}}/(w_{\text{low}} + w_{\text{AI}}) < \theta_{\text{acc}}$. Hence, our constraints read:

$$\begin{cases} w_{\text{AI}} & < \theta_{\text{conf}} \\ \frac{w_{\text{low}}}{w_{\text{low}}+w_{\text{AI}}} & < \theta_{\text{acc}} \end{cases} . \tag{4.7}$$

Taking the extreme case where a user becomes self-validating: $w_{\text{low}} = \theta_{\text{conf}}$, we obtain that $w_{\text{AI}} > \theta_{\text{conf}} \left(\frac{1-\theta_{\text{acc}}}{\theta_{\text{acc}}}\right)$. And using the first condition in Equation (4.7), we obtain the bounds

$$\theta_{\text{conf}} \left(\frac{1-\theta_{\text{acc}}}{\theta_{\text{acc}}}\right) < w_{\text{AI}} < \theta_{\text{conf}} \; (\Longleftrightarrow 0.85 < w_{\text{AI}} < 2) \quad . \tag{4.8}$$

As more than a million observations from our dataset only have two votes, one way to choose the AI weight is to consider that the AI can invalidate two erroneous non-experts that would both have just enough weights to make the observation valid: $1.95 = w_{\text{low}} < 2$. Then, the AI weight should be greater than their cumulated confidence: $w_{\text{AI}} > 2w_{\text{low}} \left(\frac{1-\theta_{\text{acc}}}{\theta_{\text{acc}}}\right)$. We finally take the upper rounded value $w_{\text{AI}} = 1.70$ (which satisfies Equation (4.8)).

4.4.3 Results on integrating the AI in the aggregation

The current trained neural network model in Pl@ntNet's system can make predictions based on its training on the Pl@ntNet database (across different floras). We compare the four following strategies – **AI as user**, **fixed weight AI**, **invalidating AI** and **confident AI**, presented in Section 4.4.1 to integrate the AI vote into the Pl@ntNet label aggregation strategy. For the confident AI strategy, we evaluate multiple thresholds θ_{score}. Note that if $\theta_{\text{score}} = 0$ the AI votes for all observations and if $\theta_{\text{score}} = 1$ the AI does not vote and we recover the performance of the current Pl@ntNet aggregation strategy presented in Algorithm 5. We see in Figure 4.23 that the confident AI strategy with $\theta_{\text{score}} = 0.7$ seems to perform best and keep the most data in both $\mathcal{D}_{\text{SWE}}$ and $\mathcal{D}_{\text{expert}}$.

4.4.4 Can we trust our current predicted probabilities?

As for the inclusion of the AI vote, some concerns should be raised. First, as the AI model is trained from the aggregated labels and observations, integrating its vote should not make the AI predictions run out of control. If we consider the AI as a user, as we are in iterative training, the system fails to learn from the human labels. However, using

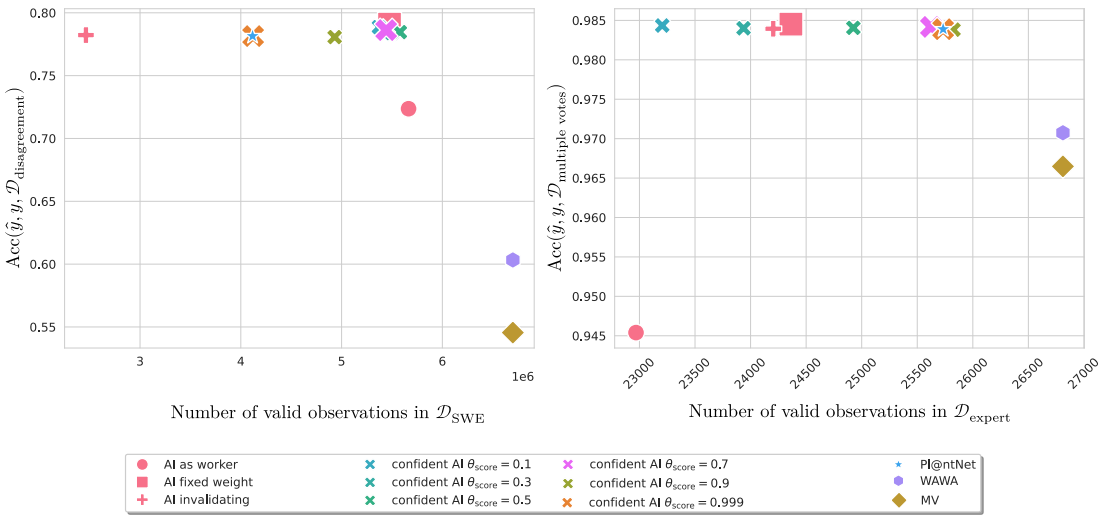
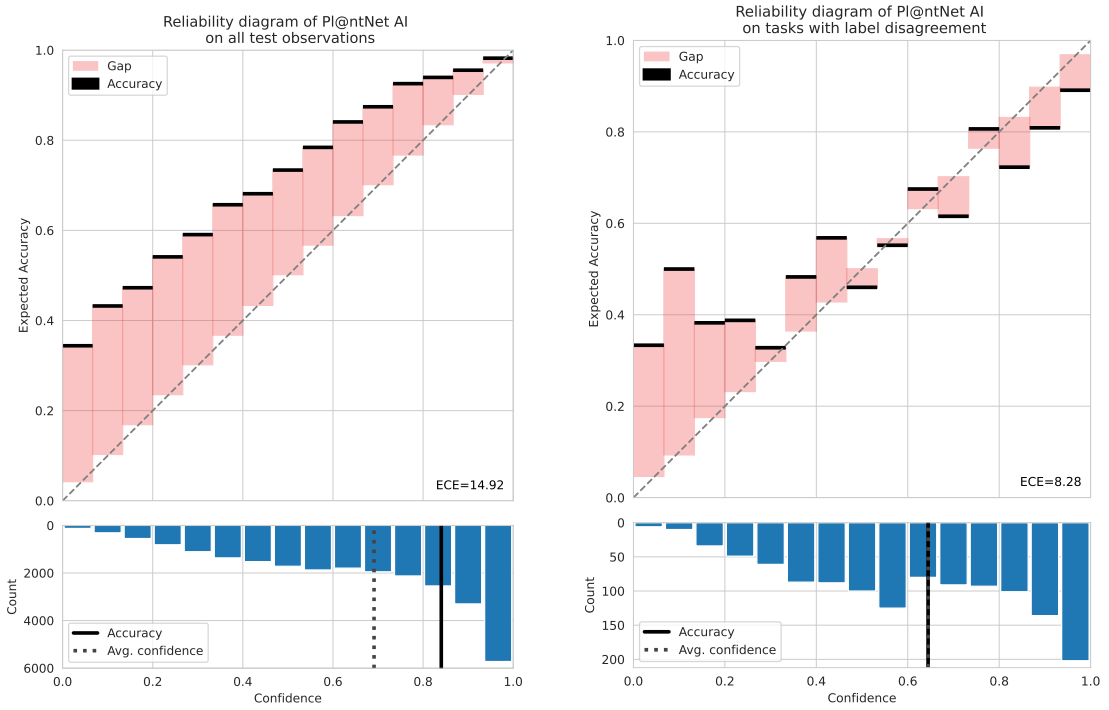


Figure 4.23 Performance in label recovery and number of observations marked as valid depending on how the AI vote is integrated. MV, WAWA and Pl@ntNet strategy without AI vote are used as reference. The best-performing strategy overall is **confident AI** with $\theta_{\text{score}} = 0.7$. We also see that when θ_{score} tends to 1, we recover the vanilla Pl@ntNet aggregation strategy.

the AI vote to invalidate the data with a fixed weight can help clean the database, and with enough weight other users can switch its validity back. However, this would not help in switching the wrong label. To do so, we investigate in Section 4.4.1 to only consider a fixed weight label with enough confidence from the AI model. We observe that this strategy leads to better performance. As we use the output probabilities we should discuss the calibration of our network too.

Calibration is the measure of how close the output confidence is to the true probability (Niculescu-Mizil and Caruana, 2005). Currently, the Pl@ntNet AI is not calibrated using post-processing methods (Platt, 1999; Guo et al., 2017). We discuss hereafter the calibration of the current AI model and possible guidelines for further integration of AI votes.

From Figure 4.24a, we see that currently, Pl@ntNet AI is underconfident. Meaning that it consistently underestimates its confidence and outputs to users more uncertainty than it should. One factor that can influence the results is that the calibration is computed on the test set where experts either authored or voted on observations. Botanical experts have more experience with taking pictures of plants and better equipment than the average citizen. Thus, the observation quality – and subsequently the probability distribution output by the AI – can be biased. Another factor known for leading to such suboptimal predictions is the data augmentation (Kapoor et al., 2022). As the model trains on multiple versions of each original sample with multiple distortions, these variations can become unrepresentative of the underlying sample distribution and cause unnecessary prediction difficulties. The data augmentation is used to mitigate the species imbalance



(a) Reliability diagram of Pl@ntNet AI on $\mathcal{D}_{\text{expert}}$

(b) Reliability diagram of Pl@ntNet AI on $\mathcal{D}_{\text{disagreement}}$

Figure 4.24 Reliability diagrams for the Pl@ntNet AI on the expert dataset and on the more ambiguous $\mathcal{D}_{\text{disagreement}}$ subset. The AI is overall underconfident (A). However, on more ambiguous observations it is overconfident for observations leading to high predicted probabilities (B).

of the database.

However, this imbalance is also known to lead to miscalibrations in predictions (Ao et al., 2023). On Figure 4.24b, we see that for ambiguous observations (where users disagree), the AI is overconfident in its highest predictions – which represents half of the dataset – and underconfident in the other half. These different calibration behaviors inform us that, if a given strategy should incorporate the AI votes in the label aggregation based on the output probabilities, we need to be able to rely on such probabilities. Therefore, even if the confident AI strategy leads to the best performance in Section 4.4.1, it should not be used directly without recalibration of the model – using for example temperature scaling (Guo et al., 2017). In future work, more study is needed to investigate the confidence gap of the model and the observations' ambiguity from users' labels. The current large-scale and interpretable aggregation strategy from Pl@ntNet already outperforms others without the AI votes.

4.5 Conclusion

We demonstrated that collaborative identification of plant species can effectively be used to obtain expert-level labels. Releasing a large subset of millions of observations and thousands of users from the Pl@ntNet organization, we investigate a label aggregation strategy that weighs user answers based on their estimated number of species correctly identified without using prior expert knowledge. Many strategies used previously either do not scale to the magnitude of the current databases – either Pl@ntNet, iNaturalist or eBird – or are outperformed by our aggregation.

Our strategy weighs users based on the number of correctly identified species. This weight is interpretable and shows the diversity of the user's skill set. It can be directly applied to other crowdsourced frameworks with a high number of classes like TwoThird or eBird. The values for both hyperparameters θ_{conf} and θ_{acc} which respectively handle the cumulated weight on observation and the agreement level for the given label can be applied as is.

Note that Pl@ntNet's label control system heavily rests on visual analysis of the observations and inter-user agreements. Additional metadata such as geolocation, date, phenological stage or visual description can be registered in Pl@ntNet and help identify the plant's species but are currently not directly taken into account for user evaluation. Such information – in particular spatial information – could also be used to generate more interaction between users and collect more votes through possible common interests. In addition, users are helped by the system with images similar to the identification proposed in a given checklist. The additional information could guide users in their vote – for example by notifying a possible incoherence between the current botanical knowledge on a species and the metadata entered (such as the altitude, the distance to the sea, a species not known to survive in a given botanical area).

We also investigated the integration of the AI votes in the aggregation strategy. We concluded that a strategy involving a threshold on the prediction could be integrated, on the condition that the predicted probabilities are calibrated beforehand.

5 Conclusion and perspectives

5.1 Conclusion

In this thesis, we considered the problem of image classification via crowdsourced labels. First, we proposed a novel method – the WAUM – to identify possibly ambiguous images in a crowdsourced dataset. We showed qualitatively and quantitatively that the WAUM can help enhance computer vision model performances on crowdsourced datasets by identifying the tasks to remove. It is another step towards the automation of the data-cleaning process in crowdsourced datasets.

Second, we introduced the `peerannot` library to handle crowdsourced datasets in image classification settings. Our open-source library is designed to be user-friendly and efficient, and it provides a set of tools to preprocess, analyze, and train computer vision models from crowdsourced datasets. We also created a benchmark in the `benchopt` library for label aggregation strategies so that the community can easily compare their methods on crowdsourced datasets.

Finally, we considered the Pl@ntNet project framework and evaluated the performance of their label aggregation strategy on a newly released dataset. Very few label aggregation strategies can be run on a dataset with such a large number of tasks, workers and classes. Experimental results showed that the Pl@ntNet label aggregation strategy performs best in this setting. We also proposed different strategies to improve it, and the current solution found is to use the probability output of Pl@ntNet's model as prior knowledge in the label aggregation strategy – with a threshold on the probability considered.

5.2 Perspectives

Each work presented in the chapters of this thesis has immediate and long-term perspectives that we detail below.

5.2.1 WAUM project.

The WAUM method is a weighted average. The weights are currently relying on the DS confusion matrices, which prevents them from being used in a setting with a large number of classes. A long-term perspective would be to consider other weights, that do

not rely on DS strategies, to make the WAUM method usable in a large number of class settings. Preferably weights that can be theoretically evaluated to have performance guarantees.

In the shorter term, the current way to use the WAUM method is to remove the tasks with the WAUM below a quantile $\alpha \in \mathbb{R}$. This is class-agnostic and, in highly imbalanced settings, could remove a class entirely from the tIn the longer term, `peerannot` has been created as an organization to provide multiple modules for other crowdsourced frameworks. Having a module to consider reinforcement learning for data collection strategies would be a great addition to the library. Indeed, in this thesis, we focused on how to handle collected data. However, the data collection process is also crucial in crowdsourced settings. Having a tool to evaluate the influence of actions and recommender systems for crowdsourced platforms would help researchers design better data collection strategies. raining set. A by-class quantile sequence $(\alpha_k)_{k\in[K]}$ could be introduced to only prune a few tasks per class and not globally. This would create a finer exploration of ambiguity in the training set.

5.2.2 Peerannot project.

We created the `peerannot` library to handle crowdsourced datasets in image classification settings. The current modules of `peerannot` allow users to identify poorly performing workers, and ambiguous tasks, to train computer vision models from aggregated labels, or to train a model that directly handles crowdsourced data. The library is designed to be user-friendly and efficient, and we hope that it will be used by the community to handle crowdsourced datasets.

In the short term, we plan to add more functionalities and strategies to the library. For example, the more flexible the input data, the more users can use the library.

In the longer term, `peerannot` has been created as an organization to provide multiple modules for other crowdsourced frameworks. Having a module to consider reinforcement learning for data collection strategies would be a great addition to the library. Indeed, in this thesis, we focused on how to handle collected data. However, the data collection process is also crucial in crowdsourced settings. Having a tool to evaluate the influence of actions and recommender systems for crowdsourced platforms would help researchers design better data collection strategies. This proactive approach would be a great addition to the library and could mitigate problems encountered later in the training pipeline.

5.2.3 Pl@ntNet project.

Finally, with the Pl@ntNet project, we evaluated the performance of their label aggregation strategy on a newly released dataset and proposed different strategies to improve it. The current solution found is to use the probability output of Pl@ntNet's model as prior knowledge in the label aggregation strategy – with a threshold on the probability considered. While this solution could help in practice, before deploying it, we need to

recalibrate the current model to have a better probability estimation. Indeed, if the strategy considers the probability output as the threshold, this probability should be well-calibrated to have a good performance.

In a longer term, the Pl@ntNet project is a citizen science project with a large number of users. However, we saw in practice that a large number of users contribute very few times to the platform. Related to the former perspective, a recommender system could be used to recommend tasks to users and have them engage more in the annotation process. The main difficulty is to provide users with tasks that are not too difficult for them, and that could be interesting for them to annotate, with very little knowledge about most users. Furthermore, the recommender system should also consider the quality of the annotations provided by the users to recommend tasks to them. And, the recommender system should be run on the fly with very little computation time to provide users with tasks to annotate directly on the smartphone application. More engagement would also help the overall quality of the Pl@ntNet computer vision model if it is not noisy, thus the label aggregation strategy should in parallel be updated to consider the engagement of each user with the recommendations. And, of course it would be better if there could be theoretically guaranteed on the performance of the recommender system or at least simulations of such processes.

A quick travel in the history of crowdsourcing

A.1 History of crowdsourcing

Crowdsourcing in its most general form is not new. We have records of large crowdsourcing experiments leading to high achievements in socio-economics fields and even playing parts in wars. Let us consider one historical example.

One of the eldest and most impactful happened in 1848 and is reported extensively in Hearn (2002). Led by Matthew Fontaine Maury (an oceanographer between many other titles), the U.S. Naval Observatory distributed free copies of his book *Wind and Current Charts* which described how to effectively reduce time travel at sea, see Figure A.1 – *shippers knew that every day saved at sea cut costs and lessened the danger of losing ships, cargoes and crews.* In return, Maury asked sailors to record standardized logs and return them to him at the U.S. Naval Observatory. This form came with a ten-page instruction pamphlet on how to interpret the charts and how to fill out the logs. The experiment ran for years, and even at the early beginning shipmasters wanted to increase their profit and participate: *By September, the logs began flowing into the observatory, and Maury and his staff were busy trying to handle the volume.* Collecting this information led to reducing the Rio de Janeiro - New York was then fifty-five days and was reduced to twenty-three in 1853. They simplified and extended the charts with the full support of Navy captains and merchant companies like Forbes of Boston and Robert C. Wright of Baltimore for their sailors to fill out these forms. More importantly, they noticed some sailors did not use the charts correctly, and this helped prove the importance of taking into account winds and currents at sea. The newly collected data also helped create monthly recommendation charts for sailors.

But the history of Maury's logs implications did not stop at currents and winds. In 1851, they released charts showing the distribution of sperm whales in oceans for each season – an example is shown in Figure A.2. These tracks were used by whalers, but also during the Civil War by confederates to track Union ships that captured and harvested animals [1]. This disrupted the North's economy as sperm whale's oil was popular for

[1] `https://www.nytimes.com/1865/08/27/archives/the-pirate-shenandoah-her-cruise-in-the-arctic-seas-wholesale.html`

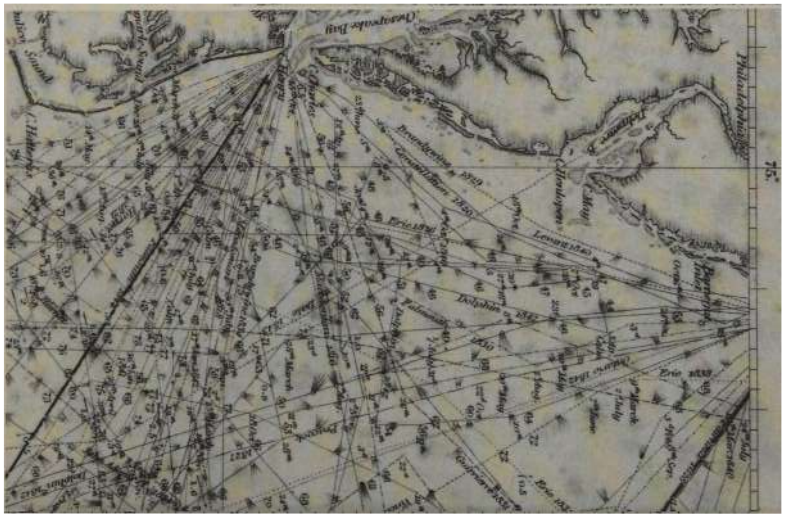


Figure A.1 Extract of the first edition of Wind and Current Chart of the North-Atlantic (1848) by Matthew Fontaine Maury. The brushes show the direction wind blows and their strength. The currents are shown with arrows. We can also see the water temperatures around the Chesapeake Bay.

soaps, lubricants and also as light sources [2].

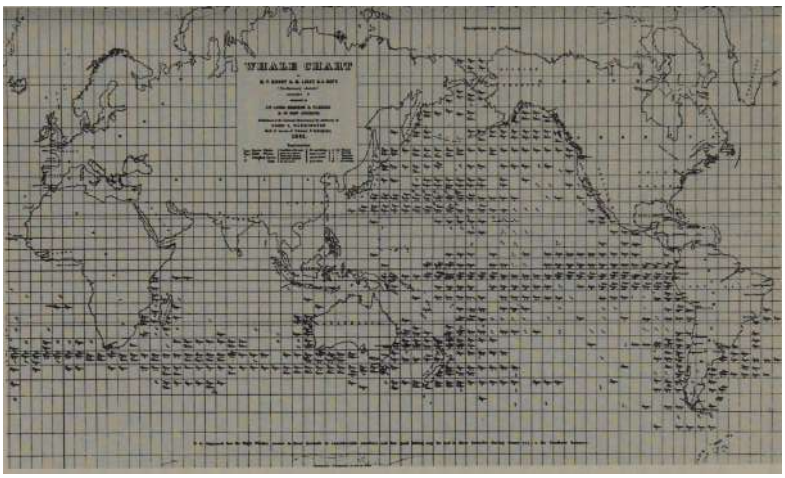


Figure A.2 Whale chart by Matthew Fontaine Maury from 1851, series F showing the distribution of sperm whales across oceans.

A.2 Defining crowdsourcing

Even though crowdsourcing isn't new, this is not the case for its proposed definition. Because of its large possibilities of applications and the increasingly extensive use of the

[2] `https://www.nytimes.com/2008/08/03/nyregion/03towns.html`

internet to conduct crowdsourcing experiments, there are multiple definitions of the term.

One of the first was recorded in 2006 by two journalists at Wired, Jeff Howe and Mark Robinson:

> *Simply defined, crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. This can take the form of peer-production (when the job is performed collaboratively), but is also often undertaken by sole individuals. The crucial prerequisite is the use of the open call format and the large network of potential laborers.*

Later, Howe proposed to divide it into two definitions[3]:

- *The White Paper Version: Crowdsourcing is the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call.*
- *The Soundbyte Version: The application of Open Source principles to fields outside of software.*

With the increasing use and misuse of the word crowdsourcing, Estellés-Arolas and González-Ladrón-de Guevara (2012) provided a broader internet-based definition that was extracted from merging definitions from forty papers – from a database of journal and conference papers about crowdsourcing containing 209 papers. The definition is as follows:

> *Crowdsourcing is a type of participative online activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task, of variable complexity and modularity, and in which the crowd should participate bringing their work, money, knowledge and/or experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need, be it economic, social recognition, self-esteem, or the development of individual skills, while the crowdsourcer will obtain and utilize to their advantage that what the user has brought to the venture, whose form will depend on the type of activity undertaken.*

One of their criteria was to explicitly take into account the "internet" factor to define modern crowdsourcing, thus this does not apply to experiments like Matthew Fontaine

[3] `https://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html`

Maury's (Appendix A.1). However, in the context of this thesis, and given how the vast majority of crowdsourcing experiments are nowadays used thanks to the internet, this definition is sufficient for our research purposes.

More importantly, the definition of crowdsourcing does not include peer production. These two are often mixed up, but experts (Brabham, 2013) differentiate them with multiple criteria – the most important being the guidelines. Let us take from Brabham (2013) the example of one of the largest peer-production projects: Wikipedia. It can not be considered a crowdsourcing project as there is no guideline given on what should be in an article or how it should be structured. There is also no vertical control. Wikipedia is controlled via peer assessments, so it is a horizontal control through dialogues. In a crowdsourcing experiment, a crowdsourcer (company, laboratory,...) provides the task(s) and what should be done to complete them. This can be as simple as *Describe the image* or *click on the bike*, or more complicated like *can the second part of the proposition be inferred from the first part?* but the main point is that there is a clear guideline given directly to the worker to complete the task(s) at hand.

A.3 Explicit and implicit crowdsourcing

From the multiple definitions of the term crowdsourcing, we see that broad implications lead to multi-faceted types of work. More than the task itself to be completed, it is important to record how the tasks were presented to workers. We know from Appendix A.2 that a guideline plays a crucial part in the experiment. However, a guideline can hide the purpose of an experiment. So crowdsourcing has been divided between two categories (Andro and Saleh, 2017):

- implicit crowdsourcing: recourse to involuntary work by Internet users,
- explicit crowdsourcing: recourse to voluntary work from voluntary Internet users.

A famous example of implicit crowdsourcing tasks is the `reCAPTCHA` created by Luis von Ahn. The principle is simple, we need to archive old books, and sometimes text recognition systems can not recognize the words – for numerous reasons such that, and not exhaustive, paper quality, symbols and calligraphy, overlapping words – and those words are given to write out to people on the internet just as ordinary security systems (see Figure A.3). Most of the time, there are two `CAPTCHA`s to solve, one for control and then the actual task. During the whole solving time, the worker does not know that they are solving an actual problem for a crowdsourcer, hence the implicit. Multiple other examples for each are presented in Appendix A.4 with a more detailed classification of crowdsourcing tasks.

Figure A.3 Example of `reCAPTCHA` from 2014 for word recognition tasks. The word here is `morning` and the control word `upon`.

A.4 Current ways to classify and run experiments

Defining types of crowdsourcing is not the same as defining types of crowds. In Kazai et al. (2011) created five worker profiles: spammer, sloppy, incompetent, competent, and diligent. In Martineau (2012) workers are classified as communal, lurkers, utilizers and aspirers. This quest for worker profile identification is still an active field of research.

But taking a step back in this problem, Brabham (2013) proposes a typology to classify crowdsourcing experiments: not tasks nor workers.

- knowledge discovery and management: find and collect standardized information – *e.g* Flag association application to register discrimination and act of violence against LGBTQ+ people for the French government[4],
- broadcast search: solve empirical problems – *e.g* coding challenges as Kaggle
- peer-vetted creative production: create and select creatives ideas – *e.g* Threadless[5] allow its community to create designs for shirts or prints and then puts them to a vote each week,
- Distributed-human-intelligence tasking; analyze large amounts of information – *e.g* label images or part of images like Pl@ntNet.

In practice, running a crowdsourcing experiment has been eased up by platforms such as Amazon Mechanical Turk[6] or Toloka[7]. These marketplaces allow to conduct of a paid experiment – ethical considerations apart (and discussed in section 1.1.3) – that is planned with extensive parametrization. The data labeling is outsourced so you don't need to think about the resource allocations. They also have algorithms (some like

[4] `https://www.flagasso.com/`
[5] `https://www.threadless.com/`
[6] `https://mkturk.com/`
[7] `https://toloka.yandex.com`

Raykar and Yu (2011) that we discuss in Chapter 3) to detect unreliable workers to have better data.

Figure A.4 Example of ESP game image (from `https://edutechwiki.unige.ch`). The image represents four sheep laying in the grass. The first worker proposes `sheep` and then `sheep` at the second round. Note that to avoid having the same labels by multiple workers for the images, taboo words were introduced. In this case, workers were forbidden to answer the words `peace` and `lay` to describe the image. Not every image had taboo words.

However, paying workers is not the only way to collect crowdsourced data. The gamification of crowdsourcing tasks has proved to be quite efficient with experiments like Eyewire (Tinati et al., 2017), ThePlantGame (Servajean et al., 2016), *etc.* Luis Von Ahn (the `CAPTCHA` creator) was one of the first researchers on GWAPs (Games With A Purpose). He famously also founded in 2004 the ESP (ExtraSensory Perception) game (Von Ahn and Dabbish, 2005) to create metadata on images – presented in Figure A.4. Two players are paired up randomly and shown the same image. They can not communicate, but they each can provide a single word to describe the image presented. Once they both provide the same word, the game stops. They have 2 minutes and 30 seconds to label 15 images. At one point, they can both choose to pass on a single image. A license was bought by Google to create the Google Image Labeler[8] from 2006 to 2011.

We have seen paid workers and playing workers, but sometimes workers just participate voluntarily without any second motivation from the crowdsourcer. This is the case in Pl@ntNet where the workers' only gain in participating is providing more data to improve the model's prediction for the community. Here, the gain is scientific and the participation is based on providing better tools for the community – a sense of having participated to help others. RTE, France also created a crowdsourcing platform[9] to identify photovoltaic panels on images and delimit their area of occupancy (Kasmi et al., 2023).

[8] `http://news.bbc.co.uk/2/hi/technology/7395751.stm`
[9] `https://www.bdpv.fr/_BDapPV/`

In short, with crowdsourcing, we can classify workers, types of tasks/experimentation and the incentives to make workers answer those tasks. Each of these parts can play a role in the data collection process and quality of said data.

B Peerannot appendix

B.1 Code structure to implement the DS model

In `peerannot`, iterative label aggregation strategies need a `.run()` method. We present an example of such a structure with the DS model in Listing 11.

B.2 Simulated mistakes with discrete difficulty levels on tasks

For an additional experiment, we consider the so-called discrete difficulty setting 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 difficulties: `easy` or `hard`. The keyword `ratio-diff` indicates the prevalence of each level of difficulty of tasks as:

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

Tasks that are `easy` are answered correctly by every assigned 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 200 tasks and 100 workers with 35% of good workers and 50% of `easy` tasks (`ratio-diff`$= 1$). There are $K = 5$ classes. Each task receives $|\mathcal{A}(x_i)| = 10$ votes.

```
$ 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
```

Listing 10 Simulation of discrete difficulty levels crowdsourced datasets in `peerannot`.

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

Table B.1 AccTrain metric on simulated mistakes made when tasks are associated with a difficulty level considering classical feature-blind label aggregation strategies.

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

strategies (see Table B.1). Note that in this case, creating clusters of answers leads to worse decisions than an MV aggregation.

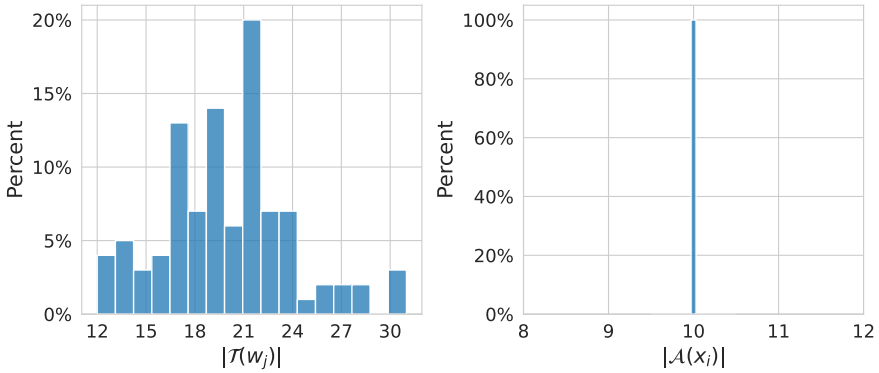


Figure B.1 Distribution of the number of tasks given per worker (left) and of the number of labels per task (right) in the setting with simulated discrete difficulty levels.

```python
class Dawid_Skene(CrowdModel):
    def __init__(self, answers, n_classes, **kwargs):
        super().__init__(answers)
        ...

    def get_crowd_matrix(self):
        ... # Convert json answers to tensor (task, worker, label)

    def init_T(self):
        ... # Initialize the confusion matrices

    def m_step(self):
        """Maximizing log likelihood
        Returns:
            p: (p_j)_j class marginals
            pi: confusion matrices
        """
        ...

    def e_step(self):
        """Estimate indicator variables
        Returns:
            T: New estimate for the labels (n_task, n_worker)
        """
        ...

    def log_likelihood(self):
        ... # Compute the log likelihood of the model

    def run(self, epsilon=1e-6, maxiter=50):
        self.get_crowd_matrix()
        self.init_T()
        k, eps, ll = 0, np.inf, []
        while k < maxiter and eps > epsilon:
            self.m_step()
            self.e_step()
            likeli = self.log_likelihood()
            ll.append(likeli)
            if len(ll) >= 2:
                eps = np.abs(ll[-1] - ll[-2])
            k += 1

    def get_probas(self):
        return self.T

    def get_answers(self):
        return np.vectorize(self.converter.inv_labels.get)(
            np.argmax(self.get_probas(), axis=1)
        )
```

Listing 11 MWE for the DS label aggregation in `peerannot`.

Résumé et partie de l'introduction en français

C.1 Résumé

Alors que les ensembles de données de classification sont composés d'un nombre croissant d'éléments, le besoin d'expertise humaine pour les étiqueter est toujours présent. Les plateformes d'apprentissage participatif sont un moyen de recueillir les commentaires d'experts à faible coût. Cependant, la qualité de ces étiquettes n'est pas toujours garantie. Dans cette thèse, nous nous concentrons sur le problème de l'ambiguïté des étiquettes dans l'apprentissage participatif. L'ambiguïté des étiquettes a principalement deux sources : la capacité du travailleur et la difficulté de la tâche. Nous présentons tout d'abord un nouvel indicateur, le WAUM (aire sous la marge pondérée), pour détecter les tâches ambiguës confiées aux travailleurs. Basé sur le AUM existant dans le cadre supervisé classique, il nous permet d'explorer de grands ensembles de données tout en nous concentrant sur les tâches qui pourraient nécessiter une expertise plus pertinente ou qui devraient être éliminées de l'ensemble de données actuel. Nous présentons ensuite une nouvelle bibliothèque Python open-source, PeerAnnot, que nous avons développée pour traiter les ensembles de données en apprentissage participatif dans la classification d'images. Nous avons créé une référence d'état de performance dans la bibliothèque Benchopt pour évaluer nos stratégies d'agrégation d'étiquettes afin d'obtenir des résultats plus reproductibles. Enfin, nous présentons une étude de cas sur l'ensemble de données Pl@ntNet, où nous évaluons l'état actuel de la stratégie d'agrégation d'étiquettes de la plateforme et proposons des moyens de l'améliorer. Ce contexte avec un grand nombre de tâches, d'experts et de classes est très difficile pour les stratégies d'agrégation d'apprentissage participatif actuelles. Nous faisons état de performances systématiquement supérieures à celles de nos concurrents et proposons une nouvelle stratégie d'agrégation qui pourrait être utilisée à l'avenir pour améliorer la qualité de l'ensemble de données Pl@ntNet. Nous publions également ce grand ensemble de données de commentaires d'experts qui pourrait être utilisé pour améliorer la qualité des méthodes d'agrégation actuelles et fournir un nouveau point de référence.

C.2 Introduction: apprentissage participatif en classification d'images

Suite à la révolution de l'apprentissage profond, on entraîne aujourd'hui des modèles avec des millions voire des milliards de paramètres à optimiser. Et pour ce faire, nous avons besoin d'augmenter la taille de nos ensembles de données d'entraînement. Le problème est que dans un ensemble d'entraînement en classification supervisée (et nous considérerons principalement la classification d'images pour le reste de ce travail), les images doivent être accompagnées d'une étiquette à partir de laquelle l'entraînement est effectué. Bien que des recherches aient été menéees sur l'inférence de cette étiquette à partir des données elles-mêmes, dans les ensembles de données classiques d'apprentissage supervisé, ces étiquettes ont été collectées et partagées à un moment donné au cours de la création de l'ensemble de données.

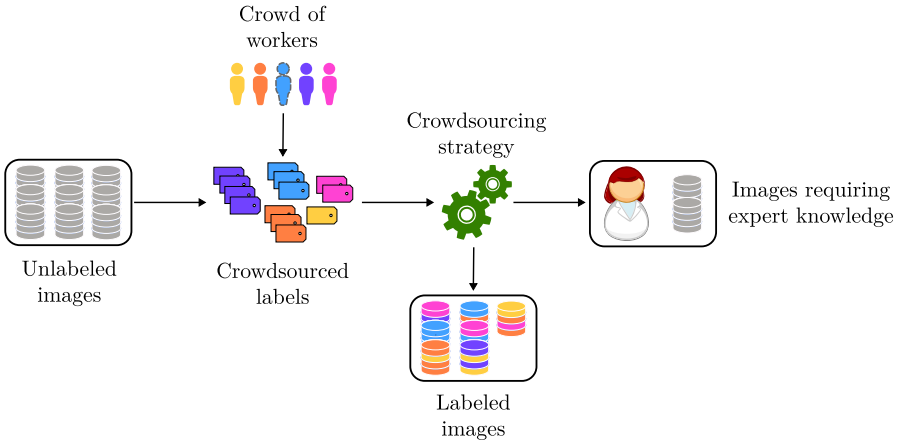


Figure C.1 L'apprentissage participatif peut être utilisé pour créer des ensembles de données, mais il permet également de soulager la tension que subissent les experts lorsqu'ils évaluent de nouvelles données. En faisant appel à une foule de travailleurs, de nombreuses tâches peuvent être étiquetées sans nécessiter l'évaluation d'un expert.

La tâche consistant à collecter des données afin de créer un ensemble de données pour la classification des images a été de plus en plus associée à l'extraction automatique de données en ligne (Rhodes et al., 2015). Toutefois, il a été démontré à de nombreuses reprises que cette stratégie de collecte de données entraîne des erreurs d'étiquetage (Northcutt et al., 2021a; Vasudevan et al., 2022). Dans d'autres domaines de recherche, comme les prestataires de soins de santé, les images ne proviennent pas nécessairement du web mais sont collectées à plusieurs endroits, comme les scanners des hôpitaux. Lorsqu'il s'agit de classer un scanner comme "présence ou absence de tumeurs", c'est souvent un expert sur le terrain qui fournit l'étiquette. Cependant, ces professionnels de la santé ne peuvent pas étiqueter des milliers d'images. Dans ce cas, la puissance de la foule peut être utilisée pour étiqueter les scanners et fournir un certain niveau d'incertitude. Il appartient alors au scientifique des données, en collaboration avec les experts, de déterminer quels travailleurs ont été utiles dans l'expérience et lesquels ne l'ont pas été,

puis de demander aux experts d'étiqueter uniquement les scans les plus ambigus - qui ne représenteraient qu'une petite fraction de l'ensemble de données original (comme illustré dans Figure 1.1).

C.2.1 Le crowdsourcing est partout

Cette collaboration entre citoyens et scientifiques n'est pas une niche technique, mais elle passe souvent inaperçue. L'utilisation du crowdsourcing pour collecter des données a été utilisée en médecine, dans les systèmes de recommandation vidéo, et dans bien d'autres domaines de la recherche et du développement. Nous souhaitons présenter une liste - non exhaustive et non ordonnée - de ces projets qui s'appuient sur les interactions humaines pour collecter des données, et pas seulement pour la classification d'images :

- Pl@ntNet : application de reconnaissance des espèces végétales (Barthélémy et al., 2011),
- Eyewire : un jeu pour cartographier les neurones de la rétine (Tinati et al., 2017),
- Tournesol : système de recommandation de vidéos d'intérêt public sur YouTube (Hoang et al., 2021), (OpenAI, 2023), (OpenAI, 2023), (OpenAI, 2023), (OpenAI, 2023),
- Duolingo : corrections de traduction et améliorations[1],
- RTR : segmentation des panneaux photovoltaïques dans les images (Kasmi et al., 2023),
- Twitter/𝕏 via Birdwatch : identifier les informations trompeuses (Wojcik et al., 2022),
- Spotify[2], TripAdvisor[3], SNCF[4],...

Avec cette liste, nous mettons en évidence les vastes domaines d'application qui permettent aux humains - et aux citoyens - de rester dans la boucle, et donc la nécessité de poursuivre les recherches sur les paramètres, les stratégies et les implications du crowdsourcing. Dans cette thèse, nous nous concentrons principalement sur la classification d'images.

[1] `https://www.theguardian.com/education/2014/aug/27/luis-von-ahn-ceo-duolingo-interview`

[2] `https://community.spotify.com/t5/Content-Questions/Shutting-down-Line-In/td-p/4557664`

[3] `https://www.kingigilbert.com/crowdsourcing-tripadvisor/`

[4] `https://www.usine-digitale.fr/article/tranquilien-quand-open-data-et-crowdsourcing-profitent-aux-voyageurs-franciliens.N200017`

C.2.2 Le projet de Pl@ntNet

Créé par Alexis Joly et Pierre Bonnet, Pl@ntNet (Barthélémy et al., 2011) est la rencontre de deux communautés – les botanistes et l'informatique – pour identifier des espèces végétales à partir de photos prises sur le vif. Le projet est né en 2008, a sorti sa première application web en 2011, son application mobile en 2013 et a reçu en 2020 le prix *Prix de l'innovation Inria - Académie des sciences - Dassault Systèmes.*

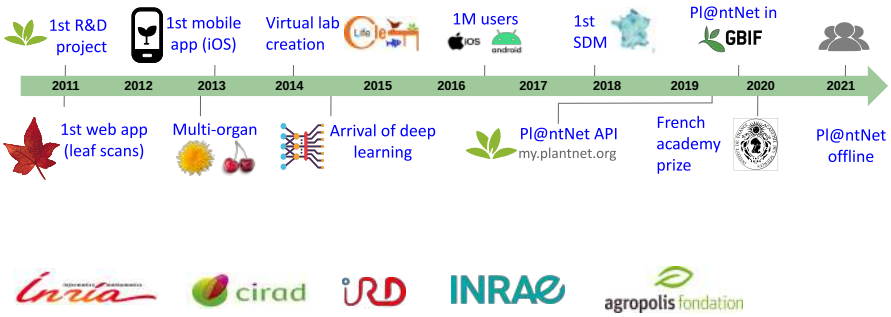


Figure C.2 Timeline du projet Pl@ntNet

Le système Pl@ntNet peut être interprété comme suit. Un utilisateur enregistre une observation (une image ou un groupe d'images de la même plante) et envoie une requête d'identification. La version actuelle du modèle de vision par ordinateur de Pl@ntNet (appelé modèle IA de Pl@ntNet et détaillé dans Chapter 4) fait une prédiction et fournit l'espèce la plus probable. L'utilisateur peut alors approuver le modèle d'IA ou saisir une autre espèce. Avec les probabilités prédites, des observations similaires avec les espèces indiquées sont montrées à l'utilisateur pour l'aider à prendre sa décision. L'observation est ensuite partagée (avec l'accord de l'utilisateur) avec la communauté et peut faire l'objet d'un vote de la part des autres utilisateurs/

Une fois l'observation enregistrée, elle peut être révisée à tout moment par d'autres utilisateurs. Actuellement, Pl@ntNet enregistre plus de 50K espèces, 6M utilisateurs enregistrés et 21M observations réparties sur 77 flores. Plus d'un milliard de requêtes d'images ont été effectuées et l'application a été téléchargée plus de 10 millions de fois sur Google AppStore. Au total, plus de 22 millions de votes ont été exprimés au niveau international. Chaque observation partagée publiquement présente, comme dans Figure 1.4 avec un auteur associé, les votes actuels et l'état de la détermination actuelle de l'espèce. Sur la plateforme, les utilisateurs peuvent voter pour une étiquette malformée, si l'image concerne une feuille, une fleur, un fruit, l'écorce, le port ou autre. Dans cette thèse, nous ne prenons en compte que la détermination de l'espèce, et non les autres votes.

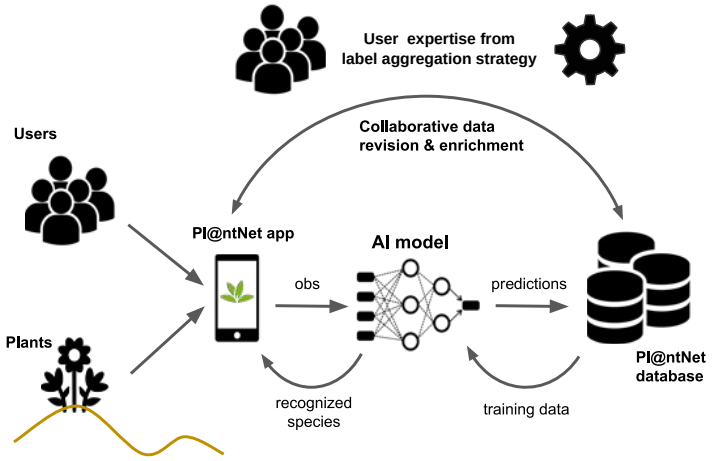
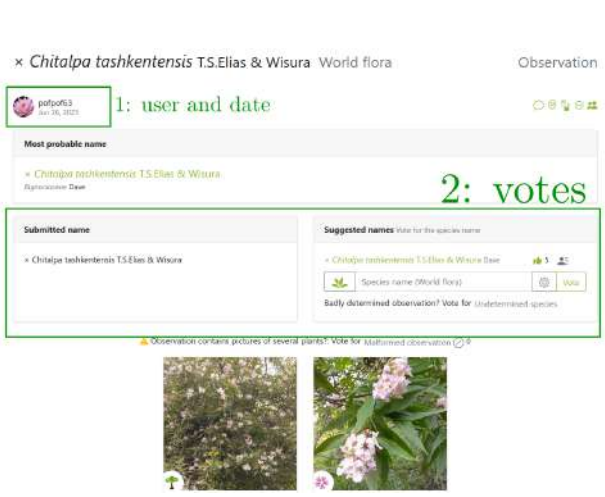
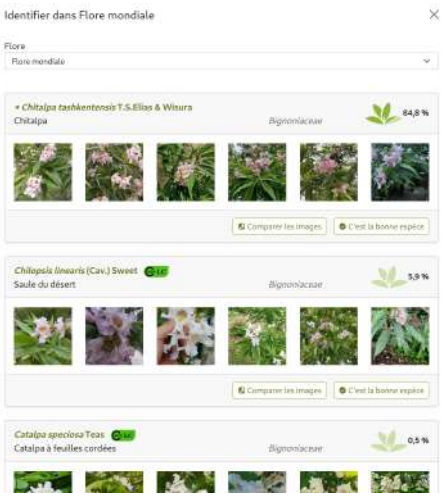


Figure C.3 Pl@ntNet pipeline from the observation taken in the field to the active training of the computer vision model.



(a) Identification webpage. L'observation est composée de deux images, l'une de l'organe (à gauche) et l'autre de la fleur (à droite).



(b) Vision de l'utilisateur sur les prédictions du modèle actuel. Des exemples pour chaque espèce sont proposés à des fins de comparaison visuelle.

Figure C.4 Exemple d'observation à partir de l'interface en ligne de Pl@ntNet. L'utilisateur représenté est l'auteur de l'observation. Nous enregistrons le nom de l'espèce initialement soumis, ainsi que les différents votes (ici, 5 utilisateurs sont d'accord sur la détermination de l'étiquette). Si les images d'une observation ne contiennent pas la même plante, les utilisateurs peuvent voter pour une étiquette malformée. En cliquant sur l'icône près du champ d'identification, les utilisateurs peuvent voir la prédiction actuelle du modèle de vision par ordinateur.

Lors du vote, les utilisateurs voient apparaître la prédiction actuelle de vision par ordinateur sur l'observation – comme indiqué dans Figure 1.4b. Cette prédiction influence

le vote des utilisateurs, mais contient également des informations. Au moment de la rédaction de ce document, le modèle de vision par ordinateur de Pl@ntNet est un réseau DINOv2 (Oquab et al., 2024) basé sur des transformateurs. En utilisant l'apprentissage contrastif (Waida et al., 2023) *i.e.* représentant des images similaires sous forme d'encastrement proche pour apprendre des caractéristiques similaires pour des observations similaires, puis en utilisant l'apprentissage supervisé pour affiner le modèle.

Dans Chapter 4, nous présentons la stratégie d'agrégation des étiquettes de Pl@ntNet, mais aussi la manière dont nous pouvons incorporer les prédictions du modèle dans l'agrégation des étiquettes. La prise en compte du vote du réseau est une question délicate, car nous ne voulons pas que l'IA devienne trop confiante dans ses prédictions en s'appuyant principalement sur ses prédictions et en écartant les votes humains. L'expertise humaine étant l'objectif principal de la plateforme, nous voulons garder les humains dans la boucle, en particulier les experts botaniques, afin de continuer à améliorer les identifications de plantes des utilisateurs finaux.

Bibliography

(2013). The plant list. Published on the Internet. Accessed 1st January.

(2024). International plant names index. Published on the Internet.

Affouard, A., Goëau, H., Bonnet, P., Lombardo, J.-C., and Joly, A. (2017). Pl@ ntnet app in the era of deep learning. In *ICLR: International Conference on Learning Representations*.

Aitchison, L. (2021). A statistical theory of cold posteriors in deep neural networks. In *ICLR*.

Andro, M. and Saleh, I. (2017). Digital libraries and crowdsourcing: A review. *Collective Intelligence and Digital Archives: Towards Knowledge Ecosystems*, 1:135–161.

Angelova, A. (2004). *Data pruning*. PhD thesis, California Institute of Technology.

Ao, S., Rueger, S., and Siddharthan, A. (2023). Two sides of miscalibration: identifying over and under-confidence prediction for network calibration. In *Uncertainty in Artificial Intelligence*, pages 77–87. PMLR.

Baker, M. (2016). Reproducibility crisis. *nature*, 533(26):353–66.

Bao, H., Dong, L., Piao, S., and Wei, F. (2021). Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*.

Barthélémy, D., Boujemaa, N., Mathieu, D., Molino, J., Joly, A., Goëau, H., Bonnet, P., Mouysset, E., Birnbaum, P., Roche, V., Ouertani, W., and Grard, P. (2011). The Pl@ntNet project: Plant Computational Identification and Collaborative Information System. In *Biosystematics 2011 - Digital identification methods in the digital age - morphological and molecular, computer-aided and fully automated, research and citizen science*.

Bartlett, P., Freund, Y., Lee, W., and Schapire, R. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*.

Bonnet, P., Joly, A., Faton, J.-M., Brown, S., Kimiti, D., Deneu, B., Servajean, M., Affouard, A., Lombardo, J.-C., Mary, L., et al. (2020). How citizen scientists contribute to monitor protected areas thanks to automatic plant identification tools. *Ecological Solutions and Evidence*, 1(2):e12023.

Borowiec, M. L., Dikow, R., Frandsen, P. B., McKeeken, A., Valentini, G., and White, A. E. (2022). Deep learning as a tool for ecology and evolution. *Methods in Ecology and Evolution*, 13(8):1640–1660.

Brabham, D. C. (2013). *Crowdsourcing*. The MIT Press.

Brown, E. D. and Williams, B. K. (2019). The potential for citizen science to produce reliable and useful information in ecology. *Conservation Biology*, 33(3):561–569.

Brummitt, R. K., Pando, F., Hollis, S., and Brummitt, N. (2001). *World geographical scheme for recording plant distributions*, volume 951. International working group on taxonomic databases for plant sciences (TDWG) ....

Budach, L., Feuerpfeil, M., Ihde, N., Nathansen, A., Noack, N., Patzlaff, H., Naumann, F., and Harmouch, H. (2022). The effects of data quality on machine learning performance.

Cachuela-palacio, M. (2006). Towards an index of all known species: the catalogue of life, its rationale, design and use. *Integrative zoology*, 1(1):18–21.

Cao, P., Xu, Y., Kong, Y., and Wang, Y. (2019). Max-mig: an information theoretic approach for joint learning from crowds. In *ICLR*.

Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. (2020). Unsupervised learning of visual features by contrasting cluster assignments. *Advances in neural information processing systems*, 33:9912–9924.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. (2021). Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660.

Chamberlain, J., Kruschwitz, U., and Poesio, M. (2018). Optimising crowdsourcing efficiency: Amplifying human computation with validation. *it-Information Technology*, 60(1):41–49.

Chu, Z., Ma, J., and Wang, H. (2021). Learning from crowds by modeling common confusions. In *AAAI*, pages 5832–5840.

Collins, K. M., Bhatt, U., and Weller, A. (2022). Eliciting and learning with soft labels from every annotator. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, pages 40–52.

Dawid, A. and Skene, A. (1979). Maximum likelihood estimation of observer error-rates using the EM algorithm. *J. R. Stat. Soc. Ser. C. Appl. Stat.*, 28(1):20–28.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 39(1):1–22.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

Dong, M. (2018). Convolutional neural network achieves human-level accuracy in music genre classification. *arXiv preprint arXiv:1802.09697*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Estellés-Arolas, E. and González-Ladrón-de Guevara, F. (2012). Towards an integrated crowdsourcing definition. *Journal of Information science*, 38(2):189–200.

Experts, D. L. T. C. F. (2018). Plant identification: Experts vs. machines in the era of deep learning. *Multimedia Tools and Applications for Environmental & Biodiversity Informatics*, page 131.

Freiberg, M., Winter, M., Gentile, A., Zizka, A., Muellner-Riehl, A. N., Weigelt, A., and Wirth, C. (2020). The leipzig catalogue of vascular plants (lcvp)–an improved taxonomic reference list for all known vascular plants. *BioRxiv*, pages 2020–05.

Gao, G. and Zhou, D. (2013). Minimax optimal convergence rates for estimating ground truth from crowdsourced labels. *arXiv preprint arXiv:1310.5764*.

Gao, M., Xu, W., and Callison-Burch, C. (2015). Cost optimization in crowdsourcing translation: Low cost translations made even cheaper. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 705–713.

Garcin, C., Servajean, M., Joly, A., and Salmon, J. (2022). Stochastic smoothing of the top-k calibrated hinge loss for deep imbalanced classification. In *ICML*.

Govaerts, R. (2023). The world checklist of vascular plants (wcvp). Checklist dataset. Accessed via GBIF.org on 2024-01-30.

Govaerts, R., Nic Lughadha, E., Black, N., Turner, R., and Paton, A. (2021). The world checklist of vascular plants, a continuously updated resource for exploring global plant diversity. *Scientific data*, 8(1):215.

Granese, F., Romanelli, M., Gorla, D., Palamidessi, C., and Piantanida, P. (2021). Doctor: A simple method for detecting misclassification errors. *Advances in Neural Information Processing Systems*, 34:5669–5681.

Gruber, S. G. and Buettner, F. (2022). Better uncertainty calibration via proper scores for classification and beyond. In *Advances in Neural Information Processing Systems*.

Guérin, J., Delmas, K., Ferreira, R., and Guiochet, J. (2023). Out-of-distribution detection is not all you need. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 14829–14837.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. (2017). On calibration of modern neural networks. In *ICML*, page 1321.

Han, J., Luo, P., and Wang, X. (2019). Deep self-learning from noisy labels. In *ICCV*, pages 5138–5147.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*, pages 770–778.

Hearn, C. (2002). *Tracks in the Sea: Matthew Fontaine Maury and the Mapping of the Oceans*. International Marine/Ragged Mountain Press.

Hoang, L., Faucon, L., Jungo, A., Volodin, S., Papuc, D., Liossatos, O., Crulis, B., Tighanimine, M., Constantin, I., Kucherenko, A., Maurer, A., Grimberg, F., Nitu, V., Vossen, C., Rouault, S., and El-Mhamdi, E. (2021). Tournesol: A quest for a large, secure and trustworthy database of reliable human judgments. *arXiv preprint arXiv:2107.07334*.

Hovy, D., Berg-Kirkpatrick, T., Vaswani, A., and Hovy, E. (2013). Learning whom to trust with mace. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1120–1130.

Ilyas, A., Park, S., Engstrom, L., Leclerc, G., and Madry, A. (2022). Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*.

Imamura, H., Sato, I., and Sugiyama, M. (2018). Analysis of minimax error rate for crowdsourcing and its application to worker clustering model. In *ICML*, pages 2147–2156.

Jiang, X., Osl, M., Kim, J., and Ohno-Machado, L. (2012). Calibrating predictive model estimates to support personalized medicine. *J. Am. Med. Inform. Assoc.*, 19(2):263–274.

Joly, A. and Buisson, O. (2008). A posteriori multi-probe locality sensitive hashing. In *Proceedings of the 16th ACM international conference on Multimedia*, pages 209–218.

Joly, A. and Buisson, O. (2011). Random maximum margin hashing. In *CVPR 2011*, pages 873–880. IEEE.

Ju, C., Bibaut, A., and van der Laan, M. (2018). The relative performance of ensemble methods with deep convolutional neural networks for image classification. *J. Appl. Stat.*, 45(15):2800–2818.

Kapoor, S., Maddox, W., Izmailov, P., and Wilson, A. (2022). On uncertainty, tempering, and data augmentation in bayesian classification.

Karger, D., Oh, S., and Shah, D. (2011). Iterative learning for reliable crowdsourcing systems. *Advances in neural information processing systems*, 24.

Kasmi, G., Saint-Drenan, Y.-M., Trebosc, D., Jolivet, R., Leloux, J., Sarr, B., and Dubus, L. (2023). A crowdsourced dataset of aerial images with annotated solar photovoltaic arrays and installation metadata. *Scientific Data*, 10(1):59.

Kazai, G., Kamps, J., and Milic-Frayling, N. (2011). Worker types and personality traits in crowdsourcing relevance labels. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1941–1944.

Kelling, S., Johnston, A., Hochachka, W. M., Iliff, M., Fink, D., Gerbracht, J., Lagoze, C., La Sorte, F. A., Moore, T., Wiggins, A., Wong, W., Wood, C., and Yu, J. (2015). Can observation skills of citizen scientists be estimated using species accumulation curves? *PLOS ONE*, 10(10):1–20.

Khattak, F. (2017). *Toward a Robust and Universal Crowd Labeling Framework*. PhD thesis, Columbia University.

Krippendorff, K. (1980). Validity in content analysis. In *Computerstrategien für die Kommunikationsanalyse*, volume 69.

Krippendorff, K. (2004). Reliability in content analysis: Some common misconceptions and recommendations. *Human communication research*, 30(3):411–433.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Kumar, A., Liang, P. S., and Ma, T. (2019). Verified uncertainty calibration. In *NeurIPS*, volume 32.

Lapin, M., Hein, M., and Schiele, B. (2016). Loss functions for top-k error: Analysis and insights. In *CVPR*, pages 1468–1477.

Lease, M., Hullman, J., Bigham, J., Bernstein, M., Kim, J., Lasecki, W., Bakhshi, S., Mitra, T., and Miller, R. (2013). Mechanical turk is not anonymous. *Available at SSRN 2228728*.

Lefort, T., Charlier, B., Joly, A., and Salmon, J. (2022). Identify ambiguous tasks combining crowdsourced labels by weighting areas under the margin. *arXiv preprint arXiv:2209.15380*.

Lehikoinen, P., Rannisto, M., Camargo, U., Aintila, A., Lauha, P., Piirainen, E., Somervuo, P., and Ovaskainen, O. (2023). A successful crowdsourcing approach for bird sound classification. *Citizen Science: Theory and Practice*, 8(1):16.

Loshchilov, I. and Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Ma, Q. and Olshevsky, A. (2020). Adversarial crowdsourcing through robust rank-one matrix completion. In *NeurIPS*, volume 33, pages 21841–21852.

Mäder, P., Boho, D., Rzanny, M., Seeland, M., Wittich, H. C., Deggelmann, A., and Wäldchen, J. (2021). The flora incognita app–interactive plant species identification. *Methods in Ecology and Evolution*, 12(7):1335–1342.

Maennel, H., Alabdulmohsin, I., Tolstikhin, I., Baldock, R., Bousquet, O., Gelly, S., and Keysers, D. (2020). What do neural networks learn when trained with random labels? *Advances in Neural Information Processing Systems*, 33:19693–19704.

Marcel, S. and Rodriguez, Y. (2010). Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA. Association for Computing Machinery.

Martineau, E. (2012). *A typology of crowdsourcing participation styles*. PhD thesis, Concordia University.

Mayer, R. (2007). What's wrong with exploitation? *Journal of Applied Philosophy*, 24(2).

Merler, S., Caprile, B., and Furlanello, C. (2004). Bias-variance control via hard points shaving. *International Journal of Pattern Recognition and Artificial Intelligence*.

Naeini, M., Cooper, G., and Hauskrecht, M. (2015). Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI conference on artificial intelligence*.

Niculescu-Mizil, A. and Caruana, R. (2005). Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632.

Northcutt, C., Athalye, A., and Mueller, J. (2021a). Pervasive label errors in test sets destabilize machine learning benchmarks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*.

Northcutt, C., Jiang, L., and Chuang, I. (2021b). Confident learning: Estimating uncertainty in dataset labels. *J. Artif. Intell. Res.*, 70:1373–1411.

Nyström, T. (2021). Exploring the darkness of gamification: you want it darker? In *Intelligent Computing: Proceedings of the 2021 Computing Conference, Volume 3*. Springer.

OpenAI (2023). Gpt-4 technical report.

Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, P., Massa, F., El-Nouby, A., Assran, M., Ballas, N., Wojciech, G., Russell, H., Po-Yao, H., Shang-Wen, L., Ishan, M., Michael, R., Vasu, S., Synnaeve, G., Xu, H., Jegou, H., Mairal, J., Labatut, P., Joulin, A., and Bojanowski, P. (2024). Dinov2: Learning robust visual features without supervision.

Park, S. Y. and Caragea, C. (2022). On the calibration of pre-trained language models using mixup guided by area under the margin and saliency. In *ACML*, pages 5364–5374.

Passonneau, R. J. and Carpenter, B. (2014). The benefits of a model of annotation. *Transactions of the Association for Computational Linguistics*, 2:311–326.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830.

Peterson, J. C., Battleday, R. M., Griffiths, T. L., and Russakovsky, O. (2019). Human uncertainty makes classification more robust. In *ICCV*, pages 9617–9626.

Pirkkalainen, H. and Salo, M. (2016). Two decades of the dark side in the information systems basket: Suggesting five areas for future research. In *European Conference on Information Systems*. European Conference on Information Systems.

Platt, J. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74.

Pleiss, G., Zhang, T., Elenberg, E. R., and Weinberger, K. Q. (2020). Identifying mislabeled data using the area under the margin ranking. In *NeurIPS*.

POWO (2024). Plants of the world online. Published on the Internet.

Prechelt, L. (2002). Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer.

Raykar, V. C. and Yu, S. (2011). Ranking annotators for crowdsourced labeling tasks. In *NeurIPS*, pages 1809–1817.

Rhodes, B. B., Kim, A., and Loomis, B. R. (2015). Vaping the web: Crowdsourcing and web scraping for establishment survey frame generation. In *Proceedings of the 2015 Federal Committee on Statistical Methodology Research Conference*, volume 20.

Rodrigues, F., Lourenco, M., Ribeiro, B., and Pereira, F. C. (2017). Learning supervised topic models for classification and regression from crowds. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2409–2422.

Rodrigues, F. and Pereira, F. (2018). Deep learning from crowds. In *AAAI*, volume 32.

Rodrigues, F., Pereira, F., and Ribeiro, B. (2014). Gaussian process classification and active learning with multiple annotators. In *ICML*, pages 433–441. PMLR.

Rogl, R. (2016). No work and all play – the intersections between labour, fun and exploitation in online translation communities. *European Journal of Applied Linguistics*, 4(1):117–138.

Schellenberger Costa, D., Boehnisch, G., Freiberg, M., Govaerts, R., Grenié, M., Hassler, M., Kattge, J., Muellner-Riehl, A. N., Rojas Andrés, B. M., Winter, M., et al. (2023). The big four of plant taxonomy–a comparison of global checklists of vascular plant names. *New Phytologist*, 240(4):1687–1702.

Schmidt, F. o. C. and Computing, G. (2013). The good, the bad and the ugly: Why crowdsourcing needs ethics. In *IEEE*.

Schorn, C. and Gauerhof, L. (2020). Facer: A universal framework for detecting anomalous operation of deep neural networks. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE.

Servajean, M., Joly, A., Shasha, D., Champ, J., and Pacitti, E. (2016). Theplantgame: Actively training human annotators for domain-specific crowdsourcing. In *Proceedings of the 24th ACM International Conference on Multimedia*, MM '16, page 720–721, New York, NY, USA. Association for Computing Machinery.

Servajean, M., Joly, A., Shasha, D., Champ, J., and Pacitti, E. (2017). Crowdsourcing thousands of specialized labels: A Bayesian active training approach. *IEEE Transactions on Multimedia*, 19(6):1376–1391.

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

Snow, R., O'Connor, B., Jurafsky, D., and Ng, A. (2008). Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks. In *Conference on Empirical Methods in Natural Language Processing*, pages 254–263. Association for Computational Linguistics.

Sullivan, B., Wood, C. L., Iliff, M., Bonney, R., Fink, D., and Kelling, S. (2009). ebird: A citizen-based bird observation network in the biological sciences. *Biological conservation*, 142(10):2282–2292.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision.

Tanno, R., Saeedi, A., Sankaranarayanan, S., Alexander, D. C., and Silberman, N. (2019). Learning from noisy labels by regularized estimation of annotator confusion. *CoRR*, abs/1902.03680.

Telenius, A. (2011). Biodiversity information goes public: Gbif at your service. *Nordic Journal of Botany*, 29(3):378–381.

Tinati, R., Luczak-Roesch, M., Simperl, E., and Hall, W. (2017). An investigation of player motivations in eyewire, a gamified citizen science project. *Computers in Human Behavior*, 73:527–540.

Turland, N. J., Wiersema, J. H., Barrie, F. R., Greuter, W., Hawksworth, D. L., Herendeen, P. S., Knapp, S., Kusber, W.-H., Li, D.-Z., Marhold, K., May, T. W., McNeill, J., Monro, A. M., Prado, J., Price, M. J., and Smith, G. F. (2018). *International Code of Nomenclature for algae, fungi, and plants (Shenzhen Code) adopted by the Nineteenth International Botanical Congress Shenzhen, China, July 2017*, volume 159 of *Regnum Vegetabile*. Koeltz Botanical Books, Glashütten.

Van Horn, G., Mac Aodha, O., Song, Y., Cui, Y., Sun, C., Shepard, A., Adam, H., Perona, P., and Belongie, S. (2018). The inaturalist species classification and detection dataset. In *CVPR*, pages 8769–8778.

Vanschoren, J., van Rijn, J., Bischl, B., and Torgo, L. (2013). Openml: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60.

Vasudevan, V., Caine, B., Gontijo Lope, R., Fridovich-Keil, S., and Roelofs, R. (2022). When does dough become a bagel? analyzing the remaining mistakes on imagenet. *Advances in Neural Information Processing Systems*, 35.

Vidal, M., Wolf, N., Rosenberg, B., Harris, B. P., and Mathis, A. (2021). Perspectives on individual animal identification from biology and computer vision. *Integrative and comparative biology*, 61(3):900–916.

Von Ahn, L. and Dabbish, L. (2005). Esp: Labeling images with a computer game. In *AAAI spring symposium: Knowledge collection from volunteer contributors*, volume 2, page 1.

Vuurens, J., de Vries, A. P., and Eickhoff, C. (2011). How much spam can you take? an analysis of crowdsourcing results to increase accuracy. In *Proc. ACM SIGIR Workshop on Crowdsourcing for Information Retrieval (CIR'11)*, pages 21–26.

Waida, H., Wada, Y., Andéol, L., Nakagawa, T., Zhang, Y., and Kanamori, T. (2023). Towards understanding the mechanism of contrastive learning via similarity structure: A theoretical analysis.

Wang, H., Li, Z., Feng, L., and Zhang, W. (2022). Vim: Out-of-distribution with virtual-logit matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4921–4930.

Wang, W. and Zhou, Z. (2015). Crowdsourcing label quality: a theoretical analysis. *Sci. China Inf. Sci.*, 58(11):1–12.

Welinder, P., Branson, S., Belongie, S., and Perona, P. (2010). The Multidimensional Wisdom of Crowds. In *NIPS*.

Wen, Y., Jerfel, G., Muller, R., W. Dusenberry, M., Snoek, J., Lakshminarayanan, B., and Tran, D. (2021). Combining ensembles and data augmentation can harm your calibration. In *ICLR*.

Whitehill, J., Wu, T., Bergsma, J., Movellan, J., and Ruvolo, P. (2009). Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NeurIPS*, volume 22.

Wojcik, S., Hilgard, S., Judd, N., Mocanu, D., Ragain, S., Fallin Hunzaker, M. B., Coleman, K., and Baxter, J. (2022). Birdwatch: Crowd wisdom and bridging algorithms can inform understanding and reduce the spread of misinformation.

Wongon, K., Heonjun, Y., Guesuk, L., Taejin, K., and Byeng, D. (2020). A new calibration metric that considers statistical correlation: Marginal probability and correlation residuals. *Reliability Engineering & System Safety*, 195.

Wright, J., Palosanu, I.-L., Clift, L., Garcia Seco De Herrera, A., and Chamberlain, J. (2021). Pixelwise annotation of coral reef substrates. In *CEUR Workshop Proceedings*, volume 2936, pages 1394–1404. CEUR Workshop Proceedings.

Yang, F. and Koyejo, S. (2020). On the consistency of top-k surrogate losses. In *ICML*, pages 10727–10735.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2021). Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115.

Zhang, H., Cissé, M., Dauphin, Y. N., and Lopez-Paz, D. (2018). mixup: Beyond empirical risk minimization. In *ICLR*.

Zhong, A., Cui, J., Liu, S., and Jia, J. (2021). Improving calibration for long-tailed recognition. In *CVPR*, pages 16489–16498.

Abstract

While classification datasets are composed of more and more data, the need for human expertise to label them is still present. Crowdsourcing platforms are a way to gather expert feedback at a low cost. However, the quality of these labels is not always guaranteed. In this thesis, we focus on the problem of label ambiguity in crowdsourcing. Label ambiguity has mostly two sources: the worker's ability and the task's difficulty. We first present a new indicator, the WAUM (Weighted Area Under the Magin), to detect ambiguous tasks given to workers. Based on the existing AUM in the classical supervised setting, this lets us explore large datasets while focusing on tasks that might require more relevant expertise or should be discarded from the actual dataset. We then present a new open-source Python library, PeerAnnot, that we developed to handle crowdsourced datasets in image classification. We created a benchmark in the Benchopt library to evaluate our label aggregation strategies for more reproducible results. Finally, we present a case study on the Pl@ntNet dataset, where we evaluate the current state of the platform's label aggregation strategy and propose ways to improve it. This setting with a large number of tasks, experts and classes is highly challenging for current crowdsourcing aggregation strategies. We report consistently better performance against competitors and propose a new aggregation strategy that could be used in the future to improve the quality of the Pl@ntNet dataset. We also release this large dataset of expert feedback that could be used to improve the quality of the current aggregation methods and provide a new benchmark.

Abstract

Alors que les jeux de données de classification sont composés d'un nombre croissant de données, le besoin d'expertise humaine pour les étiqueter est toujours présent. Les plateformes de crowdsourcing sont un moyen de recueillir les commentaires d'experts à faible coût. Cependant, la qualité de ces étiquettes n'est pas toujours garantie. Dans cette thèse, nous nous concentrons sur le problème de l'ambiguïté des étiquettes dans le crowdsourcing. L'ambiguïté des étiquettes a principalement deux sources : la capacité du travailleur et la difficulté de la tâche. Nous présentons tout d'abord un nouvel indicateur, le WAUM (Weighted Area Under the Magin), pour détecter les tâches ambiguës confiées aux travailleurs. Basé sur le AUM existant dans le cadre supervisé classique, il nous permet d'explorer de grands jeux de données tout en nous concentrant sur les tâches qui pourraient nécessiter une expertise plus pertinente ou qui devraient être éliminées du jeu de données actuel. Nous présentons ensuite une nouvelle bibliothèque Python open-source, PeerAnnot, développée pour traiter les jeux de données crowdsourcées dans la classification d'images. Nous avons créé un benchmark dans la bibliothèque Benchopt pour évaluer nos stratégies d'agrégation d'étiquettes afin d'obtenir des résultats reproductibles facilement. Enfin, nous présentons une étude de cas sur l'ensemble de données Pl@ntNet, où nous évaluons l'état actuel de la stratégie d'agrégation d'étiquettes de la plateforme et proposons des moyens de l'améliorer. Ce contexte avec un grand nombre de tâches, d'experts et de classes est très difficile pour les stratégies d'agrégation de crowdsourcing actuelles. Nous faisons état de performances constamment supérieures à celles de nos concurrents et proposons une nouvelle stratégie d'agrégation qui pourrait être utilisée à l'avenir pour améliorer la qualité de l'ensemble de données Pl@ntNet. Nous publions également en plus de ce grand jeu de données, des annotations d'experts qui pourraientt être utilisées pour améliorer la qualité des méthodes d'agrégation actuelles et fournir un nouveau point de référence.