MisDetect: Iterative Mislabel Detection using Early Loss

Yuhao Deng
dyh18@bit.edu.cn
Beijing Institute of Technology

Chengliang Chai*
ccl@bit.edu.cn
Beijing Institute of Technology

Lei Cao
lcao@csail.mit.edu
University of Arizona/MIT

Nan Tang
nantang@hkust-gz.edu.cn
HKUST (GZ)

Jiayi Wang
jiayi-wa20@mails.tsinghua.edu.cn
Tsinghua University

Ju Fan
fanj@ruc.edu.cn
Renmin University of China

Ye Yuan
yuan-ye@bit.edu.cn
Beijing Institute of Technology

Guoren Wang
wanggrbit@126.com
Beijing Institute of Technology

ABSTRACT

Supervised machine learning (ML) models trained on data with mislabeled instances often produce inaccurate results due to label errors. Traditional methods of detecting mislabeled instances rely on data proximity, where an instance is considered mislabeled if its label is inconsistent with its neighbors. However, it often performs poorly, because an instance does not always share the same label with its neighbors. ML-based methods instead utilize trained models to differentiate between mislabeled and clean instances. However, these methods struggle to achieve high accuracy, since the models may have already overfitted mislabeled instances.

In this paper, we propose a novel framework, MisDetect, that detects mislabeled instances during model training. MisDetect leverages the early loss observation to *iteratively* identify and remove mislabeled instances. In this process, influence-based verification is applied to enhance the detection accuracy. Moreover, MisDetect automatically determines when the early loss is no longer effective in detecting mislabels such that the iterative detection process should terminate. Finally, for the training instances that MisDetect is still not certain about whether they are mislabeled or not, MisDetect automatically produces some pseudo labels to learn a binary classification model and leverages the generalization ability of the machine learning model to determine their status. Our experiments on 15 datasets show that MisDetect outperforms 10 baseline methods, demonstrating its effectiveness in detecting mislabeled instances.

PVLDB Reference Format:
Yuhao Deng, Chengliang Chai, Lei Cao, Nan Tang, Jiayi Wang, Ju Fan, Ye Yuan, and Guoren Wang. MisDetect: Iterative Mislabel Detection using Early Loss. PVLDB, 17(6): 1159 - 1172, 2024.
doi:10.14778/3648160.3648161

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/anonymous-repo1/misdetect.

*Chengliang Chai is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 6 ISSN 2150-8097.
doi:10.14778/3648160.3648161

1 INTRODUCTION

State-of-the-art supervised ML techniques, like deep learning, require a large number of accurately labeled data to achieve their full potential. This is especially crucial for mission-critical applications such as medical AI, which require millions of labels to train a robust and accurate model that ensures the safety of passengers or patients. To collect labels at this scale, they are often sourced from non-experts or obtained through web annotations, which can introduce errors and inaccuracies. Mislabeled instances in the training set can significantly degrade the model's performance and potentially endanger human lives. Therefore, there is an urgent need to effectively identify mislabeled instances in a training set with label errors, which is a critical data cleaning task.

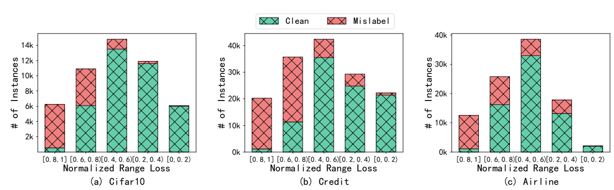

Figure 1: Normalized Loss for Clean/Mislabel Instances.

Existing Solutions. Traditional methods [12, 47] rely on the data proximity to detect mislabeled instances. For example, to determine whether an instance is mislabeled, the classical KNN method [12] checks its neighbors, based on the assumption that an instance should have consistent labels with its neighbors. Otherwise, it tends to be mislabeled. Other methods mainly use the output of the trained models to detect mislabeled instances. For instance, ensemble-based methods [15, 30, 60] train multiple models on the labeled dataset and consider an instance as mislabeled if the models are inconsistent on their predictions. However, models tend to disagree with each other even if the instances are correctly labeled. Cleanlab [40] is a state-of-the-art approach that utilizes confident learning to estimate the joint distribution of the entire dataset and leverages the learned distribution to distinguish clean and mislabeled instances. However, because it trains models with both correct and incorrect labels, this often makes the model overfit the dirty data, thus reducing its ability to distinguish between mislabeled and correctly labeled instances.

In this work, we target solving this fundamental problem in mislabel detection. The key idea is to *distinguish mislabeled and correctly labeled instances before the model starts overfitting the mislabeled instances, a.k.a. early detection.*

Key Observation. The loss values of clean training instances behave very differently from the mislabeled instances in early training phases. Specifically, mislabeled instances tend to incur higher losses than clean ones in *first few* training epochs. We thus call this the **early loss** observation. Early detection can be achieved if we come up with a way to effectively leverage this observation.

Empirical Verification. As shown in Figure 1, on each of the three real-world datasets, we train a multi-layer perceptron (MLP) model and collect the early loss of each instance. The x-axis denotes the range of normalized loss after training the first epoch, and the y-axis denotes the number of instances falling into the corresponding loss range. The mislabeled instances always have large losses, while the losses of the clean ones are relatively small. This verifies that SGD tends to preferentially fit clean instances; and at early training epochs, the training loss of clean instances tends to be smaller than that of mislabeled instances.

The high-level intuition of this observation is that clean instances always show more regular and clear patterns than mislabeled ones, and thus clean instances tend to dominate the average gradient which is used to update the model. Hence, the loss of clean instances decreases faster. Next, we use a toy example to better illustrate the loss difference between clean and mislabeled instances.

Example 1. *Consider a wine dataset with four features* `ProductID`*,* `Alcohol`*,* `Color`*,* `Capacity`*, and the label* `Type`*. Given two instances* o_1 : $(0001, 4\%, Golden, 500ml, Beer)$ *and* o_2 : $(0002, 4\%, Golden, 450ml, Liquor)$ *which have very similar features but different labels, suppose that* o_1 *has the correct label while* o_2 *is mislabeled. As long as the model learns the regular patterns that beer shows,* o_1 *will have a small loss. On the contrary, because* o_2 *has similar features to* o_1 *but a different label, it tends to incur a large loss on this model.*

Challenges. Although early loss potentially can be valuable in detecting mislabeled instances, to make it work, some major challenges have to be addressed. First, there is not a clear-cut between the loss of mislabeled instances and that of clean labels for the following reasons: (1) at the early training stages, many clean instances might have not been well fitted by the randomly initialized model, resulting in relatively large loss; (2) In some cases, clean but difficult training instances incur a large loss as well due to the irregularity of their features. Furthermore, when the model starts fitting the mislabels, early loss will no longer be effective in detecting mislabels. Clearly, relying on the users to manually set a proper cutoff is not practical, because it varies across different types of ML models trained on different training datasets. Ideally, an effective mislabel detection approach should automatically determine when to terminate the early loss-based detection process.

Our Proposal. We propose an iterative detection framework, called `MisDetect`, to address the above challenges. The key idea is to leverage early loss to iteratively identify the most obvious clean and mislabeled instances and then use them as pseudo labels to train a classification model that determines if the remaining uncertain instances are clean or not. In this way, we fully explore the early loss observation as well as the generalization ability of the machine learning model to effectively identify mislabels. In addition, `MisDetect` automatically terminates the detection process using an entropy-based mechanism. `MisDetect` consists of three modules:

Early Loss-based Iterative Detection. `MisDetect` iteratively identifies the instances with the largest loss as potential mislabeled instances and removes them from the training process. In this way, `MisDetect` mitigates the impact of the mislabels on the model training such that it can more effectively fit the clean instances, making early loss more effective in detecting mislabels. At the same time, `MisDetect` iteratively discovers the clean instances that have the smallest loss as pseudo labels to train the classification model.

Moreover, `MisDetect` monitors the distribution of the training loss during the iterative process and terminates it before the model begins to fit mislabeled data. To establish this *stop condition*, we leverage the observation that the entropy of the training loss effectively reflects the progress of the training.

Influence-based Verification. Meanwhile, in the iterative process, the influence-based verification module double-checks whether the mislabeled candidates discovered via early loss are indeed mislabeled or not. The observation is that a mislabeled instance tends to have a larger influence on the model's performance than a clean instance. We thus leverage this observation to separate difficult but clean instances from mislabeled instances.

Uncertain Instances Classification. Finally, `MisDetect` uses the clean and mislabeled instances discovered in the above iterative process to train a classifier that eventually determines the status (clean or mislabeled) of the remaining uncertain instances. To train an effective classification model, `MisDetect` augments the features of each instance using the information of its neighbors. Then an attention layer in this model converts the features into an embedding which is effective in classifying this instance.

Contributions. Our main contributions include:

(1) We develop a framework, called `MisDetect`, which iteratively detects mislabels during ML training, thus achieving high accuracy. (Sections 3)

(2) We use the training losses produced in early training stages as an effective signal to distinguish mislabeled instances from clean ones; and we invent a loss entropy-based method to decide when to terminate the iterative mislabel detection process, effectively preventing the model from overfitting mislabeled instances. Furthermore, we design a hybrid approach that effectively leverages the influence function to further improve the detection accuracy. (Section 4)

(3) We train a classification model (Section 5) to determine the status of the instances that `MisDetect` is still unsure about after the iterative detection process. Note `MisDetect` does not require humans to manually supply any labels to train this model.

(4) We conduct extensive experiments using 15 datasets and compared our `MisDetect` to 10 baselines (Section 6). Our results demonstrate the superiority of `MisDetect` over existing methods. Specifically, our approach achieves up to 31% (14% on average) higher F1-score compared to the state-of-the-art method Cleanlab, where the F1-score is measured by the detection of mislabeled instances with respect to the ground truth labels.

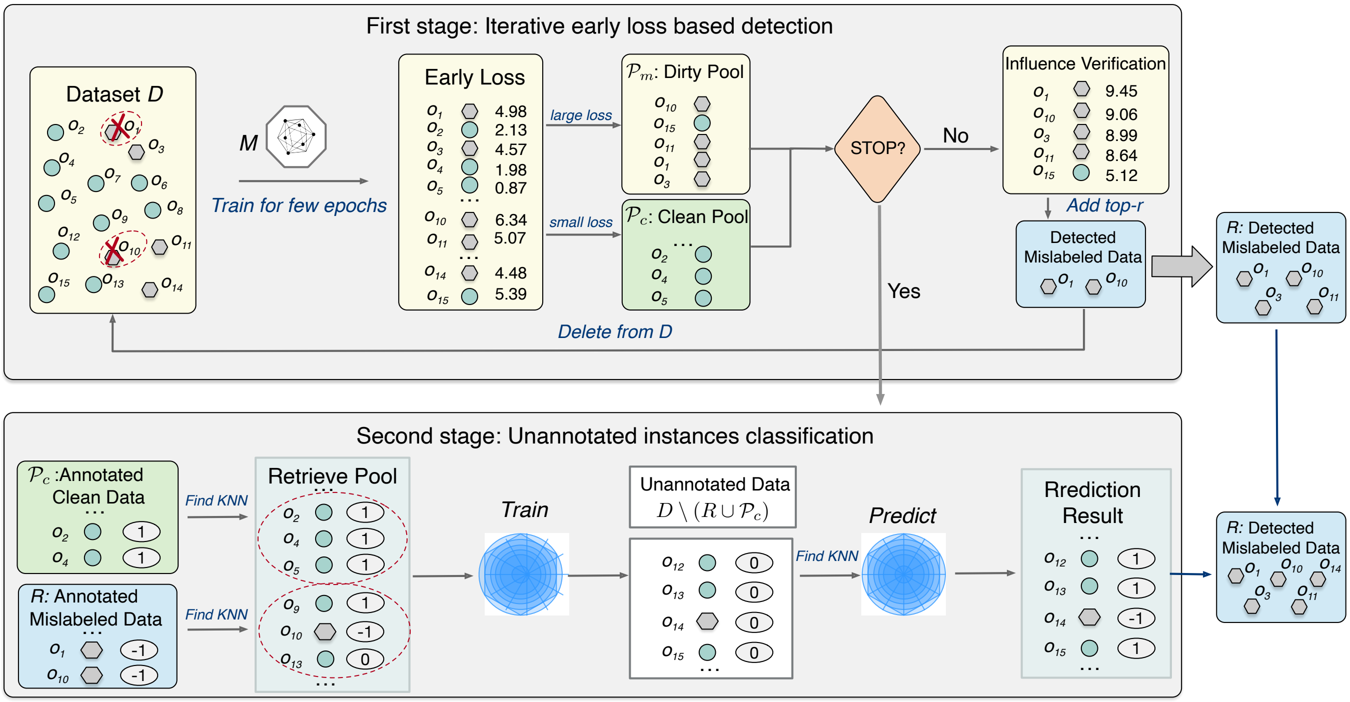

Figure 2: MisDetect Framework

2 MISLABEL DETECTION

Supervised ML. Without loss of generality, we consider a classification model $f : X \longrightarrow Y$ that maps an input x_i to a label y_i from Y. Given a training set $D = \{(x_i, y_i)\}$ $(i \in [1, N])$ with N training data instances, the **objective** of supervised ML training is to learn the optimal parameters that minimize the training loss, as:

$$w^* = \arg\min_{w \in \mathcal{W}} \mathcal{F}(w), \mathcal{F}(w) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(o_i, w) \tag{1}$$

where $\mathcal{W}$ represents the parameter space, and $\mathcal{L}(o_i, w)$ denotes the loss of the i-th training instance.

Note that we support multi-classification tasks. For ease of representation, we abbreviate $\mathcal{L}(o_i, w)$ as $\mathcal{L}(o_i)$.

Mislabels in supervised ML. We define the ground truth label for each training instance x_i as y_i^*. Based on this label, a training dataset D can be partitioned into two disjoint sets: D_c, containing all instances $o_j = (x_j, y_j)$ where $y_j = y_j^*$, and D_m, containing all other instances $o_k = (x_k, y_k)$ where $y_k \neq y_k^*$. Clearly, $D = D_c \cup D_m$; and D_c and D_m do not overlap with each other (*i.e.*, $D_c \cap D_m = \emptyset$).

In ML, labels play a critical role as a training set with accurate labels leads to a smoother training process and better models. Hence, it is crucial to identify mislabeled instances within the train set to prevent them from negatively impacting model performance.

Mislabel detection. The objective of mislabel detection is to identify all mislabeled instances in the training dataset D. The effectiveness of the mislabel detection algorithm is evaluated by comparing the set of detected mislabeled instances D'_m to the set of actually mislabeled instances D_m, using precision, recall, or F1-score.

In addition to improved model accuracy and reduced training cost, mislabel detection, which improves the quality of the training data, can benefit many downstream applications for different purposes.

Remark. Note that although we currently focus on discussing and evaluating the classification task, we believe that our proposed method can also be extended to the regression task for the following reason: we leverage the loss to detect mislabeled instances. Given a regression task, if the labels of the training set are incorrect, the early loss will also be large. In addition, training algorithms that do not use gradient descent (e.g., random forest) cannot be utilized to detect mislabels. Our method is automatic and no human is involved, and a test set is not needed.

3 THE MISDETECT FRAMEWORK

As discussed in Sec. 2, the goal of ML optimization is to learn the model parameters that minimize the loss of the training instances. Therefore, the loss the training instances incur during each training epoch contains valuable information reflecting the interaction between the model and the training data. Our fundamental idea is based on the observation w.r.t. training loss: correctly labeled and mislabeled instances exhibit distinct characteristics in training loss, particularly during the early stages of training (or epochs).

Building on this principle, our proposed framework, called `MisDetect`, is an iterative approach that leverages few-epoch training to iteratively detect mislabeled instances with high accuracy. In each iteration, `MisDetect` identifies mislabeled instances as well as correctly labeled instances with high confidence. It then pivots on the correctly labeled instances to detect additional mislabeled instances in the second stage. Next, we provide more details of the `MisDetect` framework.

3.1 Stage 1: Iterative Detection

Early loss-based candidates generation. As discussed in Section 1, mislabeled instances often exhibit a large early loss due to the model failing to fit them in the early stages of training. In light of this, as depicted in the top half of Figure 1, our framework iteratively identifies instances with the largest loss during the initial training epochs to form a dynamic dirty pool, denoted by $\mathcal{P}_m$. These instances are likely to be mislabeled.

Concurrently, we also maintain a clean pool $\mathcal{P}_c$ consisting of instances with the smallest loss. Because these instances are very likely to be correctly labeled, we use the model trained over $\mathcal{P}_c$ to double-check the instances in the dirty pool. Some instances in the dirty pool might in fact have clean labels due to their large loss. To achieve this, we leverage the concept of influence, which will be discussed more shortly.

In addition, `MisDetect` incorporates a mechanism to automatically determine whether the model M has started fitting the mislabeled instances such that the loss is no longer a reliable indicator of mislabel status. If this occurs, `MisDetect` will stop constructing the dirty and clean pools and proceed to the second stage, where a binary classification model is trained to determine the status of the uncertain instances. We will elaborate on this process in Section 5.

Influence-based verification. At each iteration, based on the parameter of the model trained over $\mathcal{P}_c$, we conduct an *influence* evaluation to refine the instances in the dirty pool. The intuition is that mislabeled instances tend to have a negative impact on the learned model [32, 43]. This negative impact is reflected on the update of the model's parameters when a mislabeled instance is added to this clean training set. Therefore, by measuring the parameter update of the learned model and hence the *influence* of each instance, we can verify whether an instance is truly mislabeled or not.

To measure this influence, a straightforward solution would be to add the instance to the training set and retrain the model. However, this would result in prohibitive training costs. To overcome this challenge, influence function [24, 32, 43] can be leveraged to measure the influence of an instance on a model without retraining. However, directly applying it to our context tend to be less effective.

This is because the influence function always assumes that the given instance has already been involved in the training of the model and estimates afterward how the model's performance will change if this instance is excluded. However, `MisDetect` aims to identify mislabels, and involving a number of mislabeled instances in training will inevitably compromise the model's performance. Furthermore, estimating the influence of an instance on a model with poor performance tends to produce misleading results.

To solve this problem, we introduce a variation of the influence function that estimates the influence more effectively in Section 4.2.

Overall, as shown in Figure 2, at each iteration, the r instances in $\mathcal{P}_m$ with the largest influence values will be added into the result set R, because they are more likely to be mislabeled. In addition, we consider instances that have been put into $\mathcal{P}_c$ as clean, because they have the smallest loss, while the model tends to fit clean instances at the beginning. Therefore, in the first stage, instances in R and $\mathcal{P}_c$ are annotated as mislabeled and clean correspondingly.

3.2 Stage 2: Unannotated Instances Classification

Besides the annotated instances in the first stage, there remain many instances ($D \setminus (R \cup \mathcal{P}_c)$) that are not annotated, *i.e.*, whether they are mislabeled or not is still uncertain. The main reason is that some instances may be located at the decision boundary of the model and thus have similar loss values. Therefore, it is hard to separate them at the first stage. Hence, we incorporate a classification model to further identify the rest instances, fully exploring its generalization ability.

As shown in the lower part of Figure 1, we use the instances in the current result set and clean pool as training data, with the mislabeled instances tagged as "-1" and the clean instances tagged as "1", to train a binary classification model. This model is used to determine the status of these uncertain instances and classify them as either clean or mislabeled. We will provide more details about this process in Section 5.

In addition to the raw features of these instances, we also incorporate the original class labels and the annotated tags ("-1", "1", or "0") of their K nearest neighbors (KNN) as features. This is because the labels of an instance's close neighbors can be helpful in verifying if it is mislabeled or not [12]. The classical KNN classification methodology shows that a correctly labeled instance tends to have a consistent label with its nearest neighbors. On the other hand, a mislabeled instance often has different labels with its KNN. Therefore, by incorporating the KNN labels as features, the binary classification model can learn to distinguish correctly labeled and mislabeled instances more accurately.

Note that at the first stage, `MisDetect` might not be aware yet if the retrieved neighbors are mislabeled or not. We thus annotate these uncertain instances with a tag "0".

At the inference phase, given an uncertain instance, we feed its enhanced features into the trained model to predict if it is indeed mislabeled (-1). The final output of `MisDetect` includes the mislabeled instances identified at both the first and second stages.

3.3 Algorithm: Putting Two Stages Together

Next, we use pseudocode (Algorithm 1) and Figure 2 to further illustrate the framework.

Stage 1 (lines 4-9). We first iteratively train the model over D and detect mislabeled instances, as shown in Figure 2. At each iteration, we obtain the loss of all instances, denoted by a set L (line 4), and then construct the two pools (line 5). Note that $\mathcal{P}_c$ is incrementally built at each iteration, generating a relatively small set $\mathcal{P}'_c$ of small loss instances, and unioning with the previous clean pool. Based on the model trained on $\mathcal{P}_c$, we evaluate the influence of each instance in $\mathcal{P}_m$ and select the top-r instances with the highest influence as mislabeled ones (line 7).

Example 2. *Figure 2 shows an example. In the current iteration, $\mathcal{P}_m$ consists of $o_{10}, o_{15}, o_{11}, o_1, o_3$. Then we double-check these instances by evaluating their influence on the model. Although o_{15} incurs a large loss, its influence is not large. Hence, o_{15} will not be considered mislabeled. Because o_1 and o_{10} show high influence,*

Algorithm 1: `MisDetect` Framework

Input: The original dataset D, model M, r (number of detected instances per iteration).
Output: The result set R of mislabeled instances.

1 $R = \emptyset$;
2 /* The first stage */
3 **while** *stop condition is not satisfied* **do**
4 $\quad L$ = `Train_few_Epochs`(D);
5 $\quad \mathcal{P}_m, \mathcal{P}'_c$ = `Pool_Generation`(L);
6 $\quad \mathcal{P}_c = \mathcal{P}_c \cup \mathcal{P}'_c$;
7 $\quad top\_r$=`Influence_Evaluation`($\mathcal{P}_m, \mathcal{P}_c, r$);
8 $\quad$ Add top_r into R;
9 $\quad$ Remove top_r from D;
10 /* The second stage */
11 R_K = `Retrieve_KNN`(R);
12 C_K = `Retrieve_KNN`($\mathcal{P}_c$);
13 M_K = `Train`($R_K \cup C_K$);
14 P_K = `Retrieve_KNN`($D \setminus (R \cup \mathcal{P}_c)$);
15 R' = `Mislabel_Predict`(M_K, P_K);
16 $R = R \cup R'$;
17 **return** R;

they are confirmed as mislabeled instances (line 8) and will be excluded from the training in the next iteration (line 9). This process repeats until meeting the stop condition. In this example, suppose that $r = 2$, and `MisDetect` *stops after two iterations, it will recognize $R = \{o_1, o_3, o_{10}, o_{11}\}$ as mislabeled after the first stage.*

In the second stage, we consider the instances in R as negative training instances, while the instances in the clean pool $\mathcal{P}_c$ are regarded as positive training instances.

Stage 2 (lines 11-18). For each of these training instances, we retrieve their KNN neighbors as additional features (lines 11-12).

EXAMPLE 3. *Consider the scenario where o_9 and o_{13} are the neighbors of o_{10} (suppose that $K = 2$). In this case,* `MisDetect` *combines these three instances to form a single training instance, where the features include the raw features of all three instances, their original class labels, and the tags assigned to them during the first stage. As o_{13} has not been identified in the first stage, it is assigned a tag of 0 which is also included as a feature.*

Then we train a model M_K (line 13) to predict if the remaining instances ($D \setminus (R \cup \mathcal{P}_c)$) are mislabeled. The instances predicted as -1 constitute R', which is then combined with R of the first stage as the final output (line 15-16). For example, because in the first stage, `MisDetect` is uncertain about o_{14}, `MisDetect` then retrieves the KNN of o_{14}, feeds the constructed feature into the model, and predicts it as mislabeled (*i.e.*, $R' = \{o_{14}\}$). The final output is $R = \{o_1, o_3, o_{10}, o_{11}, o_{14}\}$.

4 ITERATIVE MISLABEL DETECTION

In this section, we first show why mislabeled instances tend to have larger training losses than clean instances, and then introduce the details of our early loss-based mislabel detection method as well as the *stop condition*. Next, we present the influence-based verification to make the mislabel detection more robust. Finally, we prove the convergence of the proposed iterative method.

4.1 Early Loss-based Detection

Loss of an instance. Typically, training a neural network model starts with initializing the parameters randomly. Therefore, on the initial model, all instances tend to incur large losses. Then the neural network optimizes the parameters typically using the stochastic gradient descent (SGD) algorithm [13]. Next, we show how SGD impacts mislabeled data and clean data differently.

Formally, the gradient of an instance can be represented as:

$$d_i = \nabla_\theta \mathcal{L}(o_i) \tag{2}$$

Clearly, given an instance o_i, updating the model parameters in the direction of its negative gradient $-d_i$ would most effectively reduce its training loss.

Average gradient. However, SGD in fact updates the model parameters based on the average gradient ($-\overline{d}$) of an entire batch of instances, as shown in Eq. 3. Therefore, this update is not optimal with respect to each individual instance.

$$\overline{d} = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \mathcal{L}(o_i) \tag{3}$$

Loss reduction for an instance. Putting Eq. 2 and 3 together, whenever SGD updates the model parameters based on the average gradients, the impact that this update causes to a given instance o_i, namely the amount of loss reduction, can be measured by projecting $-\overline{d}$ onto $-d_i$.

$$E_i = -\overline{d} \cdot \frac{-d_i}{\| - d_i\|} = \|\overline{d}\| \cos(\overline{d}, d_i) \tag{4}$$

In Eq. 4, cos() denotes the cosine similarity between two vectors, while E_i denotes how much the loss of o_i will be reduced.

At a high level, supervised ML learns a mapping from features to labels. Intuitively, clean instances tend to show more converged and regular patterns in the mapping than mislabeled ones. Moreover, in real applications, the clean instances usually are the majority of the training data (*e.g.*, in CleanML [35] benchmark, all datasets have dirty data ratios no more than 40%). Hence, clean instances tend to dominate the direction of the average gradient, *i.e.*, the directions of clean instance gradients tend to be more aligned with that of the average gradient. As a result, given a clean instance o_c and a mislabeled instance o_m, very likely $\cos(\overline{d}, d_c) > \cos(\overline{d}, d_m)$, and thus $E_c > E_m$ (as shown in Figure 3), which indicates that gradient descent tends to be more effective in minimizing the loss of clean instances.

The Detection Algorithm. Next, we further discuss some details of our early loss-based method.

(1) Compute the loss of each instance: at each iteration, we train for few epochs, say 3 epochs, and derive the set of cross entropy loss $L = \{\mathcal{L}(o_i), i \in [1, N] | \mathcal{L}(o_i) = \sum_{c=1}^{M} y_{ic} log(p_{ic})\}$, where M denotes the number of classes, y_{ic} is an indicator that if $y_i^* = c$, then $y_{ic} = 1$, otherwise $y_{ic} = 0$, and p_{ic} denotes the probability of o_i belonging to c.

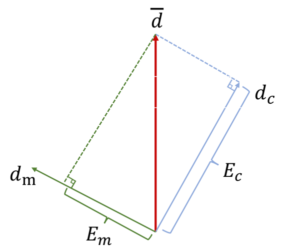

Figure 3: Gradient Projection.

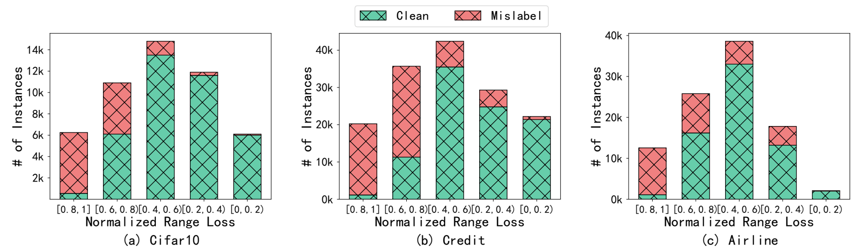

Figure 4: Normalized Range Loss for Clean/Mislabel Instances.

(2) Dynamically construct the dirty pool $\mathcal{P}_m$: at each iteration, we use the mean μ and the standard deviation σ of L to identify the mislabel candidates. More specifically, an instance is considered as potentially mislabeled if its loss is *larger* than $\mu + \sigma$. In this way, MisDetect avoids introducing a hyper-parameter to set the pool size and instead it automatically establishes a cutoff threshold.

(3) Maintain the clean pool $\mathcal{P}_c$: similar to $\mathcal{P}_m$, we consider an instance as clean if its loss is *smaller* than $\mu - \sigma$. We iteratively expand $\mathcal{P}_c$ by unioning the clean instances discovered in the current epoch ($\mathcal{P}_c'$). We observe that the loss of these instances does not change significantly. Therefore, $\mathcal{P}_c$ tends to be stable.

Stop condition. After the model has fitted most of the clean instances and starts fitting the mislabeled ones, the early loss will no longer be effective in detecting mislabels. Therefore, the early loss-based mislabel detection stage, namely the first stage, has to be stopped immediately.

One intuitive way to stop the detection would be to introduce hyper-parameters. That is, we ask the users to determine after how many epochs the early loss-based detection should stop. However, it will be a hyper-parameter that is hard for the users to set appropriately, because it largely relies on the characteristics of the data sets as well as the model architecture.

In this work, we propose an *entropy*-based method to automatically determine the stop condition. The key idea is to continuously test the entropy of the training loss in each epoch and stop when the entropy reaches the lowest value.

The intuition is that entropy effectively reflects the distribution of the loss. When the model is still fitting the clean instances, their loss decreases rapidly. On the other hand, the loss of other instances remains large. The distribution of the loss thus tends to be rough and changes dramatically in this process. Accordingly, its entropy will keep decreasing when the model is fitting the clean instances. However, after the model begins to fit mislabeled instances, the loss distribution will become flatter and flatter, resulting in an increase in entropy. This observation inspires us with a solution to automatically stop the early loss-based cleaning process. That is, when the entropy reaches a turning point, we should stop using early loss to detect mislabels.

More specifically, we compute the entropy of the loss distribution as $\sum_{i=1}^{N} q_i \log q_i$, where $q_i = \frac{\mathcal{L}(o_i)}{\sum_{i=1}^{N} \mathcal{L}(o_i)}$. Figure 5 shows how the loss entropy varies as the number of epochs increases (on MNIST dataset). In general, the entropy decreases first and then increases.

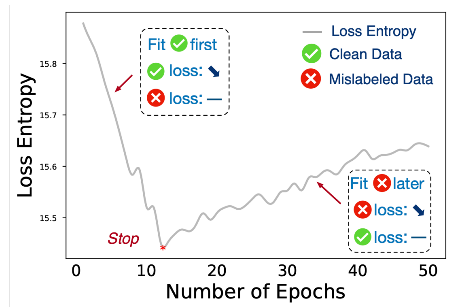

Figure 5: Illustration of the Stop Condition.

In this work, we stop the early loss-based identification process after the entropy has increased for three successive epochs.

4.2 Influence-based Verification

Although early loss is effective in identifying mislabeled instances, it tends to introduce false positives. In particular, we observe that some clean instances also get large training loss and hence are erroneously detected as mislabels. In these cases, the model has not yet fitted these instances well in the early epochs due to various reasons, for example, the irregularity of their features or lack of similar training examples. To this end, we propose an *influence*-based method that further improves the detection performance.

The intuition is that mislabeled instances tend to impact the modeling in a negative way [32]. Therefore, if we can effectively measure the influence of each individual training instance on a trained model, we will be able to purify the mislabeled candidates in the dirty pool $\mathcal{P}_m$ by excluding the instances that have a small influence on the model.

Given a training instance o, we propose to measure its influence based on how large it would make the model parameter deviate from the parameter learned from a purely clean dataset if o was used in the training process.

More specifically, given a noisy training set D, suppose there is a perfect model parameterized with w_c^*, which is learned from the clean instances in D denoted as D_c, where all mislabeled instances are excluded, then the influence of instance o can be measured by $f(o) = \|w_o - w_c^*\|$, where w_o is the parameter learned over $D_c \cup \{o\}$.

However, to make this solution effective, we have two problems. First, we do not know w_c^* and D_c beforehand. Otherwise, we would not have the mislabel identification problem to solve. Second, even if

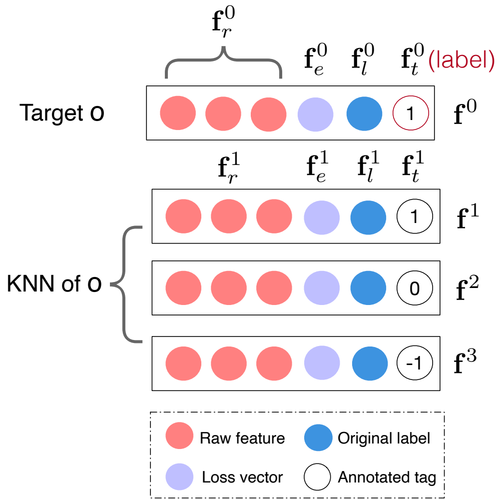

Figure 6: Extracted Features of a Training Instance.

we know w_c^* apriori, how can we obtain w_o efficiently? Repeatedly retraining the model over $D_c \cup \{o\}$ would be prohibitively expensive.

w_c for influence evaluation. For the first problem, we propose to use w_c of the model M_c to replace w_c^*, where M_c is learned from the clean set $\mathcal{P}_c$ produced in the early loss-based mislabel detection stage. That is, we use the objects in $\mathcal{P}_c$ as the clean set D_c to learn w_c. Because $\mathcal{P}_c$ tends to be clean and contains no or at most a few mislabeled instances, adding a mislabeled instance o into the training process potentially will largely alter w_c and in turn produce a large influence $f(o)$.

Based on the above analysis, we propose to use Equation 5 to measure the influence of each object and identify the mislabel:

$$o^* = \arg\max_{o \in \mathcal{P}_m} \| w_o^+ - w_c \| \tag{5}$$

where w_o^+ is the parameter of the model learned over $\mathcal{P}_c \cup \{o\}$. By Equation 5, an object in the dirty pool will be indeed identified as mislabeled if it has the largest influence.

Efficiently Compute the Influence. For the second problem, rather than computing the influence $f(o)$ by first obtaining w_o^+ through retraining, `MisDetect` directly estimates the $f(o)$ based on the model parameter w_c learned over the clean pool. Based on the derivation in [32], we get $f(o) = \| w_o^+ - w_c \| = \| \frac{\nabla \mathcal{L}(o, w_c)}{N_c \mathcal{H}(w_c)} \|$, where $\mathcal{L}(o, w_c)$ represents the cross entropy loss function of o, and $\mathcal{H}(w_c) \overset{def}{=} \frac{1}{N} \sum_{i=1}^{N} \nabla \mathcal{L}(o_i, w_c)$ represents the hessian matrix. N_c is the number of instances in $\mathcal{P}_c$. The computation of the Hessian matrix can also be accelerated using the method proposed in [42].

Note unlike the classical method [32] which tests the influence of one training instance that already exists in the training set used to train the given model, our approach estimates the influence of a training instance o not shown in the training process of the model. This is because a model that has not seen o tends to be more sensitive to o.

5 UNANNOTATED INSTANCES CLASSIFICATION

Next, we propose to learn a machine learning model M_K to determine the status of the rest unannotated instances, namely the instances that `MisDetect` is not sure whether they are clean or mislabeled in the first stage.

Our approach does not rely on humans to supply any annotated data. Instead, it uses the clean and mislabeled instances that `MisDetect` has already recognized as the supervised training data to train a binary classification model. `MisDetect` then leverages the generalization ability of the machine learning model to infer the status of uncertain instances.

More specifically, we denote this automatically generated training set as T. For each instance $o \in T$, it is tagged as -1 or 1, indicating it is clean or mislabeled. The tag corresponds to the ground truth label used to train the model.

Now to train a classification model, the only missing piece is the features of each instance o. Next, we discuss in detail how to construct features that are effective in distinguishing clean and mislabeled instances. We then demonstrate the architecture of our model in Sec. 5.2.

5.1 Feature Extraction

In addition to the raw feature of each instance o, we use the information of its neighbors as an enhancement.

Intuitively, to check whether an instance o is mislabeled, its neighbors could play an important role. This is because a clean instance tends to share the same label with its neighbors because of their similar features. Naturally, if an instance o has inconsistent labels with its neighbors such as its KNN, it is suspicious and thus might be mislabeled. Inspired by this data locality observation, given an instance o, we propose to encode the information of its KNN into its features to better classify mislabels.

We denote each instance $o \in D$ as (x, y, t), where x is the feature, y is its class label in the original classification task, and $t \in \{-1, 0, 1\}$ is its tag automatically annotated by `MisDetect`.

Retrieve KNN. Given an instance o, we first retrieve its KNN from the whole dataset D. Each of its KNN thus can be either an instance in the training set T or an uncertain instance in T', where $T' = D \backslash T$. For each uncertain object, because we are still unsure if it is clean or mislabeled, its tag t will be 0, indicating its uncertain status.

Feature Encoding. We use $knn(o)$ to denote the KNN of o (including o itself). For each $knn(o), o \in T$, we have four types of features to encode, as shown in Figure 6.

- The red circles: The raw features of instances in $knn(o)$, *i.e.*, $\mathbf{f}_r = \{x | (x, y, t) \in knn(o)\}$.
- The purple circles: The vectors of early loss in early three iterations, *i.e.*, $\mathbf{f}_e = \{[\mathcal{L}_1(o), \mathcal{L}_2(o), \mathcal{L}_3(o)] | o \in knn(o)\}$, where $\mathcal{L}_1(o)$ denotes the loss of instance o at the first iteration.
- The blue circles: The original labels of instances in $knn(o)$, *i.e.*, $\mathbf{f}_l = \{y | (x, y, t) \in knn(o)\}$.
- The white circles: The tags of instances in $knn(o)$, *i.e.*, $\mathbf{f}_t = \{t | (x, y, t) \in knn(o)\}$.

We use $\mathbf{f}_r^j, j \in [1, K]$ to denote the raw feature of j-th neighbor of o. Besides, when $j = 0$, $\mathbf{f}_r^0 = x$. This denotation rule applies equally

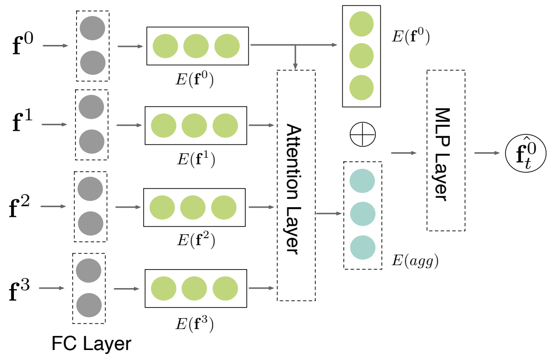

Figure 7: Model Architecture.

to $\mathbf{f}_e^j, \mathbf{f}_l^j, \mathbf{f}_t^j, j \in [1, K]$. Note that $\mathbf{f}_t^0$ is regarded as the label of an instance, and others will be taken as the features. More specifically, to encode each $o_j \in knn(o)$, we concatenate the four types of features and get $\mathbf{f}^j = \mathbf{f}_r^j \oplus \mathbf{f}_e^j \oplus \mathbf{f}_l^j \oplus \mathbf{f}_t^j, j \in [1, K]$, which will be fed into the model for training. Note that since $\mathbf{f}_t^0$ is the label, $\mathbf{f}^0 = \mathbf{f}_r^0 \oplus \mathbf{f}_e^0 \oplus \mathbf{f}_l^0$.

5.2 Model Architecture

Figure 7 shows the model architecture. It is composed of a fully-connected (FC) layer, an attention layer, and an MLP layer. The fully-connected layer converts each $\mathbf{f}^j$ to a $d-$dimension vector, denoted by $E(\mathbf{f}^j)$. Then the attention mechanism (Equation 6) is applied among these instances, which guides the model to concentrate on some neighbors that matter more to the classification task. For example, the model is likely to focus more on the instance with similar important features to the target instance to predict. Formally, we have:

$$\rho_j = \frac{e^{E(\mathbf{f}^j)^T W E(\mathbf{f}^0)}}{\sum_{v=1}^{K} e^{E(\mathbf{f}^v)^T W E(\mathbf{f}^0)}}, j \in [1, K] \tag{6}$$

where W is the parameter matrix of the attention layer. Afterwards, we obtain the aggregated feature $E(agg) = \sum_{j=1}^{K} \rho_j E(\mathbf{f}^j)$. Then we concatenate the embedding of o and the aggregated feature. Using this aggregated feature as input, the MLP layer produces the final prediction as $\hat{\mathbf{f}}_t^0 = \Sigma(\mathsf{MLP}([E(\mathbf{f}^0), E(agg)]))$, where Σ is the sigmoid function. We use the cross entropy loss for this binary prediction. Overall, the optimization goal of M_K with parameter θ can be formulated as:

$$\theta^* = \arg\min_{\theta} \frac{1}{|T|} \sum_{o=(x,y,t)\in T} \mathcal{L}(\hat{\mathbf{f}}_t^0 = f_\theta(o, knn(o)), \mathbf{f}_t^0) \tag{7}$$

6 EXPERIMENTS

In the experiments, we compare `MisDetect` against the state-of-the-art on the precision, recall and F1-score of mislabel detection, evaluate the key modules (influence evaluation, classification model, stop condition) of `MisDetect`, and conduct some other experiments.

6.1 Experimental Settings

Datasets. We evaluate our approach on 15 real-world image and tabular datasets from diverse domains. The size of the datasets varies from the magnitude of 10^2 to 10^6. The number of classes in each dataset ranges from 2 to 100. Among the 15 datasets, 3 (`USCensus`, `Credit`, `EEG`) of them are used by CleanML [35], which is a benchmark of data cleaning for ML. For the other 12 datasets, following the existing works like [25, 29, 45, 59], we inject synthetic mislabels with two methods, *i.e.*, random injection and equal injection. More specifically, given an expected proportion (say 20%) of mislabeled instances, random injection randomly selects 20% instances from the dataset and flips each of them to a random label different from the ground truth. Equal injection instead flips the same number of instances in each class. For example, on the `MNIST` dataset, because it has ten classes, we randomly flip $2\% \times N$ of instance in each class. In the experiments, we focus on random injection, while showing that our approach works well on both types of synthetic mislabels in Section 6.4. We randomly pick the proportion of mislabeled instances from {5%, 10%, 20%, 30%, 40%}. We do not consider a proportion larger than 40%, because it is rare in real applications [35]. Table 1 shows the statistics of the datasets.

Baselines.We compare our approach against 10 baselines, including existing works and the variants of our own approach:

(1) `K-Nearest Neighbor` [12] (`KNN`). Given an instance, if it has the same label with the majority of its K nearest neighbors, it is considered to be clean. Otherwise, it is mislabeled. We vary K from 1 to 30 and report the best result.

(2) `Ensemble-based method via majority vote` [15] (`E-MV`). It ensembles multiple independent classifiers with a majority vote. An instance will be marked as mislabeled if the prediction is different from its label.

(3) `Forgetting Events` [48] (`F-E`) identifies mislabeled instances if their prediction results vary frequently during training.

(4) `Clean Pool` uses $\mathcal{P}_c$ to train a classification model and then predicts each instance in D. An instance will be considered as mislabeled if the prediction is inconsistent with its label.

(5) `MentorNet` [29]. It is a reweighting-based robust learning method. The key idea is to use MentorNet to produce a smaller weight for potentially mislabeled instances and a higher weight for the clean. We train the robust model over D and then mark the misclassified training instances as mislabeled.

(6) `Co-teaching` [25]. As discussed in the related work section, `Co-teaching` is a classical robust learning method. Similar to `MentorNet`, we use the robust model trained by `Co-teaching` to detect mislabels as the training instances misclassified by the model.

(7) `Cleanlab` [40] uses confident learning to distinguish mislabeled instances and clean ones, which implements a Python library to detect mislabels. It takes as input D as well as an ML model. For the model, we use the same type as ours for a fair comparison.

(8) `Non-iter` is a baseline that trains only one iteration and then uses early loss to detect mislabels.

(9) `MisDetect Without Influence and Classification Model` (`M-W-IM`) is a variation of our method that only uses early loss to detect mislabels, while disables the influence-based verification and classification model.

(10) `MisDetect Without Classification Model`(`M-W-M`) is another variation that uses early loss and influence-based verification while disabling the classification model.

Table 1: Statistics of Datasets.

Dataset	#-Items	#-Attributes	#-Classes	Mislabel Ratio	Classification Task
USCensus [35]	32,561	14	2	5%	*If an adult earns more than $ 50,000.*
Wine [5]	6,497	12	7	20%	*Different types of wine quality.*
Credit [35]	150,000	10	2	16.55%	*If a client will experience financial distress.*
Mobile-Price [6]	2,000	20	4	30%	*The price range of a mobile.*
Airline [7]	103,905	24	2	40%	*The airline satisfaction level.*
SVHN [4]	630,420	$3 \times 32 \times 32$	10	10%	*Different street view house numbers.*
MNIST [2]	70,000	$1 \times 28 \times 28$	10	10%	*Different handwritten numbers.*
EEG [35]	14,980	14	2	5%	*If an eye-state is closed or open.*
CIFAR-10 [3]	60,000	$3 \times 32 \times 32$	10	20%	*Different universal objects.*
CIFAR-100 [3]	60,000	$3 \times 32 \times 32$	100	20%	*Different universal objects.*
Heart [8]	919	11	2	30%	*If a patient has heart disease.*
Hotel [9]	36,276	18	2	30%	*If a hotel booking status is canceled or not.*
KMNIST [10]	70,000	$1 \times 28 \times 28$	10	40%	*Different types of Japanese cursive scripts.*
Fashion-MNIST [11]	70,000	$1 \times 28 \times 28$	10	10%	*Different types of products.*
CoverType [1]	581,013	54	7	40%	*Different forest cover types.*

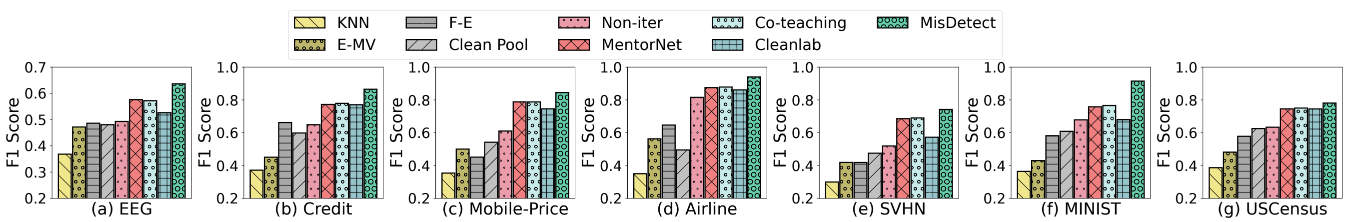

Figure 8: F1-score Comparison of Baselines.

Table 2: F1-score Comparison of Baselines for Other Datasets.

	CIFAR-10	CIFAR-100	Heart	Hotel	KMNIST	Wine	Fashion-MNIST	CoverType
KNN	0.3548	0.3166	0.4518	0.4616	0.3980	0.3887	0.4096	0.5039
E-MV	0.3213	0.3753	0.5679	0.6123	0.4566	0.4611	0.4207	0.6704
F-E	0.4141	0.3619	0.4257	0.5391	0.4881	0.5626	0.4121	0.6411
Clean Pool	0.5049	0.5031	0.5661	0.6163	0.5187	0.6156	0.4778	0.7485
Non-iter	0.6166	0.5110	0.5800	0.6213	0.6150	0.6464	0.4898	0.7082
MentorNet	0.7842	0.6091	0.7211	0.6581	0.7405	0.6798	0.6135	0.7333
Co-teaching	0.7920	0.6101	0.7354	0.6629	0.7579	0.6812	0.6365	0.7465
Cleanlab	0.7395	0.4835	0.6059	0.6233	0.6866	0.6095	0.5011	0.6818
MisDetect	**0.8622**	**0.7942**	**0.8000**	**0.6932**	**0.8008**	**0.7748**	**0.6844**	**0.8096**

Hyper-parameter Setting. We set each iteration to include 3 epochs. For our classification model, we use Adam optimizer [31]. The learning rate is set to 0.002. We use a fully-connected layer with a length of 256, followed by a 3-layer perceptron.

Although we have a well-designed stopping mechanism, it is still important to roughly know the proportion of mislabeled instances in a given dataset apriori. Otherwise, it is hard to decide how many instances to be annotated and removed from D at each iteration, *i.e.*, the parameter r. When the proportion is high, we should set r to be relatively large. Otherwise, we will only be able to discover a small number of mislabeled instances at the first stage. On the other hand, when the proportion is small, we should set r to be small to avoid misclassifying clean instances as mislabeled. However, rather than assume we are aware of the exact fraction, we only know it falls into some range, *i.e.*, (0, 10%], (10%, 30%], (30%, 40%]. The range can be estimated by for example asking experts to annotate a small sample of D. On these datasets, we empirically observe that on average, the model begins to fit mislabeled instances after 5 iterations (15 epochs). Hence, given a range, *e.g.*, (10%, 30%], we set $r = \frac{10\%+30\%}{2}/5 \times N$, which roughly ensures that we can discover enough mislabeled instances before dirty data starts getting fitted, while the stop condition designed by us determines the exact termination point.

Evaluation Metrics. We focus on evaluating the effectiveness of our approach. So we take precision, recall, and F1-score as metrics. Denote the set of mislabeled data that we detect correctly as D_T. The precision is computed by $pre = \frac{|D_T|}{|R|}$, the recall is computed by $recall = \frac{|D_T|}{|D_m|}$, while the F1-score is $F1 = \frac{2 \times pre \times recall}{pre + recall}$.
Environment. All experiments were implemented in Python, performed on a Ubuntu Server with an Intel (R) Xeon (R) Silver 4110 2.10GHz CPU having 32 cores, a Nvidia Geforce 2080ti GPU, and 128GB DDR4 main memory without SSD.

6.2 Comparison with Baselines

For baselines (1)-(8), because they directly output the detection results in a non-iterative way, we evaluate the final F1-score, recall, and precision. Baselines (9)-(11) are iterative. Therefore, we show their F1-score produced in each iteration. For 7 datasets, we display the F1-score, recall, and precision with figures, each of which corresponds to one metric w.r.t. one dataset. Due to the space constraint, for other datasets, we only show the F-1 score in Table 2.

6.2.1 Comparison with non-iterative baselines. As shown in Figure 8, `MisDetect` outperforms all the baselines in terms of F1-score. For example, the KNN-based method only achieves an F1-score less than 40% on all datasets, mainly because it is not informative enough to just rely on labels of near neighbors to identify mislabels. The ensemble-based solution performs poorly as well (around 50% F1-score), mainly because it trains over both mislabeled and clean instances and thus the trained model is not accurate enough.

`MisDetect` outperforms `F-E`, indicating that it is not sufficient to purely rely on the fluctuation of prediction results to detect mislabels. `Clean Pool` does not perform well, because it only uses a small fraction of clean instances to train the model, which is representative enough. `Non-iter` does not achieve good performance because the initial model it relies on is not effective enough to distinguish mislabeled and clean instances.

`MisDetect` also outperforms robust learning-based methods. For example, on dataset `Credit`, `MisDetect` achieves 0.87 F1-score, while `MentorNet` and `Co-teaching` are 0.77 and 0.78 respectively. This is because in many cases the robust machine learning models still tend to overfit some mislabeled instances, thus failing to separate them from clean labels. Finally, we compare against the state-of-the-art mislabel detection method `Cleanlab`. `MisDetect` significantly outperforms it. For example, on `EEG` dataset, our F1-score is 0.64 while `Cleanlab` is 0.53. This is because `Cleanlab` learns the distribution from the data already contaminated by mislabels, which is often not effective in separating mislabels from clean labels. `MisDetect` instead continuously monitors and analyzes the loss and influence in the iterative training process and iteratively removes mislabeled instances to mitigate their impact.

For precision and recall, as shown in Figures 10 and 9, we outperform all baselines on almost all datasets. However, the precision of `Co-teaching` (the state-of-the-art method in robust machine learning) is a little higher than ours. This is because the training mechanism of `Co-teaching` ensures that the training instances that cannot fit well are usually mislabeled. However, its recall is low, because it tends to perfectly fit some mislabeled instances and erroneously consider them as clean instances. Therefore, overall, our method outperforms all baselines clearly.

6.2.2 Comparison with iterative baselines. In this set of experiments, we evaluate how our influence-based verification and unannotated instance classification techniques work. Figure 11 displays the F-1 score in each iterative. The X-axis denotes the number of epochs during training. On all datasets, we show the first three iterations (each iteration contains 3 epochs). For ease of presentation, we only show the F1-score when the stop condition is met because different datasets may stop at different epochs. As we can see, as the number of epochs increases, the F1-score of all iterative methods continuously increases until the stop condition is achieved. This is mainly because the recall keeps increasing. For example, on dataset `USCensus`, during the iterative process, `MisDetect` outperforms `M-W-IM` by about 3%, because the influence-based verification effectively reduces the false positive rate. Because the classification model is only applied at the last iteration, `MisDetect` shows the same performance with `M-W-M` in the earlier iterations. `MisDetect` has a 7.6% higher F1-score than `M-W-M` at the last iteration, confirming that the classification model indeed is effective in annotating the instances that `MisDetect` is unsure about in the first stage.

6.3 Ablation Studies

6.3.1 K in KNN. We also evaluate how the number of neighbors (K) influences the classification model. In Figure 12, when K is relatively small, the F1 score is relatively low. This is because the retrieved K neighbors are not sufficient in representing the neighborhood of a given instance. On the other hand, if K is too large, the performance also degrades because in this case some of the retrieved KNN are not similar enough to the given instance. Empirically, we observe that setting $K = 20$ leads to good performance.

6.3.2 Influence-based Verification. Finally, we evaluate the effectiveness of influence-based verification. We compare against the baseline `Deletion` which trains a model over D and then uses the influence function to evaluate the influence of an instance by deleting it from D. As shown in Figure 13, `MisDetect` achieves a higher F1-score. This is because we instead evaluate the influence on a model trained with a clean training set that has not seen the to-be-evaluated instance and thus not been contaminated by it.

6.4 Mislabel Injection Evaluation

6.4.1 Mislabel Ratio. First, we evaluate how `MisDetect` performs when the proportion of mislabeled data varies within the range [5%,40%]. In Figure 14, the performance is not sensitive to mislabel fraction. For example, on dataset `MNIST`, as the fraction of mislabeled data increases, the F1-score does not change much (around 88%). The reason is that even when the mislabel ratio increases, the clean instances might still dominate the direction of the average gradient, because the gradients of the mislabeled instances are more disparate in direction. So our method still works.

6.4.2 Mislabel Distribution. We test two mislabel injection methods mentioned in Section 6.1. The results (Figure 15) on 3 datasets

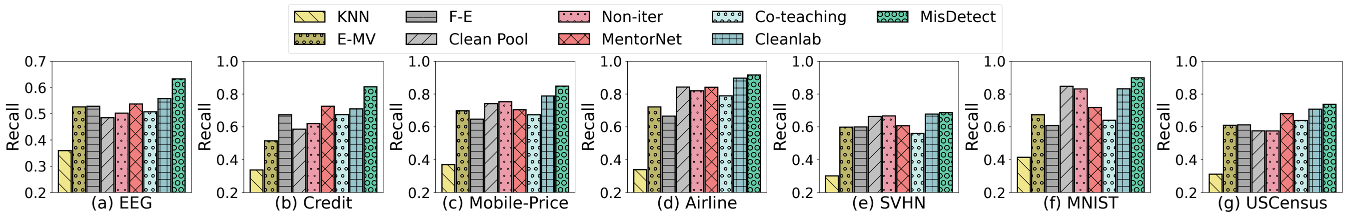

Figure 9: Recall Comparison of Baselines.

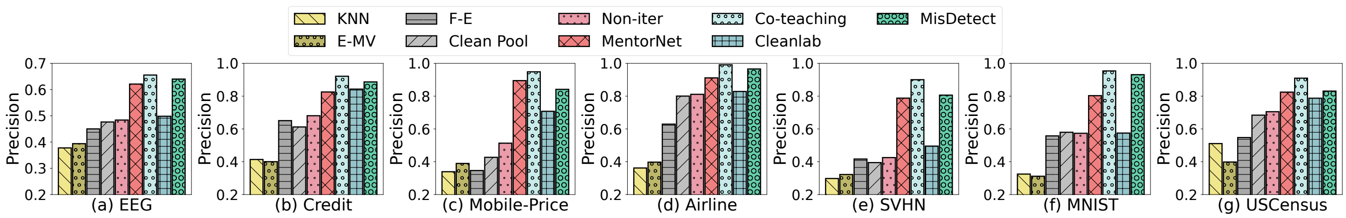

Figure 10: Precision Comparison of Baselines.

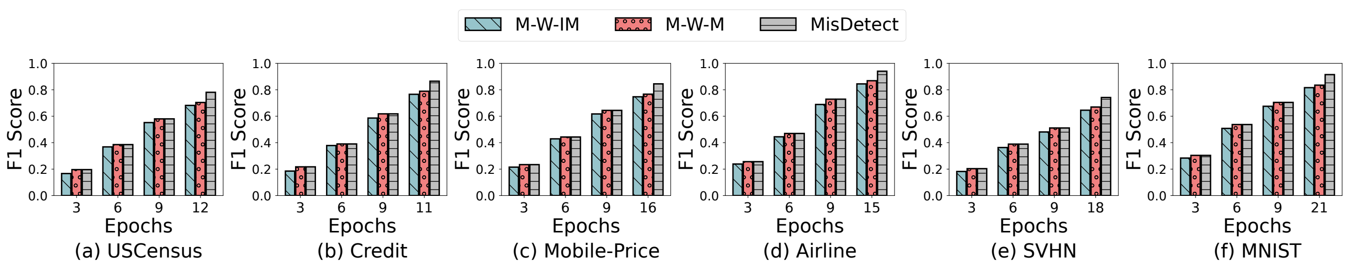

Figure 11: F1-score Comparison for Iterative Baselines.

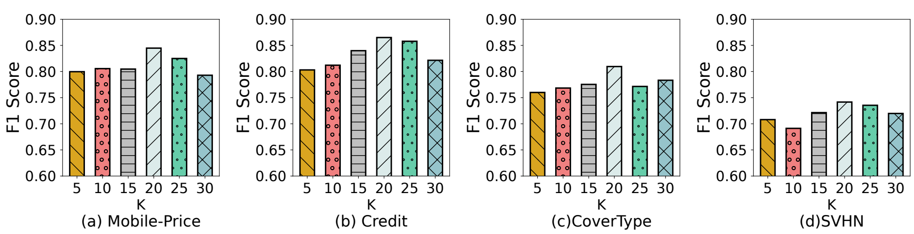

Figure 12: F1-score of Different K for Classification Model.

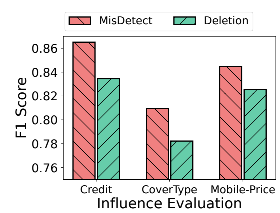

Figure 13: Influence Eval.

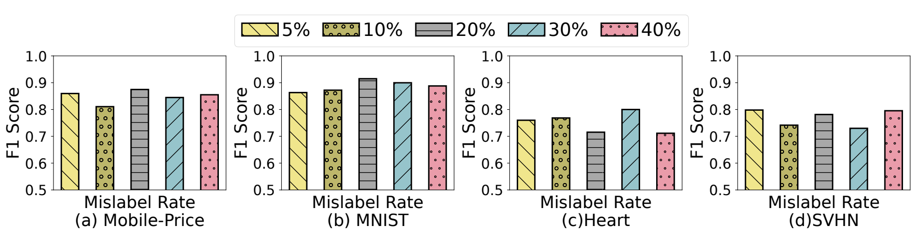

Figure 14: Varying Mislabel Rate.

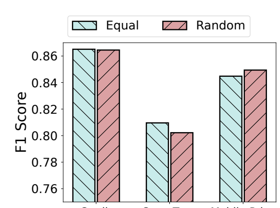

Figure 15: Injection Methods.

show that our method is robust to different injection methods because the first stage is able to identify accurate mislabel annotations, which provides high-quality labels to the second stage for further verification.

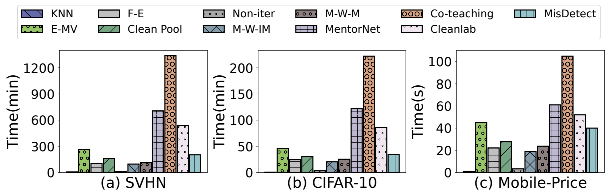

Figure 16: Efficiency Evaluation.

6.5 Efficiency Evaluation

As shown in Figure 16, traditional methods (KNN and NCN) are the most efficient. However, they cannot achieve high accuracy, which is our main target. Ensemble-based methods are not efficient because they have to train multiple ML models. The SOTA approaches like `Cleanlab` and `Co-teaching` are slow because they have to learn the distribution over the noisy training data, which takes many iterations to converge. Our method is more efficient and effective than `Cleanlab` and `Co-teaching`.

7 RELATED WORK

Traditional mislabel detection methods. KNN-based methods [12, 47] determine whether an instance is mislabeled or not based on its KNN neighbors. Badenas et al. [12] consider an instance as clean if it has a consistent label with the majority of its K nearest neighbors (KNN). Otherwise, it is mislabeled. Sánchez et al. [47] adopt the similar idea. However, they find the KNN and make sure that they are distributed diversely around the given instance. This ensures that the KNN are not too similar to each other or even redundant. Valizadegan et al. [49] model the mislabel detection problem as an optimization problem and use the kernel-based method to solve it. But it only targets binary classification datasets.
Mislabel detection using ML. Ensemble-based solutions [15, 30, 60] leverage the key idea that different classifiers tend to produce conflicting predictions on a mislabeled instance. Toneva et al. [48] assume that the prediction of a mislabeled instance tends to fluctuate greatly during training and detect mislabeled instances accordingly. Zhang et al. [57] propose to discover a small set of mislabeled instances and clean them such that the performance of a validation set can be improved most. Different from us, this method does not aim to detect all mislabeled instances and it needs a validation set. Cleanlab [40] is the state-of-the-art method for mislabel detection, which takes the dataset and an ML model as input, utilizing confident learning [22, 23] to estimate the joint distribution of all training instances based on the trained model. A mislabeled instance is then identified if it deviates from this distribution.
Robust ML. Robust ML aims to learn a well-performed model from a noisy training dataset, which can be classified into 4 categories: (1) Using models that are known to be robust to polluted data such as random forest and some ensemble classifiers [14, 39, 46]. (2) Reweighting techniques [29, 45, 51, 56] that aim to weigh the instances based on their cleanness. Small weights will be assigned to the potentially noisy labels to mitigate their impact on the model. (3) Robust loss function [28, 41, 52, 58], which makes ML models noise-tolerant. Zhang et al. [58] combine mean absolute loss and the cross entropy loss. Loss correction approaches [28, 41, 52] use instance predictions to estimate the noise transition matrix to optimize the loss function. (4) Co-teaching [25, 34, 37, 38], which at a high level, initializes two models and trains them alternatively. The first model selects some instances with small losses and then feeds them into the second model for training. The second model conducts the same process. The above steps repeat until they converge.

To summarize, although some robust machine learning methods (*e.g.*, co-teaching) leverage the loss to help distinguish clean instances from dirty ones, their goal is different from ours. To be specific, `MisDetect` aims to accurately identify *all mislabeled instances.* On the other hand, the goal of co-teaching is to improve the model accuracy. However, to achieve this goal, it is not necessary for co-teaching to detect all mislabels. Only identifying some highly confident mislabeled instances often is enough to improve the model, even if some mislabeled instances are missed. This leads to high precision, but poor recall and in turn low F-1 score, as confirmed by our experiments in Figures 9 and 10.
Data cleaning v.s. ML techniques. Holoclean [44, 53] leverages the factor graph to repair dirty instances (*e.g.*, duplicates, inconsistency), but it does not focus on mislabel detection and requires additional repairing constraints as input. Holodetect [27] utilizes few-shot learning to detect errors, which also does not consider mislabeled data, and it needs an additional clean train set. CHEF [55] iteratively cleans the most influential instances so as to maximize the model performance. Different from us, this method involves human annotators to verify mislabels and the goal is to save human cost while improving the performance of the model. TARS [21] is also a label-cleaning framework that involves humans to clean mislabels, so as to improve the model performance. Rain [54] identifies data errors that cause unexpected ML prediction results, but it needs users' compliant as input.
Data management for ML. Besides, existing methods also focus on improving machine learning model performance as well as the efficiency of ML model training using data management technology, including data discovery [18–20, 50], data cleaning [16, 26, 36] and data labeling [17, 33].

8 CONCLUSION

We study the problem of mislabel detection using early loss during iterative training. At early epochs, we regard training instances with large losses as mislabeled ones and iteratively remove them. Besides, we leverage the influence of instances to double-check the removed instances. We also design a classification model trained over the instances identified by the iterative process, to annotate the rest instances accurately. Extensive experiments show our superiority over state-of-the-art baselines.

ACKNOWLEDGMENTS

Chengliang Chai is supported by NSF of China (62102215). Ye Yuan is supported by the National Key R&D Program of China(Grant No.2022YFB2702100), the NSFC (Grant Nos. 61932004, 62225203, U21A20516) and the DITDP (Grant No. JCKY2021211B017). Guoren Wang is supported by the NSFC (U2001211). Lei Cao is supported by NSF DBI-2327954. Ju Fan was supported by the NSF of China (62122090 and 62072461), the Beijing Natural Science Foundation (L222006), and the Research Funds of Renmin University of China.

REFERENCES

[1] 1998. https://archive.ics.uci.edu/ml/datasets/Covertype.
[2] 1999. https://yann.lecun.com/exdb/mnist/.
[3] 2009. http://www.cs.toronto.edu/~kriz/cifar.html.
[4] 2011. http://ufldl.stanford.edu/housenumbers/.
[5] 2023. https://www.kaggle.com/datasets/ghassenkhaled/wine-quality-data.
[6] 2023. https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification.
[7] 2023. https://www.kaggle.com/datasets/teejmahal20/airline-passenger-satisfaction.
[8] 2023. https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction.
[9] 2023. https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset.
[10] 2023. http://codh.rois.ac.jp/kmnist/index.html.en.
[11] 2023. https://www.kaggle.com/datasets/zalando-research/fashionmnist.
[12] AI Marques R Alejo J Badenas, JS Sanchez, and R Barandela. 2000. Decontamination of training data for supevised pattern recognition. Advances in Pattern Recognition Lecture Notes in Computer Science 1876 (2000), 621–630.
[13] Léon Bottou and Olivier Bousquet. 2007. The tradeoffs of large scale learning. Advances in neural information processing systems 20 (2007).
[14] Leo Breiman. 2001. Random forests. Machine learning 45 (2001), 5–32.
[15] Carla E. Brodley and Mark A. Friedl. 1999. Identifying Mislabeled Training Data. J. Artif. Intell. Res. 11 (1999), 131–167. https://doi.org/10.1613/jair.606
[16] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. 2020. Human-in-the-loop Outlier Detection. In Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 19–33. https://doi.org/10.1145/3318464.3389772
[17] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. 2016. Cost-Effective Crowdsourced Entity Resolution: A Partial-Order Approach. In Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 969–984. https://doi.org/10.1145/2882903.2915252
[18] Chengliang Chai, Jiabin Liu, Nan Tang, Ju Fan, Dongjing Miao, Jiayi Wang, Yuyu Luo, and Guoliang Li. 2023. GoodCore: Data-effective and Data-efficient Machine Learning through Coreset Selection over Incomplete Data. Proc. ACM Manag. Data 1, 2 (2023), 157:1–157:27. https://doi.org/10.1145/3589302
[19] Chengliang Chai, Jiabin Liu, Nan Tang, Guoliang Li, and Yuyu Luo. 2022. Selective Data Acquisition in the Wild for Model Charging. Proc. VLDB Endow. 15, 7 (2022), 1466–1478. https://doi.org/10.14778/3523210.3523223
[20] Chengliang Chai, Jiayi Wang, Nan Tang, Ye Yuan, Jiabin Liu, Yuhao Deng, and Guoren Wang. 2023. Efficient Coreset Selection with Cluster-based Methods. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023, Ambuj K. Singh, Yizhou Sun, Leman Akoglu, Dimitrios Gunopulos, Xifeng Yan, Ravi Kumar, Fatma Ozcan, and Jieping Ye (Eds.). ACM, 167–178. https://doi.org/10.1145/3580305.3599326
[21] Mohamad Dolatshah, Mathew Teoh, Jiannan Wang, and Jian Pei. 2018. Cleaning Crowdsourced Labels Using Oracles For Statistical Classification. Proc. VLDB Endow. 12, 4 (2018), 376–389. https://doi.org/10.14778/3297753.3297758
[22] Charles Elkan. 2001. The Foundations of Cost-Sensitive Learning. In IJCAI 2001. Morgan Kaufmann, 973–978.
[23] George Forman. 2005. Counting Positives Accurately Despite Inaccurate Classification. In Machine Learning: ECML 2005 (Lecture Notes in Computer Science), Vol. 3720. Springer, 564–575. https://doi.org/10.1007/11564096_55
[24] Chuan Guo, Tom Goldstein, Awni Y. Hannun, and Laurens van der Maaten. 2020. Certified Data Removal from Machine Learning Models. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research), Vol. 119. PMLR, 3832–3842. http://proceedings.mlr.press/v119/guo20c.html
[25] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor W. Tsang, and Masashi Sugiyama. 2018. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In NeurIPS 2018. 8536–8546. https://proceedings.neurips.cc/paper/2018/hash/a19744e268754fb0148b017647355b7b-Abstract.html
[26] Shuang Hao, Chengliang Chai, Guoliang Li, Nan Tang, Ning Wang, and Xiang Yu. 2020. Outdated Fact Detection in Knowledge Bases. In 36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020. IEEE, 1890–1893. https://doi.org/10.1109/ICDE48307.2020.00196
[27] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In SIGMOD Conference 2019. ACM, 829–846. https://doi.org/10.1145/3299869.3319888
[28] Dan Hendrycks, Mantas Mazeika, Duncan Wilson, and Kevin Gimpel. 2018. Using Trusted Data to Train Deep Networks on Labels Corrupted by Severe Noise. In NeurIPS 2018. 10477–10486. https://proceedings.neurips.cc/paper/2018/hash/ad554d8c3b06d6b97ee76a2448bd7913-Abstract.html
[29] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. 2018. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. In ICML 2018 (Proceedings of Machine Learning Research), Vol. 80. PMLR, 2309–2318. http://proceedings.mlr.press/v80/jiang18c.html
[30] Yuan Jiang and Zhi-Hua Zhou. 2004. Editing Training Data for kNN Classifiers with Neural Network Ensemble. In ISNN 2004 (Lecture Notes in Computer Science), Fuliang Yin, Jun Wang, and Chengan Guo (Eds.), Vol. 3173. Springer, 356–361. https://doi.org/10.1007/978-3-540-28647-9_60
[31] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).
[32] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In International conference on machine learning. PMLR, 1885–1894.
[33] Guoliang Li, Chengliang Chai, Ju Fan, Xueping Weng, Jian Li, Yudian Zheng, Yuanbing Li, Xiang Yu, Xiaohang Zhang, and Haitao Yuan. 2018. CDB: A Crowd-Powered Database System. Proc. VLDB Endow. 11, 12 (2018), 1926–1929. https://doi.org/10.14778/3229863.3236226
[34] Junnan Li, Richard Socher, and Steven C. H. Hoi. 2020. DivideMix: Learning with Noisy Labels as Semi-supervised Learning. In ICLR 2020. OpenReview.net. https://openreview.net/forum?id=HJgExaVtwr
[35] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks. In ICDE 2021. IEEE, 13–24. https://doi.org/10.1109/ICDE51399.2021.00009
[36] Yuyu Luo, Chengliang Chai, Xuedi Qin, Nan Tang, and Guoliang Li. 2020. VisClean: Interactive Cleaning for Progressive Visualization. Proc. VLDB Endow. 13, 12 (2020), 2821–2824. https://doi.org/10.14778/3415478.3415484
[37] Eran Malach and Shai Shalev-Shwartz. 2017. Decoupling "when to update" from "how to update". In NeurIPS 2017. 960–970. https://proceedings.neurips.cc/paper/2017/hash/58d4d1e7b1e97b258c9ed0b37e02d087-Abstract.html
[38] Devraj Mandal, Shrisha Bharadwaj, and Soma Biswas. 2020. A Novel Self-Supervised Re-labeling Approach for Training with Noisy Labels. In WACV 2020. IEEE, 1370–1379. https://doi.org/10.1109/WACV45572.2020.9093342
[39] Prem Melville, Nishit Shah, Lilyana Mihalkova, and Raymond J. Mooney. 2004. Experiments on Ensembles with Missing and Noisy Data. In MCS 2004 (Lecture Notes in Computer Science), Vol. 3077. Springer, 293–302. https://doi.org/10.1007/978-3-540-25966-4_29
[40] Curtis G. Northcutt, Lu Jiang, and Isaac L. Chuang. 2021. Confident Learning: Estimating Uncertainty in Dataset Labels. J. Artif. Intell. Res. 70 (2021), 1373–1411. https://doi.org/10.1613/jair.1.12125
[41] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. 2017. Making Deep Neural Networks Robust to Label Noise: A Loss Correction Approach. In CVPR 2017. IEEE Computer Society, 2233–2241. https://doi.org/10.1109/CVPR.2017.240
[42] Barak A. Pearlmutter. 1994. Fast Exact Multiplication by the Hessian. Neural Comput. 6, 1 (1994), 147–160. https://doi.org/10.1162/neco.1994.6.1.147
[43] Romila Pradhan, Jiongli Zhu, Boris Glavic, and Babak Salimi. 2022. Interpretable Data-Based Explanations for Fairness Debugging. In SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 247–261. https://doi.org/10.1145/3514221.3517886
[44] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. Proc. VLDB Endow. 10, 11 (2017), 1190–1201. https://doi.org/10.14778/3137628.3137631
[45] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. 2018. Learning to Reweight Examples for Robust Deep Learning. In ICML 2018 (Proceedings of Machine Learning Research), Vol. 80. PMLR, 4331–4340. http://proceedings.mlr.press/v80/ren18a.html
[46] Juan José Rodríguez, Ludmila I. Kuncheva, and Carlos J. Alonso. 2006. Rotation Forest: A New Classifier Ensemble Method. IEEE Trans. Pattern Anal. Mach. Intell. 28, 10 (2006), 1619–1630. https://doi.org/10.1109/TPAMI.2006.211
[47] José Salvador Sánchez, Ricardo Barandela, A. I. Marqués, Roberto Alejo, and Jorge Badenas. 2003. Analysis of new techniques to obtain quality training sets. Pattern Recognit. Lett. 24, 7 (2003), 1015–1022. https://doi.org/10.1016/S0167-8655(02)00225-8
[48] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. 2019. An Empirical Study of Example Forgetting during Deep Neural Network Learning. In ICLR 2019. OpenReview.net. https://openreview.net/forum?id=BJlxm30cKm
[49] Hamed Valizadegan and Pang-Ning Tan. 2007. Kernel Based Detection of Mislabeled Training Examples. In SIAM 2007. SIAM, 309–319. https://doi.org/10.1137/1.9781611972771.28
[50] Jiayi Wang, Chengliang Chai, Nan Tang, Jiabin Liu, and Guoliang Li. 2022. Coresets over Multiple Tables for Feature-rich and Data-efficient Machine Learning. Proc. VLDB Endow. 16, 1 (2022), 64–76. https://doi.org/10.14778/3561261.3561267
[51] Yisen Wang, Weiyang Liu, Xingjun Ma, James Bailey, Hongyuan Zha, Le Song, and Shu-Tao Xia. 2018. Iterative Learning With Open-Set Noisy Labels. In CVPR 2018. Computer Vision Foundation / IEEE Computer Society, 8688–8696.

https://doi.org/10.1109/CVPR.2018.00906

[52] Zhen Wang, Guosheng Hu, and Qinghua Hu. 2020. Training Noise-Robust Deep Neural Networks via Meta-Learning. In CVPR 2020. Computer Vision Foundation / IEEE, 4523–4532. https://doi.org/10.1109/CVPR42600.2020.00458

[53] Richard Wu, Aoqian Zhang, Ihab F. Ilyas, and Theodoros Rekatsinas. 2020. Attention-based Learning for Missing Data Imputation in HoloClean. In MLSys 2020. mlsys.org. https://proceedings.mlsys.org/book/307.pdf

[54] Weiyuan Wu, Lampros Flokas, Eugene Wu, and Jiannan Wang. 2020. Complaint-driven Training Data Debugging for Query 2.0. In SIGMOD Conference 2020], June 14-19, 2020. ACM, 1317–1334. https://doi.org/10.1145/3318464.3389696

[55] Yinjun Wu, James Weimer, and Susan B. Davidson. 2021. CHEF: A Cheap and Fast Pipeline for Iteratively Cleaning Label Uncertainties. Proc. VLDB Endow. 14, 11 (2021), 2410–2418. https://doi.org/10.14778/3476249.3476290

[56] Dingwen Zhang, Deyu Meng, and Junwei Han. 2017. Co-Saliency Detection via a Self-Paced Multiple-Instance Learning Framework. IEEE Trans. Pattern Anal. Mach. Intell. 39, 5 (2017), 865–878. https://doi.org/10.1109/TPAMI.2016.2567393

[57] Xuezhou Zhang, Xiaojin Zhu, and Stephen J. Wright. 2018. Training Set Debugging Using Trusted Items. In AAAI 2018. AAAI Press, 4482–4489. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16155

[58] Zhilu Zhang and Mert R. Sabuncu. 2018. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. In NeurIPS 2018. 8792–8802. https://proceedings.neurips.cc/paper/2018/hash/f2925f97bc13ad2852a7a551802feea0-Abstract.html

[59] Zizhao Zhang, Han Zhang, Sercan Ömer Arik, Honglak Lee, and Tomas Pfister. 2020. Distilling Effective Supervision From Severe Label Noise. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020. Computer Vision Foundation / IEEE, 9291–9300. https://doi.org/10.1109/CVPR42600.2020.00931

[60] Xingquan Zhu, Xindong Wu, and Qijun Chen. 2003. Eliminating Class Noise in Large Datasets. In ICML 2003, Tom Fawcett and Nina Mishra (Eds.). AAAI Press, 920–927. http://www.aaai.org/Library/ICML/2003/icml03-119.php