MisDetect: Iterative Mislabel Detection using Early Loss

Anonymous Author(s)

ABSTRACT

Abstract here.

ACM Reference Format:
Anonymous Author(s). 2023. MisDetect: Iterative Mislabel Detection using Early Loss. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (SIGMOD' 24).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Data quality is one of the most significant problems in data management, among which mislabels are a common dirty data type that could directly lead to low-quality data analysis and misleading business decisions. Labels are always large-scale and error-prone because they may be crowdsourced from non-experts or collected from web annotations, so it is inevitable to use automatic methods to detect mislabels. However, existing mislabel detection approaches suffer from fair accuracy.

Existing Solutions. Traditional methods [] rely on the data locality to detect mislabels. For example, in order to whether the label of an instance in a dataset is correct, the typical KNN method [] checks its neighbors, and if they have inconsistent labels, the instance tends to be mislabeled, otherwise it is a clean one. This type of methods has low detection accuracy because they just consider the local neighbors of each instance rather than the entire dataset. Therefore, to improve the accuracy, machine learning (ML) models are incorporated to help mislabel detection [] because intuitively, mislabels definitely have impact on the supervised model training. For instances, ensemble-based methods [] involve multiple models to train on the entire dataset and check the consistency of the prediction results from these models for each instance. Cleanlab [] utilizes confident learning to estimate the joint distribution of mislabels and correct labels. This line of methods can capture the entire data distribution of the dataset through ML and avoid the data locality problem. However, the accuracy is still not high because they learn the distribution from both the incorrect and correct labels. In this way the model has already fitted on the dirty data, and thus the learned distribution is not accurate, leading to inaccurate prediction results.

Challenge. As discussed above, to pursue high accuracy of mislabel detection, we have to consider the data distribution of the entire dataset rather than the locality. In terms of using ML to capture the distribution, it is challenging to eliminate the impact of mislabels on the learned distribution. Ideally, if we know the distribution learned from all clean instances, we can easily use it to detect the dirty ones, but unfortunately, we cannot know it in advance.

Based on the challenge, we have the following observation and proposed methods.

Observation. Since it is inappropriate to detect mislabels after training because the model has already fitted the dirty data. We consider whether we can leverage the metadata during (or at the beginning of) training to help detect mislabels. The high level intuition is that this strategy clearly avoids data locality because all instances participate in training, while it can also capture the entire data distribution to a certain degree before the model fits dirty instances.

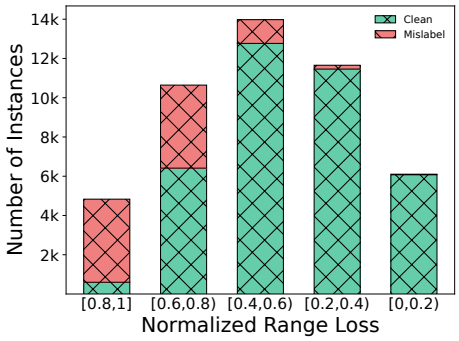

Figure 1: An Example of Early Loss.

Therefore, we observed that the *early loss* can well distinguish mislabeled instances from clean ones. Early loss means the loss of each instance at early training epochs. As shown in Figure 1, 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 range. We can see that the mislabeled instances always have large loss, while the clean ones are associated with relatively small loss.

Our proposal. Based on the above observation, generally speaking, we can conclude that mislabeled instances are harder for an ML model to fit than the clean instances especially at early epochs, leading to large loss. Therefore, we propose our MisDetect framework that leverages the early loss to detect mislabeled instances in an ML training set. The key idea is straightforward yet effective that at each early epoch, we sort the instances in the training set according to the loss in descending order. Then we tend to iteratively identify top instances as mislabeled ones and remove them from the training set. We remove because we hope to navigate the model towards a well-performed one, as if it is trained over clean instances. In this way, at the subsequent epochs, the model can better fit clean instances and recognize mislabeled ones.

However, to make the model better, purely removing instances with the largest value is sub-optimal because an instance with a large loss does not imply it has a large impact on the model. Hence, MisDetect has another module to estimate the influence of each instance. A higher influence indicates that if we identify the corresponding instance as mislabeled and remove it, the model will be more affected and quickly lead to a well-performed model trained over clean instances. Besides, we also design a strategy to estimate the rate of mislabeled instances, so as to help judge when the MisDetect stops. What's more, there are likely to exist some outliers in each dataset, which are also hard to be fitted by a model

Table 1: Notation Table

Notations	Description
D	Entire train set including mislabeled instances.
D_m	The set of mislabeled instances ($D_m \subseteq D$).
D_c	The set of clean instances ($D = D_m \cup D_c$).
M	Downstream model for mislabel detection.
M	Downstream model for mislabel detection.
M	Downstream model for mislabel detection.
M	Downstream model for mislabel detection.
M	Downstream model for mislabel detection.

at early epochs with large loss. If they are identified as mislabeled, our precision goes down. Hence, we also propose a module to cope with these outliers.

Contributions. Our contributions are summarized as follows:
(1) We formally model the mislabel detection problem and design an iterative mislabel detection framework `MisDetect` with high accuracy based on machine learning training.
(2) We incorporate the early loss to help determine whether an instance is mislabeled, which is very effective.
(3) We propose an influence computation method customized to our problem to further improve the performance of mislabel detection, based on the influence function.
(4) Sufficient experiments on XX datasets over XX baselines demonstrate that `MisDetect` outperforms them up to XX in terms of the F1-score.

The paper is organized as follows. We first introduce the preliminaries including the problem definition in Section 2. Then the overall framework is introduced in Section 3. Next, we introduce the details of how to use early loss to detect mislabels (Section 4), followed by using the instance influence to further improve the performance Section 5. Furthermore, we design other optimization techniques like mislabel rate estimation, and outlier processing in Section 6. Related work is discusses in Section 8 and we conduct experiments in Section 7. The significant notations used in this paper are listed in Table 1.

2 PRELIMINARY

In this Section, we will first introduce some basic concepts, followed by the problem definition.

Mislabel detection. Given a dataset $D = \{o_1, o_2, ..., o_N\}$, each instance $o_i = (x_i, y_i), i \in [1, N]$ is associated with the feature x_i and a corresponding label y_i. Labels are error-prone due to many reasons like crowdsourcing workers' error, so each instance has an unknown ground truth label y_i^*. The problem of mislabel detection is to identify all mislabeled instances, *i.e.*, $D_m = \{o_i | o_i \in D, y_i \neq y_i^*\}$. Besides, we use D_c to denote the set of clean instances, *i.e.*, $D = D_c \cup D_m$. Note that we could support any data type, and in practice, we test images and tabular data.

Supervised machine learning. As we all know, labels play a significant role in supervised ML training. Clean training instances will no doubt lead to a smooth training process and better ML model, and thus it is reasonable to leverage ML model to detect mislabels []. Given a clean dataset, *e.g.*, D_c , the objective of training is to compute the best parameter so as to minimize the training loss. To be specific,

$$w^* = \arg\min_{w \in \mathcal{W}} l(w), l(w) = \frac{1}{|D_c|} \sum_{i=1}^{|D_c|} l_i(w, x_i) \tag{1}$$

where $\mathcal{W}$ represents the parameter space, and $l_i(w, x_i)$ denotes the loss of the $i-$th training instance. Typically, the gradient decent algorithm is often applied to optimize Equation 1. To achieve high efficiency, we can use stochastic gradient decent to accelerate the training process. Note that we can support multi-classification tasks, *i.e.*, the label y has to be categorical values. For ease of representation, we can abbreviate $l_i(w, x_i)$ as $l_i(w)$.

Our problem definition. As discussed above, if we decide to use machine learning model to help detect mislabeled instances, we can have the following formulation.

$$D_m = \arg\min_{D' \subseteq D} \|w_{D'} - w^*\| \tag{2}$$

where $w_{D'}$ represents the model parameter trained over the dataset $D \setminus D'$. Overall, Equation 2 is easy to understand that we aim to detect a set of mislabeled instances D' such that training over $D \setminus D'$ is equivalent to training over D_c. Obviously, when we detect exactly all the mislabeled instances, the above optimization goal will be achieved. However, Equation 2 is rather challenging to be solved because (1) we cannot obtain w^* because we do not know D_c in advance, and (2) assuming that we can know w^*, it is prohibitively expensive to enumerate the subsets of D to find D_m.

Next, we introduce the over framework of `MisDetect` to address the mislabel detection problem using ML model.

3 FRAMEWORK

To address the above challenges, we design the `MisDetect` framework that introduces the early loss to well distinguish mislabeled instances from clean ones. As shown in Figure 2, the basic idea of `MisDetect` is that with the number of epochs increases from the beginning, at early epochs, we can regard a number of instances with small loss as clean instances, over which we can train a model almost without dirty data information. Then we can leverage the parameter of this model as a replacement of w^*. Meanwhile, we can also let a number of instances with large loss constitute a pool of mislabeled instance candidates, among which we pick some instances with high influence based on w^*. Afterwards, we identify these instances as mislabeled and remove them, leading to more accurate mislabel detection for further epochs.

Early loss based detection. As discussed in Section 1, mislabeled instances tend to have a large early loss because the model is hard to fit them, especially at early training stages. In Figure 2, we can see that we detect mislabels iteratively as the training progresses. Specifically, at the j-th epoch, we define the loss of the i-th instance as $l_i^j = y_i - w_j \cdot x_i$, where w_j is the model parameter of this epoch. Then we include a proportion of instances with large loss as the candidate pool of mislabeled instances, denoted by $\mathcal{P}_m$, because they are likely to be dirty. Similarly, the instances with small loss

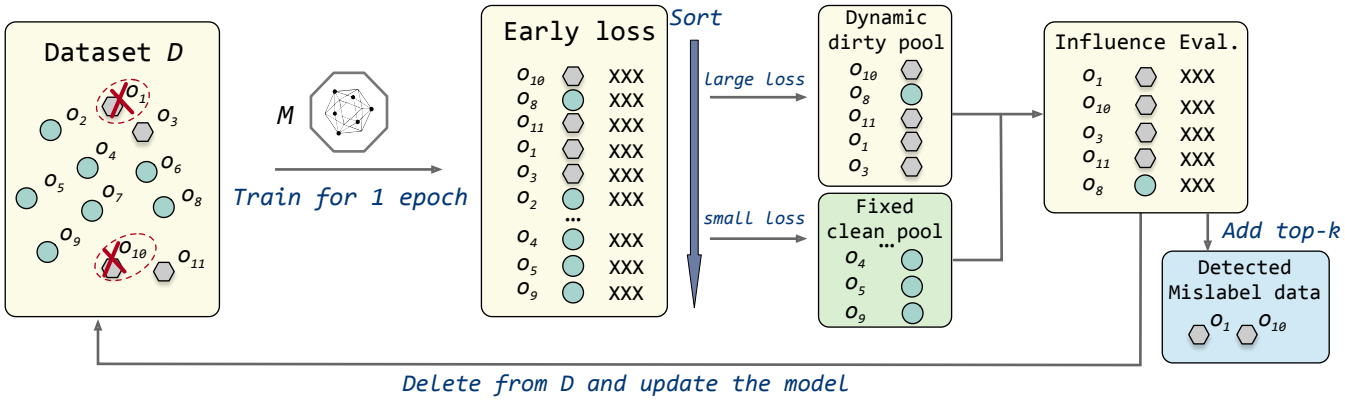

Figure 2: MisDetect Framework

are very likely to be clean because they can be well fitted by the model at early epochs. Therefore, we construct another pool $\mathcal{P}_c$ of clean instances, over which we train a model to measure the influences of instances in $\mathcal{P}_m$. This will be briefly introduced next. Note that we fix $\mathcal{P}_c$ after the first epoch because these instances that have already been fitted with small loss will not vary much afterwards, so we do not have to iteratively update it considering the efficiency. However, $\mathcal{P}_m$ has to be dynamically updated since the loss is likely to vary when it is large at the beginning.

Influence evaluation. Given $\mathcal{P}_m$, a straightforward method is to just directly remove all instances in it or someones with the largest loss, but although the removed instances tend to be mislabeled, they may not influence the current model much. Recap that in Equation 2, we aim to remove mislabeled instances such that the model can be close to w^*. And the more quickly (with fewer epochs) we approach w^*, the easier for the model to detect mislabels because w^* is the ideal parameter trained over all clean instances.

Since we cannot know w^* in advance, in order to measure the influence of removing each instance on approaching w^*, a basic solution is to instead measure the influence of removing the instance on the model trained with current dataset. Although we can compute this influence efficiently without retraining using the influence function [], it does not perform well. The reason is that such strategy cannot provide evident signal because current dataset is mixed with dirty and clean instances, and thus no matter how large the influence of the instance in fact is, the result calculated by the influence function is not obvious.

Therefore, we propose a new influence computation method customized to our problem. As introduced before, based on the early loss, we construct a clean pool $\mathcal{P}_c$, in which the instances are very confidently clean. We train a model M_c with parameter w_c over $\mathcal{P}_c$. For each instance o_i, we measure its influence of adding o_i to $\mathcal{P}_c$. Since $\mathcal{P}_c$ is clean, adding a dirty instance will be much more sensitive than the aforementioned method. Intuitively, if an instance has a large impact on a model after adding it into the dataset $w.r.t.$ w_c, removing it will be rather helpful for quickly approaching w^* because w_c can be regarded as an approximation of w^*. We will introduce this in detail in Section 5.

False positive elimination. During the process of mislabel detection, there inevitably exist some false positive instances, $i.e.$, they are in fact clean but are identified as mislabeled ones. One main reason is that there exist some outliers in D, which will also produce large loss at early stages of training. To address this, we propose to detect outliers before training, and temporarily put them on hold without participating in mislabel detection during training. In this way, these outliers will not be identified as false positives although they have large loss. When the mislabel detection process terminates, we can leverage the mislabeled/clean instances that have already been detected to check which ones among these outliers are mislabeled.

On the other hand, there will also be some clean instances that are not classified as outliers, but they could produce large loss as they are hard to fit for a model. We also expect to identify these false positives. In practice, we observe that they are harder to fit than most clean instances, butx still easier than those mislabeled ones, so we propose to address this problem by tracking the variation of loss. Intuitively, if the loss of an instance with initial large loss quickly decreases, it tends to be a false positive. Otherwise, we still keep it as a mislabeled instance. We will discuss how to eliminate false positives in detail in Section 6.

CC **[Current figure does not show false positive elimination.]**

Overall MisDetect Framework. To summarize, given a train set D, we first identify a subset of outliers O in it and temporarily remove these outliers from D ($i.e.$, $D = D/O$). CC **[Next, an important step is to estimate the number of mislabeled instances in D such that we can decide when to stop the detection process (Section 4).]** Afterwards, we iterative train over D and detect mislabeled instances in each epoch, as shown in Figure 2. At each epoch, we can obtain the loss of each instance. Then, we will extract the top largest ones to constitute $\mathcal{P}_m$, and the bottom smallest ones to constitute $\mathcal{P}_c$. On the basis of $\mathcal{P}_c$, we can well evaluate the influence of each instance in $\mathcal{P}_m$ and select the top-k instances with high influence as mislabeled ones. For example, at current epoch in Figure 2, $\mathcal{P}_m$ consists of $o_{10}, o_8, o_{11}, o_1, o_3$. Next, we evaluate the influence. Although o_8 is a clean instance with a large loss, its influence is not large if we put it into $\mathcal{P}_c$ that is full of clean instances. Hence, o_8 will not be detected in this epoch, and o_1, o_{10} with high influence will be

detected and removed. Then, another epoch starts, and this repeats until we have detected the number of estimated mislabels. Note that during the training process, we will also track the loss variation of each instance to identify some candidates that we suspect they are false positives. Finally, we will look back to these suspicious candidates as well as O to check whether they are mislabeled to further improve the detection accuracy.

4 EARLY LOSS BASED ALGORITHM

5 INFLUENCE EVALUATION

6 FALSE POSITIVE ELIMINATION

7 EXPERIMENT

7.1 Experimental Settings

Dataset.We evalute our algorithm on 7 different real-world datasets from a diverse array of domains, whose size varying from the magnitude of 10^3 to 10^6 and the number of class differ from 2 to 100. We first used the dataset form CleanML, which have been performed synthetic mislabel injection with the strategy that flipping 5% of the labels in each class. Then, we also use different kinds of dataset including image and table and dataset with large number of classes to further verify the effectiveness and scalability of our algorithm. The details are listed as follows:

(1) USCensus: This dataset contains 32,561 items about US Census records for adults. Each item has 14 attributes, such as age, education, sex, etc. The classification goal is to predict whether the adult earns more than $50,000.
(2) Marketing: A total of 8993 records are contained in this dataset, which information about household income including education, sex. etc. The goal is to make predictions about whether the annual family income is less than $25,000.
(3) EEG: This dataset consists of 14,980 Electroencephalogram recordings with 14 Electroencephalogram attributes varying from AF3 to, AF4 The classification task is to predict whether the eye-state is closed or open.
(4) CIFAR-10: The dataset is a computer vision data set for universal object recognition, which contains 50000 32 X 32 RGB color pictures, a total of 10 categories. The task is to predict which kind does the picture belong to. Mislabeled data are artificially introduced by flipping labels of 40% for each type of the dataset randomly. **(need explain?)**
(5) CIFAR-100: This dataset is like CIFAR-10, except that it has a total of 100 classes, and each class contains 500 images. The classification task is also to predict the category of a given picture.
(6) CovType: This is also a muti-classification dataset, which contains 7 different forest cover type. With a total of 581012 samples, and each sample consist of 54 attributes, such as Elevation, Wilderness Area, Horizontal Distance To Roadway etc.
(7) Mobile Price Prediction: This is a small tabular dataset with only 2000 records. The task is to predict price range of the mobile on the basis of the information about the mobile, specification like Battery power, 3G enabled , wifi ,Bluetooth, Ram etc.**(need remove?)**

Method.We compare our approach against several competing mislabel detection methods. First, we consider 10 baselines:

1. **K-Nearest Neighbor(KNN):** The method counts the number of inconsistencies between the label of a training instance and the labels of its surrounding neighbors. If there is strong evidence of distinction among the labels, the training instance is marked as mislabeled. This kind of method is called local learning method.

2. **Nearest Centroid Neighborhood (NCN):** This method is also belong to local learning method, which assumes that the labels of mislabeled instances tend to disagree with the labels of other instances in their surrounding neighborhood. The difference between KNN and this method is the approach about how to find the nearest neighbor.

3. **Training Set Debugging Using Trusted Items(DUTI):** DUTI utilizes a small set of additional "trusted items"to help detect incorrectly labeled item, which core is to finds the smallest changes for the labels in training set such that the classifier trained on the changed dataset classify all the trusted items correctly. DUTI proposes an iterative algorithm to achieve its goal.

4. **Forgetting Events:** In the process of model training, a sample has been correctly classified by the model. With the update of model parameters, the sample has been incorrectly classified. This process is called a forgetting event of the sample. This method identifies a mislabel sample based on one assumption that noisy sample often experience more forgetting events than normal samples during model training.

5. **Ensemble-based method with consensus filter:** Ensemble-based method assumes that multiple, independent clissifiers often result in conflicting labels about incorrectly labeled training sample. Algorithms that belong to this category vary in terms of how the different classifiers are constructed. Besides, the consensus filter is a strategy which means that a training example could be marked as a mislabel data only if it is misclassified by all the classifiers in the ensemble.

6. **Ensemble-based method with majority vote:** The main idea of this method is tha same as the Ensemble-based method with consensus filter.The only inconsistency between them is that majority vote strategy considers an example to be mislabeled if it disagrees with the majority vote of the classifiers.

7. **Cleanlab:** Cleanlab proposes an emerging framework called confident learning, which core for detection include two main steps: Count and Clean. The first step is to estimate a joint distribution directly between incorrectly labeled data and uncorrupted(real) data based on the assumption of classification noise process(CNP)[]. After that, some instances in the training set would be tabbed as mislabel based on the joint distribution matrix computed above in the second step.

8. **Noisy Cross-Validation(NCV):** This method first randomly devides a noisy training set into two halves, then train a neural network for these two half separately. After that, the network which is trained on one half will be applied to another half of the dataset. A sample would be identified as mislabel when its current label is different from its predicted label.

9. **Iterative Noisy Cross-Validation(INCV):** Obviously, this is a iterative method about noisy cross-validation. Apart from selecting mislabel samples, the INCV removes samples that have large categorical cross entropy loss at each iteration.

10. **Partition Filter:** This method partitions a noisy dataset into multi-subsets, and then construct a good classifier from each subset iterately. For a given training sample, all classifier will be applied on it, the mislabel sample often have higher probability to have larger misclassified times.

We also consider three variants of our algorithm. The goal is to compare the effectiveness of the cross-validation using both early loss and parameter influence. For all variations, we use the same training paradigms. We consider the following variants:

1. **MisDetect Without Iteration:** This variant just does the detection once, which means choicing a very big dirty pool containing instances with large early loss and a clean pool to test the influence of each instance, then directly filter all the mislabel instances with large influence in the dirty pool without iteration.

2. **MisDetect Without Parameter Influence:** Apparently, this variant does not test the parameter influence after dirty pool is established at each iteration. It just consider the loss of each instance at initial training stage.

3. **MisDetect Without Early Loss:** This variant using parameter influence as the main detect method, in other words, it does not consider the loss of every instance in the training set except needing a clean pool which is created through early loss, the instance with large influence will be detected as mislabel data in this variant.

Hyper-prameter Setting. We train convolution neural network for image-classification task and simple 3-layers perceptron for tabular dataset.Both networks apply learning rate with 0.001 and optimizer with Adam. For CIFAR-10, CIFAR-100, CovType and Mobile Price Prediction datasets, we inject mislabel data by flipping labels of 20% for each type of the dataset randomly. For MisDetect, we chose XXX as the clean pool len.For Ensemble-based method, we use a toal of 10 different models,such as AdaBoost, RandomForest, Support Vector Machine, etc. For Cleanlab, we use LogisticRegression to compute the thresholds for noisy estimation.**(need?)**

Evaluation Metrics.We mainly focus on the effectiveness of our algorithm and other baselines, so we take precision, recall and F1-score as the most important metrics.

7.2 Comparison with Baselines

Given that MisDetect is a iterative detection method,

7.3 Mislabel Ratio Evaluation

7.4 Mislabel Distribution Evaluation

We test the algorithms on two types of label noise: random and systematic label noise. In random label noise, each label is randomly flipped to any of the remaining classes with equal probability, whereas for systematic noise, label flipping is done in a set of similar classes. For example, for a class âĂIJaâĂİ, all its mislabel samples are given the identical noisy label âĂIJbâĂİ.

Intuitively, random noisy labels are easier to fit than systematic ones since the model will learn a more directed and hence stronged signal. In this section , we compares the precision, recall and F1-score in all datasets between the two types of label noise.

(Result.) As shown in figure X.

7.5 Model Evaluation

7.6 Stop Condition Evaluation

8 RELATED WORK

9 CONCLUSION

REFERENCES