Learned Label Aggregation for Weak Supervision

Renzhi Wu
Georgia Tech
renzhiwu@gatech.edu

Shen-En Chen
Georgia Tech
achen353@gatech.edu

Xu Chu
Georgia Tech
xu.chu@cc.gatech.edu

Abstract

The lack of labeled training data is the bottleneck of machine learning in many applications. To resolve the bottleneck, one promising direction is the data programming approach that aggregates different sources of weak supervision signals to generate labeled data easily. Data programming encodes each weak supervision source with a labeling function (LF), a user-provided program that predicts noisy labels. The quality of the generated labels depends on a label aggregation model that aggregates all noisy labels from all LFs to infer the ground-truth labels.

Existing label aggregation methods typically rely on various assumptions and are not robust across datasets, as we will show empirically. We for the first time provide an analytical label aggregation method that makes minimum assumption and is optimal in minimizing a certain form of the averaged prediction error. Since the complexity of the analytical form is exponential, we train a model that learns to be the analytical method. Once trained, the model can be used for any unseen datasets and the model predicts the ground-truth labels for each dataset in a single forward pass in linear time. We show the model can be trained using synthetically generated data and design an effective architecture for the model. On 14 real-world datasets, our model significantly outperforms the best existing methods in both accuracy (by 3.5 points on average) and efficiency (by six times on average).

1 Introduction

The lack of labeled training data is a major challenge impeding the practical application of machine learning (especially deep learning) techniques. Traditionally, labeled data is obtained with human annotators manually annotating each data point. This process is expensive and does not scale to labeling large amounts of unlabeled data points. Therefore, practitioners have been increasingly turned to weak supervision in which large amounts of cheaply generated noisy labels are used. There are many forms of weak supervision sources, e.g., external knowledge bases [42], existing pre-trained models [15], and heuristics/rules [53]. To unify different sources, the data programming paradigm [48, 46] was proposed. In data programming, the user expresses each available weak and noisy supervision signal from different sources with a labeling function (LF), a small program that takes in a data point and outputs a noisy label. After that, each LF is applied to unlabeled data of arbitrary size to obtain a noisy label vector; then, a label aggregation model (also referred as label model in literature) is used to aggregate all noisy label vectors to infer the unknown ground-truth labels. The inferred labels can then be used to train any downstream end models, just like the manually provided labels. The data programming paradigm has been successful in various tasks [60, 21, 38, 37, 22, 51] and industry scenarios [40, 8, 20].

The core challenge in data programming is how to aggregate all noisy label vectors to infer the ground-truth labels. The existing approaches typically rely on various assumptions that may not hold in practice. For example, some works assume that the noise of each LF is only dependent on the hidden ground truth (e.g., DS [17]), while the noise can be distributed arbitrarily in practice. Many works assume that the dependency structure of the LFs is known as a prior or can be inferred reliably.
without ground truth (e.g. DP [48], FS [22] and MeTaL [47]), while in practice the dependency structure is never provided as a prior nor can be inferred reliably. Some probabilistic graphical model (PGM) based methods (e.g. DP [48], FS [22] and NPLM [61]) make the Markov assumption (or its simplified form, LFs are conditionally independent), which also may not hold in practice. Due to these assumptions, existing approaches do not have robust performance across datasets, as we will show in experiments. The existing approaches also involve an unsupervised learning process for each dataset which can be very expensive (e.g. Gibbs sampling on PGM [48]) for big datasets.

In this paper, we for the first time present an analytical label aggregation method which does not make any of the aforementioned assumptions. The analytical method is optimal in the sense that it minimizes a certain form of averaged prediction error. The analytical method is conceptually simple but is of exponential complexity to compute. Fortunately, we show that we can train an ML model that learns to be the analytical method. Once trained, the model predicts the ground-truth labels in a single forward pass with a complexity linear to the input size. To train a model, we need to consider what training data to use and what model architecture to use. We first design a training data generation method and formally prove that a model trained on our synthetically generated data learns to be the analytical method. We then design an effective model architecture that makes the trained model able to work on arbitrary number of noisy label vectors of arbitrary size and ensures the trained model to be invariant (and equivariant) to random permutations of LFs (and data points).

We highlight that, once our model is trained on synthetically generated data, it can be used for any unseen real datasets without needing any forms of update or fine-tuning. This is due to the fact that our model learns to be the analytical solution which is applicable to any datasets. We also highlight that, all existing methods (except majority vote) involve an unsupervised learning process, while our model predicts the ground-truth labels in a single forward pass.

Contributions. We make the following contributions:

• We for the first time present an analytical method for label aggregation which is optimal in the sense that it minimizes a certain form of the averaged prediction error, though directly using the analytical method is of exponential complexity.

• We train a model to learn the analytical method. The trained model can be used to infer the ground-truth labels for any unseen dataset in a single forward pass.

• We design a training data generation method and prove that a model trained on the synthetically generated data learns to be the analytical method.

• We design an effective model architecture so that the trained model is applicable to arbitrary number of LF label vectors of arbitrary size and is invariant/equivariant to the permutation of LF label vectors and data points.

• We empirically show that our method significantly outperforms the best existing methods over 14 real-world weak supervision datasets in both accuracy (by 3.5 points on average) and efficiency (by a speedup of six times on average).

2 Related Work

Label Aggregation for Weak Supervision. Most existing methods use probabilistic graphical models (PGM) [48, 22, 61]. These methods can have some guarantees under some assumptions [48, 22]. The popular system Snorkel [55] currently adopts a matrix completion model (MeTaL [47]) which is shown to be more efficient and accurate than PGM-based methods. The existing methods typically assume the dependency structure of the LFs is given by the users [47, 48, 22] and there is another line of work on inferring the dependency of LFs [7, 58]. Our method provides an analytical form with minimum assumption and is fundamentally different from all existing methods.

Joint Training of Label Aggregation Model and End Model. There are methods that jointly train a label aggregation model and an end model (e.g. classifier on images) [51, 64]. These methods have the benefit of incorporating additional information from the raw data (e.g. images). However, training an end model is very slow, while LF development is a trial-and-error process [12] in which immediate feedback from a label aggregation model can be crucial for debugging the LFs [59, 51]. Therefore, coupling with end model training inevitably hampers the LF development process [51]. Also, the aggregated labels (or even the developed LFs) are tied to the particular end model and may not work
well when the model architecture or loss function changes. For these reasons, in most applications, label aggregation and end model training are two separate processes \[40\] [8] [20] [60] [21] [38] [22].

Crowdsourcing. Crowdsourcing has a similar setting where each crowd worker can be seen as a LF. The Dawid and Skene’s method (DS) \[17\] is the seminal method that has been widely used even until recently [68, 54]. It assumes each crowd worker to be independent and assumes the noise of each crowd worker is only dependent on the hidden ground-truth labels. Many methods have been developed by extending DS (e.g. to have theoretical guarantees under some assumptions or to improve efficiency) [67, 29, 14, 54]. A recent method (EBCC) that models the joint distribution of the crowd workers with a mixture of low dimensional tensors achieves the state-of-the-art performance [36]. One major difference of crowdsourcing from weak supervision is that crowd workers provide labels without coordination so typically one can assume them to be independent, while in weak supervision one tends to write multiple LFs with a similar labeling heuristic [41].

3 Problem Setup

Let \( n \) and \( m \) denote the number of data points and the number of LFs respectively. Let \( X \in \{+1, -1, 0\}^{n \times m} \) denote a label matrix where \( X[i, j] \in \{+1, -1, 0\} \) denotes the weak label of the \( i^{th} \) data point provided by the \( j^{th} \) LF \((j \in [0, m - 1])\). The values \(+1\) and \(-1\) denote the positive and negative classes respectively and \(0\) denotes abstention (meaning that an LF does not have enough information to label a data point as either positive or negative [48]). The goal of a label aggregation model is to infer the unknown ground-truth label vector \( y \in \{+1, -1\}^n \) using \( X \).

The Better-than-random Assumption. In principle, \( X \) could be any matrix in \( \{+1, -1, 0\}^{n \times m} \) and \( y \) can be any vector in \( \{+1, -1\}^n \). For an arbitrary \( X \) and an arbitrary \( y \), there is no way to infer \( y \) from \( X \) with a better performance than random guess. In weak supervision literature, it is assumed that \( X \) encodes some knowledge about \( y \) so that one can make a meaningful estimation of \( y \) using \( X \). Specifically, most existing work on weak supervision [48, 22, 47, 51] explicitly or implicitly makes the assumption that each LF is a better-than-random estimation of \( y \). Formally, the \( j^{th} \) LF being better-than-random can be described as \( p(X[i, j] = y|y[i] = y) > p(X[i, j] = -y|y[i] = y) \) for \( y \in \{+1, -1\} \). Replacing the probability as empirical frequencies observed in \((X, y)\), the assumption can be written as:

\[
g(X, y, j, +1) = 1 \text{ and } g(X, y, j, -1) = 1
\]

where
\[
g(X, y, j, y) = \begin{cases} 1, & \text{if } \sum_{i=0}^{n-1} X[i, j] = y \land y[i] = y > \sum_{i=0}^{n-1} X[i, j] = -y \land y[i] = y \\ 0, & \text{otherwise}; \end{cases}
\]

Intuitively, \( g(X, y, j, y) \) denotes whether the \( j^{th} \) LF is better than random for class \( y \). When \( g(X, y, j, +1) = g(X, y, j, -1) = 1 \), the \( j^{th} \) LF is better than random for both the positive class and the negative class, and hence the LF is said to be better than random.

A Weaker Form of the Better-than-random Assumption. The assumption that every LF being better-than-random intuitively makes sense because LFs are developed by humans. However, humans tend to make mistakes, so it is unrealistic to assume every LF is better than random. Further, it is possible that one LF makes highly accurate predictions for one class while being worse than random for another class, so we should also consider different classes separately. Therefore, we propose to loosen the assumption to be that, for each class, the majority of LFs are better than random; formally:

\[
\sum_{j=0}^{m-1} g(X, y, j, +1) > \frac{m}{2} \quad \text{and} \quad \sum_{j=0}^{m-1} g(X, y, j, -1) > \frac{m}{2}
\]

We define \( \sigma(X, y) = 1 \) when Equation (2) is satisfied and \( \sigma(X, y) = 0 \) otherwise. We say a pair \((X, y)\) is valid when \( \sigma(X, y) = 1 \). Intuitively, \( \sigma \) constrains the space of the predicted label vector \( \hat{y} \) and we would only predict one of those label vectors with \( \sigma(X, y) = 1 \) for a label matrix \( X \). Note the method we are going to propose is not tied to this form of the assumption, and it also works with the original assumption (or even any other assumptions to define \( \sigma \)). We will show the gain of the weaker form of the assumption in ablation study in Section 5.5.

3
4 Learned Label Aggregation

An Optimal Estimator of the Ground-truth Label Vector. For each label matrix \( X \), let \( U(X) = \{ y \in \sigma(X) \mid y = 1 \} \) denote the set of valid candidate ground-truth label vectors for \( X \). The expected error of an estimator \( h \) of the ground-truth label vector on each \( X \) is:

\[
\epsilon(X, h) = \sum_{y \in U(X)} p(y|X)||y - h(X)||
\]

where \( p(y|X) \) is a distribution of \( y \) defined on set \( U(X) \) and \( || \cdot || \) denotes L2 loss (i.e. squared error). \( p(y|X) \) is unknown and can be different in different real-world applications. Without additional information apart from \( X \), there is no way to determine the preference of some valid choices of \( y \) over other valid choices of \( y \), so the uniform distribution (i.e. \( p'(y|X) = \frac{1}{|U(X)|} \)) is intuitively the “best” approximate for the unknown \( p(y|X) \). In fact, using the uniform distribution has optimalities in both the worst case and the average case. To maintain the flow of the paper, we defer the formal definition and proof of the optimalities of using the uniform distribution to Appendix B. Replacing \( p(y|X) \) by the uniform distribution, Equation 3 becomes:

\[
e'(X, h) = \frac{1}{|U(X)|} \sum_{y \in U(X)} ||y - h(X)||
\]

\( e'(X, h) \) can be interpreted as the average error of all possible outcomes. An estimator \( h \) can be said to be optimal if it minimizes the error \( e'(X, h), \forall X \).

**Theorem 1.** \( h^*(X) = \frac{1}{|U(X)|} \sum_{y \in U(X)} y, \forall X \) is the optimal estimator for the ground-truth vector.

We omit the proof as it is straightforward (The mean minimizes mean squared error.). Theorem 1 makes sense intuitively: since \( X \) is the only information we have, \( y \) can be any element in \( U(X) \) and there is no information to support preferences of some elements over other elements in \( U(X) \), so the best prediction one can make is the average of all elements in \( U(X) \).

Although we have the analytical form of the optimal estimator \( h^* \), computing it is of exponential complexity as \( U(X) \) is exponentially large for any \( X \). Therefore, we propose to train a model \( h \) to learn the optimal estimator. Once trained, the inference complexity of the model will be linear to the size of the input, i.e. \( O(nm) \). To materialize this idea, we need to answer two questions: (1) What training data to use? (2) What model architecture to use? We discuss both in the following sections.

**Discussion.** We note that our overall method is not tied to the choice of using an uniform distribution for \( p(y|X) \). If one can come up with a better distribution, our overall method still applies and one only needs to make two changes: The form of \( h^* \) will be a weighted average of the \( y \) vectors i.e. \( h^*(X) = \sum_{y \in U(X)} p(y|X)y \) and our data generation method (to be introduced in the next section) should also be adapted according to the new distribution. Then, the overall pipeline will be the same.

4.1 Training Data Generation.

We aim to synthetically generate many pairs of \((X, y)\) as training data. Let \( D = \{(X_1, y_1), \ldots \} \) denote the training set. We design a way of generating the training set \( D \) so that the model \( h \) trained on \( D \) by minimizing the cross entropy loss learns the optimal estimator \( h^* \).

Let \( U = \{(X, y) \mid \sigma(X, y) = 1 \} \) denote the set of all valid pairs. Any method \( M \) to generate a training set \( D \) is to sample \( |D| \) elements from set \( U \) by a certain distribution \( p^M(X, y) \) that is defined on \( U \). It can be shown that, when \( |D| \to +\infty \), for each \( X \), \( h(X) = \sum_{y \in U(X)} p^M(y|X)y \) minimizes the cross entropy loss (see proof in Appendix C). In other words, for each \( X \), the model \( h \) learns to predict the expected label vector. If the data generation method \( M \) makes \( p^M(y|X) \) uniform, i.e. \( p^M(y|X) = \frac{1}{\sigma(X)} \), the model learns the optimal estimator \( h^*(X) = \frac{1}{\sigma(X)} \sum_{y \in U(X)} y = h^*(X) \).

We present a training data generation method \( M \) that ensures \( p^M(y|X) \) to be uniform. Note there is no constraint on the distribution \( p^M(X) \). We first randomly generate the shape of \( X \) by randomly draw \( m \) (and \( n \)) from a uniform distribution \([L_m, H_m]\) (and \([L_n, H_n]\)). We provide details of how to choose \( L_m, H_m, L_n \) and \( H_n \) in Appendix F and show the trained model generalizes very well outside of the regions \([L_m, H_m]\) and \([L_n, H_n]\) in experiments. Given each sampled \( m \) and \( n \), the shape of \( X \) and \( y \) is determined, we then generate the values in \( X \) and \( y \) uniformly at random. In other words,
each element \( X[i, j] \) in the matrix \( X \) is sampled from the uniform distribution on set \{ +1, -1, 0 \} and each element \( y[i] \) in vector \( y \) is sampled from the uniform distribution on set \{ +1, -1 \}. If \( \sigma(X, y) = 1 \), we keep it as a training data point; otherwise, we randomly generate another pair; This process is repeated many times. Apparently, since \( y \) is generated uniformly, for any two different vectors \( y_1 \) and \( y_2 \) with \( \sigma(X, y_1) = \sigma(X, y_2) = 1 \), the probability of generating \( y_1 \) equals to the probability of generating \( y_2 \). Therefore, the condition that \( p^M(y | X) \) is uniform is satisfied. The data generation method is also efficient because the probability of generating a valid pair in one trial is about 0.2 (see Appendix D).

### 4.2 Model Architecture

When training on data generated by the above method, the model \( h \) learns to be the optimal estimator \( h^* \). However, how well it learns depends on the architecture design of \( h \). The input of the model \( h \) is a matrix \( X \) of size \( n \times m \) and the output of the model \( h \) is a vector \( \hat{y} \) of size \( n \). The model \( h \) should satisfy the following three properties: (1) Ability to Accept Arbitrary Input Size. The number of data points \( n \) and LFs \( m \) can be different for different datasets. The model \( h \) should be able to accept an input matrix \( X \) of arbitrary size. (2) Invariance to Permutation of LFs. Intuitively, randomly shuffling the LFs should not change the prediction of any data point. Formally, let \( P_m \) denote one arbitrary permutation of the \( m \) integers in \([0, m - 1]\). For example when \( m = 4 \), \( P_m \) could be \{3, 0, 1, 2\}. Invariance to permutation of LFs means that \( h(\hat{X}[:P_m]) = h(X) \), \( \forall P_m \). (3) Equivariance to Permutation of Data Points. Intuitively, randomly shuffling the data points should not change the prediction of each data point. Formally, equivariance to permutation of data points means that \( h(\hat{X}[P_m,:]) = h(X)[P_m] \), \( \forall P_m \) where \( P_m \) is defined similarly as \( P_m \).

Our intuition is that a graph neural network (GNN) can accept input graph of arbitrary size and is permutation equivariant to the nodes [52]. Therefore, it is possible that by somehow representing the input matrix \( X \) as a graph and then use a GNN, we can have an architecture that satisfies all the above three properties. Specifically, we first represent the input matrix \( X \) as a graph and then use a GNN. For example, the left-most matrix and graph in Figure 1 illustrate how we represent an input matrix of size \( 3 \times 2 \) as a graph. The weak label of the \( i^{th} \) data point provided by the \( j^{th} \) LF is represented as a node \( V_{i,j} \) with value \( X[i, j] \). There are two types of edges: solid yellow edge and dashed blue edge. Nodes from the same LF (i.e. same column in matrix \( X \)) are connected with solid yellow edges and nodes from the same data point (i.e. same row in matrix \( X \)) are connected with dashed blue edges. It is easy to see the graph representation \( G \) loses no information as one can easily recover \( X \) (or its permutation \( X[P_m, P_m] \)) from \( G \). In graph \( G \), if we only look at dashed blue edges, there would be \( n \) strongly connected components and each corresponds to one data point. Specifically, the strongly connected component \( SCC_i = \{ V_{i,0}, V_{i,1}, \ldots \} \) corresponds to the \( i^{th} \) data point.

![Figure 1: Overall network architecture.](image)

The idea of designing a model architecture with the mentioned three properties is that: first encode the graph with a GNN of \( K \) layers and each node \( V_{i,j} \) is encoded with embedding \( V_{i,j}^k \) at the \( k^{th} \) layer; then after the final layer, we obtain an embedding for each SCC \( i \) (i.e. each data point) by pooling all of its nodes \( \tilde{V}^{K}_{i} = \frac{1}{n_i} \sum_j V_{i,j}^K \); The embedding of each SCC \( i \) is passed to a Multilayer perceptron (MLP) to obtain the final prediction. The overall model architecture is shown in Figure 1. It’s straightforward to see that the architecture satisfies all three mentioned properties (see Appendix E.1).

We adopt the standard design of GNN. Since we have two types of edges, we perform message passing for neighboring nodes connected with different edges separately. Specifically, at the \( k^{th} \) layer
The Matrix Layer with such equivariance property is proposed in [25]. In fact, each layer of our GNN Permutation Equivariance/Invariance to Input Matrix.

Abstention (denoted by $C_0$) turn out that our trained model for binary labels can be easily used to support multi-class classification on each label matrix of each class and then all labels are obtained by $h'(X)$. There are also methods that represent the matrix as a bipartite graph where each row is represented as one type of node and each column is represented as another type of node to achieve permutation equivariance [43].

4.3 Supporting Semi-supervised/Multi-class Label Aggregation

Supporting Semi-supervised Label Aggregation with Fine-tuning. We pretrain a model $h_0$ on our synthetically generated dataset. When no ground-truth label is provided, we can infer the ground-truth label vector $y$ from $X$ in a simple forward pass on $h_0$. When a small set of ground-truth labels is provided, our method can easily incorporate the labels by fine-tuning the model on the provided labels. Let $I$ denote the set of indices of the elements in $y$ that are provided. For example, when $I = [2, 3]$, it means $y[2]$ and $y[3]$ are provided. Fine tuning is done by minimizing the loss $\sum_{i \in I} \text{CrossEntropy}(h(X)[i], y[i])$ and $h$ is initialized as the pretrained model $h_0$. After fine-tuning we obtain a model $h'$, and then all labels are obtained by $h'(X)$.

Supporting Multi-class Datasets. We have only considered the binary labels and one can extend our training data generation and network architecture to natively support multi-class tasks. However, it turns out that our trained model for binary labels can be easily used to support multi-class classification datasets by decomposing a multi-class task with $C$ classes to be $C$ one-vs-rest binary classification tasks. For multi-class tasks, we have $X[i, j] \in \{0, 1, 2, \ldots, C\}$ where 0 still denotes abstention and other numbers denote all the classes. We construct the label matrix for the $c^{th}$ class as $X_c[i, j] = 1$ if $X[i, j] = c, X_c[i, j] = 0$ if $X[i, j] \neq c$, and otherwise $X_c[i, j] = -1$. In this way, we obtain $C$ label matrices $\{X_1, \ldots, X_C\}$. We apply our pre-trained model $h_0$ on each label matrix of each class and obtain $C$ predicted probability vectors $(p_1, \ldots, p_C)$. Then, for the $i^{th}$ data point, its soft label over the $C$ classes is $\left(\frac{p_1[i]}{\sum_{j} p_1[j]}, \ldots, \frac{p_C[i]}{\sum_{j} p_C[j]}\right)$. We show in experiments this simple method works well.
5 Experiments

We evaluate from two major aspects (1) the performance of label aggregation and (2) the performance of end model trained on the aggregated labels. Additional experimental results on running time and the presence of adversarial LFs are in Appendix H.1 and Appendix H.2. The code and detailed instructions to reproduce the experiments are available in supplementary materials.

Datasets. We use all 14 classification datasets in a recent weak supervision benchmark [65] that are from diverse domains (e.g. income/sentiment/spam/relation/question/topic classification tasks). We highlight these datasets are only used for evaluation after our model is trained on synthetically generated data, and we never used these datasets during training. Table 1 shows the statistics of all datasets. Note that, the original benchmark splits each dataset to training/validation/test sets for the purpose of evaluating supervised end models. Since our paper focuses on label aggregation, there is no need to split the datasets [2, 41] and we only split the datasets when we use the generated labels to train an end model in Section 5.4. Different datasets may require different performance metrics based on their application background, so we use the metric adopted by the benchmark [65] for each dataset. All LFs are from the original authors of each dataset and all LFs are hosted in the benchmark [3].

| Dataset          | Census | IMDB   | Yelp   | YouTube | SMS    | Spouse | CDR    | Commercial | Tennis | Basketball | AGNews | TREC | SemEval | ChemProt |
|------------------|--------|--------|--------|---------|--------|--------|--------|------------|--------|------------|--------|------|---------|---------|
| #class           | 2      | 2      | 2      | 2       | 2      | 2      | 2      | 2          | 2      | 2          | 4      | 6    | 9       | 10      |
| metric           | F1     | acc    | acc    | acc     | F1     | F1     | F1     | F1          | F1     | F1         | F1     | acc  | acc     | acc     |
| #LF              | 83     | 5      | 8      | 10      | 73     | 9      | 33     | 4           | 6      | 8          | 164    | 26   | 68      | 164     |
| #Data            | 31925  | 25000  | 38000  | 1956    | 5571   | 27766  | 14023  | 8803        | 20256  | 120000     | 5965   | 2641 | 16075   | 16075   |

Baselines. We compare our method Learn Label Aggregation (LELA) to all label aggregation methods in the weak supervision benchmark [65], and a method published very recently [61]. We also include two representative methods from crowdsourcing as it has a similar setting.

- **Majority Vote (MV).** The predicted label of each data point is the most common label given by LFs.
- **Data Programming (DP)** [48]. DP uses a probabilistic graph model (PGM) where each LF is a node and the hidden ground truth is a latent variable.
- **Flyingsquid (FS)** [22]. FS also uses a PGM but gives a closed-form solution with some assumptions.
- **MeTaL** [47]. MeTaL infers the ground truth using a matrix completion model. The latest version of the popular Snorkel system [55] adopts MeTaL as its default label aggregation method.
- **NPLM** [61]. This method is also based on a PGM and assumes LFs are conditionally independent. It supports partial LFs that predict a subset of class labels and is designed to be very efficient.
- **Dawid and Skene’s method (DS)** [17]. DS models the confusion matrix of each LF with respect to the ground truth labels. This method is widely used in crowdsourcing and is the recommended method for classification tasks in a benchmark on crowdsourcing [68].
- **Enhanced Bayesian Classifier Combination (EBCC)** [36]. This method models the joint distribution of crowd workers as a mixture of multiple low dimensional tensors. This method achieves the state-of-the-art results on 17 crowdsourcing datasets [36].

There are methods that learn a label aggregation model and an end model at the same time (e.g. WeaSEL [51] and AMCL [41]). We note they are in a different setting and are not directly comparable to methods that focus on label aggregation [65, 64]. We discussed the benefits, downsides, and applications of both types of methods in related work (Section 2).

Implementation. We provide the implementation details of our method (e.g. setups and all parameters in data generation/model architecture/model training/validation) in Appendix F and implementation details of the experiments (e.g. hardware/datasets/baselines/setups) in Appendix G.

5.1 Label Aggregation Performance

The performance of all methods on all 14 datasets averaged over five runs are shown in Table 2. To maintain the table to be readable, we only show the error bars for the averaged scores. Again, for our method LELA, we note only synthetically generated data is used for training and the 14 datasets are only used to evaluate the trained model. For LELA, inference on the real datasets is deterministic, but the training process on the synthetic data has randomness, so the error bar is obtained by repeating the training process multiple times and then performing inference with different trained models.
We highlight that the performance of a supervised method random forest converges at about $lr = 0.0001$ to prevent overfitting (originally $lr = 0.001$). This is because all prior methods (except Majority Vote) require an unsupervised learning process while LELA performs prediction in a single forward pass just like Majority Vote. We note that these 14 benchmark datasets are relatively small (as creating a large benchmark dataset is expensive). In industry scenarios, LFs can be applied on millions of data points to create labels [8]. The runtime gain of LELA will be more significant and LELA will enable the LF development process to be more interactive.

### 5.2 Semi-supervised Label Aggregation

When the ground-truth labels for some data points are provided i.e. some elements in $y$ are known, LELA can also easily incorporate these labels through the fine-tuning method in Section 4.5.

For each dataset, we randomly sample $N_{gt}$ data points as the data points with known ground-truth labels and we evaluate on the remaining data points. When $N_{gt} > 0.7n$, we only select 0.7$n$ data points to keep 30% of the data for evaluation in order to have a reliable evaluation score. We vary $N_{gt}$ from 10 to 10000. Since the baseline label aggregation methods don’t support semi-supervised label aggregation, we train a random forest classifier as our baseline using the provided labels (with $X$ as the feature matrix). The hyper-parameters of random forest are selected by cross validation. When finetuning LELA, we use a smaller learning rate $lr = 0.0001$ to prevent overfitting (originally $lr = 0.001$). Intuitively, when $N_{gt}$ is small, we trust the pre-trained LELA more than the provided labels; when $N_{gt}$ is large, we trust the provided labels more than the pre-trained LELA. Therefore, we relate the number of finetuning epochs to $N_{gt}$ by setting the number of epochs as $\sqrt{N_{gt}}$.

The results are shown in Figure 2. When $N_{gt}$ is small, semi-supervised LELA can be slightly worse than unsupervised LELA (i.e. the pretrained LELA on synthetic data without fine-tuning). This is because finetuning on a small number of label data causes overfitting. When $N_{gt} > 30$, semi-supervised LELA outperforms unsupervised LELA. Random forest requires about 800 labels to match the performance of unsupervised LELA and 4000 labels to catch up with semi-supervised LELA. When $N_{gt} > 4000$, the performance of semi-supervised LELA and random forest is the same.

We highlight that the performance of a supervised method random forest converges at about 70.1 which can be seen as an empirical upperbound for any unsupervised label aggregation method. Unsupervised LELA is able to achieve 69.0, only 1.57% lower than the empirical upperbound (and the best existing method is 6.56% lower). This suggests LELA is empirically very close to the optimal unsupervised label aggregation method. Finally, we note semi-supervised LELA is also very efficient, e.g. when $N_{gt} = 10000$, the running time averaged over all datasets is 3.1 seconds for semi-supervised LELA and is 4.8 seconds for random forest.
5.3 Ablation Study

We perform ablation study in three aspects: (1) We replace our data generation method with the one proposed in [65] that was originally used to generate LFs to evaluate label aggregation models. (2) We replace our model architecture with a naive architecture (see Appendix G). (3) We replace our proposed weaker form of the better-than-random assumption in Equation 2 with the original form in Equation 1. The results are shown in Table 3. Replacing each component reduces performance. In particular, the original form of the better-than-random assumption decreases performance because the assumption that each LF is better-than-random on each class is not satisfied in the real-world datasets. In fact, every dataset has at least one LF that is worse-than-random and on average 27.7% of the LFs are worse-than-random. This verifies our choice of the weaker assumption.

5.4 End Model Performance

We use the generated labels of each method to train an end model for each dataset. We consider the two best performing baselines MeTaL and MV. We use the test split provided by the benchmark [65, 3] for each dataset because some datasets only have ground-truth labels for data points in the provided test split. We then randomly split the remaining data points to be a training set and a validation set with a 3:1 ratio. The labels in the training set and validation set are generated labels by each label aggregation method, while the labels in the test set are ground-truth labels for evaluation. Following prior work [48, 65], the probabilistic labels instead of the hard labels are used to train the end model when possible. We adopt the end models used in (and their implementations provided by) the benchmark [65, 3], i.e. a pretrained BERT model [19] for textual datasets and a multi-layer perception (MLP) for datasets with numeric features. We report the results on test set in Table 4. Again, to maintain the table to be readable, we only show the error bars for the averaged scores.

Table 4: Performance of end model trained with labels generated by each method.

| Dataset       | Census | IMDb | Yelp | Youtube | SMS | Spouse | CDR | Commercial | Basketball | AGNews | TREC | SemEval | ChemProt | Avg.  |
|---------------|--------|------|------|---------|-----|--------|-----|------------|------------|--------|------|---------|----------|-------|
| End model     | MLP    | BERT | BERT | BERT    | BERT| MLA    | BERT| MLP        | BERT       | MLA    | BERT | BERT    | BERT     | BERT  |
| MV            | 31.7   | 74.7 | 74.2 | 90.9    | 83.5| 51.6   | 62.9| 90.1       | 83.5       | 13.4   | 81.9 | 63.9    | 76.8     | 54.6  |
| MeTaL         | 11.6   | 74.4 | 70.5 | 87.1    | 87.3| 51.0   | 63.5| 88.2       | 83.5       | 14.5   | 82.0 | 63.7    | 83.1     | 52.5  |
| LELA          | 56.3   | 74.7 | 75.7 | 93.0    | 82.4| 52.3   | 64.1| 87.1       | 83.5       | 17.0   | 80.8 | 68.4    | 82.8     | 53.1  |

Our results align with those in the benchmark [65] where the end model trained on labels generated by MeTaL is slightly worse than that by MV. Overall, LELA outperforms the other two methods. On Yelp, Spouse, and SemEval, LELA tied with MV in label quality (see Table 2) but has better end model performance as LELA’s probabilistic labels can be more informative. Note the scores of the end model can be higher than that of the generated labels (as also observed in the benchmark [65] and prior work [46]) because the end model incorporates additional information from the raw data.

6 Conclusion

We present an analytical label aggregation method for weak supervision, which is optimal in the sense that it minimizes a certain form of the averaged prediction error. However, the analytical form is of exponential complexity to compute. We propose to train a model to learn the analytical method and
once learned the inference complexity of the model is only linear to input size. We design a synthetic
training data generation method that ensures the model trained on the generated data learns to be
the analytical method. We design an effective model architecture that enables the trained model to
handle arbitrary number of LFs and arbitrary number of data points and also makes the trained model
invariant/equivariant to the permutation of LFs/data points. We experimentally verify the superiority
of our method in both accuracy and efficiency with both unsupervised and semi-supervised label
aggregation settings over 14 datasets. We further show that the end model trained using the labels
generated by our method has better performance than existing methods on 14 datasets.

Acknowledgments and Disclosure of Funding

Use unnumbered first level headings for the acknowledgments. All acknowledgments go at the
end of the paper before the list of references. Moreover, you are required to declare funding
(financial activities supporting the submitted work) and competing interests (related financial activities
outside the submitted work). More information about this disclosure can be found at: https://
neurips.cc/Conferences/2022/PaperInformation/FundingDisclosure

Do not include this section in the anonymized submission, only in the final paper. You can use
the ack environment provided in the style file to automatically hide this section in the anonymized
submission.

References

[1] 2022. sklearn.ensemble.RandomForestClassifier. https://scikit-learn.org/stable/
modules/generated/sklearn.ensemble.RandomForestClassifier.html [Online; ac-
cessed 2. May 2022].

[2] 2022. Wrench Github Issue. “Question on train/val/test split when evaluating label model.”.
https://github.com/JieyuZ2/wrench/issues/27 [Online; accessed 12. May. 2022].

[3] 2022. Wrench Project Homepage. https://github.com/JieyuZ2/wrench [Online; ac-
cessed 12. May. 2022].

[4] Túlio C Alberto, Johannes V Lochter, and Tiago A Almeida. 2015. Tubespam: Comment spam
filtering on youtube. In 2015 IEEE 14th international conference on machine learning and
applications (ICMLA). IEEE, 138–143.

[5] Tiago A Almeida, José María G Hidalgo, and Akebo Yamakami. 2011. Contributions to the
study of SMS spam filtering: new collection and results. In Proceedings of the 11th ACM
symposium on Document engineering. 259–262.

[6] Abhijeet Awasthi, Sabyasachi Ghosh, Rasna Goyal, and Sunita Sarawagi. 2020. Learning from
rules generalizing labeled exemplars. arXiv preprint arXiv:2004.06025 (2020).

[7] Stephen H Bach, Bryan He, Alexander Ratner, and Christopher Ré. 2017. Learning the structure
of generative models without labeled data. In International Conference on Machine Learning.
PMLR, 273–282.

[8] Stephen H Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Cassandra Xia,
Souvik Sen, Alex Ratner, Braden Hancock, Houman Alborzi, et al. 2019. Snorkel drybell:
A case study in deploying weak supervision at industrial scale. In Proceedings of the 2019
International Conference on Management of Data. 362–375.

[9] BatsResearch. 2022. yu-aistats22-code. https://github.com/BatsResearch/
yu-aistats22-code [Online; accessed 1. May 2022].

[10] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi,
Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner,
et al. 2018. Relational inductive biases, deep learning, and graph networks. arXiv preprint
arXiv:1806.01261 (2018).
[11] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. \textit{arXiv preprint arXiv:1706.02263} (2017).

[12] Mobilize Center. 2019. \textit{Snorkel Workshop 2018: Best Practices for Improving Your Labeling Functions}. https://www.youtube.com/watch?v=mrTkus844B4 [Online; accessed 27. Apr. 2022].

[13] David PA Corney, Dyaa Albakour, Miguel Martinez-Alvarez, and Samir Moussa. 2016. What do a million news articles look like?. In \textit{NewsIR@ ECIR}. 42–47.

[14] Nilesh Dalvi, Anirban Dasgupta, Ravi Kumar, and Vibhor Rastogi. 2013. Aggregating crowdsourced binary ratings. In \textit{Proceedings of the 22nd international conference on World Wide Web}. 285–294.

[15] Nilaksh Das, Sanya Chaba, Renzhi Wu, Sakshi Gandhi, Duen Horng Chau, and Xu Chu. 2020. Goggles: Automatic image labeling with affinity coding. In \textit{Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data}. 1717–1732.

[16] Allan Peter Davis, Cynthia J Grondin, Robin J Johnson, Daniela Sciaky, Benjamin L King, Roy McMorran, Jolene Wiegers, Thomas C Wiegers, and Carolyn J Mattingly. 2017. The comparative toxicogenomics database: update 2017. \textit{Nucleic acids research} 45, D1 (2017), D972–D978.

[17] Alexander Philip Dawid and Allan M Skene. 1979. Maximum likelihood estimation of observer error-rates using the EM algorithm. \textit{Journal of the Royal Statistical Society: Series C (Applied Statistics)} 28, 1 (1979), 20–28.

[18] Frederik Michel Dekking, Cornelis Kraaikamp, Hendrik Paul Lopuhaä, and Ludolf Erwin Meester. 2005. \textit{A Modern Introduction to Probability and Statistics: Understanding why and how}. Springer. 181–190 pages.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. \textit{arXiv preprint arXiv:1810.04805} (2018).

[20] Jared A Dunnmon, Alexander J Ratner, Khaled Saab, Nishith Khandwala, Matthew Markert, Hersh Sagi, Ragesh Sagi, Roger Goldman, Christopher Lee-Messer, Matthew P Lungren, Daniel L Rubin, et al. 2020. Cross-modal data programming enables rapid medical machine learning. \textit{Patterns} 1, 2 (2020), 100019.

[21] Jason A Fries, Paroma Varma, Vincent S Chen, Ke Xiao, Heliodoro Tejeda, Priyanka Saha, Jared Dunnmon, Henry Chubb, Shiraz Maskati, Madalina Fiterau, et al. 2019. Weakly supervised classification of aortic valve malformations using unlabeled cardiac MRI sequences. \textit{Nature communications} 10, 1 (2019), 1–10.

[22] Daniel Fu, Mayee Chen, Frederic Sala, Sarah Hooper, Kayvon Fathalian, and Christopher Ré. 2020. Fast and three-rious: Speeding up weak supervision with triplet methods. In \textit{International Conference on Machine Learning}. PMLR, 3280–3291.

[23] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In \textit{International conference on machine learning}. PMLR, 1263–1272.

[24] Godfrey Harold Hardy, John Edensor Littlewood, George Pólya, György Pólya, et al. 1988. "Hölder's Inequality and Its Extensions.”. In \textit{Inequalities, 2nd ed.} Cambridge university press, Chapter 2.7-2.8, 21–26.

[25] Jason Hartford, Devon Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. 2018. Deep models of interactions across sets. In \textit{International Conference on Machine Learning}. PMLR, 1909–1918.

[26] HazyResearch. 2022. flyingsquid. https://github.com/HazyResearch/flyingsquid [Online; accessed 26. Apr. 2022].
[27] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid O Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2019. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. arXiv preprint arXiv:1911.10422 (2019).

[28] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. Averaging weights leads to wider optima and better generalization. arXiv preprint arXiv:1803.05407 (2018).

[29] David R Karger, Sewoong Oh, and Devavrat Shah. 2014. Budget-optimal task allocation for reliable crowdsourcing systems. Operations Research 62, 1 (2014), 1–24.

[30] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836 (2016).

[31] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).

[32] Ron Kohavi et al. 1996. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid.. In Kdd, Vol. 96. 202–207.

[33] Martin Krallinger, Obdulia Rabal, Saber A Akhondi, Martín Pérez Pérez, Jesús Santamaría, Gael Pérez Rodríguez, Georgios Tsatsaronis, Ander Intxaurrondo, José Antonio López, Umesh Nandan, et al. 2017. Overview of the BioCreative VI chemical-protein interaction Track. In Proceedings of the sixth BioCreative challenge evaluation workshop, Vol. 1. 141–146.

[34] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the loss landscape of neural nets. Advances in neural information processing systems 31 (2018).

[35] Xin Li and Dan Roth. 2002. Learning question classifiers. In COLING 2002: The 19th International Conference on Computational Linguistics.

[36] Yuan Li, Benjamin Rubinstein, and Trevor Cohn. 2019. Exploiting worker correlation for label aggregation in crowdsourcing. In International Conference on Machine Learning. PMLR, 3886–3895.

[37] Yinghao Li, Pranav Shetty, Lucas Liu, Chao Zhang, and Le Song. 2021. BERTifying the Hidden Markov Model for Multi-Source Weakly Supervised Named Entity Recognition. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 6178–6190.

[38] Pierre Lison, Jeremy Barnes, Aliaksandr Hubin, and Samia Touileb. 2020. Named Entity Recognition without Labelled Data: A Weak Supervision Approach. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 1518–1533.

[39] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies. 142–150.

[40] Jose Mathew, Meghana Negi, Rutvik Vijjali, and Jairaj Sathyarayana. 2021. DeFraudNet: An End-to-End Weak Supervision Framework to Detect Fraud in Online Food Delivery. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 85–99.

[41] Alessio Mazzetto, Cyrus Cousins, Dylan Sam, Stephen H Bach, and Eli Upfal. 2021. Adversarial Multi Class Learning under Weak Supervision with Performance Guarantees. In International Conference on Machine Learning. PMLR, 7534–7543.

[42] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP. 1003–1011.
[43] Federico Monti, Michael Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. *Advances in neural information processing systems* 30 (2017).

[44] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. 2021. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment* 2021, 12 (2021), 124003.

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[46] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, Vol. 11. NIH Public Access, 269.

[47] Alexander Ratner, Braden Hancock, Jared Dunnmon, Frederic Sala, Shreyash Pandey, and Christopher Ré. 2019. Training complex models with multi-task weak supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4763–4771.

[48] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems* 29 (2016).

[49] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. 2019. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237* (2019).

[50] Wendi Ren, Yinghao Li, Han Ting Su, David Kartchner, Cassie Mitchell, and Chao Zhang. 2020. Denoising multi-source weak supervision for neural text classification. *arXiv preprint arXiv:2010.04582* (2020).

[51] Salva Rühling Cachay, Benedikt Boecking, and Artur Dubrawski. 2021. End-to-End Weak Supervision. *Advances in Neural Information Processing Systems* 34 (2021).

[52] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B Wiltschko. 2021. A gentle introduction to graph neural networks. *Distill* 6, 9 (2021), e33.

[53] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. 2015. Incremental knowledge base construction using deepdive. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, Vol. 8. NIH Public Access, 1310.

[54] Vaibhav B Sinha, Sukrut Rao, and Vineeth N Balasubramanian. 2018. Fast dawid-skene: A fast vote aggregation scheme for sentiment classification. *arXiv preprint arXiv:1803.02781* (2018).

[55] snorkel team. 2022. snorkel. [https://github.com/snorkel-team/snorkel](https://github.com/snorkel-team/snorkel) [Online; accessed 26. Apr. 2022].

[56] snorkel team. 2022. snorkel-extraction. [https://github.com/snorkel-team/snorkel-extraction/blob/master/snorkel/learning/gen_learning.py](https://github.com/snorkel-team/snorkel-extraction/blob/master/snorkel/learning/gen_learning.py) [Online; accessed 26. Apr. 2022].

[57] sukrutrao. 2022. Fast-Dawid-Skene. [https://github.com/sukrutrao/Fast-Dawid-Skene](https://github.com/sukrutrao/Fast-Dawid-Skene) [Online; accessed 26. Apr. 2022].

[58] Paroma Varma, Frederic Sala, Ann He, Alexander Ratner, and Christopher Ré. 2019. Learning dependency structures for weak supervision models. In *International Conference on Machine Learning*. PMLR, 6418–6427.

[59] Renzhi Wu, Prem Sakala, Peng Li, Xu Chu, and Yeye He. 2021. Demonstration of panda: a weakly supervised entity matching system. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2735–2738.
[60] Sen Wu, Luke Hsiao, Xiao Cheng, Braden Hancock, Theodoros Rekatsinas, Philip Levis, and Christopher Ré. 2018. Fonduer: Knowledge base construction from richly formatted data. In Proceedings of the 2018 international conference on management of data. 1301–1316.

[61] Peilin Yu, Tiffany Ding, and Stephen H. Bach. 2022. Learning from Multiple Noisy Partial Labelers. In Artificial Intelligence and Statistics (AISTATS).

[62] Yue Yu, Simiao Zuo, Haoming Jiang, Wendi Ren, Tuo Zhao, and Chao Zhang. 2020. Fine-tuning pre-trained language model with weak supervision: A contrastive-regularized self-training approach. arXiv preprint arXiv:2010.07835 (2020).

[63] yuan li. 2022. https://github.com/yuan-li/truth-inference-at-scale [Online; accessed 1. May 2022].

[64] Jieyu Zhang, Cheng-Yu Hsieh, Yue Yu, Chao Zhang, and Alexander Ratner. 2022. A Survey on Programmatic Weak Supervision. arXiv preprint arXiv:2202.05433 (2022).

[65] Jieyu Zhang, Yue Yu, , Yujing Wang, Yaming Yang, Mao Yang, and Alexander Ratner. 2021. WRENCH: A Comprehensive Benchmark for Weak Supervision. In Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, J. Vanschoren and S. Yeung (Eds.), Vol. 1.

[66] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. Advances in neural information processing systems 28 (2015).

[67] Yuchen Zhang, Xi Chen, Dengyong Zhou, and Michael I Jordan. 2014. Spectral methods meet EM: A provably optimal algorithm for crowdsourcing. Advances in neural information processing systems 27 (2014).

[68] Yudian Zheng, Guoliang Li, Yuanbing Li, Caihua Shan, and Reynold Cheng. 2017. Truth inference in crowdsourcing: Is the problem solved? Proceedings of the VLDB Endowment 10, 5 (2017), 541–552.

[69] Wenxuan Zhou, Hongtao Lin, Bill Yuchen Lin, Ziqi Wang, Junyi Du, Leonardo Neves, and Xiang Ren. 2020. Nero: A neural rule grounding framework for label-efficient relation extraction. In Proceedings of The Web Conference 2020. 2166–2176.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Appendix A.1
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix A.2
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Appendix B and Appendix C
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix B and Appendix C

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix D and Appendix E

14
(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report the scores averaged over datasets with error bars for label aggregation in the last column in Table 2. We plot the error bars for semi-supervised label aggregation in Figure 2. We report the scores averaged over datasets with error bars for the end model in the last column in Table 4.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See the Hardware paragraph in Appendix G.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Table 1.
   (b) Did you mention the license of the assets? [Yes] See readme in supplemental material.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [Yes] See the Datasets paragraph in Appendix G.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] See the Datasets paragraph in Appendix G.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Limitations and Broader Impact

A.1 Limitations

In certain cases, one might be able to know the quality of the LFs or the dependency structure of the LFs as a prior. Our method would not be able to incorporate the prior information. It would be an interesting future work to extend our method to be able to accept these types of prior information as additional input. Another limitation is memory consumption during inference. For some existing methods, they can first perform unsupervised learning on some data points to learn the model parameters, then inference can be done one data point by one data point. Therefore, these methods have low memory consumption during inference. In contrast, our method performs inference in a single forward pass for all data points all in once. When the dataset is extremely huge, inference might not fit in GPU memory. One can alleviate the issue by performing inference on CPU, or one can divide the dataset into smaller batches and perform inference batch by batch.

A.2 Broader Impact

The lack of labeled training data is a major impediment for the adoption of machine learning techniques to a boarder range of applications. Weak supervision is a promising direction to lift the impediment. Our work significantly improves the accuracy and efficiency of label aggregation in weak supervision. We hope our work will enable more applications to use machine learning techniques.

Potential negative social impact: since weak supervision is able to create labeled data from unlabeled data easily, the tech companies will have the incentive to collect more user data which might have a negative impact on privacy.

B Optimalities of the Uniform Distribution

We aim to approximate an unknown distribution \( p(y|X) \) (which can be different in different applications) with an fixed distribution \( q(y|X) \). Since both distributions are defined on a finite set \( U(X) \), we can use the probabilities of the elements in \( U(X) \) to represent each of the two distributions.
Specifically, we represent \( p(y|X) \) as \( p = \{p_1, \ldots, p_{|U(X)|}\} \) and \( q(y|X) \) as \( q = \{q_1, \ldots, q_{|U(X)|}\} \). Similarly, we denote the uniform distribution (i.e. \( p'(y|X) = \frac{1}{|U(X)|} \)) as \( u = \{u_1, \ldots, u_{|U(X)|}\} \).

Apparently, \( \forall i \), we have \( 0 \leq p_i \leq 1 \), \( 0 \leq q_i \leq 1 \), \( u_i = \frac{1}{|U(X)|} \), \( \sum_i p_i = 1 \), \( \sum_i q_i = 1 \), and \( \sum_i u_i = 1 \). Using the uniform distribution \( u \) to approximate \( p \) is the optimal in both the worst case and the average case. The two optimalities are formally defined as the following:

**1) Worst-case Optimal:** The uniform distribution \( u \) has the minimum maximum distance to the unknown distribution \( p \):

\[
u = \operatorname{argmin}_q \max_p \operatorname{dist}(p, q)
\]

where "\( \operatorname{dist} \)" can be the KL divergence or \( L_\alpha \) distance \( \forall \alpha > 1 \). Note the conventional name is "\( L_p \)" distance, but to avoid reusing the same notation \( p \) for different meanings, we use the name "\( L_\alpha \)" distance instead.

**2) Average-case Optimal:** The uniform distribution \( u \) has the minimum expected KL divergence to the unknown distribution \( p \) under a mild assumption. Let \( \mathcal{P}(p) \) denote the probability of the unknown distribution being a specific distribution \( p \) (e.g. \( \mathcal{P}(u) \) would be denoting the probability of the unknown distribution being the uniform distribution i.e. \( p(p = u) \)). Formally:

\[
u = \operatorname{argmin}_q \mathbb{E}_p [\operatorname{KL}(p, q)] = \operatorname{argmin}_q \int_p \operatorname{KL}(p, q) \mathcal{P}(p) dp
\]

under the assumption that \( \mathcal{P}(p) \) is centrally symmetric, formally:

\[
\int_p \mathcal{P}(p) dp = u
\]

We provide a formal proof for the two optimalities in the following:

**Proof for Worst-case Optimal.**

*Proof. *We first prove Equation 6 for KL divergence.

\[
\max_p \operatorname{KL}(p, q) = \max_p \sum_i p_i \log \frac{p_i}{q_i} = \max_p \sum_i p_i \log p_i + p_i \log \frac{1}{q_i}
\]

The maximum of the first term is zero, as \( p_i \log p_i \leq 0 \) due to \( p_i \geq 0 \) and \( \log p_i \leq 0 \). The maximum is obtained when there is a \( j \) such that \( p_j = 1 \) and \( p_i = 0, \forall i \neq j \);

\( j \) also comes into play in the maximum of the second term. We have \( \sum_i p_i \log \frac{1}{q_i} \leq \sum_i p_i \max_i \log \frac{1}{q_i} = \max_i \log \frac{1}{q_i} \). Therefore, the maximum of the second term is \( \max_i \log \frac{1}{q_i} \), which is obtained when \( j = \arg \max_i \log \left( \frac{1}{q_i} \right), p_j = 1 \) and \( p_i = 0, \forall i \neq j \).

We can see that the maximum of both terms is achieved at the same time with \( j = \arg \max_i \log \left( \frac{1}{q_i} \right), p_j = 1 \) and \( p_i = 0, \forall i \neq j \). The maximum value is \( \max_i \log \frac{1}{q_i} \).

Therefore,

\[
\max_p \operatorname{KL}(p, q) = \max_i \log \frac{1}{q_i}
\]

\[
= \log \frac{1}{\min_i q_i}
\]

\[
\geq \log |U(X)|
\]

The inequality is because \( \min_i q_i \leq \frac{1}{|U(X)|} \) (otherwise, \( \sum_i q_i > \sum_i \frac{1}{|U(X)|} > 1 \)). The equality of the inequality is obtained when \( q \) is the uniform distribution, i.e. \( q = u \). Therefore, \( u = \operatorname{argmin}_q \max_p \operatorname{KL}(p, q) \).
Next, we prove Equation 6 for $L_\alpha$ distance $\forall \alpha > 1$. The $L_\alpha$ distance is defined as:

$$L_\alpha(p, q) = (\sum_i |p_i - q_i|^\alpha)^{1/\alpha} \tag{11}$$

Take the derivative of $L_\alpha(p, q)$ with respect to a $p_i$:

$$\frac{\partial L_\alpha(p, q)}{p_i} = \begin{cases} \frac{1}{\alpha} (\sum_i |p_i - q_i|^\alpha)^{1/\alpha - 1} \alpha(p_i - q_i)^{\alpha - 1} & \text{if } p_i \geq q_i \\ \frac{-1}{\alpha} (\sum_i |p_i - q_i|^\alpha)^{1/\alpha - 1} \alpha(q_i - p_i)^{\alpha - 1} & \text{otherwise} \end{cases} \tag{12}$$

This means if $p_i - q_i \geq p_j - q_j$, $\frac{\partial L_\alpha(p, q)}{p_i} \geq \frac{\partial L_\alpha(p, q)}{p_j}$. Therefore, replacing $p_i, p_j$ with $p_i + \delta, p_j - \delta$ where $\delta > 0$ increases $L_\alpha(p, q)$ and eventually replacing $p_i, p_j$ with $p_i + p_j, 0$ increases $L_\alpha(p, q)$. Let $k = \arg \max_i p_i - q_i$. For each pair $(p_k, p_i) i \neq k$, we replace $p_k$ to be $p_k + p_i$ and $p_i$ to be 0 and eventually we have $p_k = 1$ and $p_i = 0, i \neq k$:

$$\left( \sum_i |p_i - q_i|^\alpha \right)^{1/\alpha} \leq \left( \sum_{i \neq k} |q_i|^\alpha + |1 - q_k|^\alpha \right)^{1/\alpha} \tag{13}$$

Apparently, when $k = \arg \min_i q_k$, the right hand side is further maximized. Without loss of generality, we can assume $q_1 \leq q_2 \leq \cdots \leq q_{U(X)}$. Therefore:

$$\max_p L_\alpha(p, q) = (1 - q_1)^\alpha + \sum_{i \geq 1} q_i^\alpha \tag{14}$$

By the Hölder’s inequality [24]:

$$\sum_{i \geq 1} q_i^\alpha \geq (|U(X)| - 1)^{1-\alpha}(\sum_{i \geq 1} q_i)^\alpha = (|U(X)| - 1)^{1-\alpha}(1 - q_1)^\alpha \tag{15}$$

where equality in the inequality is obtained when $q_2 = \cdots = q_{U(X)}$. Therefore:

$$\max_p L_\alpha(p, q) \geq (1 - q_1)^\alpha + (|U(X)| - 1)^{1-\alpha}(1 - q_1)^\alpha)^{1/\alpha} \tag{16}$$

$$\geq ((1 - \frac{1}{|U(X)|})^\alpha + (|U(X)| - 1)^{1-\alpha}(1 - \frac{1}{|U(X)|})^\alpha)^{1/\alpha}$$

where the second inequality is because $(1 - q_1)^\alpha + (|U(X)| - 1)^{1-\alpha}(1 - q_1)^\alpha)^{1/\alpha}$ monotonically decreases as $q_1$ increases and we have $q_1 \leq \frac{1}{|U(X)|}$ because $q_1$ is the minimum in $q$, i.e. $q_1 \leq q_2 \leq \cdots q_{U(X)}$.

In summary, the minimum of $\max_p L_\alpha(p, q)$ is obtained when $q_2 = q_3 = \cdots = q_{U(X)}$ and $q_1 = \frac{1}{|U(X)|}$, which means $q_1 = q_2 = q_3 = \cdots = q_{U(X)} = \frac{1}{|U(X)|}$. In other words, $q = u$. Therefore, $u = \arg \min_q \max_p L_\alpha(p, q)$.

Proof for Average-case Optimal.

Proof.

$$E_p[KL(p, q)] = \int_p KL(p, q) \mathcal{P}(p) dp$$

$$= \int_p \mathcal{P}(p) \sum_i p_i \log \frac{p_i}{q_i} dp$$

$$= \int_p \mathcal{P}(p) \sum_i p_i \log p_i dp - \int_p \mathcal{P}(p) \sum_i p_i \log q_i dp \tag{17}$$
Since the first term is irrelevant to \( q \), we have:

\[
E_p[KL(p, q)] = \text{constant} - \int_p \mathcal{P}(p) \sum_i p_i \log q_i dp
= \text{constant} - \sum_i \log q_i \int_p \mathcal{P}(p)p_i dp = \text{constant} - \sum_i u_i \log q_i
\]  

(18)

where the last equation is by the assumption that \( \mathcal{P}(p) \) is centrally symmetric, i.e. \( \int_p \mathcal{P}(p)p dp = u \). Therefore:

\[
E_p[KL(p, q)] = \text{constant} - \sum_i u_i \log q_i
= \text{constant} - \frac{1}{|U(X)|} \sum_i \log q_i
= \text{constant} - \frac{1}{|U(X)|} \log \prod_i q_i
\]

(19)

\[
\geq \text{constant} - \frac{1}{|U(X)|} \log \left( \frac{\sum_i q_i}{|U(X)|} \right)
= \text{constant} + \log(|U(X)|)
\]

where the inequality is the inequality of arithmetic and geometric means. The equality of the inequality is obtained when \( q_1 = q_2 = \cdots = q_{|U(X)|} = \left( \frac{1}{|U(X)|} \right) \), i.e. \( q = u \). Therefore, \( u = \arg\min_q E_p[KL(p, q)] \).

**C Expected Label Vector Minimizes Cross Entropy Loss**

When training a model \( h(X) \) on data set \( D = \{(X_1, y_1), \ldots \} \). When \( |D| \to +\infty \), for each \( X \):

\[
h(X) = \sum_{y \in U(X)} p^M(y|X) y
\]

(20)

minimizes the cross entropy loss function.

**Proof.** For each \( X \), let \( D(X) \) denote the subset \( \{(X, y_1^1), (X, y_2^2), \ldots \} \) of \( D \). The cross entropy loss on \( D(X) \) is:

\[
- \sum_{i=1}^{|D(X)|} \sum_{j=1}^n y_i^j[j] \log(h(X)[j]) + (1 - y_i^j[j]) \log(1 - h(X)[j])
\]

(21)

where \( n \) denotes the number of rows in \( X \). By taking derivative and setting it to zero, the above equation is minimized when:

\[
h(X)[j] = \frac{\sum_{i=1}^{|D(X)|} y_i^j[j]}{|D(X)|}, \forall j
\]

(22)

By the law of large numbers \[18\], when \( |D| \to +\infty \) (so that \( |D(X)| \to +\infty \)), \( h(X)[j] = \frac{\sum_{i=1}^{|D(X)|} y_i^j[j]}{|D(X)|} = E(y[j]|X) = \sum_{y \in U(X)} p^M(y|X) y[j] \) for \( \forall j \). This means \( h(X) = \sum_{y \in U(X)} p^M(y|X) y \).

**D Probability of Generating a Valid Pair**

To simplify our analysis, in the following, we only consider \( y \) that contains both \(-1\) and \(+1\), which has a probability \( p_0 = 1 - \frac{2}{M} \). \( p_0 \approx 1 \) when \( n \geq 100 \) (When generating data, we sample \( n \) from \([L_n, H_n] = [100, 2000] \) which we explain in Appendix \[\] .)
Let $S$ denote the set of all possible pairs of $(X, y)$. $S$ is made up by three subsets: $U = \{(X, y)\mid \sigma(X, y) = 1\}$, $S_e = \{(X, y)\mid \sum_{j=0}^{m-1} g(X, y, j, 1) = \frac{m}{2}\}$ and $S_c = S - U - S_e$. Apparently, $S_e$ is also made up by three subsets, i.e., $S_e = S_{e_1} \cup S_{e_2} \cup S_{e_3}$ where $S_{e_1} = \{(X, y)\mid \sum_{j=0}^{m-1} g(X, y, j, -1) < \frac{m}{2}\}$ and $S_{e_2} = \{(X, y)\mid \sum_{j=0}^{m-1} g(X, y, j, 1) > \frac{m}{2}\}$ and $S_{e_3} = \{(X, y)\mid \sum_{j=0}^{m-1} g(X, y, j, -1) > \frac{m}{2}\} \text{ and } \sum_{j=0}^{m-1} g(X, y, j, 1) < \frac{m}{2}\}$. 

**Lemma 2.** $|S_{e_1}| = |S_{e_2}| = |S_{e_3}| = |U|$. 

**Proof.** For each element $(X, y)$ in $S_{e_1}$, $\sum_{j=0}^{m-1} g(X, y, j, -1) < \frac{m}{2}$ and $\sum_{j=0}^{m-1} g(X, y, j, 1) > \frac{m}{2}$. We can flip $X[y = 1, :]$ to be $-X[y = 1, :]$ and flip $X[y = -1, :]$ to be $-X[y = -1, :]$. After flipping, we obtain pair $(X', y)$, and apparently $(X', y) \in U$. This means for each element in $S_{e_3}$ there is a corresponding element in $U$, so we have $|S_{e_1}| \leq |U|$. Similarly, for each element in $U$, we can do flipping to get an element in $S_{e_3}$, so we also have $|U| \leq |S_{e_1}|$. Therefore, $|U| = |S_{e_1}|$. Similarly, one can show that $|U| = |S_{e_2}|$ and $|U| = |S_{e_3}|$. \qed

By Lemma [2], $|S_{e_1}| = |S_{e_2}| + |S_{e_3}| = 3|U|$. When $m$ is odd, apparently, $|S_e| = 0$. Therefore, the probability of a randomly generated pair being valid is:

$$p((X, y) \in U|m) = \frac{|U|}{|S_e|} = \frac{|U|}{|U| + |S_e| + |S_c|} = \frac{1}{4}$$ (23)

Next, we consider when $m$ is even. To simplify our analysis, approximately, $p(g(X, y, j, -1)) = \frac{1}{2}$ and $p(g(X, y, j, 1)) = \frac{1}{2}$. This is because the probability that the number of correct elements exactly equal to the number of incorrect elements for each class is extremely small due to $n$ being relatively large. Therefore, we have:

$$p((X, y) \in S_e|m) = \left(\frac{m/2}{2^m}\right)$$ (24)

Therefore:

$$p((X, y) \in U|m) = \left(1 - \frac{m/2}{2^m}\right) \frac{1}{4}$$ (25)

Since $m$ is uniformly sampled from $[L_m, H_m] = [2, 60]$ (which we explain in Appendix F), we have:

$$p((X, y) \in U) = \sum_{m, m\%2=1, L_m \leq m \leq H_m} \frac{1}{H_m - L_m} \frac{1}{4} + \sum_{m, m\%2=0, L_m \leq m \leq H_m} \frac{1}{H_m - L_m} \left(1 - \frac{m/2}{2^m}\right) \frac{1}{4}$$

$$= \frac{1}{2} \times \frac{1}{4} + \sum_{m, m\%2=0, L_m \leq m \leq H_m} \frac{1}{H_m - L_m} \left(1 - \frac{m/2}{2^m}\right) \frac{1}{4}$$

$$\approx 0.232$$ (26)

This means the probability of generating a valid pair in one trial is about 0.232.

### E. Discussions

#### E.1 The Proposed Architecture Satisfies the Three Properties

To see how the proposed architecture in Figure 1 satisfies the three properties mentioned in the beginning of Section 4.2. First, GNN accepts arbitrary input size, so $X$ can be of any size. Second, GNN is permutation equivariant to the nodes, so the output embeddings of GNN are equivariant to the permutation of data points and LFs. After average pooling for each data point over all LFs (each SCC with dashed blue edges), the network is invariant to the permutation of LFs and is still equivariant to the permutation of data points.

19
E.2 Crowdsourcing Methods for Weak Supervision

The two crowdsourcing methods have the worst performance in Table 2. The reason that crowdsourcing methods don’t work well on weak supervision datasets has not been investigated or discussed in prior work, and we provide our conjecture. First, the label matrix in crowdsourcing tends to be extremely sparse as there can be many crowd workers while each crowd worker might annotate a few data points then quit [68]. In contrast, in weak supervision, each LF is applied to each data point. Second, since crowd workers are humans, the labels provided by the crowd workers tend to have higher accuracy; In contrast, a LF when applied on data unseen by the LF developer can predict very noisy labels. In other words, the existing crowdsourcing methods are designed to work in the sparse scenario with weak labels of higher accuracy, so that they don’t work well in the weak supervision setting with a denser and noisier label matrix.

F Implementation Details of LEIA

Data Generation. When generating each pair \((X, y)\), we first randomly generate \(n\) and \(m\), the number of rows/columns of matrix \(X\). Note \(n\) is the number of data points and \(m\) is the number of LFs. As we mentioned, we first sample \(n\) and \(m\) uniformly from \([L_n, H_n]\) and \([L_m, H_m]\) respectively. We set \([L_n, H_n] = [100, 2000]\) where \(L_n = 100\) is because typically there are at least hundreds of data points otherwise it is not necessary to write LFs as one can just manually label all data points and we set \(H_n = 2000\) due to memory limit during model training. We set \([L_m, H_m] = [2, 60]\) where \(L_m = 2\) is because when there is only one LF there is no need to aggregate and we set \(H_m = 60\) due to memory limit during model training; We highlight our trained model generalizes well to number of LFs and number of data points (see Table 1) that are not in the region \([L_m, H_m]\) and \([L_n, H_n]\) as we have shown in experiments. Once we have \(n\) and \(m\), we invoke the method mentioned in Section 4.1 to generate \((X, y)\).

Since our data is synthetically generated, there is no need to generate a fixed training set. Our training data is generated on the fly, i.e. during training when the data loader fetches the next pair of \((X, y)\), a new pair is immediately generated and returned.

Model Architecture. We implement our model architecture in Pytorch [45]. We use \(K = 4\) layers of GNN. The embedding dimension of GNN is 32, i.e. each node in the graph is encoded with a 32 dimensional embedding. The final MLP consists of three linear layers; the first two linear layers use Relu activation and the last linear layer uses Sigmoid activation.

Model Training. We use the Adam optimizer [31]. We set amsgrad to be true for better convergence [49] and keep all other parameters as the default values provided by Pytorch (e.g. learning rate \(lr = 0.001\)). We use a batch size of 50, i.e. each batch consists of 50 pairs of generated \((X, y)\). We tested different batch sizes of 16 and 250 and observed no meaningful difference. We train our model until training loss converges (loss doesn’t decrease in \(10^4\) iterations), which takes about one day with \(5 \times 10^4\) iterations on a K80 GPU. Note one iteration means one gradient update/one batch, and we don’t have the notion of epoch as training data is generated on-the-fly for each batch.

Validation. We also need to prevent our model from overfitting the training set. We highlight that, different from typical ML settings where one gets access to a validation set that is similar to the test set, in our setting we have no validation set that is similar to the test set. Again, when training our model, the real test datasets are unseen and we only have access to synthetic data. Our intuition is that when the model overfits the synthetically generated training set \(D\), its performance will be poor on data that is different from the training set, for example, on another synthetic dataset \(D'\) that is generated in a different way. We synthetically generate the validation set \(D'\) with size \(|D'| = 100\) according to the generation method proposed in [65]; In this method, LFs are independent from each other conditioned on the ground-truth label.

We note that the way we use the validation set is also different from a typical setting. We train the model until training loss converges (this typically requires about \(5 \times 10^4\) iterations), and repeat 10 runs (i.e. train our model 10 times from scratch). We then select the run with the highest averaged validation accuracy over all iterations (as validation accuracy might fluctuate over iterations); We use the learned model at the final iteration of the selected run in our experiments. We provide our reasoning of doing this: (1) We do not use the validation set to do early stopping (i.e. to select the best iteration in a run). In a typical ML setting, the validation set is used to select the best epoch/iteration.
This is possible because in a typical ML setting the validation set is similar to the test set and the validation set provides very strong signal towards which iteration is a good iteration for the test set. In our case, the validation set $D'$ can be very different from the test set, thus the selected iteration based on $D'$ might not be a good iteration for the test set. (2) We use the validation set to select the best run. We observed that at different runs, the curve of validation accuracy vs number of iterations can be different (e.g. the two runs in Figure 3), so the test accuracy of the model in different runs can be different. We would like to select the best run using the validation set $D'$. Intuitively, one run with better validation accuracy on average over all iterations is stably better (e.g. the yellow run in Figure 3), so we select the run with an best averaged validation accuracy over iterations. As an example, for the two runs in Figure 3, although the highest validation accuracy of purple run can be higher than that of the yellow run, the yellow run has a higher averaged validation accuracy over iterations and is much more stable, so we select the yellow run. We also observed this run to have a less degree of fluctuation in validation accuracy, as shown in Figure 3. This suggests the model converges at a flat minima, which is known to generalize better [34,30,28].

One natural question is that why it is possible to select the best run but it is not possible to select the best iteration. The reason is that selecting the best run out from 10 runs require much less information than selecting the best iteration out from $5 \times 10^4$ iterations. Since the validation set $D'$ can be very different from the test set, the information provided by $D'$ is very limited.

An interesting phenomenon in the validation accuracy curve in Figure 3 for the yellow run is that validation accuracy first increases then decreases and finally increases. This is known as the Epoch-wise double descent phenomenon [44].

G Implementation Details of Experiments

Hardware. All of our experiments were performed on a machine with a 2.20GHz Intel Xeon(R) Gold 5120 CPU, a K80 GPU and with 96GB 2666MHz RAM.

Datasets. We use the datasets prepared by the wrench benchmark on Github [3] [65]. All the datasets and LFs are publicly released by previous work [65]. All datasets do not contain any personally identifiable information [65].

Originally, each dataset include three files "train.json", "valid.json" and "test.json". Following the suggestion in a reported issue of the wrench benchmark [2], we combine all three files to get a single matrix $X$ and single ground-truth label vector $y$ for the experiments on label aggregation. We then split the datasets using the original split for the experiment on end model (Section 5.4). The information of the LFs as well as the raw data for each dataset can be found in the wrench benchmark project on Github [3].

Baselines. For each baseline, we use existing open-source implementations. The implementations of DS, DP, FS, MeTaL, EBCC, and NPLM are from [57], [56], [26], [55], [63], and [9] respectively. For baselines that require class weights as priors, we report the best results from using uniform weights and using the weights estimated by majority vote.

Setup in Semi-supervised Label Aggregation. When sampling $N_{gt}$ data points as the data points with known labels, we make sure that each class has at least two data points. For random forest, we use the scikit-learn implementation [1]. When training the random forest classifier, we use five fold
cross validation to perform grid search for the "max_depth" parameter in range [2, 4, 8, 16, 32, None] and the "min_samples_split" parameter in range [2, 5]. We repeat five runs and report results with error bars in Figure 2.

**Setup in Ablation Study.** For the naive model architecture, the input is a flattened vector of a fixed size matrix 2000 × 50 and the network has several linear layers.

**Setup in End Model Experiment.** When training the end model, the training set and validation set both use generated labels by each method and the test set uses ground-truth labels. For the two end model MLP and BERT, we use the implementation provided by the benchmark [3][65]. We use grid search to tune hyper-parameters for each end model based on validation set performance. We use the same search space as the benchmark [65], as summarized in Table 5. We repeat five runs and report the scores averaged over runs in Table 4.

### Table 5: Hyper-parameters and search space for the end models.

| End Model | Hyper-parameter | Description | Range |
|-----------|-----------------|-------------|-------|
| MLP       | batch_size      | batch size  | 32,128,512 |
| MLP       | lr              | learning rate | 1e-5,1e-4,1e-3,1e-2,1e-1 |
| MLP       | weight_decay    | weight decay | 1e-5,1e-4,1e-3,1e-2,1e-1 |
| MLP       | ffn_num_layer   | number of MLP layers | 2 |
| MLP       | ffn_hidden_size | hidden size of MLP layers | 100 |
| BERT      | batch_size      | batch size  | 16,32 |
| BERT      | lr              | learning rate | 2e-5,3e-5,5e-5 |

### H Additional Experiment Results

#### H.1 Running time

Table 6 shows the running time of all label aggregation methods on all datasets. When measuring the running time, we use GPU for methods that support GPU (MeTaL, NPLM, and LELA).

### Table 6: Running time (seconds) of label aggregation on all datasets

| Dataset | Census | IMDB | Yelp | Youtube | SMS | Spouse | CDR | Commercial | Tennis | Basketball | AGNews | TREC | SemEval | ChemProt | AVG. |
|---------|--------|------|------|---------|-----|--------|-----|------------|--------|------------|--------|-------|---------|----------|------|
| MV      | <0.1   | <0.1 | <0.1 | <0.1   | <0.1| <0.1   | <0.1| <0.1       | <0.1   | <0.1       | <0.1   | <0.1 | <0.1   | <0.1    | <0.1 |
| DP      | 147.8  | 18.8 | 40.5 | 2.5     | 14.4| 8.4    | 29.5| 8.5        | 10.0   | 14.9       | 225.0  | 100.8 | 190.2  | 213.0    | 73.2 |
| FS      | 21.1   | 1.7  | 3.7  | 0.2     | 3.2 | 0.8    | 3.7 | 0.6        | 0.6    | 14.9       | 22.1   | 16.3 | 69.0   | 26.4     | 12.2 |
| MeTaL   | 0.5    | 0.3  | 0.4  | 0.4     | 0.4 | 0.3    | 0.4 | 0.4        | 0.4    | 0.5        | 3.6    | 4.6   | 3.6    | 1.2      |      |
| NPLM    | 15.7   | 4.0  | 5.7  | 0.4     | 2.2 | 1.8    | 6.3 | 11.2       | 1.5    | 3.4        | 27.9   | 5.4   | 3.4    | 12.1     | 7.2  |
| EBCC    | 3.9    | 5.1  | 52.5 | 2.2     | 2.8 | 2.3    | 5.8 | 3.0        | 2.5    | 6.0        | 18.0   | 9.0   | 9.8    | 84.8     | 14.8 |
| LELA    | 0.1    | 0.1  | 0.1  | 0.1     | 0.1 | 0.2    | 0.2 | 0.3        | 0.2    | 0.3        | 0.2    | 0.3   | 0.2    | 0.2      |      |

### H.2 Adversarial LFs

We test the behavior of all methods with the presence of noisy/adversarial LFs that are worse than random. Following the setup in prior work [51], we create an adversarial LF for each dataset, then duplicate it \( \beta m \) times \((m \text{ is the original number of LFs})\) and append the labels of the \( \beta m \) LFs to the label matrix. We vary \( \beta \) from 0 to 1 and report the label aggregation results in Figure 4. As \( \beta \) increases, the performance of all methods decreases. In particular, the performance of the best performing baseline MeTaL decreases more significantly than other baselines. LELA always achieves the best results.
Figure 4: Averaged score vs amount of adversarial LF duplicates.