Lecture Notes on Weak Supervision

Mayee Chen, Frederic Sala, Chris Ré

November 10, 2019

1 Motivation

Modern ML is data-hungry. To train models, we need labeled data—and lots of it. Acquiring unlabeled data is easier than ever, but labeling this data tends to be an expensive and slow process. Instead, techniques relying on weaker forms of supervision have become popular.

Given data without labels, we will rely on such noisy *weak supervision sources* to approximate what the true labels are. There are many ways to build such sources:

- Heuristic rules,
- Encoding constraints,
- Off-the-shelf models,
- Lookups in knowledge bases,

and many others. The key intuition is that while we do not observe the true label at all, by obtaining multiple noisy sources of information, we can learn a model over the sources and the unobserved label, enabling us to infer this label. This places weak supervision somewhere between supervised and unsupervised learning.

The idea of assembling multiple noisy voters brings crowdsourcing to mind. In fact, crowdsourcing can often be modeled by weak supervision approaches—we discuss this connection in Appendix A. Crowdsourcing often studies the accuracy of voters, while assuming that their voting noise is independent. We begin our discussion of weak supervision from the same starting point. Afterwards we show how we can extend our approach to more challenging settings.

2 Problem Setup

Suppose we have a training set with n data points $x_1, \dots, x_n \in \mathcal{X}$ and corresponding labels $y_1, \dots, y_n \in \mathcal{Y}$. Here, the points (X, Y) are drawn from some distribution D. However, we only observe the sampled x's, and not any of the corresponding y's. That is, Y is a *latent* *variable.* For simplicity, we discuss the case where $Y \in \{-1, +1\}$, though we will mention how to extend this setup beyond binary classes.

Instead of accessing the labels, we have *m weak supervision sources*, which are functions from $\mathcal{X}$ to $\{-1, +1\}$ that output labels—noisy estimates of Y. The random variables that are outputs of these functions are denoted by $\lambda_1, \ldots, \lambda_m$, and their random vector is $\lambda = [\lambda_1, \ldots, \lambda_m]^T$. We sometimes also call these variables labeling functions (LFs). Note that we observe a labeling function output for each data point, so we see $m \times n$ samples in total.

Our goal is to *learn a model* over the sources and the latent variable Y. Such a model will enable us to compute a probabilistic estimate of the true label $P(Y|\lambda)$ given the observed labeling function outputs. Intuitively, this model must capture, for example, the accuracies and correlations of the labeling functions.

Training an End Model Once we have learned a label model and can compute probabilistic training data $P(Y|\lambda)$, we can then use it to train an end model—using standard supervised learning. In fact, as long as we learn the label model, and this model corresponds to the true distribution, we can then bound the performance of the end model (i.e., its ability to generalize) and compare it to the model trained on data with the true labels, as in the supervised case. In fact, the end model trained with our probabilistic data can have the same asymptotic performance as if we really had the true labels. Thus the goal is to figure out how to learn the label model.

Sampling Our goal is to figure out the behavior of unobserved variables (like the true label Y) based on observed variables (like the sources λ_i). In fact our core tool will be information about the rates of agreement and disagreement between observed terms. If we had infinite data, we would know, for example, moments like $\mathbb{E}[\lambda_i]$ and $\mathbb{E}[\lambda_i \lambda_j]$ and covariances like $\mathbf{Cov}[\lambda_i, \lambda_j]$. These are *population*-level statistics. We develop our algorithms as if we had access to these (i.e., with infinite data).

However, in practice, we only have access to n samples, and we have to empirically estimate the terms above. This induces *sampling error.* This sampling error can be bounded (with tools from probability like tail bounds), and we can propagate these bounds through the algorithm to determine *rates*—which tell us the error of our algorithm (on average) in terms of the number of samples we see. Below, we show everything algebraically, with infinite data, and then comment on sampling at the end.

3 Learning the Accuracies of Weak Supervision Sources

We start with the setting where the sources have independent noise, i.e., the conditionally independent label model. In this case, the only parameters we need to learn are the accuracies of the sources.

Afterwards, we get rid of the conditional independence assumption by allowing dependencies. We will later tackle even more complex models. The key thread running throughout these approaches is exploiting *independencies.*

3.1 Fully Independent Noise

An important setting is that of uncorrelated noise, similar to the crowdsourcing model described earlier. Although any two sources λ_i and λ_j are not independent, we assume that $\lambda_i Y$ and $\lambda_j Y$ are independent for all distinct i, j, meaning that pairwise noise is uncorrelated. Later, in Section 4.1 we will connect this notion to the conditional independence $\lambda_i \perp\!\!\!\perp \lambda_j | Y$, which can be encoded by graphical models.

Note that since the λ_i's and Y are in $\{-1, +1\}$, $\lambda_i Y$ is 1 when λ_i votes on Y correctly, and -1 otherwise. Moreover, $\mathbb{E}\left[\lambda_i Y\right] = 2P(\lambda_i = Y) - 1$, so we are justified in thinking of $\mathbb{E}\left[\lambda_i Y\right]$ as a way to measure a type of *accuracy*, but scaled to $[-1, +1]$. We write it as a_i. Then, $a_i = 1$ indicates that λ_i always outputs correct labels, $a_i = -1$ indicates that λ_i is always incorrect, and $a_i = 0$ indicates that λ_i is a random guess.

How shall we exploit independencies to estimate $a_i = \mathbb{E}\left[\lambda_i Y\right]$? From our independence assumption,

$$\mathbb{E}\left[\lambda_i \lambda_j\right] = \mathbb{E}\left[\lambda_i Y \lambda_j Y\right] = \mathbb{E}\left[\lambda_i Y\right] \mathbb{E}\left[\lambda_j Y\right] = a_i a_j,$$

where first we used the fact that $Y^2 = 1$, and then independence.

Note that while we have no way of directly observing a_i, the product of $a_i a_j$ is just the rate at which a pair of LFs vote together—and this is observable.

Now, since we can see $a_i a_j$, we can also look at a third source λ_k, allowing us to see $a_i a_k$ and $a_j a_k$. This gives us enough information to solve for the accuracies up to sign, e.g., $|a_i|, |a_j|, |a_k|$—we'll see how to solve this system with three equations and three variables below.

The sign tells us whether each LF has a better-than-random, random, or worse-than-random chance of voting on Y correctly. If we make the assumption that all of our LFs have a better than random probability of voting correctly—even as low as $\frac{1}{2} + \varepsilon$, then we know the signs are positive, and we have identified the accuracies. We will later significantly relax this assumption.

Now we show how to formalize this approach to obtaining the accuracies.

Accuracy Recovery Procedure Let M be the $m \times m$ first moment matrix of the LFs, defined where

$$M_{ij} = \begin{cases} \mathbb{E}[\lambda_i \lambda_j] & i \neq j, \\ 1 & i = j. \end{cases}$$

Solving with SGD We can use the observable M matrix to solve for the accuracies of the labeling functions. Let a be the vector with the a_i's. Now, consider the covariance matrix of the $\lambda_i Y$ terms. This covariance, by assumption, is diagonal, and for these diagonal entries, we know that $Var(\lambda_i Y) = \mathbb{E}\left[\lambda_i Y \lambda_i Y\right] - \mathbb{E}\left[\lambda_i Y\right]^2 = 1 - a_i^2$. This means that it is $I - \text{diag}(a^2)$, where $\text{diag}(a^2)$ is computed through element-wise squaring. We also know that elements of the covariance matrix are equal to $\mathbb{E}[\lambda_i Y \lambda_j Y] - \mathbb{E}[\lambda_i Y]\mathbb{E}[\lambda_j Y] = M_{ij} - a_i a_j$ for any i, j, so overall the covariance matrix can be written as $M - aa^T$. Setting these equal, we get that

$$M = I - \text{diag}(a^2) + aa^T$$

It is possible to solve this using SGD or other optimization methods by computing

$$a^* = \text{argmin}_a ||M - I + \text{diag}(a^2) - aa^T||_F.$$

Solving the System However, there is a more direct way of computing a exactly by writing out the system of equations with the off-diagonal entries of M.

$$\begin{aligned} a_1 a_2 &= M_{12} \\ a_1 a_3 &= M_{13} \\ &\vdots \\ a_{m-1} a_m &= M_{m-1,m} \end{aligned}$$

While it is a possibility to use all $O(m^2)$ equations to obtain the m accuracies, a simpler approach is to select disjoint sets of triplets of sources and solve exactly for three accuracies at a time to avoid redundancy and potentially conflicting information in M. Define T_i to be a set of three labeling functions such that $T_i(1), T_i(2), T_i(3)$ represent their indices for $i \in 1, 2, \ldots, \lceil \frac{m}{3} \rceil$. When m is divisible by 3, all T_i will be disjoint; otherwise, there will be an extra T_i that includes some labeling functions whose accuracies have already been computed. Then we have

$$\begin{aligned} a_{T_i(1)} a_{T_i(2)} &= M_{T_i(1)T_i(2)}, \\ a_{T_i(1)} a_{T_i(3)} &= M_{T_i(1)T_i(3)}, \\ a_{T_i(2)} a_{T_i(3)} &= M_{T_i(2)T_i(3)}. \end{aligned}$$

It is easy to solve for a_{T_i}:

$$|a_{T_i(1)}| = \sqrt{\frac{|M_{T_i(1)T_i(2)}||M_{T_i(1)T_i(3)}|}{|M_{T_i(2)T_i(3)}|}}, |a_{T_i(2)}| = \sqrt{\frac{|M_{T_i(2)T_i(3)}||M_{T_i(2)T_i(1)}|}{|M_{T_i(1)T_i(3)}|}},$$
$$\text{and } |a_{T_i(3)}| = \sqrt{\frac{|M_{T_i(1)T_i(3)}||M_{T_i(2)T_i(3)}|}{|M_{T_i(1)T_i(2)}|}}.$$

Recoverability What assumptions are needed for the above procedure to work? We will discuss two types of conditions, involving the minimum (absolute value) of an accuracy, and the signs of accuracies—that is, dealing with symmetries.

First, note that we divide by some M_{ij} to solve for $|a|$, so when $M_{ij} \approx 0$ then our corresponding element of $|a|$ cannot be calculated.

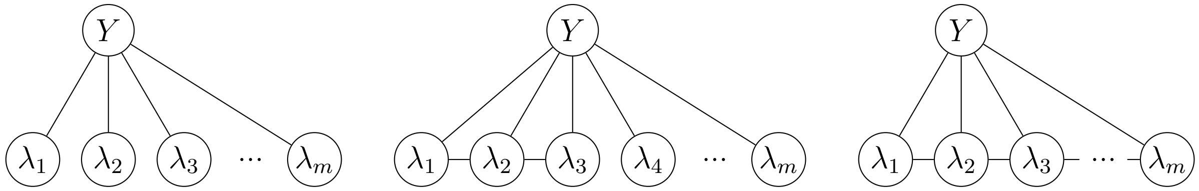

Figure 1: Left: Conditionally independent weak supervision sources; Middle: Adding dependencies. Right: Chain graph.

When can this take place? Recall that $M_{ij} = a_i a_j$, so if $M_{ij} = 0$, then one of a_i and a_j must be 0. That is, we only run into trouble precisely when $P(\lambda_i = Y) = 0.5$, e.g. a random guess. Thus we require that the a_j's are bounded away from 0.

Later we will discuss the sampling case, where we do not have access to the true values of M but an empirical estimate, $\hat{M}$, from our data. Then, even if the a_i's are bounded away from 0, due to sampling noise we might have some $\hat{M}_{ij} = 0$. Fortunately, this is not a problem either: with high probability, $\hat{M}_{ij}$ will be within ε of the value of $M_{ij} \neq 0$. More on the sampled first moment matrix $\hat{M}$ will be discussed in Section 5.

Since the system of equations only provides a straightforward way of computing $|\hat{a}|$, we will also need to find the sign of each accuracy. Fortunately, when all labeling functions are conditionally independent given Y, we only need to know the sign of one accuracy. For example, if we know if a_1 is positive or negative, we can use $a_1 a_2 = M_{12}$, $a_1 a_3 = M_{13}, \ldots, a_1 a_m = M_{1m}$ to recover all other signs. This is much less restrictive—we no longer need each accuracy to be better than random chance.

What if we don't know even a single sign? We only need to break one symmetry: is the sum of the signs smaller or greater than 0? Thus we could add the following weak assumption: require that $\sum_i a_i > 0$, and still recover.

3.2 Adding Dependencies

We now consider the more general case where *not* all pairs of λ_i, λ_j are conditionally independent given Y. Now we have correlations: dependencies between LFs. These are natural: often LFs use similar information or principles, and thus no longer have independent noise.

One way to represent these dependencies is via graphical models. Let $G = (V, E)$. V is the set of random variables $\{\lambda_1, \ldots, \lambda_m\} \cup \{Y\}$, and E is the edge set. When our LFs have dependencies, we put an edge between them. Naturally, we make all of the LFs connected to Y. We shall be more formal in the following section.

The independent noise case is then just the left of Figure 1. Once we start adding dependencies, we have cases like the center graph. A more challenging case is the *chain* graph on the right.

How can we handle the middle case? Suppose the graph can be split up into d conditionally independent subgraphs of weak sources such that the vertices $V \backslash \{Y\}$ are partitioned into the sets $V_1, \ldots, V_d$. This means that for two of these sets V_a and V_b, $\lambda_i \perp\!\!\!\perp \lambda_j | Y$ for any $\lambda_i \in V_a$

and $\lambda_j \in V_b$. If we define the set of edges among the LFs to be $E^Y = \{(u, v) \in E : u, v \neq Y\}$, this is equivalent to there being no path using E^Y between λ_i and λ_j.

If $d \geq 3$, we can apply the same method of selecting conditionally independent triplets of LFs where their accuracies are independent. Rather than having $\lceil \frac{m}{3} \rceil$ sets of triplets however, we will have $\max_i |V_i|$ sets, because no two LFs in the same subgraph can have their accuracies computed by one triplet set and thus must each use separate sets to estimate their accuracies. Note that $\max_i |V_i| \leq m - d + 1$ in the case of all other subgraphs besides the largest one being single vertices, so we're only using at most 3 times the number of triplet sets as the full conditional independence case.

This general case requires more information about the signs of the accuracies, however. Given the sign of the accuracy of a LF in V_i, we will be able to directly compute the accuracies of the LFs in all other subgraphs, but not of the other LFs in V_i. Therefore, the minimum number of signs we need to recover all remaining signs of accuracies is $\min_i |V_i|$. This expression can be 1 when there is one LF conditionally independent of all other LFs given Y, but it can also be as much as $\frac{m}{3}$ when we have equal-sized subgraphs.

4 Weak Supervision, Graphical Models, and the Inverse Covariance Matrix

Next we relax our assumption of there being conditionally independent subgraphs and handle more complicated distributions and settings by exploiting the properties of graphical models and exponential families. For example, note the chain graph on the right of Figure 1; the previous methods fail because we have no independent subgraphs. We can also handle more challenging cases, like the multiclass labels. Before we begin, we'll briefly introduce some basics from the theory of probabilistic graphical models.

4.1 Undirected Graphical Models

Graphical models are ways to express multivariate probability distributions in a convenient way. They have a deep statistical and algorithmic theory. There are two ways to think of such undirected graphical models. Both involve a graph $G = (V, E)$, where V indicates the random variables we are working with $x_1, \ldots, x_m$ and E involves a type of dependency between a pair of variables.

Gibbs fields and MRFs The first concept, the Gibbs field, operates over the *cliques* of G; these are fully connected subsets of nodes. The set of such cliques is C. We create functions called clique potentials ψ_i for each clique $c_i \in C$; these are functions from the variables involved in each clique to the positive reals. We can think of these functions as defining a certain weight for a configuration of the variables they work on. Note, however, that they do not need to sum to 1 over all configurations.

Next, we define the density for the distribution as

$$f(x_1, \dots, x_n) = \frac{1}{Z} \prod_{c_i \in C} \psi_i(c_i), \tag{1}$$

where Z is selected to ensure that this is a valid probability distribution. That is, $Z = \sum_x \prod_{c_i \in C} \psi_i(c_i)$. Note that (1) specifically defines the *factorization* of the distribution. There is a trivial way to describe a Gibbs field: one giant clique over the entire graph. This does not help reduce the complexity of the distribution, however. Note also that we can write (1) as an energy function: $f(x) = 1/Z \exp\left(-\sum_{c_i \in C} f_i(c_i)\right)$.

A second approach is called a Markov random field (MRF). In this case, the graph encodes Markov properties: if $\mathcal{N}(x_i)$ denotes the set of neighbors of x_i in the associated graph G, then we have that

$$x_i \perp\!\!\!\perp x_j \mid \mathcal{N}(x_i),$$

for $x_j \notin \mathcal{N}(x_i)$.

These two approaches are equivalent for positive densities, as shown by the Hammersley-Clifford theorem (for example, check out [10]). Let's see a simple example of how this works. Consider a zero-mean multivariate Gaussian where $x = [x_1, x_2]^T$ and $x \sim \mathcal{N}(0, \Sigma)$. Let's say that Σ is diagonal, so that our variables our independent. Then, recall that the density is

$$f_X(x_1, x_2) = \frac{1}{\sqrt{(2\pi)^2|\Sigma|}} \exp\left(-\frac{1}{2} x^T \Sigma^{-1} x\right).$$

What is the graph here? We have $V = \{x_1, x_2\}$ and $E = \emptyset$. Let's see why the independence property (i.e., a Markov property coming from the MRF point of view) is the same as a factorization property (from the Gibbs field point of view) with respect to the graph G:

The energy function interpretation of the Gibbs field tells us the factorization inside the exp should give us the sum $f_1(x_1) + f_2(x_2)$. This is because there are two cliques: $\{x_1\}$ and $\{x_2\}$. The MRF property tells us that our variables are independent, which is equivalent to zero covariance; that is $f(x_1, x_2) = f(x_1)f(x_2)$, which means that $\Sigma^{-1}_{12} = \Sigma^{-1}_{21} = 0$ in our density.

Note that these are equivalent: if we require that $x^T \Sigma^{-1} x = \Sigma^{-1}_{11} x_1^2 + \Sigma^{-1}_{22} x_2^2 + (\Sigma^{-1}_{12} + \Sigma^{-1}_{21}) x_1 x_2$ is everywhere equal to $f_1(x_1) + f_2(x_2)$, by taking derivatives, we obtain $\Sigma^{-1}_{12} + \Sigma^{-1}_{21} = 0$. Since these are equal (our covariance matrix is symmetric), they are both zero. Note that we needed symmetry, otherwise we could have $\Sigma^{-1}_{12} = -\Sigma^{-1}_{21}$.

The Hammersley-Clifford theorem formalizes these equivalences between Gibbs fields and MRFs for other distributions and more general graphs—the idea is that factorizing according to a graph G is equivalent to the Markov properties. Note how the inverse covariance matrix played a role as well; we shall later see how to make it useful for weak supervision.

Junction trees In addition to being compact ways to represent distributions that have independencies/factorize, graphical models allow for efficient algorithms to perform inference, marginalization, etc. We briefly discuss a useful tool for such tasks, the *junction tree*.

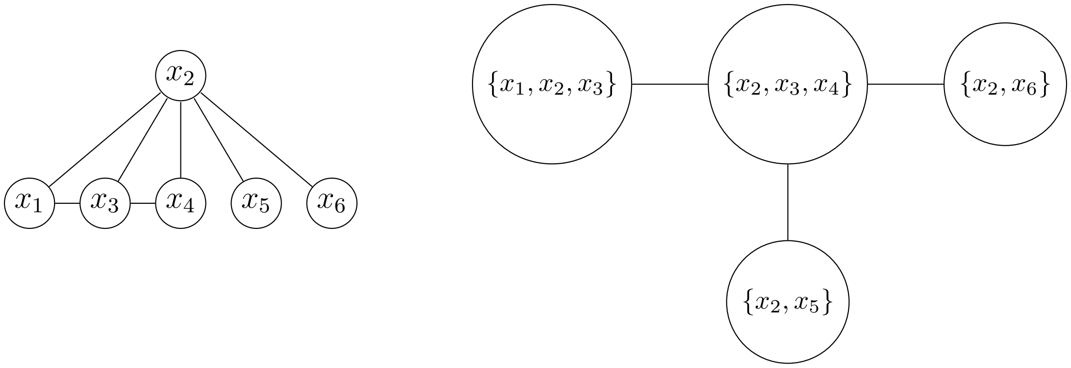

Figure 2: Left: undirected graph G; Right: junction tree T for G.

Junction trees arise from doing operations on the cliques of the graph G; the idea is to have an efficient way to move through the graph when marginalizing.

Junction trees are built from the original graph. First, each node of the tree T is now a cluster that contains one or more nodes in G. Next, each clique in G is part of some cluster in G. Finally, for any pair of clusters that contain some node x_i, each of the clusters on the path connecting the pairs contains x_i (the *running intersection* property).

This is easier to understand by example. Consider Figure 2. Note that the maximal cliques on the left are $\{x_1, x_2, x_3\}, \{x_3, x_4, x_2\}, \{x_2, x_5\}, \{x_2, x_6\}$. Each of these has become a cluster in the tree T on the right, so that there are 4 nodes. The edges here are set to ensure the running intersection property holds: For example, x_2 is contained in the left-most node and the bottom node, so it must be in each node on the path, in this case the $\{x_2, x_3, x_4\}$ node.

Finally we note a couple of useful facts. First, there is always a junction tree that follows from a having *triangulated* graph G, one where there is no cycle of length ≥ 4 without a chord breaking it up. Indeed, this is the case for the graph G on the left in Figure 2. Lastly, the edges can be thought of as the intersections of the clusters in the tree T , these are referred to as *separator sets*.

4.2 Weak Supervision Models

After this brief interlude, we return to weak supervision. First, we model our setup. See [1] for more details.

Model Setup We model the joint distribution of $\lambda_1, \lambda_2, \ldots, \lambda_m, Y$ via an exponential family/MRF with associated graph $G = (V, E)$ with $V = \{\lambda_1, \ldots, \lambda_m\} \cup \{Y\}$. For our binary setting, this produces what is known as an Ising model:

$$f_G(\lambda_1, \ldots, \lambda_m, y) = \frac{1}{Z} \exp\left(\sum_{\lambda_i \in V} \theta_i \lambda_i + \sum_{(\lambda_i, \lambda_j) \in E} \theta_{i,j} \lambda_i \lambda_j + \theta_Y y + \sum_{\lambda_i \in V} \theta_{Y,i} y \lambda_i \right), \quad (2)$$

The Ising model has only pairwise interactions; that is, there is no monomial inside the exponent term above that has more than two variables. Note how (2) is also written as the energy function formulation of the Gibbs field we discussed in Section 4.1 (in terms of cliques and clique potentials). On the other hand, it is also an MRF. For more details on these ideas, [6] is a good reference.

We can use a more compact notation for (2). We write the variables as a vector $x = [\lambda_1, \dots, \lambda_m, Y]^T$. Next, we also use similar notation for the θ parameters: we let $A = [\theta_1, \dots, \theta_m, \theta_Y]^T$ and B to be a $(m+1) \times (m+1)$ matrix where $B_{i,j} = \theta_{i,j}$ for $(\lambda_i, \lambda_j) \in E$, $B_{i,m+1} = B_{m+1,i} = \theta_{Y,i}$ for $1 \leq i \leq m$ and 0 otherwise. Then, (2) can be written as

$$f_G(x) = \frac{1}{Z} \exp\left(A^T x + x^T B x\right), \tag{3}$$

which shows the more common expression for distributions that are part of an exponential family.

Above, the θ's are called the canonical or natural parameters, and Z is the partition function (recall that this is a normalization term that ensures the density is valid). The partition function Z satisfies

$$Z = \sum_{x \in \{-1,+1\}^{m+1}} \exp\left(A^T x + x^T B x\right).$$

As we alluded to above, one of the nice features of the approach above is that we can model more complex settings than we could in the previous section. For example, by including additional terms in (2), we can learn models with $\lambda_i \in \{-1, 0, 1\}$, so that 0 indicates an *abstain*. We can include more classes for Y (beyond binary) and even consider multitask settings. We can also learn class-conditional accuracies. However, for simplicity, we present the binary case below.

Recall that in the prior section, we assumed that the noise of each labeling function was uncorrelated, e.g. $\lambda_i Y \perp\!\!\!\perp \lambda_j Y$ for all i, j, which allowed us to write off-diagonal entries of M as products of accuracies. We can recover this setting: if we set $\theta_i = 0$ (i.e., no unary potentials), we can prove the uncorrelated noise property directly from the model (2).

Goal Our goal is to learn the parameters of the model. On one hand, we could directly learn the θ canonical parameters. However, in most cases it is equivalent to learn what are called the *mean* parameters, which are also sufficient to characterize the model. The mean parameters μ are the expectations of the *sufficient statistics* of the model, which, in our case, are all of the terms with θ's attached to them $(\lambda_i, Y, \lambda_i Y, \lambda_u \lambda_v)$. So, we simply need to learn

- $\mathbb{E}[\lambda_i]$ and $\mathbb{E}[\lambda_i \lambda_j]$,
- $\mathbb{E}[Y]$ and $\mathbb{E}[Y \lambda_i]$.

The first two of these are immediate: we observe the quantities. The latter two are the challenging cases, since we don't observe Y. In the following, we will mainly focus on ways to estimate $\mathbb{E}[Y\lambda_i]$, which are exactly the accuracies we discussed earlier. In fact there are ways to estimate the *class balance* $\mathbb{E}[Y]$ without ever seeing samples of Y. In more complex settings, we will also need to estimate parameters over cliques.

Graph Structure in the Inverse Covariance Matrix Our approach is going to involve estimating certain quantities related to the inverse covariance matrix of the model defined by (2). These quantities, combined with structural information about the inverse covariance matrix, will give us enough information to estimate the parameters of the label model. We start by explaining what this structural information is.

To understand the intuition behind the connection of the inverse covariance matrix and the graph G, let's return to our zero-mean multivariate normal distribution. Let Σ be its covariance matrix, so that $X \sim \mathcal{N}(0, \Sigma)$. We no longer require Σ to be independent. The density is

$$f_X(x) = \frac{1}{\sqrt{(2\pi)^m |\Sigma|}} \exp\left(-\frac{1}{2}x^T\Sigma^{-1}x\right).$$

Note that this expression is just like (3), with $A = 0$ and $B = -\frac{1}{2}\Sigma^{-1}$. Since a multivariate normal is indeed an MRF and a Gibbs field, as we discussed earlier, it both factorizes and encodes independencies. Thus, we can obtain the graph G directly from the canonical parameters, which are simply entries of $B = -\frac{1}{2}\Sigma^{-1}$. We see that if $\theta_{i,j} = 0$ and thus $B_{i,j} = 0$, we must have also that $\Sigma^{-1}_{i,j} = 0$. That is, the inverse covariance matrix reflects the structure of G and is hence *graph-structured*: there is no edge between a pair of variables iff the corresponding entry in the inverse covariance matrix is 0.

This beautiful property was easy to derive for Gaussians precisely because the inverse covariance matrix Σ^{-1} directly expresses the canonical parameters[1] θ, but this is not necessarily the case with other distributions, including discrete distributions like the ones we are working with. Fortunately, there is an elegant result for Ising and other discrete exponential models as well, developed in [3].

To present this result, we need a little bit more notation. Let us write K for the inverse covariance matrix for the model in (2). Now we use the junction tree notions we introduced in Section 4.1.

First, we will triangulate our graph. Next, we consider the resulting junction tree T; recall that a separator S is the intersection of adjacent clusters in T (e.g., edges in the junction tree). Then $S = C_1 \cap C_2$ for clusters C_1 and C_2, for example. Next, consider the sufficient statistic Ψ_S associated with S; in the binary case, if $C_1 = \{\lambda_1, \lambda_2, Y\}$, $C_2 = \{\lambda_2, \lambda_3, Y\}$, then $S = \{\lambda_2, Y\}$ and $\Psi_S = \lambda_2 Y$. By augmenting the covariance matrix Σ with all such Ψ_S, then the resulting inverse matrix K' will exhibit the graph-structured property.

Proposition 1 $K' :=$ ***Cov*** $[X, \Psi]$ *is graph-structured.*

[1] That is, the relationship between the mean and canonical parameters for zero-mean Gaussian MRFs is extremely simple: they are inverses. In general, this does not hold, and the relationship could be very complex.

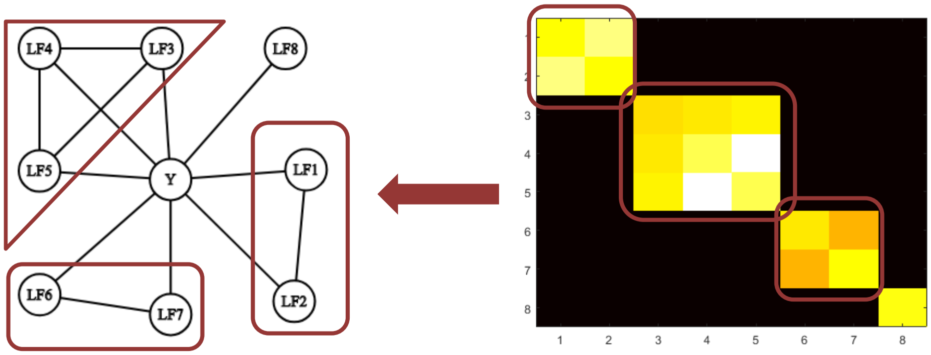

Figure 3: Left: graph G for weak supervision. Right: K_O, corresponding inverse covariance matrix. This is an example of a weak supervision label model. There are eight LFs in four blocks: one triplet, two pairs, and one singleton. $\{Y\}$, the latent variable, is the only separator, so $K = \Sigma^{-1}$, the inverse covariance matrix shows the graph structure, as shown to the right. We only show K_O, the submatrix consisting of the sources. Note that the black areas are 0: there is no edge between the different groups of LFs, so their entries in the inverse covariance are zero.

Note that if our model satisfies the *singleton separator set* assumption, then each S only contains one variable (that is, our junction tree here is trivial). These S's are already in the covariance matrix, so K itself is graph-structured, without further augmentation. An example of this idea is shown in Figure 3. Note that each of the LFs are organized as cliques, and the intersection of these cliques is simply $\{Y\}$, so that we are in the singleton separator set regime. Then, K is graph-structured, as shown on the right of Figure 3.

4.3 Learning the Label Model With The Inverse

The material in this section is presented in [1]. We want to learn the parameters of our model given access the m weak supervision sources and no ground truth labels. Let $O = \{\lambda_1, \dots, \lambda_m\}$ be the observed labels from the weak supervision sources, and $S = \{Y\}$ be the unobserved latent variable. We want to take advantage of the known structure of graph G. In Appendix **??**, we'll show how to learn the graph directly. Let us see how to connect G to items we can observe.

First, let's set up some additional notation:

$$\textbf{Cov}\left[O \cup \mathcal{S}\right] := \Sigma = \begin{bmatrix} \Sigma_O & \Sigma_{O\mathcal{S}} \\ \Sigma_{O\mathcal{S}}^T & \Sigma_{\mathcal{S}} \end{bmatrix}.$$

Next, we write

$$\Sigma^{-1} := K = \begin{bmatrix} K_O & K_{O\mathcal{S}} \\ K_{O\mathcal{S}}^T & K_{\mathcal{S}} \end{bmatrix}.$$

Then, using Prop. 1, K is *graph-structured*: there is no edge between λ_i and λ_j in G when the corresponding term in Σ^{-1} is 0.

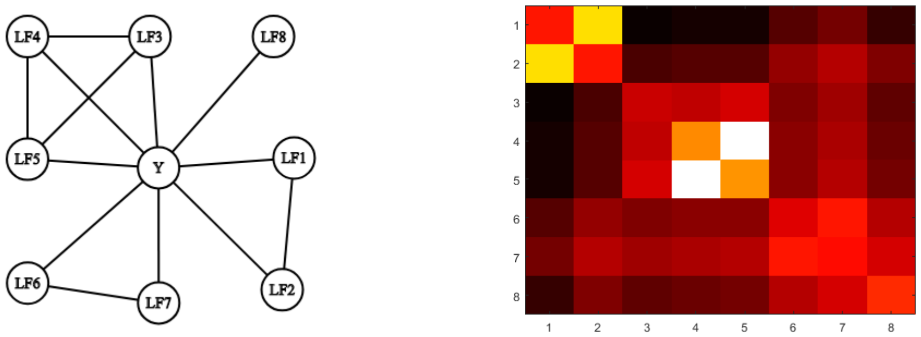

Figure 4: Left: graph G for weak supervision. Right: Σ_O^{-1}, corresponding inverse of the observed block Σ_O of the covariance matri Σ. Note that unlike with K_O, Σ_O^{-1} does not exhibit graph structure.

What's the problem? We never know Y, so we cannot observe the full covariance matrix Σ, or the graph-structured $K = \Sigma^{-1}$. Of course, we do see Σ_O, and we can invert it. Unfortunately, $(\Sigma_O)^{-1} \neq K_O$, unless K is block-diagonal, and in our setting, it is not. We might guess that the Σ^{-1} term still has graph structure, but it does not, as shown in Figure 4.

Note that while we can recognize some of the behavior of K_O in the color pattern shown by Σ^{-1}, it is masked. For example, the term corresponding to LF7 and LF4 is clearly non-zero—without seeing the corresponding entry in K_O, we cannot say whether or not there should be an edge between these two or not.

Using Block Matrix Inversion Clearly K_O and Σ_O^{-1} are not the same, so we start by figuring out the difference. Using the block matrix inversion formula,

$$K_O = \Sigma_O^{-1} + c\Sigma_O^{-1}\Sigma_{OS}\Sigma_{OS}^T\Sigma_O^{-1}, \tag{4}$$

where $c = \left(\Sigma_S - \Sigma_{OS}^T\Sigma_O^{-1}\Sigma_{OS}\right)^{-1} \in \mathbb{R}^+$. Let $z = \sqrt{c}\Sigma_O^{-1}\Sigma_{OS}$; we can write (4) as

$$\Sigma_O^{-1} = K_O - zz^T. \tag{5}$$

Note that we are back to an equation similar to earlier approach with accuracies. We have an observable term Σ_O^{-1}, a term where we know certain entries K_O (i.e., we know the 0's in K_O from the graph G), and we know that zz^T is low-rank (in this case, it is rank-1). This information is often enough to solve the problem. An illustration of all of these terms is shown in Figure 5.

Here, by solving, we will recover z, which also enables us to recover the full Σ. This takes us back to our goal: The entries of Σ corresponding to the "missing" row involving λ_i and Y will almost immediately tell us the mean parameter $\mathbb{E}\left[\lambda_i Y\right]$. That is, $\mathbf{Cov}\left[\lambda_i, Y\right] = \mathbb{E}\left[\lambda_i Y\right] - \mathbb{E}\left[\lambda_i\right]\mathbb{E}\left[Y\right]$, and the right-hand term there consists of the class balance (which, as we mentioned, we can also estimate) and the observable mean for λ_i.

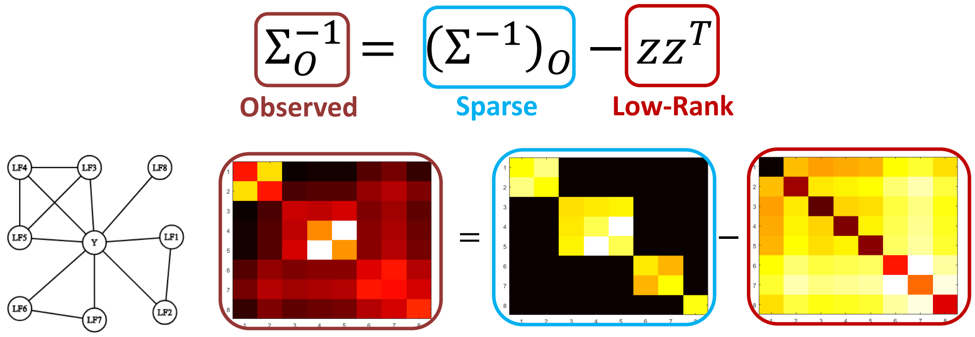

Figure 5: The difference between the observable Σ_O^{-1} term and the structured and sparse K_O block of the inverse covariance matrix is just a low-rank matrix zz^T.

Solving The Matrix Equation Thus our goal is to solve Equation (5). zz^T only has m parameters, while we observe $\Theta(m^2)$ values in Σ_O^{-1}. The challenge, however, is that we do not know all of K_O, only the entries where it is 0, corresponding to non-edges in the graph G.

We then set up a mask $\Omega = \{(i,j) \ : \ (\lambda_i, \lambda_j) \notin E(G)\}$. Now Ω consists of pairs of entries where we know K_O is 0. By applying this mask to the two matrices, we get the equation

$$0 = (\Sigma_O^{-1})_\Omega + (zz^T)_\Omega. \tag{6}$$

Now we are ready to solve. We can do this by minimizing the objective $\left|\left|\hat{\Sigma}_O^{-1} + zz^T\right|\right|_\Omega$ via SGD. This is shown in Algorithm 1. Note that the norm here is the Frobenius norm, but only computed on the values in the mask Ω. The algorithm estimates $\hat{z}$ and then computes the terms in (4), finally producing estimates of the mean parameters, as was our goal.

The Rank-One Case Note that Algorithm 1 runs for zz^T that can be higher than rank-one. This happens, for example, when Y is multi-class: if there are k classes, then Σ_{OS} and zz^T are rank-k. However, in the binary setting z is a vector and zz^T is rank-one. In this case, we can perform the optimization step in the algorithm by solving a linear system. This approach resembles the one we took for the sources in Section 3.

From (6), we have that if $(i,j) \in \Omega$, then $z_i z_j = (\Sigma_O^{-1})_{i,j}$. If we take the log of squares of both sides of this equation, we get

$$\log z_i^2 + \log z_j^2 = \log(((\Sigma_O^{-1})_{i,j})^2).$$

We have such an equation for each pair $(i,j) \in \Omega$. Then we can produce the corresponding linear system. We write M_Ω for a $|\Omega| \times m$ matrix where the row corresponding to (i,j) has a 1 in positions i and j and 0 elsewhere. We set $\ell_i = \log(z_i^2)$ and $q_{(i,j)} = \log(((\Sigma_O^{-1})_{i,j})^2)$. Then we have the linear system

$$M_\Omega \ell = q_\Omega. \tag{7}$$

Algorithm 1 Source Estimation for Weak Supervision

```
Input: Labeling rates and covariance $\mathbb{E}[O]$, $\Sigma_O$; class balance and variance $\mathbb{E}[Y], \Sigma_S$; dependency mask $\Omega$
$\hat{z} \leftarrow \text{argmin}_z \left|\left|\Sigma_O^{-1} + zz^T\right|\right|_\Omega$
$\hat{c} \leftarrow \Sigma_S^{-1}(1 + \hat{z}^T\Sigma_O\hat{z})$
$\hat{\Sigma}_{OS} \leftarrow \Sigma_O\hat{z}/\sqrt{\hat{c}}$
$\hat{\mathbb{E}}[\lambda_i Y] \leftarrow \hat{\Sigma}_{OS} + \mathbb{E}[Y]\,\mathbb{E}[O]$
Return: Mean parameters $\hat{\mathbb{E}}[\lambda_i Y]$
```

This system is now easy to solve, e.g., by least squares. Note that when solving it, we do not directly recover estimates of z_i, but rather z_i^2. That is, we recover up to sign. Next, we discuss conditions for recovery.

Conditions For Recovery and Symmetries Clearly, to solve the system (7) uniquely we must have full rank in the matrix M_Ω. In other words, we must have enough independence to have a chance to solve. We also need Σ_O to be invertible (which requires at least $n \geq m$ samples), and for the entries in $(\Sigma_O)_\Omega^{-1}$ to be bounded away from 0.

Once we've solved the system, we need to perform sign recovery: since we only obtain $\hat{z}_i^2$, we only know $|\hat{z}_i|$ and we must obtain its sign. Clearly, both $+|\hat{z}_i|$ and $-|\hat{z}_i|$ are valid solutions of (7).

We must *break the symmetry* here. We do so by relying on our conditions that at least some of the accuracies are positive (which translates into signs for the recovered $\hat{z}$'s). Note also that for each connection in Ω (these are precisely the non-edges of G), if we recover a single sign, we recover all of the other ones. This is because $z_i z_j = (\Sigma_O^{-1})_{i,j}$, and if we know the sign of two of these terms, we know the third. For example, if we are working with the case where there are no dependencies, then $\Omega = \{(i,j) \ : i < j\}$, and a single sign generates all of the other signs.

What about the higher-rank case, as in Algorithm 1? In this case, breaking symmetries is more challenging, because we are dealing with more than reflections. Note if we have some $\hat{z}$ recovered, for any orthogonal matrix O, $\hat{z}O$ is also a solution, since $(\hat{z}O)(\hat{z}O)^T = \hat{z}OO^T\hat{z}^T = \hat{z}\hat{z}^T$. In this case, we have to break both rotations as well.

5 Sampling Error

Up until this point, we have been treating terms like $E[\lambda_i], M, \Sigma_O$ as the exact *population* mean vectors, first moment matrices, and covariance matrices in our algorithms. However, in reality these are computed from our observed *sample* of data points. More precisely, rather than using some μ vector of means of the outputs of the LFs, we define $\hat{\mu} = \frac{1}{n}\sum_{i=1}^n l_i$ where l_i is a length m vector containing labels for the data point x_i. Rather than using M we must use some $\hat{M} = \frac{1}{n}\sum_{i=1}^n l_i l_i^T$, and we define $\hat{\Sigma}_O = \hat{M} - \hat{\mu}\hat{\mu}^T$.

How many data points do we need to obtain a good estimate of these population values, and how do these impact the final value of accuracy a? The way such results work are by bounding the sampling error via concentration inequalities. For example, since we often use the 3×3 subblocks R of M for our triplet approaches, we can use the Matrix Hoeffding inequality to write

$$P(||\hat{R} - R||_2 \geq \gamma) \leq 6 \exp\left(-\frac{n\gamma^2}{288}\right).$$

By integrating the bound above with respect to γ, we can obtain a bound on $\mathbb{E}\left[\|\hat{R} - R\|_2\right]$. Then, we propagate this error through our algorithm from Section 3. We get the following result:

Theorem 2 *Let $\hat{a}$ be an estimate of the accuracies a using $\hat{M}$ computed from n samples, where all LFs are conditionally independent given Y. Assume that the signs of a are recoverable, that $|a_i| \geq a_{\min}$ is bounded away from 0, and that c is an absolute constant. Then, with high probability, we have*

$$\mathbb{E}[||\hat{a} - a||_2] \leq \frac{c}{a_{\min}^5}\sqrt{\frac{m}{n}}.$$

Let's interpret the behavior of the bound. First, the expression goes to 0 as we increase the number of samples n. Next, we are estimating m parameters (because there are m accuracies, and these are independent). Thus, if we wanted a fixed error, we need about m samples. Next, the closer $a_{\min}$ gets to 0, the more samples we need (but, this is a constant independent of m or n).

Of course, this was for the independent case. Here we were able to limit ourselves to working with triplets and thus only estimate $O(m)$ parameters. However, when we use the 'inverse' method, we need to estimate all the entries in Σ_O in order to compute Σ_O^{-1}, so that we need to estimate $O(m^2)$ parameters. To efficiently estimate such matrices, we use a powerful tool, the *effective rank*, which allows us to tighten our bounds on sample complexity even for full-rank matrices.

Effective rank The *effective rank* of the $m \times m$ matrix Σ, described by Vershynin in [7], is defined as

$$r_e(\Sigma) = \text{tr}(\Sigma)/\|\Sigma\|_2$$

where $\|\Sigma\|_2$ is the largest singular value of the matrix Σ. It is always true that $r_e(\Sigma) \leq \text{rank}(\Sigma) \leq m$.

Effective rank is a tool often applied to covariance matrix estimation and essentially measures the dimension of the subspace of the distribution of the random variables over which the covariance matrix is calculated. Therefore, a low effective rank shows that the data distribution is low dimensional.

The reason we're interested in the effective rank is that it yields tighter control over the error. Consider for example the error in estimating the covariance matrix for the inverse method. A beautiful result from Bunea [4] states that

$$\mathbb{E}[||\hat{\Sigma} - \Sigma||_2] \leq c\|\Sigma\|_2 \max\left\{\sqrt{\frac{r_e(\Sigma)\log mn}{n}}, \left(\frac{r_e(\Sigma)\log mn}{n}\right)\right\}.$$

Examining the right-hand side, if we replace $r_e(\Sigma)$ with worst-case, the full rank m, we would need $m\|\Sigma\|_2 \log m$ samples just to control the multiplicative error. However, the effective rank can be much smaller—even a constant, as we sketch out below.

Bounding the Effective Rank For the case described in section 3 of all weak sources being conditionally independent, we may have that the M matrix we estimate is full rank—but we shall see that its effective rank captures the notion that this is an easy case. Recall that M can be written as $M = I - \text{diag}(a^2) + aa^T$.

The trace of M is m since each diagonal element is 1. For $||M||$, the largest singular value of M, we can find a lower bound using Weyl's inequality. The eigenvalues of $I - \text{diag}(a^2)$ are $1 - a^2_{|min|} \geq \cdots \geq 1 - a^2_{|max|}$ where we order the accuracies in terms of absolute value, and the eigenvalues of aa^T are $||a||_2^2$ with multiplicity 1 and 0 with multiplicity 0. This is easy to see because we want $aa^Tv = \lambda v \Rightarrow a^Taa^Tv = \lambda a^Tv \Rightarrow ||a||_2^2 a^Tv = \lambda a^Tv$, so $\lambda = ||a||_2^2$. Furthermore, we can find $m-1$ v's that are orthogonal to a such that $a^Tv = 0$, in which case $\lambda = 0$. Weyl's inequality states that the largest eigenvalue of M will be greater than the smallest eigenvalue of $I - \text{diag}(a^2)$ plus the largest eigenvalue of aa^T, so

$$\sigma_{max}(M) \geq 1 - a^2_{|max|} + ||a||_2^2 = 1 + \sum_{i \neq |max|}^{m} a_i^2 \geq 1 + (m-1)a^2_{|min|} \geq ma^2_{|min|}.$$

Therefore, the effective rank of M is at most

$$r_e(M) = \frac{m}{\sigma_{max}(M)} \leq \frac{m}{1 - a^2_{|max|} + ||a||_2^2} \leq \frac{1}{a^2_{|min|}}.$$

Note how our effective rank is also intimately tied to $a_{\min}$: the closer to random the accuracy, the worse off we are.

References

[1] A. J. Ratner, B. Hancock, J. Dunnmon, F. Sala, S. Pandey, and C. Ré. Training complex models with multi-task weak supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Honolulu, Hawaii, 2019.

[2] P. Varma, F. Sala, A. He, A. Ratner, and C. Ré. Learning dependency structures for weak supervision models. In *ICML*, pp 6418-6427, 2019.

[3] P.-L. Loh and M. Wainwright. Structure estimation for discrete graphical models: Generalized covariance matrices and their inverses. *Annals of Statistics*, vol. 41, no. 6, 2013.

[4] F. Bunea and L. Xiao. On the sample covariance matrix estimator of reduced effective rank population matrices, with applications to fpca. *Bernoulli*, 21(5):1200-1230, 2015.

[5] D. R. Karger, S. Oh, and D. Shah. Iterative Learning for Reliable Crowdsourcing Systems. *Proc. Advances in Neural Information Processing Systems 24*, 2011.

[6] M. Wainwright and M. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1-305, 2008.

[7] R. Vershynin. Introduction to the Non-Asymptotic Analysis of Random Matrices. *Compressed Sensing*, pages 210268. Cambridge University Press, 2012.

[8] E. J. Candes, X. Li, Y. Ma, and J. Wright. Robust principal component analysis?. *Journal of the ACM*, vol. 58, no. 11, 2011.

[9] V. Chandrasekaran, S. Sanghavi, P. A. Parrilo, and A. S. Willsky. Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*, vol. 21, no. 2, 2011.

[10] S. Lauritzen. Graphical Models. Clarendon Press, 1996.

[11] F. Sala, P. Varma, S. Sagawa, J. Fries, D. Fu, S. Khattar, A. Ramamoorthy, K. Xiao, K. Fatahalian, J. Priest, and C. Ré. Multi-resolution weak supervision for sequential data. *Neural Information Processing Systems (NeurIPS)*, 2019.

A Connections to Crowdsourcing

The key idea behind crowdsourcing is easy to see by looking at a simple noise model. Say we have m workers that seek to estimate some value Y, and that each voter produces an estimate $Y + \varepsilon$, for $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. If each worker's noise is chosen i.i.d., then we can simply take the average of the workers' estimates, and a simple Chernoff bound shows that the probability that the error is greater than some fixed δ decreases exponentially fast in m.

The idea of averaging worker estimates has a discrete analog. For example, when each worker produces a vote in $\{-1, +1\}$, we can take the *majority vote*. These techniques work well when we are in the restrictive setting where each voter has independent and identical error probability. Unfortunately, for both crowdsourcing and weak supervision, this is unlikely to be true: workers and weak sources vary in their accuracy, and may be correlated with each other.

More modern approaches to crowdsourcing (e.g., Karger, Oh, and Shah in [5]) estimate the accuracies of the workers. One of the scenarios for their approach is the spammer/hammer model: a worker is either a hammer, and has probability 1 of predicting the correct value, or a spammer, who has probability 1/2, and thus has no useful information. For this model, the algorithm in [5] follows a message passing-style approach. Here, the graph being used is a bipartite graph with connections between (multiple) tasks and workers when a particular worker is allocated to a task. This yields opportunities to see distinct groups of agreements and disagreements. This approach, however, still requires independent noise for each worker.

Note that we can in fact model the scenario in [5] with our tools, in particular the ones from [1] and [**?**]. In particular, we can model the tasks in [5] with the task graph of [1]. Then, we can use the conditionally independent setting and run Algorithm 1. In addition, if we wish to avoid the matrix inversion step, we can apply the work in [**?**].

B Structure Learning

In the body of these notes, we assumed that we have access to the graph structure G. What if we do not have access to it? In general, this is known as the structure learning problem. In the typical setting, all of the variables are observed; the fact that we have a latent variable (the label) makes our case more challenging.

Below we briefly sketch out a surprising solution by applying a tool called robust PCA. Recall that PCA produces a low-rank decomposition. It can tolerate low-magnitude noise, but very large-magnitude noise is difficult. Robust PCA algorithms are designed to handle this specific case.

In robust PCA, we only get to observe a matrix $M \in \mathbb{R}^{m \times m}$ that is equal to the sum of a low-rank matrix and a sparse matrix,

$$M = L + S,$$

where $\text{rank}(L) = r$ and $|\text{supp}(S)| = k$. The goal is to decompose M into L and S. The we can keep L and get rid of the noise S.

Note that the decomposition $M = L + S$ is not identifiable without additional conditions. For example, if $M = e_i e_j^T$, M is itself both sparse and low-rank, and thus the pairs $(L, S) = (M, 0)$ and $(L, S) = (0, M)$ are equally valid solutions. Therefore, the fundamental question of robust PCA studies is to determine conditions under which the decomposition can be recovered. The two seminal works on robust PCA [8, 9] studied *transversality* conditions for identifiability.

Robust PCA for Structure Learning How do we connect this notion of robust PCA to our approach? Let's go back to (5), which says that

$$\Sigma_O^{-1} = K_O - zz^T.$$

Algorithm 2 Weak Supervision Structure Learning

```
Input: Estimate of the covariance matrix $\hat{\Sigma}_O$, parameters $\lambda_n, \gamma$, threshold $T$
Solve: $(\hat{S}, \hat{L}) = \text{argmin}_{(S,L)} - \log\det(S - L) + \text{tr}((S - L)\hat{\Sigma}_O) + \lambda_n(\gamma\|S\|_1 + \|L\|_*)$
  s.t. $S - L \succ 0, L \succeq 0$
$\hat{E} \leftarrow \{(i, j) : i < j, \hat{S}_{ij} > T\}$
Return: $\hat{G} = (V, \hat{E})$
```

Before, we knew the structure, which was reflected in the sparsity pattern of K_O. Now we do not know it—we just assume there is sparsity. We only get to observe Σ_O^{-1}. Note, however, that the problem matches the robust PCA setup:

- Like M, Σ_O^{-1} is the sum of:
  - a low-rank term $L = -zz^T$,
  - and a sparse term $S = K_O$.

Thus, if we run an algorithm for robust PCA, we will decompose the observed Σ_O^{-1} into zz^T and K_O. Note how the roles here are reversed: normally, in robust PCA, S is a noise term that we wish to remove, while in our case K_O contains the desired structure. See [2] for details.

Note that we can even learn the parameters by using zz^T via this approach—but we may choose not to. We can learn the structure with high probability with potentially a smaller number of samples, and then run the simpler Algorithm 1 on more or new samples. We may also run the structure learning algorithm on just a binary setting while leaving the full multiclass approach to Algorithm 1 once the structure is known.

Finally, we show a convex program (in fact, a semidefinite program) for performing the decomposition. Note how each of the terms in the objective function encourages the right behavior: the nuclear norm serves to encourage low-rank in L and the ℓ_1 norm encourages sparsity in S.