ACCELERATING MACHINE LEARNING WITH TRAINING DATA MANAGEMENT

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Alexander Jason Ratner

August 2019

Abstract

One of the biggest bottlenecks in developing machine learning applications today is the need for large hand-labeled training datasets. Even at the world's most sophisticated technology companies, and especially at other organizations across science, medicine, industry, and government, the time and monetary cost of labeling and managing large training datasets is often the blocking factor in using machine learning. In this thesis, we describe work on *training data management systems* that enable users to programmatically build and manage training datasets, rather than labeling and managing them by hand, and present algorithms and supporting theory for automatically modeling this noisier process of training set specification in order to improve the resulting training set quality. We then describe extensive empirical results and real-world deployments demonstrating that programmatically building, managing, and modeling training sets in this way can lead to radically faster, more flexible, and more accessible ways of developing machine learning applications.

We start by describing *data programming*, a paradigm for labeling training datasets programmatically rather than by hand, and Snorkel, an open source training data management system built around data programming that has been used by major technology companies, academic labs, and government agencies to build machine learning applications in days or weeks rather than months or years. In Snorkel, rather than hand-labeling training data, users write programmatic operators called *labeling functions*, which label data using various heuristic or weak supervision strategies such as pattern matching, distant supervision, and other models. These labeling functions can have noisy, conflicting, and correlated outputs, which Snorkel models and combines into clean training labels without requiring any ground truth using theoretically consistent modeling approaches we develop. We then report on extensive empirical validations, user studies, and real-world applications of Snorkel in industrial, scientific, medical, and other use cases ranging from knowledge base construction from text data to medical monitoring over image and video data.

Next, we will describe two other approaches for enabling users to programmatically build and manage training datasets, both currently integrated into the Snorkel open source framework: Snorkel MeTaL, an extension of data programming and Snorkel to the setting where users have multiple related classification tasks, in particular focusing on *multi-task learning*; and TANDA, a system for optimizing and managing strategies for *data augmentation*, a critical training dataset management technique wherein a labeled dataset is artificially expanded by transforming data points. Finally, we will conclude by outlining future research directions for further accelerating and democratizing machine learning workflows, such as higher-level programmatic interfaces and massively multi-task frameworks.

Acknowledgements

I owe my career to my advisor Christopher Ré, many times over. When I entered the PhD program at Stanford: I did not have a true academic or professional mentor; I had never had a driving purpose in my professional life, despite this being what I had always most wanted; and I had the lingering sense that I had never yet lived up to the full potential of who I could or should be. Chris changed all that. Whether it was more painful for him or for me is not worth debating, but I know that he did it with great expenditure of effort, time, and ultimately, care; and to this I am eternally grateful. In addition to all the many things he has taught me about being a scientist, researcher, person, leader, mentor, professional, and beyond, he also left me with one singular goal: to be even a fraction of the mentor he has been to me. If I can do that, I will have had a career to be truly proud of.

I am also incredibly grateful for my labmates and friends at Stanford, without whom I could never have gotten to the point of writing this thesis: Daniel Selsam, who first urged me to rotate with Chris, and Will Hamilton, who was there on the fourth floor to witness the consequences of this decision, were both better friends than I could ever have deserved and constant pillars of support throughout the program; Theodoris Rekatsinas and Stephen Bach, who were beyond giving in the time they spent mentoring, counseling, occasionally consoling, and working late nights with me, and had a greater impact on my PhD than I think they even suspect; Chris Aberger, Vincent Chen, Chris De Sa, Jared Dunnmon, Henry Ehrenberg, Jason Fries, Braden Hancock, Bryan He, Fred Sala, Jaeho Shin, Virginia Smith, Paroma Varma, Sen Wu, and the many others I had the extreme fortune to work closely with, and who made coming in to the office worth it each day even when exhaustion had tempered the academic motivation; and the many, many others in the lab, at Stanford, and beyond who I was lucky enough to interact and work with.

I am also especially grateful to my thesis committee and other unofficial advisors who provided me with incredibly giving support, honest feedback, and insightful advice: Gill Bejerano, John Duchi, Kayvon Fatahalian, Percy Liang, Kunle Olukotun, and Daniel Rubin. I am of course also incredibly grateful to the many users, contributors, and other collaborators of the Snorkel project, without whom I certainly could not have written this thesis.

Finally, and most importantly, I owe absolutely everything to my parents, and the love and support they have never stopped giving me my whole life; my brother; and the love of my life, my wife Julia. Words could never express my gratitude to my family.

Finally, I am grateful to the Stanford Morgridge Family SIGF and Bio-X SIGF fellowships for supporting my PhD, and to many sources for having supported the Snorkel project more broadly, including: DARPA under No. N66001-15-C-4043 (SIMPLEX), No. FA8750-17-2-0095 (D3M), No. FA8750-12-2-0335, and No. FA8750-13-2-0039; NSF IIS-1247701, NSF IIS-1353606, CCF-1111943, and CCF-1337375; DOE 108845; NIH U54EB020405; ONR under No. N000141210041 and No. N000141310129; the Moore Foundation, the Okawa Research Grant, American Family Insurance, Accenture, Toshiba, and members of the Stanford DAWN project: Google, Intel, Microsoft, Teradata, and VMware. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of DARPA, NSF, DOE, NIH, ONR, or the U.S. Government.

The results mentioned in this dissertation come from previously published work, including most centrally [Ratner et al., 2016, 2017c; Bach et al., 2017; Ratner et al., 2017b,a, 2019a, 2018, 2019b,c]. Some descriptions are directly from these papers. In particular,

- Chapter 3 presents content from [Ratner et al., 2016] and [Ratner et al., 2019b], and briefly, from [Bach et al., 2017; Varma et al., 2019, 2017];
- Chapter 4 presents content from [Ratner et al., 2017a] (and the extended award edition [Ratner et al., 2019a]), and also provides a brief overview of application results from [Ratner et al., 2017b; Birgmeier et al., 2017; Fries et al., 2017; Dunnmon et al., 2019; Bach et al., 2019; Kuleshov et al., 2019; Callahan et al., 2019; Wu et al., 2018; Bringer et al., 2019];
- Chapter 5 presents content from [Ratner et al., 2019b] and [Ratner et al., 2018];
- Chapter 6 presents content from [Ratner et al., 2017c];
- Chapter 7 includes content from [Ratner et al., 2019c].

These papers were joint efforts with different authors, including Stephen Bach, Christopher De Sa, Jared Dunnmon, Henry Ehrenberg, Jason Fries, Braden Hancock, Bryan He, Christopher Ré, Frederic Sala, Daniel Selsam, Paroma Varma, Sen Wu, and many others. The collection of research presented in this dissertation would not have been possible without the contributions of all these collaborators.

Software, Data, and Further Reading

- Code, tutorials, blog posts, and related publications about Data Programming and Snorkel (Chapters 3 and 4) can be found at `https://snorkel.org`
- Code and tutorials for Snorkel MeTaL (Chapter 5) can be found at `https://github.com/HazyResearch/metal`; however, note that codebase has since been deprecated, as the core functionalities of Snorkel MeTaL have moved into the Snorkel repo above as of Snorkel version 0.9.
- Code and tutorials for TANDA (Chapter 6) can be found at `https://github.com/HazyResearch/tanda-release`, and are also integrated into the Snorkel repo above as of Snorkel version 0.9.
- Further information and links relevant to this dissertation can be found at `https://ajratner.github.io`

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent advances in techniques and infrastructure have led to a flurry of excitement about the capabilities of machine learning (ML), leading some to even call it a new "Software 2.0" [Karpathy, 2017]. At the core of this excitement is a breed of new (mostly deep learning) models that learn their own features from data, leading to qualitative leaps in performance on traditionally challenging benchmark tasks, while obviating the need for what many machine learning developers and organizations previously spent years doing-engineering model features by hand. Together with massive technical and financial investment in open-source machine learning frameworks like TensorFlow [Abadi et al., 2016], PyTorch [Paszke, 2017], and others, and 'model zoos' like Onnx [Bai et al., 2019], state-of-the-art machine learning approaches have in many ways never been more accessible or efficient to apply.

The rise of modern representation learning methods, supported by robust and standardized frameworks, offers the potential for fundamentally simpler, more accessible, and more flexible ways of developing data-driven software. For example, even five years ago, a developer attempting to solve a complex task with machine learning–for example, an information extraction task like extracting facts about chemical-disease correlations from the scientific literature, or an image classification task like triaging chest radiographs–might have had to invest a PhD-length time period into developing problem-specific features (e.g. complex linguistic and biological prefix features for relation extraction, or Sobel operator and Fourier analysis-based features for radiograph images); models defined over those features; and algorithms for performing learning and inference. Today, to a first approximation, practitioners can often apply a state-of-the-art machine learning model to these or many other tasks in a few hours or less of writing Python code within a standardized, open source machine learning framework. This shift has had impact everywhere from small academic labs to the largest machine learning organizations in the world. For example, Google reportedly reduced one of its translation code bases from 500 thousand to approximately 500 lines of much more standardized, deployable, and easily maintainable code [Wu et al., 2016; Dean], and it has become commonplace for individuals and organizations alike to quickly spin up high-performing machine learning-based applications where years of effort might have once been required.

However, these increasingly accessible and powerful machine learning approached all rely on one key limiting reagent: large, hand-labeled *training datasets*. In supervised machine learning, which we will consider for the remainder of this dissertation, models are learned by fitting to training datasets consisting of data points labeled, generally by hand, according to the desired classification. While training data has always been both a bottleneck and an accelerant to machine learning, modern machine learning models generally achieve their impressive feat of automated representation learning by being massively overparameterized, often having hundred of millions of free parameters. This model complexity in turn means that they require massive training sets in order to reach peak performance, and the cost and difficulties of building these training sets are often the achilles heel of modern machine learning approaches.

Training datasets, especially ones requiring domain expertise and dealing with private data that cannot be shipped external to an organization, are often prohibitively expensive and slow to create; and, as real world data distributions and modeling objectives shift and evolve, labeled training sets are completely inflexible, and thus lead to the need for expensive and frequent re-labeling. In the Google translation example mentioned, the performance, deployment, and maintainability benefits of the new deep learning approach were predicated on 36 million hand-labeled examples per language pair. And the rosy picture painted of a practitioner applying modern state-of-the-art models to chemical-disease information extraction or radiological imaging tasks in hours is often bottlenecked on the prerequisite of a biologist or radiologist spending person months or years hand-labeling

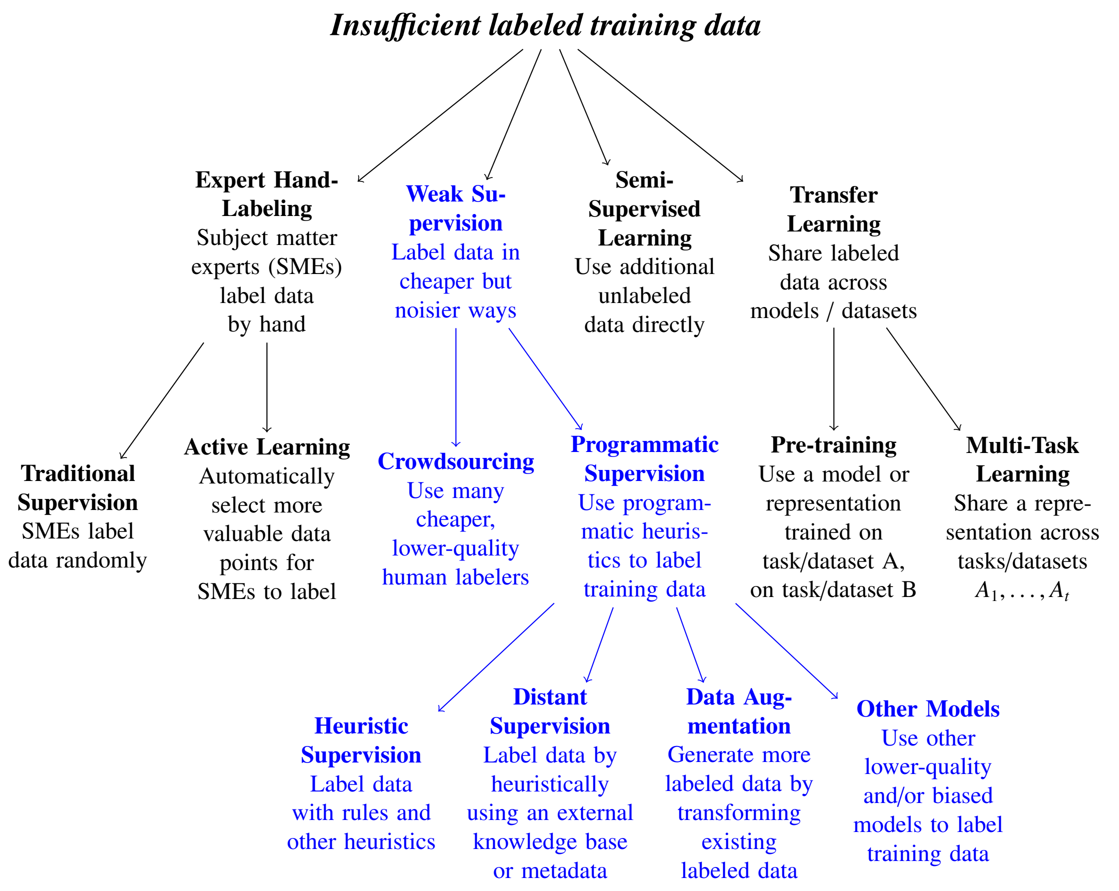

Insufficient labeled training data
Expert Hand-Labeling
Subject matter experts (SMEs) label data by hand
Weak Supervision
Label data in cheaper but noisier ways
Semi-Supervised Learning
Use additional unlabeled data directly
Transfer Learning
Share labeled data across models / datasets
Traditional Supervision
SMEs label data randomly
Active Learning
Automatically select more valuable data points for SMEs to label
Crowdsourcing
Use many cheaper, lower-quality human labelers
Programmatic Supervision
Use programmatic heuristics to label training data
Pre-training
Use a model or representation trained on task/dataset A, on task/dataset B
Multi-Task Learning
Share a representation across tasks/datasets $A_1, \ldots, A_t$
Heuristic Supervision
Label data with rules and other heuristics
Distant Supervision
Label data by heuristically using an external knowledge base or metadata
Data Augmentation
Generate more labeled data by transforming existing labeled data
Other Models
Use other lower-quality and/or biased models to label training data

Figure 1.1: A high level mapping of several classic and popular approaches for handling a lack of labeled training data. At the topmost level, we group the strategies into those that (i) involve having subject matter experts label individual data points, (ii) label data in weaker ways—the primary focus of this thesis, and highlighted in blue—(iii) utilize additional unlabeled data directly, and (iv) try to share labeled training data across tasks. These methods of course have many connections and commonalities not illustrated in this figure (for example, in Chapter 5 we describe work on weak supervision *for* multi-task learning models).

training data.

The dependence of modern machine learning approaches on large labeled training datasets has led to a resurgence of interest in various techniques, both classic and new, for dealing with a lack of labeled training data (Figure 1.1). These include *active learning*, where the goal is to solicit expert-annotated labels for specially-chosen data points, rather than over random samples, with the goal being to ultimately require fewer labeled data points [Settles, 2009]; *semi-supervised learning*, in which, in addition to a small labeled training set, various heuristics or regularizers are used over a larger unlabeled dataset [Chapelle et al., 2009]; and *transfer learning*, in which the high level goal is to share information across different models or datasets [Pan and Yang, 2010]; for more detail, see Section 2.2.

However, another approach, which we aim to support, formalize, and demonstrate the effectiveness of in this thesis, is *weak supervision*, where training data is labeled or otherwise generated in noisier, cheaper, often programmatic ways. While the aforementioned methods attempt to do more with fewer hand-annotated labels, weak supervision changes the type of input that users are asked to provide to supervise a machine learning model. Classic examples include *distant supervision* [Mintz et al., 2009; Craven et al., 1999], where an external knowledge base is heuristically used to label data; *crowdsourcing* approaches [Dawid and Skene, 1979; Karger et al., 2011; Dalvi et al., 2013] where crowd workers of unknown reliability are used to label data; and many others (see Section 3.5). Weak supervision–and in particular, the types of *programmatic supervision* that the systems developed in this thesis support–has the appeal of still providing a direct and pragmatic interface for users to supervise models, but in higher level ways that are far more scalable, efficient, interpretable, and adaptable. For this reason, weak supervision has had a resurgence of interest in the modern deep learning era.

The rise of weaker supervision can be seen as a fundamental shift in how practitioners principally interact with and *program* machine learning models: via the creation, engineering, and *management* of training data. Increasingly, this training data engineering is a central development activity which is done in higher-level, more programmatic ways, and can be seen as an entirely new way of programming the new ML stack. Emerging techniques, which this thesis advances and evaluates, include *labeling* data in higher-level,

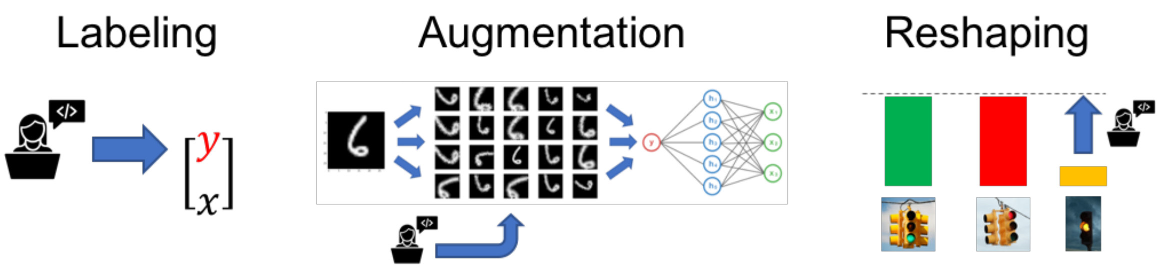


Figure 1.2: Machine learning developers increasingly interact with models not by traditional activities such as feature engineering or model architecture development, but rather through the creation, engineering, and more broadly, *management* of labeled training data through activities such as labeling, augmenting, and reshaping datasets.

programmatic, and/or noisier ways (often called weak supervision), such as using heuristics, patterns, existing datasets and models, or crowd labelers to label training data; *augmenting* datasets by creating transformed copies of labeled data points, thereby expressing data invariances (e.g. rotational or shift symmetries in images) in a simple, model-agnostic fashion; *reshaping* datasets, e.g. to emphasize performance critical subsets; and *combining* datasets, e.g. across related tasks. However, to date, these emerging approaches have generally been applied in heavily manual and ad hoc ways, relegated to the preprocessing and data loader scripts of machine learning pipelines where they are seen as 'tricks', rather than supported and formalized as key first-class operators of a new approach to machine learning.

Contributions and Outline In this thesis, we describe work on *training data management systems* that support the emergence of training data engineering as a first-class citizen of the machine learning workflow by enabling users to programmatically build and manage training datasets rather than label them by hand. We present algorithms and supporting theory for automatically modeling this noisier process of training set specification, which place these new techniques on more solid statistical and systems ground. Finally, we present empirical validations, user studies, and real-world deployments demonstrating that this new approach of programmatically building, managing, and modeling training datasets can lead to radically faster, more flexible, and more accessible ways of developing machine learning applications.

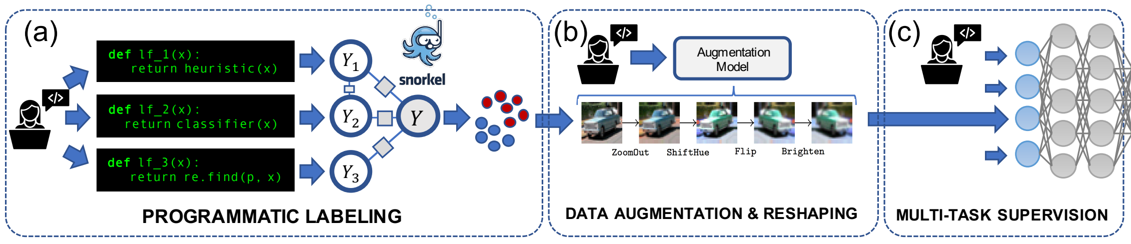

```
def lf_1(x):
    return heuristic(x)
def lf_2(x):
    return classifier(x)
def lf_3(x):
    return re.find(p, x)
```

Figure 1.3: *Creating and managing training data* has emerged as one of the key ways that developers can effectively *program* the modern machine learning stack. This dissertation covers work on systems and approaches aimed at formalizing, accelerating, and supporting techniques such as (a) programmatic data labeling (Chapters 3 and 4); (b) data augmentation and training set reshaping (Chapter 6) and (c) multi-task supervision (Chapter 5), which in practice form key parts of the emerging training data management pipeline.

In Chapter 2, we start by reviewing some preliminaries of this thesis. In Chapter 3, we then describe *data programming* [Ratner et al., 2016], an approach whereby practitioners, rather than hand-labeling training data, write *labeling functions* that heuristically label some subset of an unlabeled dataset. These labeling functions can express various heuristic or *weak supervision*strategies, such as distant supervision, crowdsourcing, pattern-based labeling, and arbitrary domain heuristics, and in general will have unknown accuracies and correlations, leading to overlaps and conflicts in their outputs. To handle this, we learn a generative *label model* to attempt to optimally re-weight and combine the noisy outputs of the labeling functions. The key challenge is learning this model in the absence of ground truth labels; this can be viewed as a novel type of *data cleaning* problem, or equivalently a latent variable model estimation problem for an extended class of labeling models. We describe two techniques for overcoming this technical challenge, and provide corresponding theoretical guarantees: in Section 3.2, a maximum marginal likelihood approach which we implement using stochastic gradient descent and Gibbs sampling [Ratner et al., 2016]; and in Section 3.3, a matrix-completion style approach [Ratner et al., 2019b]. In Section 3.4, we also describe several methods for estimating the structure of correlations between the labeling functions, which is essential to handling labeling functions that are correlated e.g. to shared data resource, code, or underlying heuristics [Bach et al., 2017; Varma et al., 2019, 2017].

Given the estimated label model, we then reweight and combine the outputs of the labeling functions to produce a clean, confidence-weighted set of training labels, which we can then use to train an arbitrary machine learning model, where the goal is for this end model to generalize beyond the labeling functions. We establish theoretical conditions under which, as the number of *unlabeled* data points the labeling functions are applied to increases, the generalization error of this end model converges at the same asymptotic rate as in traditional supervised methods- except in our case, with respect to unlabeled data.

In Chapter 4, we describe Snorkel [Ratner et al., 2017a], an end-to-end training data management system built around the data programming paradigm for rapidly and programmatically labeling training sets. In Snorkel, users start by writing labeling functions using various tools and interfaces, including common declarative weak supervision operators, that are applied over unlabeled data stored in a hierarchical data model. Snorkel then learns the structure and parameters of a generative label model over the matrix of labeling function outputs, and finally uses this to produce a set of probabilistic training labels which can be used to train any standard machine learning model e.g. in TensorFlow or PyTorch.

In Section 4.2, we discuss a new tradeoff space around when, and with what correlation structure density, to use the generative label model in the context of iterative user development where speed of iteration is given a significant premium. In Section 4.3, we experimentally validate Snorkel on six datasets, including two based on real-world collaborations—one around information extraction from electronic health records with the U.S. Department of Veterans Affairs and Stanford Hospital and Clinics, and one around information extraction from the scientific literature with the U.S. Food and Drug Administration—and two *cross-modal* settings where the labeling functions are applied to one feature set or modality (e.g. text) that is disjoint from the feature set or modality (e.g. images) that the end model is trained over and applied to, showing the flexibility of Snorkel to effectively transfer domain knowledge from one modality to another.

The broader goal of this thesis, and of the open source Snorkel project[1], is to demonstrate that enabling users to programmatically build, manage, and model training datasets can provide a new interface to machine learning that is both more accessible and more effective in real-world settings. In Section 4.4, we provide validation for the accessibility of these approaches by describing a Snorkel user study, conducted with fifteen researchers that

[1]`https://snorkel.org`

were invited to attend a two-day Snorkel workshop, having had no prior Snorkel (and minimal programming or machine learning) experience. We show that Snorkel indeed leads to more accessible machine learning, and better performance than spending a comparable amount of time simply hand-labeling training data. Finally, in Section 4.5, we provide validation for the real-world effectiveness of these approaches by giving a brief overview of several real-world deployments of Snorkel in industry, at Google [Bach et al., 2019], Intel [Bringer et al., 2019], and others; in medicine, in collaboration with Stanford Radiology and Neurology [Dunnmon et al., 2019], and others; in information extraction for genomics [Kuleshov et al., 2019]; and in other settings, where in many of these applications, Snorkel leads to building machine learning applications in days or weeks of development rather than months or years of hand-labeling.

In Chapter 5, we extend the data programming paradigm and Snorkel system to settings where users have more than one, possibly related, modeling tasks that they need to build and manage training datasets for, and investigate whether modeling this *multi-task supervision* jointly can improve performance. We extend the matrix completion-style data programming approach in Section 3.3 to the multi-task setting, and validate it on several fine grain entity and relation extraction tasks. We also briefly describe a new open source system for multi-task supervision and multi-task learning, Snorkel MeTaL[2], which has since been merged into the main Snorkel framework[3].

In Chapter 6, we consider a second distinct but complementary way of programmatically building and managing training datasets–*data augmentation*, in which transformed copies of labeled training data points are used to expand the size of a training set–and develop a paradigm and approach for modeling, tuning, and managing this form of programmatic weak supervision input [Ratner et al., 2017c]. The canonical example of data augmentation in practice is randomly rotating, stretching, and blurring labeling images before training a computer vision model; more broadly, data augmentation is applied in many settings and data modalities where there are transformation operations that with reasonable

[2]`https://github.com/HazyResearch/metal`

[3]As of version 0.9.

likelihood will preserve the class label of transformed data points. In many machine learning applications—for example, nearly all state-of-the-art models in image classification—data augmentation is an absolutely critical technique for enhancing performance, and can be viewed as a way of imposing knowledge of invariances in a model-agnostic way, by expressing it via the training data. However, data augmentation strategies can be hard to tune and compose for new datasets (e.g., 'how much to rotate? How much to blur?'), and in practice are mostly applied in ad hoc and manually-tuned ways without any formal support or optimization. We describe an approach wherein users provide incremental transformations as programmatic operators called *transformation functions*, and we then automatically learn to tune and compose them using a generative adversarial approach. We describe a system for data augmentation built around this approach, TANDA[4], and describe empirical validation on a range of text and image datasets.

Finally, in Chapter 7, we review some concluding thoughts, and outline some future directions for work on programmatically building, managing, and modeling training data, and beyond.

[4]urlhttps://github.com/HazyResearch/tanda

Chapter 2

Preliminaries

In this section, we provide additional background both to situate this work and contextualize some key technical pieces. In Section 2.1, we provide a brief additional perspective on the shift to training data as a focal point for machine learning development. In Section 2.2 we then provide a high level overview of traditional approaches for dealing with the bottleneck of labeled training data, which naturally come to the forefront as training data becomes increasingly critical. Next, in Section 2.3, we briefly review the basics of prior approaches to learning the parameters of latent variable models, e.g. for learning the accuracies of weak supervision sources, a key technical building block of the algorithms and results in Sections 3, 4, and 5. Finally, in Section 2.4, we briefly review existing work related to data management for training data.

2.1 The Transition to Training Data as the Bottleneck

Supervised Learning In this thesis, we focus on *supervised learning*, in which the goal is to use a *training dataset* of labeled data points to select a model that correctly labels new data points. That is, we have data points $x \in \mathcal{X}$ (e.g. a document or an image) and labels $y \in \mathcal{Y}$ (e.g. binary or categorical labels), and a dataset of labeled examples, $T = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$, which we refer to as a *labeled training set*. Our goal is then to select a function $h : \mathcal{X} \mapsto \mathcal{Y}$ that, when given a new unseen *test* data point, $x \in \mathcal{X}$, accurately predicts the corresponding label y. In general, we assume that all data points,

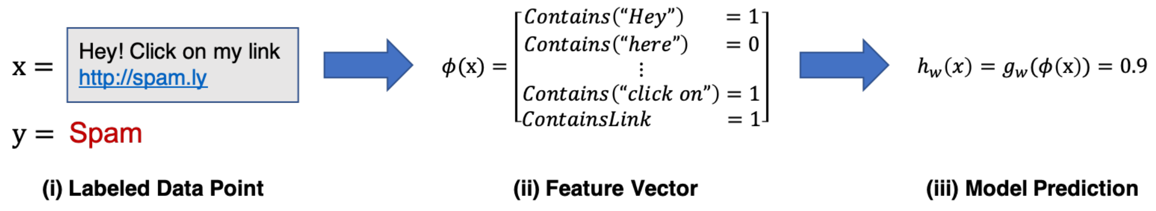

x =
Hey! Click on my link http://spam.ly
y = Spam
$\phi(x) = \begin{bmatrix} Contains(\text{"Hey"}) = 1 \\ Contains(\text{"here"}) = 0 \\ \vdots \\ Contains(\text{"click on"}) = 1 \\ ContainsLink = 1 \end{bmatrix}$
$h_w(x) = g_w(\phi(x)) = 0.9$
(i) Labeled Data Point
(ii) Feature Vector
(iii) Model Prediction

Figure 2.1: A simple example of a supervised learning task where the goal is to classify text comments as spam or not. In a traditional supervised learning approach, our training set would consist of labeled examples (i); we would then extract features, for example indicating whether certain words, word sequences, or other objects were present in in the text comment (ii); and finally, we would train a model defined over those features–that is, select a set of parameters w–such that the model predictions matched the training labels as much as possible (iii).

both training and test, are i.i.d. sampled from some underlying distribution, $(x, y) \sim \mathcal{D}$.

As a concrete example, we consider a canonical supervised learning problem: classifying whether or not a text comment represents spam (i.e. irrelevant or malicious content) or not (Figure 2.1). Here, our data point x is a string of characters, and $y \in \{0, 1\}$ where 1 denotes spam, and 0 denotes not spam. Our goal is to train a machine learning model that can accurately classify new, unlabeled data points.

The standard procedure in supervised learning is to first select a model or *hypothesis class*, $\mathcal{H}$, such that $h \in \mathcal{H}$. In general, we consider hypothesis classes that are parameterized, meaning that a vector w specifies a hypothesis function $h_w \in \mathcal{H}$, and use this notation from here on out.

Let $l : \mathcal{Y} \times \mathcal{Y} \mapsto [0, 1]$ be a *loss function*, for example the zero-one loss $l(\hat{y}, y) = \mathbb{1}\{\hat{y} \neq y\}$. Then, given a fixed hypothesis class, our goal in supervised learning can be succinctly described as finding parameters w^* which minimize the expected loss or *risk* R:

$$w^* = \text{argmin}_w\, R(w) = \text{argmin}_w\, \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[l(h_w(x), y)\right]$$

The standard way we approach this is to select the parameters that minimize the loss over the training set, or the *empirical risk* $\hat{R}$, termed the *empirical risk minimizer*

$$\hat{w} = \text{argmin}_w \hat{R}(w) = \text{argmin}_w \frac{1}{n} \sum_{(x,y)\in T} l(h_w(x), y)$$

Feature Engineering In many real-world settings of interest, x is a very complex and high-dimensional object, making the selection of a model challenging. For example, x might be an *unstructured* data point like a document or an image that we wish to classify. Even in our running spam example, where x is a short text comment, the space of possible configurations is combinatorially large. A traditional approach to handling this type of data is to introduce an intermediate step of extracting a subset of *features* of x, $\phi(x) \in \mathbb{R}^d$, that the model takes as input. For simplicity, we can represent this *feature extraction* or engineering step as part of the hypothesis class definition, where now

$$h_w = g_w \circ \phi$$

that is, we view feature engineering as a form of model engineering. For example, in our running example, we might think that the absence or presence of certain words, word sequences, other objects like links, and potentially other complex patterns (e.g. having to do with grammar, tone, format, or structure) might all be relevant features for making the spam-versus-not-spam classification (Figure 2.1 (ii)). Given that we are using a machine learning approach, we do not have to set the weights of how important these features are–our model will learn this from data–however coming up with the relevant feature set is still a tricky and onerous task.

For many years in practice, this task of *feature engineering*—i.e. the process of designing this function ϕ—was arguably *the* dominant activity that a machine learning engineer engaged in, as well as a significant focal point of research. For example, in natural language processing, a wealth of work has been performed exploring the optimal set of linguistic and semantic features (e.g. words, word sequences or "n-grams", grammatical dependency substructures, etc.) to extract for various modeling tasks. Similarly, in image classification or computer vision, various sophisticated approaches to feature extraction have been the focus of heavy research. A review of the feature engineering literature is outside the scope of this thesis, however good overviews can be found in [Guyon and Elisseeff, 2003; Guyon et al., 2006] as well as in a wide range of machine learning textbooks, blog posts, and tutorials.

Modern Representation Learning Methods One of the most impactful and widely-observed trends in machine learning over the last several years has been the rise of *deep learning* model architectures, which learn their own feature extractors, or *representations* of the raw input data, and have largely obviated the traditional practice of feature engineering in a range of traditionally challenging domains such as computer vision, natural language processing, and beyond. While these new deep learning approaches rest on decades of research and a recent surge of academic and industrial innovation, their success and popularity can also be largely attributed a heavily-funded ecosystem of open source platforms (e.g. TensorFlow [Abadi et al., 2016], PyTorch [Paszke, 2017], and others), model zoos (e.g. Onnx [Bai et al., 2019]), a wide range of associated tools, and increasingly declarative interfaces (e.g. Keras [Chollet et al., 2015], Ludwig [Molino, 2019], and many others). The net effect is that where a decade ago building a machine learning pipeline might have required thousands of lines of feature engineering and learning algorithm code, sophisticated models can now be defined in dozens of lines of code or less.[1] For instance, in our running spam example, rather than having to do any feature engineering, we might simply feed the raw text data, x, into a Long Short-Term Memory (LSTM) network or other recurrent neural network architecture, which due to modern machine learning frameworks, would require only several lines of code.

While this new deep learning tool chain has raised a range of challenging research questions and practical issues—e.g. around interpretability, scalability, robustness, and the like—we focus here on the practical impact: it has made building machine learning models easier than ever before where large labeled training sets are available. An increasingly large number of applications in traditionally challenging domains like computer vision and natural language processing now get state-of-the-art scores using standard, effectively commoditized model architectures like LSTMs and Convolutional Neural Networks (CNNs). One application which exemplifies this is Google's machine translation system: in 2016,

[1]E.g. see Keras's intro tutorial, "30 seconds to Keras", `https://keras.io/#getting-started-30-seconds-to-keras`

a Google team reported on a new deep learning-based machine translation model that reduced errors by an average of 60% compared to a previous production model [Wu et al., 2016]. However, arguably the biggest impact was that this model enabled the team to throw out approximately five hundred thousand lines of feature engineering code and replace it with approximately five hundred lines of generic and portable TensorFlow code[2].

The catch in general is that these new deep learning models are highly complex, often with hundreds of millions of parameters, and require massive labeled training datasets to reach peak performance [Sun et al., 2017]. For example, the aforementioned Google Translate model relied on a manually labeled training dataset of 36 million examples, for one language pair; and other state-of-the-art deep learning results have often relied on similarly large labeled training datasets. A survey of deep learning methods and systems, and their dependence on large volumes of labeled training data, is once again outside the scope of this thesis. Instead, we anchor on the broad idea that in a wide range of settings, machine learning has become vastly easier to use without nearly any feature engineering- *if* a large enough labeled training dataset is available.

The Transition from Feature to Training Data Engineering This thesis is motivated by the observation that many practitioners have begun to use deep learning models that require very little feature engineering, but that in turn require large volumes of labeled training data. As a result, these practitioners have shifted from largely focusing on traditional data and feature engineering activities, to increasingly spending their time on *training data engineering:* labeling, building, and managing training datasets.

One perspective is that this shift in machine learning development effort can be seen as flipping the old intuition of how to build and improve machine learning models on its head. More concretely, we can view this through the lens of traditional machine learning theory. While this theoretical perspective is not meant to be taken as a literal guide—and potentially lacks explanatory power in the context of today's deep, over-parameterized model classes in a more serious way [Zhang et al., 2016a]—it has served as the root of machine learning developers' practical rules of thumb for many years, and so provides some relevant intuition in this context.

[2]https://twitter.com/DynamicWebPaige/status/915326707107844097

We define $VC(\mathcal{H})$ as the *VC dimension* of our model or hypothesis class, a classic measure of the complexity of $\mathcal{H}$, and recall that we parameterize $\mathcal{H}$ by w. Let n once again be the number of training data points—which we will assume are sampled i.i.d. from some distribution $\mathcal{D}$ and are labeled—and recall our definition of the risk, $R(w)$ and empirical risk $\hat{R}(w)$, and let w^* and $\hat{w}$ be the risk and empirical risk minimizers respectively. Then, a classic result bounds the *generalization error*—i.e. the difference in risk between the optimal model w^* and empirically estimated model $\hat{w}$—by [Liang, 2019]:

$$R(\hat{w}) - R(w^*) \leq O\left(\sqrt{\frac{VC(\mathcal{H})}{n/\log(n)}}\right) \tag{2.1}$$

At a high level, we can view the feature engineering-approach of machine learning development as an attempt to improve model performance by reducing the complexity of the model (the numerator of the right-hand side of (2.1)) given a relatively small, fixed training set size n. That is, features are easy to think of (e.g. for images: any indicator for a specific combination of pixels could be a feature), and so feature engineering primarily consists of attempting to select a good *subset* of the possible features, i.e. attempting to reduce the model complexity. On the other hand, we can view the approach taken with modern representation learning models as motivated by the exact opposite strategy: developers use massively complex model classes, and instead focus on building large enough training sets (the denominator of the right-hand side of (2.1)).

Regardless of the above intuition, a large part of the machine learning ecosystem today exists in a state where practitioners are most often bottlenecked on needing *more training data*, and thus increasingly turn to a range of techniques, both classic and new, for dealing with this, which we now provide high level background on.

2.2 Dealing with Limited Labeled Data

For reasons outlined above, having to do with the performance and accessibility advantages of modern machine learning models that are complex and data-hungry, getting large enough training datasets has emerged as one of the most prominent bottlenecks in machine learning application development. However, a diverse range of classical techniques can be viewed as ways of addressing this issue of limited labeled training data, which we briefly review in this subsection. To additionally ground and motivate this section—and to a large degree, this thesis overall—we refer to the human labelers that can label training or test data with high enough accuracy as the *subject matter experts (SMEs)*, and are especially motivated by settings where the SMEs must have some non-trivial domain knowledge, e.g. doctors, analysts, etc. For example, in our simple running example of comment spam classification, a SME might be someone who is well versed in the nuances of the spam policy of the forum hosting the comments.

At a high level, we consider four classic strategies for dealing with the problem of insufficient labeled training data (Figure 1.1):

1. **Expert Hand-Labeling:** The standard approach in supervised learning of having SMEs label individual training data points by hand (often still with several labelers per data point in difficult or critical settings). Here, concretely, the input to our overall learning procedure is a labeled training set $T = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$.

2. **Weak Supervision:** The broad class of approaches where training data is labeled in heuristic, often programmatic, and/or noisier ways that are cheaper and more efficient than expert hand-labeling. Weak supervision is the focus of the methods and systems described in this thesis. Concretely, here the input to our learning procedure is a weakly supervised labeled training set, $\tilde{T} = \{(x^{(1)}, \tilde{y}^{(1)}), \dots, (x^{(n)}, \tilde{y}^{(n)})\}$, where $\tilde{y}^{(i)}$ represents our weak labels and could be a vector of potentially conflicting labels, and moreover might be generated programmatically, as in the approaches described in this thesis.

3. **Semi-Supervised Learning:** The approach of using an unlabeled dataset as a complement to a smaller, expert-labeled training set. While there are many similarities to weak supervision, semi-supervised approaches generally use domain agnostic constraints or assumptions over the unlabeled data, as opposed to input from SMEs, and also require some set of expert-labeled training data. Here our input is a small labeled training set as above, $T = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$, and a larger unlabeled training set $T_U = \{x^{(n+1)}, \dots, x^{(n+n_U)}\}$.

4. **Transfer Learning:** The high-level strategy of transferring models or learned representations from one modeling task and/or dataset or another, in order to get more out of fewer labeled training datasets. Here we might have multiple training sets, T_t, and the goal is to somehow transfer information between them in ways that increase practical efficiencies.

We now briefly review each of these categories of approaches; further detail is given in the cited survey references and, where relevant, in the related work sections of subsequent chapters.

Expert Hand-Labeling: The standard approach in supervised learning is to have subject matter experts (SMEs) label individual training data instances by hand, often with multiple labelers per data point. Especially for settings where domain expertise is required (e.g. requiring doctors of a certain specialty for a medical triaging problem), data privacy is a concern (where labelers therefore must have proper clearance), and where problem input data or output schema are often changing (thus necessitating re-labeling), this standard approach of hand-labeling training data can be prohibitively expensive, slow, and static.

Active learning is one classic means of addressing this, wherein the goal is to make use of subject matter experts more efficiently by having them label data points which are estimated to be most valuable to the model; for a good survey, see [Settles, 2009]. Traditionally, applied to the standard supervised learning setting, this means selecting new data points to be labeled–for example, in our running spam example, we might hope to iteratively select comments that are very unique and/or close to the current decision boundary for SME labeling. However, we could also just ask for weaker supervision pertinent to these data points, in which case active learning is perfectly complementary with weak supervision; as one example of this, see [Druck et al., 2009].

Weak Supervision: *Weak supervision* is the approach of labeling training data in cheaper and/or higher-level, often programmatic ways. In general, weak supervision centrally involves human input, but either from lower-quality sources—e.g. non-expert crowd workers—or provided in higher-level, more efficient ways than labeling data points individually.

Weak supervision is the major focus of this thesis; we provide a brief overview of approaches here, and later in Sections 3.5 and 4.6.

A classic weak supervision strategy is *crowdsourcing* [Krishna et al., 2016; Gao et al., 2011], where a larger pool of non-expert and potentially unreliable 'crowd workers' are used in lieu of carefully-vetted subject matter experts. A wide range of classic work has treated both practical systems, algorithmic, and theoretical aspects of this strategy [Dawid and Skene, 1979; Karger et al., 2011; Parisi et al., 2014; Berend and Kontorovich, 2014; Zhang et al., 2016b; Dalvi et al., 2013; Joglekar et al., 2015], which we build on in Chapter 3.

Distant supervision is another classic approach wherein existing knowledge sources or metadata is used to heuristically label training sets. The canonical example is relation extraction from text, wherein a knowledge base of known relations is heuristically mapped to label a set of mentions in an input corpus as ground truth examples [Craven et al., 1999; Mintz et al., 2009; Zhang et al., 2017a]. Other extensions take steps towards modeling the quality of the distant supervision and other variations [Riedel et al., 2010; Hoffmann et al., 2011; Roth and Klakow, 2013a; Alfonseca et al., 2012; Roth and Klakow, 2013b; Takamatsu et al., 2012].

Another broader type of weak supervision is to use *rules, patterns, or other heuristics* to label training data [Bunescu and Mooney, 2007; Shin et al., 2015; Mallory et al., 2015; Gupta and Manning, 2014; Zhang et al., 2017a]. Weak supervision approaches also include non-traditional types of supervision, such as having SMEs *label features directly* [Zaidan and Eisner, 2008], directly specify *expected label or feature distributions* [Mann and McCallum, 2010; Liang et al., 2009], or specify *constraints* (which can also be viewed as specifying a label distribution) [Stewart and Ermon, 2017; Clarke et al., 2010; Guu et al., 2017].

Finally another popular and empirically critical technique that can be viewed as a form of weak supervision is the practice of *data augmentation*, in which labeled training data points are transformed in order to programmatically expand, or augment, the training dataset; the canonical example is randomly rotating images before training a computer vision model, but many more advanced techniques for performing and/or automating data augmentation have been proposed [Graham, 2014; Dosovitskiy et al., 2015; Uhlich et al., 2017; Lu et al., 2006; Ciresan et al.; Dosovitskiy et al., 2015; Chawla et al., 2002; DeVries and Taylor, 2017; Hauberg et al., 2016; Teo et al., 2008; Fawzi et al., 2016; Sixt et al., 2016]. We can view this as a way of programmatically generating training data points by using domain knowledge of invariances. We discuss data augmentation further in Chapter 6.

Overall, this thesis focuses on new data management systems, algorithmic approaches, and theoretical grounding for weak supervision, and aims to build on, support, and subsume many of the weak supervision approaches used in practice.

Semi-Supervised Learning: *Semi-supervised learning* considers the setting of a small expert-labeled training set and a much larger unlabeled data set. At a high level, the approach is to then use some type of domain-agnostic assumption—e.g. about smoothness, low dimensional structure, or distance metrics—to leverage the unlabeled data (either as part of a generative model, as a regularizer for a discriminative model, or to learn a compact data representation). For instance, in our running spam example, we might have access to a very large unlabeled corpus of text comments, and choose to regularize our model such that it tends to make a strong decision on each of these, or to select features such that the unlabeled examples fall into discrete clusters (as two examples of classic semi-supervised techniques). For a good survey of classic techniques see [Chapelle et al., 2009]. More recent methods use adversarial generative [Salimans et al., 2016], heuristic transformation models [Laine and Aila, 2016], and other generative approaches to effectively help regularize decision boundaries. Broadly, rather than soliciting more input from subject matter experts, the idea in semi-supervised learning is to leverage domain- and task-agnostic assumptions to exploit the unlabeled data that is often cheaply available in large quantities.

Transfer Learning: In the standard *transfer learning* setting, the goal is to take one or more models already trained on a different dataset or modeling task and apply them to a dataset or task of interest; for a good overview see [Pan and Yang, 2010]. For example, we might have a large training set for a text email classification problem, with classifiers already trained on this set, and wish to apply these somehow to our own text comments problem. A common and popular transfer learning approach today is to *pre-train* a model on one large dataset, and then "fine-tune" it on the task of interest. Another related and diverse line of work is *multi-task learning*, where several tasks are learned jointly [Caruana, 1997]; for a good survey of modern multi-task learning approaches, see [Ruder, 2017].

2.3 Modeling Weak Supervision Sources

Many of the core algorithmic and theoretical challenges that this thesis examines focus on the challenges of dealing with multiple *weak supervision* sources—i.e. sources of training labels or other signals—that may have diverse and unknown accuracies, correlations, and expertise areas. The key technical challenge in our setting is estimating and accounting for these attributes *in the absence of ground truth labels*. To provide background for the approaches we take to address this challenge, we start by reviewing a simple, classically-considered setting that we extend and build on in this thesis: estimating the accuracies of different conditionally-independent labelers in the absence of ground truth labels. We start by setting up this basic problem and model in more detail. We then review two classic approaches for solving it, in this simple conditionally-independent setting.

Problem Setup We consider a simple setting where, as in Section 2.1, labeled data points are sampled i.i.d. from an underlying distribution, $(x, y) \sim \mathcal{D}$. However, in the weak supervision setting, we do not observe the labels y. Instead, for each data point x, we observe *weak* labels from m sources, $\lambda_1, \ldots, \lambda_m$, $\lambda_j \in \mathcal{Y}$, which we write in vector form as λ.

In this preliminary example, we assume a simple model where these label sources are *conditionally independent* given the unobserved true label y, i.e. $\lambda_j \perp \lambda_{k \neq j} \mid y$. In the crowdsourcing setting, this is a classic setting—often referred to as the *Dawid-Skene model* [Dawid and Skene, 1979]—which represents an assumption that the weak sources make uncorrelated errors. To simplify further, we assume that we are in a binary setting with balanced classes, i.e. $\mathcal{Y} = \{-1, 1\}$, $P(y = 1) = P(y = -1) = 1/2$. Furthermore, we assume a model where the label sources have class-symmetric conditional probabilities that are independent of the data point being labeled. That is, letting θ be the parameters of our weak supervision model, we assume:

$$p_\theta(\lambda_j = 1|y = 1) = p_\theta(\lambda_j = -1|y = -1)$$

where p_θ represents the probability under the model parameterized by θ, which we shortly define more explicitly.

We can then represent the joint distribution of our model of the weak supervision λ and unobserved, or latent, true label y—which we refer to as the *label model*—by the following distribution which factorizes:

$$p_\theta(\lambda, y) = \prod_{j=1}^{m} p_\theta(\lambda_j|y)p_\theta(y) = \frac{1}{2}\prod_{j=1}^{m} p_\theta(\lambda_j|y)$$

where recall that this is the joint distribution for a single data point x.

Note that since we assume the data points are i.i.d. sampled, we can simply take the product of the above distribution over all data points in e.g. our training set. Let $\Lambda \in \mathbb{R}^{n\times m}$ be the *label matrix* of all weak supervision labels for all n data points in a given dataset, and let $\vec{y} \in \mathcal{Y}^n$ be the corresponding vector of true labels; then we write:

$$p_\theta(\Lambda, \vec{y}) = \prod_{i=1}^{n} p_\theta(\Lambda_i, \vec{y}_i) \equiv \prod_{i=1}^{n} p_\theta(\lambda^{(i)}, y^{(i)})$$

For simplicity of notation however, whenever possible we avoid writing this out explicitly, and consider a single i.i.d. data point.

We can now introduce a convenient representation of our parameterization; let:

$$p_\theta(\lambda_j = 1|y = 1) = \frac{\exp(\theta_j)}{\exp(\theta_j) + \exp(-\theta_j)}$$

where θ is in this context a vector $\theta = [\theta_1, \ldots, \theta_m]$.

Then we can represent our label model distribution in a simple exponential family form:

$$p_\theta(\lambda, y) = \frac{1}{2}\prod_{j=1}^{m} \frac{\exp(\theta_j\lambda_j y)}{\exp(\theta_j) + \exp(-\theta_j)} = \frac{\exp(\theta^T\lambda y)}{\sum_{\lambda'\in\{-1,1\}^m, y\in\{-1,1\}} \exp(\theta^T\lambda')} = Z^{-1}(\theta)\exp(\theta^T\lambda y)$$

where in our simple setting, λy is the *sufficient statistic* vector—or alternatively, in the factor graph view we will use in certain chapters, the vector of *factor functions*—and $Z(\theta) = \sum_{\lambda' \in \{-1,1\}^m, y \in \{-1,1\}} \exp(\theta^T \lambda')$ is the *partition function*. In the more complex versions of the label model we consider in this thesis, we will retain the same exponential family form, but with more complex sufficient statistic / factor functions and corresponding parameters.

Our objective is now to learn the parameters θ of this label model—corresponding in this model to the *accuracies* of the weak supervision sources. If we can recover θ, then we can estimate the true label, $p_\theta(y|\lambda)$. **The key technical challenge—which we will review briefly here, and then more extensively tackle in this thesis—is learning these parameters θ given that y is unobserved, or latent.**

2.3.1 Classic Approaches

Expectation Maximization (EM) Algorithm The classic approach to solving this latent variable model estimation problem is to use the *expectation maximization (EM)* algorithm, and many prior works in the area of weak supervision modeling (most commonly, crowd-sourcing) have relied on this algorithm. In the EM algorithm, we iteratively alternate between two steps in order to attempt to maximize the marginal likelihood of the observed variables; in our setting, this is:

$$L(\theta; \lambda) = p_\theta(\Lambda) = \sum_{\vec{y} \in \mathcal{Y}^n} p_\theta(\Lambda, \vec{y})$$

To do this, we start with the *expectation* step, where we compute the expected value of the log likelihood function with respect to the current estimate of the latent variables, i.e. using the current parameters $\theta^{(t)}$ at time step t. In our setting this quantity is:

$$\begin{aligned}
\frac{1}{n} Q(\theta|\theta^{(t)}) &= \frac{1}{n} \mathbb{E}_{\vec{y} \sim p_{\theta^{(t)}}(\vec{y}|\Lambda)} \left[\log p_\theta(\Lambda, \vec{y})\right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{y \sim p_{\theta^{(t)}}(y|\lambda^{(i)})} \left[\log p_\theta(\lambda^{(i)}, y)\right] \\
&= -\log Z(\theta) + \frac{1}{n} \mathbb{E}_{\vec{y} \sim p_{\theta^{(t)}}(\vec{y}|\Lambda)} \left[\vec{y}^T \Lambda \theta\right]
\end{aligned}$$

Next, in the *maximization* step, we update our parameter estimates to maximize $Q(\theta|\theta^{(t)})$, in other words $\theta^{(t+1)} = \operatorname{argmax}_\theta Q(\theta|\theta^{(t)})$. To do this, we can compute the gradient:

$$0 = \nabla_\theta \left(\frac{1}{n} Q(\theta|\theta^{(t)}) \right) = -\mathbb{E}_{(\lambda,y)\sim p_\theta} [\lambda y] + \frac{1}{n} \mathbb{E}_{\vec{y}\sim p_{\theta^{(t)}}(\vec{y}|\Lambda)} \left[\Lambda^T \vec{y} \right]$$

where we use a useful lemma about the gradient of log partition function, that $\nabla_\theta \log(Z(\theta)) = \mathbb{E}_{(\lambda,y)\sim p_\theta} [\lambda y]$ (see Appendix 1).

This results in an intuitive update to our parameters. We see, letting α_j be the accuracy of the jth weak supervision source, that:

$$\left(\mathbb{E}_{(\lambda,y)\sim p_\theta} [\lambda y] \right)_j = p_\theta(\lambda_j = y) - p_\theta(\lambda_j \neq y) = 2\alpha_j - 1$$

where note that in our parameterization, $\alpha_j = \exp(\theta_j)/(\exp(\theta_j) + \exp(-\theta_j))$. That is, in the setting we consider here, the EM algorithm procedure is to iteratively match our current estimate of the label source accuracies to their empirical accuracies according to the conditional estimate of the latent variable y from the previous iteration:

$$\alpha_j^{(t+1)} = \frac{1}{2} \left(1 + \frac{1}{n} \mathbb{E}_{\vec{y}\sim p_{\theta^{(t)}}(\vec{y}|\Lambda)} \left[\Lambda^T \vec{y} \right] \right)$$

The EM algorithm is a commonly applied approach, however has at least two drawbacks. First, the EM algorithm is only guaranteed to find a local minimum, i.e. to iteratively increase the marginal likelihood; if we want to make any sort of convergence guarantees, we need to move beyond this. Second, performing the update step as above—while appealingly simple in this setting—required deriving a closed-form expression for $\mathbb{E}_{(\lambda,y)\sim p_\theta} [\lambda y]$, which can be difficult in more complex weak supervision models that we will consider.

Spectral & Tensor Decomposition Approaches Another set of approaches have used *spectral* techniques: broadly, linear algebraic approaches involving computing the eigenvectors of some function of the moments of the observed label matrix Λ. We briefly review one of these approaches, due to Gosh et. al. [Ghosh et al., 2011]. Here, we consider the matrix $\Lambda\Lambda^T$, and consider the expected value of one entry conditioned on the unobserved $\vec{y}$—in other words, the second moment of Λ with respect to the distribution conditioned on

$\vec{y}$:

$$\mathbb{E}_{\Lambda\sim p_\theta(\cdot|\vec{y})}\left[\left(\Lambda\Lambda^T\right)_{i,j}\right] = \sum_{k=1}^{m}\mathbb{E}_{\Lambda\sim p_\theta(\cdot|\vec{y})}\left[\lambda_k^{(i)}\lambda_k^{(j)}\right]$$

Under our simple model, we have, for $i \neq j$:

$$\mathbb{E}_{\Lambda\sim p_\theta(\cdot|\vec{y})}\left[\lambda_k^{(i)}\lambda_k^{(j)}\right] = y^{(i)}y^{(j)}\left(\alpha_k^2 + (1-\alpha_k)^2 - 2\alpha_k(1-\alpha_k)\right) = y^{(i)}y^{(j)}\left(2\alpha_k - 1\right)^2$$

where again we write $\alpha_j = p_\theta(\lambda_j = y)$. Thus, we have:

$$\mathbb{E}_{\Lambda\sim p_\theta(\cdot|\vec{y})}\left[\left(\Lambda\Lambda^T\right)_{i,j}\right] = \begin{cases}\left(\sum_{k=1}^m (2\alpha_k - 1)^2\right)y^{(i)}y^{(j)} & i \neq j \\ m & i = j\end{cases}$$

We can alternatively express this in matrix form; letting $\kappa = \sum_{k=1}^m (2\alpha_k - 1)^2$:

$$\mathbb{E}\left[\Lambda\Lambda^T\right] = \kappa\vec{y}\vec{y}^T + (m-\kappa)I$$

We see that $\vec{y}$ is an eigenvector with eigenvalue $\kappa\,\|yv\|^2 + (m-\kappa)$, and that the remaining eigenvalues are $(m-\kappa)$; thus $\vec{y}$ is the top eigenvector. Thus, Gosh et. al. propose to take the sign of the top eigenvector of $\Lambda\Lambda^T$ as an estimate of $\vec{y}$ (from which we can of course compute θ). A remaining issue in this (and other similar) approaches is resolving a final, fundamental symmetry: note that if $\vec{y}$ is an eigenvector, then so is $-\vec{y}$. Gosh et. al. propose using a single "trusted" non-adversarial label source to resolve this ambiguity.

Other spectral approaches look at the matrix $\Lambda^T\Lambda$ instead [Dalvi et al., 2013], combine spectral and EM approaches [Zhang et al., 2016b], and a range of other approaches. We note that while spectral approaches like the one above are appealing from a simplicity and theoretical analysis perspective, they can be difficult to extend to more complex label models.

Another line of related methods, generally referred to as *tensor decomposition* or *factorization* approaches, handle this type of latent variable problem by considering the equivalent of a spectral decomposition of higher-order tensors that are again usually formed as some function of the higher-order moments of the observed variables, e.g. Λ. For a good overview see [Anandkumar et al., 2014].

Further Modeling Challenges In this section, we looked at several traditional approaches to modeling weak supervision in the absence of ground truth labels. More broadly, however, we can view modeling weak supervision as consisting of three core challenges:

1. **Modeling accuracies:** Estimating the accuracies (or more specifically, the conditional probabilities $p_\theta(\boldsymbol{\lambda}|y)$) of the weak supervision sources.

2. **Modeling correlations:** Estimating and handling conditional correlations between the weak supervision sources, i.e. sources that are not conditionally-independent as in this section.

3. **Modeling expertise:** Estimating the data point-conditional accuracies of the weak supervision sources, i.e. modeling the fact that they may be more or less accurate for different data points.

In the prior approaches we considered above, we only treated a simple version of (1); and in general, most prior methods do not handle more complex label models that address challenges which uniquely arise in the programmatic labeling setting we consider, such as (2) handling correlations between weak supervision sources. In this thesis we consider weak supervision modeling approaches which can handle this broader class of challenges, in particular focusing on (1) and (2) in settings more general than the simple (but often-considered) conditionally-independent model used in this section.

2.4 Data Management Systems for Training Data

The broader—and central—focus of this thesis beyond algorithmic approaches for modeling weak supervision is building end-to-end data management systems for training data. We cover the main *training data management* system proposed in this thesis, Snorkel[3], in detail in Chapter 4. Here, we provide a very brief overview of other data management efforts around or related to training data.

[3]`snorkel.org`

Crowdsourcing As reviewed in Section 2.3, the area of crowdsourcing—i.e. soliciting labels from low-cost and potentially unreliable on-demand workers—is one that has been traditionally studied from an algorithmic and theoretical perspective. However, a wide body of work has been done building and studying end-to-end crowdsourcing management systems, which handle all aspects ranging from data management and modeling to crowd worker interfaces, communication, and incentives. A full survey of this literature is beyond the scope of this thesis; for good high level overviews see [Yuen et al., 2011; Doan et al., 2011]. Broadly, the work of this thesis naturally subsumes some parts of these systems—in particular, the modeling of different crowd worker accuracies, and management of crowd label lineage—while being orthogonal to other aspects of this body of work, such as the workflow, incentive, and interface management of large crowd worker fleets.

Data Integration & Cleaning Two traditional problems in data management that are closely related to the assembly and maintenance of training datasets are *data integration* and *data cleaning*. In data integration, the traditional task definition is to integrate data from multiple different sources, and is often decomposed into sub-tasks such as data extraction, e.g. from unstructured raw input data; schema alignment; entity linkage; and data fusion, or the resolution of conflicts between data sources, which is often very related to the weak supervision modeling approaches in the previous section [Rekatsinas et al., 2017b; Zhao et al., 2012; Pochampally et al., 2014; Li et al., 2015]. For a good survey of traditional data integration techniques and systems, see [Dong and Srivastava, 2015]. Recently, there has been renewed interest in how traditional data integration techniques can both be used for machine learning, and in turn use machine learning; for a good survey see [Rekatsinas and Xin, 2018].

Data cleaning is the task of detecting and removing or correcting incorrect data records, traditionally approached in the database community from the perspective of identifying tuples that logically conflict with the pre-specified constraints and schema of a database [Cong et al., 2007; Papotti et al., 2013]. Recently, new techniques that leverage statistical signals of the data as well have been proposed [Rekatsinas et al., 2017a]. The broader process of preparing, remapping, and potentially featurizing data—including for use as training data in machine learning—has also received considerable attention from the data management community, often under the name of *data wrangling* [Kandel et al., 2011].

Broadly, this thesis proposes and considers a new type of data management system for programmatically building, modeling, and managing training datasets for machine learning (Chapter 4). However, this work clearly relates to and builds on the traditions and techniques of traditional data management tasks like integration, cleaning, and wrangling.

Data Management in ML Frameworks We note briefly that while a range of machine learning frameworks [Abadi et al., 2016; Paszke, 2017] and related work address various challenges around managing training data for supervised machine learning, they nearly universally assume that this data has already been labeled, e.g. by hand. This starting point of large labeled training sets as served as the bedrock for much of machine learning's meteoric progress over the last several years, but remains a quandary of ad-hoc and heavily manual efforts in real practice. This is the machine learning systems and data management gap that we centrally address in this thesis.

Chapter 3

Data Programming

In this chapter we introduce *data programming*, a new paradigm in which users write *labeling functions* to programmatically label training data, rather than labeling it by hand. These labeling functions serve as a simple abstraction for various forms of heuristic or *weak* supervision, but may be inaccurate, correlated with each other, and conflicting in their outputs. In data programming, we automatically estimate their accuracies and correlations in order to reweight and combine their outputs into a clean, confidence-weighted set of training labels.

In this chapter we introduce the basic abstraction of a labeling function and describe two novel approaches for modeling and estimating their accuracies and correlations along with accompanying theoretical results. We show that using these approaches, we can recover in a theoretically consistent way not just the accuracies of noisy labeling sources, as prior approaches have studied, but the correlations between programmatic labeling sources, and thereby subsume a wide range of prior ad hoc or heuristic weak supervision techniques.

In Chapter 4 we then present an end-to-end system for machine learning based around data programming, Snorkel, along with empirical results from applying data programming and Snorkel to various real-world problems, providing validation for the core thesis that enabling users to programmatically build, manage, and model training datasets can be a productive and accessible way to build machine learning applications.

Motivation Many of the major machine learning breakthroughs of the last decade have been catalyzed by the release of a new labeled training dataset.[1] Supervised learning approaches that use such datasets have increasingly become key building blocks of applications throughout science and industry. This trend has also been fueled by the recent empirical success of automated feature generation approaches, notably deep learning methods such as long short term memory (LSTM) networks [Hochreiter and Schmidhuber, 1997], which ameliorate the burden of feature engineering given large enough labeled training sets. For many real-world applications, however, large hand-labeled training sets do not exist, and are prohibitively expensive to create due to requirements that labelers be experts in the application domain. Furthermore, applications' needs often change, necessitating new or modified training sets.

Data Programming To help reduce the cost of training set creation, in this section we describe *data programming*, a paradigm for the programmatic creation of training datasets. Data programming extends the idea of *distant supervision*, in which an external knowledge base is mapped onto an input dataset to generate training examples [Mintz et al., 2009], and serves as a general framework for a broad range of noisier, higher-level labeling strategies, often referred to as *weak supervision* (see Chapter 2). In data programming, users provide a set of heuristic *labeling functions*, which are user-defined programs that each provide a label for some subset of the data, and collectively generate a large but noisy training set. These labeling functions can express a broad range of programmatic or weak supervision strategies—they can use external knowledge bases (as in distant supervision), model an individual annotator's labels (as in crowdsourcing), leverage a combination of domain-specific patterns and dictionaries, or use external pre-trained models. In Section 3.1, we outline the basic syntax of labeling functions; we then provide more detail and examples of labeling functions in Chapter 4, when we describe Snorkel, the system built around the concepts in data programming.

The core challenge inherent in data programming is that these labeling functions are a practically advantageous but extremely messy form of supervision. More concretely, they may have widely varying error rates, may overlap, and may conflict on certain data points.

[1] `http://www.spacemachine.net/views/2016/3/datasets-over-algorithms`

To address this, we model the labeling functions as a generative process, which lets us automatically denoise the resulting training set by learning the accuracies of the labeling functions along with their correlation structure. In turn, we use this model of the training set to optimize a stochastic version of the loss function of the discriminative model that we desire to train.

However, it is not at all obvious how we can solve for the parameters of this model—e.g. the accuracies and correlation weights of the labeling functions—given that we do not necessarily observe any ground truth labels. We show that we can in fact provably recover these parameters even in the absence of ground truth, and outline two approaches for doing so: in Section 3.2, by minimizing the maximum marginal likelihood of the observed labeling function outputs, using stochastic gradient descent and Gibbs sampling; and in Section 3.3, using a matrix completion-style approach over a specialized form of the covariance matrix of these labeling function outputs. In each setting, we show theoretically that, given certain conditions on the labeling functions, our method achieves the same asymptotic scaling as supervised learning methods, but that our scaling depends on the amount of *unlabeled* data—using only a fixed number of labeling functions, which is small relative to the training set size.

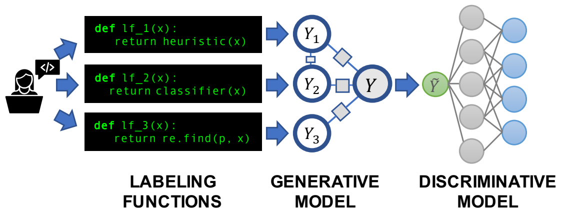
Data programming is in part motivated by the challenges that users faced when applying prior programmatic supervision approaches, and is intended to be a new software engineering paradigm for the creation and management of training sets. For example, consider the scenario when two labeling functions of differing quality and scope overlap and possibly conflict on certain training examples; in prior approaches the user would have to decide which one to use, or how to somehow integrate the signal from both. In data programming, we accomplish this automatically by learning a model of the training set that includes both labeling functions. Additionally, users are often aware of, or able to induce, dependencies between their labeling functions. In data programming, users can provide a dependency graph to indicate, for example, that two labeling functions are similar, or that one “fixes” or “reinforces” another. We describe cases in which we can learn the strength of these dependencies, and for which our generalization is again asymptotically identical to the supervised case.

One further motivation for our method is driven by the observation that users often struggle with selecting *features* for their models, which is a traditional development bottleneck given fixed-size training sets. However, feedback from users suggests that writing labeling functions in the framework of data programming may be easier (see Chapter 4 for further detail). While the impact of a feature on end performance is dependent on the training set and on statistical characteristics of the model, a labeling function has a simple and intuitive optimality criterion: that it labels data correctly. Motivated by this, we explore whether we can flip the traditional machine learning development process on its head, having users instead focus on generating training sets large enough to support automatically-generated features.

Outline of Chapter In this chapter we describe data programming, a new paradigm for the programmatic labeling, modeling, and integration of training datasets for machine learning:

- *In Section 3.1* we describe the basic idea and syntax of a labeling function, and the basic model we use to model their differing qualities and correlations in order to ultimately reweight and combine their outputs into clean training labels.
- *In Section 3.2* we describe an approach to learning this model without ground truth labels, by maximizing the marginal likelihood of the observed outputs with stochastic gradient descent and Gibbs sampling.
- *In Section 3.3,* we describe an alternative approach using a matrix completion-style objective.
- *In Section 3.4* we briefly describe methods to learn the structure of labeling function correlations and more complex variants of the model.
- *In Section 3.5* we outline related work on modeling weak supervision without ground truth labels.

In Chapter 4, we then present an end-to-end system for machine learning built around the core paradigm of data programming, Snorkel, along with further empirical results from applying data programming and Snorkel to various real-world problems.



```
def lf_1(x):
   return heuristic(x)
```

```
def lf_2(x):
  return classifier(x)
```

```
def lf_3(x):
   return re.find(p, x)
```

Figure 3.1: In data programming, rather than labeling training data by hand, users write *labeling functions*, which programmatically label data points or abstain. These labeling functions will have different unknown accuracies and correlations; we model and combines their outputs using a generative *label model*, and then use the resulting probabilistic labels to train an end discriminative model.

3.1 A Syntax and Model for Weak Supervision

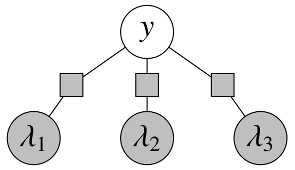
In many applications, we would like to use machine learning, but we face the following challenges: (i) *hand-labeled* training data is not available, and is prohibitively expensive to obtain in sufficient quantities as it requires expensive domain experts; (ii) *related external knowledge bases* are either unavailable or insufficiently specific, precluding a traditional distant supervision or co-training approach; (iii) *application specifications* are in flux, changing the model we ultimately wish to learn.

In such a setting, we would like a simple, scalable and adaptable approach for supervising a model applicable to our problem. More specifically, in a more theoretical phrasing: we would ideally like our approach to achieve ϵ expected loss with high probability, given $O(1)$ *inputs* of some sort from a domain-expert user, rather than the traditional $\tilde{O}(\epsilon^{-2})$ hand-labeled training examples required by most supervised methods (where $\tilde{O}$ notation hides logarithmic factors). To this end, we propose *data programming*, a paradigm for the programmatic creation of training sets, which enables domain-experts to more rapidly train machine learning systems and has the potential for this type of scaling of expected loss. In data programming, rather than manually labeling each example, users instead describe the *processes by which* these points could be labeled by providing a set of heuristic rules, or other programmatic labelers, called *labeling functions*.

The overall goal of this approach is to label training data for a final discriminative model we are aiming to train (Figure 3.1). That is, the labeling functions do not need to be executable at test time, nor do they need to be comprehensive in what they label;

```
def lambda_1(x):
    return 1 if (x.gene,x.pheno) in KNOWN_RELATIONS_1 else None

def lambda_2(x):
    return (*@-@*)1 if re.match(r'.*not_cause.*', x.text_between) else None

def lambda_3(x):
    return 1 if re.match(r'.*associated.*', x.text_between)
              and (x.gene,x.pheno) in KNOWN_RELATIONS_2 else None
```

(a) An example set of three labeling functions written by a user, with `None` representing an abstention.



(b) The generative model of a training set defined by the user input (unary factors omitted).

Figure 3.2: An example genomics application in which our goal is to extract mentions of gene-disease relations (roughly, *"Gene A causes disease B"*) from the scientific literature.

our goal is simply to use their output labels—re-weighted and combined by the generative modeling approach in data programming, to be described—to train a model that can learn to generalize beyond their labels (see Chapter 4 for further detail here).

3.1.1 Labeling Functions

Formally, given input data points $x \in \mathcal{X}$, and output labels $y \in \mathcal{Y}$, a labeling function $\lambda_j : \mathcal{X} \mapsto \mathcal{Y} \cup \{\emptyset\}$ is a user-defined function that encodes some domain heuristic, which either provides a label, or abstains to provide one (denoted by $\emptyset$), for $x \in \mathcal{X}$. As part of a *data programming specification*, a user provides some m labeling functions, which we denote in vectorized form as $\lambda : \mathcal{X} \mapsto (\mathcal{Y} \cup \{\emptyset\})^m$. In this section, we will consider the binary classification case, where $\mathcal{Y} = \{-1, 1\}$, for simplicity.

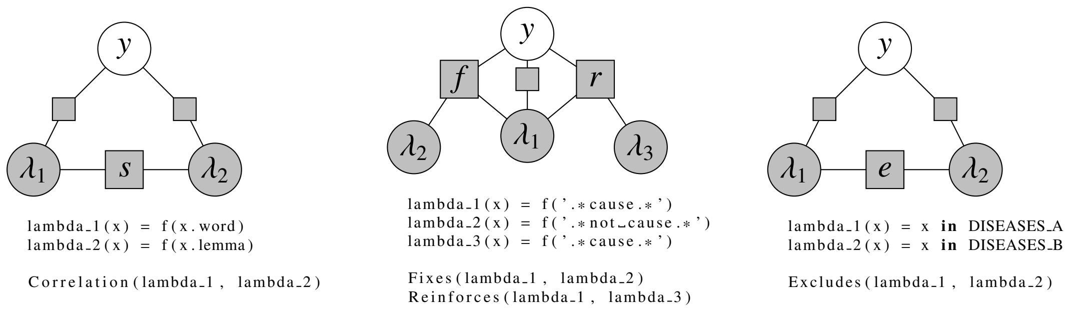
Example 3.1.1. To gain intuition about labeling functions, we describe a simple text relation extraction example. In Figure 3.2, we consider the task of classifying co-occurring gene and disease mentions as either expressing a causal relation or not. For example, given the sentence "Gene A causes disease B", the object $x = (A, B)$ has true class $y = 1$, meaning it is indeed a relation we wish to extract into our knowledge base. To construct a training set, the user writes three labeling functions (Figure 3.2a). In λ_1, an external structured knowledge base is used to label a few objects with relatively high accuracy, and is equivalent to a traditional distant supervision rule (see Chapter 2). λ_2 uses a purely heuristic approach to label a much larger number of examples with lower accuracy. Finally, λ_3 is a "hybrid" labeling function, which leverages both an external knowledge base and a heuristic filter.

A labeling function need not have perfect accuracy or recall; rather, it represents a pattern that the user wishes to impart to their model and that is easier to encode as a labeling function than as a set of hand-labeled examples. As illustrated in Ex. 3.1.1, labeling function can be based on external knowledge bases, libraries or ontologies, could be purely a heuristic pattern, or some hybrid of these types; see Chapter 4 for further detail and examples. The use of labeling functions is also strictly more general than manual annotations, as a manual annotation can always be directly encoded by a labeling function. Importantly, labeling functions can overlap, conflict, and even have dependencies which users can provide as part of the data programming specification, or learn through various automated approaches (Section 3.4); our approach provides a simple framework for these inputs.

3.1.2 Generative Label Models

We now outline three versions of the generative *label model* that models the qualities and correlations of the labeling functions: (i) a simple model where we consider modeling binary labeling functions as conditionally independent; (ii) an extended model where we consider modeling binary labeling functions with different types of pairwise dependencies; (iii) and a more general model of k-ary labeling functions with arbitrary pairwise correlations.

Conditionally-Independent, Binary, and Class-Symmetric Model We first describe a simple model in which (i) the labeling functions label independently, given the true label class—in other words, we model them as being conditionally independent, $\lambda_i \perp \lambda_{j \neq i} | y$, or equivalently making uncorrelated errors; (ii) the labeling functions abstain uniformly; and (iii) the labeling functions have the same accuracy regardless of the true underlying class. Under this model, each labeling function λ_j has some probability $\beta_j = P(\lambda_j \neq \emptyset)$ of labeling an object and then some class-symmetric probability $\alpha_j = P(\lambda_j = y | \lambda_j \neq \emptyset)$ of labeling the object correctly; for simplicity, in this model we also assume a uniform prior, i.e. $P(y = 1) = P(y = -1) = 1/2$. Let the label model parameters be $\theta = [\alpha; \beta]$, then this



```
lambda_1(x) = f(x.word)
lambda_2(x) = f(x.lemma)

Correlation(lambda_1, lambda_2)
```

```
lambda_1(x) = f('.*cause.*')
lambda_2(x) = f('.*not_cause.*')
lambda_3(x) = f('.*cause.*')

Fixes(lambda_1, lambda_2)
Reinforces(lambda_1, lambda_3)
```

```
lambda_1(x) = x in DISEASES_A
lambda_2(x) = x in DISEASES_B

Excludes(lambda_1, lambda_2)
```

Figure 3.3: Examples of labeling function dependency predicates.

label model has distribution, for a single data point x with true (unobserved) label y:

$$p_\theta(\lambda, y) = \frac{1}{2} Z_\theta^{-1} \prod_{j=1}^{m} \left(\beta_j \alpha_j \mathbb{1}\left\{\lambda_j = y\right\} + \beta_j (1-\alpha_j) \mathbb{1}\left\{\lambda_j = -y\right\} + (1-\beta_j)\mathbb{1}\left\{\lambda_j = \emptyset\right\}\right), \quad (3.1)$$

where $\lambda \in \{-1, 1, \emptyset\}^m$ contains the labels output by the labeling functions for data point x and Z_θ is the normalizing partition function. If we allow the parameters $\alpha \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^m$ to vary, (3.1) specifies a family of generative label models, similar (but slightly more complex, due to the abstains, than) the simple conditionally-independent weak supervision model introduced in Section 2.3.

Pairwise-Dependent Binary Model Next we describe a model of binary labeling functions where assumptions (i-iii) are relaxed, in other words we consider that labeling functions may have dependencies, and may have class-specific parameters, $\alpha_{(j,\lambda,y)} = P(\lambda_j = \lambda | y)$, including for abstains (i.e. modeling non-uniform abstentions). This first relaxation is motivated by the fact that users often write labeling functions that have clear dependencies among them (Chapter 3). As more labeling functions are added as the system is developed, an implicit dependency structure arises naturally amongst the labeling functions: modeling these dependencies can in some cases significantly improve accuracy, and/or mitigate error modes, such as e.g. double-counting the 'votes' of highly-correlated labeling functions. We describe a method by which the user can specify this dependency knowledge as a *dependency graph*, and show how the system can use it to produce better parameter estimates.

To support the injection of dependency information into the model, we augment the data programming specification with a *label function dependency graph*, $G_\lambda \subset D \times \{1, \ldots, m\} \times \{1, \ldots, m\}$, which is an undirected graph over the labeling functions, each of the edges of which is associated with a *dependency type* from a class of dependencies D appropriate to the domain.

In some settings we have utilized four commonly-occurring types of dependencies as illustrative examples: *correlation*, *fixing*, *reinforcing*, and *exclusive* (see Figure 3.3). For example, suppose that we have two functions λ_1 and λ_2, and λ_2 typically labels only when (i) λ_1 also labels, (ii) λ_1 and λ_2 disagree in their labeling, and (iii) λ_2 is actually correct. We call this a fixing dependency, since λ_2 *fixes* mistakes made by λ_1. If λ_1 and λ_2 were to typically agree rather than disagree, this would be a reinforcing dependency, since λ_2 reinforces the label output by λ_1.

The presence of dependency information means that we can no longer model our labels using the simple Bayesian network in (3.1). Instead, we model our distribution as a factor graph. This standard technique lets us describe the family of generative distributions in terms of a known *factor function* $\psi : \{-1, 1, \emptyset\}^m \times \{-1, 1\} \mapsto \{-1, 0, 1\}^M$ (in which each entry ψ_i represents a factor), and an unknown parameter $\theta \in \mathbb{R}^M$ as

$$p_\theta(\lambda, y) = Z_\theta^{-1} \exp(\theta^T \psi(\lambda, y)),$$

where Z_θ is the *partition function* which ensures that p_θ is a distribution. Next, we will describe how we define ψ using information from the dependency graph.

To construct f, we will start with some base factors, which we inherit from (3.1), and then augment them with additional factors representing dependencies. To simplify, we will let $\emptyset = 0$. Then, for all $j \in \{1, \ldots, m\}$, we let

$$\psi_0(\lambda, y) = y, \quad \psi_j(\lambda, y) = \lambda_j y, \quad \psi_{m+j}(\lambda, y) = \lambda_j, \quad \psi_{2m+j}(\lambda, y) = \lambda_j^2 y, \quad \psi_{3m+j}(\lambda, y) = \lambda_j^2.$$

These factors alone are sufficient to describe any distribution for which the labels are mutually independent, given the class: this includes the independent family in (3.1).

We now proceed by adding additional factors to ψ, which model the dependencies encoded in G_λ. For each dependency edge (d, i, j), we add one or more factors to ψ as follows.

For a near-duplicate dependency on (i, j), we add a single factor $\psi_\iota(\lambda, y) = \mathbb{1}\left\{\lambda_i = \lambda_j\right\}$, which increases our prior probability that the labels will agree. For a fixing dependency, we add two factors, $\psi_\iota(\lambda, y) = \mathbb{1}\left\{\lambda_i = \emptyset \wedge \lambda_j \neq \emptyset\right\}$ and $\psi_{\iota+1}(\lambda, y) = \mathbb{1}\left\{\lambda_i = -y \wedge \lambda_j = y\right\}$, which encode the idea that λ_j labels only when λ_i does, and that λ_j fixes errors made by λ_i. The factors for a reinforcing dependency are the same, except that $\psi_{\iota+1}(\lambda, y) = \mathbb{1}\left\{\lambda_i = y \wedge \lambda_j = y\right\}$. Finally, for an exclusive dependency, we have a single factor $\psi_\iota(\lambda, y) = -\mathbb{1}\left\{\lambda_i \neq \emptyset \wedge \lambda_j \neq \emptyset\right\}$.

The theoretical analysis in Section 3.2.2 covers this full set of arbitrary dependency types. However, in the rest of this dissertation, we will focus on basic pairwise correlation dependencies for simplicity of exposition, in which case $G_\lambda \subset D \times \{1, \ldots, m\} \times \{1, \ldots, m\}$.

Pairwise-Dependent *k*-ary Model Finally, in Section 3.3 we will consider the more general version of the Pairwise-Dependent Binary Model where we handle k-ary (categorical) labeling functions—i.e. labeling functions that have outputs in some discrete set $\mathcal{Y} = \{1, ..., k\}$.

3.1.3 Training an End Discriminative Model

The ultimate goal of the label models outlined above, and data programming overall, is to generate training labels for some end discriminative model that can generalize beyond the information expressed in the labeling functions. More specifically, the reason for training this final model—rather than e.g. using the label model's predicted labels as the final outputs—is to leverage modern machine learning tools to generalize to new features, and thus either (a) learn to cover data points not labeled by the provided labeling functions, and/or (b) produce a model defined over features different than those the labeling functions apply to. For further details, see Chapter 4.

To do this, we start by using the estimated label model parameters θ to output a final predicted, *probabilistic* training label $\tilde{y} = p_\theta(y|\lambda)$ (note that this is equivalent to re-weighting and combining the individual labeling function labels). Given the parameters θ, performing this inference is straightforward; see Sections 3.2.1 and 3.3.2 for further details.

We then train a discriminative model h_w on our probabilistic labels $\tilde{y}$ by minimizing a *noise-aware* variant of the loss $l(h_w(x^{(i)}), y^{(i)})$, i.e., the expected loss with respect to $\tilde{y}$:

$$\hat{w} = \text{argmin}_w \sum_{i=1}^{n} \mathbb{E}_{y \sim p_\theta(\cdot|\lambda^{(i)})} \left[l(h_w(x^{(i)}), y) \right]$$

We now return to the core data programming challenge of learning the label model parameters θ in the absence of ground truth labels, using two different approaches in Sections 3.2 and 3.3 respectively.

3.2 Maximum Marginal Likelihood Approach

The core technical challenge introduced by the generative *label models* defined in the previous section is that of how to learn their parameters—e.g., the labeling function accuracy and correlation parameters—*without observing any ground truth labels*. In this section, we outline an approach based on using stochastic gradient descent and Gibbs sampling to maximize the marginal likelihood, and provide a theoretical analysis showing conditions under which it converges, and in fact leads to end-to-end sample complexity (i.e. the number of samples labeled by the labeling functions and label model and then used to train a final end discriminative model) that has the same asymptotic scaling as in supervised learning methods, except *with respect to number of unlabeled data points*.

3.2.1 Learning the Label Model

Our goal will be to learn which parameters θ are most consistent with our observations—our unlabeled training set—using maximum likelihood estimation. To do this for a particular training set $X_U = \{x^{(1)}, \dots, x^{(n)}\}$, we will maximize the *log marginal likelihood* of the outputs of the labeling functions applied to X_U, the label matrix $\Lambda \in \mathbb{R}^{n \times m}$:

$$\begin{aligned}
\hat{\theta} &= \mathrm{argmax}_\theta\ L_\Lambda(\theta) \\
&= \mathrm{argmax}_\theta\ \log p_\theta(\Lambda) \\
&= \mathrm{argmax}_\theta\ \sum_{i=1}^{n} \log p_\theta(\lambda^{(i)}) \\
&= \mathrm{argmax}_\theta\ \sum_{i=1}^{n} \log \left(\sum_{y' \in \mathcal{Y}} p_\theta(\lambda^{(i)}, y') \right)
\end{aligned} \tag{3.2}$$

In other words, we are maximizing the probability that the observed labels produced on our training examples occur under the generative model in (3.1). We can start by taking the gradient of L_Λ with respect to the unknown parameters θ.

$$\begin{aligned}
\nabla_\theta L_\Lambda(\theta) &= \sum_{i=1}^{n} \nabla_\theta \log \left(\sum_{y' \in \mathcal{Y}} p_\theta(\lambda^{(i)}, y') \right) \\
&= \sum_{i=1}^{n} \left(\sum_{y' \in \mathcal{Y}} p_\theta(\lambda^{(i)}, y') \right)^{-1} \left(\sum_{y' \in \mathcal{Y}} \nabla_\theta p_\theta(\lambda^{(i)}, y') \right) \\
&= \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} p_\theta(\lambda^{(i)})^{-1} \nabla_\theta p_\theta(\lambda^{(i)}, y') \\
&= \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} p_\theta(\lambda^{(i)})^{-1} \left(-Z_\theta^{-2} \exp(\theta^T \psi(\lambda^{(i)}, y')) \left(\nabla_\theta Z_\theta \right) + Z_\theta^{-1} \exp(\theta^T \psi(\lambda^{(i)}, y')) \psi(\lambda^{(i)}, y') \right) \\
&= \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} p_\theta(\lambda^{(i)})^{-1} p_\theta(\lambda^{(i)}, y') \left(\psi(\lambda^{(i)}, y') - Z_\theta^{-1} \left(\nabla_\theta Z_\theta \right) \right) \\
&= \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} p_\theta(y' | \lambda^{(i)}) \left(\psi(\lambda^{(i)}, y') - \sum_{\lambda', y''} Z_\theta^{-1} \exp(\theta^T \psi(\lambda', y'')) \psi(\lambda', y'') \right) \\
&= \sum_{i=1}^{n} \left(\sum_{y' \in \mathcal{Y}} p_\theta(y' | \lambda^{(i)}) \psi(\lambda^{(i)}, y') - \sum_{\lambda', y''} p_\theta(\lambda', y'') \psi(\lambda', y'') \right) \\
&= \sum_{i=1}^{n} \left(\mathbb{E}_{y' \sim p_\theta(\cdot | \lambda^{(i)})} \left[\psi(\lambda^{(i)}, y') \right] - \mathbb{E}_{(\lambda', y') \sim p_\theta} \left[\psi(\lambda', y') \right] \right)
\end{aligned}$$

We see that the gradient is just the difference between the expected sufficient statistics and expected conditional statistics given the observed labels Λ. Importantly, we can compute these two quantities using an approximate inference algorithm, for example Gibbs sampling. Thus, to estimate the parameters θ, in practice we can interleave approximate inference (e.g. Gibbs sampling) and stochastic gradient descent steps, leverage the frameworks for doing both of these efficiently. For further details see Appendix B.

3.2.2 Theoretical Analysis

For the theoretical analysis of the maximum marginal likelihood approach, we focus on a binary classification task in which we have a distribution $\mathcal{D}$ over object and class pairs $(x, y) \in \mathcal{X} \times \{-1, 1\}$, and we are concerned with minimizing the logistic loss under a linear model given some *features*,

$$R(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log(1 + \exp(-w^T\phi(x)y))\right],$$

where without loss of generality, we assume that $\|\phi(x)\| \leq 1$. Given that our parameter learning phase has successfully found some parameters $\hat{\theta}$ that accurately describe the training set, we can then proceed to estimate the parameter w which minimizes the expected risk of a linear model over our feature mapping ϕ, given $\hat{\theta}$. To do so, we define the *noise-aware empirical risk* $\hat{R}_{\hat{\theta}}$ with regularization parameter ρ, and compute the *noise-aware empirical risk minimizer*

$$\hat{w} = \operatorname{argmin}_w \hat{R}_{\hat{\theta}}(w, X_U) = \operatorname{argmin}_w \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{y'\sim p_{\hat{\theta}}(\cdot|\lambda^{(i)})}\left[\log\left(1 + \exp(-w^T\phi(x)y')\right)\right] + \rho\,\|w\|^2 \tag{3.3}$$

This is a logistic regression problem, so it can be solved using stochastic gradient descent as well.

Conditionally Independent, Binary, and Class-Symmetric Model We will start by analyzing the conditionally independent case. In order to expose the scaling of the expected loss as the size of the unlabeled dataset changes, we will assume here that $0.3 \leq \beta_j \leq 0.5$ and $0.8 \leq \alpha_j \leq 0.9$. We note that while these arbitrary constraints can be changed, they are roughly consistent with our applied experience, where users tend to write high-accuracy and high-coverage labeling functions.

We can in fact prove that stochastic gradient descent running on (3.2) and (3.3) is guaranteed to produce accurate estimates, under conditions which we describe now. First, the problem distribution π needs to be accurately modeled by some distribution μ in the family that we are trying to learn. That is, for some α^* and β^*, with $\theta^* = [\alpha^*; \beta^*]$,

$$\forall \lambda \in \{-1, 0, 1\}^m, y \in \{-1, 1\},\ p_{\pi^*}(\lambda, y) = p_{\theta^*}(\lambda, y). \tag{3.4}$$

Second, given an example $(x, y) \sim \pi^*$, the class label y must be independent of the features $\phi(x)$ given the labels λ. That is,

$$(x, y) \sim \pi^* \Rightarrow y \perp \phi(x) \mid \lambda. \tag{3.5}$$

This assumption encodes the idea that the labeling functions, while they may be arbitrarily dependent on the features, provide sufficient information to accurately identify the class. Third, we assume that the algorithm used to solve (3.3) has bounded generalization risk such that for some parameter χ,

$$\mathbb{E}_{\hat{w}}\left[\mathbb{E}_{X_U}\left[R_{\hat{\theta}}(\hat{w}, X_U)\right] - \min_{w} \mathbb{E}_{X_U}\left[R_{\hat{\theta}}(w, X_U)\right]\right] \leq \chi. \tag{3.6}$$

Under these conditions, we make the following statement about the accuracy of our estimates, which is a simplified version of a theorem that is detailed in Appendix B.

Theorem 1. Suppose that we run data programming, solving the problems in (3.2) and (3.3) using stochastic gradient descent to produce $\hat{\theta} = [\hat{\alpha}; \hat{\beta}]$ and $\hat{w}$. Suppose further that our setup satisfies the conditions (3.4), (3.5), and (3.6), and suppose that $m \geq 2000$. Then for any $\epsilon > 0$, if the number of labeling functions m and the size of the input dataset $n = |X_U|$ are large enough that

$$n \geq \frac{356}{\epsilon^2} \log\left(\frac{m}{3\epsilon}\right)$$

then our expected parameter error and generalization risk can be bounded by

$$\mathbb{E}\left[\|\hat{\alpha}-\alpha^*\|^2\right] \leq m\epsilon^2 \qquad \mathbb{E}\left[\left\|\hat{\beta}-\beta^*\right\|^2\right] \leq m\epsilon^2 \qquad \mathbb{E}\left[R(\hat{w})-\min_w R(w)\right] \leq \chi + \frac{\epsilon}{27\rho}.$$

We select $m \geq 2000$ to simplify the statement of the theorem and impart a sense for how ϵ scales with respect to n. The full theorem with scaling in each parameter (and for arbitrary m) is presented in Appendix B.

This result establishes that to achieve both expected loss and parameter estimate error ϵ, it suffices to have only $m = O(1)$ labeling functions and $n = \tilde{O}(\epsilon^{-2})$ training examples, which is the same asymptotic scaling exhibited by methods that use labeled data. This means that data programming achieves the same learning rate as methods that use labeled data, while requiring asymptotically less work from its users, who need to specify $O(1)$ labeling functions rather than manually label $\tilde{O}(\epsilon^{-2})$ examples. In contrast, in the crowdsourcing setting [Karger et al., 2011], the number of workers m tends to infinity while here it is constant while the dataset grows. These results provide some explanation of why our experimental results (e.g. in Chapter 4) suggest that a small number of rules with a large unlabeled training set can be effective at even complex natural language processing tasks.

Pairwise-Dependent Binary Model We can again solve a maximum likelihood problem like (3.2) to learn the parameter $\hat{\theta}$. Using the results, we can continue on to find the noise-aware empirical loss minimizer by solving the problem in (3.3). In order to solve these problems in the dependent case, we use stochastic gradient descent, using Gibbs sampling to sample from the distributions used in the gradient update. Under conditions similar to those in the previous case, we can again provide a bound the accuracy of these results. We define these conditions now. First, there must be some set $\Theta \subset \mathbb{R}^M$ that we know our parameter lies in. This is analogous to the assumptions on α_j and β_j in the previous case, and we can state the following analog of (3.4):

$$\exists\theta^* \in \Theta \text{ s.t. } \forall(\lambda, y) \in \{-1,0,1\}^m \times \{-1,1\},\ p_{\pi^*}(\lambda, y) = p_{\theta^*}(\lambda, y). \tag{3.7}$$

Second, for any $\theta \in \Theta$, it must be possible to accurately learn θ from full (i.e. labeled) samples of p_θ. More specifically, there exists an unbiased estimator $\hat{\theta}(T)$ that is a function of some dataset T of independent samples from p_θ such that, for some $c > 0$ and for all $\theta \in \Theta$,

$$\mathbf{Cov}\left(\hat{\theta}(T)\right) \preceq (2c\,|T|)^{-1} I. \tag{3.8}$$

Third, for any two feasible models θ_1 and $\theta_2 \in \Theta$,

$$\mathbb{E}_{(\lambda_1,y_1)\sim p_{\theta_1}}\left[\mathbf{Var}_{(\lambda_2,y_2)\sim p_{\theta_2}}(y_2|\lambda_1 = \lambda_2)\right] \leq cM^{-1}. \tag{3.9}$$

That is, we'll usually be reasonably sure in our guess for the value of y, even if we guess using distribution p_{θ_2} while the the labeling functions were actually sampled from (the possibly totally different) p_{θ_1}. We can now prove the following result about the accuracy of our estimates.

Theorem 2. Suppose that we run stochastic gradient descent to produce $\hat{\theta}$ and $\hat{w}$, and that our setup satisfies the conditions (3.5)-(3.9). Then for any $\epsilon > 0$, if the input dataset X_U, $n = |X_U|$, is large enough that

$$n \geq \frac{2}{c^2\epsilon^2}\log\left(\frac{2\,\|\theta_0 - \theta^*\|^2}{\epsilon}\right),$$

then our expected parameter error and generalization risk can be bounded by

$$\mathbb{E}\left[\left\|\hat{\theta} - \theta^*\right\|^2\right] \leq M\epsilon^2 \qquad\qquad \mathbb{E}\left[R(\hat{w}) - \min_w R(w)\right] \leq \chi + \frac{c\epsilon}{2\rho}.$$

As in the independent case, this shows that we need only $n = \tilde{O}(\epsilon^{-2})$ unlabeled training examples to achieve error $O(\epsilon)$, which is the same asymptotic scaling as supervised learning methods. This suggests that while we pay a computational penalty for richer dependency structures, we are no less statistically efficient. In Appendix B, we provide more details, including an explicit description of the algorithm and the step size used to achieve this result.

		KBP (News)			Genomics			Pharmacogenomics		
DM	WS	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
HT	ITR	**51.15**	26.72	35.10	83.76	41.67	55.65	68.16	49.32	57.23
	DP	50.52	**29.21**	**37.02**	**83.90**	43.43	57.24	**68.36**	54.80	**60.83**
LSTM	ITR	37.68	28.81	32.66	69.07	**50.76**	58.52	32.35	43.84	37.23
	DP	47.47	27.88	35.78	75.48	48.48	**58.99**	37.63	47.95	42.17

Table 3.1: Precision/Recall/F1 scores using two different weak supervision (WS) approaches- data programming (DP) and a distant supervision ITR approach, with two end discriminative models (DM)- one using hand-tuned (HT) features, and an LSTM.

3.2.3 Experiments

We experimentally validate two claims about our approach, data programming, using the maximum marginal likelihood approach as detailed in this section: first, that it can be an effective paradigm for building high quality machine learning systems, which we test across three real-world text relation extraction applications; and second, that it can be used successfully in conjunction with automatic feature generation methods, such as LSTM models.

Relation Mention Extraction Tasks In the *relation mention extraction* task, our objects are relation mention *candidates* $x = (e_1, e_2)$, which are pairs of entity mentions e_1, e_2 in unstructured text, and our goal is to learn a model that classifies each candidate as either a true textual assertion of the relation $R(e_1, e_2)$ or not. We examine a news application from the 2014 TAC-KBP Slot Filling challenge[2], where we extract relations between real-world entities from articles; a clinical genomics application, where we extract causal relations between genetic mutations and phenotypes from the scientific literature; and a pharmacogenomics application where we extract interactions between genes, also from the scientific literature.

For each application, we or our collaborators originally built a system where a ground truth training set was programmatically generated by ordering the labeling functions as a sequence of if-then-return statements, and for each candidate, taking the first label emitted by this script as the training label. We refer to this as the *if-then-return (ITR)* approach, and note that it often required significant domain expert development time to tune (weeks or

[2]`http://www.nist.gov/tac/2014/KBP/`

more). For this set of experiments, we then used the same labeling function sets within the framework of data programming. In Table 3.1, we see that we achieve consistent improvements: on average by 2.34 points in F1 score, including what would have been a winning score on the 2014 TAC-KBP challenge [Surdeanu and Ji, 2014].

We observed these performance gains across applications with very different labeling function sets. We describe the labeling function summary statistics—*coverage* is the percentage of objects that had at least one label, *overlap* is the percentage of objects with more than one label, and *conflict* is the percentage of objects with conflicting labels—and see in Table 3.2 that even in scenarios where m is small, and conflict and overlap is relatively less common, we still realize performance gains.

Application	# of LFs	Coverage	Overlap	Conflict	F1 Score Improvement	
					HT	LSTM
KBP (News)	40	29.39	1.38	0.15	1.92	3.12
Genomics	146	53.61	26.71	2.05	1.59	0.47
Pharmacogenomics	7	7.70	0.35	0.32	3.60	4.94
Diseases	12	53.32	31.81	0.98	N/A	N/A

Table 3.2: Labeling function summary statistics, and relative F1 score improvement over baseline IRT methods for hand-tuned (HT) and LSTM-generated (LSTM) feature sets. We include labeling function statistics from the usability study's disease mention tagging application as well, where baseline scores were not available.

Automatically-generated Features We additionally compare both hand-tuned and automatically generated features, where the latter are learned via an LSTM recurrent neural network (RNN) [Hochreiter and Schmidhuber, 1997]. Conventional wisdom states that deep learning methods such as RNNs are prone to overfitting, thus rendering them ineffective over distantly-supervised training sets. In our experiments, however, we find that training them with the data programming may be effective, reporting a 9.79 point boost to precision and a 3.12 point F1 score improvement on the benchmark 2014 TAC-KBP (News) relation extraction task, over the baseline *if-then-return* approach. Additionally for comparison, our approach is a 5.98 point F1 score improvement over a state-of-the-art LSTM approach applied to the TAC-KBP task which was trained on hand-labeled data [Verga et al., 2015]. For further experimental validation of the general data programming paradigm, see Chapter 4.

3.3 Matrix Completion-Style Approach

In this section, we outline an alternative approach to learning the parameters of the label model using a simple and scalable matrix completion-style algorithm, which we are able to analyze by applying strong matrix concentration bounds [Tropp, 2015]. This approach is advantageous in several ways. First, it leads to a more computationally efficient algorithm that, after an initial matrix multiply and (optionally) matrix inversion—both of which can be computed quickly using standard linear algebra libraries—it only requires optimizing (e.g. running SGD) over a $M \times M$, where M is proportional to the number of cliques of dependent labeling functions, which importantly has no dependence on the number of unlabeled data points being used, leading to over 100× faster runtimes compared to prior Gibbs-sampling based approaches [Ratner et al., 2016; Platanios et al., 2017], including the one in Section 3.2, and enabling simple implementation using libraries like PyTorch.

Second, many dependency structures between weak supervision labeling functions may lead to non-identifiable models of their accuracies, where a unique solution cannot be recovered. We provide a compiler-like check to establish identifiability—i.e. the existence of a unique set of source accuracies—for arbitrary dependency structures, without resorting to the standard assumption of non-adversarial labeling functions [Dawid and Skene, 1979], alerting users to this potential stumbling block that we have observed in practice.

Third, we provide sample complexity bounds that characterize the benefit of adding additional unlabeled data and the scaling with respect to the user-specified dependency structure. While previous approaches, such as the one in Section 3.2, required thousands of labeling functions to give non-vacuous bounds, we capture regimes with small numbers of labeling functions, better reflecting the real-world uses of weak supervision we have observed.

Finally, this approach can be extended to the *multi-task* setting, which we cover in Chapter 5.

3.3.1 Learning the Label Model: Simple Example

In order to establish the basic intuition behind approaching the label model parameter estimation problem as a matrix-completion style one, we will start by illustrating a matrix-completion style solution to the simple conditionally-independent, binary, class-symmetric model. In the next subsection, we will then show how it can be extended to more complex label models, e.g. those with arbitrary correlations between the weak supervision sources.

We start here by considering what we refer to here as the *empirical overlaps matrix*, $\hat{O} = \Lambda^T \Lambda$, where we see that for some $i \neq j$:

$$\frac{1}{n}\hat{O}_{i,j} = \frac{1}{n}\sum_{k=1}^{n} \lambda_i^{(k)} \lambda_j^{(k)} = \hat{\mathbb{E}}\left[\lambda_i \lambda_j\right]$$

We therefore consider $\hat{O}$ to be the noisy empirical version of a true overlaps matrix O with entries, for $i \neq j$:

$$\begin{aligned} O_{i,j} &= \mathbb{E}_{(\lambda,y)\sim p_\theta}\left[\lambda_i \lambda_j\right] \\ &= \mathbb{E}_{(\lambda,y)\sim p_\theta}\left[\mathbb{1}\left\{\lambda_i = \lambda_j\right\} - \mathbb{1}\left\{\lambda_i \neq \lambda_j\right\}\right] \\ &= p_\theta(\lambda_i = \lambda_j) - p_\theta(\lambda_i \neq \lambda_j) \\ &= \alpha_i \alpha_j + (1-\alpha_i)(1-\alpha_j) - \alpha_i(1-\alpha_j) - (1-\alpha_i)\alpha_j \\ &= (2\alpha_i - 1)(2\alpha_j - 1) \end{aligned}$$

where as in Section 2.3, we define $\alpha_j = p_\theta(\lambda_j = 1|y = 1)$. Letting μ be the vector such that $\mu_j = 2\alpha_j - 1$, we see that:

$$O = \mu\mu^T + \text{diag}(1 - \mu \circ \mu)$$

We see that even with the simple model we consider, this form does not admit the same simple spectral decomposition as in the spectral approach example of Section 2.3. However, we can simply approach this as a matrix completion or approximation-style optimization problem:

$$\hat{\mu} = \text{argmin}_{\mu} \left\|\hat{O} - \mu\mu^T\right\|_{i \neq j}$$

where we define $||A||_{i \neq j}$ as the Frobenius norm of matrix A with entries on the diagonal masked, and where we can directly obtain $\hat{\theta}$ from $\hat{\mu}$.

The challenge with the above formulation is that there is no obvious way to handle the more complex label models presented earlier–for instance any label models with correlations between the labeling functions. Next, we extend the general approach presented above to these more complex settings, and provide theoretical and empirical validation.

3.3.2 Learning the Label Model: Complete Form

In this subsection, we start by considering the Pairwise-Dependent Binary label model, and defining the inputs and syntax of this label model. We outline our approach for learning the parameters of this model using a matrix-completion style approach; establish an approach for checking the identifiability of a model; and finally, detail the algorithmic implementation of our approach. We then conclude by describing how this approach can handle the more general Pairwise-Dependent k-ary label model as well.

Model Definition

Let $x \in \mathcal{X}$ be a data point and $y \in \mathcal{Y}$ be the true label, where we consider the binary setting to start, $\mathcal{Y} = \{-1, 1\}$, and where (x, y) is drawn i.i.d. from a distribution $\mathcal{D}$. In our setting, rather than observing the true label y, we have access to m labeling functions which, when applied to x, output labels $\lambda_j \in \mathcal{Y} \cup \{\emptyset\}$, where as before $\emptyset$ denotes a special abstain value.

The user also provides the conditional dependency structure of the labeling functions as a graph $G_\lambda = (V, E)$, where $V = \{y, \lambda_1, \lambda_2, \ldots, \lambda_m\}$ (Figure 3.4). Specifically, if (λ_i, λ_j) is not an edge in G_λ, this means that λ_i is independent of λ_j conditioned on y and the other labeling function outputs. Note that if G_λ is unknown, it can be estimated using statistical techniques, covered in Section 3.4. Importantly, we do not know anything about the strengths of the correlations in G_λ, or the labeling functions' accuracies; these are captured

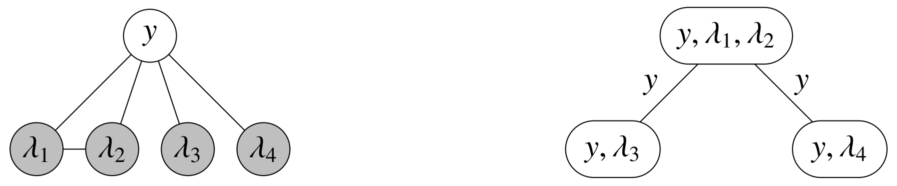


Figure 3.4: An example of a labeling function dependency graph G_λ (left) and its junction tree representation (right). Here, the output of labeling functions 1 and 2 are modeled as dependent conditioned on y. This results in a junction tree with singleton separator sets, y. Here, the observable cliques are $O = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4, \{\lambda_1, \lambda_2\}\} \subset \mathcal{C}$.

by the parameters of the label model we aim to estimate now.

Our overall goal is to apply the set of labeling functions λ to an unlabeled dataset X_U consisting of n data points, then use the resulting weakly-labeled training set to supervise the end discriminative model $h_w : \mathcal{X} \mapsto \mathcal{Y}$. This weakly-labeled training set will contain overlapping and conflicting labels, from labeling functions with unknown accuracies and correlations. To handle this, we will learn a label model $P_\theta(y|\lambda)$, parameterized by a vector of source correlations and accuracies θ, which for each data point x takes as input the noisy labels $\lambda = \{\lambda_1, \ldots, \lambda_m\}$ and outputs a single probabilistic label vector $\tilde{y}$. Succinctly, given a user-provided tuple $(X_U, \lambda, G_\lambda)$, our key technical challenge is recovering the parameters θ without access to ground truth labels y.

To learn the label model, we introduce sufficient statistics over the random variables in G_λ. Let $\mathcal{C}$ be the set of cliques in G_λ, and define an indicator random variable for the event of a clique $C \in \mathcal{C}$ of labeling functions taking on a set of values y_C:

$$\psi(C, y_C) = \mathbb{1}\left\{\cap_{i\in C} V_i = (y_C)_i\right\},$$

where $(y_C)_i \in \mathcal{Y} \cup \{\emptyset\}$. We define $\psi(C) \in \{0,1\}^{|\mathcal{Y}|^{|C|}}$ as the vector of indicator random variables for all combinations of all but one of the labels emitted by each variable in clique C—thereby defining a minimal set of statistics—and define $\psi(\mathbf{C})$ accordingly for any set of cliques $\mathbf{C} \subseteq \mathcal{C}$. Then $\theta = \mathbb{E}\left[\psi(\mathcal{C})\right]$ is the vector of sufficient statistics for the label model, which we want to learn.

Singleton Separator Sets Condition We now proceed with one simplifying condition: we consider the setting where G_λ is *triangulated* and has a junction tree representation with singleton separator sets. Intuitively, this corresponds to models where weak supervision labeling functions are correlated in fully-connected clusters, corresponding to real-world settings in which labeling functions are correlated due to shared data sources, code, or heuristics. Note, however, that we can always either (i) add edges to G_λ such that this is the case, or (ii) extend our approach to many settings where G_λ does not have singleton separator sets, as covered at the end of this section.

A Matrix Completion-Style Approach

The chief technical difficulty in our problem is that we do not observe y. We overcome this by analyzing the covariance matrix of an observable subset of the cliques in G_λ, leading to a matrix completion-style approach for recovering θ. We leverage two pieces of information:

1. The observability of *part of* $\mathbf{Cov}\left(\psi(C)\right)$, corresponding to the agreements and disagreements between labeling functions;

2. An extension of a result from [Loh and Wainwright, 2013] which states that the inverse covariance matrix $\mathbf{Cov}\left(\psi(C)\right)^{-1}$ is structured according to G_λ, i.e., if there is no edge between λ_i and λ_j in G_λ, then the corresponding entries are 0.

Since G_λ is triangulated, it admits a *junction tree* representation [Koller et al., 2009], which has maximal cliques (nodes) $\bar{C}$ and separator sets $\mathcal{S}$. Note that we follow the convention that $\mathcal{S}$ includes the full powerset of separator set cliques, i.e. all subset cliques of separator set cliques are also included in $\mathcal{S}$. Thus, under the singleton separator set condition outlined above, $\mathcal{S} = \{\{y\}\}$; note that in general we will write single-element sets without braces when their type is obvious from context, so we have $\mathcal{S} = \{y\}$.

We start by considering two disjoint subsets of $\bar{C}$: the set of observable cliques, $O \subseteq \bar{C}$—i.e., those cliques not containing y—and the separator set cliques of the junction tree, $\mathcal{S} \subseteq \bar{C}$ (which in our singleton separator set setting is $\{y\}$, but in general will always include

y and thus be unobservable)[3]. In this singleton separator set setting, we then have:

$$O = \{C \mid y \notin C, C \in \bar{\mathcal{C}}\} \qquad\qquad \mathcal{S} = \{y\}.$$

where $\psi(O)$ and $\psi(y)$ are the corresponding vectors of minimal indicator variables. We define corresponding dimensions d_O and $d_{\mathcal{S}}$, which in our binary setting are:

$$d_O = \sum_{C \in O} (|\mathcal{Y} \cup \{\emptyset\}| - 1)^{|C|} = \sum_{C \in O} 2^{|C|} \qquad\qquad d_{\mathcal{S}} = |\mathcal{Y}| - 1 = 1.$$

where note that we our sufficient statistics track all but one of the values that each variable can take on, so as to lead to a minimal set of sufficient statistics. We now decompose the generalized covariance matrix and its inverse as:

$$\mathbf{Cov}\left(\psi(O \cup \mathcal{S})\right) \equiv \Sigma = \begin{bmatrix} \Sigma_O & \Sigma_{OS} \\ \Sigma_{OS}^T & \Sigma_S \end{bmatrix} \qquad\qquad \Sigma^{-1} = K = \begin{bmatrix} K_O & K_{OS} \\ K_{OS}^T & K_S \end{bmatrix}, \tag{3.10}$$

This is similar to the form used in [Chandrasekaran et al., 2010], but with several important differences: we consider discrete (rather than Gaussian) random variables and have additional knowledge of the graph structure. Here, $\Sigma_O = \mathbf{Cov}\left(\psi(O)\right) \in \mathbb{R}^{d_O \times d_O}$ is the observable block of the generalized covariance matrix Σ, and $\Sigma_{OS} = \mathbf{Cov}\left(\psi(O), \psi(\mathcal{S})\right) \in \mathbb{R}^{d_O \times 1}$ is the unobserved vector which is a function of θ, the parameters (corresponding to labeling function and labeling function clique accuracies) we wish to recover. Finally, Σ_S is a scalar function of the class balance $P(y)$, which we assume is either known, or has been estimated according to the unsupervised approach we detail at the end of this section. Thus, Σ_O and Σ_S are known, and our goal is to recover the vector Σ_{OS}, from which we can recover θ. Applying the block matrix inversion lemma, we have:

$$K_O = \Sigma_O^{-1} + c\Sigma_O^{-1}\Sigma_{OS}\Sigma_{OS}^T\Sigma_O^{-1}, \tag{3.11}$$

[3]Note that from here on, we use O in this way, which is distinct from its usage in Section 3.3.1.

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 then express (3.11) as:

$$K_O = \Sigma_O^{-1} + zz^T \tag{3.12}$$

The right hand side of (3.12) consists of an empirically observable term, Σ_O^{-1}, and a rank-one term, zz^T, which we can solve for to directly recover θ. For the left hand side, we apply an extension of [Loh and Wainwright, 2013] (proof in Appendix C):

Corollary 1. *Let $U = O \cup \mathcal{S}$. Let Σ_U be the generalized covariance matrix for U. Then $(\Sigma_U^{-1})_{i,j} = 0$ whenever i, j correspond to cliques C_1, C_2 respectively such that C_1, C_2 are not subsets of the same maximal clique.*

We use this to conclude that K_O has graph-structured sparsity, i.e., it has zeros determined by the structure of dependencies between the labeling functions in G_λ. This suggests an algorithmic approach of estimating z as a matrix completion-style problem in order to recover an estimate of θ (Algorithm 1). In more detail: let Ω be the set of indices (i, j) where $(K_O)_{i,j} = 0$, determined by G_λ, yielding a system of equations,

$$0 = (\Sigma_O^{-1})_{i,j} + \left(zz^T\right)_{i,j} \text{ for } (i, j) \in \Omega, \tag{3.13}$$

which is now a matrix completion-style problem. Define $||A||_\Omega$ as the Frobenius norm of A with entries not in Ω set to zero; then we can rewrite (3.13) as $\left\|\Sigma_O^{-1} + zz^T\right\|_\Omega = 0$. We solve this equation to estimate z, and thereby recover Σ_{OS}, from which we can directly recover the label model parameters θ algebraically.

Checking for Identifiability

A first question is: which dependency structures G_λ lead to unique solutions for θ? This question presents a stumbling block for users, who might attempt to use non-identifiable sets of correlated weak supervision labeling functions. We provide a simple, testable condition for identifiability.

We start by defining the *inverse dependencies graph*, G_{inv}, such that G_{inv} contains an edge between two labeling functions λ_i, λ_j, whenever $(\lambda_i, \lambda_j) \notin G_\lambda$; in other words, G_{inv}

has an edge for each pair of labeling functions that we model as conditionally independent given y. Recall then that Ω is the augmented edge set of G_{inv}, in other words, a pair of indices (i, j)—corresponding to elements of $\psi(C)$, and therefore to cliques $A, B \in \mathcal{C}$—is in Ω if A, B are not part of the same maximal clique in G_λ (and therefore $(K_O)_{i,j} = 0$).

Then, given a solution z, by definition we have:

$$-(\Sigma_O^{-1})_\Omega = \left(zz^T\right)_\Omega, \tag{3.14}$$

This defines a set of $|\Omega|$ equations, which we can encode using a matrix M_Ω, where if (i, j) is the $(r-1)$th entry in Ω, then

$$(M_\Omega)_{r,s} = \begin{cases} 1 & s \in \{i, j\}, \\ 0 & \text{else.} \end{cases} \tag{3.15}$$

Let $l_i = \log(z_i^2)$ and $q_{(i,j)} = \log(((\Sigma_O^{-1})_{i,j}^2))$; then by squaring and taking the log of both sides of 3.14, we get a system of linear equations:

$$M_\Omega l = q_\Omega. \tag{3.16}$$

Thus, we can uniquely identify the z_i^2 if the system of linear equations (3.16) has a unique solution, which means that we can identify z (and therefore μ) *up to sign* in this case. And, note that we can always ensure that this system is uniquely solvable by adding labeling functions that are sufficiently independent.

Given estimates of the z_i^2, we can see that the sign of a single z_i determines the sign of all other z_j reachable from z_i in G_{inv}. Thus to ensure a unique solution, we only need to pick a sign for each connected component in G_{inv}. In the case where the labeling functions are assumed to be conditionally independent, e.g., [Dalvi et al., 2013; Zhang et al., 2016b; Dawid and Skene, 1979], it suffices to make the assumption that the labeling functions are *on average* non-adversarial; i.e., select the sign of the z_i that leads to higher average accuracies of the labeling functions. Even a single labeling function that is conditionally independent from all the other labeling functions will cause G_{inv} to be fully connected, meaning we can use this symmetry breaking assumption in the majority of cases even with correlated labeling functions. Otherwise, a sufficient condition is the standard one of assuming non-adversarial labeling functions, i.e. that all labeling functions have greater than random accuracy.

As one more intuitive example of a sufficient condition for a unique solution up to sign: note that if the inverse augmented edge graph consists of a connected triangle (or any odd-numbered cycle), e.g. $\Omega = \{(i, j), (j, k), (i, k)\}$, then we can solve for the z_i up to sign, and therefore M_Ω must be invertible:

$$z_i^2 = \frac{(\Sigma_O^{-1})_{i,j}(\Sigma_O^{-1})_{i,k}}{(\Sigma_O^{-1})_{j,k}},$$

and so on for z_j, z_k. Note additionally that if other z_i are connected to this triangle, then we can also solve for them up to sign as well. Therefore, if Ω contains at least one triangle (or odd-numbered cycle) per connected component, then M_Ω is invertible.

Also note that this is all in reference to the *inverse* dependency graph, which will generally be dense (assuming the correlation structure between labeling functions is generally sparse). For example, note that if we have one source λ_i that is conditionally independent of all the other labeling functions, then Ω is fully connected, and therefore if there is a triangle in Ω, then M_Ω is invertible.

Pairwise-Dependent Binary Label Model Parameter Estimation

Now that we know when a set of labeling functions with correlation structure G_λ is identifiable, yielding a unique z, we can estimate the accuracies θ using Algorithm 1, which consists of the following steps:

1. We begin by checking the identifiability of the problem defined by the model dependency structure G_λ (input as Ω), using the procedure detailed above.

2. Next, we estimate the class balance, $P(y)$, for $y \in \mathcal{Y}$ (or equivalently, $\hat{\mathbb{E}}[\psi(y)]$) using the ClassBalance routine. In many practical settings, $P(y)$ can be estimated from a small labeled sample, or may be known in advance. However here we consider using a subset of the labeling functions that are conditionally independent according to G_λ, $\lambda_{i_1}, \ldots, \lambda_{i_k}$, to estimate $P(y)$. We note first of all that simply taking the majority vote

Algorithm 1 Pairwise-Dependent Binary Label Model Parameter Estimation

```
Input: Observed labels $\hat{\mathbb{E}}[\psi(O)]$, covariance $\hat{\Sigma}_O$, and correlation sparsity structure $\Omega$

CheckIdentifiability($\Omega$)                                    ▷ Preliminary operations
$\hat{\mathbb{E}}[\psi(y)] \leftarrow$ ClassBalance($\hat{\mathbb{E}}[\psi(O)], \hat{\Sigma}_O, \Omega$)

$\hat{z} \leftarrow \operatorname{argmin}_z \left\|\hat{\Sigma}_O^{-1} + zz^T\right\|_\Omega$          ▷ Solve the masked matrix completion problem

$\hat{c} \leftarrow \Sigma_S^{-1}(1 + \hat{z}^T\hat{\Sigma}_O\hat{z})$             ▷ Recover the estimated label model parameters, $\hat{\theta}$
$\hat{\Sigma}_{OS} \leftarrow \hat{\Sigma}_O\hat{z}/\sqrt{\hat{c}}$
$\hat{\theta} \leftarrow$ Concat($\hat{\Sigma}_{OS} + \hat{\mathbb{E}}[\psi(y)]\hat{\mathbb{E}}[\psi(O)], \hat{\Sigma}_O + \hat{\mathbb{E}}[\psi(O)]\hat{\mathbb{E}}[\psi(O)]^T, \hat{\mathbb{E}}[\psi(y)]$)

return $\hat{\theta}$
```

of these labeling functions is a biased estimator. Instead, we consider a simplified version of the matrix completion-based approach taken so far. Denote the vector of the unary indicator statistics over the conditionally independent subset of labeling functions as ψ, and let the observed overlaps matrix between labeling functions i and j be $A_{i,j} = \mathbb{E}\left[\psi_i\psi_j^T\right]$. Note that due to the conditional independence of λ_i and λ_j, for any k, l we have:

$$\begin{aligned}
(A_{i,j})_{k,l} &= \mathbb{E}\left[(\psi_i)_k(\psi_j)_l\right] \\
&= P(\lambda_i = y_k, \lambda_j = y_l) \\
&= \sum_{y\in\mathcal{Y}} P(\lambda_i = y_k, \lambda_j = y_l | y = y)P(y = y) \\
&= \sum_{y\in\mathcal{Y}} P(\lambda_i = y_k | y = y)P(\lambda_j = y_l | y = y)P(y = y).
\end{aligned}$$

Letting B_i be the $|\mathcal{Y} \cup \{\emptyset\}| \times |\mathcal{Y}|$ matrix of conditional probabilities, $(B_i)_{j,k} = P(\lambda_i = y_j | y = y_k)$, and P be the diagonal matrix such that $P_{i,i} = P(y = y_i)$, we can re-express the above as:

$$A_{i,j} = B_i P B_j^T.$$

Since P is composed of strictly positive elements, and is diagonal (and thus PSD), we re-express this as:

$$A_{i,j} = \tilde{B}_i \tilde{B}_j^T, \tag{3.17}$$

where $\tilde{B}_i = B_i \sqrt{P}$. We could now try to recover P by decomposing the observed $A_{i,j}$ to recover the $\tilde{B}_i$, and from there recover P via the relation:

$$P = \operatorname{diag}\left(\tilde{B}_i^T \vec{1}\right)^2, \tag{3.18}$$

since summing the column of $\tilde{B}_i$ corresponding to label y is equal to:

$$\sqrt{P(y)} \sum_{y \in \mathcal{Y} \cup \{\emptyset\}} P(\lambda_i = y | y) = \sqrt{P(y)}$$

by the law of total probability. However, note that $\tilde{B}_i U$ for any orthogonal matrix U also satisfies (3.17), and could thus lead to a potentially infinite number of incorrect estimates of P.

Instead, we consider the three-way overlaps tensor observed as $A_{i,j,k}$, and perform a tensor decomposition. Note that above, the problem is that matrix decomposition is typically invariant to rotations and reflections; tensor decompositions have easier-to-meet uniqueness conditions (and are thus more rigid). To see this, we can apply Kruskal's classical identifiability condition for unique 3-tensor decomposition. Consider some tensor

$$T = \sum_{r=1}^{R} X_r \otimes Y_r \otimes Z_r,$$

where X_r, Y_r, Z_r are column vectors that make up the matrices X, Y, Z. The Kruskal rank k_X of X is the largest k such that any k columns of X are linearly independent. Then, the decomposition above is unique if $k_X + k_Y + k_Z \geq 2R + 2$ [Kruskal, 1977; Bhaskara et al., 2014]. In our case, our triple views have $R = |\mathcal{Y}|$, and we have

$$A_{i,j,k} = \tilde{B}_i \otimes \tilde{B}_j \otimes \tilde{B}_k. \tag{3.19}$$

Thus, if $k_{\tilde{B}_i} + k_{\tilde{B}_j} + k_{\tilde{B}_k} \geq 2|\mathcal{Y}| + 2$, we have identifiability. Thus, it is sufficient to have the columns of each of the $\tilde{B}_i$'s be linearly independent. Note that each of the B_i's have columns with the same sum, so these columns are only linearly dependent if they are equal, which would only be the case if the labeling functions were random voters. Thus, we can use (3.19) to recover the $\tilde{B}_i$ in a stable fashion, and then use (3.18) to recover the $P(y)$.

3. Next, we solve the core matrix completion-style problem; we note that in the binary setting we consider, this is a rank one problem:

$$\hat{z} = \operatorname{argmin}_z \left\|\Sigma_O^{-1} + zz^T\right\|_\Omega . \tag{3.20}$$

   This is similar to a standard matrix completion problem, except that (a) while the parameter matrix zz^T is low-rank (rank one in this case), Σ_O^{-1} is full-rank, not low rank; and (b) rather than observing randomly-sampled entries, Ω is a fixed mask.

   Regardless, we can solve this objective using standard approaches, such as stochastic gradient descent. Also note that we could alternatively solve the system of linear equations defined in the previous section, Equation 3.16; however the above objective allows easy incorporation of e.g. regularization amongst other advantages.

4. Once we have recovered z uniquely, we next need to recover $\Sigma_{OS} = c^{-\frac{1}{2}}\Sigma_O z$. We use the fact that $c = \Sigma_S^{-1}(1 + z^T\Sigma_O z)$, which we can confirm explicitly below, starting from the definition of c:

$$\begin{aligned}
c &= \left(\Sigma_S - \Sigma_{OS}^T\Sigma_O^{-1}\Sigma_{OS}\right)^{-1} \\
&= \left(\Sigma_S - (c^{-\frac{1}{2}}\Sigma_O z)^T\Sigma_O^{-1}(c^{-\frac{1}{2}}\Sigma_O z)\right)^{-1} \\
&= \left(\Sigma_S - c^{-1}z^T\Sigma_O z\right)^{-1} \\
\implies c^{-1} &= \Sigma_S - c^{-1}z^T\Sigma_O z \\
\implies c^{-1}\left(1 + z^T\Sigma_O z\right) &= \Sigma_S \\
\implies c &= \Sigma_S^{-1}\left(1 + z^T\Sigma_O z\right)
\end{aligned}$$

   Thus, we can directly recover an estimate of Σ_{OS} from the observed Σ_O, known Σ_S, and estimated z.

5. Finally, recall that $\theta = \mathbb{E}\left[\psi(\bar{C})\right]$, where $\bar{C}$ can be split into cliques not containing y, i.e. the set O previously defined; the cliques consisting of those cliques in O plus y; and y. We already can compute the expected value of the sufficient statistics for the first and third; for the second one, given our estimate of Σ_{OS}, we have:

$$\mathbb{E}\left[\psi(O)\psi(\mathcal{S})^T\right] = \mathbb{E}\left[\psi(O)\psi(y)^T\right] = \Sigma_{OS} + \mathbb{E}\left[\psi(O)\right]\mathbb{E}\left[\psi(y)\right]^T. \tag{3.21}$$

Here, we can clearly observe $\mathbb{E}\left[\psi(O)\right]$, and given that we know the class balance $P(y)$, we also have $\mathbb{E}\left[\psi(y)\right]$; therefore we can compute $\mathbb{E}\left[\psi(O)\psi(\mathcal{S})^T\right]$. Therefore we can recover the full θ by concatenating these vectors.

Predicting Labels with the Label Model

Once we have an estimate of θ, we can make predictions with the label model—i.e. generate our *probabilistic* training labels $P_\theta(y|\lambda)$—using the junction tree we have already defined over G_λ. Specifically, let $\bar{C}$ be the set of maximal cliques (nodes) in the junction tree, and let $\mathcal{S}$ be the set of separator sets. Then we have:

$$p_\theta(y,\lambda) = \frac{\prod_{C\in\bar{C}} P(V_C)}{\prod_{S\in\mathcal{S}} P(V_S)} = \frac{\prod_{C\in\bar{C}} \theta_{(C,(y,\lambda_C))}}{\prod_{S\in\mathcal{S}} \theta_{(S,(y,\lambda_S))}},$$

where $V_C = \{V_i\}_{i\in C}$, where $V_0 = y$ and $V_{i>0} = \lambda_i$. Thus, we can directly compute the predicted labels $P_\theta(y|\lambda)$ based on the estimated parameters θ.

Learning the Pairwise-Dependent k-ary Label Model

We can easily extend the approach outlined above to handle the k-ary setting where $r = |\mathcal{Y}| > 2$, and z is now a matrix $Z \in \mathbb{R}^{d_O\times d_S}$, leading to a rank-$(r-1)$ matrix completion-style problem (see [Ratner et al., 2019b] for details). However, we run into the difficulty that we can now only recover Z up to orthogonal transformations. We can handle this difficulty in one of two ways.

A first approach is to learn a simplified *class-conditional* model of the noisy labeling process, where we learn one accuracy parameter for each label value λ_i that each labeling function emits. This is equivalent to assuming that a source may have a different accuracy on each different class, but that if it emits a certain label incorrectly, it does so uniformly over the different true labels y. This is simpler than our general Pairwise-Dependent k-ary label model, but is still a far more expressive model than the commonly considered one, where each source is modeled by a single accuracy parameter, e.g. in [Dawid and Skene, 1979; Ratner et al., 2016]. We can see that our model estimation problem in this setting is mappable to a binarized version of the labels, $y_B = \mathbb{1}\{y = y\}$ for some y, as given the above assumption, we can recover the parameters for the normal version algebraically from the binarized one. Thus, this lets us solve a rank-one problem again, as above; for further details see [Ratner et al., 2019b].

A second approach is to perform a two-step procedure, where first we learn a model over a subset of the labeling functions that are pairwise conditionally independent as in our procedure for learning the class balance, which also returns the labeling function accuracies in a way that is stable (i.e. returns a unique solution) even in the general k-ary model; and then, we can use this solution to break symmetries (e.g. as a constraint) in our full estimation algorithm as above. For further details see [Ratner et al., 2019b].

Handling Non-Singleton Separator Sets

Finally, we briefly consider the setting where G_λ has arbitrary separator sets $\mathcal{S}$. Let $d_S = \sum_{S \in \mathcal{S}}(|\mathcal{Y}| - 1)^{|S|}$. We see that we could solve this using the approach outlined thus far, but for two changes: first, that it would now involve solving a rank-d_S matrix completion-style problem; and second, that we do not know $\Sigma_\mathcal{S}$, as it now involves terms besides the class balance.

Note first of all that we can always add edges between labeling functions to G_λ such that it has singleton separator sets (intuitively, this consists of “completing the clusters”), and as long as our problem is still identifiable, we can simply solve this instance as above.

Instead, we can also take a multi-step approach, wherein we first consider one or more subgraphs of G_λ that contain only singleton separator sets, and contain the cliques in $\mathcal{S}$. We can then solve this problem as before, which then gives us the needed information to identify the elements of $\Sigma_\mathcal{S}$ in our full problem, which we can then solve. In particular, we see that this multi-step approach is possible whenever the graph G_λ has at least three components that are disconnected except for through y.

3.3.3 Theoretical Analysis

We now return to considering the Pairwise-Dependent Binary label model, to theoretically analyze its convergence properties, and that of the end discriminative model trained with its labels, with respect to the set of labeling functions and number unlabeled data points they are applied to.

Our ultimate goal is to train an *end model* using the labeling function labels, denoised and combined by the label model $\hat{\theta}$ we have estimated. We connect the generalization error of this end model to the estimation error of Algorithm 1, ultimately showing that the generalization error scales as $n^{-\frac{1}{2}}$, where n is the number of unlabeled data points. This key result establishes the same asymptotic scaling as traditionally supervised learning methods, but with respect to *unlabeled* data points.

Let $p_{\hat{\theta}}(\tilde{y} \mid \lambda)$ be the probabilistic label (i.e. distribution) predicted by our label model, given the labeling function labels λ as input, which we compute using the estimated $\hat{\theta}$. We then train an *end* multi-task discriminative model $h_w : \mathcal{X} \mapsto \mathcal{Y}$ parameterized by w, by minimizing the expected loss with respect to the label model over n unlabeled data points. Let $l(h_w(x), y)$ be a bounded loss function such that without loss of generality $l(h_w(x), y) \leq 1$; then we minimize the empirical *noise aware loss*:

$$\hat{w} = \text{argmin}_w \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\tilde{y} \sim p_{\hat{\theta}}(\cdot|\lambda)} \left[l(w, x^{(i)}, \tilde{y}) \right], \tag{3.22}$$

and let $\tilde{w}$ be the w that minimizes the true noise-aware loss. This minimization can be performed by standard methods and is not the focus of this section; let the solution $\hat{w}$ satisfy $\mathbb{E}\left[||\hat{w} - \tilde{w}||^2\right] \leq \gamma$. We make several assumptions, following those in Section 3.2.2: (1) that for some label model parameters θ^*, sampling $(\lambda, y) \sim p_{\theta^*}$ is the same as sampling from the true distribution, $(\lambda, y) \sim \mathcal{D}$; and (2) that the labels y are independent of the features of the end model given λ sampled from p_{θ^*}, that is, the output of the optimal label model provides sufficient information to discern the true label. Then we have the following result:

Theorem 3. Let $\tilde{w}$ minimize the expected noise aware loss, using labeling function parameters $\hat{w}$ estimated with Algorithm 1. Let $\hat{w}$ minimize the empirical noise aware loss with $\mathbb{E}\left[||\hat{w} - \tilde{w}||^2\right] \leq \gamma$, $w^* = \min_w l(w, x, y)$, and let the assumptions above hold. Then the generalization error is bounded by:

$$\mathbb{E}\left[l(\hat{w},x,y) - l(w^*,x,y)\right] \leq \gamma + 4|\mathcal{Y}|\left\|\hat{\theta}-\theta^*\right\|.$$

Thus, to control the generalization error, we must control $\left\|\hat{\theta}-\theta^*\right\|$, which we do in Theorem 4:

Theorem 4. Let $\hat{\theta}$ be an estimate of θ^* produced by Algorithm 1 run over n unlabeled data points. Let $a := \left(\frac{d_O}{\Sigma_S} + \left(\frac{d_O}{\Sigma_S}\right)^2 \lambda_{\max}(K_O)\right)^{\frac{1}{2}}$ and $b := \frac{\|\Sigma_O^{-1}\|^2}{(\Sigma_O^{-1})_{\min}}$. Then, we have:

$$\mathbb{E}\left[\left\|\hat{\theta}-\theta^*\right\|\right] \leq 16(|\mathcal{Y}|-1)d_O^2\sqrt{\frac{32\pi}{n}}ab\sigma_{\max}(M_\Omega^+)\left(3\sqrt{d_O}a\lambda_{\min}^{-1}(\Sigma_O)+1\right)\left(\kappa(\Sigma_O)+\lambda_{\min}^{-1}(\Sigma_O)\right).$$

Interpreting the Bound We briefly explain the key terms controlling the bound in Theorem 4; more detail is found in Appendix C. Our primary result is that the estimation error scales as $n^{-\frac{1}{2}}$. Next, $\sigma_{\max}(M_\Omega^+)$, the largest singular value of the pseudoinverse M_Ω^+, has a deep connection to the density of the graph G_{inv}. The smaller this quantity, the more information we have about G_{inv}, and the easier it is to estimate the accuracies. Next, $\lambda_{\min}(\Sigma_O)$, the smallest eigenvalue of the observed covariance matrix, reflects the conditioning of Σ_O; better conditioning yields easier estimation, and is roughly determined by how far away from random guessing the worst labeling function is, as well as how conditionally independent the labeling functions are. $\lambda_{\max}(K_O)$, the largest eigenvalue of the upper-left block of the inverse covariance matrix, similarly reflects the overall conditioning of Σ. Finally, $(\Sigma_O^{-1})_{\min}$, the smallest entry of the inverse observed matrix, reflects the smallest non-zero correlation between labeling function accuracies; distinguishing between small correlations and independent labeling functions requires more samples.

3.3.4 Experiments

In Figure 3.5, we plot the performance of our algorithm on synthetic data, showing its scaling with the number of unlabeled data points n, the density of pairwise dependencies in G_λ, and the runtime performance as compared to the approach in Section 3.2 using Gibbs

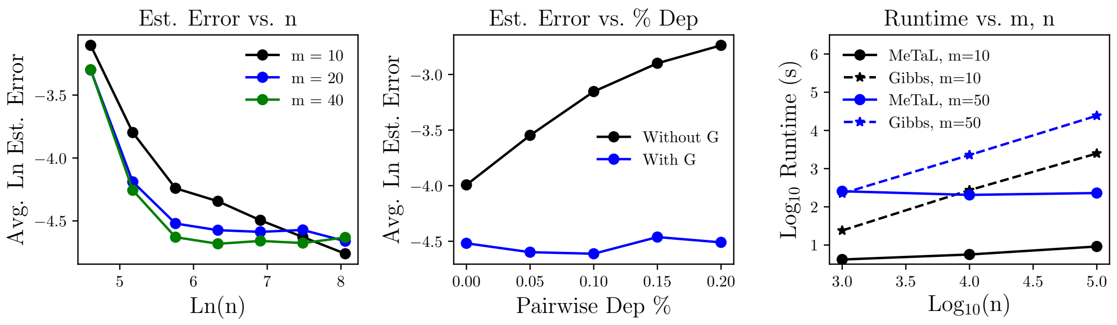


Figure 3.5: (Left) Estimation error $\left\|\hat{\theta} - \theta^*\right\|$ decreases with increasing n. (Middle) Given G_λ, our model successfully recovers the labeling function accuracies even with many pairwise dependencies among labeling functions, where a naive conditionally-independent model fails. (Right) The runtime of the approach is independent of n after an initial matrix multiply, and can thus be multiple orders of magnitude faster than Gibbs sampling-based approaches, as in Section 3.2.

sampling and SGD. We report further experiments validating this matrix-completion style approach as applied to the multi-task weak supervision problem covered in Section 5.

3.4 Structure Learning for Weak Supervision

In Sections 3.1 through 3.3, we assume that we are given a set of conditional dependencies between the labeling functions—represented in Section 3.3 as a graph of pairwise correlation edges G_λ—which defines the structure of the label model we aim to learn.

In some cases the user may be able to define this structure manually with minimal inconvenience, e.g. in settings where there are a small number of labeling functions, and a small number of more obvious dependencies to model. For example, a user might write two distant supervision-style labeling functions that use the same knowledge base, and thus the user may think modeling a dependency edge here is prudent, and can specify this.

However, in most settings, we would like to automate this process of specifying dependency edges. This can be viewed as a variant of a classic model *structure learning* problem, where we have a latent variable model due to the unobserved ground truth, and other unique aspects of our weak supervision setting to leverage. We now briefly summarize three approaches for structure learning in the data programming setting: two based on statistical approaches, and one based on analyzing the code content of the user-authored labeling functions.

3.4.1 Statistical Approaches

l_1-Regularized Marginal Pseudolikelihood Approach

A first approach is based on extending a classic structure learning technique to the data programming setting, where we do not observe the ground truth labels y. In this approach, we optimize the log marginal pseudolikelihood of the outputs of a single labeling function λ_i, i.e., conditioned on the outputs of the others $\lambda_{\setminus i}$, using ℓ_1 regularization to induce sparsity. The objective is

$$\begin{aligned} & \operatorname{argmin}_\theta \quad -\log p_\theta(\lambda_j \mid \lambda_{\setminus j}) + \epsilon \|\theta\|_1 \\ = & \operatorname{argmin}_\theta \quad -\sum_{i=1}^{n} \log \sum_{y^{(i)}} p_\theta(\lambda_j^{(i)}, y^{(i)} \mid \lambda_{\setminus j}^{(i)}) + \epsilon \|\theta\|_1, \end{aligned} \tag{3.23}$$

where $\epsilon > 0$ is a hyperparameter. By conditioning on all other labeling functions in each term $\log \sum_{y_i} p_\theta(\lambda_j^{(i)}, y^{(i)} \mid \lambda_{\setminus j}^{(i)})$, we ensure that the gradient can be computed in polynomial time with respect to the number of labeling functions, data points, and possible dependencies; without requiring any sampling or variational approximations. We optimize for each labeling function λ_j in turn, selecting those dependencies with parameters that have a sufficiently large magnitude and adding them to the estimated structure. Note that ϵ effectively controls a tradeoff space between model fidelity (density of edges) and computational complexity. We empirically and theoretically validate this basic procedure in [Bach et al., 2017], and explore the resulting tradeoff space controlled by ϵ in [Ratner et al., 2017a].

Robust PCA-Based Approach

We can also extend *robust PCA* [Candès et al., 2011; Chandrasekaran et al., 2011] to the data programming setting—and specifically the matrix completion-style formulation in Section 3.3—and use it to learn the structure of the label model. The robust PCA setup consists of a matrix $M \in \mathbb{R}^{m \times m}$ that is equal to the sum of a low-rank matrix and a sparse

```
**Algorithm 2** Weak Supervision Structure Learning with Robust PCA
  Input: Estimate of the covariance matrix $\hat{\Sigma}_O$, parameters $\lambda_n, \gamma$, threshold $T$, loss function $\mathcal{L}(\cdot, \cdot)$
  Solve:
      $(\hat{S}, \hat{L}) = \text{argmin}_{(S,L)} \mathcal{L}(S - L, \Sigma_O^{(n)}) + \lambda_n(\gamma \|S\|_1 + \|L\|_*)$
  s.t. $S - L \succ 0, L \succeq 0$
  $\hat{E} \leftarrow \{(i, j) : i < j, \hat{S}_{ij} > T\}$

  Return: $\hat{G} = (V, \hat{E})$
```

matrix, $M = L + S$, where $\text{rank}(L) = r$ and $|\text{supp}(S)| = k$. In our setting, we can let $L = zz^T$ be the rank-one (or low-rank) parameters we aim to estimate, and let $S = K_O$ be the observed block of the inverse covariance matrix, as in Section 3.3, which we assume is graph structured and sparse. Thus, recovering the sparsity pattern of K_O is our structure learning objective.

Algorithm 2 describes our latent structure learning method. We use the loss function from [Wu et al., 2017]:

$$\mathcal{L}(S - L, \Sigma_O^{(n)}) = \frac{1}{2}\text{tr}((S - L)\Sigma_(^{(n)} S - L)) - \text{tr}(S - L),$$

and implement Algorithm 2 using standard convex solvers. The recovered sparse matrix $\hat{S}$ does not have entries that are perfectly 0. Therefore, a key choice is to set a threshold T to find the zeros in $\hat{S}$ such that

$$\tilde{S}_{ij} = \begin{cases} \hat{S}_{ij} & \text{if } \hat{S}_{ij} > T, \\ 0 & \text{if } \hat{S}_{ij} \leq T. \end{cases}$$

The nonzero entries of $\tilde{S}$ then define the structure G_λ. In [Varma et al., 2019], we then analyze the theoretical convergence of Algorithm 2 under two different conditions motivated by the data programming setting, and empirically validate its performance on several datasets.

3.4.2 Using Static Analysis

In the statistical structure learning approaches of Section 3.4.2, we proceed assuming we only have access to the observed outputs of the m labeling functions. However, in many settings, these labeling functions are not black boxes, but rather consist of user-authored code that we have access to. This unique setting raises the possibility of "opening up the black boxes" of the labeling functions and applying static analysis techniques to suggest the dependency structure of our label model. For example, if two labeling functions λ_i, λ_j use the same knowledge base, data resource, model, or heuristic pattern—all easily detectable via simple static analysis of the labeling function code—then we might find it reasonable to add the pairwise correlation edge (λ_i, λ_j) to G_λ. Another example is [Varma et al., 2017]: here the authors consider the setting of data programming applied to image classification tasks, where the labeling functions are written over pre-processed features or *primitives*. In this work, whenever labeling functions utilize the same primitive, a corresponding edge is added to the dependency graph; the authors demonstrate that this leads to empirical gains over both an empty dependency graph (e.g. the conditionally-independent label model) and the structure learned via a statistical approach as in Section 3.4.2. In practice, a combination of (i) user-provided dependencies, (ii) statistically learned dependencies, and (iii) dependencies detected via static analysis of labeling function code (when available) can be used.

3.5 Related Work

Data Programming The overall data programming concept and approach presented in this section builds on many previous approaches in machine learning.

Distant supervision is one preceding paradigm for programmatically creating training sets. The canonical example is relation extraction from text, wherein a knowledge base of known relations is heuristically mapped to label a set of mentions in an input corpus as ground truth examples [Craven et al., 1999; Mintz et al., 2009; Zhang et al., 2017a]. Basic extensions group these mapped examples by the particular textual pattern w that they occur with, and cast the problem as a *multiple instance learning* one [Riedel et al., 2010; Hoffmann et al., 2011]. Other extensions actually model the accuracy of this pattern w using a discriminative feature-based model [Roth and Klakow, 2013a], or generative models such as hierarchical topic models [Alfonseca et al., 2012; Roth and Klakow, 2013b; Takamatsu et al., 2012]. Like our approach, these latter methods model a generative process of training set creation, however in a proscribed way that is not based on user input as in our approach. There is also a wealth of examples where additional heuristic patterns used to label training data are collected from unlabeled data [Bunescu and Mooney, 2007] or directly from users [Shin et al., 2015; Mallory et al., 2015], in a similar manner to our approach, but without a framework to deal with the fact that said labels are explicitly noisy. Other related approaches include pattern-based supervision [Gupta and Manning, 2014; Zhang et al., 2017a] and feature-annotation techniques [Mann and McCallum, 2010; Zaidan and Eisner, 2008; Liang et al., 2009].

Crowdsourcing is widely used for various machine learning tasks [Krishna et al., 2016; Gao et al., 2011]. Of particular relevance to our problem setting is the theoretical question of how to model the accuracy of various experts without ground truth available, classically raised in the context of crowdsourcing [Dawid and Skene, 1979]. More recent results provide formal guarantees even in the absence of labeled data using various approaches [Karger et al., 2011; Parisi et al., 2014; Berend and Kontorovich, 2014; Zhang et al., 2016b; Dalvi et al., 2013; Joglekar et al., 2015]. Our model can capture the model described in crowdsourcing, and can be equivalent in the conditionally-independent case we consider in Section 3.2. However, in addition to generalizing beyond getting inputs solely from human annotators, we also model user-supplied dependencies between the "labelers" in our model, which is not natural within the context of crowdsourcing. Additionally, while crowdsourcing results focus on the regime of a large number of labelers each labeling a small subset of the data, we consider a small set of labeling functions each labeling a large portion of the dataset.

Co-training is a classic procedure for effectively utilizing both a small amount of labeled data and a large amount of unlabeled data by selecting two conditionally independent *views* of the data [Blum and Mitchell, 1998]. In addition to not needing a set of labeled data, and allowing for more than two views (labeling functions in our case), our approach allows explicit modeling of dependencies between views, for example allowing observed issues with dependencies between views to be explicitly modeled [Krogel and Scheffer, 2004].

Boosting is a well known procedure for combining the output of many "weak" classifiers to create a strong classifier in a supervised setting [Schapire and Freund, 2012]. Recently, boosting-like methods have been proposed which leverage unlabeled data in addition to labeled data, which is also used to set constraints on the accuracies of the individual classifiers being ensembled [Balsubramani and Freund, 2015]. This is similar in spirit to our approach, except that labeled data is not explicitly necessary in ours, and richer dependency structures between our "heuristic" classifiers (labeling functions) are supported.

The general case of *learning with noisy labels* is treated both in classical [Lugosi, 1992] and more recent contexts [Natarajan et al., 2013]. It has also been studied specifically in the context of *label-noise robust* logistic regression [Bootkrajang and Kabán, 2012]. We consider the more general scenario where multiple noisy labeling functions can conflict and have dependencies.

Matrix Completion-Style Approaches The matrix completion-style approach presented in Section 3.3 has connections to the crowdsourcing literature [Karger et al., 2011; Dawid and Skene, 1979], and in particular to spectral and method of moments-based approaches [Zhang et al., 2016b; Dalvi et al., 2013; Ghosh et al., 2011; Anandkumar et al., 2014]. It is also related to recent techniques for estimating classifier accuracies without labeled data in the presence of structural constraints [Platanios et al., 2017], and uses matrix structure estimation [Loh and Wainwright, 2013] and concentration bounds [Tropp, 2015] for the core results.

Structure Learning Structure learning is a well-studied problem, but most work has assumed access to hand-labeled training data. Traditional lines of work focus on the lasso technique [Tibshirani, 1996; Zhao and Yu, 2006] and other approaches for linear models [Candes and Tao, 2007; Ng, 2004].

Regularized estimators, such as the L1 pseudolikelihood approach briefly summarized in Section 3.4, have also been used to select structures for graphical models, such as e.g. [Meinshausen and Bühlmann, 2006]. Most similar to our proposed L1 pseudo likelihood estimator, [Ravikumar et al., 2010] propose a fully supervised pseudolikelihood estimator for Ising models. Other related work includes [Chandrasekaran et al., 2012], which considers learning the structure of Gaussian graphical models with latent variables, grafting [Perkins et al., 2003; Zhu et al., 2010] and the information bottleneck approach for learning Bayesian networks with latent variables [Elidan and Friedman, 2005].

The supervised, fully observed setting includes matrix-wise methods more similar to the robust PCA-based approach briefly summarized in Section 3.4, which use the inverse covariance matrix to determine the structure [Friedman et al., 2008; Ravikumar et al., 2011; Loh and Wainwright, 2013]. In the latent variable setting, works performing structure learning via robust-PCA like approaches include [Chandrasekaran et al., 2010; Meng et al., 2014; Wu et al., 2017]. For further details on related work see [Bach et al., 2017; Varma et al., 2019].

Chapter 4

Snorkel: A System for Weak Supervision

In Chapter 3 we introduced *data programming*, a new paradigm for programmatic labeling of training datasets, and focused on the core model, algorithm, and theory components of this approach. However, the core motivation of this thesis work is to use approaches like data programming to make modern machine learning tools more efficient and accessible for real users.

In this chapter, we present Snorkel, a system for programmatically building and managing training datasets built around the core paradigm of data programming. In Snorkel, we take the core ideas and workflows proposed in data programming and add user interfaces, end-to-end data management, and optimizers for new tradeoffs introduced, and combine the entire resulting system into an open source software package that has now been deployed in a wide range of real-world use cases across industry, medicine, science, and government[1].

We start by providing an overview of the system architecture of Snorkel, describing new weak supervision modeling tradeoffs that arise, and presenting an optimizer for managing them. We then review an extensive set of experiments, user studies, and real-world applications. In two collaborations, with the U.S. Department of Veterans Affairs and the U.S. Food and Drug Administration, and on four open-source text and image data sets representative of other deployments, we find that Snorkel provides 132% average improvements to predictive performance over prior heuristic approaches and comes within an average 3.60% of the predictive performance of large hand-curated training sets. In a user study, we find

[1]`https://snorkel.org`

that subject matter experts build models 2.8x faster and increase predictive performance an average 45.5% versus seven hours of hand labeling. Finally, we briefly summarize a few of the many real-world deployments of Snorkel, including applications in knowledge base construction over the scientific literature and electronic health records, medical imaging and monitoring over radiograph, EEG, and cardiac video data, and industrial deployments at companies like Google and Intel. We present these results to provide validation for Snorkel, data programming, and more broadly, the core thesis that programmatically labeling and managing training datasets can be a powerful way to build machine learning applications in the real world.

Motivation In the last several years, there has been an explosion of interest in machine-learning-based systems across industry, government, and academia, with an estimated spend this year of $12.5 billion [Minonne et al., 2017]. A central driver has been the advent of *deep learning* techniques, which can learn task-specific representations of input data, obviating what used to be the most time-consuming development task: feature engineering. These learned representations are particularly effective for tasks like natural language processing and image analysis, which have high-dimensional, high-variance input that is impossible to fully capture with simple rules or hand-engineered features [Graves and Schmidhuber, 2005; Deng et al., 2009]. However, deep learning has a major upfront cost: these methods need massive *training sets* of labeled examples to learn from—often tens of thousands to millions to reach peak predictive performance [Sun et al., 2017].

Such training sets are enormously expensive to create, especially when domain expertise is required. For example, reading scientific papers, analyzing intelligence data, and interpreting medical images all require labeling by trained *subject matter experts* (SMEs). Moreover, we observe from our engagements with collaborators like research labs and major technology companies (see Section 4.5) that modeling goals such as class definitions or granularity change as projects progress, necessitating re-labeling. Some big companies are able to absorb this cost, hiring large teams to label training data [Metz, 2016; Eadicicco, 2017; Davis et al., 2013]. Other practitioners utilize classic techniques like *active learning* [Settles, 2009], *transfer learning* [Pan and Yang, 2010], and *semi-supervised learning* [Chapelle et al., 2009] to reduce the number of training labels needed. However, the

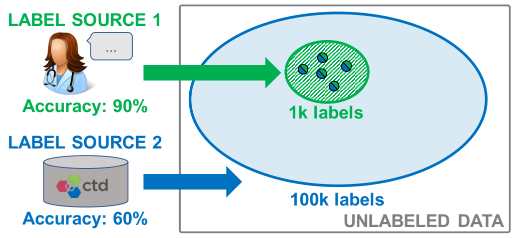


Figure 4.1: In Example 4.0.1, training data is labeled by sources of differing accuracy and coverage. Two key challenges arise in using this weak supervision effectively. First, we need a way to estimate the unknown source accuracies to resolve disagreements. Second, we need to pass on this critical lineage information to the end model being trained.

bulk of practitioners are increasingly turning to some form of *weak supervision*: cheaper sources of labels that are noisier or heuristic. The most popular form is *distant supervision*, in which the records of an external knowledge base are heuristically aligned with data points to produce noisy labels [Bunescu and Mooney, 2007; Mintz et al., 2009; Alfonseca et al., 2012]. Other forms include crowdsourced labels [Yuen et al., 2011; Quinn and Bederson, 2011], rules and heuristics for labeling data [Zhang et al., 2017a; Rekatsinas et al., 2017a], and others [Zaidan and Eisner, 2008; Liang et al., 2009; Mann and McCallum, 2010; Stewart and Ermon, 2017]. While these sources are inexpensive, they often have limited accuracy and coverage.

Ideally, we would combine the labels from many weak supervision sources to increase the accuracy and coverage of our training set. However, two key challenges arise in doing so effectively. First, sources will overlap and conflict, and to resolve their conflicts we need to estimate their accuracies and correlation structure, *without* access to ground truth. Second, we need to pass on critical lineage information about label quality to the end model being trained.

Example 4.0.1. In Figure 4.1, we obtain labels from a high accuracy, low coverage Source 1, and from a low accuracy, high coverage Source 2, which overlap and disagree (split-color points). If we take an unweighted majority vote to resolve conflicts, we end up with null (tie-vote) labels. If we could correctly estimate the source accuracies, we would resolve conflicts in the direction of Source 1.

We would still need to pass this information on to the end model being trained. Suppose that we took labels from Source 1 where available, and otherwise took labels from Source 2. Then, the expected training set accuracy would be 60.3%—only marginally better than the weaker source. Instead we should represent training label lineage in end model training, weighting labels generated by high-accuracy sources more.

In Chapter 3, we reviewed *data programming* as a paradigm for addressing both of these challenges by modeling multiple label sources without access to ground truth, and generating *probabilistic* training labels representing the lineage of the individual labels. We prove, in Sections 3.2.2 and 3.3.3, surprisingly, we can recover source accuracy and correlation structure without hand-labeled training data.

Snorkel In this chapter, we present *Snorkel*, the first end-to-end system for combining weak supervision sources to rapidly create training data. Snorkel was built as a prototype to study how people could use data programming, a fundamentally new approach to building machine learning applications. Through weekly hackathons and office hours held at Stanford University over the majority of the period covered by this dissertation, we have interacted with a growing user community around Snorkel's open source implementation.[2] We have observed SMEs in industry, science, and government deploying Snorkel for knowledge base construction, image analysis, bioinformatics, fraud detection, and more. From this experience, we have distilled three principles that have shaped Snorkel's design:

1. **Bring All Sources to Bear:** The system should enable users to opportunistically use labels from all available weak supervision sources.

2. **Training Data as the Interface to ML:** The system should model label sources to produce a single, probabilistic label for each data point and train any of a wide range of classifiers to generalize beyond those sources.

3. **Supervision as Interactive Programming:** The system should provide rapid results in response to user supervision. We envision weak supervision as the REPL-like interface for machine learning.

Our work on Snorkel in this chapter makes the following technical contributions:

[2]`http://snorkel.stanford.edu`

A Flexible Interface for Sources We observe that the heterogeneity of weak supervision strategies is a stumbling block for developers. Different types of weak supervision operate on different scopes of the input data. For example, distant supervision has to be mapped programmatically to specific spans of text. Crowd workers and weak classifiers often operate over entire documents or images. Heuristic rules are open ended; they can leverage information from multiple contexts simultaneously, such as combining information from a document's title, named entities in the text, and knowledge bases. This heterogeneity was cumbersome enough to completely block users of early versions of Snorkel.

To address this challenge, we built an interface layer around the abstract concept of a *labeling function (LF)* (Section 4.1). We developed a flexible language for expressing weak supervision strategies and supporting data structures. We observed accelerated user productivity with these tools, which we validated in a user study (Section 4.4) where SMEs build models 2.8× faster and increase predictive performance an average 45.5% versus seven hours of hand labeling.

Tradeoffs in Modeling of Sources Snorkel learns the accuracies of weak supervision sources without access to ground truth using a generative model (see Chapter 3). Furthermore, it also learns correlations and other statistical dependencies among sources, correcting for dependencies in labeling functions that skew the estimated accuracies (see Section 3.4). This paradigm gives rise to previously unexplored tradeoff spaces between predictive performance and speed. The natural first question is: when does modeling the accuracies of sources improve predictive performance? Further, how many dependencies, such as correlations, are worth modeling?

In Section 4.2, we describe the tradeoffs between predictive performance and training time in generative models for weak supervision. While modeling source accuracies and correlations will not hurt predictive performance, we present a theoretical analysis of when a simple majority vote will work just as well. Based on our conclusions, we introduce an optimizer for deciding when to model accuracies of labeling functions, and when learning can be skipped in favor of a simple majority vote. Further, our optimizer automatically decides which correlations to model among labeling functions. This optimizer correctly predicts the advantage of generative modeling over majority vote to within 2.16 accuracy points on average on our evaluation tasks, and accelerates pipeline executions by up to 1.8×. It also enables us to gain 60%–70% of the benefit of correlation learning while saving up to 61% of training time (34 minutes per execution).

First End-to-End System for Data Programming Snorkel is the first system to implement the data programming paradigm in Chapter 3. Previous ML systems [Zhang et al., 2017a] required extensive feature engineering and model specification, leading to confusion about where to inject relevant domain knowledge. While programming weak supervision seems superficially similar to feature engineering, we observe that users approach the two processes very differently. The vision of this dissertation—weak supervision as the sole port of interaction for machine learning—implies radically different workflows, requiring a proof of concept.

Snorkel demonstrates that this paradigm enables users to develop high-quality models for a wide range of tasks. In Section 4.3, we report on two deployments of Snorkel, in collaboration with the U.S. Department of Veterans Affairs and Stanford Hospital and Clinics, and the U.S. Food and Drug Administration, where Snorkel improves over heuristic baselines by an average 110%, and report results on four open-source datasets that are representative of other Snorkel deployments, including bioinformatics, medical image analysis, and crowdsourcing; on which Snorkel beats heuristics by an average 153% and comes within an average 3.60% of the predictive performance of large hand-curated training sets.

Outline of Chapter In this chapter we describe Snorkel, an end-to-end system for building machine learning applications using the data programming paradigm introduced in Chapter 3, and describe extensive experimental validation:

- *In Section 4.1,* we start by describing the architecture of Snorkel, built around the paradigm of data programming.
- *In Section 4.2,* we study new weak supervision tradeoffs introduced by data programming and Snorkel–namely, the time-performance tradeoffs around modeling the accuracies and correlation structures of the user-provided labeling functions–and introduce heuristic optimizers for managing these tradeoffs.

- *In Section 4.3,* we describe several experiments applying Snorkel to various benchmark and real-world applications, which serve to validate and ablate the contributions of different components and aspects of Snorkel.
- *In Section 4.4,* we report on one of several user studies conducted as part of this thesis work in order to validate the ease of use of Snorkel, especially insomuch as it makes modern machine learning tools more accessible to non-expert users.
- *In Section 4.5,* we report on several of the real-world applications of Snorkel, validating its broader utility.
- *Finally, in Section 4.6* we review related work.

In Chapter 5, we describe how Snorkel can be extended to the *multi-task* setting, and in Chapter 6 we introduce another form of programmatic weak supervision, *data augmentation*, both of which are now integrated into the Snorkel open source software package[3].

4.1 Snorkel Architecture

Snorkel's workflow is designed around data programming [Ratner et al., 2016; Bach et al., 2017], a fundamentally new paradigm for training machine learning models using weak supervision, and proceeds in three main stages (Figure 4.2):

1. **Writing Labeling Functions:** Rather than hand-labeling training data, users of Snorkel write labeling functions, which allow them to express various weak supervision sources such as patterns, heuristics, external knowledge bases, and more. This was the component most informed by early interactions (and mistakes) with users over initial deployments, and we present a flexible interface and supporting data model.
2. **Modeling Accuracies and Correlations:** Next, Snorkel automatically learns a *generative label model* (e.g. see Chapter 3) over the labeling functions, which allows it to

[3]`https://snorkel.org`; as of version 0.9.

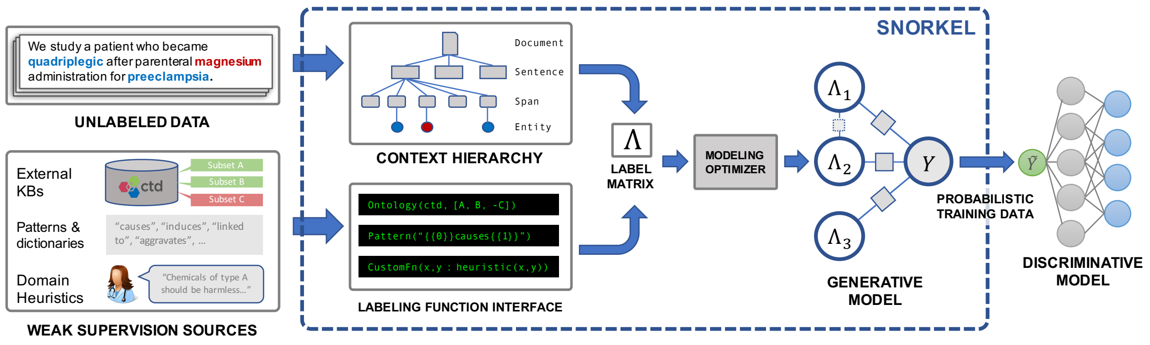

We study a patient who became quadriplegic after parenteral magnesium administration for preeclampsia.
UNLABELED DATA
External KBs
ctd
Subset A
Subset B
Subset C
Patterns & dictionaries
"causes", "induces", "linked to", "aggravates", ...
Domain Heuristics
"Chemicals of type A should be harmless..."
WEAK SUPERVISION SOURCES
SNORKEL
Document
Sentence
Span
Entity
CONTEXT HIERARCHY
Ontology(ctd, [A, B, -C])
Pattern("{{0}}causes{{1}}")
CustomFn(x,y : heuristic(x,y))
LABELING FUNCTION INTERFACE
Λ
LABEL MATRIX
MODELING OPTIMIZER
Λ1
Λ2
Λ3
Y
GENERATIVE MODEL
PROBABILISTIC TRAINING DATA
Ỹ
DISCRIMINATIVE MODEL

Figure 4.2: An overview of the Snorkel system. (1) SME users write *labeling functions (LFs)* that express weak supervision sources like distant supervision, patterns, and heuristics. (2) Snorkel applies the LFs over unlabeled data and learns a generative model to combine the LFs' outputs into probabilistic labels. (3) Snorkel uses these labels to train a discriminative classification model, such as a deep neural network.

estimate their accuracies and correlations. This step uses no ground-truth data, learning instead from the agreements and disagreements of the labeling functions. We observe that this step improves end predictive performance 5.81% over Snorkel with unweighted label combination, and anecdotally that it streamlines the user development experience by providing actionable feedback about labeling function quality.

3. **Training a Discriminative Model:** The output of Snorkel is a set of *probabilistic labels* that can be used to train a wide variety of state-of-the-art machine learning models, such as popular deep learning models. While the generative model is essentially a re-weighted combination of the user-provided labeling functions—which tend to be precise but low-coverage—modern discriminative models can retain this precision while learning to generalize beyond the labeling functions, increasing coverage and robustness on unseen data.

Next we set up the problem Snorkel addresses and describe its main components and design decisions.

Setup Our goal is to learn a parameterized classification model h_w that, given a data point $x \in \mathcal{X}$, predicts its label $y \in \mathcal{Y}$, where the set of possible labels $\mathcal{Y}$ is discrete. For simplicity, we focus on the binary setting $\mathcal{Y} = \{-1, 1\}$, though we include a multi-class application in our experiments. For example, x might be a medical image, and y a label indicating normal versus abnormal. In the relation extraction examples we look at, we often refer to x as a *candidate*. In a traditional supervised learning setup, we would learn h_w by fitting it to a *training set* of labeled data points. However, in our setting, we assume that we only have access to unlabeled data for training. We do assume access to a small set of labeled data used during development, called the *development set*, and a blind, held-out labeled *test set* for evaluation. These sets can be orders of magnitudes smaller than a training set, making them economical to obtain.

The user of Snorkel aims to generate training labels by providing a set of labeling functions, which are black-box functions, $\lambda_j : \mathcal{X} \to \mathcal{Y} \cup \{\emptyset\}$, that take in a data point and output a label where we again use $\emptyset$ to denote that the labeling function abstains. Given n unlabeled data points and m labeling functions, Snorkel applies the labeling functions over the unlabeled data to produce a matrix of labeling function outputs $\Lambda \in (\mathcal{Y} \cup \{\emptyset\})^{n \times m}$. The goal of the remaining Snorkel pipeline is to synthesize this label matrix Λ—which may contain overlapping and conflicting labels for each data point—into a single vector of *probabilistic training labels* $\tilde{y} = (\tilde{y}^{(1)}, ..., \tilde{y}^{(n)})$, where $\tilde{y}^{(i)} \in [0, 1]$. These training labels can then be used to train a discriminative model.

Next, we introduce the running example of a text relation extraction task as a proxy for many real-world knowledge base construction and data analysis tasks:

Example 4.1.1. Consider the task of extracting mentions of adverse chemical-disease relations from the biomedical literature (see CDR task, Section 4.3). Given documents with mentions of chemicals and diseases tagged, we refer to each co-occurring (chemical, disease) mention pair as a *candidate* extraction, which we view as a data point to be classified as either true or false. For example, in Figure 4.2, we would have two candidates with true labels y_1 = `True` and y_2 = `False`:

```
x_1 = Causes("magnesium", "quadriplegic")
x_2 = Causes("magnesium", "preeclampsia")
```

Data Model A design challenge is managing complex, unstructured data in a way that enables SMEs to write labeling functions over it. In Snorkel, input data is stored in a *context*

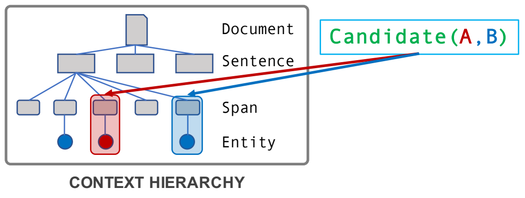


Figure 4.3: Labeling functions take as input a `Candidate` object, representing a data point to be classified. Each `Candidate` is a tuple of `Context` objects, which are part of a hierarchy representing the local context of the `Candidate`.

hierarchy. It is made up of context types connected by parent/child relationships, which are stored in a relational database and made available via an object-relational mapping (ORM) layer built with SQLAlchemy.[4] Each *context type* represents a conceptual component of data to be processed by the system or used when writing labeling functions; for example a document, an image, a paragraph, a sentence, or an embedded table. Candidates—i.e., data points x—are then defined as tuples of contexts (Figure 4.3).

Example 4.1.2. In our running CDR example, the input documents can be represented in Snorkel as a hierarchy consisting of `Documents`, each containing one or more `Sentences`, each containing one or more `Spans` of text. These `Spans` may also be tagged with metadata, such as `Entity` markers identifying them as chemical or disease mentions (Figure 4.3). A candidate is then a tuple of two `Spans`.

4.1.1 A Language for Weak Supervision

Snorkel uses the core abstraction of a labeling function to allow users to specify a wide range of weak supervision sources such as patterns, heuristics, external knowledge bases, crowdsourced labels, and more. This higher-level, less precise input is more efficient to provide (see Section 4.4), and can be automatically denoised and synthesized, as described in subsequent sections.

In this section, we describe our design choices in building an interface for writing labeling functions, which we envision as a unifying programming language for weak supervision. These choices were informed to a large degree by interactions—primarily through weekly office hours—with Snorkel users in bioinformatics, defense, industry, and other

[4]`https://www.sqlalchemy.org/`

areas.[5] For example, while we initially intended to have a more complex structure for labeling functions, with manually specified types and correlation structure, we quickly found that simplicity in this respect was critical to usability (and not empirically detrimental to our ability to model their outputs). We also quickly discovered that users wanted either far more expressivity or far less of it, compared to our first library of function templates. We thus trade off expressivity and efficiency by allowing users to write labeling functions at two levels of abstraction: custom Python functions and declarative operators.

Hand-Defined Labeling Functions: In its most general form, a labeling function is just an arbitrary snippet of code, usually written in Python, which accepts as input a `Candidate` object and either outputs a label or abstains. Often these functions are similar to extract-transform-load scripts, expressing basic patterns or heuristics, but may use supporting code or resources and be arbitrarily complex. Writing labeling functions by hand is supported by the ORM layer, which maps the context hierarchy and associated metadata to an object-oriented syntax, allowing the user to easily traverse the structure of the input data.

Example 4.1.3. In our running example, we can write a labeling function that checks if the word "causes" appears between the chemical and disease mentions. If it does, it outputs `True` if the chemical mention is first and `False` if the disease mention is first. If "causes" does not appear, it outputs `None`, indicating abstention:

```python
def LF_causes(x):
  cs, ce = x.chemical.get_word_range()
  ds, de = x.disease.get_word_range()
  if ce < ds and "causes" in x.parent.words[ce+1:ds]:
      return True
  if de < cs and "causes" in x.parent.words[de+1:cs]:
      return False
  return None
```

We could also write this with Snorkel's declarative interface:

```python
LF_causes = lf_search("{{1}}.*\Wcauses\W.*{{2}}", reverse_args=False)
```

Declarative Labeling Functions: `Snorkel` includes a library of declarative operators that encode the most common weak supervision function types, based on our experience

[5]`http://snorkel.stanford.edu#users`

with users. The semantics and syntax of these operators is simple and easily-customizable, consisting of two main types: (i) labeling function *templates*, which are simply functions that take one or more arguments and output a single labeling function; and (ii) labeling function *generators*, which take one or more arguments and output a set of labeling functions (described below). These functions capture a range of common forms of weak supervision, for example:

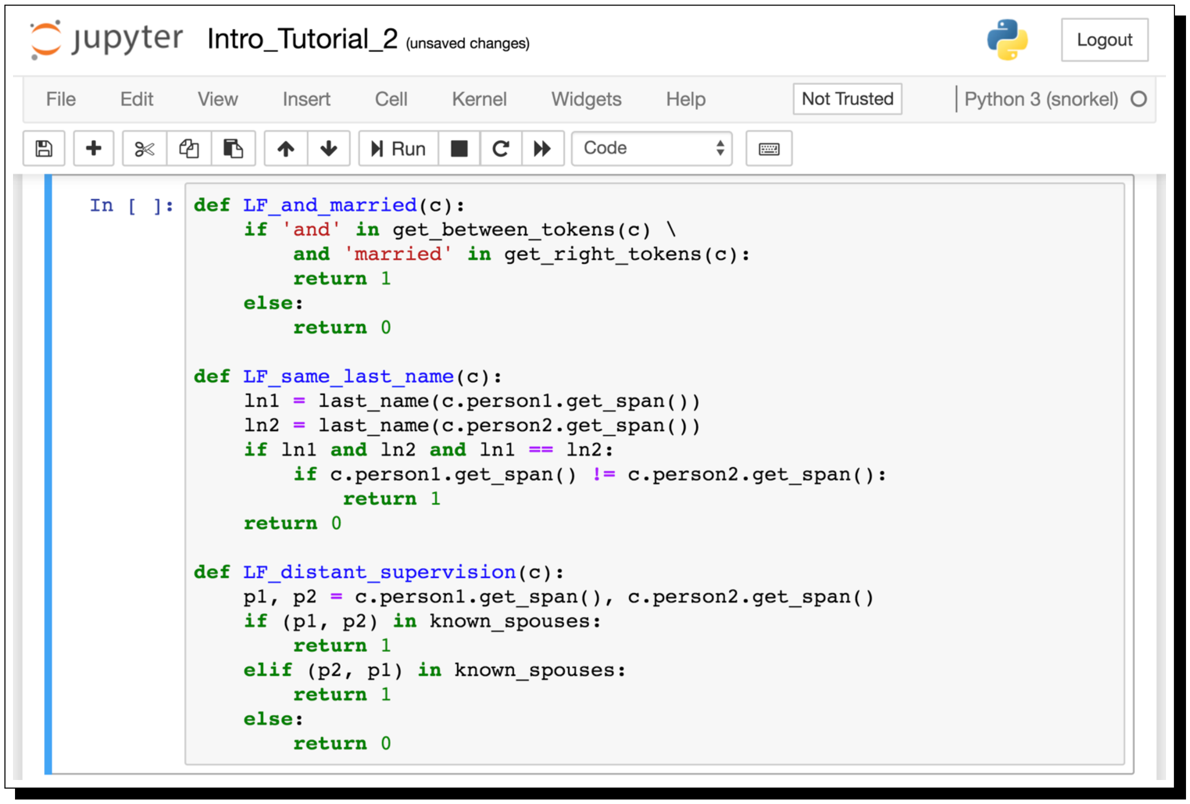
- **Pattern-based:** Pattern-based heuristics embody the motivation of soliciting higher information density input from SMEs. For example, pattern-based heuristics encompass feature annotations [Zaidan and Eisner, 2008] and pattern-bootstrapping approaches [Hearst, 1992; Gupta and Manning, 2014] (Example 4.1.3).
- **Distant supervision:** Distant supervision generates training labels by heuristically aligning data points with an external knowledge base, and is one of the most popular forms of weak supervision [Mintz et al., 2009; Alfonseca et al., 2012; Hoffmann et al., 2011].
- **Weak classifiers:** Classifiers that are insufficient for our task—e.g., limited coverage, noisy, biased, and/or trained on a different dataset—can be used as labeling functions.
- **Labeling function generators:** One higher-level abstraction that we can build on top of labeling functions in `Snorkel` is *labeling function generators*, which generate multiple labeling functions from a single resource, such as crowdsourced labels or distant supervision from structured knowledge bases (Example 4.1.4).

Example 4.1.4. A challenge in traditional distant supervision is that different subsets of knowledge bases have different levels of accuracy and coverage. In our running example, we can use the Comparative Toxicogenomics Database (CTD)[6] as distant supervision, separately modeling different subsets of it with separate labeling functions. For example, we might write one labeling function to label a candidate `True` if it occurs in the "Causes" subset, and another to label it `False` if it occurs in the "Treats" subset. We can write this using a labeling function generator,

[6]`http://ctdbase.org/`

```
LFs_CTD = Ontology(ctd,
    {"Causes": True, "Treats": False})
```

which creates two labeling functions. In this way, generators can be connected to large resources and create hundreds of labeling functions with a line of code.

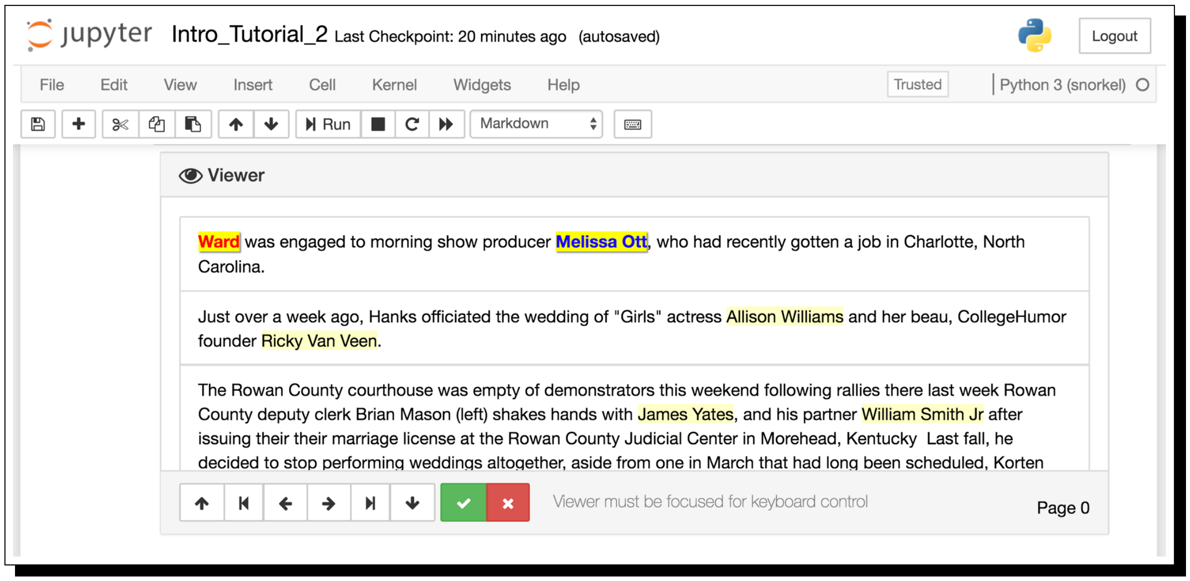


```
def LF_and_married(c):
    if 'and' in get_between_tokens(c) \
        and 'married' in get_right_tokens(c):
        return 1
    else:
        return 0

def LF_same_last_name(c):
    ln1 = last_name(c.person1.get_span())
    ln2 = last_name(c.person2.get_span())
    if ln1 and ln2 and ln1 == ln2:
        if c.person1.get_span() != c.person2.get_span():
            return 1
    return 0

def LF_distant_supervision(c):
    p1, p2 = c.person1.get_span(), c.person2.get_span()
    if (p1, p2) in known_spouses:
        return 1
    elif (p2, p1) in known_spouses:
        return 1
    else:
        return 0
```

Figure 4.4: Labeling functions expressing pattern-matching, heuristic, and distant supervision approaches, respectively, in Snorkel's Jupyter notebook interface, for the Spouses example. Full code is available in Snorkel's Intro tutorial.[7]

Interface Implementation Snorkel's interface is designed to be accessible to subject matter expert (SME) users without advanced programming skills. All components run in

[7]https://github.com/HazyResearch/snorkel/tree/master/tutorials/intro


jupyter
Intro_Tutorial_2 Last Checkpoint: 20 minutes ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3 (snorkel)
Run
Markdown
Viewer
Viewer must be focused for keyboard control
Page 0

Ward was engaged to morning show producer Melissa Ott, who had recently gotten a job in Charlotte, North Carolina.

Just over a week ago, Hanks officiated the wedding of "Girls" actress Allison Williams and her beau, CollegeHumor founder Ricky Van Veen.

The Rowan County courthouse was empty of demonstrators this weekend following rallies there last week Rowan County deputy clerk Brian Mason (left) shakes hands with James Yates, and his partner William Smith Jr after issuing their their marriage license at the Rowan County Judicial Center in Morehead, Kentucky Last fall, he decided to stop performing weddings altogether, aside from one in March that had long been scheduled, Korten

Figure 4.5: The data viewer utility in `Snorkel`, showing candidate spouse relation mentions from the Spouses example, composed of person-person mention pairs.

Jupyter iPython notebooks,[8] including writing labeling functions.[9] Users can therefore write labeling functions as arbitrary Python functions for maximum flexibility (Figure 4.4). We also provide a library of labeling function primitives and generators to more declaratively program weak supervision, and a viewer utility (Figure 4.5) that displays candidates, and also supports annotation, e.g., for constructing a small held-out test set for end evaluation.

Execution Model Since labeling functions operate on discrete candidates, their execution is embarrassingly parallel. If Snorkel is connected to a relational database that supports simultaneous connections, e.g., PostgreSQL, then the master process (usually the notebook kernel) distributes the primary keys of the candidates to be labeled to Python worker processes. The workers independently read from the database to materialize the candidates via the ORM layer, then execute the labeling functions over them. The labels are returned to

[8]`http://jupyter.org/`

[9]Note that all code is open source and available—with tutorials, blog posts, workshop lectures, and other material—at `snorkel.stanford.edu`.

the master process which persists them via the ORM layer. Collecting the labels at the master is more efficient than having workers write directly to the database, due to table-level locking.

Snorkel includes a Spark[10] integration layer, enabling labeling functions to be run across a cluster. Once the set of candidates is cached as a Spark data frame, only the closure of the labeling functions and the resulting labels need to be communicated to and from the workers. This is particularly helpful in Snorkel's iterative workflow. Distributing a large unstructured data set across a cluster is relatively expensive, but only has to be performed once. Then, as users refine their labeling functions, they can be rerun efficiently.

4.1.2 Generative Model

The core operation of `Snorkel` is modeling and integrating the noisy signals provided by a set of labeling functions. Using the data programming covered in Chapter 3, we model the true class label for a data point as a latent variable in a probabilistic model. In the simplest case, we model each labeling function as a noisy "voter" which is *independent*—i.e., makes errors that are uncorrelated with the other labeling functions. This defines a generative model of the votes of the labeling functions as noisy signals about the true label.

We can also model statistical dependencies between the labeling functions to improve predictive performance. For example, if two labeling functions express similar heuristics, we can include this dependency in the model and avoid a "double counting" problem. We observe that such pairwise correlations are the most common, so we focus on them in this paper (though handling higher order dependencies is straightforward). We use the structure learning methods briefly reviewed in Section 3.4 to select a set E of labeling function pairs (j, k)–i.e. edges in the previously defined labeling function dependency graph $G_\lambda = (V, E)$–to model as correlated (see Section 4.2.2).

Now we can construct the full generative model as a factor graph, following the maximum marginal likelihood approach presented in Section 3.2, and which we briefly review in more concrete detail now. We first apply all the labeling functions to the unlabeled data points, resulting in a label matrix Λ, where $\Lambda_{i,j} = \lambda_j^{(i)} = \lambda_j(x^{(i)})$ is the jth labeling function

[10] `https://spark.apache.org/`

applied to the *i*th data point. We then encode the generative model $p_\theta(\Lambda, \vec{y})$ using three factor types, representing the labeling propensity, accuracy, and pairwise correlations of labeling functions:

$$\begin{aligned}
\psi^{\text{Lab}}_{i,j}(\Lambda, \vec{y}) &= \mathbb{1}\{\Lambda_{i,j} \neq 0\} \\
\psi^{\text{Acc}}_{i,j}(\Lambda, \vec{y}) &= \mathbb{1}\{\Lambda_{i,j} = y^{(i)}\} \\
\psi^{\text{Corr}}_{i,j,k}(\Lambda, \vec{y}) &= \mathbb{1}\{\Lambda_{i,j} = \Lambda_{i,k}\} \qquad\qquad (j,k) \in E
\end{aligned}$$

For a given data point $x^{(i)}$, we define the concatenated vector of these factors for all the labeling functions $j = 1, ..., m$ and potential correlations E as $\psi_i(\Lambda, \vec{y})$, and the corresponding vector of parameters $\theta \in \mathbb{R}^{2m+|E|}$. This defines our model:

$$p\theta(\Lambda, \vec{y}) = Z_\theta^{-1} \exp\left(\sum_{i=1}^{n} \theta^T \psi_i(\Lambda, \vec{y})\right),$$

where Z_θ is a normalizing constant. To learn this model *without* access to the true labels $\vec{y}$, we minimize the negative log marginal likelihood given the observed label matrix Λ:

$$\hat{\theta} = \text{argmin}_\theta \; -\log \sum_{\vec{y}} p_\theta(\Lambda, \vec{y}) .$$

We optimize this objective by interleaving stochastic gradient descent steps with Gibbs sampling ones, as presented in Section 3.2. We use the Numbskull library,[11] a Python NUMBA-based Gibbs sampler. We then use the distributions $p_{\hat{\theta}}(y|\lambda)$ as *probabilistic training labels*.

4.1.3 Discriminative Model

The end goal in Snorkel is to train a model that generalizes beyond the information expressed in the labeling functions. We train a discriminative model h_w on our probabilistic labels $\tilde{y}$ by minimizing a *noise-aware* variant of the loss $l(h_w(x^{(i)}), y^{(i)})$, i.e., the expected

[11]`https://github.com/HazyResearch/numbskull`

loss with respect to $\tilde{y}$:

$$\hat{w} = \text{argmin}_w \sum_{i=1}^{n} \mathbb{E}_{\tilde{y} \sim p_{\hat{\theta}}} \left[l(h_w(x^{(i)}), \tilde{y}) \right].$$

Our formal analysis in Sections 3.2.2 and 3.3.3 shows that as we increase the amount of *unlabeled data*, the generalization error of discriminative models trained with Snorkel will decrease at the same asymptotic rate as traditional supervised learning models do with additional hand-labeled data, allowing us to increase predictive performance by adding more unlabeled data. Intuitively, this property holds because as more data is provided, the discriminative model sees more features that co-occur with the heuristics encoded in the labeling functions.

Example 4.1.5. The CDR data contains the sentence, "Myasthenia gravis presenting as weakness after magnesium administration." None of the 33 labeling functions we developed vote on the corresponding `Causes(magnesium, myasthenia gravis)` candidate, i.e., they all abstain. However, a deep neural network trained on probabilistic training labels from Snorkel correctly identifies it as a true mention in our experiments (see Section 4.3).

Snorkel provides connectors for popular machine learning libraries such as TensorFlow [Abadi et al., 2016] and PyTorch [Paszke, 2017], allowing users to exploit commodity models like deep neural networks that do not require hand-engineering of features and have robust predictive performance across a wide range of tasks.

4.2 Weak Supervision Tradeoffs

We study the fundamental question of when—and at what level of complexity—we should expect Snorkel's generative model to yield the greatest predictive performance gains. Understanding these performance regimes can help guide users, and introduces a tradeoff space between predictive performance and speed. We characterize this space in two parts:

first, by analyzing when the generative model can be approximated by an unweighted majority vote, and second, by automatically selecting the complexity of the correlation structure to model. We then introduce a two-stage, rule-based optimizer to support fast development cycles.

4.2.1 Modeling Accuracies

The natural first question when studying systems for weak supervision is, “When does modeling the accuracies of sources improve end-to-end predictive performance?” We study that question in this subsection and propose a heuristic to identify settings in which this modeling step is most beneficial.

Tradeoff Space

We start by considering the *label density* d_Λ of the label matrix Λ, defined as the mean number of non-abstention labels per data point. In the low-density setting, sparsity of labels will mean that there is limited room for even an optimal weighting of the labeling functions to diverge much from the majority vote. Conversely, as the label density grows, known theory confirms that the majority vote will eventually be optimal [Li et al., 2013]. It is the middle-density regime where we expect to most benefit from applying the generative model. We start by defining a measure of the benefit of weighting the labeling functions by their true accuracies—in other words, the predictions of a perfectly estimated generative model—versus an unweighted majority vote:

Definition 1. (Modeling Advantage) Let the weighted majority vote of m labeling functions on data point x with labeling function output vector λ be denoted as $f_\theta(\lambda) = \sum_{j=1}^{m} \theta_j \lambda_j$, and the unweighted majority vote (MV) as $f_1(\lambda) = \sum_{j=1}^{m} \lambda_j$, where we consider the binary classification setting and represent an abstaining vote for simplicity as 0. We define the **modeling advantage** A_θ as the improvement in accuracy of f_θ over f_1 for a dataset:

$$A_\theta(\Lambda, \vec{y}) = \frac{1}{n} \sum_{i=1}^{n} \Big(\mathbb{1}\left\{ y^{(i)} f_\theta(\lambda^{(i)}) > 0 \wedge y^{(i)} f_1(\lambda^{(i)}) \leq 0 \right\} \\ - \mathbb{1}\left\{ y^{(i)} f_\theta(\lambda^{(i)}) \leq 0 \wedge y^{(i)} f_1(\lambda^{(i)}) > 0 \right\} \Big)$$

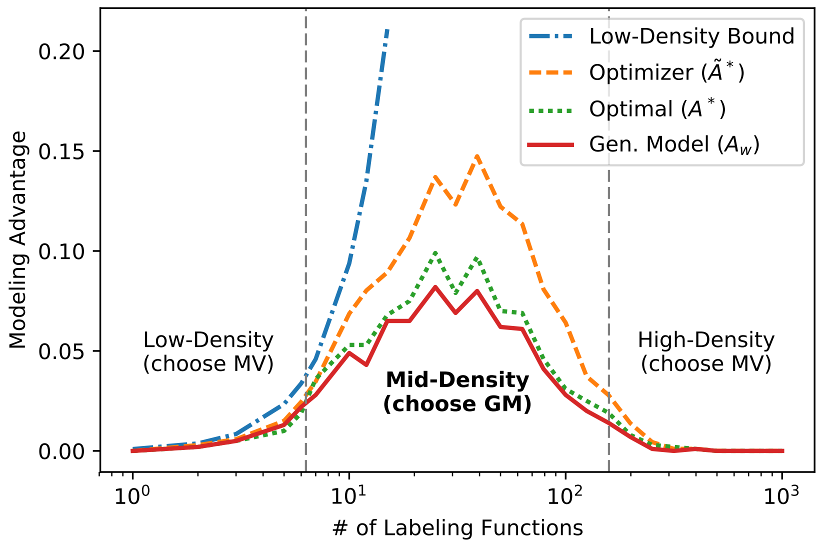


Figure 4.6: A plot of the *modeling advantage*, i.e., the improvement in label accuracy from the generative model, as a function of the number of labeling functions (equivalently, the label density) on a synthetic dataset.[12] We plot the advantage obtained by a learned generative model (GM), A_θ; by an optimal model A^*; the upper bound $\tilde{A}^*$ used in our optimizer; and the low-density bound (Proposition 1).

In other words, A_θ is the number of times f_θ correctly disagrees with f_1 on a label, minus the number of times it incorrectly disagrees. Let the **optimal advantage** $A^* = A_{\theta^*}$ be the advantage using the optimal weights θ^* (WMV*).

Additionally, let:

$$\overline{\alpha}^* = \frac{1}{m}\sum_{j=1}^{m} \alpha_j^* = \frac{1}{m}\sum_{j=1}^{m} 1/(1+\exp(\theta_j^*))$$

be the average accuracies of the labeling functions. To build intuition, we start by analyzing the optimal advantage for three regimes of label density (see Figure 4.6):

[12]We generate a class-balanced dataset of $n = 1000$ data points with binary labels, and m independent labeling functions with average accuracy 75% and a fixed 10% probability of voting.

Low Label Density In this sparse setting, very few data points have more than one non-abstaining label; only a small number have multiple conflicting labels. We have observed this occurring, for example, in the early stages of application development. We see that with non-adversarial labeling functions ($\theta^* > 0$), even an optimal generative model (WMV*) can only disagree with MV when there are disagreeing labels, which will occur infrequently. We see that the expected optimal advantage will have an upper bound that falls quadratically with label density:

Proposition 1. (Low-Density Upper Bound) Assume that $P(\Lambda_{i,j} \neq 0) = p_l \ \forall i, j$, and $\theta_j^* > 0 \ \forall j$. Then, the expected label density is $\bar{d} = mp_l$, and

$$\mathbb{E}_{\Lambda,y,\theta^*}\left[A^*\right] \leq \bar{d}^2 \overline{\alpha}^* (1 - \overline{\alpha}^*) \tag{4.1}$$

Proof Sketch: We bound the advantage above by computing the expected number of pairwise disagreements; for details, see [Ratner et al., 2017a].

High Label Density In this setting, the majority of the data points have a large number of labels. For example, we might be working in an extremely high-volume crowdsourcing setting, or an application with many high-coverage knowledge bases as distant supervision. Under modest assumptions—namely, that the average labeling function accuracy $\overline{\alpha}^*$ is greater than 50%—it is known that the majority vote converges exponentially to an optimal solution as the average label density $\bar{d}$ increases, which serves as an upper bound for the expected optimal advantage as well:

Proposition 2. (High-Density Upper Bound) Assume that $P(\Lambda_{i,j} \neq 0) = p_l \ \forall i, j$, and that $\overline{\alpha}^* > \frac{1}{2}$. Then:

$$\mathbb{E}_{\Lambda,y,\theta^*}\left[A^*\right] \leq e^{-2p_l\left(\overline{\alpha}^* - \frac{1}{2}\right)^2 \bar{d}} \tag{4.2}$$

Proof: This follows from an application of Hoeffding's inequality; for details, see [Ratner et al., 2017a].

Table 4.1: Modeling advantage A_θ attained using a generative model for several applications in Snorkel (Section 4.3), the upper bound $\tilde{A}^*$ used by our optimizer, the modeling strategy selected by the optimizer—either majority vote (MV) or generative model (GM)—and the empirical label density d_Λ.

Dataset	A_θ (%)	$\tilde{A}^*$ (%)	Modeling Strategy	d_Λ
Radiology	7.0	12.4	**GM**	2.3
CDR	4.9	7.9	**GM**	1.8
Spouses	4.4	4.6	**GM**	1.4
Chem	0.1	0.3	**MV**	1.2
EHR	2.8	4.8	**GM**	1.2

Medium Label Density In this middle regime, we expect that modeling the accuracies of the labeling functions will deliver the greatest gains in predictive performance because we will have many data points with a small number of disagreeing labeling functions. For such points, the estimated labeling function accuracies can heavily affect the predicted labels. We indeed see gains in the empirical results using an independent generative model that only includes accuracy factors $\psi_{i,j}^{\text{Acc}}$ (Table 4.1). Furthermore, the guarantees in Section 3.2.2 establish that we can learn the optimal weights, and thus approach the optimal advantage.

Automatically Choosing a Modeling Strategy

The bounds in the previous subsection imply that there are settings in which we should be able to safely skip modeling the labeling function accuracies, simply taking the unweighted majority vote instead. However, in practice, the overall label density d_Λ is insufficiently precise to determine the transition points of interest, given a user time-cost tradeoff preference (characterized by the *advantage tolerance* parameter γ in Algorithm 3). We show this in Table 4.1 using our application data sets from Section 4.3. For example, we see that the Chem and EHR label matrices have equivalent label densities; however, modeling the labeling function accuracies has a much greater effect for EHR than for Chem.

Instead of simply considering the average label density d_Λ, we instead develop a best-case heuristic based on looking at the ratio of positive to negative labels for each data point. This heuristic serves as an upper bound to the true expected advantage, and thus we can use it to determine when we can safely skip training the generative model (see Algorithm 3). Let $c_y(\Lambda_i) = \sum_{j=1}^{m} \mathbb{1}\left\{\Lambda_{i,j} = y\right\}$ be the counts of labels of class y for x_i, and assume that the true labeling function weights lie within a fixed range, $\theta_j \in [\theta_{min}, \theta_{max}]$ and have a mean $\bar{\theta}$.[13] Then, define:

$$\Phi(\Lambda_i, y) = \mathbb{1}\left\{c_y(\Lambda_i)\theta_{max} > c_{-y}(\Lambda_i)\theta_{min}\right\}$$
$$\tilde{A}^*(\Lambda) = \frac{1}{n}\sum_{i=1}^{n}\sum_{y\in\pm 1} \mathbb{1}\left\{y f_1(\Lambda_i) \le 0\right\} \Phi(\Lambda_i, y)\sigma(2 f_{\bar{\theta}}(\Lambda_i) y)$$

where $\sigma(\cdot)$ is the sigmoid function, $f_{\bar{\theta}}$ is majority vote with all weights set to the mean $\bar{\theta}$, and $\tilde{A}^*(\Lambda)$ is the predicted modeling advantage used by our optimizer. Essentially, we are taking the expected counts of instances in which a weighted majority vote could possibly flip the incorrect predictions of unweighted majority vote under best case conditions, which is an upper bound for the expected advantage:

Proposition 3. (**Optimizer Upper Bound**) Assume that the labeling functions have accuracy parameters (log-odds weights) $\theta_j \in [\theta_{min}, \theta_{max}]$, and have $\mathbb{E}[\theta] = \bar{\theta}$. Then:

$$\mathbb{E}_{y,\theta^*}\left[A^* \mid \Lambda\right] \le \tilde{A}^*(\Lambda) \tag{4.3}$$

Proof Sketch: We upper-bound the modeling advantage by the expected number of instances in which WMV* is correct and MV is incorrect. We then upper-bound this by using the best-case probability of the weighted majority vote being correct given $(\theta_{min}, \theta_{max})$.

We apply $\tilde{A}^*$ to a synthetic dataset and plot in Figure 4.6. Next, we compute $\tilde{A}^*$ for the labeling matrices from experiments in Section 4.3, and compare with the empirical advantage of the trained generative models (Table 4.1).[14] We see that our approximate

[13]We fix these at defaults of $(\theta_{min}, \bar{\theta}, \theta_{max}) = (0.5, 1.0, 1.5)$, which corresponds to assuming labeling functions have accuracies between 62% and 82%, and an average accuracy of 73%.

[14] Note that in Section 4.3, due to known negative class imbalance in relation extraction problems, we default to a negative value if majority vote yields a tie-vote label of 0. Thus our reported F1 score metric hides instances in which the generative model learns to correctly (or incorrectly) break ties. In Table 4.1, however, we do count such instances as improvements over majority vote, as these instances have an effect on the training of the end discriminative model (they yield additional

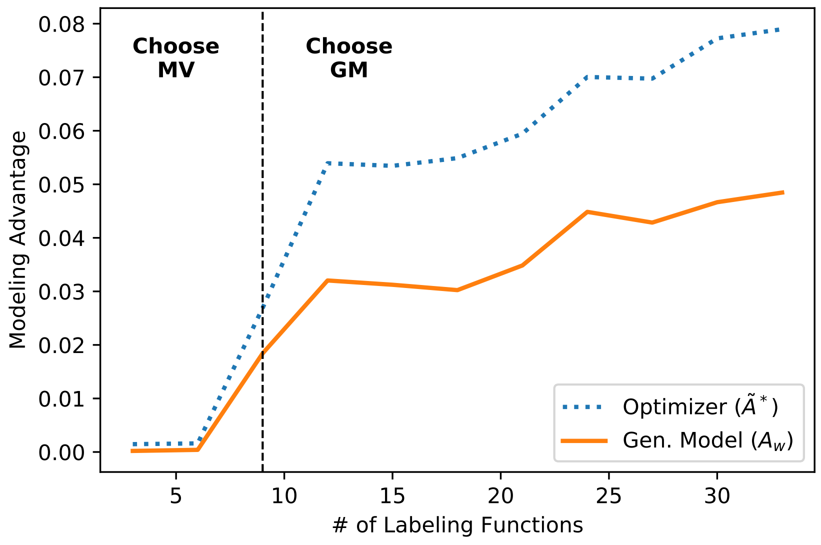


Figure 4.7: The predicted ($\tilde{A}^*$) and actual (A_θ) advantage of using the generative labeling model (GM) over majority vote (MV) on the CDR application as the number of LFs is increased. At 9 LFs, the optimizer switches from choosing MV to choosing GM; this leads to faster modeling in early development cycles, and more accurate results in later cycles.

quantity $\tilde{A}^*$ serves as a correct guide in all cases for determining which modeling strategy to select, which for the mature applications reported on is indeed most often the generative model. However, we see that while EHR and Chem have equivalent label densities, our optimizer correctly predicts that Chem can be modeled with majority vote, speeding up each pipeline execution by 1.8×.

Accelerating Initial Development Cycles

We find in our applications that the optimizer can save execution time especially during the initial cycles of iterative development. To illustrate this empirically, in Figure 4.7 we measure the modeling advantage of the generative model versus a majority vote of the labeling functions on increasingly large random subsets of the CDR labeling functions. We see that the modeling advantage grows as the number of labeling functions increases, and that our optimizer approximation closely tracks it, providing evidence that the optimizer

training labels).

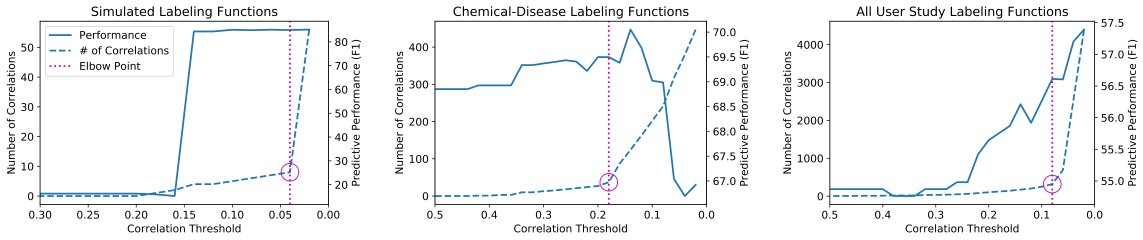


Figure 4.8: Predictive performance of the generative model and number of learned correlations versus the correlation threshold ϵ. The selected elbow point achieves a good tradeoff between predictive performance and computational cost (linear in the number of correlations). Left: simulation of structure learning correcting the generative model. Middle: the CDR task. Right: all user study labeling functions for the Spouses task.

can save execution time by choosing to skip the generative model and run majority vote instead during the initial cycles of iterative development.

4.2.2 Modeling Structure

In this subsection, we consider modeling additional statistical structure beyond the conditionally-independent model. We study the tradeoff between predictive performance and computational cost, and describe how to automatically select a good point in this tradeoff space.

Structure Learning We observe many Snorkel users writing labeling functions that are statistically dependent. Examples we have observed include:

- Functions that are variations of each other, such as checking for matches against similar regular expressions.
- Functions that operate on correlated inputs, such as raw tokens of text and their lemmatizations.
- Functions that use correlated sources of knowledge, such as distant supervision from overlapping knowledge bases.

Modeling such dependencies is important because they affect our estimates of the true labels. Consider the extreme case in which not accounting for dependencies is catastrophic:

Example 4.2.1. Consider a set of 10 labeling functions, where 5 are perfectly correlated, i.e., they vote the same way on every data point, and 5 are conditionally independent given the true label. If the correlated labeling functions have accuracy $\alpha = 50\%$ and the uncorrelated ones have accuracy $\beta = 99\%$, then the maximum likelihood estimate of their accuracies according to the independent model is $\hat{\alpha} = 100\%$ and $\hat{\beta} = 50\%$.

Specifying a generative model to account for such dependencies by hand is impractical for three reasons. First, it is difficult for non-expert users to specify these dependencies. Second, as users iterate on their labeling functions, their dependency structure can change rapidly, like when a user relaxes a labeling function to label many more candidates. Third, the dependency structure can be dataset specific, making it impossible to specify a priori, such as when a corpus contains many strings that match multiple regular expressions used in different labeling functions. We observed users of early versions of Snorkel struggling for these reasons to construct accurate and efficient generative models with dependencies. We therefore seek a method that can quickly identify an appropriate dependency structure from the labeling function outputs Λ alone.

Naively, we could include all dependencies of interest, such as all pairwise correlations, in the generative model and perform parameter estimation. However, this approach is impractical. For 100 labeling functions and 10,000 data points, estimating parameters with all possible correlations takes roughly 45 minutes. When multiplied over repeated runs of hyperparameter searching and development cycles, this cost greatly inhibits labeling function development. We therefore turn to the methods for automatically selecting which dependencies to model without access to ground truth briefly reviewed in Section 3.4; while both involve a manually set threshold hyperparameter which thereby induces a complexity-accuracy tradeoff, for concreteness we focus on the first approach in Section 3.4.1. It uses a pseudolikelihood estimator, which does not require any sampling or other approximations to compute the objective gradient exactly. It is much faster than maximum likelihood estimation over all pairwise correlations, taking 15 seconds to select pairwise correlations to be modeled among 100 labeling functions with 10,000 data points. However, this approach relies on a selection threshold hyperparameter ϵ which induces a tradeoff space between predictive performance and computational cost.

Tradeoff Space

Such structure learning methods, whether pseudolikelihood or likelihood-based, crucially depend on a selection threshold ϵ for deciding which dependencies to add to the generative model. Fundamentally, the choice of ϵ determines the complexity of the generative model.[15] We study the tradeoff between predictive performance and computational cost that this induces. We find that generally there is an “elbow point” beyond which the number of correlations selected—and thus the computational cost—explodes, and that this point is a safe tradeoff point between predictive performance and computation time.

Predictive Performance At one extreme, a very large value of ϵ will not include any correlations in the generative model, making it identical to the conditionally-independent model. As ϵ is decreased, correlations will be added. At first, when ϵ is still high, only the strongest correlations will be included. As these correlations are added, we observe that the generative model’s predictive performance tends to improve. Figure 4.8, left, shows the result of varying ϵ in a simulation where more than half the labeling functions are correlated. After adding a few key dependencies, the generative model resolves the discrepancies among the labeling functions. Figure 4.8, middle, shows the effect of varying ϵ for the CDR task. Predictive performance improves as ϵ decreases until the model overfits. Finally, we consider a large number of labeling functions that are likely to be correlated. In the user study described in Section 4.4, participants wrote labeling functions for the Spouses task. We combined all 125 of their functions and studied the effect of varying ϵ. Here, we expect there to be many correlations since it is likely that users wrote redundant functions. We see in Figure 4.8, right, that structure learning surpasses the best performing individual’s generative model (50.0 F1).

[15]Specifically, ϵ is both the coefficient of the ℓ_1 regularization term used to induce sparsity, and the minimum absolute weight in log scale that a dependency must have to be selected.

Computational Cost Computational cost is correlated with model complexity. Since learning in Snorkel is done with a Gibbs sampler, the overhead of modeling additional correlations is linear in the number of correlations. The dashed lines in Figure 4.8 show the number of correlations included in each model versus ϵ. For example, on the Spouses task, fitting the parameters of the generative model at $\epsilon = 0.5$ takes 4 minutes, and fitting its parameters with $\epsilon = 0.02$ takes 57 minutes. Further, parameter estimation is often run repeatedly during development for two reasons: (i) fitting generative model hyperparameters using a development set requires repeated runs, and (ii) as users iterate on their labeling functions, they must re-estimate the generative model to evaluate them.

Automatically Choosing a Model

Based on our observations, we seek to automatically choose a value of ϵ that trades off between predictive performance and computational cost using the labeling functions' outputs Λ alone. Including ϵ as a hyperparameter in a grid search over a development set is generally not feasible because of its large effect on running time. We therefore want to choose ϵ before other hyperparameters, without performing any parameter estimation. We propose using the number of correlations selected at each value of ϵ as an inexpensive indicator. The dashed lines in Figure 4.8 show that as ϵ decreases, the number of selected correlations follows a pattern. Generally, the number of correlations grows slowly at first, then hits an "elbow point" beyond which the number explodes, which fits the assumption that the correlation structure is sparse. In all three cases, setting ϵ to this elbow point is a safe tradeoff between predictive performance and computational cost. In cases where performance grows consistently (left and right), the elbow point achieves most of the predictive performance gains at a small fraction of the computational cost. For example, on Spouses (right), choosing $\epsilon = 0.08$ achieves a score of 56.6 F1—within one point of the best score—but only takes 8 minutes for parameter estimation. In cases where predictive performance eventually degrades (middle), the elbow point also selects a relatively small number of correlations, giving an 0.7 F1 point improvement and avoiding overfitting.

Performing structure learning for many settings of ϵ is inexpensive, especially since the search needs to be performed only once before tuning the other hyperparameters. On the large number of labeling functions in the Spouses task, structure learning for 25 values

Algorithm 3 Modeling Strategy Optimizer

```
Input: Label matrix $\Lambda \in (\mathcal{Y} \cup \{\emptyset\})^{n \times m}$,
advantage tolerance $\gamma$, structure search resolution $\eta$
Output: Modeling strategy
if $\tilde{A}^*(\Lambda) < \gamma$ then return MV
Structures ← [ ]
for $i$ from 1 to $\frac{1}{2\eta}$ do
    $\epsilon \leftarrow i \cdot \eta$
    $E \leftarrow$ LearnStructure($\Lambda, \epsilon$)
    Structures.append($|E|, \epsilon$)
$\epsilon \leftarrow$ SelectElbowPoint(Structures) return $\mathbf{GM}_\epsilon$
```

of ϵ takes 14 minutes. On CDR, with a smaller number of labeling functions, it takes 30 seconds. Further, if the search is started at a low value of ϵ and increased, it can often be terminated early, when the number of selected correlations reaches a low value. Selecting the elbow point itself is straightforward. We use the point with greatest absolute difference from its neighbors, but more sophisticated schemes can also be applied [Satopaa et al., 2011]. Our full optimization algorithm for choosing a modeling strategy and (if necessary) correlations is shown in Algorithm 3.

4.3 Experiments

We evaluate Snorkel by drawing on deployments developed in collaboration with users. We report on two real-world deployments and four tasks on open-source data sets representative of other deployments. We then cover a user study in Section 4.4, and describe other real-world applications of Snorkel in Section 4.5. Our evaluation is designed to support the following three main claims:

- **Snorkel outperforms distant supervision baselines.** In *distant supervision* [Mintz et al., 2009], one of the most popular forms of weak supervision used in practice, an external knowledge base is heuristically aligned with input data to serve as noisy training labels. By allowing users to easily incorporate a broader, more heterogeneous set of weak supervision sources—for example, pattern matching, structure-based, and other more complex heuristics—Snorkel exceeds models trained via distant supervision by an average of 132%.

- **Snorkel approaches hand supervision.** We see that by writing tens of labeling functions, we were able to approach or match results using hand-labeled training data which took weeks or months to assemble, coming within 2.11% of the F1 score of hand supervision on relation extraction tasks and an average 5.08% accuracy or AUC on cross-modal tasks, for an average 3.60% across all tasks.

- **Snorkel enables a new interaction paradigm.** In Section 4.4, we measure Snorkel's efficiency and ease-of-use by reporting on a user study of biomedical researchers from across the U.S. These participants learned to write labeling functions to extract relations from news articles as part of a two-day workshop on learning to use Snorkel, and matched or outperformed models trained on hand-labeled training data, showing the efficiency of Snorkel's process even for first-time users.

We now describe our results in detail. First, we describe the six applications that validate our claims. We then show that Snorkel's generative modeling stage helps to improve the predictive performance of the discriminative model, demonstrating that it is 5.81% more accurate when trained on Snorkel's probabilistic labels versus labels produced by an unweighted average of labeling functions. We also validate that the ability to incorporate many different types of weak supervision incrementally improves results with an ablation study.

Applications

To evaluate the effectiveness of Snorkel, we consider several real-world deployments and tasks on open-source datasets that are representative of other deployments in information extraction, medical image classification, and crowdsourced sentiment analysis. Summary statistics of the tasks are provided in Table 4.2.

Table 4.2: Number of labeling functions, fraction of positive labels (for binary classification tasks), number of training documents, and number of training candidates for each task.

Task	# LFs	% Pos.	# Docs	# Candidates
Chem	16	4.1	1,753	65,398
EHR	24	36.8	47,827	225,607
CDR	33	24.6	900	8,272
Spouses	11	8.3	2,073	22,195
Radiology	18	36.0	3,851	3,851
Crowd	102	-	505	505

Table 4.3: Evaluation of Snorkel on relation extraction tasks from text. Snorkel's generative and discriminative models consistently improve over distant supervision, measured in F1, the harmonic mean of precision (P) and recall (R). We compare with hand-labeled data when available, coming within an average of 1 F1 point.

	Distant Supervision			Snorkel (Gen.)				Snorkel (Disc.)				Hand Supervision		
Task	P	R	**F1**	P	R	**F1**	**Lift**	P	R	**F1**	**Lift**	P	R	**F1**
Chem	11.2	41.2	17.6	78.6	21.6	33.8	+16.2	87.0	39.2	54.1	+36.5	-	-	-
EHR	81.4	64.8	72.2	77.1	72.9	74.9	+2.7	80.2	82.6	81.4	+9.2	-	-	-
CDR	25.5	34.8	29.4	52.3	30.4	38.5	+9.1	38.8	54.3	45.3	+15.9	39.9	58.1	47.3
Spouses	9.9	34.8	15.4	53.5	62.1	57.4	+42.0	48.4	61.6	54.2	+38.8	47.8	62.5	54.2

Discriminative Models One of the key bets in Snorkel's design is that the trend of increasingly powerful, open-source machine learning tools (e.g., models, pre-trained word embeddings and initial layers, automatic tuners, etc.) will only continue to accelerate. To best take advantage of this, Snorkel creates probabilistic training labels for any discriminative model with a standard loss function.

In the following experiments, we control for end model selection by using currently popular, standard choices across all settings. For text modalities, we choose a bidirectional long short term memory (LSTM) sequence model [Graves and Schmidhuber, 2005], and for the medical image classification task we use a 50-layer ResNet [He et al., 2016] pre-trained on the ImageNet object classification dataset [Deng et al., 2009]. Both models are implemented in TensorFlow [Abadi et al., 2016] and trained using the Adam optimizer [Kingma and Ba, 2014], with hyperparameters selected via random grid search using a small labeled development set. Final scores are reported on a held-out labeled test set. See [Ratner et al.,

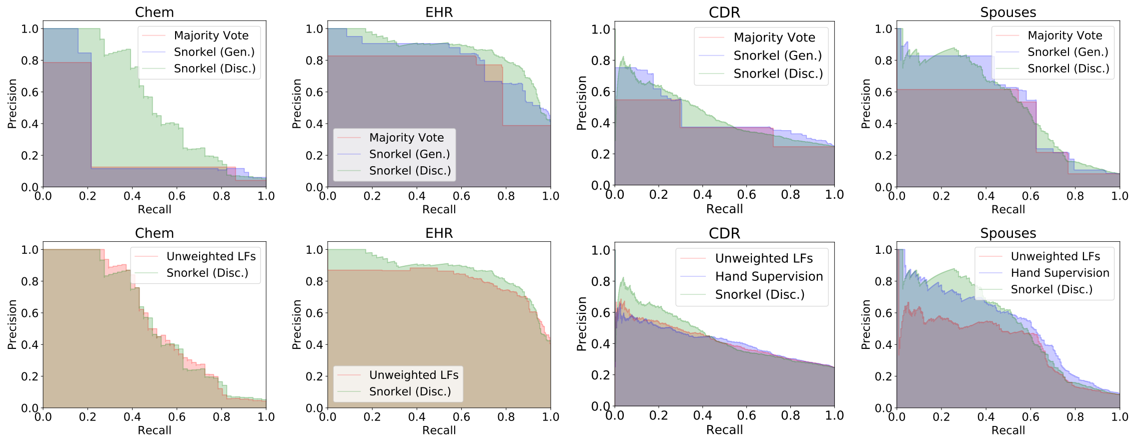


Figure 4.9: Precision-recall curves for the relation extraction tasks. The top plots compare a majority vote of all labeling functions, Snorkel's generative model, and Snorkel's discriminative model. They show that the generative model improves over majority vote by providing more granular information about candidates, and that the discriminative model can generalize to candidates that no labeling functions label. The bottom plots compare the discriminative model trained on an unweighted combination of the labeling functions, hand supervision (when available), and Snorkel's discriminative model. They show that the discriminative model benefits from the weighted labels provided by the generative model, and that Snorkel is competitive with hand supervision, particularly in the high-precision region.

2017a] for additional details.

A key takeaway of the following results is that the discriminative model generalizes beyond the heuristics encoded in the labeling functions (as in Example 4.1.5). In Section 4.3, we see that on relation extraction applications the discriminative model improves performance over the generative model primarily by increasing recall by 43.15% on average. In Section 4.3, the discriminative model classifies entirely new modalities of data to which the labeling functions cannot be applied.

Data Set Details Additional information about the sizes of the datasets are included in Table 4.4. Specifically, we report the size of the (unlabeled) training set and hand-labeled development and test sets, in terms of number of candidates. Note that the development and test sets can be orders of magnitude smaller than the training sets. Labeled development and

Table 4.4: Number of candidates in the training, development, and test splits for each dataset.

Task	# Train.	# Dev.	# Test
Chem	65,398	1,292	1,232
EHR	225,607	913	604
CDR	8,272	888	4,620
Spouses	22,195	2,796	2,697
Radiology	3,851	385	385
Crowd	505	63	64

test sets were either used when already available as part of a benchmark dataset, or labeled with the help of our collaborators, limited to several hours of labeling time maximum. Note that test sets were labeled by individuals not involved with labeling function development to keep the test sets properly blinded.

Relation Extraction from Text

We first focus on four relation extraction tasks on text data, as it is a challenging and common class of problems that are well studied and for which distant supervision is often considered. Predictive performance is summarized in Table 4.3, and precision-recall curves are shown in Figure 4.9. We briefly describe each task.

Scientific Articles (Chem) With modern online repositories of scientific literature, such as PubMed[16] for biomedical articles, research results are more accessible than ever before. However, actually extracting fine-grained pieces of information in a structured format and using this data to answer specific questions at scale remains a significant open challenge for researchers. To address this challenge in the context of drug safety research, Stanford and U.S. Food and Drug Administration (FDA) collaborators used Snorkel to develop a system for extracting chemical reagent and reaction product relations from PubMed abstracts. The goal was to build a database of chemical reactions that researchers at the FDA can use

[16]`https://www.ncbi.nlm.nih.gov/pubmed/`

to predict unknown drug interactions. We used the chemical reactions described in the Metacyc database [Caspi et al., 2016] for distant supervision.

Electronic Health Records (EHR) As patients' clinical records increasingly become digitized, researchers hope to inform clinical decision making by retrospectively analyzing large patient cohorts, rather than conducting expensive randomized controlled studies. However, much of the valuable information in electronic health records (EHRs)—such as fine-grained clinical details, practitioner notes, etc.—is not contained in standardized medical coding systems, and is thus locked away in the unstructured text notes sections. In collaboration with researchers and clinicians at the U.S. Department of Veterans Affairs, Stanford Hospital and Clinics (SHC), and the Stanford Center for Biomedical Informatics Research, we used Snorkel to develop a system to extract structured data from unstructured EHR notes. Specifically, the system's task was to extract mentions of pain levels at precise anatomical locations from clinician notes, with the goal of using these features to automatically assess patient well-being and detect complications after medical interventions like surgery. To this end, our collaborators created a cohort of 5,800 patients from SHC EHR data, with visit dates between 1995 and 2015, resulting in 500K unstructured clinical documents. Since distant supervision from a knowledge base is not applicable, we compared against regular-expression-based labeling previously developed for this task.

Chemical-Disease Relations (CDR) We used the 2015 BioCreative chemical-disease relation dataset [Wei et al., 2015], where the task is to identify mentions of causal links between chemicals and diseases in PubMed abstracts. We used all pairs of chemical and disease mentions co-occurring in a sentence as our candidate set. We used the Comparative Toxicogenomics Database (CTD) [P. et al., 2016] for distant supervision, and additionally wrote labeling functions capturing language patterns and information from the context hierarchy. To evaluate Snorkel's ability to discover previously unknown information, we randomly removed half of the relations in CTD and evaluated on candidates not contained in the remaining half.

Table 4.5: Evaluation on cross-modal experiments. Labeling functions that operate on or represent one modality (text, crowd workers) produce training labels for models that operate on another modality (images, text), and approach the predictive performance of large hand-labeled training datasets.

Task	Snorkel (Disc.)	Hand Supervision
Radiology (AUC)	72.0	76.2
Crowd (Acc)	65.6	68.8

Spouses Our fourth task is to identify mentions of spouse relationships in a set of news articles from the Signal Media dataset [Corney et al., 2016]. We used all pairs of person mentions (tagged with SpaCy's NER module[17]) co-occurring in the same sentence as our candidate set. To obtain hand-labeled data for evaluation, we crowdsourced labels for the candidates via Amazon Mechanical Turk, soliciting labels from three workers for each example and assigning the majority vote. We then wrote labeling functions that encoded language patterns and distant supervision from DBpedia [Lehmann et al., 2014].

Cross-Modal: Images & Crowdsourcing

In the *cross-modal* setting, we write labeling functions over one data modality (e.g., a text report, or the votes of crowd workers) and use the resulting labels to train a classifier defined over a second, totally separate modality (e.g., an image or the text of a tweet). This demonstrates the flexibility of Snorkel, in that the labeling functions (and by extension, the generative model) do not need to operate over the same domain as the discriminative model being trained. Predictive performance is summarized in Table 4.5.

Abnormality Detection in Lung Radiographs (Rad) In many real-world radiology settings, there are large repositories of image data with corresponding narrative text reports, but limited or no labels that could be used for training an image classification model. In this application, in collaboration with radiologists, we wrote labeling functions over the text radiology reports, and used the resulting labels to train an image classifier to detect abnormalities in lung X-ray images. We used a publicly available dataset from the OpenI

[17]`https://spacy.io/`

biomedical image repository[18] consisting of 3,851 distinct radiology reports—composed of unstructured text and Medical Subject Headings (MeSH)[19] codes—and accompanying X-ray images. Note that we briefly describe a more extensive version of this study, done subsequently in collaboration with the Stanford Radiology department, in Section 4.5.

Crowdsourcing (Crowd) We trained a model to perform sentiment analysis using crowd-sourced annotations from the weather sentiment task from Crowdflower.[20] In this task, contributors were asked to grade the sentiment of often-ambiguous tweets relating to the weather, choosing between five categories of sentiment. Twenty contributors graded each tweet, but due to the difficulty of the task and lack of crowd worker filtering, there were many conflicts in worker labels. We represented each crowd worker as a labeling function—showing Snorkel's ability to subsume existing crowdsourcing modeling approaches—and then used the resulting labels to train a text model over the tweets, for making predictions independent of the crowd workers.

Effect of Generative Modeling

An important question is the significance of modeling the accuracies and correlations of the labeling functions on the end predictive performance of the discriminative model (versus in Section 4.2.1, where we only considered the effect on the accuracy of the generative model). We compare Snorkel with a simpler pipeline that skips the generative modeling stage and trains the discriminative model on an unweighted average of the labeling functions' outputs. Table 4.6 shows that the discriminative model trained on Snorkel's probabilistic labels consistently predicts better, improving 5.81% on average. These results demonstrate that the discriminative model effectively learns from the additional signal contained in Snorkel's probabilistic training labels over simpler modeling strategies.

Table 4.6: Comparison between training the discriminative model on the labels estimated by the generative model, versus training on the unweighted average of the LF outputs. Predictive performance gains show that modeling LF noise helps.

Task	Disc. Model on Unweighted LFs	Disc. Model	Lift
Chem	48.6	54.1	+5.5
EHR	80.9	81.4	+0.5
CDR	42.0	45.3	+3.3
Spouses	52.8	54.2	+1.4
Crowd (Acc)	62.5	65.6	+3.1
Rad. (AUC)	67.0	72.0	+5.0

Scaling with Unlabeled Data

One of the most exciting potential advantages of using a programmatic supervision approach as in Snorkel is the ability to incorporate additional *unlabeled* data, which is often cheaply available. The theoretical results in Sections 3.2.2 and 3.3.3 characterizing the data programming approach used predicts that discriminative model generalization risk (i.e., predictive performance on the held-out test set) should improve with additional *unlabeled* data, at the same asymptotic rate as in traditional supervised methods with respect to labeled data. That is, with a fixed amount of effort writing labeling functions, we could then get improved discriminative model performance simply by adding more unlabeled data.

We validate this theoretical prediction empirically on three of our datasets (Figure 4.10). We see that by adding additional unlabeled data—in these datasets, *candidates* from additional documents—we get significant improvements in the end discriminative model performance, with no change in the labeling functions. For example, in the EHR experiment, where we had access to a large unlabeled corpus, we were able to achieve significant gains (8.1 F1 score points) in going from 100 to 50 thousand documents. Further empirical validation of these strong *unlabeled* scaling results can be found in follow-up work using Snorkel in a range of application domains, including aortic valve classification

[18]http://openi.nlm.nih.gov/

[19]https://www.nlm.nih.gov/mesh/meshhome.html

[20]https://www.crowdflower.com/data/weather-sentiment/

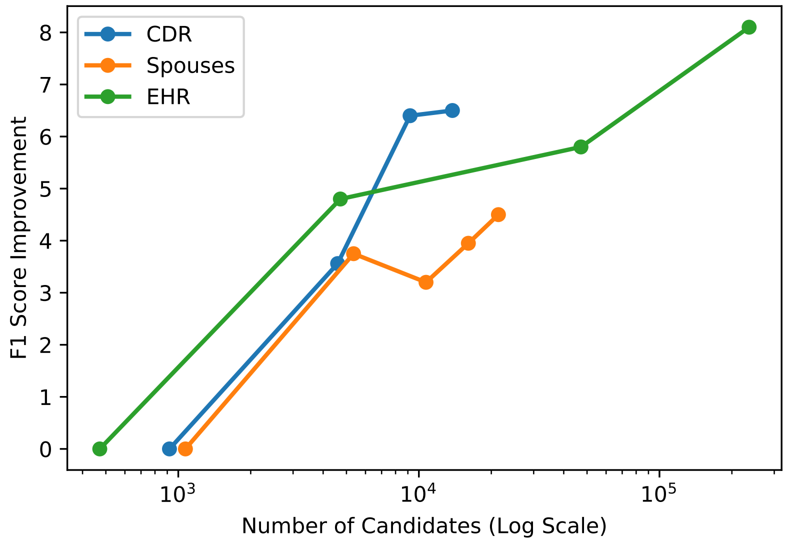


Figure 4.10: The increase in end model performance (measured in F1 score) for different amounts of unlabeled data, measured in the number of candidates. We see that as more *unlabeled* data is added, the performance increases.

in MRI videos [Fries et al., 2019], industrial-scale content classification at Google [Bach et al., 2019], fine-grained named entity recognition [Ratner et al., 2019b], radiology image triage [Khandwala et al., 2017], and others, covered in part in Section 4.5. Based on both this empirical validation, and feedback from Snorkel users in practice, we see this ability to leverage available unlabeled data without any additional user labeling effort as a significant advantage of the proposed weak supervision approach.

Labeling Function Type Ablation

We also examine the impact of different types of labeling functions on end predictive performance, using the CDR application as a representative example of three common categories of labeling functions:

- *Text Patterns:* Basic word, phrase, and regular expression labeling functions.
- *Distant Supervision:* External knowledge bases mapped to candidates, either directly or filtered by a heuristic.

Table 4.7: Labeling function ablation study on CDR. Adding different types of labeling functions improves predictive performance.

LF Type	P	R	F1	Lift
Text Patterns	42.3	42.4	42.3	
+ Distant Supervision	37.5	54.1	44.3	+2.0
+ Structure-based	38.8	54.3	45.3	+1.0

Table 4.8: Self-reported skill levels—no previous experience (New), beginner (Beg.), intermediate (Int.), and advanced (Adv.)—for all user study participants.

Subject	New	Beg.	Int.	Adv.
Python	0	3	8	4
Machine Learning	5	1	4	5
Info. Extraction	2	6	5	2
Text Mining	3	6	4	2

- *Structure-Based:* Labeling functions expressing heuristics over the context hierarchy, e.g., reasoning about position in the document or relative to other candidates.

We show an ablation in Table 4.7, sorting by stand-alone score. We see that distant supervision adds recall at the cost of some precision, as we would expect, but ultimately improves F1 score by 2 points; and that structure-based labeling functions, enabled by Snorkel's context hierarchy data representation, add an additional F1 point.

4.4 User Study

We conducted a formal study of Snorkel to (i) evaluate how quickly subject matter expert (SME) users could learn to write labeling functions, and (ii) empirically validate the core hypothesis that writing labeling functions is more time-efficient than hand-labeling data. Users were given instruction on Snorkel, and then asked to write labeling functions for the Spouses task described in the previous subsection.

Participants In collaboration with the Mobilize Center [Ku et al., 2015], an NIH-funded Big Data to Knowledge (BD2K) center, we distributed a national call for applications to attend a two-day workshop on using Snorkel for biomedical knowledge base construction. Selection criteria included a strong biomedical project proposal and little-to-no prior experience using Snorkel. In total, 15 researchers[21] were invited to attend out of 33 team applications submitted, with varying backgrounds in bioinformatics, clinical informatics, and data mining from universities, companies, and organizations around the United States. The education demographics included 6 bachelors, 4 masters, and 5 Ph.D. degrees. All participants could program in Python, with 80% rating their skill as intermediate or better; 40% of participants had little-to-no prior exposure to machine learning; and 53-60% had no prior experience with text mining or information extraction applications (Table 4.8).

Protocol The first day focused entirely on labeling functions, ranging from theoretical motivations to details of the Snorkel API. Over the course of 7 hours, participants were instructed in a classroom setting on how to use and evaluate models developed using Snorkel. Users were presented with 4 tutorial Jupyter notebooks providing skeleton code for evaluating labeling functions, along with a small labeled development candidate set, and were given 2.5 hours of dedicated development time in aggregate to write their labeling functions. All workshop materials are available online.[22]

Baseline To compare our users' performance against models trained on hand-labeled data, we collected a large hand-labeled dataset via Amazon Mechanical Turk (the same set used in the previous subsection). We then split this into 15 datasets representing 7 hours worth of hand-labeling time each—based on the crowd-worker average of 10 seconds per label—simulating the alternative scenario where users skipped both instruction and labeling function development sessions and instead spent the full day hand-labeling data. Partitions were created by drawing a uniform random sample of 2500 labels from the total Amazon Mechanical Turk-generated Spouse dataset. For 15 such random samples,

[21] One participant declined to write labeling functions, so their score is not included in our analysis.

[22] `https://github.com/HazyResearch/snorkel/tree/master/tutorials/workshop`

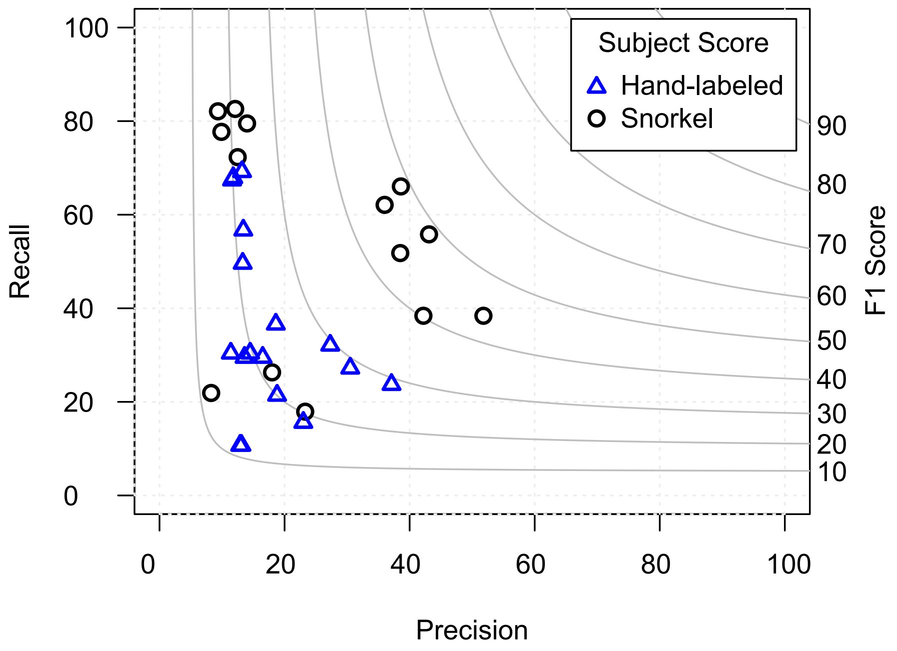


Figure 4.11: Predictive performance attained by our 14 user study participants using Snorkel. The majority (57%) of users matched or exceeded the performance of a model trained on 7 hours (2,500 instances) of hand-labeled data.

the mean F1 score was 20.9 (min:11.7, max: 29.5). Scaling to 55 random partitions, the mean F1 score was 22.5 (min:11.7, max: 34.1).

Results Our key finding is that labeling functions written in Snorkel, even by SME users, can match or exceed a traditional hand-labeling approach. The majority (8) of subjects matched or outperformed these hand-labeled data models. The average Snorkel user's score was 30.4 F1, and the average hand-supervision score was 20.9 F1. The best performing user model scored 48.7 F1, 19.2 points higher than the best supervised model using hand-labeled data. The worst participant scored 12.0 F1, 0.3 points higher that the lowest hand-labeled model. The full distribution of scores by participant, and broken down by participant background, compared against the baseline models trained with hand-labeled data are shown in Figures 4.11 and 4.13 respectively.

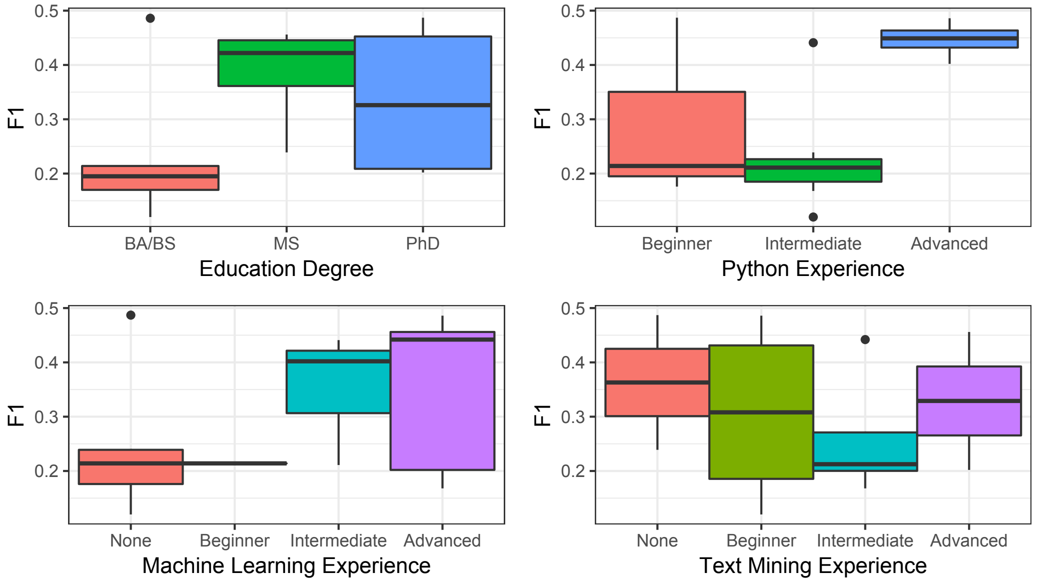


Figure 4.12: The profile of the best performing user by F1 score, was a MS or Ph.D. degree in any field, strong Python coding skills, and intermediate to advanced experience with machine learning. Prior experience with text mining added no benefit.

Additional Details We note that participants only needed to create a fairly small set of labeling functions to achieve the reported performances, writing a median of 10 labeling functions (with a minimum of 2, and a maximum of 15). In general, these labeling functions had simple form; for example, two from our user study:

```
def LF_fictional(c):
    fictional = {"played the husband", "played the wife", "plays the husband", "plays
     the wife", "acting role"}
    if re.search("|".join(fictional), c.get_parent().text, re.I):
        return -1
    else:
        return 0
```

```
def LF_family(c):
    family = {"son", "daughter", "father", "dad", "mother", "mom", "children", "child
    ", "twins", "cousin", "friend", "girlfriend", "boyfriend", "sister", "brother"}
    if len(other.intersection(get_between_tokens(c))) > 0:
        return -1
    else:
        return 0
```

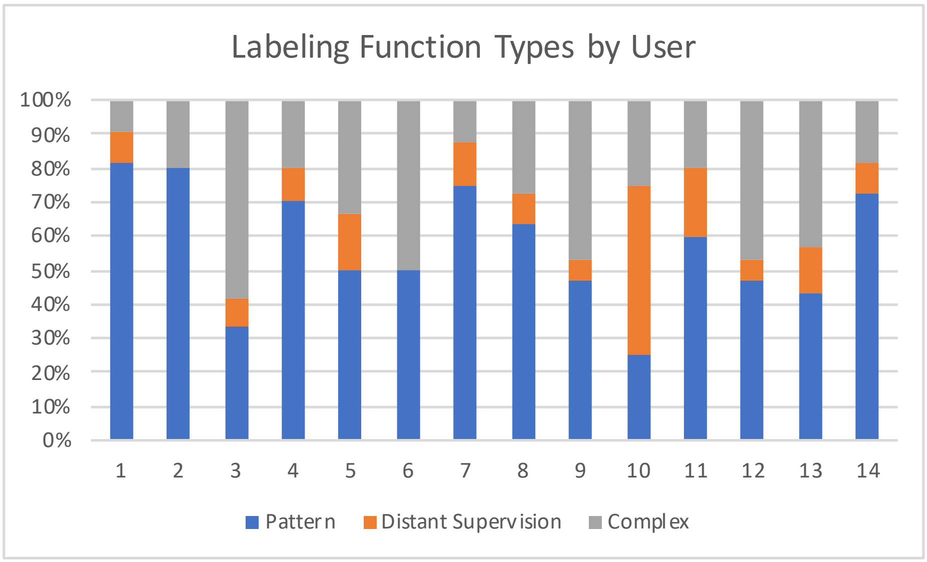


Figure 4.13: We bucketed labeling functions written by user study participants into three types—pattern-based, distant supervision, and complex. Participants tended to mainly write pattern-based labeling functions, but also universally expressed more complex heuristics as well.

Participant labeling functions had a median length of 2 lines of Python code (min:2, max:12). We grouped participant-designed functions into three types:

1. *Pattern-based* (regular expressions, small term sets)
2. *Distant Supervision* (interacts with a knowledge base)
3. *Complex* (misc. heuristics, e.g. counting PERSON named entity tags, comparing last names of a pair of PERSON entities)

On average, 58% of participant's labeling functions where pattern-based (min:25%, max: 82%). The best labeling function design strategy used by participants appeared to be defining small term sets correlated with positive and negative labels. Participants with the lowest F1 scores tended to design labeling functions with low coverage of negative labels. This is a common difficulty encountered when designing labeling functions, as writing heuristics for negative examples is sometimes counter-intuitive. Users with the highest overall F1 scores wrote 1-2 high coverage negative labeling functions and several medium-to-high accuracy positive labeling functions.

We note that the best single participant's pipeline achieved an F1 score of 48.7, compared to the authors' score of 54.2. User study participants favored pattern-based labeling functions; the most common design was creating small positive and negative term sets. Author labeling functions were similar, but were more accurate overall p (e.g., better pattern matching).

4.5 Real-World Applications

One major goal of the work in this thesis—and in particular, the work in designing and building Snorkel as an open-source framework—was to make modern machine learning tools accessible to subject matter experts and machine learning developers alike, so that they could apply these tools to impactful applications. In part, we validate the success of this accessibility objective via user studies such as the one detailed in the previous subsection. However, a bigger goal was to get Snorkel actually deployed in impactful, real-world scientific and production settings. We highlight a sample of the public deployments of Snorkel in medicine, science, and industry below. Additionally, more information can be found at `snorkel.org`, and in the linked open-source code repository.

4.5.1 Knowledge Base Construction

One of the initial focuses of Snorkel, as described in this chapter, was for information (or relation) extraction use cases, often referred to as *knowledge base construction (KBC)* [Ratner et al., 2017b; Ratner and Ré, 2018]. In the broader task of KBC—of which relation extraction, as detailed in the previous subsections, is a sub-task—the goal is ultimately to construct a queryable, structured repository of knowledge that can then be used in a variety of downstream tasks. We briefly review the applications already covered in Section 4.3, and then review several additional applications of interest.

Extracting Information from the Scientific Literature As reviewed in Section 4.3, we applied Snorkel to the challenge of extracting chemical reagent and reaction product relation mentions from PubMed abstracts in the context of drug safety research, in collaboration with researchers from Stanford and the U.S. Food and Drug Administration (FDA). More generally, Snorkel has been used to extract chemical-disease relations (see Section 4.3), genome-phenotype relations [Birgmeier et al., 2017], and other relations and entities [Fries et al., 2017] of scientific interest from the literature.

Building a Genome-Wide Association Study Knowledge Base We highlight one particular application of Snorkel to information extraction from the scientific literature, in which Snorkel was used to power a new system, GWASkb [Kuleshov et al., 2019], for automatically extracting genome-wide association study (GWAS) findings from the scientific literature. GWASkb collected over $6,000$ associations from open-access publications with an estimated recall of 60-80% and precision of 78-94% (available at `http://gwaskb.stanford.edu/`), demonstrating the potential for automated curation of a cornerstone information resource in the biomedical and genomics communities [Kuleshov et al., 2019].

Extraction from Electronic Health Records for Device Monitoring Medical device surveillance is a major challenge to manufacturers, regulatory agencies, and healthcare providers alike. Recently, building on the initial application described in Section 4.3, Callahan et. al. [Callahan et al., 2019] report on applying Snorkel to extract hip replacement implant details and reports of complications and pain from electronic health records (EHRs) with up to 97.4% F1 score, improving by 12.7-53.0% over previous rule-based approaches, and detecting over six times as many complication events compared to using structured data alone, demonstrating the potential of machine learning models—driven by Snorkel—for EHR patient and device monitoring [Callahan et al., 2019].

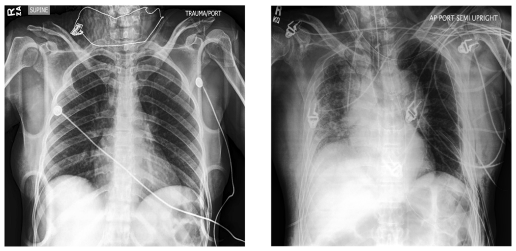
Extraction from Semi-Structured or Richly-Formatted Data Snorkel has also been used as part of a recent system, Fonduer[23], aimed at extracting information from semi-structured or *richly-formatted* data, e.g. data involving textual, structural, tabular, and visual information [Wu et al., 2018]. Fonduer achieved an average 41 F1 score point improvement over expert-curated knowledge bases in four real-world applications covering PDF electronics part sheets extraction, advertising, paleontology, and genomics, and was additionally deployed in production at a major technology company's web product.

4.5.2 Medical Imaging & Monitoring

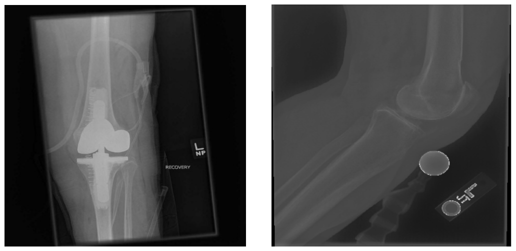
Another area where *labeling* of training data is a major bottleneck is in medical imaging: commodity image classification models (e.g. convolutional neural networks) have proven capable of achieving high performance with little out-of-the-box modification, however require massive labeled training datasets that require both highly-specialized domain expertise and institution-specific private data access in order to label by hand [Dunnmon et al., 2018; Gulshan et al., 2016; Esteva et al., 2017; Bychkov et al., 2018]. Increasingly, this same trend has extended to other medical monitoring modalities, such as EEG and other time series signals [Acharya et al., 2018]. We briefly highlight two ways in which Snorkel has been applied to these modalities in this domain. First, we highlight several *cross-modal* use cases (as mentioned already in Section 4.3), where e.g. clinician users write labeling functions over e.g. text reports available at training time, and use the resulting labels to train a discriminative model over e.g. images which will be the only data modality available at test time. Second, we highlight use cases in which Snorkel users can write labeling functions over a complex modality such as image or video data, often using pre-computed features as building blocks over which to write labeling functions [Varma et al., 2017].

Cross-Modal Medical Triaging In many medical applications—for example, triaging of new aging or EEG studies to prioritize for human inspection—standard deep learning architectures can achieve high-performance results nearly out-of-the-box [Dunnmon et al., 2018], if sufficient hand-labeled training data is present. However, while unlabeled data is

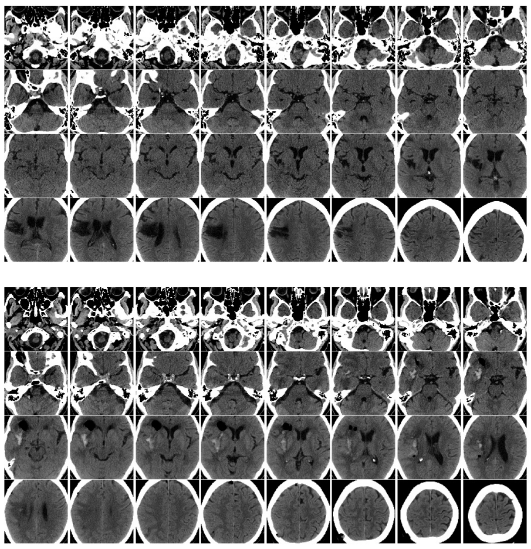
[23]`https://github.com/HazyResearch/fonduer`



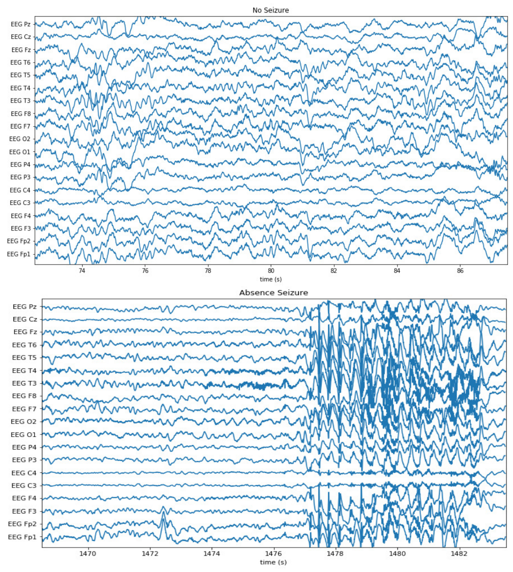
(a) **CXR:** Example normal (left) and abnormal (right) chest radiographs.



(b) **EXR:** Example normal (left) and abnormal (right) knee radiographs.

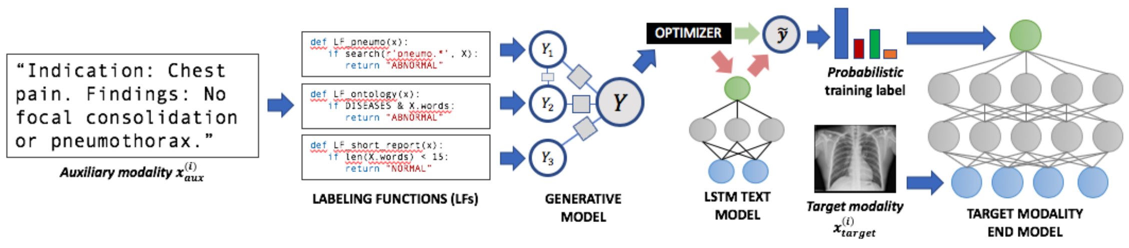


(c) **HCT:** Example HCT signals denoting no hemorrhage (top) and hemorrhage (bottom).



(d) **EEG:** Example EEG signals denoting no seizure (top) and seizure onset (bottom).

Figure 4.14: Example target modality data for the four applications surveyed, which demonstrates the breadth of applicability of the proposed cross-modal weak supervision approach; auxiliary modality data (text reports) not pictured. Panel (a) shows single 2-D chest radiographs (CXR), panel (b) shows examples of knee extremity radiographs (EXR) drawn from 2-D radiograph series, panel (c) shows 32 slices from 3-D head CT scans (HCT) with and without hemorrhage, and panel (d) shows 19-channel electroencephalography (EEG) signals with and without evidence of seizure onset. Figure from [Dunnmon et al., 2019].



```
“Indication: Chest
pain. Findings: No
focal consolidation
or pneumothorax.”
```

```
def LF_pneumo(x):
    if search(r'pneumo.*', X):
        return "ABNORMAL"
```

```
def LF_ontology(x):
    if DISEASES & X.words:
        return "ABNORMAL"
```

```
def LF_short_report(x):
    if len(X.words) < 15:
        return "NORMAL"
```

Auxiliary modality $x_{aux}^{(i)}$
LABELING FUNCTIONS (LFs)
Y_1
Y_2
Y_3
Y
GENERATIVE MODEL
OPTIMIZER
$\tilde{y}$
LSTM TEXT MODEL
Probabilistic training label
Target modality $x_{target}^{(i)}$
TARGET MODALITY END MODEL

Figure 4.15: A *cross-modal data programming* pipeline for rapidly training medical classifiers. A clinician writes labeling functions over the *auxiliary* modality, in this case a text report, which are available along with the *primary* modality, in this case a medical image, at train time. These labeling functions are combined using Snorkel's label model, and optionally used to train an LSTM model over the text report. The resulting labels are used to train a model over the target modality. At test time, the end model receives only the target modality as input, and returns predictions. Figure from [Dunnmon et al., 2019].

often available—for example, case studies in a hospital picture archiving system—labeling them according to the schema of interest is often prohibitively expensive, due to requirements of domain expertise and private patient health information (PHI) clearance. As a result, weak supervision approaches like those supported by Snorkel are an appealing option-however, it is often difficult for subject matter experts like clinicians to write labeling functions efficiently over complex medical modalities like images and time series data.

Instead, we often have access to an *auxiliary* modality at training time only—for example, unstructured text reports—which users can easily and rapidly write labeling functions over. The resulting labels can then be used to train a model over the *target* modality that will be present at test time—for example, medical images (Fig. 4.15). In a recent application of Snorkel to four medical triaging problems spanning chest (CXR) and knee extremity (EXR) radiograph triage, intracranial hemorrhage identification on head CT (HCT), and seizure onset detection on electroencephalography (EEG) (Fig. 4.14), we find that the proposed cross-modal Snorkel pipeline, using only person-days of physician and developer time, yields models that on average outperforms models trained with physician-months of hand-labeled data by 10.25 points ROC-AUC; comes within 1.75 points ROC-AUC of models trained with physician-years of hand-labeled data; and improves by an average 6 points ROC-AUC over a baseline weak supervision approach [Dunnmon et al., 2019]. Overall, we see that a weak supervision approach using Snorkel leads to a 97% average time savings—suggesting that modern weak supervision approaches such as those described in this thesis may enable significantly more rapid development and deployment of clinically-useful machine learning models.

Classification of Aortic Valve Malformations In one recent example of the second approach to weak supervision over more complex modalities—enabling users to write labeling functions directly over modalities like images and video using pre-computed features as building blocks [Varma et al., 2017]—Snorkel was used to help classify unlabeled cardiac MRI sequences for aortic valve malformations, with significant relative gains over using either a smaller hand-labeled training set or a baseline weak supervision approach [Fries et al., 2019].

4.5.3 Industrial Use Case Studies

Since introducing it as an open-source framework, Snorkel has been used in various industry settings by companies both large and small [Bach et al., 2019; Bringer et al., 2019; Mallinar et al., 2018]. We briefly highlight two of the publicly-reported use cases, which emphasize two different angles on using organizational knowledge and personnel to weakly supervise machine learning models.

Google: Snorkel DryBell In a recent paper [Bach et al., 2019] and blog post[24], we report on several internal deployments of Snorkel at Google. We focus on two aspects that generalize fa beyond Google: first, the use of *organizational knowledge*, or existing internal weak supervision sources that can be used and represented as labeling functions, and then combined using Snorkel; and second, the idea, similar to the cross-modal settings above, of using internal *non-servable* resources in the labeling functions—e.g. features too slow, private, or otherwise inaccessible to serve in production—in order to train models defined over disjoint *servable* feature sets. We show that Snorkel can lead to classifiers with comparable quality to ones trained with tens of thousands of hand-labeled examples over three content and real-time event classification applications at Google.

[24]`https://ai.googleblog.com/2019/03/harnessing-organizational-knowledge-for.html`

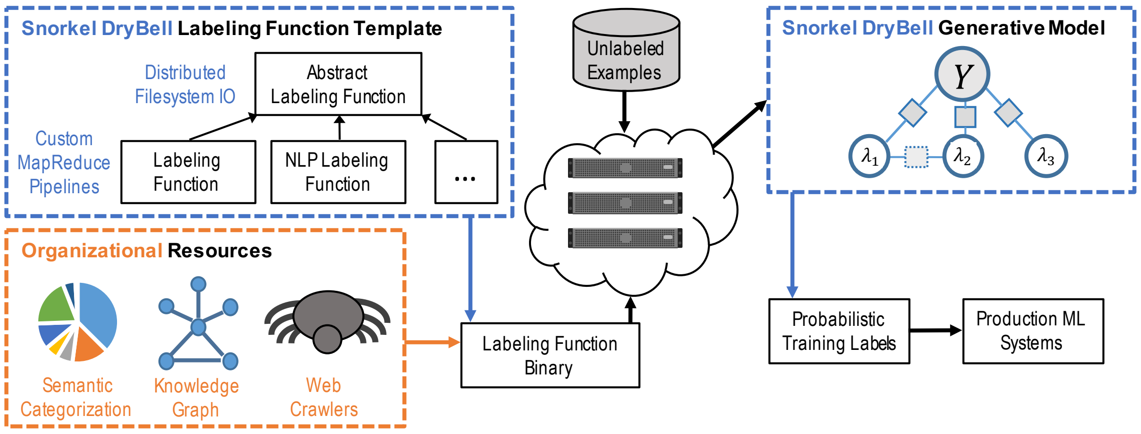


Figure 4.16: An overview of the Snorkel DryBell system. (1) Snorkel DryBell provides a library of templated C++ classes, each of which defines a MapReduce pipeline for executing a labeling function with the necessary services, such as natural language processing (NLP). (2) Engineers write methods for the MapReduce pipeline to determine a vote for each example's label, using Google resources. (3) Snorkel DryBell executes the labeling function binary on Google's distributed compute environment. (4) Snorkel DryBell loads the labeling functions' output into its generative model, which combines them into probabilistic training labels for use by production systems. Figure from [Bach et al., 2019].

Intel: Snorkel Osprey In a collaboration with Intel, we reported on Snorkel Osprey, an extension to Snorkel focused on supporting non-programmers via configurable templates that decouple business logic from code and machine learning [Bringer et al., 2019]. In Osprey, members of Intel's Sales & Marketing Group were able to use machine learning for three event-monitoring applications without programming, by entering high-level information into a declarative spreadsheet-based interface, leading to average gains of 18.5 points in precision and 28.5 points of recall at a fraction of the cost, compared to prior hand-labeled and weak supervision approaches taken.

4.6 Related Work

This section is an overview of techniques for managing weak supervision, many of which are subsumed in Snorkel. We also contrast it with related forms of supervision.

Combining Weak Supervision Sources The main challenge of weak supervision is how to combine multiple sources. For example, if a user provides two knowledge bases for distant supervision, how should a data point that matches only one knowledge base be labeled? Some researchers have used multi-instance learning to reduce the noise in weak supervision sources [Riedel et al., 2010; Hoffmann et al., 2011], essentially modeling the different weak supervision sources as soft constraints on the true label, but this approach is limited because it requires using a specific end model that supports multi-instance learning.

Researchers have therefore considered how to estimate the accuracy of label sources without a gold standard with which to compare—a classic problem [Dawid and Skene, 1979]—and combine these estimates into labels that can be used to train an arbitrary end model. Much of this work has focused on crowdsourcing, in which workers have unknown accuracy [Dalvi et al., 2013; Joglekar et al., 2015; Zhang et al., 2016b]. Such methods use generative probabilistic models to estimate a latent variable—the true class label—based on noisy observations. Other methods use generative models with hand-specified dependency structures to label data for specific modalities, such as topic models for text [Alfonseca et al., 2012] or denoising distant supervision sources [Takamatsu et al., 2012; Roth and Klakow, 2013b]. Other techniques for estimating latent class labels given noisy observations include spectral methods [Parisi et al., 2014]. Snorkel is distinguished from these approaches because its generative model supports a wide range of weak supervision sources, and it learns the accuracies and correlation structure among weak supervision sources without ground truth data.

Other Forms of Supervision Work on *semi-supervised learning* considers settings with some labeled data and a much larger set of unlabeled data, and then leverages various domain- and task-agnostic assumptions about smoothness, low-dimensional structure, or distance metrics to heuristically label the unlabeled data [Chapelle et al., 2009]. Work on *active learning* aims to automatically estimate which data points are optimal to label, thereby hopefully reducing the total number of examples that need to be manually annotated [Settles, 2012]. *Transfer learning* considers the strategy of repurposing models trained on different datasets or tasks where labeled training data is more abundant [Pan and Yang, 2010]. Another type of supervision is self-training [Scudder, 1965; Agrawala, 1970] and co-training [Blum and Mitchell, 1998], which involves training a model or pair of models on data that they labeled themselves. Weak supervision is distinct in that the goal is to solicit input directly from SMEs, however at a higher level of abstraction and/or in an inherently noisier form. Snorkel is focused on managing weak supervision sources, but combing its methods with these other types of supervision is straightforward.

Related Data Management Problems Researchers have considered related problems in data management, such as data fusion [Dong and Srivastava, 2015; Rekatsinas et al., 2017b] and truth discovery [Li et al., 2015]. In these settings, the task is to estimate the reliability of data sources that provide assertions of facts and determine which facts are likely true. Many approaches to these problems use probabilistic graphical models that are related to Snorkel's generative model in that they represent the unobserved truth as a latent variable, e.g., the latent truth model [Zhao et al., 2012]. Our setting differs in that labeling functions assign labels to user-provided data, and they may provide any label or abstain, which we must model. Work on data fusion has also explored how to model user-specified correlations among data sources [Pochampally et al., 2014]. Snorkel automatically identifies which correlations among labeling functions to model.

Chapter 5

Multi-Task Weak Supervision

In Chapter 4, we introduced Snorkel, a system for enabling users to programmatically label and manage training datasets, built around the data programming paradigm introduced in Chapter 3. However, in many real-world settings, users increasingly have not just one but multiple, often related, classification tasks that they would like to apply machine learning to. In certain large technology companies, the number of modeling tasks already reaches into the hundreds, and given the increasing ubiquity of machine learning, many other organizations are likely to follow suit. In this chapter, motivated by this trend, we extend the Snorkel system and data programming approach to the *multi-task* setting, where a user has multiple, potentially related tasks and would like to realize efficiencies by reasoning jointly across them. Concretely, we extend data programming to handle multiple tasks related by an optional user-provided *task graph*, and create a new multi-task system, Snorkel MeTaL[1], for enabling users to easily build and train multi-task learning models. We show empirically, using several hierarchical multi-task text classification problems, that using this approach and system leads to average improvements of 20.2 points in accuracy over a traditional supervised approach, 6.8 points over a weak supervision baseline, and 4.1 points over single-task data programming and Snorkel. We use this to further demonstrate that especially in complex, multi-task settings, programmatically building, managing, and modeling training datasets can be a powerful and effective interface to modern machine learning tools.

[1]Merged into Snorkel (`https://snorkel.org`) as of version 0.9.

Motivation As mentioned above, the motivation for Snorkel MeTaL stems from the increasing prevalence of users with multiple related classification tasks, and in turn, of multi-task approaches to solve them. The high level idea of modeling multiple tasks jointly in an attempt to realize sample complexity efficiencies and learn more robust representations, generally referred to as *multi-task learning* [Caruana, 1993], has of late gathered renewed popularity in the setting of modern deep learning architectures. However, while these multi-task models in theory reduce the overall number of labeled data points needed per task to achieve a given quality level, they still in general need large labeled training sets- and now, for not one but *several* tasks. Thus, we return again to the same challenge of requiring large labeled training sets–this time in the more complex multi-task setting.

Snorkel MeTaL To overcome this challenge, we propose Snorkel MeTaL, a framework for modeling and integrating weak supervision sources–represented by labeling functions–with different unknown accuracies, correlations, and pertaining to different possibly related tasks. Of these three challenges, the first two were approached in Chapters 3 and 4; however, we now describe an approach that additionally handles the third one of multi-task labeling functions. In Snorkel MeTaL, we view each labeling function as labeling one of several related sub-tasks of a problem—we refer to this as the *multi-task weak supervision* setting. We then show that given the dependency structure of the labeling functions, we extend the approach presented in Chapter 3.3 to use their observed agreement and disagreement rates to recover their unknown accuracies. Moreover, we exploit the relationship structure between tasks to observe additional cross-task agreements and disagreements, effectively providing extra signal from which to learn. We extend the matrix completion-style algorithm in Section 3.3 to learn and model the accuracies of diverse multi-task supervision sources, and then combine their labels to produce training data that can be used to supervise arbitrary models, including increasingly popular multi-task learning models [Caruana, 1993; Ruder, 2017]. Compared to the approaches in Chapters 3 and 4, however, which only handled the single-task setting, we demonstrate that our multi-task aware approach leads to average gains of 4.1 points in accuracy in our experiments.

We validate our framework on three fine-grained classification tasks in named entity recognition, relation extraction, and medical document classification, for which we have

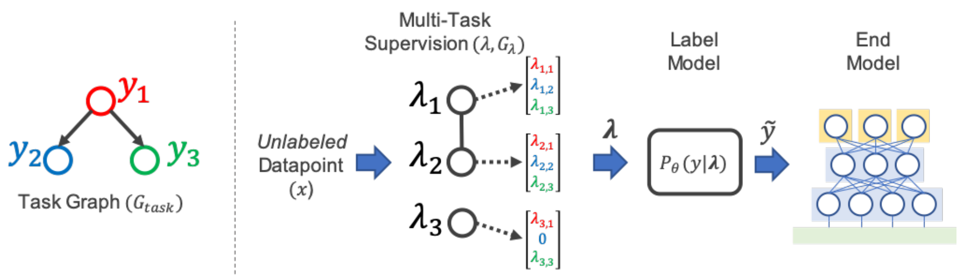

Task Graph (G_{task})
Unlabeled Datapoint (x)
Multi-Task Supervision (λ, G_λ)
Label Model
$P_\theta(y|\lambda)$
End Model

Figure 5.1: A schematic of the Snorkel MeTaL pipeline. To generate training data for an *end model*, such as a multi-task model as in our experiments, the user inputs a *task graph* G_{task} defining the relationships between *task labels* $y_1, ..., y_t$; a set of *unlabeled* data points X_U; a set of *multi-task labeling functions* which each output a vector λ_j of task labels for a data point $x \in X_U$; and the dependency structure between these labeling functions, G_λ. We train a *label model* to learn the accuracies of the labeling functions, outputting a vector of probabilistic training labels $\tilde{y}$ for training the end model.

diverse weak supervision sources at multiple levels of granularity, represented as multi-task labeling functions. We show that by modeling them as labeling hierarchically-related sub-tasks and utilizing unlabeled data, we can get an average improvement of 20.2 points in accuracy over a traditional supervised approach, 6.8 points over a basic majority voting weak supervision baseline, and 4.1 points over single-task data programming. From a practical standpoint, we argue that our framework represents an efficient way for practitioners to supervise modern machine learning models, including new multi-task variants, for complex tasks by opportunistically using the diverse weak supervision sources available to them.

Outline of Chapter In this chapter we describe Snorkel MeTaL, an extension of the Snorkel system and data programming method introduced in Chapters 3 and 4 to the multi-task setting:

- *In Section 5.1,* we start by describing the architecture and syntax of multi-task weak supervision in Snorkel MeTaL.
- *In Section 5.2,* we describe extending data programming to the multi-task setting, highlighting the sub-case of hierarchically-related tasks.
- *In Section 5.3,* we briefly describe the system architecture of Snorkel MeTaL.

- *Finally, in Section 5.4* we present empirical validation of Snorkel MeTaL.

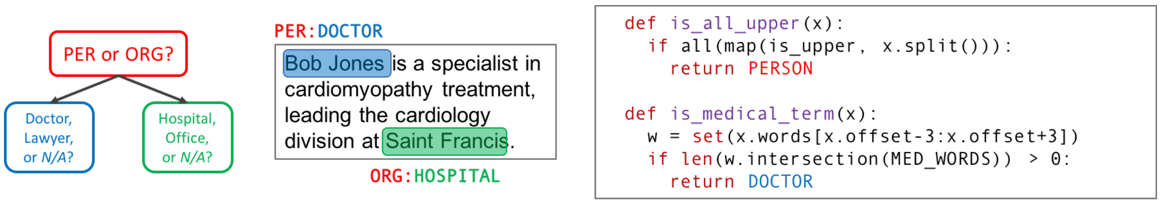
We note that Snorkel MeTaL was made available as an open source software package, although its functionality has since been merged into the Snorkel (`snorkel.org`) repository.

5.1 Using Multi-Task Weak Supervision

As modern machine learning models become both more complex and more performant on a range of tasks, developers increasingly interact with them by programmatically generating noisier or *weak* supervision. In Chapter 4 we described *data programming*, an approach for effectively *programming* machine learning models using the following pipeline: First, users provide one or more weak supervision sources as *labeling functions*, which are applied to unlabeled data to generate a set of noisy labels. These labels may overlap and conflict; we model and combine them via a *label model* in order to produce a final set of training labels. These labels are then used to train some discriminative model, which we refer to as the *end model*. This programmatic weak supervision approach can utilize sources ranging from heuristic rules to other models, and in this way can also be viewed as a pragmatic and flexible form of multi-source *transfer learning*.

In this chapter, we focus on one motivating type of multi-task setting where there are several hierarchically-related tasks, which we often refer to as having different levels of *granularity*. Importantly, we note that Snorkel MeTaL can be relevant even if a user only ultimately cares about producing a classifier for a single final task. In fact, we find in practice that users often want to be able to leverage diverse sources of supervision pertaining to multiple tasks (often referred to as *auxiliary* tasks in the multi-task learning literature), but ultimately only care about producing a classifier for one *primary* task. We consider an example:

Example 5.1.1. A developer wants to train a fine-grained Named Entity Recognition (NER)


PER or ORG?
Doctor, Lawyer, or N/A?
Hospital, Office, or N/A?
PER:DOCTOR
Bob Jones is a specialist in cardiomyopathy treatment, leading the cardiology division at Saint Francis.
ORG:HOSPITAL

```
def is_all_upper(x):
  if all(map(is_upper, x.split())):
    return PERSON

def is_medical_term(x):
  w = set(x.words[x.offset-3:x.offset+3])
  if len(w.intersection(MED_WORDS)) > 0:
    return DOCTOR
```

Figure 5.2: An example fine-grained entity classification problem, where labeling functions label three sub-tasks of different granularities: (i) `Person` vs. `Organization`, (ii) `Doctor` vs. `Lawyer` (or *N/A*), (iii) `Hospital` vs. `Office` (or *N/A*). The example weak supervision sources, expressed as labeling functions in Python, use a pattern heuristic and dictionary lookup respectively.

model to classify mentions of entities in the news (Figure 5.2). She has a multitude of available weak supervision sources which she believes have relevant signal for her problem—for example, pattern matchers, dictionaries, and pre-trained generic NER taggers. However, it is unclear how to properly use and combine them: some of them label phrases coarsely as `PERSON` versus `ORGANIZATION`, while others classify specific fine-grained types of people or organizations, with a range of unknown accuracies. In our framework, she can represent them as labeling tasks of different granularities—e.g. $y_1 = \{\texttt{Person}, \texttt{Org}\}$, $y_2 = \{\texttt{Doctor}, \texttt{Lawyer}, \texttt{N/A}\}$, $y_3 = \{\texttt{Hospital}, \texttt{Office}, \texttt{N/A}\}$, where the label `N/A` applies, for example, when the type-of-person task is applied to an organization.

In our proposed multi-task supervision setting, the user specifies a set of structurally-related *tasks*, and then provides a set of multi-task labeling functions which are user-defined functions that either label each data point or abstain for each task, and may have some user-specified dependency structure. These labeling functions can be arbitrary black-box functions, and can thus subsume a range of weak supervision approaches relevant to both text and other data modalities, including use of pattern-based heuristics, distant supervision [Mintz et al., 2009], crowd labels, other weak or biased classifiers, declarative rules over unsupervised feature extractors [Varma et al., 2017], and more. Our goal is to estimate the unknown accuracies of these labeling functions, combine their outputs, and use the resulting labels to train an end model.

5.2 Modeling Multi-Task Weak Supervision

The core technical challenge of the *multi-task weak supervision* setting is recovering the unknown *accuracies* of labeling functions given their dependency structure and a schema of the tasks they label, but without any ground-truth labeled data. We now describe how the matrix completion-style data programming algorithm from Section 3.3 can be extended for recovering the accuracies in this multi-task setting.

Problem Setup Let $x \in \mathcal{X}$ be a data point and $y = [y_1, y_2, \dots, y_t]^T$ be a vector of categorical *task labels*, $y_i \in \{1, \dots, k_i\}$, corresponding to t tasks, where (x, y) is drawn i.i.d. from a distribution $\mathcal{D}$.

The user provides a specification of how these tasks relate to each other; we denote this schema as the *task structure* G_{task}. The task structure expresses logical relationships between tasks, defining a *feasible set* of label vectors $\mathcal{Y}$, such that $y \in \mathcal{Y}$. For example, Figure 5.2 illustrates a hierarchical task structure over three tasks of different granularities pertaining to a fine-grained entity classification problem. Here, the tasks are related by logical subsumption relationships: for example, if $y_2 = \texttt{DOCTOR}$, this implies that $y_1 = \texttt{PERSON}$, and that $y_3 = \textit{N/A}$, since the task label y_3 concerns types of organizations, which is inapplicable to persons. Thus, in this task structure, $y = [\texttt{PERSON}, \texttt{DOCTOR}, \textit{N/A}]^T$ is in $\mathcal{Y}$ while $y = [\texttt{PERSON}, \textit{N/A}, \texttt{HOSPITAL}]^T$ is not. While task structures are often simple to define, as in the previous example, or are explicitly defined by existing resources—such as ontologies or graphs—we note that if no task structure is provided, our approach becomes equivalent to modeling the t tasks separately, a baseline we consider in the experiments.

In our setting, rather than observing the true label y, we have access to m *multi-task labeling functions* which emit label vectors λ_j that contain labels for some subset of the t tasks. Let $\emptyset$ denote a null or abstaining label, and let the *coverage set* $c_j \subseteq \{1, \dots, t\}$ be the fixed set of tasks for which the jth labeling function emits non-zero labels, such that $\lambda_j \in \mathcal{Y}_{c_j}$. For convenience, we let $c_0 = \{1, \dots, t\}$ so that $\mathcal{Y}_{c_0} = \mathcal{Y}$. For example, a labeling function from our previous example might have a coverage set $c_j = \{1, 3\}$, emitting coarse-grained labels such as $\lambda_j = [\texttt{PERSON}, 0, \textit{N/A}]^T$. Note that labeling functions often label multiple tasks implicitly due to the constraints of the task structure; for example, a labeling

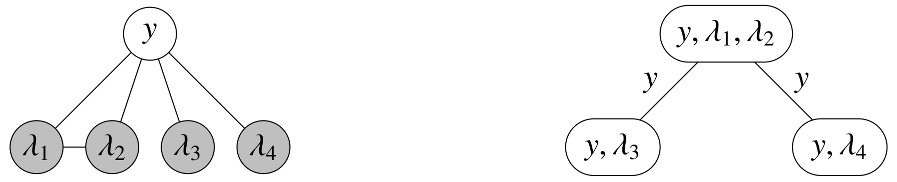


Figure 5.3: An example of a labeling function dependency graph G_λ (left) and its junction tree representation (right), where y is a vector-valued random variable with a feasible set of values, $y \in \mathcal{Y}$. Here, the output of labeling functions 1 and 2 are modeled as dependent conditioned on y. This results in a junction tree with singleton separator sets, y. Here, the observable cliques are $O = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4, \{\lambda_1, \lambda_2\}\} \subset C$.

function that labels types of people (y_2) also implicitly labels people vs. organizations (y_1 = `PERSON`), and types of organizations (as y_3 = `N/A`). Thus labeling functions tailored to different tasks still have agreements and disagreements; we use this additional *cross-task* signal in our approach.

The user also provides the conditional dependency structure of the labeling functions as a graph $G_\lambda = (V, E)$, where $V = \{y, \lambda_1, \lambda_2, \ldots, \lambda_m\}$ (Figure 5.3). Specifically, if (λ_i, λ_j) is not an edge in G_λ, this means that λ_i is independent of λ_j conditioned on y and the other labeling function labels. Note that if G_λ is unknown, it can be estimated using statistical techniques such as [Bach et al., 2017]. Importantly, we do not know anything about the strengths of the correlations in G_λ, or the labeling functions' accuracies.

Our overall goal is to apply the set of labeling functions to an unlabeled dataset X_U consisting of n data points, then use the resulting weakly-labeled training set to supervise an *end model* $h_w : \mathcal{X} \mapsto \mathcal{Y}$ (Figure 5.1). This weakly-labeled training set will contain overlapping and conflicting labels, from labeling functions with unknown accuracies and correlations. To handle this, we will learn a *label model* $p_\theta(y|\lambda)$, parameterized by a vector of labeling function correlations and accuracies θ, which for each data point x takes as input the noisy labels $\lambda = \{\lambda_1, \ldots, \lambda_m\}$ and outputs a single probabilistic label vector $\tilde{y}$. Succinctly, given a user-provided tuple $(X_U, \lambda, G_\lambda, G_{\text{task}})$, our key technical challenge is recovering the parameters θ without access to ground truth labels y.

Modeling Multi-Task Sources To learn a label model over multi-task labeling functions, we introduce sufficient statistics over the random variables in G_λ, as in Section 3.3. To recall: let $\mathcal{C}$ be the set of cliques in G_λ, and define an indicator random variable for the event of a clique $C \in \mathcal{C}$ taking on a set of values y_C:

$$\psi(C, y_C) = \mathbb{1}\left\{\cap_{i \in C} V_i = (y_C)_i\right\},$$

where $(y_C)_i \in \mathcal{Y}_{C_i}$. We define $\psi(C) \in \{0,1\}^{\prod_{i \in C}(|\mathcal{Y}_{C_i}|-1)}$ as the vector of indicator random variables for all combinations of all but one of the labels emitted by each variable in clique C—thereby defining a minimal set of statistics—and define $\psi(\mathbf{C})$ accordingly for any set of cliques $\mathbf{C} \subseteq \mathcal{C}$. Then $\theta = \mathbb{E}\left[\psi(\bar{\mathcal{C}})\right]$ is the vector of sufficient statistics for the label model we want to learn.

We work with two simplifying conditions in this section, as described in Section 3.3.2 (where the binary, single-task setting was primarily considered). First, we consider the setting where G_λ is *triangulated* and has a junction tree representation with singleton separator sets. If this is not the case, edges can always be added to G_λ to make this setting hold; otherwise, we describe how our approach can directly handle non-singleton separator sets in Section 3.3.

Second, we use a simplified *class-conditional* model of the noisy labeling process, as discussed in Section 3.3, where we learn one accuracy parameter for each label value λ_j that each labeling function emits. This is equivalent to assuming that a labeling function may have a different accuracy on each different class, but that if it emits a certain label incorrectly, it does so uniformly over the different true labels y. This is a more expressive model than the commonly considered one, where each labeling function is modeled by a single accuracy parameter, e.g. in [Dawid and Skene, 1979; Ratner et al., 2016].

Our Approach Given the above setup, we can now apply the matrix completion-style approach as detailed in Section 3.3.2. We proceed as before, now using our multi-task encoding of the problem (Algorithm 4). In this setting, we also use the function ExpandTied, which is a simple algebraic expansion of tied parameters according to the simplified class-conditional model used in this section.

Algorithm 4 Labeling Function Accuracy Estimation for Multi-Task Weak Supervision

```
Input: Observed labels $\hat{\mathbb{E}}[\psi(O)]$, covariance $\hat{\Sigma}_O$, and correlation sparsity structure $\Omega$

CheckIdentifiability($\Omega$)                                    ▷ Preliminary operations
$\hat{\mathbb{E}}[\psi(y)] \leftarrow$ ClassBalance($\hat{\mathbb{E}}[\psi(O)], \hat{\Sigma}_O, \Omega$)

$\hat{z} \leftarrow \operatorname{argmin}_z \left\|\hat{\Sigma}_O^{-1} + zz^T\right\|_\Omega$              ▷ Solve the masked matrix completion problem

$\hat{c} \leftarrow \Sigma_S^{-1}(1 + \hat{z}^T\hat{\Sigma}_O\hat{z})$              ▷ Recover the estimated label model parameters, $\hat{\theta}$
$\hat{\Sigma}_{OS} \leftarrow \hat{\Sigma}_O\hat{z}/\sqrt{\hat{c}}$
$\hat{\theta}' \leftarrow \hat{\Sigma}_{OS} + \hat{\mathbb{E}}[\psi(y)]\,\hat{\mathbb{E}}[\psi(O)]$
return ExpandTied($\hat{\theta}'$)
```

Hierarchical Multi-Task Supervision As an illustrative example, we now consider the specific case of *hierarchical* multi-task supervision, which can be thought of as consisting of coarser- and finer-grained labels, or alternatively higher- and lower-level labels, and provides a way to supervise e.g. fine-grained classification tasks at multiple levels of granularity. Specifically, consider a task label vector $y = [y_1, \ldots, y_t]^T$ as before, this time with $y_s \in \{\texttt{N/A}, 1, \ldots, k_s\}$, where we will explain the meaning of the special value `N/A` shortly. We then assume that the tasks y_s are related by a *task hierarchy* which is a hierarchy $G_{\text{task}} = (V, E)$ with vertex set $V = \{y_1, y_2, \ldots, y_t\}$ and directed edge set E. The task structure reflects constraints imposed by higher level (more general) tasks on lower level (more specific) tasks. The following example illustrates a simple tree task structure:

Example 5.2.1. Let y_1 classify a data point x as either a `PERSON` ($y_1 = 1$) or `BUILDING` ($y_1 = 2$). If $y_1 = 1$, indicating that x represents a `PERSON`, then y_2 can further label x as a `DOCTOR` or `NON-DOCTOR`. y_3 is used to distinguish between `HOSPITAL` and `NON-HOSPITAL` in the case that $y_1 = 2$. The corresponding graph G_{task} is shown in Figure 5.4. If $y_1 = 2$, then task y_2 is not applicable, since y_2 is only suitable for persons; in this case, y_2 takes the value `N/A`. In this way the task hierarchy defines a feasible set of task vector values: $y = [1, 1, \texttt{N/A}]^T, [1, 2, \texttt{N/A}]^T, [2, \texttt{N/A}, 1]^T, [2, \texttt{N/A}, 2]^T$ are valid, while e.g. $y = [1, 1, 2]^T$ is not.

As in the example, for certain configurations of y's, the parent tasks logically constrain the one or more of the children tasks to be irrelevant, or rather, to have inapplicable label

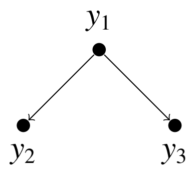


Figure 5.4: Example task hierarchy G_{task} for a three-task classification problem. Task y_1 classifies a data point x as a `PERSON` or `BUILDING`. If y_1 classifies x as a `PERSON`, y_2 is used to distinguish between `DOCTOR` and `NON-DOCTOR`. Similarly, if y_2 classifies x as a `BUILDING`, y_3 is used to distinguish between `HOSPITAL` and `NON-HOSPITAL`. Tasks y_2, y_3 are more specific, or *finer-grained* tasks, constrained by their parent task y_1.

values. In this case, the task takes on the value `N/A`. In Example 5.2.1, we have that if $y_1 = 1$, representing a building, then y_2 is inactive (since X corresponds to a building). We define the symbol `N/A` (for incompatible) for this scenario. More concretely, let $\mathcal{N}(y_i) = \{y_j : (y_j, y_i) \in E\}$ be the in-neighborhood of y_i. Then, the values of the members of $\mathcal{N}(y_i)$ determine whether $y_i = \texttt{N/A}$, i.e., $\mathbb{1}\left\{y_j = \texttt{N/A}\right\}$ is deterministic conditioned on $\mathcal{N}(y_i)$.

Hierarchical Multi-Task Sources Observe that in the mutually-exclusive task hierarchy just described, the value of a descendant task label y_d determines the values of all other task labels in the hierarchy besides its descendants. For example, in Example 5.2.1, a label $y_2 = 1 \implies (y_1 = 1, y_3 = \texttt{N/A})$; in other words, knowing that x is a `DOCTOR` also implies that x is a `PERSON` and not a `BUILDING`.

For a labeling function λ_j with coverage set c_j, the label it gives to the lowest task in the task hierarchy which is non-zero and non-`N/A` determines the entire label vector output by λ_j. E.g. if the lowest task that λ_j labels in the hierarchy is $y_1 = 1$, then this implies that it outputs vector $[1, 0, \texttt{N/A}]^T$. Thus, in this sense, we can think of each labeling functions λ_j as labeling one specific task in the hierarchy, and thus can talk about coarser- and finer-grained labeling functions.

Reduced-Rank Form: Modeling Local Accuracies In some cases, we can make slightly different modeling assumptions that reflect the nature of the task structure, and additionally can result in reduced-rank forms of our model. In particular, for the hierarchical setting introduced here, we can divide the statistics θ into *local* and *global* subsets, and for example focus on modeling only the *local* ones to once again reduce to rank-one form.

To motivate with our running example: a finer-grained labeling function that labels `DOCTOR` versus `NON-DOCTOR` probably is not accurate on the building type subtask; we can model this labeling function using one accuracy parameter for the former label set (the *local* accuracy) and a different (or no parameter) for the *global* accuracy on irrelevant tasks. More specifically, for cliques involving λ_j, we can model $p_\theta(\lambda_j, y)$ for all y with only non-$\mathtt{N/A}$ values in the coverage set of λ_j using a single parameter, and call this the *local* accuracy; and we can either model θ for the other y using one or more other parameters, or simply set it to a fixed value and not model it, to reduce to rank one form, as we do in the experiments. In particular, this allows us to capture our observation in practice that if a developer is writing a labeling function to distinguish between labels at one sub-tree, they are probably not designing or testing it to be accurate on any of the other subtrees.

5.3 Snorkel MeTaL: A System for Weak Supervision

To help validate the utility of the proposed multi-task weak supervision approach, we designed and built an open source framework, Snorkel MeTaL, extending Snorkel to the multi-task setting[2] Snorkel MeTaL implements the core functionality of the multi-task weak supervision pipeline outlined in the preceding sections (see Figure 5.1), provides basic multi-task schema definition and data management classes, and defines a new auto-compiled multi-task learning model architecture.

In Snorkel MeTaL, the user first provides a task graph to (optionally) define the relation structure of the task labels; this task graph is then used to automatically define the structure of an end multi-task deep learning model compiled in PyTorch, using the following three configurable building blocks:

- *Input Module:* To support multiple types of input data, Snorkel MeTaL's end model accepts a plug-in input module of arbitrary complexity, with parameters either pre-trained or jointly learned at test-time, which maps from a raw data point to a vector

[2]`https://github.com/HazyResearch/metal`; note that as of the date of this thesis's publication, the core functionality of Snorkel MeTaL has been merged into Snorkel (`snorkel.org`)

	NER	RE	Doc	Average
Gold (Dev)	63.7 ± 2.1	28.4 ± 2.3	62.7 ± 4.5	51.6
MV	76.9 ± 2.6	43.9 ± 2.6	74.2 ± 1.2	65.0
DP [Ratner et al., 2016]	78.4 ± 1.2	49.0 ± 2.7	75.8 ± 0.9	67.7
Snorkel MeTaL	**82.2** ± 0.8	**56.7** ± 2.1	**76.6** ± 0.4	**71.8**

Table 5.1: **Performance Comparison of Different Supervision Approaches.** We compare the micro accuracy (avg. over 10 trials) with 95% confidence intervals on the primary (finest-grained) task of an end multi-task model trained using the training labels from the hand-labeled development set (Gold Dev), hierarchical majority vote (MV), data programming (DP), and our approach (Snorkel MeTaL).

of pre-specified dimension. Snorkel MeTaL includes pre-configured input modules for modalities like text and image data.

- *Intermediate Module:* MeTaL then constructs a hierarchy of several intermediate modules—linear layers by default, but easily replaced with more complex modules.
- *Task Heads:* Finally, as in many standard MTL network designs, each task has a separate linear layer attached to the shared layers.

In the initial prototype of Snorkel MeTaL, we provide extra support for hierarchical task graphs as a special case by optionally attaching task heads to the intermediate layer corresponding to their level in the hierarchy, and optionally also pass predictions between task heads according to this graph structure; for details see [Ratner et al., 2018].

5.4 Experiments

We validate our approach on three fine-grained, multi-task classification problems—entity classification, relation classification, and document classification—where weak supervision sources are available at both coarser and finer-grained levels (e.g. as in Figure 5.2). We evaluate the predictive accuracy on the primary (finest-grained) tasks of end models supervised with training data produced by several approaches, finding that our approach outperforms traditional hand-labeled supervision by 20.2 points, a baseline majority vote weak supervision approach by 6.8 points, and the approach presented in Chapter 3 that is not multi-task-aware by 4.1 points. For performance on all tasks, see [Ratner et al., 2019b].

Datasets Each dataset consists of a large (3k-63k) amount of unlabeled training data and a small (200-350) amount of labeled data which we refer to as the *development set*, which we use for (a) a traditional supervision baseline, and (b) for hyperparameter tuning of the end model (see [Ratner et al., 2019b] for additional details). The average number of labeling functions per task was 13, with sources expressed as Python functions, averaging 4 lines of code and comprising a mix of pattern matching heuristics, external knowledge base or dictionary lookups, and pre-trained models. In all three cases, we primarily evaluate the performance on the finest-grained tasks (i.e. the union of the leaf level tasks); for performance on all tasks, see [Ratner et al., 2019b].

Named Entity Recognition (NER): We represent a fine-grained named entity recognition problem—tagging entity mentions in text documents—as a hierarchy of three sub-tasks over the OntoNotes dataset [Weischedel et al., 2011]: $y_1 \in \{\text{Person}, \text{Organization}\}$, $y_2 \in \{\text{Businessperson}, \text{Other Person}, N/A\}$, $y_3 \in \{\text{Company}, \text{Other Org}, N/A\}$, where again we use N/A to represent "not applicable". We evaluate the primary task of classifying the finest-grained labels, i.e. the union of y_2 and y_3.

Relation Extraction (RE): We represent a relation extraction problem—classifying entity-entity relation mentions in text documents—as a hierarchy of six sub-tasks which either concern labeling the subject, object, or subject-object pair of a possible or *candidate* relation in the TACRED dataset [Zhang et al., 2017b]. For example, we might label a relation as having a Person subject, Location object, and Place-of-Residence relation type. We evaluate the primary task of classifying the finest-grained labels, i.e. the relation types.

Medical Document Classification (Doc): We represent a radiology report triaging (i.e. document classification) problem from the OpenI dataset [National Institutes of Health, 2017] as a hierarchy of three sub-tasks: $y_1 \in \{\text{Acute}, \text{Non-Acute}\}$, $y_2 \in \{\text{Urgent}, \text{Emergent}, N/A\}$, $y_3 \in \{\text{Normal}, \text{Non-Urgent}, N/A\}$. We evaluate the primary task of classifying the finest-grain labels, i.e. the union of the leaf-level tasks y_2 and y_3.

End Model Protocol Our goal was to test the performance of a basic multi-task end model using training labels produced by various different approaches. We use an architecture consisting of a shared bidirectional LSTM input layer with pre-trained embeddings, shared linear intermediate layers, and a separate final linear layer ("task head") for each task. Hyperparameters were selected with an initial search for each application, then fixed.

Core Validation We compare the accuracy of the end multi-task model trained with labels from our approach versus those from three baseline approaches (Table 5.1):

- *Traditional Supervision* **[Gold (Dev)]**: We train the end model using the small hand-labeled development set.
- *Hierarchical Majority Vote* **[MV]**: We use a hierarchical majority vote of the labeling function labels: i.e. for each data point, for each task we take the majority vote and proceed down the task tree accordingly. This procedure can be thought of as a hard decision tree, or a cascade of if-then statements as in a rule-based approach.
- *Data Programming* **[DP]**: We model each task separately using the data programming approach for denoising weak supervision (Chapter 3).

In all settings, we used the same end model architecture as described above. Note that while we choose to model these problems as consisting of multiple sub-tasks, we evaluate with respect to the broad primary task of fine-grained classification (for subtask-specific scores, see [Ratner et al., 2019b]). We observe in Table 5.1 that our approach of leveraging multi-granularity weak supervision leads to large gains—20.2 points over traditional supervision with the development set, 6.8 points over hierarchical majority vote, and 4.1 points over data programming.

Ablations We examine individual factors:

Joint Task Modeling: Next, we use our algorithm to estimate the accuracies of sources for each task separately, to observe the empirical impact of modeling the multi-task setting jointly as proposed. We see average gains of 1.3 points in accuracy (see Appendix).

End Model Generalization: Though not possible in many settings, in our experiments we can directly apply the label model to make predictions. In Table 5.6, we show that

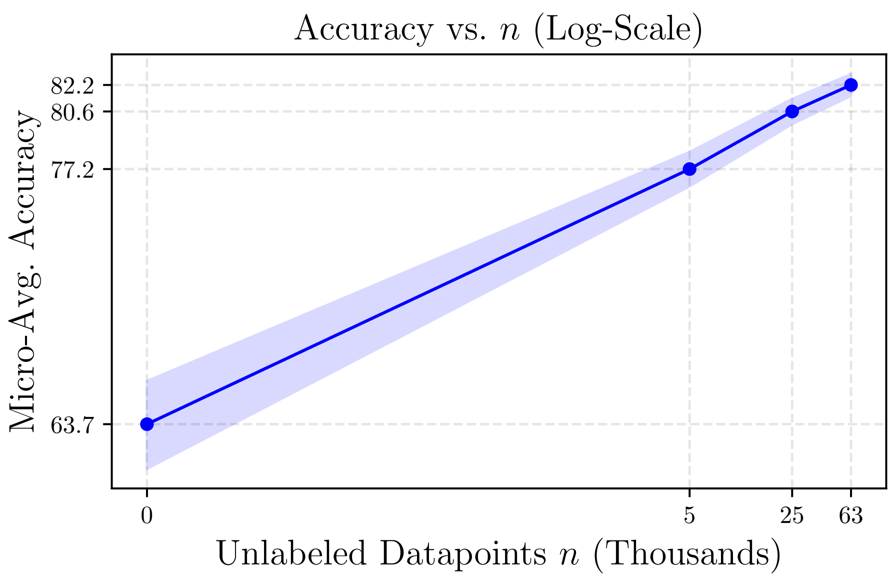


Figure 5.5: In the OntoNotes dataset, end model accuracy scales with the amount of available *unlabeled* data.

	# Train	LM	EM	*Gain*
NER	62,547	75.2	82.2	*7.0*
RE	9,090	55.3	57.4	*2.1*
Doc	2,630	75.6	76.6	*1.0*

Figure 5.6: Using the label model (LM) predictions directly versus using an end model trained on them (EM).

the end model improves performance by an average 3.4 points in accuracy, validating that the models trained do indeed learn to generalize beyond the provided weak supervision. Moreover, the largest generalization gain of 7 points in accuracy came from the dataset with the most available unlabeled data (n=63k), demonstrating scaling consistent with the predictions of our theory (Fig. 5.5). This ability to leverage additional unlabeled data and more sophisticated end models are key advantages of the weak supervision approach in practice.

Chapter 6

Data Augmentation

In Chapters 3, 4, and 5, we described methods and systems for enabling users to programmatically *label* unlabeled data to create large labeled training datasets for machine learning. However, labeling is just one of the common and critical operations of building and managing training datasets, and only one way of injecting weak supervision into the machine learning pipeline.

In this chapter, we present an approach and system for supporting another critical operation in building training datasets, *data augmentation*, in which labeled training datasets are expanded or augmented by transforming data points in class label-preserving ways; for example, the canonical example is randomly rotating images. We propose a formalization of this critical but generally ad hoc process in which users again provide simple, black-box function–in this setting, *transformation functions* that incrementally transform a labeled training point–which we then automatically model and combine using a different generative modeling approach which again leverages unlabeled data. We implement a system for data augmentation around our approach, TANDA[1], which is packaged as an open source software system that interfaces with TensorFlow and other machine learning frameworks.

In practice we find that our approach enables users to more easily develop and apply data augmentation strategies across multiple data modalities, and empirically, we find that given a fixed set of user-developed transformation functions, our approach for automatically tuning and composing them leads to an average 2.9 points of accuracy gain across

[1]`https://github.com/HazyResearch/tanda`

three competitive tasks, as compared to a standard heuristic baseline. We view this approach as another, complementary way for users to practically develop machine learning applications by programmatically building, managing, and modeling training datasets. To this end, the data augmentation approach presented above is also included as a core operation in the open source Snorkel software package[2].

Motivation Modern machine learning models, such as deep neural networks, may have billions of free parameters and accordingly require massive labeled data sets for training. In most settings, labeled data is not available in sufficient quantities to avoid overfitting to the training set. The technique of artificially expanding labeled training sets by transforming data points in ways which preserve class labels – known as *data augmentation* – is one critical and effective tool for combatting this labeled data scarcity problem. Data augmentation can be seen as a form of *weak supervision*, providing a way for practitioners to leverage their knowledge of invariances in a task or domain. And indeed, data augmentation is cited as essential to nearly every state-of-the-art result in image classification [Ciresan et al.; Dosovitskiy et al., 2015; Graham, 2014; Sajjadi et al., 2016] (see [Ratner et al., 2017c]), and is becoming increasingly common in other modalities as well [Lu et al., 2006].

Even on well studied benchmark tasks, however, the choice of data augmentation strategy is known to cause large variances in end performance and be difficult to select [Graham, 2014; Dosovitskiy et al., 2015], with papers often reporting their heuristically found parameter ranges [Ciresan et al.]. In practice, it is often simple to formulate a large set of primitive transformation operations, but time-consuming and difficult to find the parameterizations and compositions of them needed for state-of-the-art results. In particular, many transformation operations will have vastly different effects based on parameterization, the set of other transformations they are applied with, and even their particular order of composition. For example, brightness and saturation enhancements might be destructive when applied together, but produce realistic images when paired with geometric transformations.

Given the difficulty of searching over this configuration space, the de facto norm in practice consists of applying one or more transformations in random order and with random parameterizations selected from hand-tuned ranges. Recent lines of work attempt to automate

[2]As of version 0.9.

data augmentation entirely, but either rely on large quantities of labeled data [Baluja and Fischer, 2017; Mirza and Osindero, 2014], restricted sets of simple transformations [Fawzi et al., 2016; Hauberg et al., 2016], or consider only local perturbations that are not informed by domain knowledge [Baluja and Fischer, 2017; Miyato et al., 2015] (see Section 6.4). In contrast, our aim is to directly and flexibly leverage domain experts' knowledge of invariances as a valuable form of weak supervision in real-world settings where labeled training data is limited.

Automating Data Augmentation In this chapter, we present a new method for data augmentation that directly leverages user domain knowledge in the form of transformation operations, and automates the difficult process of composing and parameterizing them. We formulate the problem as one of learning a generative sequence model over black-box *transformation functions (TFs)*: user-specified operators representing incremental transformations to data points that need not be differentiable nor deterministic. For example, TFs could rotate an image by a small degree, swap a word in a sentence, or translate a segmented structure in an image (Fig. 6.1). We then design a generative adversarial objective [Goodfellow et al., 2014a] which allows us to train the sequence model to produce transformed data points which are still within the data distribution of interest, using unlabeled data. Because the TFs can be stochastic or non-differentiable, we present a reinforcement learning-based training strategy for this model. The learned model can then be used to perform data augmentation on labeled training data for any end discriminative model.

Given the flexibility of our representation of the data augmentation process, we can apply our approach in many different domains, and on different modalities including both text and images. On a real-world mammography image task, we achieve a 3.4 accuracy point boost above randomly composed augmentation by learning to appropriately combine standard image TFs with domain-specific TFs derived in collaboration with radiology experts. Using novel language model-based TFs, we see a 1.4 F1 boost over heuristic augmentation on a text relation extraction task from the ACE corpus. And on a 10%-subsample of the CIFAR-10 dataset, we achieve a 4.0 accuracy point gain over a standard heuristic augmentation approach and are competitive with comparable semi-supervised approaches. Additionally, we show empirical results suggesting that the proposed approach is robust to misspecified TFs. Our hope is that the proposed method will be of practical value to practitioners and of interest to researchers, so we have open-sourced the code at `https://github.com/HazyResearch/tanda`.

Outline of Chapter In this chapter we describe a paradigm for building data augmentation strategies as policies over user-provided *transformation functions*, and an approach for automatically learning to tune and compose them using unlabeled data:

- *In Section 6.1,* we start by describing the representation of data augmentation strategies as sequences of incremental, user-provided transformation functions, and the model we use to tune and compose them.
- *In Section 6.2,* we describe a generative adversarial approach for learning the model over data augmentation sequences using unlabeled data.
- *In Section 6.3,* we describe experiments validating the proposed approach across image and text datasets.
- *Finally, in Section 6.4* we briefly review related work.

We note that the above approach is available as an open source software framework at `https://github.com/HazyResearch/tanda`, and has also been partially integrated into Snorkel (`https://snorkel.org`) as of version 0.9.

6.1 Modeling Setup and Motivation

In the standard data augmentation setting, our aim is to expand a labeled training set by leveraging knowledge of class-preserving transformations. For a practitioner with domain expertise, providing individual transformations is straightforward. However, high performance augmentation techniques use *compositions* of finely tuned transformations to achieve state-of-the-art results [Dosovitskiy et al., 2015; Ciresan et al.; Graham, 2014], and heuristically searching over this space of all possible compositions and parameterizations for a new task is often infeasible. Our goal is to automate this task by learning to compose

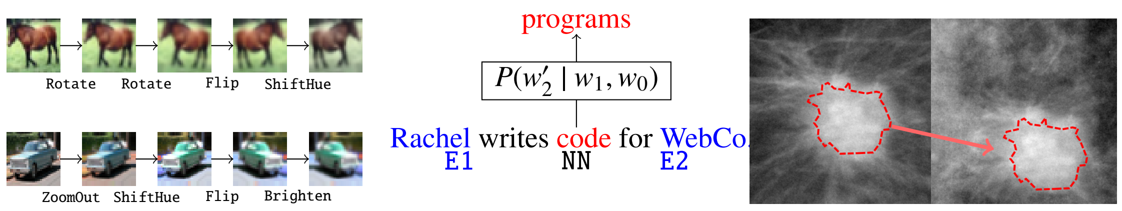


Figure 6.1: Three examples of transformation functions (TFs) in different domains: Two example sequences of incremental image TFs applied to CIFAR-10 images (*left*); a conditional word-swap TF using an externally trained language model and specifically targeting nouns (`NN`) between entity mentions (`E1,E2`) for a relation extraction task (*middle*); and an unsupervised segmentation-based translation TF applied to mass-containing mammography images (*right*).

and parameterize a set of user-specified transformation operators in ways that are diverse but still preserve class labels.

In our method, transformations are modeled as sequences of incremental user-specified operations, called transformation functions (TFs) (Fig. 6.1). Rather than making the strong assumption that all the provided TFs preserve class labels, as existing approaches do, we assume a weaker form of class invariance which enables us to use *unlabeled* data to learn a generative model over transformation sequences. We then propose two representative model classes to handle modeling both commutative and non-commutative transformations.

6.1.1 Augmentation as Sequence Modeling

In our approach, we represent transformations as sequences of incremental operations. In this setting, the user provides a set of K TFs, $\tau_i : \mathcal{X} \mapsto \mathcal{X}$, $i \in [1, K]$. Each TF performs an incremental transformation: for example, h_i could rotate an image by five degrees, swap a word in a sentence, or move a segmented tumor mass around a background mammography image (see Fig. 6.1). In order to accommodate a wide range of such user-defined TFs, we treat them as black-box functions which need not be deterministic nor differentiable.

This formulation gives us a tractable way to tune both the parameterization and composition of the TFs in a discretized but fine-grained manner. Our representation can be thought of as an implicit binning strategy for tuning parameterizations – e.g. a 15 degree

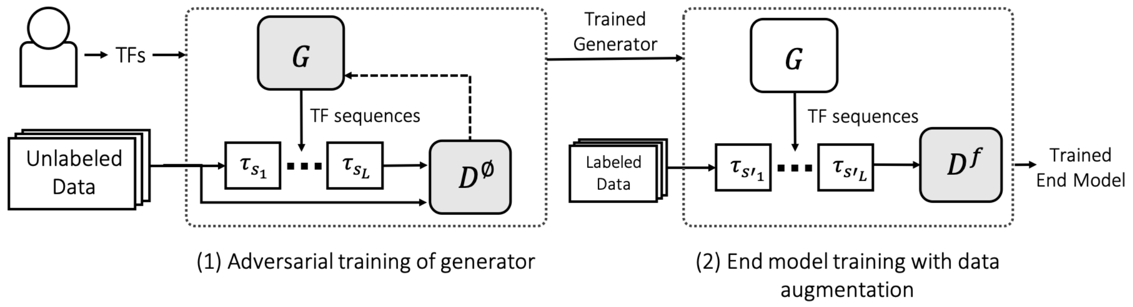


Figure 6.2: A high-level diagram of our method. Users input a set of transformation functions $\tau_1, ..., \tau_K$ and unlabeled data. A generative adversarial approach is then used to train a *null class* discriminator, $D^\emptyset$, and a generator, G, which produces TF sequences $\tau_{s_1}, ..., \tau_{s_L}$. Finally, the trained generator is used to perform data augmentation for an end discriminative model D^f.

rotation might be represented as three applications of a five-degree rotation TF. It also provides a direct way to represent compositions of multiple transformation operations. This is critical as a multitude of state-of-the-art results in the literature show the importance of using compositions of more than one transformations per image [Dosovitskiy et al., 2015; Ciresan et al.; Graham, 2014], which we also confirm experimentally in Section 6.3.

6.1.2 Weakening the Class-Invariance Assumption

Any data augmentation technique fundamentally relies on some assumption about the transformation operations' relation to the class labels. Previous approaches make the unrealistic assumption that all provided transformation operations preserve class labels for all data points. That is,

$$y(\tau_{s_L} \circ \ldots \circ \tau_{s_1}(x)) = y(x) \tag{6.1}$$

for label mapping function y, any sequence of TF indices $s_1, ..., s_L$, and *all* data points x.

This assumption puts a large burden of precise specification on the user, and based on our observations, is violated by many real-world data augmentation strategies. Instead, we consider a weaker modeling assumption. We assume that transformation operations will not map between classes, but might destructively map data points out of the distribution of

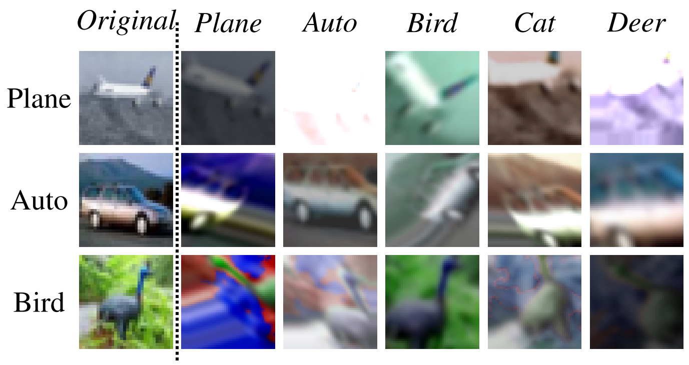


Figure 6.3: Our modeling assumption is that transformations may map out of the natural distribution of interest, but will rarely map *between* classes. As a demonstration, we take images from CIFAR-10 (each row) and randomly search for a transformation sequence that best maps them to a different class (each column), according to a trained discriminative model. The matches rarely resemble the target class but often no longer look like "normal" images at all. Note that we consider a fixed set of user-provided TFs, not adversarially selected ones.

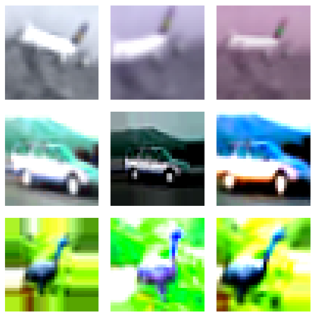


Figure 6.4: Some example transformed images generated using an augmentation generative model trained using our approach. Note that this is not meant as a comparison to Fig. 6.3.

interest entirely:

$$y(\tau_{s_L} \circ \ldots \circ \tau_{s_1}(x)) \in \{y(x), y_\emptyset\} \tag{6.2}$$

where $y_\emptyset$ represents an out-of-distribution *null class*. Intuitively, this weaker assumption is motivated by the categorical image classification setting, where we observe that transformation operations provided by the user will almost never turn, for example, a plane into a car, but may often turn a plane into an indistinguishable "garbage" image (Fig. 6.3). We are the first to consider this weaker invariance assumption, which we believe more closely matches various practical data augmentation settings of interest. In Section 6.3, we also provide empirical evidence that this weaker assumption is useful in binary classification settings and over modalities other than image data. Critically, it also enables us to learn a model of TF sequences using unlabeled data alone.

6.1.3 Minimizing Null Class Mappings Using Unlabeled Data

Given assumption (6.2), our objective is to learn a model G_θ which generates sequences of TF indices $s \in \{1, K\}^L$ with fixed length L, such that the resulting TF sequences $\tau_{s_1}, \ldots, \tau_{s_L}$ are not likely to map data points into $y_\emptyset$. Crucially, this does not involve using the class labels of any data points, and so we can use unlabeled data. Our goal is then to minimize the probability of a generated sequence mapping unlabeled data points into the null class, with respect to θ:

$$J_\emptyset = \mathbb{E}_{s\sim G_\theta}\left[\mathbb{E}_{x\sim\mathcal{U}}\left[P(y(\tau_{s_L} \circ \ldots \circ \tau_{s_1}(x)) = y_\emptyset)\right]\right] \tag{6.3}$$

where $\mathcal{U}$ is some distribution of unlabeled data.

Generative Adversarial Objective In order to approximate $P(y(\tau_{s_L} \circ \ldots \circ \tau_{s_1}(x)) = y_\emptyset)$, we jointly train the generator G_θ and a discriminative model $D_\phi^\emptyset$ using a generative adversarial network (GAN) objective [Goodfellow et al., 2014a], now minimizing with respect to θ and maximizing with respect to ϕ:

$$\tilde{J}_\emptyset = \mathbb{E}_{s\sim G_\theta}\left[\mathbb{E}_{x\sim\mathcal{U}}\left[\log(1 - D_\phi^\emptyset(\tau_{s_L} \circ \ldots \circ \tau_{s_1}(x)))\right]\right] + \mathbb{E}_{x'\sim\mathcal{U}}\left[\log(D_\phi^\emptyset(x'))\right] \tag{6.4}$$

As in the standard GAN setup, the training procedure can be viewed as a minimax game in which the discriminator's goal is to assign low values to transformed, out-of-distribution data points and high values to real in-distribution data points, while simultaneously, the generator's goal is to generate transformation sequences which produce data points that are indistinguishable from real data points according to the discriminator. For $D_\phi^\emptyset$, we use an all-convolution CNN as in [Radford et al., 2015]. For further details, see the Appendix of [Ratner et al., 2017c].

Diversity Objective An additional concern is that the model will learn a variety of null transformation sequences (e.g. rotating first left than right repeatedly). Given the potentially large state-space of actions, and the black-box nature of the user-specified TFs, it seems infeasible to hard-code sets of inverse operations to avoid. To mitigate this, we instead consider a second objective term:

$$J_d = \mathbb{E}_{s\sim G_\theta}\left[\mathbb{E}_{x\sim\mathcal{U}}\left[d(\tau_{s_L}\circ\ldots\circ\tau_{s_1}(x), x)\right]\right] \tag{6.5}$$

where $d : \mathcal{X}\times\mathcal{X}\mapsto\mathbb{R}$ is some distance function. For d, we evaluated using both distance in the raw input space, and in the feature space learned by the final pre-softmax layer of the discriminator $D_\phi^\emptyset$. Combining eqns. 6.4 and 6.5, our final objective is then $J = \tilde{J}_\emptyset + \alpha J_d^{-1}$ where $\alpha > 0$ is a hyperparameter. We minimize J with respect to θ and maximize with respect to ϕ.

6.1.4 Modeling Transformation Sequences

We now consider two model classes for G_θ:

Independent Model We first consider a *mean field* model in which each sequential TF is chosen independently. This reduces our task to one of learning K parameters, which we can think of as representing the task-specific "accuracies" or "frequencies" of each TF. For example, we might want to learn that elastic deformations or swirls should only rarely be applied to images in CIFAR-10, but that small rotations can be applied frequently. In particular, a mean field model also provides a simple way of effectively learning stochastic, discretized parameterizations of the TFs. For example, if we have a TF representing five-degree rotations, `Rotate5Deg`, a marginal value of $P_{G_\theta}(\texttt{Rotate5Deg}) = 0.1$ could be thought of as roughly equivalent to learning to rotate $0.5L$ degrees on average.

State-Based Model There are important cases, however, where the independent representation learned by the mean field model could be overly limited. In many settings, certain TFs may have very different effects depending on which other TFs are applied with them. As an example, certain similar pairs of image transformations might be overly lossy when applied together, such as a blur and a zoom operation, or a brighten and a saturate operation. A mean field model could not represent such disjunctions as these. Another scenario where an independent model fails is where the TFs are non-commutative, such as with lossy operators (e.g. image transformations which use aliasing). In both of these cases, modeling the sequences of transformations could be important. Therefore we consider a long short-term memory (LSTM) network as as a representative sequence model. The output from each cell of the network is a distribution over the TFs. The next TF in the sequence is then sampled from this distribution, and is fed as a one-hot vector to the next cell in the network.

6.2 Learning a Transformation Sequence Model

The core challenge that we now face in learning G_θ is that it generates sequences over TFs which are not necessarily differentiable or deterministic. This constraint is a critical facet of our approach from the usability perspective, as it allows users to easily write TFs as black-box scripts in the language of their choosing, leveraging arbitrary subfunctions, libraries, and methods. In order to work around this constraint, we now describe our model in the syntax of reinforcement learning (RL), which provides a convenient framework and set of approaches for handling computation graphs with non-differentiable or stochastic nodes [Schulman et al., 2015].

Reinforcement Learning Formulation Let s_i be the index of the ith TF applied, and $\tilde{x}_i$ be the resulting incrementally transformed data point. Then we consider

$$S_t = (x, \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_t, s_1, \ldots, s_t)$$

as the state after having applied t of the incremental TFs. Note that we include the incrementally transformed data points $\tilde{x}_1, \ldots, \tilde{x}_t$ in S_t since the TFs may be stochastic. Each of the model classes considered for G_θ then uses a different *state representation* $\hat{S}$. For the mean field model, the state representation used is $\hat{S}_t^{\mathrm{MF}} = \emptyset$. For the LSTM model, we use $\hat{S}_t^{\mathrm{LSTM}_\theta} = \mathsf{LSTM}_\theta(\hat{S}_{t-1}^{\mathrm{LSTM}_\theta}, s_t)$, the state update operation performed by a standard LSTM cell parameterized by θ.

Policy Gradient with Incremental Rewards Let $\ell_t(x, s) = \log(1 - D_\phi^\emptyset(\tilde{x}_t))$ be the *cumulative loss* for a data point x at step t, with $\ell_0(x) = \ell_0(x, s) \equiv \log(1 - D_\phi^\emptyset(x))$. Let

$R(S_t) = \ell_t(x, s) - \ell_{t-1}(x, s)$ be the *incremental reward*, representing the difference in discriminator loss at incremental transformation step t. We can now recast the first term of our objective $\tilde{J}_\emptyset$ as an expected sum of incremental rewards:

$$U(\theta) \equiv \mathbb{E}_{s\sim G_\theta}\left[\mathbb{E}_{x\sim\mathcal{U}}\left[\log(1 - D_\phi^\emptyset(\tau_{s_L} \circ \ldots \circ \tau_{s_1}(x)))\right]\right] = \mathbb{E}_{s\sim G_\theta}\left[\mathbb{E}_{x\sim\mathcal{U}}\left[\ell_0(x) + \sum_{t=1}^{L} R(S_t)\right]\right] \tag{6.6}$$

We omit ℓ_0 in practice, equivalent to using the loss of x as a baseline term. Next, let π_θ be the stochastic transition policy implicitly defined by G_θ. We compute the recurrent policy gradient [Wierstra et al., 2010] of the objective $U(\theta)$ as:

$$\nabla_\theta U(\theta) = \mathbb{E}_{s\sim G_\theta}\left[\mathbb{E}_{x\sim\mathcal{U}}\left[\sum_{t=1}^{L} R(S_t)\nabla_\theta \log \pi_\theta(s_t \mid \hat{S}_{t-1})\right]\right] \tag{6.7}$$

Following standard practice, we approximate this quantity by sampling batches of n data points and n_A sampled action sequences per data point. We also use standard techniques of discounting with factor $\gamma \in [0, 1]$ and considering only future rewards [Greensmith et al., 2004]. See the Appendix of [Ratner et al., 2017c] for details.

6.3 Experiments

We experimentally validate the proposed framework by learning augmentation models for several benchmark and real-world data sets, exploring both image recognition and natural language understanding tasks. Our focus is on the performance of end classification models trained on labeled datasets augmented with our approach and others used in practice. We also examine robustness to user misspecification of TFs, and sensitivity to core hyperparameters.

6.3.1 Datasets and Transformation Functions

Benchmark Image Datasets We ran experiments on the MNIST [LeCun et al., 1998] and CIFAR-10 [Krizhevsky and Hinton, 2009] datasets, using only a subset of the class labels to train the end classification models and treating the rest as unlabeled data. We used a generic set of TFs for both MNIST and CIFAR-10: small rotations, shears, central swirls, and elastic deformations. We also used morphologic operations for MNIST, and adjustments to hue, saturation, contrast, and brightness for CIFAR-10.

Benchmark Text Dataset We applied our approach to the *Employment* relation extraction subtask from the NIST Automatic Content Extraction (ACE) corpus [Doddington et al., 2004], where the goal is to identify mentions of employer-employee relations in news articles. Given the standard class imbalance in information extraction tasks like this, we used data augmentation to oversample the minority positive class. The flexibility of our TF representation allowed us to take a straightforward but novel approach to data augmentation in this setting. We constructed a trigram language model using the ACE corpus and Reuters Corpus Volume I [Lewis et al., 2004] from which we can sample a word conditioned on the preceding words. We then used this model as the basis for a set of TFs that select words to swap based on the part-of-speech tag and location relative to entities of interest (see [Ratner et al., 2017c] for details).

Mammography Tumor-Classification Dataset To demonstrate the effectiveness of our approach on real-world applications, we also considered the task of classifying benign versus malignant tumors from images in the Digital Database for Screening Mammography (DDSM) dataset [Heath et al., 2000; Clark et al., 2013; Sawyer Lee et al., 2016], which is a class-balanced dataset consisting of 1506 labeled mammograms. In collaboration with domain experts in radiology, we constructed two basic TF sets. The first set consisted of standard image transformation operations sub-selected so as not to break class-invariance in the mammography setting. For example, brightness operations were excluded for this reason. The second set consisted of both the first set as well as several novel segmentation-based transplantation TFs. Each of these TFs utilized the output of an unsupervised segmentation algorithm to isolate the tumor mass, perform a transformation operation such as rotation or shifting, and then stitch it into a randomly-sampled benign tissue image. See Fig. 6.1 (right panel) for an illustrative example, and [Ratner et al., 2017c] for further details.

6.3.2 End Classifier Performance

We evaluated our approach by using it to augment labeled training sets for the tasks mentioned above, and show that we achieve strong gains over heuristic baselines. In particular, for a given set of TFs, we evaluate the performance of mean field (*MF*) and LSTM generators trained using our approach against two standard data augmentation techniques used in practice. The first (*Basic*) consists of applying random crops to images, or performing simple minority class duplication for the ACE relation extraction task. The second (*Heur.*) is the standard heuristic approach of applying random compositions of the given set of transformation operations, the most common technique used in practice [Ciresan et al.; Graham, 2014; He et al., 2016]. For both our approaches (*MF* and *LSTM*) and *Heur.*, we additionally use the same random cropping technique as in the *Basic* approach. We present these results in Table 6.1, where we report test set accuracy (or F1 score for ACE), and use a random subsample of the available labeled training data. Additionally, we include an extra row for the DDSM task highlighting the impact of adding domain-specific (*DS*) TFs – the segmentation-based operations described above – on performance.

In Table 6.2 we additionally compare to two related generative-adversarial methods, the Categorical GAN (CatGAN) [Springenberg, 2015], and the semi-supervised GAN (SS-GAN) from [Salimans et al., 2016]. Both of these methods use GAN-based architectures trained on unlabeled data to generate new out-of-class data points with which to augment a labeled training set. Following their protocol for CIFAR-10, we train our generator on the full set of unlabeled data, and our end discriminator on ten disjoint random folds of the labeled training set not including the validation set (i.e. $n = 4000$ each), averaging the results.

In all settings, we train our TF sequence generator on the full set of unlabeled data. We select a fixed sequence length for each task via an initial calibration experiment (Fig. 6.5b). We use $L = 5$ for ACE, $L = 7$ for DDSM + DS, and $L = 10$ for all other tasks. We note that our findings here mirrored those in the literature, namely that compositions of multiple TFs lead to higher end model accuracies. We selected hyperparameters of the generator via performance on a validation set. We then used the trained generator to transform the entire training set at each epoch of end classification model training. For MNIST and DDSM we

Task	**%**	**None**	**Basic**	**Heur.**	**MF**	**LSTM**
MNIST	1	90.2	95.3	95.9	96.5	**96.7**
	10	97.3	98.7	99.0	**99.2**	99.1
CIFAR-10	10	66.0	73.1	77.5	79.8	**81.5**
	100	87.8	91.9	92.3	**94.4**	94.0
ACE (F1)	100	62.7	59.9	62.8	62.9	**64.2**
DDSM	10	57.6	58.8	59.3	58.2	61.0
DDSM + DS	10	57.6	58.8	53.7	59.9	**62.7**

Table 6.1: Test set performance of end models trained on subsamples of the labeled training data (%), not including validation splits, using various data augmentation approaches. *None* indicates performance with no augmentation. All tasks are measured in accuracy, except ACE which is measured by F1 score.

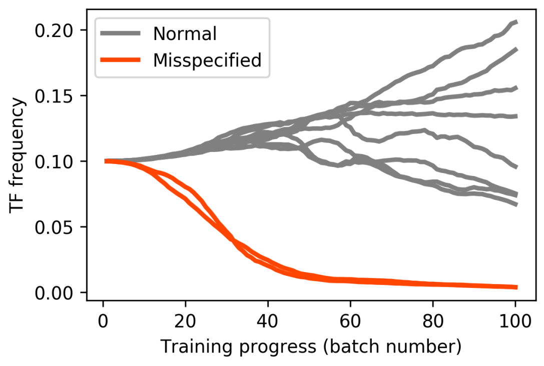
Model	**Acc. (%)**
CatGAN	80.42 ± 0.58
SS-GAN	81.37 ± 2.32
LSTM	81.47 ± 0.46

Table 6.2: Reported end model accuracies, averaged across 10% subsample folds, on CIFAR-10 for comparable GAN methods.

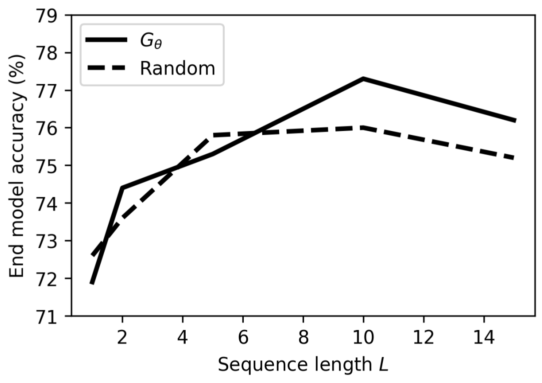
use a four-layer all-convolutional CNN, for CIFAR10 we use a 56-layer ResNet [He et al., 2016], and for ACE we use a bi-directional LSTM. Additionally, we incorporate a basic transformation regularization term as in [Sajjadi et al., 2016] (see [Ratner et al., 2017c]), and train for the last ten epochs without applying any transformations as in [Graham, 2014]. In all cases, we use hyperparameters as reported in the literature. For further details of generator and end model training see [Ratner et al., 2017c].

We see that across the applications studied, our approach outperforms the heuristic data augmentation approach most commonly used in practice. Furthermore, the LSTM generator outperforms the simple mean field one in most settings, indicating the value of modeling sequential structure in data augmentation. In particular, we realize significant gains over standard heuristic data augmentation on CIFAR-10, where we are competitive with comparable semi-supervised GAN approaches, but with significantly smaller variance. We also train the same CIFAR-10 end model using the full labeled training dataset, and again see strong relative gains (2.1 pts. in accuracy over heuristic), coming within 2.1 points of the current state-of-the-art [Huang et al., 2016] using our much simpler end model.

On the ACE and DDSM tasks, we also achieve strong performance gains, showing



(a)



(b)

Figure 6.5: (a) Learned TF frequency parameters for misspecified and normal TFs on MNIST. The mean field model correctly learns to avoid the misspecified TFs. (b) Larger sequence lengths lead to higher end model accuracy on CIFAR-10, while random performs best with shorter sequences, according to a sequence length calibration experiment.

the ability of our method to productively incorporate more complex transformation operations from domain expert users. In particular, in DDSM we observe that the addition of the segmentation-based TFs causes the heuristic augmentation approach to perform significantly worse, due to a large number of new failure modes resulting from combinations of the segmentation-based TFs – which use gradient-based blending – and the standard TFs such as zoom and rotate. In contrast, our LSTM model learns to avoid these destructive subsequences and achieves the highest score, resulting in a 9.0 point boost over the comparable heuristic approach.

Robustness to TF Misspecification One of the high-level goals of our approach is to enable an easier interface for users by not requiring that the TFs they specify be completely class-preserving. The lack of any assumption of well-specified transformation operations in our approach, and the strong empirical performance realized, is evidence of this robustness. To additionally illustrate the robustness of our approach to misspecified TFs, we train a mean field generator on MNIST using the standard TF set, but with two TFs (shear operations) parameterized so as to map almost all images to the null class. We see in Fig. 6.5a that the generator learns to avoid applying the misspecified TFs (red lines) almost entirely.

6.4 Related Work

We now review related work, both to motivate comparisons in the experiments section and to present complementary lines of work.

Heuristic Data Augmentation Most state-of-the-art image classification pipelines use some limited form of data augmentation [Graham, 2014; Dosovitskiy et al., 2015]. This generally consists of applying crops, flips, or small affine transformations, in fixed order or at random, and with parameters drawn randomly from hand-tuned ranges. In addition, various studies have applied heuristic data augmentation techniques to modalities such as audio [Uhlich et al., 2017] and text [Lu et al., 2006]. As reported in the literature, the selection of these augmentation strategies can have large performance impacts, and thus can require extensive selection and tuning by hand [Ciresan et al.; Dosovitskiy et al., 2015] (see the Appendix of [Ratner et al., 2017c] as well).

Interpolation-Based Techniques Some techniques have explored generating augmented training sets by interpolating between labeled data points. For example, the well-known SMOTE algorithm applies this basic technique for oversampling in class-imbalanced settings [Chawla et al., 2002], and recent work explores using a similar interpolation approach in a learned feature space [DeVries and Taylor, 2017]. [Hauberg et al., 2016] proposes learning a class-conditional model of diffeomorphisms interpolating between nearest-neighbor labeled data points as a way to perform augmentation. We view these approaches as complementary but orthogonal, as our goal is to directly exploit user domain knowledge of class-invariant transformation operations.

Adversarial Data Augmentation Several lines of recent work have explored techniques which can be viewed as forms of data augmentation that are adversarial with respect to the end classification model. In one set of approaches, transformation operations are selected adaptively from a given set in order to maximize the loss of the end classification model being trained [Teo et al., 2008; Fawzi et al., 2016]. These procedures make the strong assumption that all of the provided transformations will preserve class labels, or use bespoke models over restricted sets of operations [Sixt et al., 2016]. Another line of recent work has showed that augmentation via small adversarial linear perturbations can act as a regularizer [Goodfellow et al., 2014b; Miyato et al., 2015]. While complimentary, this work does not consider taking advantage of non-local transformations derived from user knowledge of task or domain invariances.

Finally, generative adversarial networks (GANs) [Goodfellow et al., 2014a] have recently made great progress in learning complete data generation models from unlabeled data. These can be used to augment labeled training sets as well. Class-conditional GANs [Baluja and Fischer, 2017; Mirza and Osindero, 2014] generate artificial data points but require large sets of labeled training data to learn from. Standard unsupervised GANs can be used to generate additional out-of-class data points that can then augment labeled training sets [Salimans et al., 2016; Springenberg, 2015]. We compare our proposed approach with these methods empirically in Section 6.3.

Chapter 7

Conclusion and Future Work

In this thesis, we described work on *training data management systems* that enable users to programmatically build, manage, and model training datasets, and described empirical results and real-world deployments demonstrating that this could be a radically faster, more flexible, and more accessible interface to machine learning. Already, we have seen many teams at various large technology companies re-organizing around these new ideas of training dataset management, with Snorkel and other programmatic ways of building, managing, and modeling training data serving as a way to represent, re-use, and combine various knowledge resources effectively across an organization [Bach et al., 2019; Bringer et al., 2019]. These changes suggest a radical shift to the way data-driven software systems are built, shared, and deployed within organizations, and point to a near future wherein large numbers of inter-related machine learning models are rapidly developed and deployed using increasingly high-level, passive, and noisy supervision, for increasingly complex tasks, and with increasing systems support for parts of the machine learning pipeline outside of the model itself.

In this Chapter, we give an overview of several exciting directions for future research, motivated by these shifts: starting with extensions to Snorkel, and systems and techniques for weak supervision more broadly, and then extending to broader topics around machine learning systems.

7.1 Snorkel & Weak Supervision Systems

We start by outlining several research directions around extensions of the techniques described in this thesis, and Snorkel specifically. However, we note that many of the following research directions pertain to weak supervision approaches and systems much more generally as well.

Handling Structured & Regression Settings A first set of directions is to extend Snorkel, and the weak supervision techniques outlined in this thesis, to various settings beyond categorical classification. Extending data programming and the label model of Chapter 3 as well as data augmentation (Chapter 6) and other weak supervision techniques to handle *structured data* is one interesting direction: for example, sequential data such as time series and video, as well as more complex structured data such as code and other objects with rich and complex structure. Extending these weak supervision approaches to handle settings beyond classification, such as *regression*, *reinforcement learning*, *anomaly detection*, and others is another natural and interesting set of next steps, as well as the complementary challenge of properly handling labeling functions that output continuous, distributional, or other more complex values.

Understanding Generalization in Weak Supervision In Chapters 3 and 4, we define a pipeline consisting of two models: first, the label model, which is a reweighted combination of the labeling function outputs; and second, some end discriminative model trained on the outputs of the label model, but defined over arbitrary input features or data. One question which immediately arises is: why should we use this second, end discriminative model at all?

In many settings, such as the cross-modal ones described in Sections 4.3 and 4.5, the answer is that the labeling functions—and therefore the label model—cannot be applied at test time, but the end discriminative model is defined over a disjoint set of features—e.g. a different data modality—and therefore can be applied. For example at training time we may have both text and image data, but at test time only image data, as in the radiology triaging setting; or we may have non-servable features that are useful in the labeling functions, but not servable in production (Section 4.5). In these settings, the reason for training both models is clear (and can be interpreted as, implicitly, a form of cross-feature distillation or transfer).

In other settings, however, the label and discriminative models are defined over the same set of features. Empirically, we show that in many of these settings, the end discriminative model is able to generalize beyond the coverage of the labeling functions, thereby increasing recall (Section 4.3). However, characterizing this weak supervision generalization in a more precise and formal way remains a theoretical challenge for future research, and one that would also have implications relevant to many weak supervision approaches beyond Snorkel. Intuitively, and based on empirical evidence, we would expect there to be two basic mechanisms: first, the use of implicit or explicit inductive biases in the end model, such as pre-trained embeddings or network layers; and second, the ability of a properly regularized end model to spread weight to features that co-occur with the labeling function outputs. Better theoretical characterization of when and how generalization occurs in these programmatic weak supervision settings would presumably help practitioners understand when, where, and how to apply weak supervision approaches with specific end model types. More broadly, considering more complex ways of connecting the user-provided weak supervision—via the label model—and the final model being trained is an interesting direction for future study; for example, the label model and end discriminative model could potentially share information in a higher-density way than training labels, be trained jointly, and/or be combined in various ways.

Formalizing & Supporting Programmatic Weak Supervision Workflows In Chapters 3 and 4, we define a basic pipeline and workflow whereby users inspect a sample of unlabeled data, or labeled data from error analysis, then write weak supervision operators (e.g. labeling or transformation functions), and then use these to train a model. This iterative workflow is simple and intuitive enough that many users, including those in the user study we report on (Section 4.4), have been able to successfully use Snorkel in a range of real world settings (e.g. see Section 4.5). However, guiding, structuring, and formalizing this programmatic weak supervision workflow can undoubtedly help real-world users to more efficiently use Snorkel and, more broadly, weak supervision techniques.

One natural direction is to consider the intersection of the data programming approach proposed in this thesis and *active learning*. Traditionally, active learning considers identifying data points to be labeled by hand, such that these labels will be more valuable to end model performance than if randomly sampled. In our setting, we might instead consider identifying sets of data points to show the user in order to (either explicitly or implicitly) prompt them to write programmatic weak supervision operators, e.g. labeling or transformation functions, that cover a specific part of the space. Other research directions might focus on auto-suggesting labeling functions [Varma and Ré, 2019] or labeling function templates; automatically completing or suggesting completions to program sketches; and prompting or guiding the iterative weak supervision development process. Broadly, programmatic weak supervision has the potential to radically transform the practice of developing supervised learning models from a largely labeling-and-tuning process into an iterative development one, and this shift will lead to many new research directions around this emerging workflow.

Ascending the Code-as-Supervision Stack The overall goal of the methods in this thesis, and Snorkel specifically, is to enable users to *program* the modern machine learning stack, by labeling training data with labeling functions rather than manual annotation. This *code-as-supervision* approach can then inherit the traditional advantages of code such as modularity, debuggability, and higher level abstraction layers. In particular, enabling this last element—even higher-level, more declarative ways of specifying labeling functions—has been a major motivation of the Snorkel project.

Since Snorkel's release, various extensions have explored higher-level, more declarative interfaces for labeling training data by building on top of Snorkel (Figure 7.1). One idea, motivated by the difficulty of writing labeling functions directly over image or video data, is to first compute a set of features or *primitives* over the raw data using unsupervised approaches, and then write labeling functions over these building blocks [Varma et al., 2017]. For example, if the goal is to label instances of people riding bicycles, we could first run an off-the-shelf pre-trained algorithm to put bounding boxes around people and bicycles, and then write labeling functions over the dimensions or relative locations of these bounding boxes.[1] In medical imaging tasks, anatomical segmentation masks provide

[1]See the image tutorial at `snorkel.stanford.edu`.

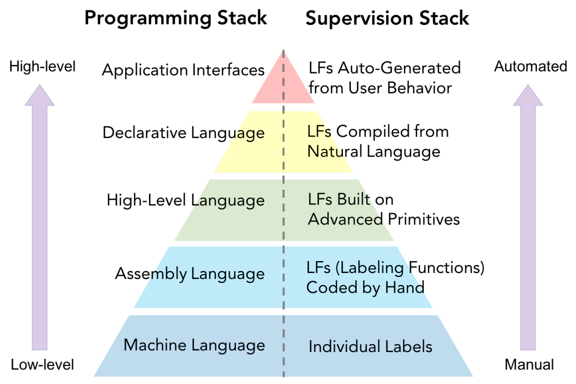


Figure 7.1: In a traditional programming stack, progressively higher-level languages and abstractions provide increasingly simple and declarative interfaces. Similarly, we envision a *code-as-supervision* stack built on top of the basic unit of labeling functions, allowing users to label training data in increasingly higher-level ways. Figure from [Ratner et al., 2019c].

a similarly intuitive semantic abstraction for writing labeling functions over. For example, in a large collection of cardiac MRI videos from the UK Biobank, creating segmentations of the aorta enabled a cardiologist to define labeling functions for identifying rare aortic valve malformations [Fries et al., 2019] (Section 4.5).

An even higher level interface is *natural language*. The Babble Labble project [Hancock et al., 2018] accepts natural language explanations of data points, and then uses semantic parsers to parse these explanations into labeling functions. In this way, users without programming knowledge have the capability to write labeling functions just by explaining reasons why data points have specific labels. Another related approach is to use program synthesis techniques, combined with a small set of labeled data points, to automatically generate labeling functions [Varma and Ré, 2019]. Finally, *observational* approaches can potentially leverage passively-collected signals such as from mouse, keyboard, and eye tracking devices, query and device logs, and more. Moving forwards, a diverse and exciting range of inputs can potentially be leveraged or collected via new interfaces, and then effectively "compiled" to programmatic supervision that is then modeled and applied to data by systems like Snorkel.

Exploring New Weak Supervision Operators In this thesis, we introduce two weak supervision operators—labeling functions and transformation functions—which serve as abstractions for various forms of weak supervision, both new and existing. However, there is a much broader range of ways that practitioners do, and could, interact with machine learning via the conduit of training data. Additionally, further empirical and theoretical exploration of how these different operators interact and are optimally combined, and how they can be used in concert with the broader spectrum of methods for addressing limited labeled training data (Section 2.2), will likely be fruitful and impactful area of future research.

New Execution Tradeoffs Finally, there are additional research directions of interest studying other tradeoffs that balance end model accuracy with other performance metrics such as memory and speed. As one example, labeling functions may at times be expensive to execute, and we could use similar techniques as those developed in this thesis—e.g. building off of structure learning techniques—to determine when execution of some of them may be skipped. Systems tradeoffs around data augmentation, especially when being used in concert with weak programmatic supervision, represent another interesting direction of inquiry.

7.2 Supporting the Broader Machine Learning Pipeline

As machine learning models become increasingly commoditized and well-supported, the processes upstream and downstream of them will become increasingly critical. This thesis focused on one core upstream task that has traditionally been done in heavily manual and/or ad hoc ways—labeling and managing training data—and attempted to formalize, support, and accelerate it. Similarly, a range of other upstream and downstream tasks should prove to be interesting and impactful targets for future research at the intersection of data management systems, machine learning algorithms, and theory.

One such task is the *collection of unlabeled data*. The approaches in this thesis, and in many other settings, assume some unbiased and i.i.d. sampling of unlabeled data, and then consider ways of labeling it (e.g. with weak supervision) or directly leveraging it

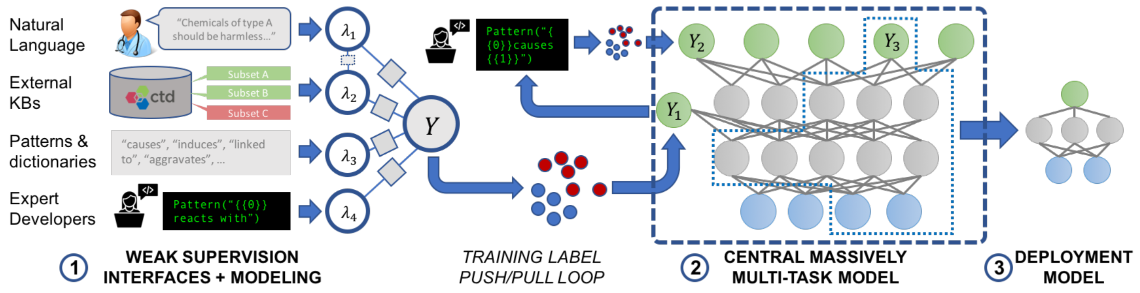

Natural Language
"Chemicals of type A should be harmless..."
External KBs
ctd
Subset A
Subset B
Subset C
Patterns & dictionaries
"causes", "induces", "linked to", "aggravates", ...
Expert Developers
Pattern("{{0}} reacts with")
λ_1
λ_2
λ_3
λ_4
Y
Pattern("{{0}}causes{{1}}")
Y_2
Y_3
Y_1
1 WEAK SUPERVISION INTERFACES + MODELING
TRAINING LABEL PUSH/PULL LOOP
2 CENTRAL MASSIVELY MULTI-TASK MODEL
3 DEPLOYMENT MODEL

Figure 7.2: In [Ratner et al., 2019c], we envision a pipeline consisting of (1) *building* training sets from weak supervision, provided via a stack of interfaces at different levels of abstraction; (2) *combining* training signals for tasks across an organization into a central *massively multitask* model, which allows developers to contribute and use task models via the simple interface of *labels*; and (3) *deploying* servable models by distilling tasks from the central model into commodity edge models.

(e.g. with semi-supervised learning). However, in reality, unlabeled data is also collected in biased, noisy, and ad hoc ways. For example, unlabeled scientific documents may come from some large, manually-constructed PubMed query; medical images may come from a specific silo, patient distribution, and/or date range. While the basic notion of distributional shift and bias in datasets is far from new, there is likely a rich area of new systems, methods, and formalizations to build around how developers in real-world settings can manage and model the collection of unlabeled data from various sources, and tie it into the rest of the ML pipeline.

Another example of an upstream process that is often done in ad hoc ways is *candidate extraction*, a process common in high-class imbalance tasks such as information extraction and segmentation, in which some set of basic objects are defined heuristically, and then a model is trained to operate over them. Putting more structure around this process, moving towards automating it—either jointly with the end model, or separately—and further formalizing the inherent tradeoffs are all interesting research directions.

7.3 Massively Multi-Task & Multi-Model Systems

In Chapter 5, we described extending the data programming and weak supervision approaches in this thesis to the *multi-task learning (MTL)* setting. MTL has recently received a surge of renewed interest in the context of new deep learning architectures [Ruder, 2017], where the focus in general is on achieving better accuracy across all tasks by jointly learning and sharing some representation. However, there are also a range of machine learning *systems* questions and challenges that arise in the MTL context, especially when these MTL models are coupled with weak supervision approaches such as those presented in this thesis, e.g. as in Chapter 5.

To start, the increasing prevalence of machine learning, especially when coupled with new weak supervision approaches that enable new models to be created with a fraction of the time and cost, points to a near term in which multi-task models contain not several tasks, but tens or hundreds of tasks. This new *massively multi-task learning (MMTL)* regime [Ratner et al., 2019c] emphasizes traditional MTL challenges at an entirely new scale: e.g., how do MMTL model maintainers ensure that new tasks do not negatively affect existing ones? How do they assemble network architectures with tens to hundreds of tasks? Especially in a setting where programmatic supervision approaches like Snorkel are used to build and manage training sets that can change every time a developer edits a labeling function, new questions arise around efficiently and incrementally maintaining and serving massively multi-task models, which may require novel techniques at the intersection of traditional data management practices and new machine learning ones. Massively multi-task models could conceivably become a new form of dynamic "codebase" for sharing, updating, and deploying learned representations and models across an organization, leading to a whole host of critical new research questions at the intersection of systems and machine learning.

Finally, while one apotheosis of modern multi-task learning efforts might be the merging of all tasks in an application or organization into a single model, engineering and social realities that many ML pipelines will involve many separate models, as they increasingly do today. For example, a standard end-to-end knowledge base construction application might consist of separate, interconnected models for crawling the web to collect documents, parsing these documents, tagging named entities, identifying relations between the tagged entities, and completing a knowledge base populated with these entities and edges. More broadly, it is increasingly common for models to be defined over the outputs of other models- either as input features, or, in the case of the pipelines as considered in this thesis, as weak supervision. As models become quicker and easier to build, using modern infrastructure and tools, and train, using weak supervision approaches as in this thesis, managing the increasingly complex and dynamic *multi-model systems* that result will be a major challenge. Aspects such as tracking heterogeneous dependencies between models, e.g. through pipelined inputs, shared representations, and weak supervision, incrementally maintaining and updating these systems, and defining design tradeoffs between these modular multi-model systems and singular multi-task models will present interesting and impactful research directions.

7.4 Conclusion

In this thesis, we focused on a tectonic shift in the machine learning development landscape away from traditional focal points like feature and model engineering, and towards a new bottleneck of *building and managing training datasets*. We viewed this often prohibitively expensive task of labeling and managing training data as a fundamental data management and machine learning problem, and proposed a new approach, data programming, in which rather than hand-labeling training data, subject matter expert users label data programmatically by writing small heuristic labeling functions. These labeling functions can have limited coverage, be noisy, correlated, and conflict with each other; to address this we developed a set of theoretically-grounded statistical modeling techniques for estimating their accuracies and correlations in the absence of ground truth, and then re-weighting and combining their outputs for use as clean, confidence-weighted training labels.

We then described Snorkel, a framework for building and managing training data using the approach of data programming, and designed around the core hypothesis that *even non-expert users can build high-performance machine learning applications almost entirely by programmatically labeling and managing training data*. We validated this empirically, theoretically, through user studies, and by supporting Snorkel as an open source framework over several years, during which it was deployed in science, medicine, and industry across problems involving text, semi-structured, image, video, time series, and other data modalities. Finally, we described two additional systems, aimed at the same goal of enabling users to build performant machine learning systems by building and managing training datasets. First, we described Snorkel MeTaL, an extension of Snorkel to the multi-task learning setting where weakly-supervised models for multiple tasks share a jointly learned representation. Second, we described techniques for programmatically defining data augmentation strategies, in which users write transformation functions that express knowledge of domain invariants, which are then automatically compiled into data augmentation policies over training datasets.

The overall goal of this thesis is to demonstrate that programmatically building, managing, and modeling training datasets can be a powerful, effective, and accessible interface to machine learning, and that in turn training data should be viewed not as a costly manual bottleneck, but as a medium for effectively *programming* modern machine learning models. Our hope is that the work in this thesis not only continues to provide real-world value, but also serves to incrementally assist a new wave of more responsive, powerful, and accessible machine learning systems.

Appendix A

Glossary of Symbols

Symbol	Definition
$\mathcal{X}$	Space of data points
x	Data point, $x \in \mathcal{X}$
n	Number of data points in the (labeled or unlabeled) training set
$\mathcal{Y}$	Space of labels
y	Label, $y \in \mathcal{Y}$
$\vec{y}$	Vector of labels, $\vec{y} \in \mathcal{Y}^n$
T	A labeled training dataset $T = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$
X_U	An unlabeled training dataset $X_U = \{x^{(1)}, \dots, x^{(n)}\}$
$\mathcal{D}$	Underlying data distribution
$\mathcal{H}$	Model or hypothesis class
w	Model parameters
h_w	Model, $h_w \in \mathcal{H}$
l	Loss function
$R, \hat{R}$	The risk and empirical risk respectively
L	The (marginal) likelihood of the observed data
ϕ	A feature extractor, mapping data points to features $\phi : \mathcal{X} \mapsto \mathbb{R}^d$
d	Dimension of the feature vector $\phi(x)$
λ_j	The output of the jth weak labeling function for a data point
m	The number of labeling functions
λ	The vector of the m labeling functions for a data point
Λ	The $n \times m$ *label matrix* of the labeling function outputs over a dataset
$\emptyset$	An output symbol denoting that the labeling function has abstained
θ	The label model parameters
ψ	A sufficient statistic or factor function in an exponential model
G_λ	The labeling function dependency graph, $G_\lambda = (V, E)$
G_{inv}	The inverse labeling function dependency graph
Ω	The augmented edge set of G_{inv}
$\mathcal{C}$	The set of cliques in G_λ
$\bar{\mathcal{C}}, \mathcal{S}$	The maximal and separator set cliques of the junction tree of G_λ
O	The observable cliques $O = \{C \mid y \notin C, C \in \bar{\mathcal{C}}\}$
d_C	The dimensions of the minimal indicator variables vector $\psi(C)$
G_{task}	The task graph in a multi-task setting
c_j	The coverage set of tasks λ_j emits non-null labels for
τ_i	A transformation function, $\tau_i : \mathcal{X} \mapsto \mathcal{X}$

Table A.1: Glossary of symbols used in this thesis.

Appendix B

Proofs: Maximum Marginal Likelihood Approach

B.1 General Theoretical Results

In this section, we will state the full form of the theoretical results we alluded to in the body of the paper. First, we restate, in long form, our setup and assumptions, focusing on the binary setting (with abstains).

We assume that, for some function $\psi : \{-1, \emptyset, 1\}^m \times \{-1, 1\} \mapsto \{-1, 0, 1\}^M$ of *sufficient statistics*, we are concerned with learning distributions, over the set $\Omega = \{-1, \emptyset, 1\}^m \times \{-1, 1\}$, of the form

$$p_\theta(\lambda, y) = \frac{1}{Z_\theta} \exp(\theta^T \psi(\lambda, y)), \tag{B.1}$$

where $\theta \in \mathbb{R}^M$ is a parameter, and Z_θ is the partition function that makes this a distribution. We assume that we are given, i.e. can derive from the data programming specification, some set Θ of *feasible parameters*. This set must have the following two properties.

First, for any $\theta \in \Theta$, learning the parameter θ from (full) samples from π_θ is possible, at least in some sense. More specifically, there exists an unbiased estimator $\hat{\theta}$ that is a function of some number D samples from p_θ (and is unbiased for all $\theta \in \Theta$) such that, for all $\theta \in \Theta$ and for some $c > 0$,

$$\mathbf{Cov}\left(\hat{\theta}\right) \preceq \frac{I}{2cD}. \tag{B.2}$$

Second, for any $\theta_1, \theta_2 \in \Theta$,

$$\mathbb{E}_{(\lambda_2,y_2)\sim p_{\theta_2}}\left[\mathbf{Var}_{(\lambda_1,y_1)\sim p_{\theta_1}}(y_1|\lambda_1 = \lambda_2)\right] \le \frac{c}{M}. \tag{B.3}$$

That is, we'll always be reasonably certain in our guess for the value of y, even if we are totally wrong about the true parameter θ.

On the other hand, we are also concerned with a distribution $\mathcal{D}$ which ranges over the set $\mathcal{X} \times \{-1, 1\}$, and represents the distribution of training and test examples we are using to learn. These objects are associated with a labeling function $\lambda : \mathcal{X} \mapsto \{-1, \emptyset, 1\}^m$ and a feature function $\phi : \mathcal{X} \mapsto \mathbb{R}^d$. We make three assumptions about this distribution. First, we assume that, given $(x, y) \sim \mathcal{D}$, the class label y is independent of the features $\phi(x)$ given the labels λ. That is,

$$(x, y) \sim \mathcal{D} \Rightarrow y \perp \phi(x) \mid \lambda. \tag{B.4}$$

Second, we assume that we can describe the relationship between λ and y in terms of our family in (B.1) above. That is, for some parameter $\theta^* \in \Theta$,

$$p_{\mathcal{D}}(\lambda, y) = p_{\theta^*}(\lambda, y). \tag{B.5}$$

Third, we assume that the features themselves are bounded; for all $x \in \mathcal{X}$,

$$\|\phi(x)\| \le 1. \tag{B.6}$$

Our goal is twofold. First, we want to recover some estimate $\hat{\theta}$ of the true parameter θ^*. Second, we want to produce a parameter $\hat{w}$ that minimizes the expected regularized logistic loss, or *risk*:

$$R(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log(1 + \exp(-w^T\phi(x)y))\right] + \rho\|w\|^2.$$

We actually accomplish this by minimizing a noise-aware loss function, given our recovered parameter $\hat{\theta}$,

$$R_{\hat{\theta}}(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{\tilde{y}\sim p_{\hat{\theta}}(\cdot|\lambda(x))}\left[\log(1 + \exp(-w^T\phi(x)\tilde{y}))\right]\right] + \rho\|w\|^2.$$

In fact we can't even minimize this; rather, we will be minimizing the empirical noise-aware loss function, which is only this in expectation. Since the analysis of logistic regression is not itself interesting, we assume that we are able to run some algorithm that produces an estimate $\hat{w}$ which satisfies, for some $\chi > 0$,

$$\mathbb{E}\left[R_{\hat{\theta}}(\hat{w}) - \min_{w} R_{\hat{\theta}}(w)\right] \leq \chi. \tag{B.7}$$

The algorithm chosen can be anything, but in practice, we use stochastic gradient descent.

We learn $\hat{\theta}$ and $\hat{w}$ by running the following algorithm.

Algorithm 5 Data Programming

```
Require: Step size $\eta$, dataset $X_U \subset \mathcal{X}$, and initial parameter $\theta_0 \in \Theta$.
  $\theta \to \theta_0$
  for all $x \in X_U$ do
      Independently sample $(\lambda, y)$ from $p_\theta$, and $\tilde{y}$ from $p_\theta(\cdot|\lambda)$.
      $\theta \leftarrow \theta + \eta(\psi(\lambda, y) - \psi(\lambda, \tilde{y}))$.
      $\theta = P_\Theta(\theta)$        ▷ Here, $P_\Theta$ denotes orthogonal projection onto $\Theta$.
  Compute $\hat{w}$ using the algorithm described in (B.6) return $(\theta, \hat{w})$.
```

Under these assumptions, we are able to prove the following theorem about the behavior of Algorithm 5.

Theorem 5. Suppose that we run Algorithm 5 on a data programming specification that satisfies conditions (B.2), (B.3), (B.4), (B.5), (B.6), and (B.7). Suppose further that, for some parameter $\epsilon > 0$, we use step size

$$\eta = \frac{c\epsilon^2}{4}$$

and our dataset is of a size $n = |X_U|$ that satisfies

$$n = \frac{2}{c^2\epsilon^2} \log\left(\frac{2\,\|\theta_0 - \theta^*\|^2}{\epsilon}\right).$$

Then, we can bound the expected parameter error with

$$\mathbb{E}\left[\left\|\hat{\theta} - \theta^*\right\|^2\right] \leq \epsilon^2 M$$

and the expected risk with

$$\mathbb{E}\left[R(\hat{\theta}) - \min_{w} R(w)\right] \leq \chi + \frac{c\epsilon}{2\rho}.$$

This theorem's conclusions and assumptions can readily be seen to be identical to those of Theorem 2 in the main body of the paper, except that they apply to the slightly more general case of arbitrary ψ, rather than ψ of the explicit form described in the body. Therefore, in order to prove Theorem 2, it suffices to prove Theorem 5, which we will do in Section B.3.

B.2 Theoretical Results for Independent Model

For the independent model, we can obtain a more specific version of Theorem 5. In the independent model, the variables are, as before, $\lambda \in \{-1, \emptyset, 1\}^m$ and $y \in \{-1, 1\}$. The sufficient statistics are, letting $\emptyset = 0$ for this subsection, $\lambda_j y$ and λ_j^2.

To produce results that make intuitive sense, we also define the alternate parameterization

$$p(\lambda_j \mid y) = \begin{cases} \beta_j \frac{1+\gamma_j}{2} & \lambda_j = y \\ (1 - \beta_j) & \lambda_j = 0 \\ \beta_j \frac{1-\gamma_j}{2} & \lambda_j = -y \end{cases}.$$

In comparison to the parameters used in the body of the paper, we have

$$\alpha_j = \frac{1 + \gamma_j}{2}.$$

Now, we are concerned with models that are feasible. For a model to be feasible (i.e. for $\theta \in \Theta$), we require that it satisfy, for some constants $\gamma_{\min} > 0$, $\gamma_{\max} > 0$, and $\beta_{\min}$,

$$\gamma_{\min} \leq \gamma_j \leq \gamma_{\max} \qquad \beta_{\min} \leq \beta_j \leq \frac{1}{2}.$$

For $0 \leq \beta \leq 1$ and $-1 \leq \gamma \leq 1$.

For this model, we can prove the following corollary to Theorem 5

Corollary 2. *Suppose that we run Algorithm 5 on an independent data programming specification that satisfies conditions (B.4), (B.5), (B.6), and (B.7). Furthermore, assume that the number of labeling functions we use satisfies*

$$m \geq \frac{9.34 \operatorname{artanh}(\gamma_{\max})}{(\gamma\beta)_{\min}\gamma_{\min}^2} \log\left(\frac{24m}{\beta_{\min}}\right).$$

Suppose further that, for some parameter $\epsilon > 0$, we use step size

$$\eta = \frac{\beta_{\min}\epsilon^2}{16}$$

and our dataset is of a size $n = |X_U|$ that satisfies

$$n = \frac{32}{\beta_{\min}^2\epsilon^2} \log\left(\frac{2\,\|\theta_0 - \theta^*\|^2}{\epsilon}\right).$$

Then, we can bound the expected parameter error with

$$\mathbb{E}\left[\left\|\hat{\theta} - \theta^*\right\|^2\right] \leq \epsilon^2 M$$

and the expected risk with

$$\mathbb{E}\left[R(\hat{w}) - \min_w R(w)\right] \leq \chi + \frac{\beta_{\min}\epsilon}{8\rho}.$$

We can see that if, as stated in the body of the paper, $\beta_j \geq 0.3$ and $0.8 \leq \alpha_j \leq 0.9$ (which is equivalent to $0.6 \leq \gamma_j \leq 0.8$), then

$$2000 \geq 1896.13 = \frac{9.34 \operatorname{artanh}(0.8)}{0.3 \cdot 0.6^3} \log\left(\frac{24 \cdot 2000}{0.3}\right).$$

This means that, as stated in the paper, $m = 2000$ is sufficient for this corollary to hold with

$$n = \frac{32}{0.3^2 \cdot \epsilon^2} \log\left(\frac{2m(\operatorname{artanh}(0.8) - \operatorname{artanh}(0.6))^2}{\epsilon}\right) = \frac{356}{\epsilon^2} \log\left(\frac{m}{3\epsilon}\right).$$

Thus, proving Corollary 2 is sufficient to prove Theorem 1 from the body of the paper. We will prove Corollary 2 in Section B.5

B.3 Proof of Theorem 5

First, we state some lemmas that will be useful in the proof to come.

Lemma 1. Given a family of maximum-entropy distributions

$$p_\theta(x) = \frac{1}{Z_\theta} \exp(\theta^T \psi(x)),$$

for some function of sufficient statistics $\psi : \Omega \mapsto \mathbb{R}^M$, if we let $J : \mathbb{R}^M \mapsto \mathbb{R}$ be the maximum log-likelihood objective for some event $A \subseteq \Omega$,

$$J(\theta) = \log p_\theta(x \in A),$$

then its gradient is

$$\nabla J(\theta) = \mathbb{E}_{x \sim p_\theta}\left[\psi(x) \mid x \in A\right] - \mathbb{E}_{x \sim p_\theta}\left[\psi(x)\right]$$

and its Hessian is

$$\nabla^2 J(\theta) = \mathbf{Cov}_{x \sim p_\theta}\left(\psi(x) | x \in A\right) - \mathbf{Cov}_{x \sim p_\theta}\left(\psi(x)\right).$$

Lemma 2. Suppose that we are looking at a distribution from a data programming label model. That is, our maximum-entropy distribution can now be written in terms of two variables, the labeling function values $\lambda \in \{-1, 0, 1\}$ and the class $y \in \{-1, 1\}$, as

$$p_\theta(\lambda, y) = \frac{1}{Z_\theta} \exp(\theta^T \psi(\lambda, y)),$$

where we assume without loss of generality that for some M, $\psi(\lambda, y) \in \mathbb{R}^M$ and $\|\psi(\lambda, y)\|_\infty \leq 1$. If we let $J : \mathbb{R}^M \mapsto \mathbb{R}$ be the maximum expected log-likelihood objective, under another distribution $\mathcal{D}$, for the event associated with the observed labeling function values λ,

$$J(\theta) = \mathbb{E}_{(\lambda^*, y^*) \sim \mathcal{D}}\left[\log p_\theta(\lambda^*)\right],$$

then its Hessian can be bounded with

$$\nabla^2 J(\theta) \preceq MI\mathbb{E}_{(\lambda^*,y^*)\sim\mathcal{D}}\left[\mathbf{Var}_{(\lambda,y)\sim p_\theta}\left(y|\lambda=\lambda^*\right)\right] - \mathcal{I}(\theta),$$

where $\mathcal{I}(\theta)$ is the Fisher information.

Lemma 3. Suppose that we are looking at a data programming distribution, as described in the text of Lemma 2. Suppose further that we are concerned with some feasible set of parameters $\Theta \subset \mathbb{R}^M$, such that the any model with parameters in this space satisfies the following two conditions.

First, for any $\theta \in \Theta$, learning the parameter θ from (full) samples from p_θ is possible, at least in some sense. More specifically, there exists an unbiased estimator $\hat{\theta}$ that is a function of some number D samples from p_θ (and is unbiased for all $\theta \in \Theta$) such that, for all $\theta \in \Theta$ and for some $c > 0$,

$$\mathbf{Cov}\left(\hat{\theta}\right) \preceq \frac{I}{2cD}.$$

Second, for any $\theta, \theta^* \in \Theta$,

$$\mathbb{E}_{(\lambda^*,y^*)\sim\mathcal{D}}\left[\mathbf{Var}_{(\lambda,y)\sim p_\theta}\left(y|\lambda=\lambda^*\right)\right] \leq \frac{c}{M}.$$

That is, we'll always be reasonably certain in our guess for the value of y, even if we are totally wrong about the true parameter θ^*.

Under these conditions, the function J is strongly concave on Θ with parameter of strong convexity c.

Lemma 4. Suppose that we are looking at a data programming maximum likelihood estimation problem, as described in the text of Lemma 2. Suppose further that the objective function J is strongly concave with parameter $c > 0$.

If we run stochastic gradient descent on objective J, using unbiased samples from a true distribution p_{θ^*}, where $\theta^* \in \Theta$, then if we use step size

$$\eta = \frac{c\epsilon^2}{4}$$

and run (using a fresh sample at each iteration) for T steps, where

$$T = \frac{2}{c^2\epsilon^2} \log\left(\frac{2\left\|\theta_0 - \theta^*\right\|^2}{\epsilon}\right)$$

then we can bound the expected parameter estimation error with

$$\mathbb{E}\left[\left\|\hat{\theta} - \theta^*\right\|^2\right] \leq \epsilon^2 M.$$

Lemma 5. Assume in our model that, without loss of generality, $\|\phi(x)\| \leq 1$ for all x, and that in our true model $\mathcal{D}$, the class y is independent of the features $\phi(x)$ given the labels λ.

Suppose that we now want to solve the expected loss minimization problem wherein we minimize the objective

$$R(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log(1 + \exp(-w^T\phi(x)y))\right] + \rho\left\|w\right\|^2.$$

We actually accomplish this by minimizing our noise-aware loss function, given our chosen parameter $\hat{\theta}$,

$$R_{\hat{\theta}}(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{\tilde{y}\sim p_{\hat{\theta}}(\cdot|\lambda)}\left[\log(1 + \exp(-w^T\phi(x)\tilde{y}))\right]\right] + \rho\left\|w\right\|^2.$$

In fact we can't even minimize this; rather, we will be minimizing the empirical noise-aware loss function, which is only this in expectation. Suppose that doing so produces an estimate $\hat{w}$ which satisfies, for some $\chi > 0$,

$$\mathbb{E}\left[R_{\hat{\theta}}(\hat{w}) - \min_w R_{\hat{\theta}}(w)\middle|\hat{\theta}\right] \leq \chi.$$

(Here, the expectation is taken with respect to only the random variable $\hat{w}$.) Then, we can bound the expected risk with

$$\mathbb{E}\left[R(\hat{w}) - \min_w R(w)\right] \leq \chi + \frac{c\epsilon}{2\rho}.$$

Now, we restate and prove our main theorem.

Theorem 5. Suppose that we run Algorithm 5 on a data programming specification that satisfies conditions (B.2), (B.3), (B.4), (B.5), (B.6), and (B.7). Suppose further that, for some parameter $\epsilon > 0$, we use step size

$$\eta = \frac{c\epsilon^2}{4}$$

and our dataset is of a size $n = |X_U|$ that satisfies

$$n = \frac{2}{c^2\epsilon^2} \log\left(\frac{2\,\|\theta_0 - \theta^*\|^2}{\epsilon}\right).$$

Then, we can bound the expected parameter error with

$$\mathbb{E}\left[\left\|\hat{\theta} - \theta^*\right\|^2\right] \leq \epsilon^2 M$$

and the expected risk with

$$\mathbb{E}\left[R(\hat{\theta}) - \min_w R(w)\right] \leq \chi + \frac{c\epsilon}{2\rho}.$$

Proof. The bounds on the expected parameter estimation error follow directly from Lemma 4, and the remainder of the theorem follows directly from Lemma 5. □

B.4 Proofs of Lemmas

Lemma 1. Given a family of maximum-entropy distributions

$$p_\theta(x) = \frac{1}{Z_\theta} \exp(\theta^T \psi(x)),$$

for some function of sufficient statistics $\psi : \Omega \mapsto \mathbb{R}^M$, if we let $J : \mathbb{R}^M \mapsto \mathbb{R}$ be the maximum log-likelihood objective for some event $A \subseteq \Omega$,

$$J(\theta) = \log p_\theta(x \in A),$$

then its gradient is

$$\nabla J(\theta) = \mathbb{E}_{x\sim p_\theta}\left[\psi(x) \mid x \in A\right] - \mathbb{E}_{x\sim p_\theta}\left[\psi(x)\right]$$

and its Hessian is

$$\nabla^2 J(\theta) = \mathbf{Cov}_{x\sim p_\theta}\left(\psi(x) | x \in A\right) - \mathbf{Cov}_{x\sim p_\theta}\left(\psi(x)\right).$$

Proof. For the gradient,

$$\begin{aligned}
\nabla J(\theta) &= \nabla \log p_\theta(A) \\
&= \nabla \log\left(\frac{\sum_{x\in A}\exp(\theta^T\psi(x))}{\sum_{x\in\Omega}\exp(\theta^T\psi(x))}\right) \\
&= \nabla \log\left(\sum_{x\in A}\exp(\theta^T\psi(x))\right) - \nabla \log\left(\sum_{x\in\Omega}\exp(\theta^T\psi(x))\right) \\
&= \frac{\sum_{x\in A}\psi(x)\exp(\theta^T\psi(x))}{\sum_{x\in A}\exp(\theta^T\psi(x))} - \frac{\sum_{x\in\Omega}\psi(x)\exp(\theta^T\psi(x))}{\sum_{x\in\Omega}\exp(\theta^T\psi(x))} \\
&= \mathbb{E}_{x\sim p_\theta(\cdot|x\in A)}\left[\psi(x)\right] - \mathbb{E}_{x\sim p_\theta}\left[\psi(x)\right].
\end{aligned}$$

And for the Hessian,

$$\begin{aligned}
\nabla^2 J(\theta) &= \nabla\frac{\sum_{x\in A}\psi(x)\exp(\theta^T\psi(x))}{\sum_{x\in A}\exp(\theta^T\psi(x))} - \nabla\frac{\sum_{x\in\Omega}\psi(x)\exp(\theta^T\psi(x))}{\sum_{x\in\Omega}\exp(\theta^T\psi(x))} \\
&= \frac{\sum_{x\in A}\psi(x)\psi(x)^T\exp(\theta^T\psi(x))}{\sum_{x\in A}\exp(\theta^T\psi(x))} - \frac{\left(\sum_{x\in A}\psi(x)\exp(\theta^T\psi(x))\right)\left(\sum_{x\in A}\psi(x)\exp(\theta^T\psi(x))\right)^T}{\left(\sum_{x\in A}\exp(\theta^T\psi(x))\right)^2} \\
&\quad - \left(\frac{\sum_{x\in\Omega}\psi(x)\psi(x)^T\exp(\theta^T\psi(x))}{\sum_{x\in\Omega}\exp(\theta^T\psi(x))} - \frac{\left(\sum_{x\in\Omega}\psi(x)\exp(\theta^T\psi(x))\right)\left(\sum_{x\in\Omega}\psi(x)\exp(\theta^T\psi(x))\right)^T}{\left(\sum_{x\in\Omega}\exp(\theta^T\psi(x))\right)^2}\right) \\
&= \mathbb{E}_{x\sim p_\theta(\cdot|x\in A)}\left[\psi(x)\psi(x)^T\right] - \mathbb{E}_{x\sim p_\theta(\cdot|x\in A)}\left[\psi(x)\right]\mathbb{E}_{x\sim p_\theta(\cdot|x\in A)}\left[\psi(x)\right] \\
&\quad - \left(\mathbb{E}_{x\sim p_\theta}\left[\psi(x)\psi(x)^T\right] - \mathbb{E}_{x\sim p_\theta}\left[\psi(x)\right]\mathbb{E}_{x\sim p_\theta}\left[\psi(x)\right]^T\right) \\
&= \mathbf{Cov}_{x\sim p_\theta(\cdot|x\in A)}\left(\psi(x)\right) - \mathbf{Cov}_{x\sim p_\theta}\left(\psi(x)\right).
\end{aligned}$$

□

Lemma 2. Suppose that we are looking at a distribution from a data programming label model. That is, our maximum-entropy distribution can now be written in terms of two variables, the labeling function values $\lambda \in \{-1, 0, 1\}$ and the class $y \in \{-1, 1\}$, as

$$p_\theta(\lambda, y) = \frac{1}{Z_\theta} \exp(\theta^T \psi(\lambda, y)),$$

where we assume without loss of generality that for some M, $\psi(\lambda, y) \in \mathbb{R}^M$ and $\|\psi(\lambda, y)\|_\infty \leq 1$. If we let $J : \mathbb{R}^M \mapsto \mathbb{R}$ be the maximum expected log-likelihood objective, under another distribution $\mathcal{D}$, for the event associated with the observed labeling function values λ,

$$J(\theta) = \mathbb{E}_{(\lambda^*, y^*) \sim \mathcal{D}} \left[\log p_\theta(\lambda^*)\right],$$

then its Hessian can be bounded with

$$\nabla^2 J(\theta) \preceq MI\mathbb{E}_{(\lambda^*, y^*) \sim \mathcal{D}} \left[\mathbf{Var}_{(\lambda, y) \sim p_\theta} (y | \lambda = \lambda^*)\right] - \mathcal{I}(\theta),$$

where $\mathcal{I}(\theta)$ is the Fisher information.

Proof. From the result of Lemma 1, we have that

$$\nabla^2 J(\theta) = \mathbb{E}_{(\lambda^*, y^*) \sim \mathcal{D}} \left[\mathbf{Cov}_{(\lambda, y) \sim p_\theta} (\psi(\lambda, y) | \lambda = \lambda^*)\right] - \mathbf{Cov}_{(\lambda, y) \sim p_\theta} (\psi(\lambda, y)). \tag{B.8}$$

We start by defining $\psi_0(\lambda)$ and $\psi_1(\lambda)$ such that

$$\psi(\lambda, y) = \psi(\lambda, 1)\frac{1 + y}{2} + \psi(\lambda, -1)\frac{1 - y}{2} = \frac{\psi(\lambda, 1) + \psi(\lambda, -1)}{2} + y\frac{\psi(\lambda, 1) - \psi(\lambda, -1)}{2} = \psi_0(\lambda) + y\psi_1(\lambda).$$

This allows us to reduce (B.8) to

$$\nabla^2 J(\theta) = \mathbb{E}_{(\lambda^*, y^*) \sim \mathcal{D}} \left[\psi_1(\lambda^*)\psi_1(\lambda^*)^T \mathbf{Var}_{(\lambda, y) \sim p_\theta} (y | \lambda = \lambda^*)\right] - \mathbf{Cov}_{(\lambda, y) \sim p_\theta} (\psi(\lambda, y)).$$

On the other hand, the Fisher information of this model at θ is

$$
\begin{aligned}
\mathcal{I}(\theta) &= \mathbb{E}\left[(\nabla_\theta \log p_\theta(z))^2\right] \\
&= \mathbb{E}\left[\left(\nabla_\theta \log\left(\frac{\exp(\theta^T\psi(z))}{\sum_{z'\in\Omega}\exp(\theta^T\psi(z'))}\right)\right)^2\right] \\
&= \mathbb{E}\left[\left(\nabla_\theta \log\left(\exp(\theta^T\psi(z))\right) - \nabla_\theta \log\left(\sum_{z'\in\Omega}\exp(\theta^T\psi(z'))\right)\right)^2\right] \\
&= \mathbb{E}\left[\left(\psi(z) - \frac{\sum_{z'\in\Omega}\psi(z')\exp(\theta^T\psi(z'))}{\sum_{z'\in\Omega}\exp(\theta^T\psi(z'))}\right)^2\right] \\
&= \mathbb{E}\left[(\psi(z) - \mathbb{E}\left[\psi(z')\right])^2\right] \\
&= \mathbf{Cov}\left(\psi(z)\right).
\end{aligned}
$$

Therefore, we can write the second derivative of J as

$$
\nabla^2 J(\theta) = \mathbb{E}_{(\lambda^*,y^*)\sim\mathcal{D}}\left[\psi_1(\lambda^*)\psi_1(\lambda^*)^T \mathbf{Var}_{(\lambda,y)\sim p_\theta}\left(y|\lambda=\lambda^*\right)\right] - \mathcal{I}(\theta).
$$

If we apply the fact that

$$
\psi_1(\lambda^*)\psi_1(\lambda^*)^T \preceq I\,\|\psi_1(\lambda^*)\|^2 \preceq MI\,\|\psi_1(\lambda^*)\|_\infty^2 \preceq MI,
$$

then we can reduce this to

$$
\nabla^2 J(\theta) \preceq MI\mathbb{E}_{(\lambda^*,y^*)\sim\mathcal{D}}\left[\mathbf{Var}_{(\lambda,y)\sim p_\theta}\left(y|\lambda=\lambda^*\right)\right] - \mathcal{I}(\theta).
$$

This is the desired result. □

Lemma 3. Suppose that we are looking at a data programming distribution, as described in the text of Lemma 2. Suppose further that we are concerned with some feasible set of parameters $\Theta \subset \mathbb{R}^M$, such that the any model with parameters in this space satisfies the following two conditions.

First, for any $\theta \in \Theta$, learning the parameter θ from (full) samples from p_θ is possible, at least in some sense. More specifically, there exists an unbiased estimator $\hat{\theta}$ that is a function of some number D samples from p_θ (and is unbiased for all $\theta \in \Theta$) such that, for all $\theta \in \Theta$ and for some $c > 0$,

$$\mathbf{Cov}\left(\hat{\theta}\right) \preceq \frac{I}{2cD}.$$

Second, for any $\theta, \theta^* \in \Theta$,

$$\mathbb{E}_{(\lambda^*,y^*)\sim\mathcal{D}}\left[\mathbf{Var}_{(\lambda,y)\sim p_\theta}\left(y|\lambda = \lambda^*\right)\right] \leq \frac{c}{M}.$$

That is, we'll always be reasonably certain in our guess for the value of y, even if we are totally wrong about the true parameter θ^*.

Under these conditions, the function J is strongly concave on Θ with parameter of strong convexity c.

Proof. From the Cramér-Rao bound, we know in general that the variance of any unbiased estimator is bounded by the reciprocal of the Fisher information

$$\mathbf{Cov}\left(\hat{\theta}\right) \succeq \left(\mathcal{I}(\theta)\right)^{-1}.$$

Since for the estimator described in the lemma statement, we have D independent samples from the distribution, it follows that the Fisher information of this experiment is D times the Fisher information of a single sample. Combining this with the bound in the lemma statement on the covariance, we get

$$\frac{I}{2cD} \succeq \mathbf{Cov}\left(\hat{\theta}\right) \succeq \left(D\mathcal{I}(\theta)\right)^{-1}.$$

It follows that

$$\mathcal{I}(\theta) \succeq 2cI.$$

On the other hand, also from the lemma statement, we can conclude that

$$MI\mathbb{E}_{(\lambda^*,y^*)\sim\mathcal{D}}\left[\mathbf{Var}_{(\lambda,y)\sim p_\theta}\left(y|\lambda = \lambda^*\right)\right] \preceq cI.$$

Therefore, for all $\theta \in \Theta$,

$$\nabla^2 J(\theta) \preceq MI \mathbb{E}_{(\lambda^*, y^*) \sim \mathcal{D}} \left[\mathbf{Var}_{(\lambda, y) \sim p_\theta} (y | \lambda = \lambda^*) \right] - \mathcal{I}(\theta) \preceq -cI.$$

This implies that J is strongly concave over Θ, with constant c, as desired. □

Lemma 4. Suppose that we are looking at a data programming maximum likelihood estimation problem, as described in the text of Lemma 2. Suppose further that the objective function J is strongly concave with parameter $c > 0$.

If we run stochastic gradient descent on objective J, using unbiased samples from a true distribution p_{θ^*}, where $\theta^* \in \Theta$, then if we use step size

$$\eta = \frac{c\epsilon^2}{4}$$

and run (using a fresh sample at each iteration) for T steps, where

$$T = \frac{2}{c^2 \epsilon^2} \log\left(\frac{2 \|\theta_0 - \theta^*\|^2}{\epsilon} \right)$$

then we can bound the expected parameter estimation error with

$$\mathbb{E}\left[\left\| \hat{\theta} - \theta^* \right\|^2 \right] \leq \epsilon^2 M.$$

Proof. First, we note that, in the proof to follow, we can ignore the projection onto the feasible set Θ, since this projection always takes us closer to the optimum θ^*.

If we track the expected distance to the optimum θ^*, then at the next time step,

$$\|\theta_{t+1} - \theta^*\|^2 = \|\theta_t - \theta^*\|^2 + 2\gamma(\theta_t - \theta^*) \nabla \tilde{J}(\theta_t) + \gamma^2 \left\| \nabla \tilde{J}_t(\theta_t) \right\|^2.$$

Since we can write our stochastic samples in the form

$$\nabla \tilde{J}_t(\theta_t) = \psi(\lambda_t, y_t) - \psi(\bar{\lambda}_t, \bar{y}_t),$$

for some samples λ_t, y_t, $\bar{\lambda}_t$, and $\bar{y}_t$, we can conclude that

$$\left\|\nabla \tilde{J}_t(\theta_t)\right\|^2 \leq M \left\|\nabla \tilde{J}_t(\theta_t)\right\|_\infty^2 \leq 4M.$$

Therefore, taking the expected value conditioned on the filtration,

$$\mathbb{E}\left[\|\theta_{t+1} - \theta^*\|^2 \mid \mathcal{F}_t\right] = \|\theta_t - \theta^*\|^2 + 2\gamma(\theta_t - \theta^*)\nabla J(\theta_t) + 4\gamma^2 M.$$

Since J is strongly concave,

$$(\theta_t - \theta^*)\nabla J(\theta_t) \leq -c \|\theta_t - \theta^*\|^2 ;$$

and so,

$$\mathbb{E}\left[\|\theta_{t+1} - \theta^*\|^2 \mid \mathcal{F}_t\right] \leq (1 - 2\gamma c) \|\theta_t - \theta^*\|^2 + 4\gamma^2 M.$$

If we take the full expectation and subtract the fixed point from both sides,

$$\mathbb{E}\left[\|\theta_{t+1} - \theta^*\|^2\right] - \frac{2\gamma M}{c} \leq (1-2\gamma c)\mathbb{E}\left[\|\theta_t - \theta^*\|^2\right] + 4\gamma^2 M - \frac{2\gamma M}{c} = (1-2\gamma c)\left(\mathbb{E}\left[\|\theta_t - \theta^*\|^2\right] - \frac{2\gamma M}{c}\right).$$

Therefore,

$$\mathbb{E}\left[\|\theta_t - \theta^*\|^2\right] - \frac{2\gamma M}{c} \leq (1 - 2\gamma c)^t \left(\|\theta_0 - \theta^*\|^2 - \frac{2\gamma M}{c}\right),$$

and so

$$\mathbb{E}\left[\|\theta_t - \theta^*\|^2\right] \leq \exp(-2\gamma c t) \|\theta_0 - \theta^*\|^2 + \frac{2\gamma M}{c}.$$

In order to ensure that

$$\mathbb{E}\left[\|\theta_t - \theta^*\|^2\right] \leq \epsilon^2,$$

it therefore suffices to pick

$$\gamma = \frac{c\epsilon^2}{4M}$$

and

$$t = \frac{2M}{c^2\epsilon^2} \log\left(\frac{2\|\theta_0 - \theta^*\|^2}{\epsilon}\right).$$

Substituting $\epsilon^2 \to \epsilon^2 M$ produces the desired result. $\square$

Lemma 5. Assume in our model that, without loss of generality, $||\phi(x)|| \leq 1$ for all x, and that in our true model $\mathcal{D}$, the class y is independent of the features $\phi(x)$ given the labels λ.

Suppose that we now want to solve the expected loss minimization problem wherein we minimize the objective

$$R(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log(1 + \exp(-w^T\phi(x)y))\right] + \rho\,||w||^2 .$$

We actually accomplish this by minimizing our noise-aware loss function, given our chosen parameter $\hat{\theta}$,

$$R_{\hat{\theta}}(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{\tilde{y}\sim p_{\hat{\theta}}(\cdot|\lambda)}\left[\log(1 + \exp(-w^T\phi(x)\tilde{y}))\right]\right] + \rho\,||w||^2 .$$

In fact we can't even minimize this; rather, we will be minimizing the empirical noise-aware loss function, which is only this in expectation. Suppose that doing so produces an estimate $\hat{w}$ which satisfies, for some $\chi > 0$,

$$\mathbb{E}\left[R_{\hat{\theta}}(\hat{w}) - \min_w R_{\hat{\theta}}(w)\right]\hat{\theta} \leq \chi.$$

(Here, the expectation is taken with respect to only the random variable $\hat{w}$.) Then, we can bound the expected risk with

$$\mathbb{E}\left[R(\hat{w}) - \min_w R(w)\right] \leq \chi + \frac{c\epsilon}{2\rho}.$$

Proof. (To simplify the symbols in this proof, we freely use θ when we mean $\hat{\theta}$.)

The loss function we want to minimize is, in expectation,

$$R(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log(1 + \exp(-w^T\phi(x)y))\right] + \rho\,||w||^2 .$$

By the law of total expectation,

$$R(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{(\bar{x},\bar{y})\sim\mathcal{D}}\left[\log(1 + \exp(-w^T\phi(x)\bar{y}))\right]\right] + \rho\,||w||^2 ,$$

Since we know from our assumptions that, for the optimum parameter θ^*,

$$p_{\mathcal{D}}(\lambda, y) = p_{\theta^*}(\lambda, y),$$

and given our conditional independence assumption, we can rewrite this as

$$R(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{\tilde{y}\sim p_{\theta^*}(\cdot|\lambda)}\left[\log(1+\exp(-w^T\phi(x)\tilde{y}))\right]\right] + \rho\|w\|^2.$$

On the other hand, if we are minimizing the model we got from the previous step, we will be actually minimizing

$$R_\theta(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{\tilde{y}\sim p_{\theta}(\cdot|\lambda)}\left[\log(1+\exp(-w^T\phi(x)\tilde{y}))\right]\right] + \rho\|w\|^2.$$

We can reduce this further by noticing that

$$\begin{aligned}
&\mathbb{E}_{\tilde{y}\sim p_\theta(\cdot|\lambda)}\left[\log(1+\exp(-w^T\phi(x)\tilde{y}))\right] \\
&\quad= \mathbb{E}_{\tilde{y}\sim p_\theta(\cdot|\lambda)}\left[\log(1+\exp(-w^T\phi(x)))\frac{1+\tilde{y}}{2} + \log(1+\exp(w^T\phi(x)))\frac{1-\tilde{y}}{2}\right] \\
&\quad= \frac{\log(1+\exp(-w^T\phi(x))) + \log(1+\exp(w^T\phi(x)))}{2} \\
&\qquad\quad + \frac{\log(1+\exp(-w^T\phi(x))) - \log(1+\exp(w^T\phi(x)))}{2}\mathbb{E}_{\tilde{y}\sim p_\theta(\cdot|\lambda)}\left[\tilde{y}\right] \\
&\quad= \frac{\log(1+\exp(-w^T\phi(x))) + \log(1+\exp(w^T\phi(x)))}{2} \\
&\qquad\quad - \frac{w^T\phi(x)}{2}\mathbb{E}_{\tilde{y}\sim p_\theta(\cdot|\lambda)}\left[\tilde{y}\right].
\end{aligned}$$

It follows that the difference between the loss functions will be

$$|R(w) - R_\theta(w)| = \left|\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\frac{w^T\phi(x)}{2}\left(\mathbb{E}_{\tilde{y}\sim p_\theta(\cdot|\lambda)}\left[\tilde{y}\right] - \mathbb{E}_{\tilde{y}^*\sim p_{\theta^*}(\cdot|\lambda)}\left[\tilde{y}^*\right]\right)\right]\right|.$$

Now, we can compute that

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\tilde{y}\sim p_\theta(\cdot|\lambda)}\left[\tilde{y}\right] &= \nabla_\theta \frac{\exp(\theta^T\psi(\lambda,1)) - \exp(\theta^T\psi(\lambda,-1))}{\exp(\theta^T\psi(\lambda,1)) + \exp(\theta^\psi(\lambda,-1))} \\
&= \nabla_\theta \frac{\exp(\theta^T\psi_1(\lambda)) - \exp(-\theta^T\psi_1(\lambda))}{\exp(\theta^T\psi_1(\lambda)) + \exp(\theta^T\psi_1(\lambda))} \\
&= \nabla_\theta \tanh(\theta^T\psi_1(\lambda)) \\
&= \psi_1(\lambda)\left(1 - \tanh^2(\theta^T\psi_1(\lambda))\right) \\
&= \psi_1(\lambda)\mathbf{Var}_{\tilde{y}\sim p_\theta(\cdot|\lambda)}\left(\tilde{y}|.\right)
\end{aligned}$$

It follows by the mean value theorem that for some $\tilde{\theta}$, a linear combination of θ and θ^*,

$$|R(w) - R_\theta(w)| = \left|\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\frac{w^T\phi(x)}{2}(\theta-\theta^*)^T\psi_1(\lambda)\mathbf{Var}_{\tilde{y}\sim p_{\tilde{\theta}}(\cdot|\lambda)}\left(\tilde{y}\right)\right]\right|.$$

Since Θ is convex, clearly $\tilde{\theta} \in \Theta$. From our assumption on the bound of the variance, we can conclude that

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbf{Var}_{\tilde{y}\sim p_{\tilde{\theta}}(\cdot|\lambda)}\left(\tilde{y}\right)\right] \le \frac{c}{M}.$$

By the Cauchy-Schwarz inequality,

$$|R(w) - R_\theta(w)| \le \frac{1}{2}\left|\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\|w\|\,\|\phi(x)\|\,\|\theta-\theta^*\|\,\|\psi_1(\lambda)\|\,\mathbf{Var}_{\tilde{y}\sim p_{\tilde{\theta}}(\cdot|\lambda)}\left(\tilde{y}\right)\right]\right|.$$

Since (by assumption) $\|\phi(x)\| \le 1$ and $\|\psi_1(\lambda)\| \le \sqrt{M}$,

$$\begin{aligned}
|R(w) - R_\theta(w)| &\le \frac{\|w\|\,\|\theta-\theta^*\|\,\sqrt{M}}{2}\left|\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbf{Var}_{\tilde{y}\sim p_{\tilde{\theta}}(\cdot|\lambda)}\left(\tilde{y}\right)\right]\right| \\
&\le \frac{\|w\|\,\|\theta-\theta^*\|\,\sqrt{M}}{2}\cdot\frac{c}{M} \\
&= \frac{c\,\|w\|\,\|\theta-\theta^*\|}{2\sqrt{M}}.
\end{aligned}$$

Now, for any w that could conceivably be a solution, it must be the case that

$$\|w\| \le \frac{1}{2\rho},$$

since otherwise the regularization term would be too large Therefore, for any possible solution w,

$$|R(w) - R_\theta(w)| \le \frac{c\,\|\theta - \theta^*\|}{4\rho\sqrt{M}}.$$

Now, we apply the assumption that we are able to solve the empirical problem, producing an estimate $\hat{w}$ that satisfies

$$\mathbb{E}\left[R_\theta(\hat{w}) - R_\theta(w^*_\theta)\right] \le \chi,$$

where w^*_θ is the true solution to

$$w^*_\theta = \arg\min_w R_\theta(w).$$

Therefore,

$$\begin{aligned}
\mathbb{E}\left[R(\hat{w}) - R(w^*)\right] &= \mathbb{E}\left[R_\theta(\hat{w}) - R_\theta(w^*_\theta) + R_\theta(w^*_\theta) - R_\theta(\hat{w}) + R(\hat{w}) - R(w^*)\right] \\
&\le \chi + \mathbb{E}\left[R_\theta(w^*) - R_\theta(\hat{w}) + R(\hat{w}) - R(w^*)\right] \\
&\le \chi + \mathbb{E}\left[|R_\theta(w^*) - R(w^*)| + |R_\theta(\hat{w}) - R(\hat{w})|\right] \\
&\le \chi + \mathbb{E}\left[\frac{c\,\|\theta - \theta^*\|}{2\rho\sqrt{M}}\right] \\
&= \chi + \frac{c}{2\rho\sqrt{M}}\mathbf{E}\left[\|\theta - \theta^*\|\right] \\
&\le \chi + \frac{c}{2\rho\sqrt{M}}\sqrt{\mathbb{E}\left[\|\theta - \theta^*\|^2\right]}.
\end{aligned}$$

We can now bound this using the result of Lemma 4, which results in

$$\begin{aligned}
\mathbb{E}\left[l(\hat{w}) - l(w^*)\right] &\le \chi + \frac{c}{2\rho\sqrt{M}}\sqrt{M\epsilon^2} \\
&= \chi + \frac{c\epsilon}{2\rho}.
\end{aligned}$$

This is the desired result. □

B.5 Proofs of Results for the Independent Model

To restate, in the independent model, the variables are, as before, $\lambda \in \{-1, \emptyset, 1\}^m$ and $y \in \{-1, 1\}$, where for simplicity in this section we let $\emptyset = 0$. The sufficient statistics are $\lambda_j y$ and λ_j^2. That is, for expanded parameter $\theta = (\theta^{\text{acc}}, \theta^{\text{cov}})$,

$$p_\theta(\lambda, y) = \frac{1}{Z_\theta} \exp((\theta^{\text{acc}})^T \lambda y + (\theta^{\text{cov}})^T \lambda^2).$$

This can be combined with the simple assumption that $P(y) = \frac{1}{2}$ to complete a whole distribution. Using this, we can prove the following simple result about the moments of the sufficient statistics.

Lemma 6. The expected values and covariances of the sufficient statistics are, for all $j \neq i$,

$$\begin{aligned}
\mathbf{E}\left[\lambda_j y\right] &= \beta_j \gamma_j \\
\mathbf{E}\left[\lambda_j^2\right] &= \beta_j \\
\mathbf{Var}\left(\lambda_j y\right) &= \beta_j - \beta_j^2 \gamma_j^2 \\
\mathbf{Var}\left(\lambda_j^2\right) &= \beta_j - \beta_j^2 \\
\mathbf{Cov}\left(\lambda_j y, \lambda_i y\right) &= 0 \\
\mathbf{Cov}\left(\lambda_j^2, \lambda_i^2\right) &= 0 \\
\mathbf{Cov}\left(\lambda_j y, \lambda_i^2\right) &= 0.
\end{aligned}$$

We also prove the following basic lemma that relates θ_j^{acc} to γ_i.

Lemma 7. It holds that

$$\gamma_j = \tanh(\theta_j^{\text{acc}}).$$

We also make the following claim about feasible models.

Lemma 8. For any feasible model, it will be the case that, for any other feasible parameter vector $\hat{\theta}$,

$$p\left((\hat{\theta}^{\text{acc}})^T \lambda y \leq \frac{m}{2} \gamma_{\min} (\gamma\beta)_{\min}\right) \leq \exp\left(-\frac{m(\gamma\beta)_{\min} \gamma_{\min}^2}{9.34 \operatorname{artanh}(\gamma_{\max})}\right).$$

We can also prove the following simple result about the conditional covariances

Lemma 9. The covariances of the sufficient statistics, conditioned on λ, are for all $j \neq j$,

$$\begin{aligned} \mathbf{Cov}\left(\lambda_j y, \lambda_i y \mid \lambda\right) &= \lambda_j \lambda_i \operatorname{sech}^2((\theta^{\text{acc}})^T \lambda) \\ \mathbf{Cov}\left(\lambda_j^2, \lambda_i^2 \mid \lambda\right) &= 0. \end{aligned}$$

We can combine these two results to bound the expected variance of these conditional statistics.

Lemma 10. If θ and θ^* are two feasible models, then for any u,

$$\mathbb{E}_{\theta^*}\left[\mathbf{Var}_\theta\left(y \mid \lambda\right)\right] \leq 3 \exp\left(-\frac{m\beta_{\min}^2 \gamma_{\min}^3}{8 \operatorname{artanh}(\gamma_{\max})}\right).$$

We can now proceed to restate and prove the main corollary of Theorem 5 that applies in the independent case.

Corollary 2. *Suppose that we run Algorithm 5 on an independent data programming specification that satisfies conditions (B.4), (B.5), (B.6), and (B.7). Furthermore, assume that the number of labeling functions we use satisfies*

$$m \geq \frac{9.34 \operatorname{artanh}(\gamma_{\max})}{(\gamma\beta)_{\min}\gamma_{\min}^2} \log\left(\frac{24m}{\beta_{\min}}\right).$$

Suppose further that, for some parameter $\epsilon > 0$, we use step size

$$\eta = \frac{\beta_{\min}\epsilon^2}{16}$$

and our dataset is of a size $n = |X_U|$ that satisfies

$$n = \frac{32}{\beta_{\min}^2 \epsilon^2} \log\left(\frac{2\left\|\theta_0 - \theta^*\right\|^2}{\epsilon}\right).$$

Then, we can bound the expected parameter error with

$$\mathbb{E}\left[\left\|\hat{\theta} - \theta^*\right\|^2\right] \leq \epsilon^2 M$$

and the expected risk with

$$\mathbb{E}\left[R(\hat{w}) - \min_{w} R(w)\right] \leq \chi + \frac{\beta_{\min}\epsilon}{8\rho}.$$

Proof. In order to apply Theorem 5, we have to verify all its conditions hold in the independent case.

First, we notice that (B.2) is used only to bound the covariance of the sufficient statistics. From Lemma 6, we know that these can be bounded by $\beta_j - \beta_j^2\gamma_j^2 \geq \frac{\beta_{min}}{2}$. It follows that we can choose

$$c = \frac{\beta_{\min}}{4},$$

and we can consider (B.2) satisfied, for the purposes of applying the theorem.

Second, to verify (B.3), we can use Lemma 10. For this to work, we need

$$3\exp\left(-\frac{m(\gamma\beta)_{\min}\gamma_{\min}^2}{9.34\,\mathrm{artanh}(\gamma_{\max})}\right) \leq \frac{c}{M} = \frac{\beta_{\min}}{8m}.$$

This happens whenever the number of labeling functions satisfies

$$m \geq \frac{9.34\,\mathrm{artanh}(\gamma_{\max})}{(\gamma\beta)_{\min}\gamma_{\min}^2}\log\left(\frac{24m}{\beta_{\min}}\right).$$

The remaining assumptions, (B.4), (B.5), (B.6), and (B.7), are satisfied directly by the assumptions of this corollary. So, we can apply Theorem 5, which produces the desired result. □

B.6 Proofs of Independent Model Lemmas

Lemma 6. The expected values and covariances of the sufficient statistics are, for all $j \neq i$,

$$\begin{aligned}
\mathbf{E}\left[\lambda_j y\right] &= \beta_j \gamma_j \\
\mathbf{E}\left[\lambda_j^2\right] &= \beta_j \\
\mathbf{Var}\left(\lambda_j y\right) &= \beta_j - \beta_j^2 \gamma_j^2 \\
\mathbf{Var}\left(\lambda_j^2\right) &= \beta_j - \beta_j^2 \\
\mathbf{Cov}\left(\lambda_j y, \lambda_i y\right) &= 0 \\
\mathbf{Cov}\left(\lambda_j^2, \lambda_i^2\right) &= 0 \\
\mathbf{Cov}\left(\lambda_j y, \lambda_i^2\right) &= 0.
\end{aligned}$$

Proof. We prove each of the statements in turn. For the first statement,

$$\begin{aligned}
\mathbb{E}\left[\lambda_j y\right] &= p_\theta(\lambda_j = y) - p_\theta(\lambda_j \neq y) \\
&= \beta_j \frac{1+\gamma_j}{2} - \beta_j \frac{1-\gamma_j}{2} \\
&= \beta_j \gamma_j.
\end{aligned}$$

For the second statement,

$$\begin{aligned}
\mathbb{E}\left[\lambda_j^2\right] &= p_\theta(\lambda_j = y) + p_\theta(\lambda_j = -y) \\
&= \beta_j \frac{1+\gamma_j}{2} + \beta_j \frac{1-\gamma_j}{2} \\
&= \beta_j.
\end{aligned}$$

For the remaining statements, we derive the second moments; converting these to an expression of the covariance is trivial. For the third statement,

$$\mathbf{Var}\left(\lambda_j y\right) = \mathbb{E}\left[(\lambda_j y)^2\right] - \mathbb{E}\left[\lambda_j y\right]^2 = \mathbb{E}\left[\lambda_j^2 y^2\right] - \beta_j^2 \gamma_j^2 = \mathbb{E}\left[\lambda_j^2\right] - \beta_j^2 \gamma_j^2 = \beta_i - \beta_j^2 \gamma_j^2$$

For the fourth statement,

$$\mathbb{E}\left[(\lambda_j^2)^2\right] - \mathbb{E}\left[\lambda_j^2\right]^2 = \mathbb{E}\left[\lambda_j^4\right] - \beta_j^2 = \mathbb{E}\left[\lambda_j^2\right] - \beta_j^2 = \beta_i - \beta_j^2$$

For subsequent statements, we first derive that

$$\mathbb{E}\left[\lambda_j y \mid y\right] = \beta_j \frac{1+\gamma_j}{2} - \beta_j \frac{1-\gamma_j}{2} = \beta_j \gamma_j$$

and

$$\mathbb{E}\left[\lambda_j^2 \mid y\right] = \beta_j \frac{1+\gamma_j}{2} + \beta_j \frac{1-\gamma_j}{2} = \beta_j.$$

Now, for the fifth statement,

$$\mathbb{E}\left[(\lambda_j y)(\lambda_i y)\right] = \mathbb{E}\left[\mathbb{E}\left[\lambda_j y \mid y\right] \mathbb{E}\left[\lambda_i y \mid y\right]\right] = \beta_j \gamma_j \beta_i \gamma_i.$$

For the sixth statement,

$$\mathbb{E}\left[(\lambda_j^2)(\lambda_i^2)\right] = \mathbb{E}\left[\mathbb{E}\left[\lambda_j^2 \mid y\right] \mathbb{E}\left[\lambda_i^2 \mid y\right]\right] = \beta_j \beta_i.$$

Finally, for the seventh statement,

$$\mathbb{E}\left[(\lambda_j y)(\lambda_i^2)\right] = \mathbb{E}\left[\mathbb{E}\left[\lambda_j y \mid y\right] \mathbb{E}\left[\lambda_j^2 \mid y\right]\right] = \beta_j \gamma_j \beta_i.$$

This completes the proof. □

Lemma 7. It holds that

$$\gamma_j = \tanh(\theta_j^{\text{acc}}).$$

Proof. From the definitions,

$$\beta_j = \frac{\exp(\theta_j^{\text{acc}} + \theta_j^{\text{cov}}) + \exp(-\theta_j^{\text{acc}} + \theta_j^{\text{cov}})}{\exp(\theta_j^{\text{acc}} + \theta_j^{\text{cov}}) + \exp(-\theta_j^{\text{acc}} + \theta_j^{\text{cov}}) + 1}$$

and

$$\beta_j \gamma_j = \frac{\exp(\theta_j^{\text{acc}} + \theta_j^{\text{cov}}) - \exp(-\theta_j^{\text{acc}} + \theta_j^{\text{cov}})}{\exp(\theta_j^{\text{acc}} + \theta_j^{\text{cov}}) + \exp(-\theta_j^{\text{acc}} + \theta_j^{\text{cov}}) + 1}.$$

Therefore,
$$\gamma_j = \frac{\exp(\theta_j^{\text{acc}} + \theta_j^{\text{cov}}) - \exp(-\theta_j^{\text{acc}} + \theta_j^{\text{cov}})}{\exp(\theta_j^{\text{acc}} + \theta_j^{\text{cov}}) + \exp(-\theta_j^{\text{acc}} + \theta_j^{\text{cov}})} = \tanh(\theta_j^{\text{acc}}),$$
which is the desired result. □

Lemma 8. For any feasible model, it will be the case that, for any other feasible parameter vector $\hat{\theta}$,
$$p\left((\hat{\theta}^{\text{acc}})^T \lambda y \leq \frac{m}{2}\gamma_{\min}(\gamma\beta)_{\min}\right) \leq \exp\left(-\frac{m(\gamma\beta)_{\min}\gamma_{\min}^2}{9.34\,\text{artanh}(\gamma_{\max})}\right).$$

Proof. We start by noticing that
$$(\hat{\theta}^{\text{acc}})^T \lambda y = \sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \lambda_j y.$$

Since in this model, all the $\lambda_j y$ are independent of each other, we can bound this sum using a concentration bound. First, we note that
$$\left|\hat{\theta}_j^{\text{acc}} \lambda_j y\right| \leq \hat{\theta}_j^{\text{acc}}.$$

Second, we note that
$$\mathbb{E}\left[\hat{\theta}_j^{\text{acc}} \lambda_j y\right] = \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j$$
and
$$\mathbf{Var}\left(\hat{\theta}_j^{\text{acc}} \lambda_j y\right) = (\hat{\theta}_j^{\text{acc}})^2 \left(\beta_j - \beta_j^2 \gamma_j^2\right)$$
but
$$\left|\hat{\theta}_j^{\text{acc}} \lambda_j y\right| \leq \hat{\theta}_j^{\text{acc}} \leq \text{artanh}(\gamma_{\max}) \triangleq \hat{\theta}_{\max}^{\text{acc}}$$
because, for feasible models, by definition
$$\gamma_{\min} \leq \text{artanh}(\gamma_{\min}) \leq \hat{\theta}_j^{\text{acc}} \leq \text{artanh}(\gamma_{\max}).$$

Therefore, applying Bernstein's inequality gives us, for any t,
$$p_{\hat{\theta}}\left(\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \lambda_j y - \sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j \leq -t\right) \leq \exp\left(-\frac{3t^2}{6\sum_{i=1}^{m} (\hat{\theta}_j^{\text{acc}})^2 \gamma_j \beta_j \gamma_j + 2\hat{\theta}_{\max}^{\text{acc}} t}\right).$$

It follows that, if we let

$$t = \frac{1}{2}\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j,$$

then we get

$$\begin{aligned}
p_{\hat{\theta}}\left(\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \lambda_j y - \sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j \leq -t\right) &\leq \exp\left(-\frac{3\left(\frac{1}{2}\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j\right)^2}{6\sum_{i=1}^{m} (\hat{\theta}_j^{\text{acc}})^2 \gamma_j \beta_j \gamma_j + 2\hat{\theta}_{\max}^{\text{acc}}\left(\frac{1}{2}\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j\right)}\right) \\
&\leq \exp\left(-\frac{3\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j}{24\gamma_{\max}\hat{\theta}_{\max}^{\text{acc}} + 4\hat{\theta}_{\max}^{\text{acc}}}\right) \\
&\leq \exp\left(-\frac{3m(1-\gamma_{\max})}{28\hat{\theta}_{\max}^{\text{acc}}}\right) \\
&\leq \exp\left(-\frac{3\left(\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j\right)^2}{24\sum_{i=1}^{m} (\hat{\theta}_j^{\text{acc}})^2 \beta_j + 4\hat{\theta}_{\max}^{\text{acc}}\left(\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j\right)}\right) \\
&\leq \exp\left(-\frac{3\gamma_{\min}\left(\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j\right)\left(\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j\right)}{24\hat{\theta}_{\max}^{\text{acc}}\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j + 4\hat{\theta}_{\max}^{\text{acc}}\left(\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j\right)}\right) \\
&\leq \exp\left(-\frac{3\gamma_{\min}\left(\sum_{i=1}^{m} \hat{\theta}_j^{\text{acc}} \beta_j \gamma_j\right)}{28\hat{\theta}_{\max}^{\text{acc}}}\right) \\
&\leq \exp\left(-\frac{m\gamma_{\min}^2(\gamma\beta)_{\min}}{9.34\hat{\theta}_{\max}^{\text{acc}}}\right).
\end{aligned}$$

This is the desired expression. □

Lemma 9. The covariances of the sufficient statistics, conditioned on λ, are for all $j \neq j$,

$$\begin{aligned}
\mathbf{Cov}\left(\lambda_j y, \lambda_i y \mid \lambda\right) &= \lambda_j \lambda_i \operatorname{sech}^2((\theta^{\text{acc}})^T \lambda) \\
\mathbf{Cov}\left(\lambda_j^2, \lambda_i^2 \mid \lambda\right) &= 0.
\end{aligned}$$

Proof. The second result is obvious, so it suffices to prove only the first result. Clearly,

$$\mathbf{Cov}\left(\lambda_j y, \lambda_i y \middle| \lambda\right) = \lambda_j \lambda_i \mathbf{Var}\left(y|\lambda\right) = \lambda_j \lambda_i \left(1 - \mathbb{E}\left[y \mid \lambda\right]^2\right).$$

Plugging into the distribution formula lets us conclude that

$$\mathbb{E}\left[y \mid \lambda\right] = \frac{\exp((\theta^{\text{acc}})^T \lambda + (\theta^{\text{cov}})^T \lambda^2) - \exp(-(\theta^{\text{acc}})^T \lambda + (\theta^{\text{cov}})^T \lambda^2)}{\exp((\theta^{\text{acc}})^T \lambda + (\theta^{\text{cov}})^T \lambda^2) + \exp(-(\theta^{\text{acc}})^T \lambda + (\theta^{\text{cov}})^T \lambda^2)} = \tanh^2((\theta^{\text{acc}})^T \lambda),$$

and so

$$\mathbf{Cov}\left(\lambda_j y, \lambda_i y \middle| \lambda\right) = \lambda_j \lambda_i \left(1 - \tanh^2((\theta^{\text{acc}})^T \lambda)\right) = \Lambda_i \Lambda_j \operatorname{sech}^2((\theta^{\text{acc}})^T \lambda),$$

which is the desired result. □

Lemma 10. If θ and θ^* are two feasible models, then for any u,

$$\mathbb{E}_{\theta^*}\left[\mathbf{Var}_\theta\left(y \mid \lambda\right)\right] \leq 3 \exp\left(-\frac{m\beta_{\min}^2 \gamma_{\min}^3}{8 \operatorname{artanh}(\gamma_{\max})}\right).$$

Proof. First, we note that, by the result of Lemma 9,

$$\mathbf{Var}_\theta\left(y|\lambda\right) = \operatorname{sech}^2((\theta^{\text{acc}})^T \lambda).$$

Therefore,

$$\mathbb{E}_{\theta^*}\left[\mathbf{Var}_\theta\left(y|\lambda\right)\right] = \mathbb{E}_{\theta^*}\left[\operatorname{sech}^2((\theta^{\text{acc}})^T \lambda)\right].$$

Applying Lemma 8, we can bound this with

$$\begin{aligned}
\mathbb{E}_{\theta^*}\left[\mathbf{Var}_\theta\left(u^T \lambda y \middle| \lambda\right)\right] &\leq \left(\operatorname{sech}^2\left(\frac{m}{2}(\gamma\beta)_{\min}\gamma_{\min}^2\right) + \exp\left(-\frac{m(\gamma\beta)_{\min}\gamma_{\min}^2}{9.34 \operatorname{artanh}(\gamma_{\max})}\right)\right) \\
&\leq \left(2 \exp\left(-\frac{m}{2}(\gamma\beta)_{\min}\gamma_{\min}^2\right) + \exp\left(-\frac{m(\gamma\beta)_{\min}\gamma_{\min}^2}{9.34 \operatorname{artanh}(\gamma_{\max})}\right)\right) \\
&\leq 3 \exp\left(-\frac{m(\gamma\beta)_{\min}\gamma_{\min}^2}{9.34 \operatorname{artanh}(\gamma_{\max})}\right).
\end{aligned}$$

This is the desired expression. □

Appendix C

Proofs: Matrix Completion-Style Approach

In this section, we focus on theoretical results for the basic rank-one model considered in the main body of the paper. In Section C.1, we provide additional interpretation for the expression of our primary theoretical result bounding the estimation error of the label model. In Section C.2, we provide the proof of Corollary 1. In Section C.3, we then provide the proof of Theorem 1, connecting this estimation error to the generalization error of the end model; and in Section C.4, we provide the full proof of the main bound.

C.1 Interpreting the Main Bound

We re-state Theorem 4, which bounds the average error on the estimate of the label model parameters, providing more detail on and interpreting the terms of the bound.

Theorem 4. Let $\hat{\theta}$ be an estimate of θ^* produced by Algorithm 1 run over n unlabeled data points. Let $a := \left(\frac{d_O}{\Sigma_S} + \left(\frac{d_O}{\Sigma_S}\right)^2 \lambda_{\max}(K_O)\right)^{\frac{1}{2}}$ and $b := \frac{\|\Sigma_O^{-1}\|^2}{(\Sigma_O^{-1})_{\min}}$. Then, we have:

$$\mathbb{E}\left[\left\|\hat{\theta} - \theta^*\right\|\right] \leq 16(|\mathcal{Y}| - 1)d_O^2 \sqrt{\frac{32\pi}{n}} ab\sigma_{\max}(M_\Omega^+)\left(3\sqrt{d_O}a\lambda_{\min}^{-1}(\Sigma_O) + 1\right)\left(\kappa(\Sigma_O) + \lambda_{\min}^{-1}(\Sigma_O)\right).$$

Influence of $\sigma_{\max}(M_\Omega^+)$ the largest singular value of the pseudoinverse M_Ω^+. Note that $||M_\Omega^+||^2 = (\lambda_{\min}(M_\Omega^T M_\Omega))^{-1}$. As we shall see below, $\lambda_{\min}(M_\Omega^T M_\Omega)$ measures a quantity related to the structure of the graph G_{inv}. The smaller this quantity, the more information we have about G_{inv}, and the easier it is to estimate the accuracies. The smallest value of $||M_\Omega^+||^2$ (corresponding to the largest value of the eigenvalue) is $\sim \frac{1}{\sqrt{m}}$; the square of this quantity in the bound reduces the m^2 cost of estimating the covariance matrix to m.

It is not hard to see that

$$M_\Omega^T M_\Omega = \text{diag}(\text{deg}(G_{\text{inv}})) + \text{Adj}(G_{\text{inv}}).$$

Here, $\text{deg}(G_{\text{inv}})$ are the degrees of the nodes in G_{inv} and $\text{Adj}(G_{\text{inv}})$ is its adjacency matrix. This form closely resembles the graph Laplacian, which differs in the sign of the adjacency matrix term: $\mathcal{L}(G) = \text{diag}(\text{deg}(G)) - \text{Adj}(G)$. We bound

$$\sigma_{\max}(M_\Omega^+) \leq (d_{\min} + \lambda_{\min}(\text{Adj}(G_{\text{inv}}))))^{-1},$$

where $d_{\min}$ is the lowest-degree node in G_{inv} (that is, the labeling function with fewest appearances in Ω). In general, computing $\lambda_{\min}(\text{Adj}(G_{\text{inv}})))$ can be challenging. A closely related task can be done via *Cheeger inequalities*, which state that

$$2h_G \geq \lambda_{\min}(\mathcal{L}(G)) \geq \frac{1}{2}h_G^2,$$

where $\lambda_{\min}(\mathcal{L}(G))$ is the smallest non-zero eigenvalue of $\mathcal{L}(G)$ and

$$h_G = \min_X \frac{|E(X, \bar{X})|}{\min\left\{\sum_{x\in X} d_x, \sum_{y\in\bar{X}} d_y\right\}}$$

is the *Cheeger constant* of the graph [Chung, 1996]. The utility of the Cheeger constant is that it measures the presence of a bottleneck in the graph; the presence of such a bottleneck limits the graph density and is thus beneficial when estimating the structure in our case. Our Cheeger-constant like term $\sigma_{\max}(M_\Omega^+)$ acts the same way.

Now, in the easiest and most common case is that of conditionally independent labeling functions [Dalvi et al., 2013; Zhang et al., 2016b; Dalvi et al., 2013; Karger et al., 2011].,

$\mathrm{Adj}(G_{\mathrm{inv}})$ has 1's everywhere but the diagonal, and we can compute explicitly that

$$\sigma_{\max}(M_\Omega^+) = \frac{1}{\sqrt{m-2}}.$$

In the general setting, we must compute the minimal eigenvalue of the adjacency matrix, which is tractable, for example, for tree structures.

Influence of $\lambda_{\min}(\Sigma_O)$ the smallest eigenvalue of the observed matrix. This quantity reflects the conditioning of the observed (correlation) matrix; the better conditioned the matrix, the easier it is to estimate Σ_O.

Influence of $(\Sigma_O^{-1})_{\min}$ the smallest entry of the inverse observed matrix. This quantity contributes to Σ^{-1}, the generalized precision matrix that we centrally use; it is a measure of the smallest non-zero correlation between labeling function accuracies (that is, the smallest correlation between non-independent labeling function accuracies). Note that the tail bound of Theorem 4 scales as $\exp(-((\Sigma_O^{-1})_{\min})^2)$. This is natural, as distinguishing between small correlations and independencies requires more samples.

C.2 Proof of Corollary 1

Corollary 1. *Let $U = O \cup \mathcal{S}$. Let Σ_U be the generalized covariance matrix for U. Then $(\Sigma_U^{-1})_{i,j} = 0$ whenever i, j correspond to cliques C_1, C_2 respectively such that C_1, C_2 are not subsets of the same maximal clique.*

Proof: We partition the cliques $\mathcal{C}$ into two sets, U and $W = \mathcal{C} \setminus U$. Let Σ be the full generalized covariance matrix (i.e. including all maximal and non-maximal cliques) and $\Gamma = \Sigma^{-1}$. Thus we have:

$$\Sigma = \begin{bmatrix} \Sigma_U & \Sigma_{UW} \\ \Sigma_{UW}^T & \Sigma_W \end{bmatrix} \qquad\qquad \Sigma^{-1} = \Gamma = \begin{bmatrix} K_U & K_{UW} \\ K_{UW}^T & K_W \end{bmatrix}.$$

By the block matrix inversion lemma we have:

$$\Sigma_U^{-1} = K_U - K_{UW} K_W^{-1} K_{UW}^T.$$

We now follow the proof structure of Corollary 1 of [Loh and Wainwright, 2013]. We know K_U is graph structured by Theorem 1 of [Loh and Wainwright, 2013]. Next, using the same argument as in the proof of Corollary 1 of [Loh and Wainwright, 2013], we know that K_W, and therefore K_W^{-1}, is block-diagonal. Intuitively, because the set U contains all of the separator set cliques, and due to the running intersection property of a junction tree, each clique in W belongs to precisely one maximal clique- leading to block diagonal structure of K_W. We thus need only to show that the following quantity is zero for two cliques C_i, C_j that are not subsets of the same maximal clique, with corresponding indices i, j:

$$\left(K_{UW} K_W^{-1} K_{UW}^T\right)_{i,j} = \sum_B (K_{UW})_{i,B} (K_W^{-1})_{B,B} (K_{UW}^T)_{B,j},$$

where B are the indices corresponding to the blocks in K_W^{-1}, which correspond to maximal cliques. Our argument follows again as in Corollary 1 of [Loh and Wainwright, 2013]: since U contains the separator sets, if the two cliques C_1, C_2 are not subsets of the same maximal clique, then for each B, either $(K_{UW})_{i,B}$ or $(K_{UW}^T)_{B,j}$ must be zero, completing the proof.

C.3 Proof of Theorem 1

Let $\mathcal{D}$ be the true data generating distribution, such that $(x, y) \sim \mathcal{D}$. Let $p_\theta(y|\lambda)$ be the label model parameterized by θ and conditioned on the observed labeling function labels λ. Furthermore, assume that:

1. For some optimal label model parameters θ^*, $p_{\theta^*}(\lambda, y) = p_{\mathcal{D}}(\lambda, y)$;
2. The label y is independent of the features of our end model given the labeling function labels λ

That is, we assume that (i) the *optimal* label model, parameterized by θ^*, correctly matches the true distribution of labeling function labels λ drawn from the true distribution; and (ii) that these labels λ provide sufficient information to discern the label y. We note that these assumptions are the same ones used in Appendix B, and are intended primarily to illustrate the connection between the estimation accuracy of $\hat{\theta}$, which we bound in Theorem 4, and the end model performance.

Now, suppose that we have an end model parameterized by w, $h_w : \mathcal{X} \mapsto \mathcal{Y}$, and that to learn these parameters we minimize a normalized bounded loss function $l(h_w(x), y)$, such that without loss of generality, $l(h_w(x), y) \leq 1$. Normally our goal would be to find parameters that minimize the expected loss or *risk*, which we denote w^*:

$$w^* = \text{argmin}_w \, R(w) = \text{argmin}_w \, \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[l(h_w(x), y)\right] \tag{C.1}$$

However, since we do not have access to the true labels y, we instead minimize the expected noise-aware loss, producing an estimate $\tilde{w}$:

$$\tilde{w} = \text{argmin}_w \, R_\theta(w) = \text{argmin}_w \, \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{\tilde{y}\sim p_\theta(\cdot|\lambda)}\left[l(h_w(x), \tilde{y})\right]\right] \tag{C.2}$$

In practice, we actually minimize the *empirical* version of the noise aware loss over an unlabeled dataset $X_U = \{x^{(1)}, \ldots, x^{(n)}\}$, producing an estimate $\hat{w}$:

$$\hat{w} = \text{argmin}_w \, \hat{R}_\theta(w) = \text{argmin}_w \, \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\tilde{y}\sim p_\theta(\cdot|\lambda^{(i)})}\left[l(h_w(x^{(i)}), \tilde{y})\right]. \tag{C.3}$$

Let w^* be the minimizer of the expected loss R, let $\tilde{w}$ be the minimizer of the noise-aware loss for estimated label model parameters θ, R_θ, and let $\hat{w}$ be the minimizer of the empirical noise aware loss $\hat{R}_\theta$. Our goal is to bound the *generalization risk*- the difference between the expected loss of our empirically estimated parameters and of the optimal parameters,

$$R(\hat{w}) - R(w^*). \tag{C.4}$$

Additionally, since analyzing the empirical risk minimization error is standard and not specific to our setting, we simply assume that the error $|R_\theta(\tilde{w}) - R_\theta(\hat{w})| \leq \gamma(n)$, where $\gamma(n)$ is a decreasing function of the number of unlabeled data points n.

To start, using the law of total expectation first, followed by our assumption (2) about conditional independence, and finally using our assumption (1) about our optimal label model θ^*, we have that:

$$\begin{aligned}
R(w) &= \mathbb{E}_{(x',y')\sim\mathcal{D}}\left[R(w)\right] \\
&= \mathbb{E}_{(x',y')\sim\mathcal{D}}\left[\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[l(h_w(x'),y)|x=x'\right]\right] \\
&= \mathbb{E}_{(x',y')\sim\mathcal{D}}\left[\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[l(h_w(x'),y)|\lambda(x)=\lambda(x')\right]\right] \\
&= \mathbb{E}_{(x',y')\sim\mathcal{D}}\left[\mathbb{E}_{\tilde{y}\sim p_{\theta^*}(\cdot|\lambda)}\left[l(h_w(x'),\tilde{y})\right]\right] \\
&= R_{\theta^*}(w).
\end{aligned}$$

Now, we have:

$$\begin{aligned}
R(\hat{w}) - R(w^*) &= R_{\theta^*}(\hat{w}) + R_\theta(\hat{w}) - R_\theta(\hat{w}) + R_\theta(\tilde{w}) - R_\theta(\tilde{w}) - R_{\theta^*}(w^*) \\
&\leq R_{\theta^*}(\hat{w}) + R_\theta(\hat{w}) - R_\theta(\hat{w}) + R_\theta(w^*) - R_\theta(\tilde{w}) - R_{\theta^*}(w^*) \\
&\leq |R_\theta(\hat{w}) - R_\theta(\tilde{w})| + |R_{\theta^*}(\hat{w}) - R_\theta(\hat{w})| + |R_\theta(w^*) - R_{\theta^*}(w^*)| \\
&\leq \gamma(n) + 2\max_{w'} |R_{\theta^*}(w') - R_\theta(w')|,
\end{aligned}$$

where in the first step we use our result that $R = R_{\theta^*}$ as well as add and subtract terms; and in the second step we use the fact that $R_\theta(\tilde{w}) \leq R_\theta(w^*)$. We now have our generalization risk controlled primarily by $|R_{\theta^*}(w') - R_\theta(w')|$, which is the difference between the expected noise aware losses given the estimated label model parameters θ and the true label model parameters θ^*. Next, we see that, for any w':

$$\begin{aligned}
|R_{\theta^*}(w') - R_\theta(w')| &= \left|\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{\tilde{y}\sim p_{\theta^*}(\cdot|\lambda)}\left[l(h_w(x),\tilde{y})\right] - \mathbb{E}_{\tilde{y}\sim p_\theta(\cdot|\lambda)}\left[l(h_w(x),\tilde{y})\right]\right]\right| \\
&= \left|\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\sum_{y'\in\mathcal{Y}} l(h_w(x),y')\left(p_{\theta^*}(y' \mid \lambda) - p_\theta(y' \mid \lambda)\right)\right]\right| \\
&\leq \sum_{y'\in\mathcal{Y}} \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[|p_{\theta^*}(y' \mid \lambda) - p_\theta(y' \mid \lambda)|\right] \\
&\leq |\mathcal{Y}| \max_{y'} \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[|p_{\theta^*}(y' \mid \lambda) - p_\theta(y' \mid \lambda)|\right],
\end{aligned}$$

where we have now bounded $|R_{\theta^*}(w') - R_\theta(w')|$ by the size of the structured output space $|\mathcal{Y}|$, and a term having to do with the difference between the probability distributions of θ and θ^*.

Now, we use the result from [Honorio, 2012] (Lemma 19) which establishes that the log probabilities of discrete factor graphs with indicator features (such as our model $p_\theta(\lambda, y)$) are $(l_\infty, 2)$-Lipschitz with respect to their parameters, and the fact that for x, y s.t. $|x|, |y| \leq 1$, $|x - y| \leq |\log(x) - \log(y)|$, to get:

$$\begin{aligned}
|p_{\theta^*}(y' \mid \lambda) - p_\theta(y' \mid \lambda)| &\leq \left|\log p_{\theta^*}(y' \mid \lambda) - \log p_\theta(y' \mid \lambda)\right| \\
&\leq \left|\log p_{\theta^*}(\lambda, y') - \log p_\theta(\lambda, y')\right| + \left|\log p_{\theta^*}(\lambda) - \log P_\theta(\lambda\right| \\
&\leq 2\,||\theta^* - \theta||_\infty + 2\,||\theta^* - \theta||_\infty \\
&\leq 4\,||\theta^* - \theta||\,,
\end{aligned}$$

where we use the fact that the statement of Lemma 19 also holds for every marginal distribution as well. Therefore, we finally have:

$$R(\hat{w}) - R(w^*) \leq \gamma(n) + 4|\mathcal{Y}|\,||\theta^* - \theta||\,.$$

C.4 Proof of Theorem 4

Proof: First we briefly provide a road map of the proof of Theorem 4. We consider estimating $\tilde{\theta}$ with our procedure in the rank-one setting, and we seek a tail bound on $||\tilde{\theta} - \theta||$. The challenge here is that the observed matrix Σ_O we see is itself constructed from a series of observed i.i.d. samples $\psi(O)^{(1)}, \ldots, \psi(O)^{(n)}$. We bound (through a matrix concentration inequality) the error $\Delta_O = \tilde{\Sigma}_O - \Sigma_O$, and view Δ_O as a perturbation of Σ_O. Afterwards, we use a series of perturbation analyses to ultimately bound $||\tilde{\Sigma}_{OS} - \Sigma_{OS}||$, and then use this directly to bound $||\tilde{\theta} - \theta||$; each of the perturbation results is in terms of Δ_O.

We begin with some notation. We write the following perturbations (note that all the terms written with Δ are additive, while the δ term is relative)

$$\begin{aligned}
\tilde{\Sigma}_{OS} &= \Sigma_{OS} + \Delta_{OS}, \\
\tilde{\Sigma}_{O} &= \Sigma_{O} + \Delta_{O}, \\
\tilde{\ell} &= \ell + \Delta_{\ell}, \\
\tilde{z} &= (I + \text{diag}(\delta_z))z.
\end{aligned}$$

Now we start our perturbation analysis:

$$\begin{aligned}
\tilde{\Sigma}_{OS} &= \frac{1}{\sqrt{\tilde{c}}}\tilde{\Sigma}_O\tilde{z} = \frac{1}{\sqrt{\tilde{c}}}(\Sigma_O + \Delta_O)(I + \text{diag}(\delta_z))z \\
&= \frac{1}{\sqrt{\tilde{c}}}\left(\Sigma_O z + \Sigma_O \text{diag}(\delta_z)z + \Delta_O(I + \text{diag}(\delta_z))z\right).
\end{aligned}$$

Subtracting $\Sigma_{OS} = \frac{1}{\sqrt{c}}\Sigma_O z$, we get

$$\Delta_{OS} = \left(\frac{1}{\sqrt{\tilde{c}}} - \frac{1}{\sqrt{c}}\right)\Sigma_O z + \frac{1}{\sqrt{\tilde{c}}}\left(\Sigma_O \text{diag}(\delta_z)z + \Delta_O(I + \text{diag}(\delta_z))z\right). \tag{C.5}$$

The rest of the analysis requires us to bound the norms for each of these terms.

Left-most term. We have that

$$\left\|\left(\frac{1}{\sqrt{\tilde{c}}} - \frac{1}{\sqrt{c}}\right)\Sigma_O z\right\| = \left|\frac{\sqrt{c}}{\sqrt{\tilde{c}}} - 1\right|\left\|\frac{1}{\sqrt{c}}\Sigma_O z\right\| = \left|\frac{\sqrt{c}}{\sqrt{\tilde{c}}} - 1\right| \|\Sigma_{OS}\| \leq \sqrt{d_O}\left|\frac{\sqrt{c}}{\sqrt{\tilde{c}}} - 1\right| \leq \sqrt{d_O}|\tilde{c} - c|.$$

Here, we bounded $\|\Sigma_{OS}\|$ by $\sqrt{d_O}$, since $\Sigma_{OS} \in [-1, 1]^{d_O}$. Then, note that $c = \Sigma_S^{-1}(1 + z^T\Sigma_O z) \geq 0$, since $\Sigma_S < 1$ and $\Sigma_O \succeq 0 \implies z^T\Sigma_O z \geq 0$, so therefore $c, \tilde{c} \geq 1$. In the last inequality, we use this to imply that $|\sqrt{c}/\sqrt{\tilde{c}} - 1| \leq |\sqrt{c} - \sqrt{\tilde{c}}| \leq |\tilde{c} - c|$. Next we work on bounding $|\tilde{c} - c|$. We have

$$\begin{aligned}
|\tilde{c} - c| &= |\Sigma_S^{-1}||\tilde{z}^T\tilde{\Sigma}_O\tilde{z} - z^T\Sigma_O z| \\
&= |\Sigma_S^{-1}||z^T(I + \mathrm{diag}(\delta_z))^T(\Sigma_O + \Delta_O)(I + \mathrm{diag}(\delta_z))z - z^T\Sigma_O z| \\
&= |\Sigma_S^{-1}||z^T\Sigma_O\mathrm{diag}(\delta_z)z + z^T\Delta_O(I + \mathrm{diag}(\delta_z))z + z^T\mathrm{diag}(\delta_z)^T(\Sigma_O + \Delta_O)(I + \mathrm{diag}(\delta_z))z| \\
&\leq |\Sigma_S^{-1}|||z||^2\left(||\Sigma_O||\left(2||\delta_z|| + ||\delta_z||^2\right) + ||\Delta_O||\left(2||\delta_z|| + ||\delta_z||^2 + 1\right)\right) \\
&\leq ||z||^2\left(||\Sigma_O||\left(2||\delta_z|| + ||\delta_z||^2\right) + ||\Delta_O||\left(2||\delta_z|| + ||\delta_z||^2 + 1\right)\right).
\end{aligned}$$

Thus,

$$\left\|\left(\frac{1}{\sqrt{\tilde{c}}} - \frac{1}{\sqrt{c}}\right)\Sigma_O z\right\| \leq \sqrt{d_O}||z||^2\left(||\Sigma_O||\left(2||\delta_z|| + ||\delta_z||^2\right) + ||\Delta_O||\left(2||\delta_z|| + ||\delta_z||^2 + 1\right)\right). \quad \text{(C.6)}$$

Bounding c. We will need a bound on c to bound z. We have that

$$c = (\Sigma_S - \Sigma_{OS}^T\Sigma_O^{-1}\Sigma_{OS})^{-1}.$$

Applying the Woodbury matrix inversion lemma, we have:

$$c = \Sigma_S^{-1} + \Sigma_S^{-1}\Sigma_{OS}^T\left(\Sigma_O - \Sigma_{OS}\Sigma_S^{-1}\Sigma_{OS}^T\right)^{-1}\Sigma_{OS}\Sigma_S^{-1}$$

Now, by the blockwise inversion lemma, we know that

$$K_O = \left(\Sigma_O - \Sigma_{OS}\Sigma_S^{-1}\Sigma_{OS}^T\right)^{-1}$$

So we then have:

$$c = \Sigma_S^{-1} + \Sigma_S^{-1}\Sigma_{OS}^T K_O\Sigma_{OS}\Sigma_S^{-1} \leq \Sigma_S^{-1} + (\Sigma_S^{-1})^2||\Sigma_{OS}||^2||K_O||$$

Bounding z. We'll use our bound on c, since $z = \sqrt{c}\Sigma_O^{-1}\Sigma_{OS}$.

$$\begin{aligned}
||z|| &= || \sqrt{c}\Sigma_O^{-1}\Sigma_{OS}|| \\
&\leq \left(\Sigma_S^{-1} + (\Sigma_S^{-1})^2||\Sigma_{OS}||^2||K_O||\right)^{\frac{1}{2}} ||\Sigma_O^{-1}||||\Sigma_{OS}|| \\
&\leq \left(\Sigma_S^{-1} + (\Sigma_S^{-1})^2 d_O||K_O||\right)^{\frac{1}{2}} ||\Sigma_O^{-1}|| \sqrt{d_O} \\
&= \frac{d_O}{\Sigma_S}\left(\frac{\Sigma_S}{d_O} + \lambda_{\max}(K_O)\right)^{\frac{1}{2}} \lambda_{\min}^{-1}(\Sigma_O)
\end{aligned}$$

In the last inequality, we used the fact that $||\Sigma_{OS}||^2 \leq d_O$. Now we want to control $||\Delta_\ell||$.

Perturbation bound. We have the perturbation bound

$$||\Delta_\ell|| \leq ||M_\Omega^+||||\tilde{q}_S - q_S||. \tag{C.7}$$

We need to work on the term $||\tilde{q}_S - q_S||$. To avoid overly heavy notation, we write $P = \Sigma_O^{-1}$, $\tilde{P} = \tilde{\Sigma}_O^{-1}$, and $\Delta_P = P - \tilde{P}$. Then we have:

$$\begin{aligned}
||\tilde{q}_S - q_S||^2 &= \sum_{(i,j)\in S} \left(\log(\tilde{P}_{i,j}^2) - \log(P_{i,j}^2)\right)^2 \\
&= 4\sum_{(i,j)\in S} \left(\log(|\tilde{P}_{i,j}|) - \log(|P_{i,j}|)\right)^2 \\
&= 4\sum_{(i,j)\in S} \left(\log(|P_{i,j} + (\Delta_P)_{i,j}|) - \log(|P_{i,j}|)\right)^2 \\
&\leq 4\sum_{(i,j)\in S} \left[\log\left(1 + \left|\frac{(\Delta_P)_{i,j}}{P_{i,j}}\right|\right)\right]^2 \\
&\leq 8\sum_{(i,j)\in S} \left(\frac{|(\Delta_P)_{i,j}|}{|P_{i,j}|}\right)^2 \\
&\leq \frac{8}{P_{\min}^2}\sum_{(i,j)\in S} (\Delta_P)_{i,j}^2 \\
&\leq \frac{8||\tilde{\Sigma}_O^{-1} - \Sigma_O^{-1}||^2}{((\Sigma_O^{-1})_{\min})^2}.
\end{aligned}$$

Here, the second inequality uses $(\log(1 + x))^2 \leq x^2$, and the fourth inequality sums over squared values. Next, we use the perturbation bound $\|\tilde{\Sigma}_O^{-1} - \Sigma_O^{-1}\| \leq \|\Sigma_O^{-1}\|^2 \|\Delta_O\|$, so that we have

$$\|\tilde{q}_S - q_S\| \leq \frac{2\sqrt{2}\|\Sigma_O^{-1}\|^2\|\Delta_O\|}{(\Sigma_O^{-1})_{\min}}.$$

Then, plugging this into (C.7), we get that

$$\|\Delta_\ell\| \leq \sigma_{\max}(M_\Omega^+)\frac{2\sqrt{2}\|\Sigma_O^{-1}\|^2\|\Delta_O\|}{(\Sigma_O^{-1})_{\min}}. \tag{C.8}$$

Bounding δ_z. Note also that $\|\Delta_\ell\|^2 = \sum_{i=1}^m (\log(\tilde{z}_i^2) - \log(z_i^2))$. We have that

$$\begin{aligned}
\|\Delta_\ell\|^2 &= \sum_{i=1}^m \log\left(\frac{\tilde{z}_i^2}{z_i^2}\right) \\
&= 2\sum_{i=1}^m \log\left(\frac{|\tilde{z}_i|}{|z_i|}\right) \\
&= 2\sum_{i=1}^m \log(1 + |(\delta_z)_i|), \\
&\geq 2\sum_{i=1}^m (\delta_z)_i^2 \\
&= 2\|\delta_z\|^2,
\end{aligned}$$

where in the fourth step, we used the bound $\log(1 + a) \geq a^2$ for small a. Then, we have

$$\|\delta_z\| \leq \frac{\sqrt{2}\|\Sigma_O^{-1}\|^2\|\Delta_O\|}{(\Sigma_O^{-1})_{\min}}\sigma_{\max}(M_\Omega^+). \tag{C.9}$$

Putting it together. Using (C.5), we have that

$$\begin{aligned}
||\Delta_{OS}|| &= \left\|\left(\frac{1}{\sqrt{\tilde{c}}} - \frac{1}{\sqrt{c}}\right)\Sigma_O z + \frac{1}{\sqrt{\tilde{c}}}\left(\Sigma_O \text{diag}(\delta_z) z + \Delta_O(I + \text{diag}(\delta_z))z\right)\right\| \\
&\leq \left\|\left(\frac{1}{\sqrt{\tilde{c}}} - \frac{1}{\sqrt{c}}\right)\Sigma_O z\right\| + \left(||\Sigma_O \text{diag}(\delta_z)|| + ||\Delta_O(I + \text{diag}(\delta_z))||\right)||z|| \\
&\leq \sqrt{d_O}||z||^2 \left(||\Sigma_O|| \left(2||\delta_z|| + ||\delta_z||^2\right) + ||\Delta_O|| \left(2||\delta_z|| + ||\delta_z||^2 + 1\right)\right) \\
&\qquad + ||\Sigma_O||||\delta_z||||z|| + ||\Delta_O||||z||(1 + ||\delta_z||) \\
&\leq \sqrt{d_O}||z||^2 \left(3||\Sigma_O||||\delta_z|| + 3||\Delta_O||||\delta_z|| + ||\Delta_O||\right) \\
&\qquad + ||\Sigma_O||||\delta_z||||z|| + ||\Delta_O||||z||(1 + ||\delta_z||) \\
&\leq ||z||\left(3\sqrt{d_O}||z|| + 1\right)\left((||\Sigma_O|| + ||\Delta_O||)||\delta_z|| + ||\Delta_O||\right)
\end{aligned}$$

Where in the first inequality, we use the triangle inequality and the fact that $\tilde{c} > 1$, and in the third inequality, we relied on the fact that we can control $||\delta_z||$ (through $||\Delta_O||$) so that we can make it small enough and thus take $||\delta_z||^2 \leq ||\delta_z||$. Now we can plug in our bounds on $||z||$ and $||\delta_z||$ from before:

$$\begin{aligned}
||\Delta_{OS}|| \leq &\left(\frac{d_O}{\Sigma_S}\left(\frac{\Sigma_S}{d_O} + \lambda_{\max}(K_O)\right)^{\frac{1}{2}} \lambda_{\min}^{-1}(\Sigma_O)\right)\left(3\sqrt{d_O}\left(\frac{d_O}{\Sigma_S}\left(\frac{\Sigma_S}{d_O} + \lambda_{\max}(K_O)\right)^{\frac{1}{2}} \lambda_{\min}^{-1}(\Sigma_O)\right) + 1\right) \\
&\times \left((||\Sigma_O|| + ||\Delta_O||)\left(\frac{\sqrt{2}||\Sigma_O^{-1}||^2||\Delta_O||}{(\Sigma_O^{-1})_{\min}}\sigma_{\max}(M_\Omega^+)\right) + ||\Delta_O||\right)
\end{aligned}$$

For convenience, we set $||\Delta_O|| = t$. Recall that

$$a = \left(\frac{d_O}{\Sigma_S} + \left(\frac{d_O}{\Sigma_S}\right)^2 \lambda_{\max}(K_O)\right)^{\frac{1}{2}}$$

and

$$b = \frac{||\Sigma_O^{-1}||^2}{(\Sigma_O^{-1})_{\min}}.$$

Then, we have

$$||\Delta_{OS}|| \leq (3\sqrt{d_O}a\lambda_{\min}^{-1}(\Sigma_O) + 1)\left(\sqrt{2}ab\kappa(\Sigma_O)\sigma_{\max}(M_\Omega^+)t + \sqrt{2}ab\frac{\sigma_{\max}(M_\Omega^+)}{\lambda_{\min}(\Sigma_O)}t^2 + a\lambda_{\min}^{-1}(\Sigma_O)t\right).$$

Again we can take t small so that $t^2 \leq t$. Simplifying further, we have

$$||\Delta_{OS}|| \leq (3\sqrt{d_O}a\lambda_{\min}^{-1}(\Sigma_O) + 1)\left(\sqrt{2}ab\sigma_{\max}(M_\Omega^+)\left[\kappa(\Sigma_O) + \lambda_{\min}^{-1}(\Sigma_O)\right] + a\lambda_{\min}^{-1}(\Sigma_O)\right)t.$$

Finally, since the $a\lambda_{\min}^{-1}(\Sigma_O)$ is smaller than the left-hand term inside the parentheses, we can write

$$||\Delta_{OS}|| \leq (3\sqrt{d_O}a\lambda_{\min}^{-1}(\Sigma_O) + 1)\left(2\sqrt{2}ab\sigma_{\max}(M_\Omega^+)\left[\kappa(\Sigma_O) + \lambda_{\min}^{-1}(\Sigma_O)\right]\right)t. \tag{C.10}$$

Concentration bound. We need to bound $t = ||\Delta_O||$, the error when estimating Σ_O from observations $\psi(O)^{(1)}, \ldots, \psi(O)^{(n)}$ over n unlabeled data points.

To start, recall that O is the set of observable cliques, $\psi(O) \in \{0,1\}^{d_O}$ is the corresponding vector of minimal statistics, and $\Sigma_O = \mathbf{Cov}(\psi(O))$. For notational convenience, let $R = \mathbb{E}\left[\psi(O)\psi(O)^T\right]$, $r = \mathbb{E}[\psi(O)]$, and $r_k = \psi(O)^{(k)}$, and $\Delta_r = \frac{1}{n}\sum_{i=1}^n r_k - r$. Then we have:

$$\begin{aligned}||\Delta_O|| = \left\|\Sigma_O - \tilde{\Sigma}_O\right\| &= \left\|(R - rr^T) - \left(\frac{1}{n}\sum_{i=1}^n r_i r_i^T - (r+\Delta_r)(r+\Delta_r)^T\right)\right\| \\ &\leq \underbrace{\left\|R - \frac{1}{n}\sum_{i=1}^n r_i r_i^T\right\|}_{\Delta_R} + \underbrace{\left\|rr^T - (r+\Delta_r)(r+\Delta_r)^T\right\|}_{\Delta_r}.\end{aligned}$$

We start by applying the matrix Hoeffding inequality [Tropp, 2015] to bound the first term, Δ_R. Let $S_k = \frac{1}{n}(R - R_k)$, and thus clearly $\mathbb{E}[S_k] = 0$. We seek a sequence of symmetric matrices A_k s.t. $S_k^2 \preceq A_k^2$. First, note that, for some vectors x, v,

$$x^T\left(||v||^2 I - vv^T\right)x = ||v||^2\,||x||^2 - \langle x, v\rangle^2 \geq 0$$

using Cauchy-Schwarz; therefore $\|v\|^2 I \geq vv^T$, so that

$$d_O^2 I \geq \|r_k\|^4 I \geq \|r_k\|^2 r_k r_k^T = (r_k r_k^T)^2.$$

Next, note that $(r_k r_k^T + R)^2 \geq 0$. Now, we use this to see that:

$$(nS_k)^2 = (r_k r_k^T - R)^2 \leq (r_k r_k^T - R)^2 + (r_k r_k^T + R)^2 = 2((r_k r_k^T)^2 + R^2) \leq 2(d_O^2 I + R^2).$$

Therefore, let $A_k^2 = \frac{2}{n^2}(d_O^2 I + R^2)$, and note that $\left\|R^2\right\| \leq \|R\|^2 \leq (d_O \|R\|_{max})^2 = d_O^2$. We then have

$$\sigma^2 = \left\|\sum_{k=1}^{n} A_k^2\right\| \leq \frac{2}{n}\left(d_O^2 + \left\|R^2\right\|\right) \leq \frac{4d_O^2}{n}.$$

And thus,

$$p\left(\|\Delta_R\| \geq \gamma\right) \leq 2d_O \exp\left(-\frac{n\gamma^2}{32d_O^2}\right). \tag{C.11}$$

Next, we bound Δ_r. We see that:

$$\begin{aligned}
\|\Delta_r\| &= \left\|rr^T - (r + \Delta_r)(r + \Delta_r)^T\right\| \\
&= \left\|r\Delta_r^T + \Delta_r r^T + \Delta_r \Delta_r^T\right\| \\
&\leq \left\|r\Delta_r^T\right\| + \left\|\Delta_r r^T\right\| + \left\|\Delta_r \Delta_r^T\right\| \\
&\leq 2\|r\|\,\|\Delta_r\| + \|\Delta_r\|^2 \\
&\leq 3\|r\|\,\|\Delta_r\| \\
&\leq 3\|r\|_1\,\|\Delta_r\|_1 \\
&\leq 3d_O^2|\Delta_r'|,
\end{aligned}$$

where Δ_r' is the perturbation for a single element of $\psi(O)$. We can then apply the standard

Hoeffding's bound to get:

$$p(\|\Delta_r\| \geq \gamma) \leq 2\exp\left(-\frac{2n\gamma^2}{3d_O^2}\right),$$

Combining the bounds for $\|\Delta_R\|$ and $\|\Delta_r\|$, we get:

$$p(\|\Delta_O\| \geq \gamma) = p(t \geq \gamma) \leq 3d_O \exp\left(-\frac{n\gamma^2}{32d_O^2}\right). \tag{C.12}$$

Final steps Now, we use the bound on t in (C.10) and the concentration bound above to write

$$\begin{aligned} p(\|\Delta_{OS}\| \geq t') &\leq p(Vt \geq t') \\ &= p\left(t \geq \frac{t'}{V}\right) \\ &\leq 2d_O \exp\left(-\frac{nt'^2}{32V^2 d_O^2}\right), \end{aligned}$$

where $V = (3\sqrt{d_O}a\lambda_{\min}^{-1}(\Sigma_O) + 1)\left(2\sqrt{2}ab\sigma_{\max}(M_\Omega^+)\left[\kappa(\Sigma_O) + \frac{1}{\lambda_{\min}(\Sigma_O)}\right]\right)$.

Given $\tilde{\Sigma}_{OS}$, we recover $\tilde{\theta}_1 = \tilde{\Sigma}_{OS} + \mathbb{E}[\psi(H)]\hat{\mathbb{E}}[\psi(O)]$. We assume $\mathbb{E}[\psi(H)]$ is known, and we can bound the error introduced by $\mathbb{E}[\psi(H)]\hat{\mathbb{E}}[\psi(O)]$ as above, which we see can be folded into the looser bound for the error in $\tilde{\Sigma}_{OS}$.

Finally, we expand the rank-one form $\tilde{\theta}_1$ into $\tilde{\theta}$ algebraically, according to our weight tying in the rank one model we use. Suppose in the rank one reduction, we let $y_B = \mathbb{1}\{y = y_1\}$. Then each element of θ_1 that we track corresponds to either the probability of being correct, $\alpha_{C,y} = p_\theta(\cap_{i \in C}\{\lambda_i = y\}, y = y)$ or the probability of being incorrect, $\frac{1}{r-1}(1 - \alpha_{C,y})$, for each labeling function clique C and label output combination y_C, and this value is simply copied $r-1$ times (for the other, weight-tied incorrect values), except for potentially one entry where it is multiplied by $(r-1)$ and then subtracted from 1 (to transform from incorrect to correct). Therefore, $\|\Delta_\theta\| = \left\|\theta - \tilde{\theta}\right\| \leq 2(r-1)\left\|\theta_1 - \tilde{\theta}_1\right\|$. Thus, we have:

$$p(\|\Delta_\theta\| \geq t') \leq 4(r-1)d_O \exp\left(-\frac{nt'^2}{32V^2d_O^2}\right),$$

where V is defined as above. We only have one more step:

$$\begin{aligned}
\mathbb{E}\left[\left\|\tilde{\theta}-\theta\right\|\right] &= \int_0^\infty p(\|\tilde{\theta}-\theta\| \geq \gamma)d\gamma \\
&\leq \int_0^\infty 4(r-1)d_O \exp\left(-\frac{n}{32V^2d_O^2}\gamma^2\right)d\gamma \\
&= \frac{4(r-1)d_O\sqrt{\pi}}{2\sqrt{\frac{n}{32V^2d_O^2}}} \\
&= 4(r-1)d_O^2\sqrt{\frac{32\pi}{n}}V.
\end{aligned}$$

Here, we used the fact that $\int_0^\infty \exp(-a\gamma^2)d\gamma = \frac{\sqrt{\pi}}{2\sqrt{a}}$. □

Bibliography

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

U. R. Acharya, S. L. Oh, Y. Hagiwara, J. H. Tan, and H. Adeli. Deep convolutional neural network for the automated detection and diagnosis of seizure using EEG signals. *Computers in Biology and Medicine*, 100:270–278, September 2018.

A. K. Agrawala. Learning with a probabilistic teacher. *IEEE Transactions on Infomation Theory*, 16:373–379, 1970.

E. Alfonseca, K. Filippova, J.-Y. Delort, and G. Garrido. Pattern learning for relation extraction with a hierarchical topic model. In *Meeting of the Association for Computational Linguistics (ACL)*, 2012.

A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1): 2773–2832, 2014.

S. H. Bach, B. He, A. Ratner, and C. Ré. Learning the structure of generative models without labeled data. In *International Conference on Machine Learning (ICML)*, 2017.

S. H. Bach, D. Rodriguez, Y. Liu, C. Luo, H. Shao, C. Xia, S. Sen, A. Ratner, B. Hancock, H. Alborzi, R. Kuchhal, C. Ré, and R. Malkin. Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 362–375. ACM, 2019.

J. Bai, F. Lu, K. Zhang, et al. Onnx: Open neural network exchange. `https://github.com/onnx/onnx`, 2019.

A. Balsubramani and Y. Freund. Scalable semi-supervised aggregation of classifiers. In *Advances in Neural Information Processing Systems*, pages 1351–1359, 2015.

S. Baluja and I. Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *arXiv preprint arXiv:1703.09387*, 2017.

D. Berend and A. Kontorovich. Consistency of weighted majority votes. In *NIPS 2014*. 2014.

A. Bhaskara, M. Charikar, and A. Vijayaraghavan. Uniqueness of tensor decompositions with applications to polynomial identifiability, 2014.

J. Birgmeier, M. Haeussler, C. A. Deisseroth, K. A. Jagadeesh, A. J. Ratner, H. Guturu, A. M. Wenger, P. D. Stenson, D. N. Cooper, C. Ré, J. A. Bernstein, and G. Bejerano. Amelie accelerates mendelian patient diagnosis directly from the primary literature. *bioRxiv*, 2017.

A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 92–100, 1998.

J. Bootkrajang and A. Kabán. Label-noise robust logistic regression and its applications. In *Machine Learning and Knowledge Discovery in Databases*, pages 143–158. Springer, 2012.

E. Bringer, A. Israeli, A. Ratner, and C. Ré. Osprey: Weak supervision of imbalanced extraction problems without code. *SIGMOD DEEM Workshop*, 2019.

R. C. Bunescu and R. J. Mooney. Learning to extract relations from the Web using minimal supervision. In *Meeting of the Association for Computational Linguistics (ACL)*, 2007.

D. Bychkov, N. Linder, R. Turkki, S. Nordling, P. E. Kovanen, C. Verrill, M. Walliander, M. Lundin, C. Haglund, and J. Lundin. Deep learning based tissue analysis predicts outcome in colorectal cancer. *Scientific Reports*, 8(1):3395, 2018.

A. Callahan, J. A. Fries, C. Ré, J. I. H. III, N. J. Giori, S. L. Delp, and N. H. Shah. Medical device surveillance with electronic health records. *CoRR*, abs/1904.07640, 2019.

E. Candes and T. Tao. The Dantzig selector: Statistical estimation when p is much larger than n. *The Annals of Statistics*, 35(6):2313–2351, 2007.

E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of the ACM*, 58(11), 2011.

R. Caruana. Multitask learning: A knowledge-based source of inductive bias, 1993.

R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997.

R. Caspi, R. Billington, L. Ferrer, H. Foerster, C. A. Fulcher, I. M. Keseler, A. Kothari, M. Krummenacker, M. Latendresse, L. A. Mueller, Q. Ong, S. Paley, P. Subhraveti, D. S. Weaver, and P. D. Karp. The MetaCyc database of metabolic pathways and enzymes and the BioCyc collection of pathway/genome databases. *Nucleic Acids Research*, 44(D1): D471–D480, 2016.

V. Chandrasekaran, P. A. Parrilo, and A. S. Willsky. Latent variable graphical model selection via convex optimization. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1610–1613. IEEE, 2010.

V. Chandrasekaran, S. Sanghavi, P. A. Parrilo, and A. S. Willsky. Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*, 21(2):572–596, 2011.

V. Chandrasekaran, P. A. Parrilo, and A. S. Willsky. Latent variable graphical model selection via convex optimization. *The Annals of Statistics*, 40(4):1935–1967, 2012.

O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. Adaptive Computation and Machine Learning. MIT Press, 2009.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

F. Chollet et al. Keras. `https://keras.io`, 2015.

F. R. K. Chung. Laplacians of graphs and cheeger inequalities. 1996.

D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition, 2010. *Cited on*, 80.

K. Clark, B. Vendt, K. Smith, J. Freymann, J. Kirby, P. Koppel, S. Moore, S. Phillips, D. Maffitt, M. Pringle, L. Tarbox, and F. Prior. The cancer imaging archive (TCIA): Maintaining and operating a public information repository. *Journal of Digital Imaging*, 26(6):1045–1057, 2013.

J. Clarke, D. Goldwasser, M.-W. Chang, and D. Roth. Driving semantic parsing from the world's response. In *Proceedings of the fourteenth conference on computational natural language learning*, pages 18–27. Association for Computational Linguistics, 2010.

G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *Proceedings of the 33rd international conference on Very large data bases*, pages 315–326. VLDB Endowment, 2007.

D. Corney, D. Albakour, M. Martinez, and S. Moussa. What do a million news articles look like? In *Workshop on Recent Trends in News Information Retrieval*, 2016.

M. Craven, J. Kumlien, et al. Constructing biological knowledge bases by extracting information from text sources. In *ISMB*, volume 1999, pages 77–86, 1999.

N. Dalvi, A. Dasgupta, R. Kumar, and V. Rastogi. Aggregating crowdsourced binary ratings. In *Proceedings of the 22Nd International Conference on World Wide Web (WWW)*, 2013.

A. P. Davis et al. A CTD–Pfizer collaboration: Manual curation of 88,000 scientific articles text mined for drug–disease and drug–phenotype interactions. *Database*, 2013.

A. P. Dawid and A. M. Skene. Maximum likelihood estimation of observer error-rates using the EM algorithm. *Journal of the Royal Statistical Society C*, 28(1):20–28, 1979.

J. Dean. Twiml: Systems and software for machine learning at scale with jeff dean. `https://twimlai.com/`

`twiml-talk-124-systems-software-machine-learning-scale-jeff-dean/`. Accessed: 2019-05-29.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, IEEE Conference on (CVPR)*, 2009.

T. DeVries and G. W. Taylor. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538*, 2017.

A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, 2011.

G. R. Doddington, A. Mitchell, M. A. Przybocki, L. A. Ramshaw, S. Strassel, and R. M. Weischedel. The automatic content extraction (ace) program-tasks, data, and evaluation. In *LREC*, volume 2, page 1, 2004.

X. L. Dong and D. Srivastava. *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.

A. Dosovitskiy, P. Fischer, J. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks, arxiv preprint. *arXiv preprint arXiv:1506.02753*, 2015.

G. Druck, B. Settles, and A. McCallum. Active learning by labeling features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pages 81–90. Association for Computational Linguistics, 2009.

J. Dunnmon, A. Ratner, N. Khandwala, K. Saab, M. Markert, H. Sagreiya, R. Goldman, C. Lee-Messer, M. Lungren, D. Rubin, et al. Cross-modal data programming enables rapid medical machine learning. *arXiv preprint arXiv:1903.11101*, 2019.

J. A. Dunnmon, D. Yi, C. P. Langlotz, C. Ré, D. L. Rubin, and M. P. Lungren. Assessment of convolutional neural networks for automated classification of chest radiographs. *Radiology*, page 181422, 2018.

L. Eadicicco. Baidu's Andrew Ng on the future of artificial intelligence, 2017. `http://time.com/4631730/andrew-ng-artificial-intelligence-2017/`Time [Online; posted 11-January-2017].

G. Elidan and N. Friedman. Learning hidden variable networks: The information bottleneck approach. *Journal of Machine Learning Research*, 6:81–127, 2005.

A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-Level Classification of Skin Cancer with Deep Neural Networks. *Nature*, 542(7639):115–118, 1 2017.

A. Fawzi, H. Samulowitz, D. Turaga, and P. Frossard. Adaptive data augmentation for image classification. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 3688–3692. IEEE, 2016.

J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

J. A. Fries, S. Wu, A. Ratner, and C. Ré. Swellshark: A generative model for biomedical named entity recognition without labeled data. *CoRR*, abs/1704.06360, 2017.

J. A. Fries, P. Varma, V. S. Chen, K. Xiao, H. Tejeda, P. Saha, J. Dunnmon, H. Chubb, S. Maskatia, M. Fiterau, S. Delp, E. Ashley, C. Ré, and J. Priest. Weakly supervised classification of rare aortic valve malformations using unlabeled cardiac mri sequences. *Nature Communications*, 2019.

H. Gao, G. Barbier, R. Goolsby, and D. Zeng. Harnessing the crowdsourcing power of social media for disaster relief. Technical report, DTIC Document, 2011.

A. Ghosh, S. Kale, and P. McAfee. Who moderates the moderators?: Crowdsourcing abuse detection in user-generated content, 2011.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014a.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

B. Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

E. Greensmith, P. L. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov): 1471–1530, 2004.

V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, R. Kim, R. Raman, P. C. Nelson, J. L. Mega, and D. R. Webster. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *JAMA*, 316(22): 2402, 12 2016.

S. Gupta and C. D. Manning. Improved pattern learning for bootstrapped entity extraction. In *CoNLL*, 2014.

K. Guu, P. Pasupat, E. Z. Liu, and P. Liang. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. *arXiv preprint arXiv:1704.07926*, 2017.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, Mar. 2003.

I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh. *Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 3540354875.

B. Hancock, P. Varma, S. Wang, M. Bringmann, P. Liang, and C. Ré. Training classifiers with natural language explanations. In *Proceedings of ACL*, 2018.

S. Hauberg, O. Freifeld, A. B. L. Larsen, J. Fisher, and L. Hansen. Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation. In *Artificial Intelligence and Statistics*, pages 342–350, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Meeting of the Association for Computational Linguistics (ACL)*, 1992.

M. Heath, K. Bowyer, D. Kopans, R. Moore, and W. P. Kegelmeyer. The digital database for screening mammography. In *Proceedings of the 5th international workshop on digital mammography*, pages 212–218. Medical Physics Publishing, 2000.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

R. Hoffmann, C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the ACL*, 2011.

J. Honorio. Lipschitz parametrization of probabilistic graphical models. *arXiv preprint arXiv:1202.3733*, 2012.

G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Comprehensive and reliable crowd assessment algorithms. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, 2015.

S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM, 2011.

D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *Advances in neural information processing systems*, pages 1953–1961, 2011.

A. Karpathy. Software 2.0. `https://medium.com/@karpathy/software-2-0-a64152b37c35`, 2017.

N. Khandwala, A. Ratner, J. Dunnmon, R. Goldman, M. Lungren, D. Rubin, and C. Ré. Cross-modal data programming for medical images. *NIPS ML4H Workshop*, 2017.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

D. Koller, N. Friedman, and F. Bach. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *arXiv preprint arXiv:1602.07332*, 2016.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

M.-A. Krogel and T. Scheffer. Multi-relational learning, text mining, and semi-supervised learning for functional genomics. *Machine Learning*, 57(1-2):61–81, 2004.

J. B. Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18(2):95–138, 1977.

J. P. Ku, J. L. Hicks, T. Hastie, J. Leskovec, C. Ré, and S. L. Delp. The Mobilize center: an NIH big data to knowledge center to advance human movement research and improve mobility. *Journal of the American Medical Informatics Association*, 22(6):1120–1125, 2015.

V. Kuleshov, J. Ding, C. Vo, B. Hancock, A. Ratner, Y. Li, C. Ré, S. Batzoglou, and M. Snyder. A machine-compiled database of genome-wide association studies. *Nature Communications*, 2019.

S. Laine and T. Aila. Temporal ensembling for semi-supervised learning. *CoRR*, abs/1610.02242, 2016.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web Journal*, 2014.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 5(Apr):361–397, 2004.

H. Li, B. Yu, and D. Zhou. Error rate analysis of labeling by crowdsourcing. In *ICML Workshop: Machine Learning Meets Crowdsourcing. Atalanta, Georgia, USA*, 2013.

Y. Li, J. Gao, C. Meng, Q. Li, L. Su, B. Zhao, W. Fan, and J. Han. A survey on truth discovery. *SIGKDD Explor. Newsl.*, 17(2), 2015.

P. Liang. Stanford cs229t notes. 2019. URL `https://web.stanford.edu/class/cs229t/2015/notes.pdf`.

P. Liang, M. I. Jordan, and D. Klein. Learning from measurements in exponential families. In *International Conference on Machine Learning (ICML)*, 2009.

P.-L. Loh and M. J. Wainwright. Structure estimation for discrete graphical models: Generalized covariance matrices and their inverses. *Annals of Statistics*, 41(6):3022–3049, 2013.

X. Lu, B. Zheng, A. Velivelli, and C. Zhai. Enhancing text categorization with semantic-enriched representation and training data augmentation. *Journal of the American Medical Informatics Association*, 13(5):526–535, 2006.

G. Lugosi. Learning with an unreliable teacher. *Pattern Recognition*, 25(1):79 – 87, 1992.

N. Mallinar, A. Shah, R. Ugrani, A. Gupta, M. Gurusankar, T. K. Ho, Q. V. Liao, Y. Zhang, R. K. E. Bellamy, R. Yates, C. Desmarais, and B. McGregor. Bootstrapping conversational agents with weak supervision. *CoRR*, abs/1812.06176, 2018.

E. K. Mallory, C. Zhang, C. Ré, and R. B. Altman. Large-scale extraction of gene interactions from full-text literature using deepdive. *Bioinformatics*, 2015.

G. S. Mann and A. McCallum. Generalized expectation criteria for semi-supervised learning with weakly labeled data. *Journal of Machine Learning Research*, 11:955–984, 2010.

N. Meinshausen and P. Bühlmann. High-dimensional graphs and variable selection with the lasso. *The Annals of Statistics*, 34(3):1436–1462, 2006.

Z. Meng, B. Eriksson, and A. O. H. III. Larning latent variable gaussian graphical models. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, Beijing, China, 2014.

C. Metz. Google's hand-fed AI now gives answers, not just search results, 2016. `https://www.wired.com/2016/11/googles-search-engine-can-now-answer-questions-human-help/`Wired [Online; posted 29-November-2016].

A. Minonne, D. Schubmehl, J. George, and J. Cai. Worldwide semiannual cognitive/artificial intelligence systems spending guide. Technical report, International Data Corporation, 2017.

M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Meeting of the Association for Computational Linguistics (ACL)*, 2009.

M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677*, 2015.

P. Molino. Ludwig: a type-based declarative deep learning toolbox. *To appear*, 2019.

N. Natarajan, I. S. Dhillon, P. K. Ravikumar, and A. Tewari. Learning with noisy labels. In *Advances in Neural Information Processing Systems 26*. 2013.

National Institutes of Health. Open-i. 2017. URL `https://openi.nlm.nih.gov/`.

A. Y. Ng. Feature selection, l_1 vs. l_2 regularization, and rotational invariance. In *International Conference on Machine Learning (ICML)*, 2004.

D. A. P., C. J. Grondin, R. J. Johnson, D. Sciaky, B. L. King, R. McMorran, J. Wiegers, T. Wiegers, and C. J. Mattingly. The comparative toxicogenomics database: update 2017. *Nucleic Acids Research*, 2016.

S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

P. Papotti, X. Chu, and I. F. Ilyas. Holistic data cleaning: Putting violations into context. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, pages 458–469. IEEE Computer Society, 2013.

F. Parisi, F. Strino, B. Nadler, and Y. Kluger. Ranking and combining multiple predictors without labeled data. *Proceedings of the National Academy of Sciences of the USA*, 111 (4):1253–1258, 2014.

A. e. a. Paszke. Automatic differentiation in pytorch, 2017.

S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3:1333–1356, 2003.

E. Platanios, H. Poon, T. M. Mitchell, and E. J. Horvitz. Estimating accuracy from unlabeled data: A probabilistic logic approach, 2017.

R. Pochampally, A. Das Sarma, X. L. Dong, A. Meliou, and D. Srivastava. Fusing data with correlations. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2014.

A. J. Quinn and B. B. Bederson. Human computation: A survey and taxonomy of a growing field. In *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2011.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

A. Ratner and C. Ré. Knowledge base construction in the machine-learning era. *Queue*, 16 (3):50, 2018.

A. Ratner, C. De Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *Neural Information Processing Systems (NIPS)*, 2016.

A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *Proc. VLDB Endow.*, 11(3):269–282, Nov. 2017a.

A. Ratner, B. Hancock, J. Dunnmon, R. Goldman, and C. Ré. Snorkel metal: Weak supervision for multi-task learning. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, page 3. ACM, 2018.

A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: rapid training data creation with weak supervision. *The VLDB Journal*, Jul 2019a.

A. Ratner, B. Hancock, J. Dunnmon, F. Sala, S. Pandey, and C. Ré. Training complex models with multi-task weak supervision. *AAAI*, 2019b.

A. Ratner, B. Hancock, and C. Ré. The role of massively multi-task and weak supervision in software 2.0. In *Conference on Innovative Data Systems Research*, 2019c.

A. J. Ratner, S. H. Bach, H. R. Ehrenberg, and C. Ré. Snorkel: Fast training set generation for information extraction. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1683–1686. ACM, 2017b.

A. J. Ratner, H. Ehrenberg, Z. Hussain, J. Dunnmon, and C. Ré. Learning to compose domain-specific transformations for data augmentation. In *Advances in neural information processing systems*, pages 3236–3246, 2017c.

P. Ravikumar, M. J. Wainwright, and J. D. Lafferty. High-dimensional Ising model selection using ℓ_1-regularized logistic regression. *The Annals of Statistics*, 38(3):1287–1319, 2010.

P. Ravikumar, M. J. Wainwright, G. Raskutti, and B. Yu. High-dimensional covariance estimation by minimizing ℓ_1-penalized log-determinant divergence. *Electronic Journal of Statistics*, 5:935–980, 2011.

T. Rekatsinas and L. D. Xin. Data integration and machine learning: a natural synergy, 2018. URL `http://www.dataintegration.ml/`.

T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. HoloClean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017a.

T. Rekatsinas, M. Joglekar, H. Garcia-Molina, A. Parameswaran, and C. Ré. SLiMFast: Guaranteed results for data fusion and source reliability. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2017b.

S. Riedel, L. Yao, and A. McCallum. Modeling relations and their mentions without labeled text. In *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, 2010.

B. Roth and D. Klakow. Feature-based models for improving the quality of noisy training data for relation extraction. In *Proceedings of the 22nd ACM Conference on Knowledge management*. ACM, 2013a.

B. Roth and D. Klakow. Combining generative and discriminative model scores for distant supervision. In *Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2013b.

S. Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

M. Sajjadi, M. Javanmardi, and T. Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. *CoRR*, abs/1606.04586, 2016.

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan. Finding a “kneedle” in a haystack: Detecting knee points in system behavior. In *International Conference on Distributed Computing Systems Workshops*, 2011.

R. Sawyer Lee, F. Gimenez, A. Hoogi, and D. Rubin. Curated breast imaging subset of DDSM. In *The Cancer Imaging Archive*, 2016.

R. E. Schapire and Y. Freund. *Boosting: Foundations and algorithms*. MIT press, 2012.

J. Schulman, N. Heess, T. Weber, and P. Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3528–3536, 2015.

H. J. Scudder. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Infomation Theory*, 11:363–371, 1965.

B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, and C. Ré. Incremental knowledge base construction using deepdive. *Proceedings of the VLDB Endowment*, 8(11):1310–1321, 2015.

L. Sixt, B. Wild, and T. Landgraf. Rendergan: Generating realistic labeled data. *arXiv preprint arXiv:1611.01331*, 2016.

J. T. Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015.

R. Stewart and S. Ermon. Label-free supervision of neural networks with physics and other domain knowledge. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. *arXiv preprint arXiv:1707.02968*, 2017.

M. Surdeanu and H. Ji. Overview of the english slot filling track at the tac2014 knowledge base population evaluation. In *Proc. Text Analysis Conference (TAC2014)*, 2014.

S. Takamatsu, I. Sato, and H. Nakagawa. Reducing wrong labels in distant supervision for relation extraction. In *Meeting of the Association for Computational Linguistics (ACL)*, 2012.

C. H. Teo, A. Globerson, S. T. Roweis, and A. J. Smola. Convex learning with invariances. In *Advances in neural information processing systems*, pages 1489–1496, 2008.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B*, 58(1):267–288, 1996.

J. A. Tropp. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230, 2015.

S. Uhlich, M. Porcu, F. Giron, M. Enenkl, T. Kemp, N. Takahashi, and Y. Mitsufuji. Improving music source separation based on deep neural networks through data augmentation and network blending. *Submitted to ICASSP*, 2017.

P. Varma and C. Ré. Snuba: Automating weak supervision to label training data. In *Proceedings of VLDB*, 2019.

P. Varma, B. He, P. Bajaj, N. Khandwala, I. Banerjee, D. Rubin, and C. Ré. Inferring generative model structure with static analysis. In *Proceedings of NIPS*, 2017.

P. Varma, F. Sala, A. He, A. Ratner, and C. Ré. Learning dependency structures for weak supervision models. *arXiv preprint arXiv:1903.05844*, 2019.

P. Verga, D. Belanger, E. Strubell, B. Roth, and A. McCallum. Multilingual relation extraction using compositional universal schema. *arXiv preprint arXiv:1511.06396*, 2015.

C.-H. Wei, Y. Peng, R. Leaman, D. A. P., C. J. Mattingly, J. Li, T. Wiegers, and Z. Lu. Overview of the BioCreative V chemical disease relation (CDR) task. In *BioCreative Challenge Evaluation Workshop*, 2015.

R. Weischedel, E. Hovy, M. Marcus, M. Palmer, R. Belvin, S. Pradhan, L. Ramshaw, and N. Xue. Ontonotes: A large training corpus for enhanced processing. *Handbook of Natural Language Processing and Machine Translation. Springer*, 2011.

D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Recurrent policy gradients. *Logic Journal of IGPL*, 18(5):620–634, 2010.

C. Wu, H. Zhao, H. Fang, and M. Deng. Graphical model selection with latent variables. *Electronic Journal of Statistics*, 11:3485–3521, 2017.

S. Wu, L. Hsiao, X. Cheng, B. Hancock, T. Rekatsinas, P. Levis, and C. Ré. Fonduer: Knowledge base construction from richly formatted data. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1301–1316. ACM, 2018.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

M.-C. Yuen, I. King, and K.-S. Leung. A survey of crowdsourcing systems. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pages 766–773. IEEE, 2011.

O. F. Zaidan and J. Eisner. Modeling annotators: A generative approach to learning from annotator rationales. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016a.

C. Zhang, C. Ré, M. Cafarella, C. De Sa, A. Ratner, J. Shin, F. Wang, and S. Wu. DeepDive: Declarative knowledge base construction. *Commun. ACM*, 60(5):93–102, 2017a.

Y. Zhang, X. Chen, D. Zhou, and M. I. Jordan. Spectral methods meet EM: A provably optimal algorithm for crowdsourcing. *Journal of Machine Learning Research*, 17:1–44, 2016b.

Y. Zhang, V. Zhong, D. Chen, G. Angeli, and C. D. Manning. Position-aware attention and supervised data improve slot filling, 2017b.

B. Zhao, B. I. Rubinstein, J. Gemmell, and J. Han. A Bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.

P. Zhao and B. Yu. On model selection consistency of lasso. *Journal of Machine Learning Research*, 7:2541–2563, 2006.

J. Zhu, N. Lao, and E. P. Xing. Grafting-Light: Fast, incremental feature selection and structure learning of Markov random fields. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2010.