Data Programming with DDLite: Putting Humans in a Different Part of the Loop

Henry R. Ehrenberg, Jaeho Shin, Alexander J. Ratner, Jason A. Fries, Christopher Ré
Stanford University
353 Serra Mall
Stanford, California 94305
{henryre, jaeho, ajratner, jfries, chrismre}@cs.stanford.edu

ABSTRACT

Populating large-scale structured databases from unstructured sources is a critical and challenging task in data analytics. As automated feature engineering methods grow increasingly prevalent, constructing sufficiently large labeled training sets has become the primary hurdle in building machine learning information extraction systems. In light of this, we have taken a new approach called *data programming* [7]. Rather than hand-labeling data, in the data programming paradigm, users generate large amounts of noisy training labels by programmatically encoding domain heuristics as simple rules. Using this approach over more traditional distant supervision methods and fully supervised approaches using labeled data, we have been able to construct knowledge base systems more rapidly and with higher quality. Since the ability to quickly prototype, evaluate, and debug these rules is a key component of this paradigm, we introduce DDLite, an interactive development framework for data programming. This paper reports feedback collected from DDLite users across a diverse set of entity extraction tasks. We share observations from several DDLite hackathons in which 10 biomedical researchers prototyped information extraction pipelines for chemicals, diseases, and anatomical named entities. Initial results were promising, with the disease tagging team obtaining an F_1 score within 10 points of the state-of-the-art in only a single day-long hackathon's work. Our key insights concern the challenges of writing diverse rule sets for generating labels, and exploring training data. These findings motivate several areas of active data programming research.

1. INTRODUCTION

Knowledge base construction (KBC) is the task of extracting structured facts in the form of *entities* and *relations* from unstructured input data such as text documents, and has received critical attention from both academia and industry over the last several years. DeepDive, our framework for large-scale KBC, is used in a wide range of fields including law enforcement, pharmaceutical and clinical genomics, and paleobiology[1]. Our experience shows that leveraging user domain knowledge is critical for building high-quality information extraction systems. However, encoding domain knowledge in manual features and hand-tuned algorithms resulted in lengthy development cycles. To enable faster development of these machine learning-based systems, we explored automated feature extraction libraries [2, 13] and accelerated inference methods [14, 9]. After implementing these improvements, we found that constructing large, labeled training data sets had become the primary development bottleneck.

We introduced the *data programming method* [7] as a framework to address this challenge. Data programming is a new paradigm—extending the idea of distant supervision—in which developers focus on programmatically creating labeled data sets to build machine learning systems. Domain knowledge is encoded in a set of heuristic labeling rules, referred to as *labeling functions*. Each labeling function emits labels for a subset of the input data. Collectively, the labeling functions generate a large, noisy, and potentially conflicting set of labels for the training data. Our data programming research investigates how best to model and denoise the evidence provided by labeling functions. Key points are summarized in Section A.

The primary contribution of this paper is DDLite, a novel system for creating information extraction applications using data programming. DDLite provides a lightweight platform for rapidly creating, evaluating, and debugging labeling functions. Users interact with DDLite through Jupyter Notebooks,[2] and all parts of the information extraction pipeline are designed to minimize ramp up time. DDLite has simple Python syntax, does not require complex setup with databases as DeepDive does, easily connects to domain-specific libraries, and has self-contained tooling for core pipeline tasks including data preprocessing, candidate and feature extraction, labeling function evaluation, and statistical learning and inference.

DDLite has users focus primarily on this iterative development of labeling functions, rather than on the traditional machine learning development task of feature engineering. Recently, automated feature generation methods have attained empirical successes and become prevalent in machine learning systems. These approaches generally require large training sets—such as those created by the data programming approach. We are motivated by the idea that labeling function development may be a far more efficient and intuitive task for non-expert users. We have seen many users struggle with feature engineering, as the optimality of a feature is a function of the statistics of the training set and model. On the other hand, a labeling function has a simple and intuitive optimality criterion: that it labels subsets of the data correctly. Initial results suggest that writing and evaluating labeling functions in DDLite is indeed a far

[1] http://deepdive.stanford.edu/showcase/apps

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
HILDA'16, June 26 2016, San Francisco, CA, USA
©2016 ACM. ISBN 978-1-4503-4207-0/16/06 ...$15.00
DOI: http://dx.doi.org/10.1145/2939502.2939515.

[2] http://ipython.org/notebook.html

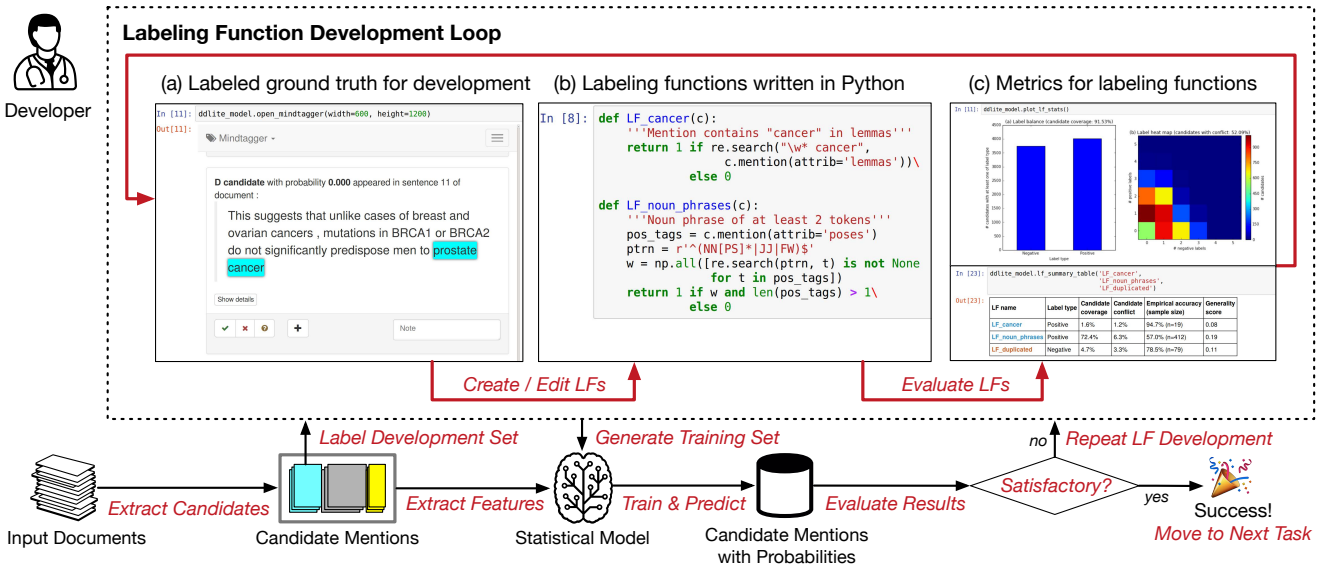


Figure 1: DDLite users rapidly prototype information extraction systems using the data programming method. The workflow is focused on labeling function iteration, in which the user creates, revises, and debugs rules in response to feedback from DDLite. The entire pipeline covers data preprocessing, candidate extraction, labeling function iteration, and learning and evaluation.

easier and faster process for users.

Data programming and DDLite address a core challenge in creating machine learning systems: obtaining large collections of labeled data is difficult for most real-world domains. *Crowdsourcing* is a popular, general method for collecting large volumes of training data with reasonable quality at relatively low cost. However, many tasks involving technical data require domain expertise, and thus, cannot be crowdsourced. Professional readers and domain experts often collect, or *curate* labeled data to build scientific knowledge bases, such as PharmGKB [12], MeSH [8], PubTator[11], and PaleoBioDB [1]. This approach yields highly accurate data, but suffers from scalability and cost constraints. Users of DDLite and the data programming method devote effort to writing labeling functions that can generate new labels from more data, rather than labeling subsets of data directly for training purposes. DDLite builds on the guided data exploration techniques of machine teaching systems [10] by enabling users to directly encode domain expertise as labeling functions rather than labeled examples and engineered features. The data programming approach also allows for instantaneous performance feedback as users rapidly iterate on labeling functions, whereas more feature engineering-based approaches cannot evaluate performance without training a model.

We built DDLite to address new challenges arising from incorporating human supervision in a different part of the development loop. Some challenges were easy to anticipate and we engineered DDLite features to address them before shipping it out to users. For instance, characterization of the labeling functions, such as accuracy and coverage, as well as easy data exploration methods are vital for users to understand when to debug, revise, or add new rules. However, it is not readily clear which other performance representations or exploration methods would help users make more concrete and informed decisions. DDLite is a rapidly evolving system, and we rely heavily on direct user feedback to address these challenges and motivate development.

In the following sections, we describe data programming with DDLite for rapid information extraction. We report our experience and observations from a biomedical hackathon using DDLite to try to achieve benchmark performance. We then conclude by raising a few technical questions we believe are important for accelerating human-in-the-loop machine learning system development under our proposed data programming paradigm.

2. DDLITE WORKFLOW AND HACKATHON CASE STUDY

Using DeepDive, teams of developers have built large-scale, competition-winning KBC applications, usually spending months building and tuning these systems [3, 5, 6]. We now describe the lighter-weight process of using data programming and DDLite. Given a set of input documents, our goal is to produce a set of extracted entity or relation *mentions*. We do this by first heuristically extracting a set of *candidate* mentions, then learning a model using the training data to predict which of these candidates are actual mentions of the desired entity or relation type. The user follows four core steps in this process as shown in Figure 1:

1. **Data Preprocessing:** Given a set of input text documents, basic preprocessing of the raw text is performed, which may include using domain-specific tokenizers or parsers.
2. **Candidate Extraction:** The user defines the set of candidate mentions to consider during training by matching the parsed text to dictionaries and regular expression patterns. DDLite provides a library of general candidate extraction operators.
3. **Labeling Function Development:** Following the paradigm of data programming, the user develops a set of labeling functions by iterating between exploring data (and optionally hand-labeling small subsets for error analysis) and analyzing labeling function performance.
4. **Learning and Evaluation:** In the DDLite setup, features are automatically generated for the candidates, and then the model is trained using the labeling functions developed so far. The user then analyzes system performance as measured on a separate labeled *holdout set*.

Our goal in this paper is to evaluate the effectiveness of this new process, specifically whether it constitutes a more efficient interaction model for users. To this end, we organized a hackathon for creating biomedical entity tagging systems for academic text, using open abstracts and full text documents from PubMed Central (PMC). The hackathon consisted of 10 participants working in bioinformatics labs at Stanford. Several teams worked on three separate tagging systems, which were designed to identify all entity mentions of chemical/drug names, diseases, and human anatomical locations (such as muscle groups or internal organs).

We now describe the DDLite workflow in more detail, in the context of the bioinformatics hackathon:

Data Preprocessing. The first step is to preprocess the raw text, parsing it into sentences and words, and adding standard annotations such as lemmatized word forms, part-of-speech tags, and dependency path structure which provide useful signal for downstream labeling and learning tasks. Experience has shown us that providing an easily customizable preprocessing pipeline is important, especially when working in technical domains where standard tools often fail. In DDLite, we strike a balance by utilizing a lightweight parallelization framework which has CoreNLP[3] as a simple "push-button" default, but which also allows users to swap in custom tokenizers, taggers, and parsers.

EXAMPLE 1. *Some teams in our hackathon—such as the disease extraction team—used the standard tools without issue. However, many had domain-specific issues with various parts of the pipeline. This was most striking in the chemical name task, where complicated surface forms (e.g., "9-acetyl-1,3,7-trimethyl- pyrimidinedione") caused tokenization errors which had significant effects on system recall. This was corrected by utilizing a domain-specific tokenizer.*

Candidate Extraction. The next stage of the DDLite workflow is the extraction of candidate mentions of entities or relations, which our learned model will then classify as true or false. When there is not a clearly defined, closed candidate set (such as the set of human genes), feedback showed that candidate extraction can be a difficult stage for users. The core goal is to strike a balance between being too precise—since end-system recall is trivially upper-bounded by the recall at the candidate extraction stage—and being too broad—since this can lead to combinatorial blow-up in the number of candidates considered.

Based on user feedback, our solution in DDLite is to provide a simple set of compositional dictionary and pattern matching-based operators, thus guiding users to construct simple yet expressive candidate extractors that easily integrate domain-specific resources. For example, in our hackathon, the separate codebase ddbiolib[4] was used to interface with existing biomedical ontologies and build entity dictionaries. ddbiolib utilities were then directly connected to DDLite.

EXAMPLE 2. *In the disease tagging example, the phrase "prostate cancer" (Figure 1(a)) was identified as a candidate by matching it to a user-supplied dictionary of common disease names, using DDLite's* `DictionaryMatch` *operator. However, error analysis suggested that this approach was limiting recall, so the team also tried to augment with compound noun phrases where one of the words matched a custom word-form pattern, such as* `'degen.+'`*. This identified additional compound phrases not in their base dictionary, such as "macular degeneration".*

Labeling Function Iteration. The primary focus of DDLite is the iterative development of labeling functions. A labeling function is written as a simple Python function, which takes as input a candidate entity or relationship mention, along with relevant local context. This includes the annotations from preprocessing for the sentence the mention occurs in. Labeling functions can depend on external libraries—such as ddbiolib in our hackathon—and thus represent an extremely flexible means of encoding domain knowledge. Labeling functions can either abstain from labeling (by emitting a `0`) or label the candidate as positive or negative case of an actual entity or relation mention (by emitting a `-1` or `1`, respectively).

EXAMPLE 3. *Figure 1(b) shows two labeling functions written by the disease tagging team:* `LF_noun_phrases` *and* `LF_cancer`*. The first marks the candidate as positive if "cancer" is the word lemmas composing the mention, and the second marks it as positive if the mention is a noun phrase with at least two tokens.*

In order to evaluate and debug labeling functions during development, and to help generate ideas for new ones, users iteratively hand-label some subset of the data within the DDLite framework (Fig. 1(a)). We refer to the aggregate of all such hand-labels generated by the user during labeling function development as the *development set*. To prevent bias, we do not use this set for any end-system evaluation. We also assume that the quantity of labeled data in the development set is far less than what would be needed for a traditional supervised learning scenario.

Several key metrics are then computed to characterize the performance of labeling functions. Figure 1(c) shows DDLite plots (illustrating the performance of the entire set) and tables (detailing the performance of individual functions), which summarize these key metrics:

- *Coverage* is the fraction of candidates which were labeled, either by any labeling function or by a specific labeling function.
- *Empirical accuracy* is the class-dependent accuracy of a given labeling function with respect to the development set.
- *Empirical accuracy generalization score* is computed as the absolute difference between the accuracy scores of a given labeling function applied to two different random splits of the development set.
- *Conflict* is the fraction of candidates that at least one labeling function labeled as positive and another as negative. Counterintuitively, controversial candidates are actually highly useful for learning noise-aware models in the framework of data programming [7], and can also be used for debugging the current labeling function set.

As opposed to engineered features, labeling functions can be evaluated during development without training the full model. DDLite is therefore able to provide on-the-spot performance feedback as plots and tables, allowing the user to iterate more rapidly on system design and navigate the the key decision points in the data programming workflow:

- *Improve coverage or accuracy?* When users develop labeling functions, they can either focus on improving coverage or accuracy. If the plots show low coverage, the user should explore unlabeled training examples and write new labeling functions based on novel observations. If labeling functions have low accuracies, or low accuracy generalization scores, this could be indication that labeling functions have bugs or need to be revised.

[3] `http://stanfordnlp.github.io/CoreNLP/`

[4] `https://github.com/HazyResearch/ddbiolib`

- *Develop labeling functions or run learning algorithm?* When coverage, accuracy, and conflict are sufficient, users can proceed to learn a predictive model on the data. Scores are reported on the blind test set, and the user can continue to iterate on labeling functions or finalize their application.

EXAMPLE 4. *The plots in Figure 1(c) reveal that the disease team's labeling function set has high coverage on the development set. The sample of labeling functions detailed in the table shows a mix of rule types, with a very high-accuracy, low-coverage labeling function (`LF_cancer`) and a very high-coverage, low-accuracy one (`LF_noun_phrase`). In particular, we note that although `LF_noun_phrase` has a low accuracy as shown in Figure 1(c), it has high coverage and accuracy generalization score, indicating that it may contribute meaningfully to the model.*

The most significant observation during the labeling function iteration phase at the hackathon was that users are naturally inclined to write highly accurate rules, rather than ones with high coverage The observed bias towards accuracy also stems from the challenges of crafting high-coverage rules *de-novo*, motivating a key area of future work.

EXAMPLE 5. *The anatomy team wrote rules looking for the tokens "displacement", "stiffness", and "abnormality" near mentions, since these generally accompany true mentions of anatomical structures. While each of the three labeling functions had very high accuracy, they abstained on more than 99.9% of the candidates. The anatomy team eventually wrote a rule simply checking for nouns and adjectives in the mentions. This rule had high coverage, labeling over 400 candidates in the training set with 52.8% accuracy.*

Learning and Evaluation. Once users are satisfied with their current labeling function set as judged by coverage, conflict, and accuracy metrics, they fit a model to the training set and its performance is automatically evaluated on the test set. Features are generated automatically in DDLite at the candidate extraction stage. Although the envisioned DDLite workflow is centered around labeling function development and not feature engineering, users can fully interact with and customize the feature set if desired.

In the data programming paradigm, we first model the accuracy of each labeling function, and then train a model on the features using a *noise-aware* loss function to take into account the inaccuracies of imperfect and potentially conflicting labeling functions. The formal guarantees that we show for this approach are outside the scope of this paper, but details can be found in [7]. This general data programming approach can handle more complex models that incorporate logical relations between variables—such as Markov logic network semantics, as in DeepDive—but for our initial version of DDLite we use a logistic regression-based model as user feedback indicates that this is sufficient for rapid prototyping and iteration.

The test set precision, recall, and F_1 score indicate how well the characterization of the data provided by the labeling functions generalizes as a representation over the features. These scores also serve as a means to discriminate between nuanced labeling function tradeoffs. For instance, it may not be immediately obvious from the on-the-spot feedback metrics whether a user should sacrifice some accuracy of a particular labeling function for a coverage increase. However, end system performance can guide subtle design decisions like these.

Hackathon Results. Our goal in this study was to evaluate the effectiveness of the data programming and DDLite approach, and to do this we set an extremely ambitious goal of coming close to benchmark performance in less than a day, *using no hand-labeled training data*. We emphasize that these benchmark systems used manually labeled training sets, and took months or years to build. Although we fell short of state-of-the-art performance, our overall results were very encouraging with respect to this aspect of radically increasing user interaction efficiency. Table 1 summarizes the labeling function (LF) sets produced by the hackathon teams, where Coverage refers to to the percent of the dataset used during the hackathon that was labeled by at least one labeling function, and Mean accuracy refers to the mean individual labeling function accuracy on the test set.

Table 1: Teams, Tasks, and Labeling Functions

Entity (Team)	Team size	# LFs	Coverage	Mean accuracy
Anatomy (1)	4	10	6.8%	75.0%
Anatomy (2)	3	14	75.5%	86.6%
Diseases (1)	1	4	100%	81.6%
Diseases (2)	2	11	95.3%	79.3%

Table 2 reports scores from a data programming logistic regression model trained using the labeling functions and automatically generated features. Two test sets were used for reporting, composed of colleague-annotated data (User) and independent benchmark data (NCBI Dev), neither of which were viewed during development. As a rough baseline for current disease tagging performance, we use scores reported by Dogan et al. [4] computed using BANNER, a biomedical NER tagging system.

Table 2: Disease Name Tagging Performance

Entity (Team)	Holdout annotations	Method	P	R	F_1
Anatomy (1)	User	DDLite	0.63	0.96	0.76
Anatomy (2)	User	DDLite	0.62	1.00	0.77
Diseases (1)	User	DDLite	0.77	0.88	0.82
Diseases (2)	User	DDLite	0.75	0.98	0.85
Diseases (1+2)	NCBI Dev	DDLite	0.78	0.62	0.69
Diseases (1+2)	NCBI Dev	DDL + LSTM	0.76	0.69	0.72
Diseases	NCBI Dev	BANNER	0.82	0.82	0.82

We note that the hackathon user-annotated scores for diseases and anatomy tend to overestimate recall due to small test set sizes. The NCBI development set, which consists of curated gold standard data adjudicated by independent annotators, is a more representative sample of mentions and provides a better assessment of true recall.

The first anatomy team tended to write very accurate, but low-coverage labeling functions that returned negative labels. This resulted in poor system precision. For disease tagging, the lower recall scores in benchmark vs. user-annotated reflect current limitations in DDLite's candidate extraction step.

We experimented with using features generated by long short-term memory (LSTM) recurrent neural networks trained using the labeling function outputs, instead of the default text sequence-based features in DDLite. We observed a three point F_1 score boost in the disease application.

While these initial DDLite scores fall short of the benchmark set by manually supervised systems, they represent the collective

work of a single afternoon versus weeks or months commonly seen in DeepDive development cycles.

3. FUTURE WORK AND CONCLUSIONS

Data programming with DDLite is a rapidly evolving methodology, responding to advances in theoretical work and feedback from users like the biomedical hackathon participants. Currently, there are three major areas of focus for DDLite development:

Useful Metrics. Coverage and empirical accuracy scores provide an essential evaluation of labeling functions, and users rely heavily on these basic metrics. However, other characterizations can be useful for developing labeling function sets. For instance, conflicts play a crucial role in denoising individually low accuracy labeling functions and improving the set of accuracy estimates [7]. How best to present a depiction of conflict that leads to performance-improving action is an open question. Writing rules to promote conflict is unintuitive for most users, and so informative metrics must be combined with directed development principles.

Data Exploration and Rule Suggestion. After creating and receiving feedback for an initial set of labeling functions, candidate exploration via random sampling is no longer an effective means of rule discovery. Without support for guided data exploration, the labeling function writing task can resemble feature engineering, where the development process stagnates due to undirected effort. Observing that writing high coverage rules *de-novo* was especially challenging, we added sampling of only candidates not labeled by any rule in DDLite. We are actively developing other smart sampling techniques for efficient, guided data exploration DDLite. For example, exploring candidates with high conflict or low empirical accuracy from the current set of labeling functions is useful for improving precision. We are also investigating methods for suggesting new, high coverage rules directly, including a curated library of examples and token frequency-based clustering approaches.

Principled Management of Labeled Data. While many domains have gold labels which can be used for external validation, in practice most problems require users to develop their own benchmark datasets. Having independent annotators create labeled holdout sets is another hidden bottleneck to the overall development process. Because individual DDLite users are actively annotating during development, there is clearly value in enabling multiple users to exchange their labeled data. To address bias concerns, i.e., users writing labeling functions that are evaluated against their own annotations, we are implementing a more principled way of managing labeled data. We can maintain a record of annotation provenance such that the evaluation metrics for labeling functions are strictly computed from labeled candidates that are disjoint from their own annotations. There is also a need for a well defined protocol when working in a collaborative environment, since labeling function sharing may cause overfitting or biased performance evaluations.

Given the rise of automated feature extractors, we argued that human-in-the-loop training set development plays an important role in creating information extraction systems. We described the DDLite framework, reported on its initial use by 10 biomedical researchers, and identified key areas of future work from direct user feedback.

Acknowledgments. A special thanks to Abhimanyu Banerjee, Alison Callahan, David Dindi, Madalina Fiterau, Jennifer Hicks, Emily Mallory, and Raunaq Rewari for participating in our inaugural biohackathon.

We gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA) SIMPLEX program under No. N66001-15-C-4043, the National Science Foundation (NSF) CAREER Award under No. IIS- 1353606, the Office of Naval Research (ONR) under awards No. N000141210041 and No. N000141310129, the Sloan Research Fellowship, the Moore Foundation, Toshiba, and Intel. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, NSF, ONR, or the U.S. government.

4. REFERENCES

[1] The paleobiology database. http://paleobiodb.org/.

[2] M. R. Anderson, D. Antenucci, V. Bittorf, M. Burgess, M. J. Cafarella, A. Kumar, F. Niu, Y. Park, C. Ré, and C. Zhang. Brainwash: A data system for feature engineering. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org, 2013.

[3] G. Angeli, S. Gupta, M. Jose, C. D. Manning, C. Ré, J. Tibshirani, J. Y. Wu, S. Wu, and C. Zhang. Stanford's 2014 slot filling systems. *TAC KBP*, 2014.

[4] R. I. Doğan and Z. Lu. An improved corpus of disease mentions in pubmed citations. In *Proceedings of the 2012 workshop on biomedical natural language processing*, pages 91–99. Association for Computational Linguistics, 2012.

[5] E. K. Mallory, C. Zhang, C. Ré, and R. B. Altman. Large-scale extraction of gene interactions from full-text literature using DeepDive. *Bioinformatics*, 32(1):106–113, Jan. 2015.

[6] S. E. Peters, C. Zhang, M. Livny, and C. Ré. A machine reading system for assembling synthetic paleontological databases. *PLoS one*, 9(12):e113523, 2014.

[7] A. Ratner, C. D. Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. 2016. https://arxiv.org/abs/1605.07723.

[8] F. Rogers. Medical subject headings. *Bulletin of the Medical Library Association*, 51:114, 1963.

[9] J. Shin, S. Wu, F. Wang, C. D. Sa, C. Zhang, and C. Ré. Incremental knowledge base construction using DeepDive. *PVLDB*, 8(11):1310–1321, 2015.

[10] P. Simard, D. Chickering, A. Lakshmiratan, D. Charles, L. Bottou, C. G. J. Suarez, D. Grangier, S. Amershi, J. Verwey, and J. Suh. Ice: Enabling non-experts to build models interactively for large-scale lopsided problems, 2014.

[11] C.-H. Wei, H.-Y. Kao, and Z. Lu. Pubtator: a web-based text mining tool for assisting biocuration. *Nucleic acids research*, page gkt441, 2013.

[12] M. Whirl-Carrillo, E. McDonagh, J. Hebert, L. Gong, K. Sangkuhl, C. Thorn, R. Altman, and T. E. Klein. Pharmacogenomics knowledge for personalized medicine. *Clinical pharmacology and therapeutics*, 92(4):414, 2012.

[13] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. In C. E. Dyreson, F. Li, and M. T. Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 265–276. ACM, 2014.

[14] C. Zhang and C. Re. DimmWitted: A study of main-memory statistical analytics. *PVLDB*, 7(12):1283–1294, 2014.

APPENDIX

A. LEARNING PROCEDURES FOR DATA PROGRAMMING

We assume that mention and true class label pairs $(x, y) \in \mathcal{X} \times \{-1, 1\}$ are distributed according to π. Given the feature mapping $f : \mathcal{X} \to \mathbb{R}^d$, our objective is to learn a set of linear weights $w \in \mathbb{R}^d$ over the features to minimize the logistic loss with respect to the true class label:

$$l(w) = \mathbf{E}_{(x,y)\sim\pi}\left[\log(1 + \exp(-w^T f(x)y))\right].$$

We denote the set of m user-specified labeling functions as $\lambda : \mathcal{X} \to \{-1, 0, 1\}^m$ and their instantiation over the training data $S \subset \mathcal{X}$ as $\Lambda \in \{-1, 0, 1\}^m$. To provide an overview of learning in the data programming paradigm, we consider the case where we assume that the labeling functions label each example independently, given the true class label. The generative framework is flexible, and allows dependency modeling between the labeling functions with only minor modification.

Each labeling function λ_i is assumed to have probability β_i of labeling any given example, and probability α_i of labeling an example correctly given that it does indeed label it. Assuming for simplicity that the classes are balanced, the model distribution is given by

$$\mu_{\alpha,\beta}(\Lambda, Y) = \frac{1}{2}\prod_{i=1}^{m}\left(\beta_i\alpha_i\mathbf{1}_{\{\Lambda_i=Y\}} + \beta_i(1-\alpha_i)\mathbf{1}_{\{\Lambda_i=-Y\}} + (1-\beta_i)\mathbf{1}_{\{\Lambda_i=0\}}\right).$$

We then solve a maximum likelihood estimation problem to learn parameter estimates $\hat{\alpha}$ for $\alpha = \{\alpha_i\}_{i=1}^m$, and $\hat{\beta}$ for $\beta = \{\beta_i\}_{i=1}^m$:

$$(\hat{\alpha}, \hat{\beta}) = \arg\max_{\alpha,\beta} \sum_{x\in S} \log \mathbf{P}_{(\Lambda,Y)\sim\mu_{\alpha,\beta}}(\Lambda = \lambda(x)).$$

Note that this estimation does not depend on the feature set f. DDLite uses stochastic gradient descent and Gibbs sampling to solve this problem. Given the estimates $\hat{\alpha}$ and $\hat{\beta}$, we use these to learn the weights w over the features. We pose the following regularized logistic regression problem in which we minimize the noise-aware empirical risk:

$$\hat{w} = \arg\min_w L_{\hat{\alpha},\hat{\beta}}(w; S) = \arg\min_w \frac{1}{|S|}\sum_{x\in S} l_\Lambda(w) + \rho R(w),$$

where

$$l_\Lambda(w) = \mathbb{E}_{(\Lambda,Y)\sim\mu_{\hat{\alpha},\hat{\beta}}}\left[\log(1 + \exp(-w^T f(x)y)|\Lambda = \lambda(x)\right].$$

DDLite supports elastic-net regularization, where

$$R(w) = R_\gamma(w) = \frac{\gamma}{2}\|w\|_2 + \frac{1-\gamma}{2}\|w\|_1$$

for any $\gamma \in [0, 1]$. DDLite also uses stochastic gradient descent to solve this logistic regression problem, and the penalty parameter ρ can be chosen automatically using a validation set. Given a constant number of labeling functions and a sufficiently large training set, we can bound the expected parameter estimation error for $\hat{\alpha}$ and $\hat{\beta}$ to arbitrarily small values, and the expected generalization risk of $l(\hat{w})$ arbitrarily close to the algorithm's generalization risk. Further details can be found in [7].