Data Programming by Demonstration: A Framework for Interactively Learning Labeling Functions

Paper ID: 1385
Emails
Institutions

ABSTRACT

Data programming was introduced to reduce the cost of labeled data collection through denoising labeling functions written (manually) by experts with domain knowledge. Writing labeling functions manually requires, however, both programming literacy and domain expertise. Transferring domain expertise into labeling functions by enumerating rules and associated thresholds is not only time consuming but also inherently difficult for nontrivial labeling tasks, further inhibiting broader adoption of data programming. In this paper we propose a new framework, data programming by demonstration (DPBD), to interactively learn labeling functions. DPBD aims to relieve the burden of writing labeling functions from users, enabling them to focus on higher-level semantics such as identifying relevant signals for labeling tasks. We operationalize our framework with Ruler, an interactive system for text labeling, and gather qualitative feedback through a user study with six data scientists in creating labeling functions for sentiment and spam classification tasks. Results show that Ruler enables easier and faster creation of labeling functions without performance loss.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Knowledge representation and reasoning**; • **Information systems** → **Data mining**.

KEYWORDS

Data programming, weak supervision, labeling, machine learning, deep learning, interactive systems, programming by demonstration, program synthesis

ACM Reference Format:

Paper ID: 1385. 2020. Data Programming by Demonstration: A Framework for Interactively Learning Labeling Functions. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 22–27, 2020, San Diego, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330993

1 INTRODUCTION

The success of machine learning in general and deep learning in particular has dramatically increased the demand for applying machine learning to solve problems across domains. Machine learning models used in practice today are predominantly supervised models and rely on large datasets labeled by domain experts for training. However, labeled training datasets are expensive to acquire and maintain [36]. The cost of labeled training data remains a bottleneck for training supervised models, particularly high-capacity models such as deep neural networks, and limits broader utilization of these models outside resource rich settings.

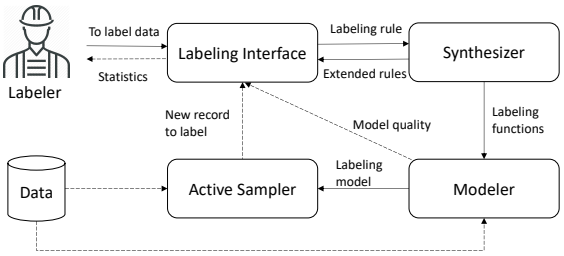

Figure 1: Overview of the data programming by demonstration framework. Straight lines indicate the flow of domain knowledge, and dashed lines indicate the flow of data.

Weak supervision methods such as crowdsourcing [17], distant supervision [30], and user-defined heuristics [12] enable the use of noisy or imprecise sources to gather large training data for supervised learning. Data programming [8, 34, 35] aims to address the difficulty of collecting labeled data by using a programmatic approach to weak supervision by heuristics, where domain experts are expected to provide functions incorporating their domain knowledge to label subsets of a large training dataset. Since these labeling functions may overlap or conflict with each other, they are denoised (i.e., the optimum corresponding weights are learned) using inference over a generative graphical model. The denoised functions are then applied to the large unlabeled dataset to obtain probabilistic labels and train standard machine learning models in a noise-aware manner.

Prior work on data programming focuses on modeling and aggregating simple labeling functions written manually [34, 35] or generated automatically [15, 39] to denoise labeling functions. However, little is known about user experience in writing labeling functions and how to improve it [4]. Writing data programs or labeling functions can be challenging. Most domain experts or lay users do not have programming literacy. Crucially, it is often difficult to convert domain knowledge to a set of rules through enumeration even for those who are proficient programmers. The accessibility of writing labeling functions is a challenge in a wider use of data programming.

To address this challenge, we draw from two lines of prior research. The first is programming by demonstration (PBD) or example (PBE), e.g., [11, 26], which aims to make programming easier by synthesizing programs based on user interactions or input and output

KDD '20, August 22–27, 2020, San Diego, CA, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330993

examples. The second is interactive learning from user-provided features or rationales [42, 43]. Building on this earlier work in part, we introduce a new framework, data programming by demonstration (DPBD), to make creating labeling functions easier by learning them through user interactions. DPBD aims to move the burden of writing labeling functions to an intelligent synthesizer while enabling users to steer the synthesis process at multiple semantic levels, from providing rationales relevant for their labeling choices to interactively filtering the proposed functions.

We operationalize the DPBD framework with Ruler, an interactive system for generating labeling functions to create training data for text classification models. We use Ruler to compare our framework with two alternative methods: manually writing labeling functions [34] and natural language explanation [15]. To this end, we conduct a within-subjects study with six participants and collect qualitative feedback in performing two labeling tasks on two different corpora: YouTube Comments [5] and Amazon Reviews [16]. Results suggest that the DPBD framework enables easier and faster generation of labeling functions for text than possible with the state-of-the-art approaches. We highlight the main contributions of this paper as follows:

- We propose a general data independent DPBD framework for interactively learning labeling rules (Section 3).
- We design an interactive system Ruler to demonstrate how our framework can be applied on text dataset (Section 4).
- We conduct an in-depth user study to empirically confirm the accessibility and expressiveness of Ruler and collect feedback on Ruler as well as alternative approaches (Section 5 and Section 6).

All our research artifacts, including the Ruler code and demo, are available at https://github.com/ruleauthors/ruler.

2 RELATED WORK

Our work builds on earlier work of weak supervision (Section 2.1), programming by demonstration (Section 2.2), and learning from feature annotations provided by users (Section 2.3).

2.1 Weak Supervision and Data Programming

Curating training data is one of the major rate-limiting steps in supervised machine learning. Prior ML research introduces approaches based on semi-supervised learning, weak supervision, and transfer learning to alleviate this problem. Methods based on weak supervision leverage noisy, limited, or low precision sources such as crowdsourcing [17], distant supervision [30], and user-defined heuristics [12] to gather large training data for supervised learning. Data programming [34, 35] is a programmatic approach to weak supervision by heuristics, where functions provided by domain experts to label subsets of a training dataset are used to create training data at scale and train ML models using probabilistic labels. Recognizing the difficulties around writing labeling functions, BabbleLabble [15] enables users to provide a natural language explanation for each labeling decision and converts these explanations into programmatic labeling functions. DPBD aims to make data programming easier by improving the accessibility and effectiveness of the process of writing labeling functions. In our comparison (Section 5), our DPBD system Ruler significantly outperforms existing labeling function creation methods in ease of use without sacrificing labeling qualities.

2.2 Program Synthesis by Demonstration

Automated synthesis of programs that satisfy a given specification is a classical artificial intelligence (AI) problem [41]. Generating programs by examples or demonstration is an instance of this problem and has been an interest of researchers in multiple fields, including robotics, machine learning, programming languages, databases, and human-computer interaction. The terms programming by example (PBE), or programming by demonstration (PBD) have often been used interchangeably, though their adoption and exact meaning might diverge across fields and applications.

There is a rich research literature of PBD systems, which generate programs satisfying given input-output examples, have been applied to automate various data analysis tasks [11]. PBD systems aim to empower the end user programming in order to improve user productivity [6, 20, 22, 27, 31, 32]. One of the core research questions is PBD is how to generalize from seen examples or demonstrations. To generalize, PBD systems need to resolve the semantic meaning of user action on relevant (e.g., data) items. Prior approaches incorporate a spectrum of user involvement, from making no inference (e.g., [13, 32]) to using AI models with no or minimal user involvement, to synthesize a generalized program (e.g., [11, 19, 23, 28, 29]). Our proposed framework is a hybrid approach within the spectrum above and combines statistical ranking and evaluation along with interactive demonstration.

2.3 Learning from Feature Annotations

Prior work proposes methods for learning from user provided features [10, 25, 33], rationales [7, 40, 42, 43], and natural language explanations [15, 38]. BabbleLabble [15] uses a rule-based parser to turn natural language explanations into labeling functions and aggregates these functions using Snorkel. DPBD can also be seen as a framework for learning labeling functions by enabling users to provide high level imprecise rationales through a visual interaction language. Our framework is inspired in part by the work on learning from user rationales, where users express their labeling rationales by marking parts of data examples (text span, image region, etc.) in context. We significantly extend this earlier work through active learning [37] and programming by demonstration with visual interaction in order to improve the accessibility of data programming.

3 FRAMEWORK

Problem Statement Given a dataset D as a set of data records $\{t_1,...,t_n\}$, and a set of labels $L = \{l_1,...,l_m\}$. We aim to design a framework that allows the data labeler to visually provide noisy explanation e_i on assigning $t_p \in D' \subset D$ ($|D'| \ll |D|$) with the most meaningful label $l_q \in L$, and based on the set of explanations $E = \{e_1,...,e_i,...,e_k\}$, we want to build a labeling classifier $M_{D',E}$, which automatically labels $t \in D \setminus D'$ with one label $l \in L$, i.e., $M : D \rightarrow L$.

Framework Overview Figure 1 shows an overview of the data programming by demonstration (DPBD) framework. The framework has two input sources: the human *labeler*, and the *data* that is to be labeled. The labeler is the subject matter expert who has domain knowledge on understanding and extracting useful signals from

data, and assumes no background about writing computer programs. The data, which can be in the type of text, image or audio, contains a set of data records. Each data record is expected to be assigned with one label from a finite set of labels. Given a dataset, our framework enables the labeler to label each record with a categorical label, while providing her labeling rationales by interactively marking relevant parts of the record and specifying semantics and relationships among them. The output of our framework is a labeling model, which is trained to automatically produce labels for the large set of unlabeled data. Inherited from the traditional data programming [34], our framework also assumes that a set of labeled data is available for tuning model hyperparameters.

The DPBD framework consists of four components. First, the labeler interacts with data via the *labeling interface* (Section 3.1). The labeling interface records the labeler's interaction and compiles the interaction into a labeling rule. Second, the *synthesizer* synthesizes more labeling rules and translates labeler's accepted rules to runnable labeling functions (Section 3.2). Third, the labeling functions are passed over to the *modeler*, which builds a labeling model by holistically leveraging all labeling functions (Section 3.3). The fourth component is the *active sampler*, which samples a new data record each time for the human labeler to label in the next round (Section 3.4). The process continues until a certain stopping condition is met (e.g., reaching a desired model quality) or the labeler decides to exit. In the rest of this section, we describe the details of each component.

3.1 Labeling Interface

The labeling interface is the workplace where the labeler encodes the domain knowledge into labeling rules. It provides a way to express noisy explanations for labeling decisions using a visual interaction language. The primary goals of the the labeling interface are twofold: 1) it defines a general labeling model on how interaction is specified and compiled; and 2) it provides a set of user-friendly operations and dashboard that enable the labeler to express the domain knowledge for labeling.

3.1.1 Generalized Labeling Model. Our framework defines a generalized labeling model (GLM) to abstract the common practices in labeling processes. Inspired by the entity-relationship model [9] in database modeling, GLM models the data records by *concept* and *relationship*. Given one data record whose format depending on the task can be text, image or audio, GLM views the data record as a series of tokens. A token is a continuous subset of a record with no semantics attached. For example, on text data, a token can be any span (single char to multiple words) of the data record; on an image data record, it would be a 2-dimensional region, rectangular or free form; and on an audio data record, it would be a 1-dimensional window of the data record (e.g., a phoneme). A concept is a group of tokens that the labeler believes share common semantics. For instance, the labeler might define a concept of positive adjective consisting of a set of tokens, each of which can imply a positive review. In the task of labeling image data, the labeler may create an animal concept to generalize the dog and cat objects. When labeling audio data, the labeler can create a concept to aggregate all phonemes related to happiness. This abstraction allows the user to capture specific examples as well as to generalize.

The other important aspect that GLM models is the relationship. A relationship represents a binary correlation between token-token, token-concept, or concept-concept. Example correlations are membership (e.g., a token is in a concept), co-existence (e.g., opinion and aspect tokens [24]), and positional (e.g., a person is standing left to a table [14]).

Table 1: Mapping from GLM elements to labeling operations.

Element in GLM	Operations in Labeling Interface
token	select, assign_concept
concept	create, add, delete
relationship	link, direct_to

3.1.2 Mapping GLM elements to operations. Given the GLM specification described above, our framework also defines the operations that can be applied on GLM elements. Table 1 lists the GLM elements and the corresponding operations. The labeler can select a token, and assign to a concept as a member. For concept, the labeler can create a new concept, add tokens to it, or delete a concept. For relationship, two tokens can be linked together to represent co-occurrence; also, one originating token can direct to a receiving token to denote the relative positional arrangement.

The implementation of the labeling interface and operations would vary across data types. Depending on data types, the data record is presented based on its most natural way. For example, it is natural to present the content of an audio record in sound wave, rather than in string format for the text record. Accordingly, different operations can be implemented. For instance, the token select operation on text data can be achieved by highlighting the token, while on image data it can be implemented via a rectangular selection.

Note that, compared to the traditional data programming by writing programs, which gives the labeler more flexibility and expressiveness, our labeling interface only supports a finite set of operations. In the data programming by demonstration framework, we tradeoff flexibility for accessibility. As we will show in our user study (Section 5), the labeling interface is expressive enough to complete labeling tasks while ensure reasonable labeling quality. Also, the set of operations in Table 1 can cover all operations that most labelers will write.

3.1.3 Compiling operations into a labeling rule. We asks the labeler to express one labeling knowledge on one label per data record. Once the labeler finishes labeling using the operations, the labeling interface compiles all operations into exact one labeling rule (with one label). A labeling rule serves as an intermediate language that is interpretable by both the labeler and the subsequent synthesizer. The synthesizer will later extend more labeling rules based on the rule that is directly compiled from labler's operations, and present the extended labeling rules for the labeler to verify. The labeler should be able to interpret the extended labeling rules naturally and choose desired ones based on the domain knowledge. In our framework, we adapt the notation of domain relational calculus [18] to represent a labeling rule.

A labeling rule compiled from labeler's operations can be expressed using the following template: {`tokens` | `conditions`} ⇒ `label`. The `tokens` is a sequence of tokens with existential quantification; the `conditions` is a conjunctive formula over boolean predicate

that makes testing using `tokens` on a data record. A predicate is a first-order expression, which can be expressed as a tuple (T,*lhs*,*op*,*rhs*). T is an optional transformation function on a token identifier. Transformation is a process of mapping the raw token to other forms. Example transformations are to lemmatize a word in text labeling, to apply a speech-to-text detection in audio labeling, to recognize an object in image labeling, and to get the context (e.g., relative location of a token). *lhs* is a token, while *rhs* is can be either token, literal or a set. In case that *rhs* denotes a token, the transformation function T may also apply to *rhs*. *op* is an operator. Depending on *rhs*, *op* is in the set of $\{>,\geq,<,\leq,=,\neq\}$ if *rhs* is a token or literal, or in its set operators of $\{>\text{ANY}(),\geq\text{ANY}(),<\text{ANY}(),\leq\text{ANY}(),\in,\notin\}$ if *rhs* is a set. Since the `conditions` is in the conjunctive form, the order of labeler's interactions does not matter. The conjunctive formula is motivated by the OccamâĂŹs razor principle. This principle suggests that for all possible hypotheses to label a data record, the simple conjunctive form which combines only necessary predicates is always preferred.

A labeling rule described in above format can be straightforwardly interpreted by the labeler, as well as translated into labeling functions by the synthesizer as that, once the `conditions` are evaluated to be `True` on a data record, then it assigns the data record with the `label`. Otherwise, the labeling rule is not activated and does not label on this data record.

Example: Consider labeling binary (positive or negative) sentiment on Amazon review data [16]. Given a text review:

`This book was so great! I loved and read it so many times that I will soon have to buy a new copy.`

Assume the labeler thinks of this data records as a `positive` sentiment. She can narrow down to *program* her knowledge using GLM. First, the labeler may select two tokens that are related to the sentiment: `book` and `great`. Consider there are two concepts the labeler previously created : 1) `item`= {`book, electronics`}; and 2) `padj`= {`wonderful`}.

Then, the labeler realizes the token `great` can be generalized by the `padj` concept, which means that the labeling rule will still be valid if this token is replaced by any tokens in the concept. Hence, she adds this token to concept, and assigns the token to `padj` to indicate that it can be generalized. Finally, the labeler creates a positional relationship from `book` to token `great` to indicate `book` appears before `great`, before completing the labeling process.

The labeling interface compiles above operations into a labeling rule $r: \{t_1,t_2 \mid t_1 = \texttt{book} \wedge t_2 \in \texttt{padj} \wedge idx(t_1) < idx(t_2)\} \Rightarrow \texttt{positive}$ □

Once the labeler completes the labeling process, the labeling rule is sent to the synthesizer for expansion and program synthesis.

3.2 Synthesizer

Given the compiled labeling rule from the labeling interface, the synthesizer does the following tasks: 1) it extends one single labeling rule from labeler's interaction to a set of more general labeling rules; and 2) it translates the labeling rules into computer programs.

As for the second goal, it is straightforward to translate the rules into runnable computer programs (labeling functions). In this section, we focus on how to synthesize the extended labeling rules.

Given the labeling rule compiled directly from labeler's interaction, the synthesizer aims to generate more labeling rules, with two competing goals: 1) it aims to maximize generalization, so that more data can be labeled; and 2) it also aims to maximize the coverage of labeler's interaction, simply because labeler's interaction is the most valuable signal for labeling from domain knowledge. We assume that there is little redundancy in labeler's interaction, and hence covering all labeler's interaction is more important to express the labeler's knowledge and as a result, more accurate for the labeling task. Therefore, the synthesizer only searches the constrained space in which the labeling rules can cover all labler's interaction.

In our framework, we design a heuristics based search algorithm to generate more labeling rules. The generalization in general can be achieved using the following heuristics: 1) substituting tokens with concepts; 2) replacing positional relationships with co-existence ones; and 3) flipping over binary labels.

Algorithm 1 illustrates the process of extending labeling rules. Since the labeling rule in GLM has conjunctive conditions, Algorithm 1 (Line 3-17) generalizes each predicate in the conditions. Inside, Line 6 to Line 10 substitute token with concept. Line 6 can be implemented explicitly by matching token to concept set, as well as sophisticated data-dependent processing via transformation . For example, in our system for text labeling (Section 4), in addition to matching values with labeler defined concepts, we also apply named-entity recognition (NER) where the named-entities are implicit concepts that a token can be a member of. Line 12 to Line 14 replace the positional with co-occurrence relationship by removing the condition that specifies the positional context. The conditions for extended labeling rules is a conjunctive combination of single predicates, one from each extended condition set (Line 19). In addition, for special case of binary labeling, the algorithm also considers the rule which flips over the label by adding negation to the conditions (Line 22).

Once the extended rules are generated, Line 24 ranks the rules by their generalization score. The generalization score measures how applicable a certain labeling rule applies. We define a data-independent generalization score for a labeling rule r as:

$$G(r) = \prod_{c \in r.conditions} |c.rhs|$$

Intuitively, $G(r)$ is calculated by counting how many different data instances that r can be used. It prefers labeling rules using large sets to match tokens in the data record.

Example: Continue our Amazon review example, the synthesizer can derive multiple labeling rules from r using Algorithm 1. The extended labeling rules are:

(1) $\{t_1,t_2 \mid t_1 \in \texttt{item} \wedge t_2 \in \texttt{padj}\} \Rightarrow \texttt{positive}$
(2) $\{t_1,t_2,t_3 \mid t_1 \in \texttt{item} \wedge t_2 \in \texttt{padj} \wedge t_3 \in \texttt{neg}\} \Rightarrow \texttt{negative}$
(3) $\{t_1,t_2 \mid t_1 \in \texttt{item} \wedge t_2 \in \texttt{padj} \wedge idx(t_1) < idx(t_2)\} \Rightarrow \texttt{positive}$
(4) $\{t_1,t_2,t_3 \mid t_1 \in \texttt{item} \wedge t_2 \in \texttt{padj} \wedge idx(t_1) < idx(t_2) \wedge t_3 \in \texttt{neg}\} \Rightarrow \texttt{negative}$
(5) $\{t_1,t_2 \mid t_1 = \texttt{book} \wedge t_2 \in \texttt{padj}\} \Rightarrow \texttt{positive}$
(6) $\{t_1,t_2,t_3 \mid t_1 = \texttt{book} \wedge t_2 \in \texttt{padj} \wedge t_3 \in \texttt{neg}\} \Rightarrow \texttt{negative}$

Labeling rule (1) is generated using heuristics (1) and (2). Labeling rule (3) and (5) are synthesized by using heuristics (1) and (2), respectively. Labeling rule (1) is more general than (3) and (5) because all data records that can be labeled by (3) and (5) will be labeled the same way using labeling rule (1). Labeling rules (2, 4, 6) are due to flipping over the binary label with heuristics (3). □

Algorithm 1 Extend labeling rules

Require: r ▷ The labeling rule from interaction
procedure EXTEND(r)
  $rules, cond \leftarrow \emptyset$
  for all $(T, lhs, op, rhs) \in r.conditions$ **do**
    $c \leftarrow \{(T, lhs, op, rhs)\}$
    if rhs is a literal **then**
      find R s.t. $rhs \in R$
      if $R \neq$ NULL **then**
        $\hat{op} \leftarrow$ corresponding operator of op for set
        add $(T, t, \hat{op}, R)$ to c ▷ heuristic 1
      end if
    else if rhs is a token identifier **then**
      if T transforms to positional context **then**
        add NULL to c ▷ heuristic 2
      end if
    end if
    add c to to $cond$
  end for
  for all $cnf \leftarrow$ conjunctive combinations of $cond$ **do**
    add $(\exists r.tokens \mid cnf \mid r.label)$ to $rules$
  end for
  if $|labels| == 2$ **then** ▷ Binary labels, heuristic 3
    add $(\exists r.tokens, t \mid cnf \wedge t \in \mathsf{neg} \mid (r.label)^c)$ to $rules$
  end if
  rank $rules$ by the generalization score
  return $rules$
end procedure

Once the extended labeling rules are generated, the labeler can help confirm the validity in order to achieve faster convergence. The top-k candidates ranked by the generalization score can be displayed in the labeling interface for the labeler to accept or reject.

3.3 Modeler

The goal of the modeler component is to train a model that can be used to automatically annotate unlabeled dataset. Naïvly aggregating the labeling functions can be either inaccurate (since labeling functions can be conflicting and correlated), or does not scale with large set of unlabeled data [34]. To achieve this goal, the modeler encapsulates the ideas from traditional data programming to first build a generative model to denoise labeling functions, and then train a discriminative model to leverage other features beyond what are expressed by the labeling functions. For the purpose of description completeness, we summarize the core ideas from traditional data programming [8, 34, 35]

3.3.1 Generative model. Given the labeling functions, the modeler starts with building a generative model to weight each of the labeling functions to maximize the consistency with the true labels in the the training data. The generative model is constructed as a factor graph. Consider there are m data records X whose true labels are Y, in total n labeling functions F synthesized by the synthesizer, and the set of all labels L. We can construct a $m \times n$ matrix M, where each coordinates (i, j) denotes the label $F_j(X_i)$ (i.e., $M(i,j) \in L \cup \emptyset$). The factor graph encodes three types of factors:

(1) (Data coverage) $\phi_{i,j}(M,Y) = \mathbb{1}\{M(i,j) \neq \emptyset\}$
(2) (Label accuracy) $\phi_{i,j}(M,Y) = \mathbb{1}\{M(i,j) = Y_i\}$
(3) (Function correlation) $\phi_{i,j,k}(M,Y) = \mathbb{1}\{M(i,j) = M(i,k)\}$

Let w be a weight vector to denote the weight of the concatenated vector from above factors. Then the generative model is defined as:

$$p_w(M,Y) = \frac{1}{Z_w} \exp\left(\sum_{i=1}^{m} w^T \phi_i(M, Y_i)\right)$$

where Z_w is the partition function. In general, the true labels Y are unknown or unavailable during learning, so the objective is to minimize the negative log marginal likelihood:

$$\hat{w} = \operatorname*{argmin}_w -\log \sum_Y p_w(M,Y)$$

For X, we use the predications $\hat{Y} = p_{\hat{w}}(Y|M)$ as the probabilistic training labels.

3.3.2 Discriminative model. The next step is to use X and the probabilistic training labels $\hat{Y}$ to train a discriminative model h_θ, whose objective is to minimize the expected loss with respect to $\hat{Y}$:

$$\hat{\theta} = \operatorname*{argmin}_\theta \sum_{i=1}^{m} \mathbb{E}[Loss(h_\theta(x_i), \hat{Y}_i)]$$

h_θ is the final output labeling model. In our framework where the labeler iteratively creates more labeling functions, hence h_θ is iteratively built and evaluated.

3.4 Active Sampler

To improve the model quality with faster convergence, our framework has an active sampler to choose next data record for the labeler to interact in the next loop. Given the current learned labeling model, the active sampler adapts a uncertainty based sampling strategy [21]: it seeks a data sample x^* from the unlabeled data about which the labeling model is least certain to label. The uncertainty is measured by the entropy among the probabilities of all labels:

$$x^* = \operatorname*{argmax}_x -\sum_i^{|L|} p_\theta(L_i \mid x) \log p_\theta(L_i \mid x)$$

To reduce the cost of the active sampler, it first randomly samples a subset of unlabeled data (subsample), and then actively samples a record from the subsample with highest uncertainty based on above equation.

4 RULER

The framework we described in Section 3 is a data independent framework that can be in general applied to label various formats of data, such as text, image and audio data. To empirically show the effectiveness of the framework, and as the first step towards building a data labeling platform, we instantiate our framework and implement RULER, which is a web-based system of data programming by demonstration for text data.

RULER supports two human roles: the owner and the labeler. In addition to the labeler's interaction described in Section 3.1, in RULER, an owner creates a labeling task by uploading the data (in csv format) and specifying the metadata (e.g., labels, stopping condition). Then, the owner posts the task and invites one or more lablers to do the task.

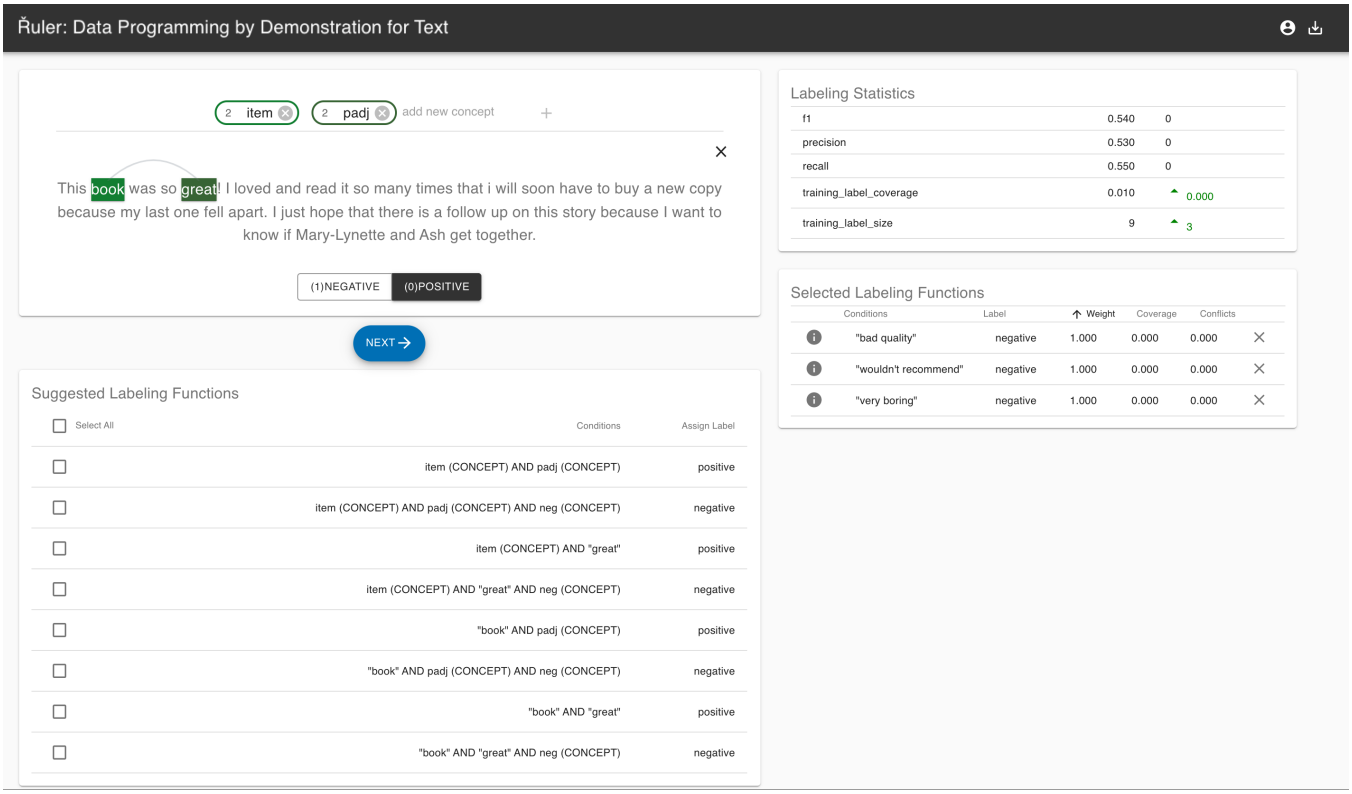

Figure 2: RULER user interface. RULER synthesizes labeling rules based on rationales expressed by users by interactively marking relevant parts of the example and specifying implied group or pairwise semantic relations among them.

From the owner's console, the owner can review labelers' progress or even consolidate the labeling rules from multiple labelers to train a multi-user labeling model. The generated labeling model and the automatically labeled data can be exported at the end of the task. Both the roles interact with RULER's frontend in the browser.

In the rest of this section, we are going to describe the implementation details from the labeler's view.

Frontend Figure 2 captures a snapshot of RULER's labeling interface implemented in React JavaScript library [1]. Recall in Section 3.1 that the purpose of the labeling interface is to enable the labeler to visually encodes domain knowledge into rules. In RULER, there are four main parts in the labeling interface: 1) Left-up is the labeling panel, where the labeler interacts with the data records and GLM elements. All the labeling operations in Table 1 are implemented in the context of text data. For example, the token select operation is achieved by highlighting the token span. When a token is selected, RULER pops up a menu for further operations, such as to create relationships with another token and to assign concept to generalize the token. The concept is created as a set, which can consist of string literals or regular expressions to match. The candidate labels are also shown in this panel, so that the labeler can easily choose one to label the text record in view. 2) Left-down part automatically shows the top 10 synthesized labeling rules from the synthesizer (Section 3.2) based on labeler's real-time operations. The list is ordered by the generalization score $G(r)$, and meaning is self explanatory. The labeler can select a subset from the list by excluding the wrong synthesized ones. Only the selected ones will be sent to the modeler to build models, once the labeler proceeds by clicking `next`. 3) Right-up shows a group of current statistics, including the quality of the final labeling model, and differential changes incurred by the last previous labeled data record. 4) Right-down lists the current selected labeling ruels included in building the labeling model, and their respective statistics in our modeler's backend implementation.

Backend RULER's backend is consisted of the components of the synthesizer (Section 3.2), the modeler (Section 3.3) and the active sampler (Section 3.4). The backend components are all implemented in the Python environment (version 3.6).

In addition to follow the explicit description in Section 3.2, RULER's synthesizer also augments the labeling rule, implicitly. It includes a natural language based pre-processing to lemmatize and standardize words, and enhanced concepts such as `person` and `location` from named entities using the spaCy library [3]. RULER encapsulates Snorkel [2], which is the state-of-the-art library for data programming and has build-in generative and discriminative models, into the modeler. When the labeler clicks `next` in the labeling interface, RULER's active sampler randomly chooses a subsample of 30 unlabled data records, and returns the one record from the subsample with the highest uncertainty score (Section 3.4).

5 EVALUATION

We evaluate our framework together with two alternative methods for creating labeling functions, manual [34] (Snorkel) and natural language based [15](BabbleLabble). Our primary goal is to better understand the trade offs afforded by each method based on the qualitative feedback by participants. To this end, we conducted a user study with six participants and measured their task performance accuracy in two labeling tasks on two different corpora, YouTube Comments [5] and Amazon Reviews [16]. In addition to task performance, we also analyzed the accessibility, expressivity, and interpretability of all the three methods using the qualitative feedback elicited from participants and our observations gathered during the study sessions.

Experimental Design We carried out our study using a between subjects design, where participants performed both tasks using only one of the three methods (conditions). For this, we divided the participant pool into three random groups each with two participants. We then randomly assigned conditions to groups, one for each. To avoid ordering effects, we counterbalanced the presentation of tasks within each group. The sole independent variable we controlled was the method of creating labeling function, which had three conditions; Snorkel, BabbleLabble, and Ruler. Note that these three tools correspond to three different forms of creating labeling functions, manual (or conventional programming), using natural language explanations, and using visual interactive demonstration, respectively.

Participants We recruited six participants (all male) from our lab. Participants had either "research scientist" or "software engineer" as a job title. Five of them held PhDs and one held BS, all in computer science. All participants had significant programming experience (avg=16.5, std=9.2). Their experience with Python programming ranged from 1 to 8 years with an average of 4.6 years (std=2.5).

Tasks and Data We asked participants to write labeling functions for two prevalent labeling tasks, spam detection and sentiment classification. They performed these two tasks on YouTube Comments and Amazon Reviews, respectively. We asked participants to write as many functions as they considered necessary for the goal of the task. They were given 30 mins to complete each task. Participants were also tutored for 15 mins on writing labeling functions using a topic classification (electronics vs. guns) task on a newsgroup dataset.

Procedure Participants started their session by signing a consent form and completing a questionnaire that elicited information on their educational background and programming and model development experiences. Next they went through a tutorial on writing labeling functions using the tool or method slotted for their group. After the tutorial, participants proceeded to perform the two tasks in the order assigned to them. The tutorial lasted 15 mins and participants were given 30 mins for each task. Following the tutorial or between the tasks, participants were encouraged to take a break. We recorded the labeling functions created by participants and these functions' individual and aggregate performances on each task. At the end of the session, participants completed an exit survey to provide their qualitative feedback.

6 RESULTS

Expressivity and Interpretability In the exit surveys we inquired participants opinions about ease of use, expressivity, and ease of learning along with overall satisfaction with the tool (see Figure 3).

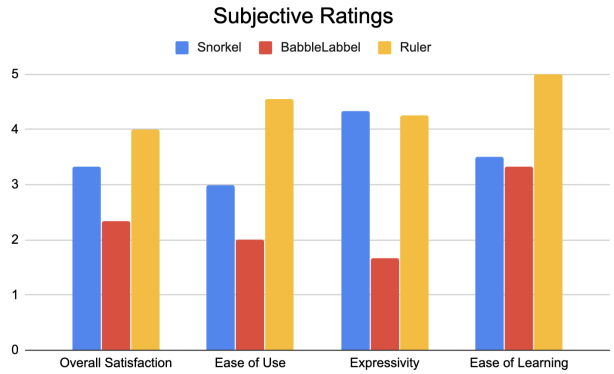

Figure 3: Participants' subjective ratings on ease of use, expressivity, ease of learning and overall satisfaction.

While we have limited data, with two participants in each condition, participants rated Ruler highest in all subjective questions except expressivity by a small margin. Largest margin is observed in ease of learning (1.5) and ease of use (1.25). on a 5-point Likert scale. Ruler is found to be almost as expressive as Snorkel by a small margin (0.08) and significantly more expressive than BabbleLabble (2.58).

Participants were also asked about opinions on 1) strategies used and problems observed, and 2) potential improvements to the systems examined in free-form text. Below we report results to these questions.

Strategies and Problems For the Snorkel condition, implementing anything beyond simple labeling functions was a major issue, including the difficulty to use libraries (P1) and incorporate complex logic, to such a degree that P2 felt 'split between producing labeling functions and implementing them correctly.' P2 suggested 'template functions' abstracting commonly-used labeling functions for specific data domains. P2 also expressed lack of a clear understanding of Snorkel that led him to be unsure as to 'how much logic to put into one labeling function vs. create simple functions and have Snorkel sort it out.' Due to the extent of the observe- code-analyze loop where each iteration took a long time, P2 felt that it was 'hard to see if [it was] improving or worsening.'

As for BabbleLabble, participants found it to be limited both in terms of what patterns could be expressed 'restricted to use patterns of word (co-)occurance' and how to express, as they 'tried to express it in a parsable sentence' (P3) and 'got multiple parsing errors, ..., and need[ed] to look into the tutorial to figure out what's the right syntax' (P3). While participants suggested many improvements after completing the study session such as ability to apply logical operations (e.g. and/or) to 'connect rules' (P4), use 'phrase-level explanations' (P4) during the study they were unsure about what could be expressed (P3). P3 and P4 did not find error analysis 'insightful enough' and had difficulty in attributing performance to individual rules, as P4 argued: '...on an individual basis, rules seemed to perform poorly (less than 50 percent), but if I removed the rules from the batch used for training, overall performance would drop.'

All the participants who used either Snorkel or BabbleLabble found going back and forth (scrolling up and down) between writing

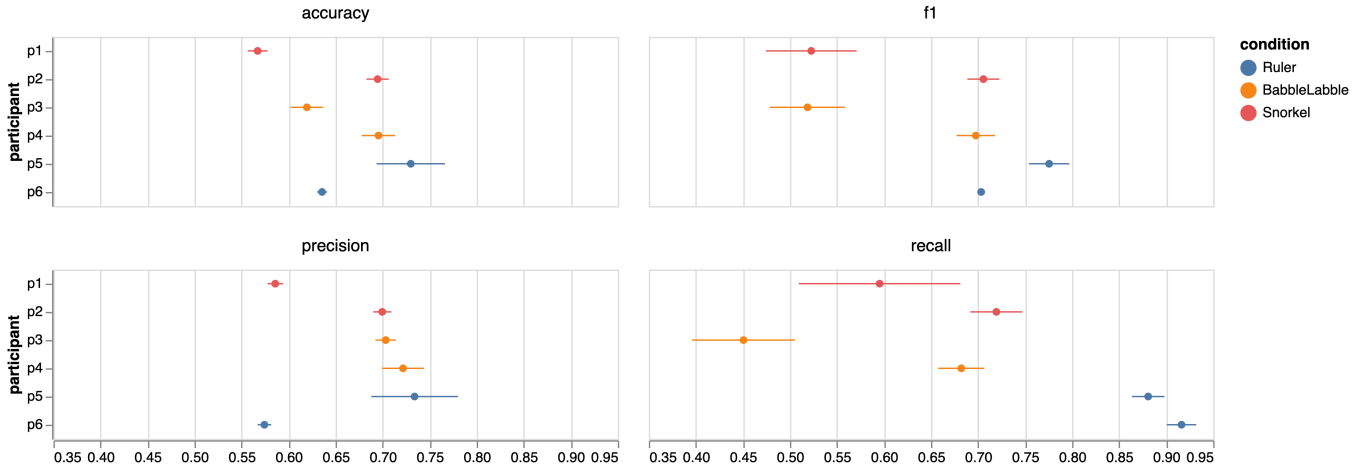

Figure 4: Performances of the classifier models trained on the probabilistic labels generated by participants' labeling models.

labeling functions and evaluating them in the linear interface of Jupyter Notebook cumbersome.

Overall, participants found Ruler easy to use and 'cognitively very simple' (P6). Features such as 'immediate visual feedback on how much a new rule increased/decreased the metrics' (P6) leading to shorter iterations 'encouraged [participants] not to be hesitant about trying out stuff'(P5) and 'create more labeling functions (and not be picky) and monitor how labeling functions interact and learn from those' (P5). Preliminary results suggest 4 that the ease and speed afforded by Ruler does not incur a cost on the end model performance.

Improvements For Snorkel, participants suggested more transparency into how the tool works, particularly in aspects of evaluation. Participant P1 suggested that 'aggregator function which combines the labeling functions is sort of a black box' and makes it 'not obvious why things don't improve when adding a new function'. Participant P2 suggested mainly suggested ways to improve speed of iterations such as 'templates', 'better search through documents', 'comparative analysis of labeling functions.'

Regarding BabbleLabble, participants' feedback on improvement focused on the 'capabilities of the parser' and 'dependence on parsing natural language'. Participants suggested improvements such as 'support for phrases with more than one token', 'fine-grained tokenization', 'editable aliases' (P4), 'support for synonyms', and better user experience that 'makes clear what explanation patterns are supported' (P3), 'support for regular expressions', 'transparent statistics for individual rules that reflect how they contribute to the system' and a 'simpler interface in general' (P4)

Finally for Ruler, participants offered many improvements to overall usability that would allow them to create rules faster such as warnings to the user 'you haven't checked any of the available rules' and 'visual feedback in the list of existing rules that shows you which of them improve/worsen' (P5). A particularly interesting suggestion was the ability to directly 'add a rule even if it does not apply to the current example' (P5). While this would seem to be a deviation from the demonstrational approach, it is still data-driven and would clearly improve the usability of the tool and allow increased coverage. Beyond that participants offered many ways to improve expressivity such as 'connecting multiple operand (beyond pairwise)' (P6), 'common regular expressions', expand examples to cover 'different tenses of verbs', for example (P6). Suggestions on offering multiple levels of surfaces forms (e.g. phrase, sentence, document) to apply to labeling functions is particularly intriguing to improve expressivity to create rules with conditions such as 'if two words appear in the same sentence', and 'if the review is short and has token X' (P5).

7 DISCUSSION

Accessibility is a key to wider adoption of any framework and machine learning (ML) frameworks are no exception. In this paper, we introduced a new framework, data programming by demonstration (DPBD) to ease writing labeling functions, improving the accessibility and efficiency of data programming. To show the viability of our framework, we also presented Ruler, a practical DPBD system for generating labeling functions to create training data for text classification models. Finally, through a user study with six data scientist performing real world labeling tasks for classification, we evaluated our framework together with the state-of-the-art methods.

While we were expecting Ruler to lead in terms of ease of use and learning, equalling expressivity of a rich programming language with many constructs while being perceived as much richer than natural language came as a bit of surprise. Initial results suggest that providing a simpler grammar with few constructs could achieve good results when considered in aggregates where learning of respective weights is left up to machine learning. As such users attention can be more on choosing the right generalization of observed instances, not on the implementation details in a programming language nor on how to express it in an intrinsically vague natural language with potentially invisible rules of interpretation. In the meantime, DPBD provides an effective basis for writing labeling functions by interactive demonstration. To facilitate future research and applications, we make Ruler available as an open source software at https://github.com/rulerauthors/ruler.

REFERENCES

[1] React: A javascript library for building user interfaces. https://reactjs.org/.
[2] Snorkel. https://www.snorkel.org/.
[3] spaCy: Industrial-strength natural language processing. https://spacy.io/.
[4] Interactive programmatic labeling for weak supervision. In *KDD DCCL Workshop*, 2019.
[5] T. C. Alberto, J. V. Lochter, and T. A. Almeida. Tubespam: Comment spam filtering on youtube. In *ICMLA*, pages 138–143. IEEE, 2015.
[6] J. Allen, N. Chambers, G. Ferguson, L. Galescu, H. Jung, M. Swift, and W. Taysom. Plow: A collaborative task learning agent. In *AAAI*, volume 7, pages 1514–1519, 2007.
[7] S. Arora and E. Nyberg. Interactive annotation learning with indirect feature voting. In *Procs. NAACL-HLT Student Research Workshop and Doctoral Consortium*, 2009.
[8] S. H. Bach, B. He, A. Ratner, and C. Ré. Learning the structure of generative models without labeled data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 273–282. JMLR. org, 2017.
[9] P. P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.
[10] G. Druck, B. Settles, and A. McCallum. Active learning by labeling features. In *Proc. EMNLP*, 2009.
[11] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, 2011.
[12] S. Gupta and C. Manning. Improved pattern learning for bootstrapped entity extraction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 98–108, 2014.
[13] D. C. Halbert. Watch what i do. chapter SmallStar: Programming by Demonstration in the Desktop Metaphor, pages 103–123. MIT Press, Cambridge, MA, USA, 1993.
[14] M. Haldekar, A. Ganesan, and T. Oates. Identifying spatial relations in images using convolutional neural networks. In *IJCNN*, pages 3593–3600, 2017.
[15] B. Hancock, M. Bringmann, P. Varma, P. Liang, S. Wang, and C. Ré. Training classifiers with natural language explanations. In *Proceedings of the conference. Association for Computational Linguistics. Meeting*, volume 2018, page 1884. NIH Public Access, 2018.
[16] R. He and J. J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*, pages 507–517, 2016.
[17] 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.
[18] M. Lacroix and A. Pirotte. Domain-oriented relational languages. In *Proceedings of the Third International Conference on Very Large Data Bases, October 6-8, 1977, Tokyo, Japan*, pages 370–378, 1977.
[19] T. Lau, S. A. Wolfman, P. Domingos, and D. S. Weld. Programming by demonstration using version space algebra. *Machine Learning*, 53(1-2):111–156, 2003.
[20] G. Leshed, E. M. Haber, T. Matthews, and T. Lau. Coscripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1719–1728. ACM, 2008.
[21] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *SIGIRâĂŹ94*, pages 3–12. Springer, 1994.
[22] T. J.-J. Li, A. Azaria, and B. A. Myers. Sugilite: creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 6038–6049. ACM, 2017.
[23] T. J.-J. Li and O. Riva. Kite: Building conversational bots from mobile apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 96–109. ACM, 2018.
[24] Y. Li, A. Feng, J. Li, S. Mumick, A. Y. Halevy, V. Li, and W. Tan. Subjective databases. *PVLDB*, 12(11):1330–1343, 2019.
[25] P. Liang, M. I. Jordan, and D. Klein. Learning from measurements in exponential families. In *Proc. ICML*, 2009.
[26] H. Lieberman. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann Publishers, 2001.
[27] G. Little, T. A. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan. Koala: capture, share, automate, personalize business processes on the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 943–946. ACM, 2007.
[28] R. G. McDaniel and B. A. Myers. Getting more out of programming-by-demonstration. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 442–449. ACM, 1999.
[29] A. Menon, O. Tamuz, S. Gulwani, B. Lampson, and A. Kalai. A machine learning framework for programming by example. In *International Conference on Machine Learning*, pages 187–195, 2013.
[30] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.
[31] B. A. Myers. Watch what i do. chapter Peridot: Creating User Interfaces by Demonstration, pages 125–153. MIT Press, 1993.
[32] B. A. Myers. Scripting graphical applications by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 534–541. ACM Press/Addison-Wesley Publishing Co., 1998.
[33] H. Raghavan, O. Madani, and R. Jones. Interactive feature selection. In *Proc. IJCAI*, 2005.
[34] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017.
[35] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *Advances in neural information processing systems*, pages 3567–3575, 2016.
[36] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*, pages 2503–2511, 2015.
[37] B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.
[38] S. Srivastava, I. Labutov, and T. Mitchell. Joint concept learning and semantic parsing from natural language explanations. In *Procs. EMNLP*, 2017.
[39] P. Varma and C. Ré. Snuba: automating weak supervision to label training data. *Proceedings of the VLDB Endowment*, 12(3):223–236, 2018.
[40] L. Von Ahn, R. Liu, and M. Blum. Peekaboom: a game for locating objects in images. In *Proc. CHI*, 2006.
[41] R. J. Waldinger and R. C. T. Lee. Prow: A step toward automatic program writing. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 241–252, 1969.
[42] O. Zaidan and J. Eisner. Modeling annotators: A generative approach to learning from annotator rationales. In *Proc. EMNLP*, 2008.
[43] O. Zaidan, J. Eisner, and C. Piatko. Using "annotator rationales" to improve machine learning for text categorization. In *Procs. NAACL-HLT*, 2007.