Ruler: Data Programming by Demonstration for Document Labeling

Sara Evensen
Megagon Labs
Mountain View, CA, USA
sara@megagon.ai

Chang Ge
University of Waterloo
Waterloo, ON, Canada
c4ge@uwaterloo.ca

Çağatay Demiralp
Megagon Labs
Mountain View, CA, USA
cagatay@megagon.ai

Abstract

Data programming aims to reduce the cost of curating training data by encoding domain knowledge as labeling functions over source data. As such it not only requires domain expertise but also programming experience, a skill that many subject matter experts lack. Additionally, generating functions by enumerating rules is not only time consuming but also inherently difficult, even for people with programming experience. In this paper we introduce RULER, an interactive system that synthesizes labeling rules using span-level interactive demonstrations over document examples. RULER is a first-of-a-kind implementation of data programming by demonstration (DPBD). This new framework aims to relieve users from the burden of writing labeling functions, enabling them to focus on higher-level semantic analysis, such as identifying relevant signals for the labeling task. We compare RULER with conventional data programming through a user study conducted with 10 data scientists who were asked to create labeling functions for sentiment and spam classification tasks. Results show RULER is easier to learn and to use, and that it offers higher overall user-satisfaction while providing model performances comparable to those achieved by conventional data programming.

1 Introduction

Machine learning (ML) models used today are predominantly supervised and rely on large datasets labeled for training. However, the cost of collecting and maintaining labeled training data remains a bottleneck for training high-capacity supervised models [33].

Weak supervision methods such as crowdsourcing [15], distant supervision [26], and user-defined heuristics [10] enable the use of noisy or imprecise sources to gather large training datasets. Data programming [6, 30, 31] 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 data programs (labeling functions) incorporating their domain knowledge. Prior work on data programming focuses on modeling and aggregating labeling functions written manually [30, 31] or generated automatically [13, 35] to create training data. However, little is known about user experience in writing labeling functions and how to improve it [2]. Many domain experts or lay users have little or no programming literacy. Even for proficient programmers, it is often difficult to convert domain knowledge to a set of rules by writing programs.

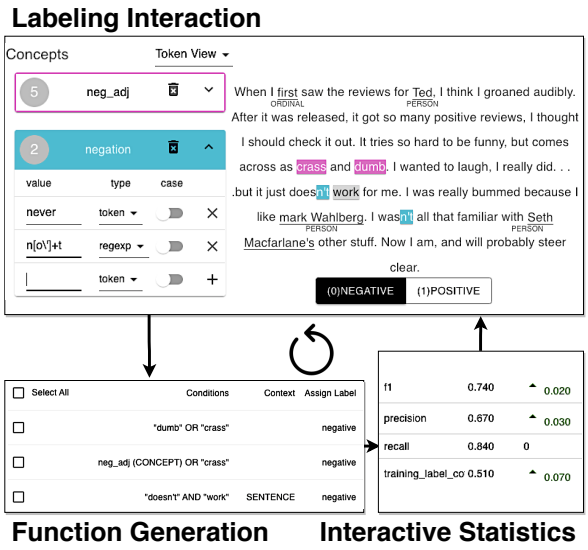

Figure 1: RULER enables the user to interactively generate a diverse set of labeling functions through simple, non-programmatic text annotations. Dynamically updated statistics allow the user to quickly test and evaluate ideas.

We introduce RULER (Figure 1), an interactive system that enables more accessible data programming to create labeled training datasets for

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 1996–2005

document classification models. RULER automatically generates labeling rules from user's labeling rationales or intents as demonstrated by span-level annotations and their relations provided by the user for specific examples. RULER bears some similarities to rule-based information extraction systems, that have been made accessible to domain experts by prior work such as PropMiner [3]. RULER, however, has a focus on creating training data, rather than a final model. This distinction is critical because while such information extraction systems focus on high-accuracy rules, RULER intentionally guides the user towards noisy heuristics in order to intelligently combine them, making the resulting data useful for training a supervised model that can better handle diverse inputs such as ungrammatical text. Additionally, RULER leverages features to classify the documents themselves, rather than for information extraction.

We also introduce DPBD, a new human-in-the-loop framework that moves 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. RULER is based on this framework, demonstrating how these concepts can apply to text documents. The DPBD framework builds primarily on two lines of research: the first is programming by demonstration (PBD) or example (PBE), e.g., [9, 22], which aims to make programming easier by synthesizing programs based on user interactions or input and output examples. The second is interactive learning from user-provided features or rationales [38, 39].

Through a user study conducted with 10 data scientists, we evaluate RULER alongside manual data programming using Snorkel [30]. We measure the predictive performances of models created by participants for sentiment classification and spam detection. We also elicit ratings and qualitative feedback from participants on multiple measures, including ease of use, ease of learning, expressivity, and overall satisfaction. We find RULER better facilitates the creation of labeling functions without any loss in the quality of learned labeling models.

Our main contributions include (1) DPBD, a general data-independent framework for interactively learning labeling rules; (2) an interactive system RULER based on our framework to enable labeling rule generation by interactive demonstration for document classification tasks; and (3) a comparative user study conducted with data scientists in performing real-world tasks to evaluate RULER and conventional data programming. We also make our research artifacts, including the RULER code and demo, publicly available [1].

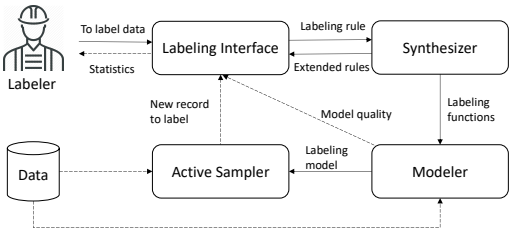

Figure 2: Overview of the data programming by demonstration (DPBD) framework. Straight lines indicate the flow of domain knowledge, and dashed lines indicate the flow of data. By extending data programming with programming by example, we bridge the gap between scalable training data generation and domain experts.

2 DPBD Framework

Problem Statement Given a dataset $D = \{d_1,...,d_m\}$ of data records and a set of labels $L = \{l_1,...,l_n\}$, we aim to develop a framework that enables human labelers to assign a label from L for each data record intelligently sampled from $D' \subset D$ ($|D'| \ll |D|$), while demonstrating their rationales for label assignments through visual interaction. Given a triplet (d'_i, v_i, l_j) of a data record, a visual interaction from the labeler, and the label assigned, we want this framework to effectively synthesize and propose labeling rules $R_{ij} = \{r_1,...,r_k\}$ for the labeler to choose from. Finally, we want the framework to optimally aggregate all the chosen rules (labeling functions) in order to create a labeled training set from $D \setminus D'$ with probabilistic labels in order to subsequently train discriminative models on it.

Framework Overview

The data programming by demonstration (DPBD) framework (Figure 2) has two input sources: the human labeler and the raw text data. The labeler is the subject matter expert who has sufficient domain understanding to extract useful data signals and does not necessarily have programming experience. Given a dataset, our framework enables the labeler to label each record with a categorical label, while providing their labeling rationales by interactively marking relevant parts of the record and specifying relationships.

[1] https://github.com/megagonlabs/ruler

The output is a labeling model, trained to produce labels for the large unlabeled dataset automatically.

The DPBD framework has four main components. The labeler interacts with data via the *labeling interface*. The labeling interface records the labeler's interaction and compiles the interaction into a set of conditions. The *synthesizer* synthesizes labeling rules from these conditions and translates those chosen by the labeler into functions. Then, the selected functions are passed to the *modeler*, which builds a labeling model by optimally aggregating the generated functions.

Until a particular stopping criterion is met (e.g., reaching the desired model quality) or the labeler decides to exit, the *active sampler* selects the next data record to present the labeler.

2.1 Labeling Interface

The labeling interface is the workplace where the labeler quickly and intuitively encodes domain knowledge into labeling rules. It provides a way to express noisy explanations for labeling decisions using a visual interaction language, allowing the user to express domain knowledge without formalizing their ideas into computer programs or natural language explanations. This allows for more focus on patterns in the data while abstracting away any implementation concerns.

2.2 Generalized Labeling Model

The generalized labeling model (GLM) models the data records with *concepts* and *relationships* in a way that is interpretable to the user. The GLM views the data record as a series of tokens, where a token is a continuous subset of a record with no semantics attached. For example, in text data, a token can be any span (single char to multiple words) of the data record; in an image data record, it would be a 2D region, rectangular or free form; and in an audio data record, it would be a 1D 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, over text data, the labeler might define a concept of positive adjectives consisting of a set of tokens, each of which can imply a positive review. When labeling audio data, the labeler might create a concept to aggregate all clips that denote excitement or a specific speaker. This abstraction allows the user to teach the GLM what generalizations are relevant to the task.

A *relationship* represents a binary correlation between token-token, token-concept, or concept-concept. Some examples are membership (e.g., a token is in a concept), co-existence (e.g., opinion and aspect tokens), and positional (e.g., a person is standing left to a table [12]).

Table 1: Mapping from GLM elements to operations in the labeling interface.

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

Mapping GLM Elements to Operations Given the GLM specification described above, our framework also defines the operations that can be applied to those elements. Table 1 lists the GLM elements and the corresponding operations.

The implementation of both the labeling interface and the operations described in Table 1 would vary across data types and token definitions. The GLM may also perform transformations over the set of tokens to add expressivity, as we describe in the next section.

Compiling Operations into Labeling Rules Once the labeler finishes annotating an example using the provided operations and selects a label, the tokens are extracted from the annotation and used as the initial set of conditions to build rules. The synthesizer combines these conditions into labeling rules by selecting subsets of the conditions combined with different conjunctive formulas, according to the relationships the user has annotated. The synthesizer presents these extended labeling rules for the labeler to select from, choosing desired ones based on domain knowledge.

A labeling rule serves as an intermediate representation, interpretable by both the labeler and the synthesizer. In our framework, we adapt the notation of domain relational calculus [16] to represent these rules, which can be expressed as: $\{\texttt{tokens} \mid \texttt{conditions}\} \Rightarrow \texttt{label}$. The variable `tokens` is a sequence of tokens with existential quantification, and `conditions` is a conjunctive formula over boolean predicates that is tested over `tokens` on a data record. The predicates are first-order expressions, and each can be expressed as a tuple (T,lhs,op,rhs). T is an optional transformation function on a token identifier, a process of mapping the raw token to more generalized forms. Some example transformations are word lemmatization for text labeling, speech-to-text

detection in audio labeling, or object recognition in image labeling. lhs is a token, while rhs can be either a token, literal, or set. If rhs denotes a token, the transformation function T may also apply to rhs. op is an operator whose type depends of the type of rhs. If rhs is a token or literal, op detects a positional or an (in)equality relationship. Otherwise, if rhs is a set, op is one of the set operators $\{\in, \notin\}$. Since the `conditions` is in the conjunctive form, the order of labeler's interactions does not matter.

Example: Consider the binary sentiment classification (positive or negative) task on Amazon review data [14]. Observe the following review:

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

If the labeler thinks this data record has a positive sentiment, she can express her decision rationale using GLM. First, she may select two tokens that are related to the sentiment: `book` and `great`. Assume there are two concepts the labeler previously created: (1) `item`= {`book, electronics`}; and (2) `padj`= {`wonderful`}. The labeler realizes the token `great` can be generalized by the `padj` (positive adjective) concept, which means that the labeling rule will still be valid for any tokens in the concept, so she adds this token to the concept. Finally, the labeler creates a positional relationship from `book` to token `great` to indicate that they appear in the same sentence, before completing the labeling process. These operations compile into the labeling rule $r : \{t_1, t_2 \mid t_1 = \texttt{book} \wedge t_2 \in \texttt{padj} \wedge sentence\_idx(t_1) == sentence\_idx(t_2)\} \Rightarrow \texttt{positive}$. This rule is sent to the synthesizer for expansion and program synthesis.

2.3 Synthesizer

Given the compiled labeling rule from the labeling interface, the synthesizer extends one single labeling rule from the labeler's interaction to a set of more general labeling rules; and translates those labeling rules into computer programs. It is straightforward to translate the rules into executable computer programs (labeling functions), so in this section, we focus on how to synthesize the extended labeling rules.

The synthesizer generates labeling rules by optimizing two competing goals: maximizing generalization, so that more (unseen) data can be accurately labeled; and maximizing the coverage of the labeler's interaction, simply because labeler's interaction is the most valuable signal for labeling from domain knowledge. Of course, the larger the set of annotations in an interaction, the larger the set of labeling functions that can be synthesized. To keep rule selection as straightforward as possible for the user, we prioritize rules that cover more of the interaction, assuming that there is little redundancy.

We achieve generalization of the given rules using the following heuristics: (1) substituting tokens with concepts; (2) replacing general co-existence relationships with position-specific ones; and (3) applying the available transformations over the tokens. In RULER, we implement transformations that recognize named entity types such as `person` and `location`, extracted using the spaCy library [1]. These annotations are made visible to the user, and annotations containing named entities will generate functions that generalize to all instances of that entity.

Once the extended rules are generated, the rules are ranked by their generalization score—a measurement of how applicable a particular rule is. We define a data-independent generalization score for a labeling rule r as: $G(r) = \prod_{c \in r.conds} |c.rhs|$. Intuitively, $G(r)$ is calculated by counting how many different data instances that r can be used.

Example: Continuing with our Amazon review example, the synthesizer can derive the following labeling rules from r using these heuristics:

1. $\{t_1, t_2 \mid t_1 \in \texttt{item} \wedge t_2 \in \texttt{padj}\} \Rightarrow \texttt{positive}$
2. $\{t_1, t_2 \mid t_1 \in \texttt{item} \wedge t_2 \in \texttt{padj} \wedge idx(t_1) < idx(t_2)\} \Rightarrow \texttt{positive}$
3. $\{t_1, t_2 \mid t_1 = \texttt{book} \wedge t_2 \in \texttt{padj}\} \Rightarrow \texttt{positive}$

Note that labeling rule (1) is more general than (2) and (3) because all data records that can be labeled by (2) and (3) will be labeled the same way using labeling rule (1).

The top-k candidates ranked by the generalization score are displayed in the labeling interface for the labeler to accept or reject.

2.4 Modeler

The modeler component trains a model that can be used to annotate unlabeled datasets automatically. Naively aggregating the labeling functions can be either inaccurate (since labeling functions can be conflicting and correlated) or does not scale with large sets of unlabeled data [30]. Instead, the modeler encapsulates the ideas from traditional data programming [6, 30, 31] to build a generative

model that denoises the labeling functions and create training data. The user can then train a discriminative model to leverage other features beyond what is expressed by the labeling functions.

2.5 Active Sampler

To improve the model quality at faster rates, our framework uses an active sampler to choose the next data record for labeling. The active sampler selects the data record x^* with the highest entropy (i.e., the one that the labeling model is currently the most uncertain about): $x^* = \text{argmax}_x - \sum_i^{|L|} p_\theta(L_i \,|\, x) \log p_\theta(L_i \,|\, x)$ where $p_\theta(L_i \,|\, x)$ is the probability that example x belongs to class L_i, as predicted by the trained label model.

3 Ruler Interface

RULER is a web-based interactive system that builds on the data programming by demonstration (DPBD) framework introduced above to facilitate labeled training data preparation for document-level text classification models. For this, RULER leverages span-level features and relations in text documents demonstrated through visual interactions by users (labelers), as formalized by the DPBD framework. To begin a labeling task, the data owner needs to upload their unlabelled dataset, in addition to a small labeled development set, and optionally a small test and validation set. This mirrors the data requirements of Snorkel, which the underlying DPBD modeler encapsulates. In the rest of this section, we discuss the user interface and interactions of RULER along with its implementation details in operationalizing DPBD for text document classification.

Recall that the purpose of the labeling interface in DPBD (Section 2.1) is to enable the labeler to encode domain knowledge into rules through visual interaction. To this end, RULER interface provides affordances through 6 basic views (Figure 3), which we briefly describe below—the letters A-F refer to annotations in Figure 3.

Labeling Pane (A) is the main view where the user interacts with document text. The Labeling Pane (Figure 4) shows a single document at a time and supports all the labeling operations defined by the GLM in the context of text data. The user can annotate spans by highlighting them directly with the cursor or adding them to a concept. These spans can be linked together if the relationship between them is significant to the user. Once the user selects a document label (class) from the options displayed, the system generates a diverse set of labeling functions to suggest to the user.

Concepts Pane (B) allows users to create concepts, add and edit tokens (whole words surrounded by non-alphabetical characters) or regular expressions, and see annotations over their text automatically added when a match is found (Figure 5). This interaction allows users to abstract away details about specific language use by grouping tokens or regular expressions into concepts.

Suggested Functions (C) shows the labeling functions suggested by the system. The user can select any functions that seem reasonable, and only then are they added to the underlying labeling model that is iteratively built.

Labeling Statistics (D) displays current statistics of the label model computed over the development set, and differential changes incurred by the last data interaction. Because this panel updates as the user interacts, the user can explore the space of labeling functions efficiently in terms of time, computation, and human effort.

End-model Statistics (E) shows the performance statistics for an end-discriminative model for which the user intends to collect training data. For example, in our user study, we used a logistic regression model with a bag of words features on the generated training data. We evaluate this model on the small held-out test set and show the statistics in this pane. This panel updates only when the user chooses to retrain the model, to deter from overfitting to the development set.

Selected Functions (F) lists of currently selected labeling rules that make up the labeling model and shows each rule's performance statistics based on the development set. The user can click to open a details panel showing observed incorrect labels and sample texts labeled by this function.

4 Evaluation

We evaluated RULER alongside manual data programming using Snorkel [30]. Although non-programmer domain experts are a target audience for this technology, we wanted our evaluation to show that the RULER labeling language is expressive enough to create models comparable to manual data programming. We also wanted to understand the trade-offs afforded by each method in order to help programming-proficient users decide which is best for their situation. In order to make these comparisons, we conducted

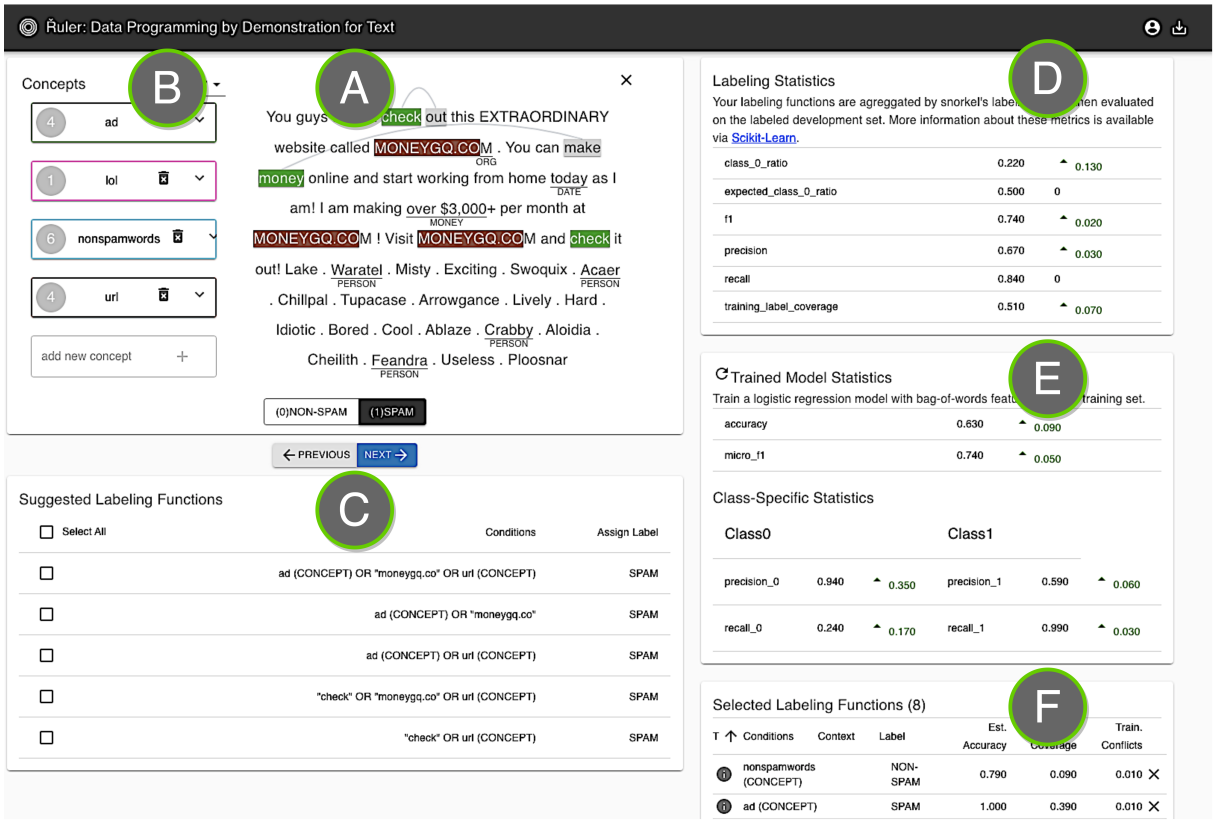

Figure 3: RULER User Interface. RULER synthesizes labeling rules based on rationales expressed by users by interactively marking relevant parts of the example and specifying implied semantic relations among them.

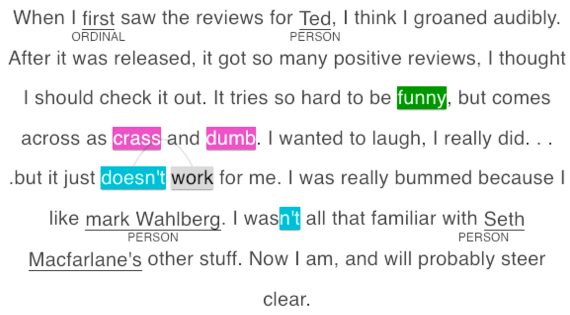

Figure 4: RULER Labeling Pane: User conveys domain knowledge using a visual interaction language. Annotations are color coded by the concepts they are assigned to. This example is from the Amazon reviews dataset [14].

Figure 5: Left: Example concept created to capture negation. Right: example text highlighting as concept elements are matched in the text, and annotations created once the element is submitted.

a user study with 10 programming-proficient data scientists and measured their task performance accuracy in completing two labeling tasks using the two methods. In addition to task performance, we analyzed both accessibility and expressivity using the qualitative feedback elicited from participants.

In an initial pilot study, we included a third condition, BabbleLabble [13]. For this method, users express labeling rationales in natural language, which the tool then parses into labeling rules. Participants found BabbleLabble to be limited in terms of what patterns they could express and how to express them, as they "tried to express it in a parsable sentence" and faced errors. The preliminary results led us to believe that although BabbleLabble may be suitable for high-volume approaches like crowd-sourcing, it can be frustrating for a domain expert or lay user who is both providing the explanations and creating and debugging the label model. Based on these observations, we removed BabbleLabble from our evaluation.

Participants We recruited participants with Python programming experience through our professional network (none were involved in this project). Note that RULER can be used by programmers and non-programmer domain experts alike, but a fair

comparison with Snorkel requires proficiency in conventional programming. All participants had significant programming experience (avg=12.1 years, std=6.5). Their experience with Python programming ranged from 2 to 10 years, with an average of 5.2 years (std=2.8).

Experimental Design We carried out the study using a within-subjects experiment design, where all participants performed tasks using both conditions (tools). The sole independent variable controlled was the method of creating labeling functions. We counterbalanced the order in which the tools were used, as well as which classification task was performed with which tool.

Tasks and Procedure We asked participants to write labeling functions for two prevalent labeling tasks: spam detection and sentiment classification. These tasks were chosen because the user does not need to be a domain expert to understand the differences between the classes. Participants performed these two tasks on YouTube Comments and Amazon Reviews, respectively. Participants received 15 mins of instruction on how to use each tool, using a topic classification task (electronics vs. guns) over a newsgroup dataset [32]. We asked participants to write as many functions as they considered necessary for the goal of the task. There were given 30 mins to complete each task, and we recorded the labeling functions they created as well as these functions' individual and aggregate performances. After completing both tasks, participants also filled out an exit survey, providing their qualitative feedback.

For the manual programming condition, we iteratively developed a Jupyter notebook interface based on the Snorkel tutorial. We provided a section for writing functions, a section with diverse analysis tools, and a section to train a logistic regression model on the labels they had generated (evaluated on the test set shown to the user, which is separate from our held-out test set used for the final evaluation).

5 Results

To analyze the performance of participants' labeling functions for each condition, we select the labeling model that achieves the highest f1 score on the development set. For example, if a user performs the spam classification task using RULER, and the set of functions that they created 20 minutes into their 30-minute session attains the highest score on the development set, we use that model to evaluate this condition, rather than strictly using whatever

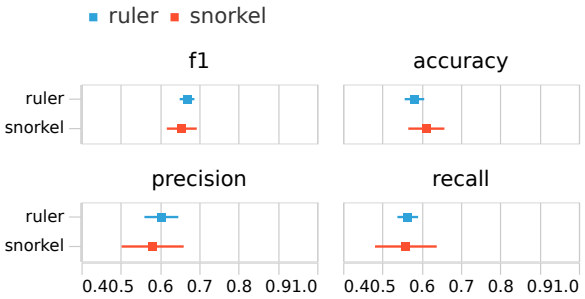

Figure 6: Performances of the classifier models trained on the probabilistic labels generated by participants' labeling models. Error bars show one standard deviation. Although manual programming allows participants to use existing packages (e.g., sentiment analysis packages), RULER performs comparably with Snorkel in both tasks.

model the participant has at the end of the 30-minute session. We use each label model to generate a training dataset, which we then use to train a logistic regression model with bag-of-words features. Finally, we evaluate the logistic regression model's performance on a held-out test set (400 examples). We also analyze participants' subjective ratings on a Likert scale of 5 (1–5, higher is better) in their exit surveys. We use the paired Wilcoxon signed-rank test to assess the significance of differences in prediction metrics and subjective ratings between RULER and Snorkel. We also report the effect size r for all our statistical comparisons.

Model Performance We find that RULER and Snorkel provide comparable model performances (Figure 7). The logistic regression models trained on data produced by labeling models created using RULER have slightly higher f1 ($W = 35$, $p = 0.49$, $r = 0.24$), precision ($W = 30$, $p = 0.85$, $r = 0.08$), and recall ($W = 25$, $p = 0.85$, $r = 0.08$) scores on average. Conversely, accuracy is slightly higher ($W = 17$, $p = 0.32$, $r = 0.15$) for Snorkel models on average than RULER. While these differences are not statistically significant, it indicates that the users achieved compareable performance through a demonstration as opposed to programming labeling functions, suggesting a broader user base.

Subjective Ratings and Preferences Participants find RULER to be significantly easier to use ($W = 34$, $p = 0.03 < 0.05$, $r = 0.72$) than Snorkel. Similarly, they consider RULER easier to learn ($W = 30$, $p = 0.1$, $r = 0.59$) than Snorkel. On the other hand, as we expected, participants report Snorkel to be more expressive ($W = 0$, $p = 0.05$, $r = 0.70$) than RULER. However, our participants appear to consider accessibility (ease of use and ease of learning)

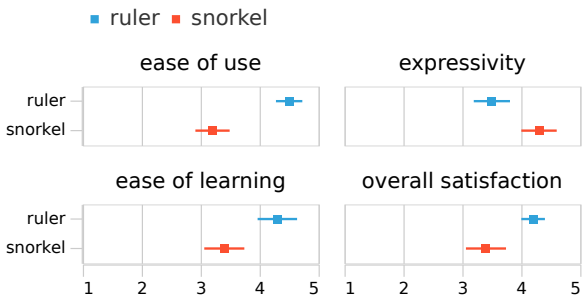

Figure 7: Participants' subjective ratings on ease of use, expressivity, ease of learning and overall satisfaction, on a 5-point Likert scale. Error bars show one standard deviation. RULER is considered by participants to easier to use and learn, though using Snorkel alone is considered to be more expressive.

to be more important criteria, rating RULER higher ($W = 43$, $p = 0.12$, $r = 0.51$) than manual data programming with Snorkel for overall satisfaction.

When asked which tool they prefer overall, two users preferred Snorkel, four preferred RULER, and the remaining four said it depends on the task and data. If they wanted to get data quickly or the dataset required many domain-specific keywords, they would opt for RULER, whereas Snorkel would be preferred if given more time. One user summarized it as "Simple label function[s] that rely on keywords are much easier and faster to write with RULER. For both tasks, I did not write complex label logic, so with the same time, I can write more label functions with RULER."

The reason some users preferred Snorkel in certain situations was expressivity, yet interestingly almost three-quarters (72.3%) of the functions that users wrote in Snorkel could be captured through RULER interactions. The types of functions not captured included: functions that used Python sentiment analysis packages, and functions that counted the number of occurrences of a word, the length of the text, or, in one case, the ratio of alphabetical characters. This suggests that even skilled programmers can benefit from using both systems, using RULER to more quickly capture domain-specific concepts and language use, and then manually adding functions based on their new understanding of the data.

For users who are not skilled at programming, RULER is, to the best of our knowledge, the only tool available to help leverage data programming with full control over the functions. Our user study shows that in addition to the benefit RULER provides to this group, it may even help skilled programmers save time and create better models, either in conjunction with traditional programming or alone.

6 Related Work

We build on earlier work in weak supervision, programming by demonstration, and learning from feature annotations provided by users.

Weak Supervision In order to reduce the cost of labeled data collection, weak supervision methods leverage noisy, limited, or low precision sources such as crowdsourcing [15], distant supervision [26], and user-defined heuristics [10] to gather large training data for supervised learning. Data programming [30, 31] is a programmatic approach to weak supervision by heuristics, where domain experts provide functions which are then used to label training data at scale and train ML models using probabilistic labels. RULER aims to make data programming easier and more accessible for document classification tasks.

Program Synthesis by Demonstration Automated synthesis of programs that satisfy a given specification is a classical artificial intelligence (AI) problem [37]. Generating programs by example or demonstration is an instance of this problem. The terms programming by example (PBE), or programming by demonstration (PBD) are often 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, being applied to automate various data analysis tasks [9]. PBD systems aim to empower end-user programming in order to improve user productivity [4, 7, 18, 19, 23, 27, 28]. One of the core research questions in PBD is how to generalize from seen examples or demonstrations. To generalize, PBD systems need to resolve the semantic meaning of user actions over relevant (e.g., data) items. Prior approaches incorporate a spectrum of user involvement, from making no inference (e.g., [11, 28]) to using AI models with no or minimal user involvement, to synthesize a generalized program (e.g., [9, 17, 20, 24, 25]). Our framework takes a hybrid approach within the spectrum above and combines inference and statistical ranking along with interactive demonstration.

Learning from Feature Annotations Prior work proposes methods for learning from user provided features [8, 21, 29], rationales [5, 36, 38, 39], and natural language explanations [13, 34]. BabbleLabble [13] uses a rule-based parser to turn natural language explanations into labeling functions and aggregates these functions using Snorkel. RULER

also learns labeling functions from high level imprecise explanations and aggregates them using the Snorkel framework. However, RULER enables users to supply their rationales through interactive visual demonstrations, removing the cognitive load of having to formalize one's intuition into either a programming or natural language.

7 Discussion

RULER prioritizes accessibility over expressivity. Is this trade-off inevitable? There are many ways we could improve the expressivity of RULER, for example, by extending analysis on the context of user demonstrations, or by giving users more direct control over the synthesized labeling functions. However, many of these improvements can only be informed by how people use RULER in real-world applications. With this in mind, deriving additional insights into how users without programming proficiency use RULER is an essential area to explore, and open-sourcing RULER is a step forward in this direction. We especially hope that RULER can be beneficial for domains like healthcare, where the domain expert's time is very valuable and there is little tolerance for low-quality models.

Future research also includes developing fast search and ranking algorithms, and experimenting with different active learning strategies to effectively search and navigate the vast joint space of labeling functions and data examples.

Accessibility is a key to wider adoption of any technology and machine learning (ML), especially in data-hungry supervised forms, is no exception. In this paper, we presented RULER, a data programming by demonstration (DPBD) system for quickly generating labeling functions to create training datasets for document-level classification tasks. RULER uses the DPBD framework to convert user rationales, interactively expressed as span-level annotations and relations, to labeling rules. DPDB is a general human-in-the-loop framework that aims to ease writing labeling functions, improving data programming's accessibility and efficiency. Through a user study with 10 data scientists performing real-world labeling tasks for classification, we evaluated RULER together with conventional data programming and found that RULER enables more accessible data programming without any loss of performance in the final models. Our study results also suggest that RULER may benefit even skilled programmers, as many functions can be captured more easily through visual interactions using our system than by coding them from scratch. We release RULER as open-source software to support future applications and extended research.

References

[1] spaCy: Industrial-strength natural language processing. https://spacy.io/.

[2] Interactive programmatic labeling for weak supervision. In *KDD DCCL Workshop*, 2019.

[3] A. Akbik, O. Konomi, and M. Melnikov. Propminer: A workflow for interactive information extraction and exploration using dependency trees. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 157–162, 2013.

[4] 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.

[5] S. Arora and E. Nyberg. Interactive annotation learning with indirect feature voting. In *Procs. NAACL-HLT Student Research Workshop and Doctoral Consortium*, 2009.

[6] 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.

[7] V. Dibia and Ç. Demiralp. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE Computer Graphics and Applications*, 39(5):33–46, 2019.

[8] G. Druck, B. Settles, and A. McCallum. Active learning by labeling features. In *Proc. EMNLP*, 2009.

[9] 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.

[10] 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.

[11] 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.

[12] M. Haldekar, A. Ganesan, and T. Oates. Identifying spatial relations in images using convolutional neural networks. In *IJCNN*, pages 3593–3600, 2017.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] P. Liang, M. I. Jordan, and D. Klein. Learning from measurements in exponential families. In *Proc. ICML*, 2009.

[22] H. Lieberman. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann Publishers, 2001.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] B. A. Myers. Watch what i do. chapter Peridot: Creating User Interfaces by Demonstration, pages 125–153. MIT Press, 1993.

[28] 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.

[29] H. Raghavan, O. Madani, and R. Jones. Interactive feature selection. In *Proc. IJCAI*, 2005.

[30] 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.

[31] 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.

[32] J. Rennie and K. Lang. The 20 newsgroups data set, 2008.

[33] 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.

[34] S. Srivastava, I. Labutov, and T. Mitchell. Joint concept learning and semantic parsing from natural language explanations. In *Procs. EMNLP*, 2017.

[35] P. Varma and C. Ré. Snuba: automating weak supervision to label training data. *Proceedings of the VLDB Endowment*, 12(3):223–236, 2018.

[36] L. Von Ahn, R. Liu, and M. Blum. Peekaboom: a game for locating objects in images. In *Proc. CHI*, 2006.

[37] 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.

[38] O. Zaidan and J. Eisner. Modeling annotators: A generative approach to learning from annotator rationales. In *Proc. EMNLP*, 2008.

[39] O. Zaidan, J. Eisner, and C. Piatko. Using "annotator rationales" to improve machine learning for text categorization. In *Procs. NAACL-HLT*, 2007.