Ruler: Data Programming by Demonstration for Document Labeling

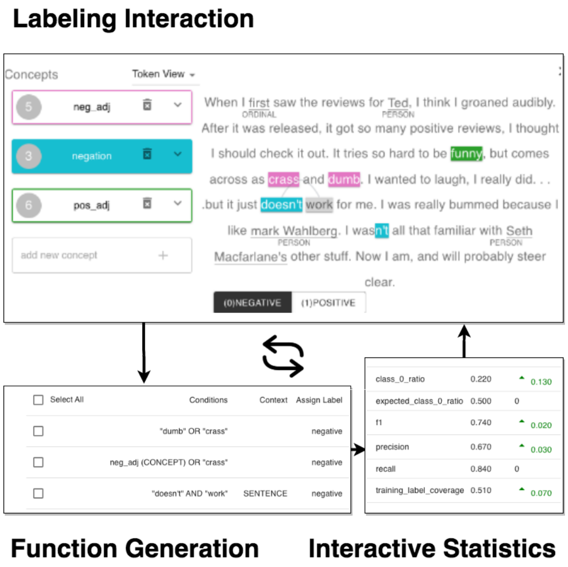
Anonymous EMNLP submission

Abstract

Data programming was introduced to reduce the cost of curating labeled training data. Writing data programs (labeling functions) manually requires both programming literacy and domain expertise. However, many subject matter experts are not proficient programmers and transferring domain expertise into labeling functions by enumerating rules and thresholds is not only time consuming but also inherently difficult. Here we introduce RULER, an interactive system that synthesizes labeling rules using span-level interactive demonstrations of users on document examples. RULER is an instance of data programming by demonstration (DPBD), a new framework that 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 compare RULER with conventional data programming through a user study conducted with 10 data scientists creating labeling functions for sentiment and spam classification tasks. Results show RULER is easier to use and learn and offers higher overall satisfaction, while providing discriminative model performances comparable to ones achieved by conventional data programming.

1 Introduction

Machine learning (ML) models used in practice today are predominantly supervised models 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 [34].



When I first saw the reviews for Ted, I think I groaned audibly. After it was released, it got so many positive reviews, I thought I should check it out. It tries so hard to be funny, but comes across as crass and dumb. I wanted to laugh, I really did. . . .but it just doesn't work for me. I was really bummed because I like mark Wahlberg. I wasn't all that familiar with Seth Macfarlane's other stuff. Now I am, and will probably steer clear.

□ Select All	Conditions	Context	Assign Label
□	"dumb" OR "crass"		negative
□	neg_adj (CONCEPT) OR "crass"		negative
□	"doesn't" AND "work"	SENTENCE	negative

class_0_ratio	0.220	▲ 0.130
expected_class_0_ratio	0.500	0
f1	0.740	▲ 0.020
precision	0.670	▲ 0.030
recall	0.840	0
training_label_coverage	0.510	▲ 0.070

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.

Weak supervision methods such as crowdsourcing [15], distant supervision [27], and user-defined heuristics [10] enable the use of noisy or imprecise sources to gather large training datasets. Data programming [5, 31, 32] 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 [31, 32] or generated automatically [13, 36] to denoise labeling functions. However, little is known about user experience in writing labeling functions and how to improve it [2]. Writing data programs can be challenging. Most domain experts or lay users have no or little programming literacy, and even for those who are proficient programmers, it is often difficult to convert domain knowledge to a set of rules by writing programs.

We introduce RULER (Figure 1), an interactive system that enables more accessible data programming to create labeled training datasets for document classification models. RULER automatically generates labeling rules from user's labeling rationales or intents demonstrated with span-level annotations and their relations on specific examples. Underlying RULER is our Data Programming by Demonstration (DPBD) framework, which RULER operationalizes for text documents.

DPBD is a new framework that aims to make creating labeling functions easier by learning them from users' interactive visual demonstrations. DPBD 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. We partly built DPBD on the basis of two lines of prior research. The first is programming by demonstration (PBD) or example (PBE), e.g., [9, 23], 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 [39, 40].

Through a user study conducted with 10 data scientists, we evaluate RULER alongside manual data programming using Snorkel [31]. We measure the predictive performances of models created by participants for two common labeling tasks, 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 facilitates more accessible creation of labeling functions without a 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].

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 interactively assign a label from L for each data record efficiently 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.

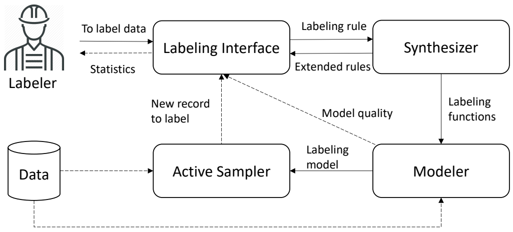


Figure 2: Overview of the data programming by demonstration (DPBD) framework. Straight lines indicate the flow of domain knowledge, and dashed lines indicate the flow of data. By extending data programming with programming by example, we bridge the gap between scalable training data generation and domain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

Framework Overview The data programming by demonstration (DPBD) framework (Figure 2) has two input sources: the human *labeler*, and the *data* that is to be labeled. The labeler is the subject matter expert who has sufficient domain understanding to extract useful signals from data, and is assumed to have no or little 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 semantics and relationships among them. The output is a labeling model, which is trained to automatically produce labels for the large set of unlabeled data.

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 labeling rule. The *synthesizer* synthesizes labeling rules and translates those chosen by the labeler into program functions. Third, the selected functions are passed to the *modeler*, which builds a labeling model by optimally aggregating the gen-

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

erated functions. Until a certain stopping criterion is met (e.g., reaching a 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 encodes domain knowledge into labeling rules. It provides a way to express noisy explanations for labeling decisions using a visual interaction language, which allows the user to express domain knowledge without having to formalize 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

Inspired by the entity-relationship model [6] in database modeling, the generalized labeling model (GLM) models the data records with *concepts* and *relationships*. 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 express excitement, or of a specific speaker. This abstraction allows the user to teach the GLM which 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 [21]), and positional (e.g., a person is standing left to a table [12]).

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.

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

The implementation of both the labeling interface and the operations described in Table 1 would vary across data types and token definitions. To add expressivity, the GLM may also perform transformations over the set of tokens, 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 from which to build rules. The synthesizer combines these conditions into labeling rules by selecting subsets of the conditions to be combined with different conjunctive formulas, according to the relationships the user has annotated. The synthesizer extends the initial set of labeling rules and presents the extended labeling rules for the labeler to select from, choosing desired ones based on domain knowledge.

A labeling rule serves as an intermediate language, 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 is can be either token, literal or a 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` concept, which means that the labeling rule will still be valid if this token is replaced by 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 $r : \{t_1,t_2 \mid t_1 = \texttt{book} \wedge t_2 \in \texttt{padj} \wedge idx(t_1) < idx(t_2)\} \Rightarrow \texttt{positive}$.

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 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.

Given the labeling rule compiled from a labeler's interaction, the synthesizer generates more labeling rules while 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 easy as possible for the user, in this case 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 (for example, object recognition in a section of an image).

Once the extended rules are generated, the rules are ranked by their generalization score—a measurement of how applicable a certain 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. $\{t_1,t_2 \mid t_1 \in \texttt{item} \wedge t_2 \in \texttt{padj} \wedge idx(t_1) < idx(t_2)\} \Rightarrow$ `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).

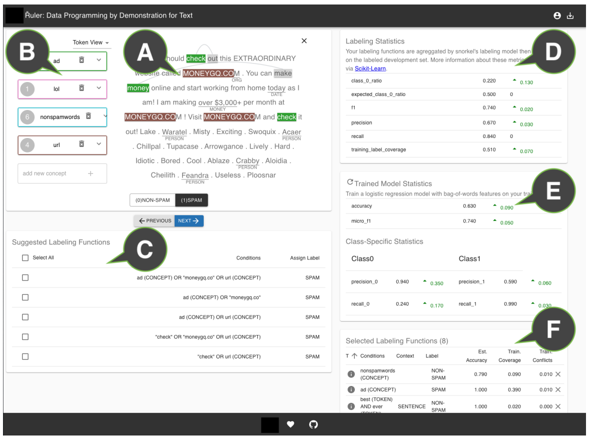
The top-k candidates ranked by the generalization score are displayed in the labeling interface for the labeler to accept or reject.

2.4 Modeler

The modeler component trains a model that can be used to automatically annotate unlabeled datasets. Naively 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 [31]. Instead, the modeler encapsulates the ideas from traditional data programming [5, 31, 32] 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.

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): $x^* = \text{argmax}_x - \sum_i^{|L|} p_\theta(L_i \mid x) \log p_\theta(L_i \mid x)$ where $p_\theta(L_i \mid x)$ is the probability that example x belongs to class L_i, as predicted by the trained label model.



…hould check out this EXTRAORDINARY website called MONEYGQ.COM . You can make money online and start working from home today as I am! I am making over $3,000+ per month at MONEYGQ.COM ! Visit MONEYGQ.COM and check it out! Lake . Waratel . Misty . Exciting . Swoquix . Acaer . Chillpal . Tupacase . Arrowgance . Lively . Hard . Idiotic . Bored . Cool . Ablaze . Crabby . Aloidia . Cheilith . Feandra . Useless . Ploosnar

Suggested Labeling Functions

□ Select All	Conditions	Assign Label
□	ad (CONCEPT) OR "moneygq.co" OR url (CONCEPT)	SPAM
□	ad (CONCEPT) OR "moneygq.co"	SPAM
□	ad (CONCEPT) OR url (CONCEPT)	SPAM
□	"check" OR "moneygq.co" OR url (CONCEPT)	SPAM
□	"check" OR url (CONCEPT)	SPAM

Labeling Statistics

Your labeling functions are agreggated by snorkel's labeling model then on the labeled development set. More information about these metric via Scikit-Learn.

class_0_ratio	0.220	▲ 0.130
expected_class_0_ratio	0.500	0
f1	0.740	▲ 0.020
precision	0.670	▲ 0.030
recall	0.840	0
training_label_coverage	0.510	▲ 0.070

Trained Model Statistics

Train a logistic regression model with bag-of-words features on your tra

accuracy	0.630	▲ 0.090
micro_f1	0.740	▲ 0.050

Class-Specific Statistics

Class0			Class1		
precision_0	0.940	▲ 0.350	precision_1	0.590	▲ 0.060
recall_0	0.240	▲ 0.170	recall_1	0.990	▲ 0.030

Selected Labeling Functions (8)

T ↑ Conditions	Context	Label	Est. Accuracy	Train. Coverage	Train. Conflicts
nonspamwords (CONCEPT)		NON-SPAM	0.790	0.090	0.010 ✕
ad (CONCEPT)		SPAM	1.000	0.390	0.010 ✕
best (TOKEN) AND ever (TOKEN)	SENTENCE	NON-SPAM	1.000	0.020	0.000 ✕

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 group or pairwise semantic relations among them.

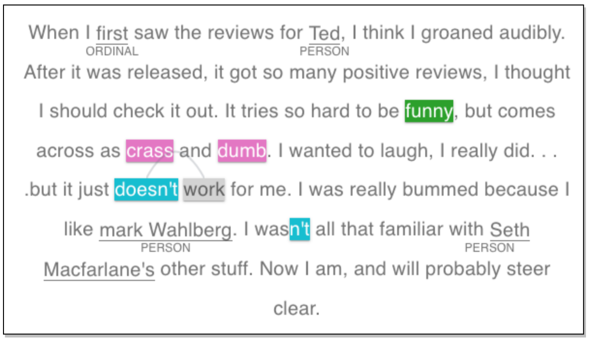
3 Ruler

RULER is a web-based interactive system that builds on the data programming by demonstration (DPBD) framework introduced above to facilitate easier 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.

3.1 User Interface and Interactions

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.

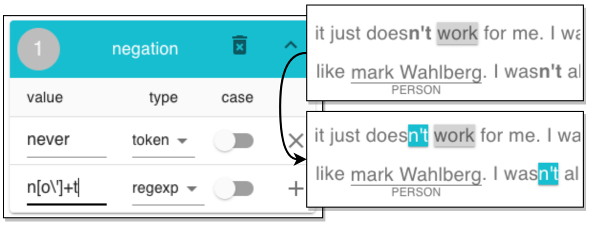
Labeling Pane (A) is the main view where the user interacts with document text. Labeling Pane (Figure 4) shows contents of a single document at a time and supports all the labeling operations defined by the GLM in the context of text data. The user can annotate spans either by highlighting them directly 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.



When I first saw the reviews for Ted, I think I groaned audibly. After it was released, it got so many positive reviews, I thought I should check it out. It tries so hard to be funny, but comes across as crass and dumb. I wanted to laugh, I really did. . . .but it just doesn't work for me. I was really bummed because I like mark Wahlberg. I wasn't all that familiar with Seth Macfarlane's other stuff. Now I am, and will probably steer clear.

Figure 4: RULER Labeling Pane, where the user conveys domain knowledge using a visual interaction language. Annotations are color coded by the concepts they are assigned to.

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.



value	type	case
never	token	
n[o\']+t	regexp	

it just doesn't work for me. I wa
like mark Wahlberg. I wasn't al

it just doesn't work for me. I wa
like mark Wahlberg. I wasn't al

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.

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, they can quickly explore the space of labeling functions with a very low cost 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. This model is evaluated on the small held-out test set, and statistics are shown in this pane.

Selected Functions (F) lists of currently selected labeling rules that make up the labeling model along 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.

3.2 Server and Model

RULER's backend comprises the synthesizer (Section 2.3), modeler (Section 2.4) and active sampler (Section 2.5) components. The backend components are all implemented in Python 3.6. In addition to the function generation defined in Section 2.3, RULER's synthesizer also augments labeling rules using existing text processing libraries. It enhances the text with 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. In our implementation, relationships can encompass co-occurrence in the same sentence as well as in the document. RULER encapsulates the Snorkel library [31] into its modeler to aggregate the generated labeling functions.

4 Evaluation

We evaluate RULER alongside manual data programming using Snorkel [31]. Our goal is to better understand the trade-offs afforded by each method. To this end, we conducted a user study with 10 data scientists and measured their task performance accuracy in completing two labeling tasks. In addition to task performance, we also analyzed the accessibility and expressivity of both methods using the qualitative feedback elicited from participants and our observations gathered during the study sessions.

In an initial pilot study we included a third condition, BabbleLabble [13], where users express labeling rationales in natural language which are then parsed into labeling rules. Participants found BabbleLabble to be limited both in terms of what patterns could be expressed and how to express, as they "tried to express it in a parsable sentence" and faced errors. This leads us to believe that although BabbleLabble may be suitable for high-volume approaches like crowd-sourcing, it can 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 we performed with which tool.

Tasks and Procedure 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. Participants received 15 mins of instruction on how to use each tool, using a topic classification task (electronics vs. guns) over a newsgroup dataset [33] as an example. 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 and 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 heldout test set used for the final evaluation).

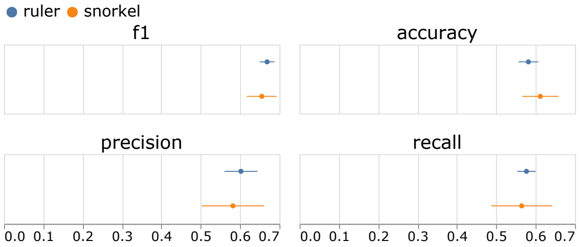


Figure 6: Performances of the classifier models trained on the probabilistic labels generated by participants' labeling models. Although manual programming allows participants to use existing packages (e.g., sentiment analysis packages), Ruler performs comparably with Snorkel in both tasks.

5 Results

To analyze the performance of the labeling functions created by participants, for each participant we select and task the labeling model that achieved the highest f1 score on the development set. For each labeling model, we then train a logistic regression model on a training dataset generated by the model. We finally evaluate the performance of the logistic regression model on a heldout test set (400 examples). We also analyze the subjective ratings provided by participants 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. However these differences are not statistically significant.

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

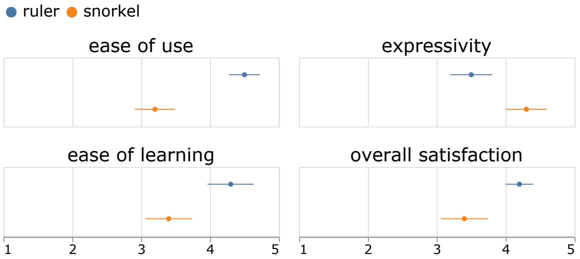


Figure 7: Participants' subjective ratings on ease of use, expressivity, ease of learning and overall satisfaction, on a 5-point Likert scale. Ruler is considered by participants to easier to use and learn, though using Snorkel alone is considered to be more expressive.

When asked which tool they prefer overall, 2 users prefered Snorkel, 4 prefered Ruler, and the remaining 4 said it depends on the task and data. If they wanted to get data quickly, or if the dataset required a lot of domain-specific keywords, many would opt for Ruler, whereas Snorkel would be useful given more time. One user summarized it thus "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 users preferred Snorkel in certain situations was for added 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: those using 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 the user who is 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 To reduce the cost of labeled data collection, weak supervision methods leverage noisy, limited, or low precision sources such as crowdsourcing [15], distant supervision [27], and user-defined heuristics [10] to gather large training data for supervised learning. Data programming [31, 32] 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. RULER aims to make data programming easier for document classification tasks.

Program Synthesis by Demonstration Automated synthesis of programs that satisfy a given specification is a classical artificial intelligence (AI) problem [38]. Generating programs by examples or demonstration is an instance of this problem. 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 [9]. PBD systems aim to empower end user programming in order to improve user productivity [3, 7, 18, 19, 24, 28, 29]. 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, 29]) to using AI models with no or minimal user involvement, to synthesize a generalized program (e.g., [9, 17, 20, 25, 26]). 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, 22, 30], rationales [4, 37, 39, 40], and natural language explanations [13, 35]. 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 to formalize one's intuition in a programming or natural language.

7 Discussion

RULER prioritizes accessibility over expressivity. Is this trade-off inevitable? The expressivity of RULER can be enhanced by extended semantic and syntactic analysis of the document context of user demonstrations. Enabling manual revision of synthesized labeling functions at multiple levels of abstraction can be also useful. In this context, further improving the expressivity of RULER through use cases without diminishing its accessibility is an important area of future research. Deriving additional insights into how users with limited or no programming proficiency would use RULER is another area of future work, and open sourcing RULER is a step forward in this direction. 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.

In this paper we presented RULER, a data programming by demonstration (DPBD) system for easily generating labeling functions to create training datasets for document-level classification tasks. RULER converts user rationales interactively expressed as span-level annotations and relations to labeling rules using the DPBD framework. DPDB is a general human-in-the-loop framework that aims to ease writing labeling functions, improving the accessibility and efficiency of data programming. 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 loss in the performance of labeling models created. Results of our study also suggested that, even for skilled programmers, the majority of functions they write can be captured more easily through visual interactions using our system. We release RULER as an 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] 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.

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

[5] 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.

[6] P. P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[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] Y. Li, A. Feng, J. Li, S. Mumick, A. Y. Halevy, V. Li, and W. Tan. Subjective databases. *PVLDB*, 12(11):1330–1343, 2019.

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

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

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

[31] 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.

[32] 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.

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

[34] 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.

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

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

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

[38] 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.

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

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