The ML Data Prep Zoo: Towards Semi-Automatic Data Preparation for ML

Vraj Shah Arun Kumar
University of California, San Diego
{vps002,arunkk}@eng.ucsd.edu

ABSTRACT

Data preparation (prep) time is a major bottleneck for many ML applications. It is often painful grunt work that is handled manually by data scientists, reducing their productivity and raising costs. It is also a roadblock for emerging AutoML platforms. We envision a new line of community-driven research to tackle this bottleneck based on a simple philosophy: *use ML to semi-automate data prep for ML*. For impactful research on this problem, we believe the major impediment is not new algorithms or theory but rather common task definitions and benchmark labeled datasets. To this end, we formalize a few major data prep tasks for ML over structured data as applied ML tasks. We discuss research challenges in scaling up data labeling, defining accuracy metrics, and creating practical tool support. We present a case study of our progress on a key data prep task: ML schema inference. Finally, we propose a public "zoo" of labeled datasets and pre-trained ML models for data prep tasks to act as a community-led repository for further research on this problem.

1 INTRODUCTION

Surveys of data scientists show that ML data prep often dominates their time and effort, even up to 80% [7]. It is tedious grunt work involving tasks such as identifying feature types and extracting feature values. Today, it is performed mostly manually in tools like Python and R, reducing data scientists' productivity and raising costs. Modern datasets also often have 1000s of columns, worseing this issue. Furthermore, SalesForce, Google, and other cloud vendors are starting to offer end-to-end AutoML platforms for enterprises; manual data prep at this scale of millions of datasets is untenable [8].

Challenge: Semantic Gap. While the DB community has long studied data cleaning/prep for SQL analytics, little work has studied the peculiarities of ML data prep. The *semantic gap* between what an *attribute* is in a DB/data file and what a *feature* is for ML means many tasks have fallen through the cracks. Thus, a pressing grand challenge for the DEEM community is to construct a shared understanding/terminology of such tasks, understand why they are hard to automate, and standardize evaluation of (semi-)automated tools.

Our Vision. To meet the above challenge, we envision a community-driven effort for semi-automating ML data prep. Our philosophy is to *abstract specific ML data prep tasks and cast them as applied ML tasks*. This raises 3 questions. What are the tasks and what is their role? How to cast them as applied ML tasks? How to create benchmark datasets for comparing tools? In particular, we believe the critical limiting factor for impactful and replicable research in this space is not fancier algorithms or theory but the *availability of large high-quality labeled datasets* for ML data prep tasks. As an analogy, the formalization of the ImageNet task and dataset spurred major recent advances in ML-based vision.

This Paper. We present our vision of the *ML Data Prep Zoo*, a repository of common ML data prep task definitions, benchmark labeled datasets, and pre-trained ML models. Figure 1 illustrates 6 tasks we have defined so far based on our conversations with data scientists. In Section 2, we explain these tasks and how to cast them as applied ML tasks. In Section 3, we discuss key research questions in realizing this vision and explain our plans. In Section 4, we present a case study of our progress on the first task: *ML schema inference*. For instance, our labeled data-based applied ML approach yielded a whopping 30% *lift* for identifying numeric features among attributes compared to existing rule-based approaches in Python Pandas and TensorFlow DataValidation [2]. In Section 5, we describe the ML Data Prep Zoo repository for our datasets and models and announce competitions for community contributions. Section 6 discusses related work.

2 DATA PREP TASKS FOR ML

2.1 Current Scope

We focus on tabular/relational data, typically exported from a DBMS with DB schema or managed as files (CSV, JSON, etc.). Either way, we assume the loaded dataset is a single table with column names available. ML users usually load such data into a Python/R "dataframe" for ML. This stage is our focus; this work does *not* focus on feature engineering (e.g., binning, removing outliers, Word2Vec, etc.). We call ML models to be trained on the prepared data "target models." Figure 1 shows a typical data prep workflow. Next, we dive into a few major steps in this workflow and discuss how we could cast the human's intuition as an applied ML task.

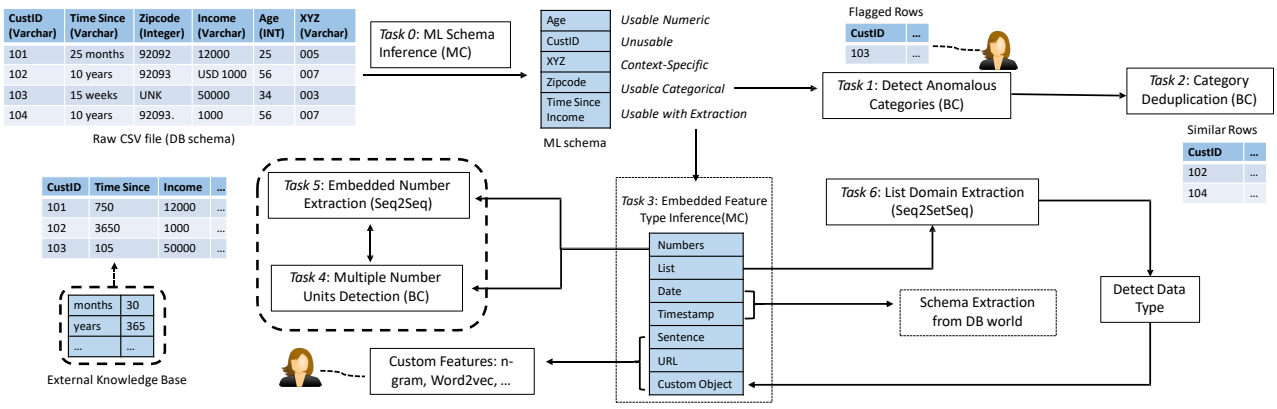

Figure 1: Illustrating major data prep tasks. The user loads a customers table to train, say, a churn predictor. BC stands for binary classification. MC stands for multi-class classification. Seq2Seq stands for sequence-to-sequence learning. Seq2SetSeq stands for sequence-to-set-of-sequence learning.

2.2 Task 0: ML Schema Inference

Description and Example. The first step is to infer the "ML schema": what is the feature type of each column? Most ML models recognize only *numeric* or *categorical* features. This task is surprisingly hard to automate accurately due to the *semantic gap* between DB and ML schemas. For example, `Zipcode` in Figure 1 is an integer; so, Pandas will call it a numeric feature, which is nonsensical! This semantic gap is bridged today manually by converting it to categorical. This issue is common in real datasets, since categories are often stored as integers, e.g., disease codes, product types, etc. Real datasets also often have 100s of features, which means the manual grunt work quickly adds up.

Casting as an ML Task. We bridge the semantic gap by casting this task as an ML classification task. In fact, our trained ML models yield 30% lift for predicting *numeric* vs. *non-numeric* features against existing rule-based heuristics in Pandas and TensorFlow Data Validation (TFDV) [2] (details in Section 4). The raw features are a whole column, including name such as "ZipCode" and sample values such as 90292, 92093, etc. in the above example. Two classes are *numeric* and *categorical*. But there is often not enough information in the data file to identify a column type, even for humans. This necessitates more classes; we created a 5-class vocabulary.

(1) *Usable-Numeric* and (2) *Usable-Categorical*: These are for columns that can be (almost) directly used for the target model, e.g., `Age` in Figure 1 is *Usable-Numeric*, while *Zipcode* is *Usable-Categorical*. (3) *Usable-with-Extraction*: Such columns have "messy" values, preventing direct use as numeric or categorical features, e.g., `Income` and `TimeSince` require custom extraction before being used as numeric features. Such extraction is hard to automate fully, but we later discuss a few common extraction tasks that can be cast as applied ML tasks. (4) *Unusable*: Such columns can *not* be used as features for the target model because they are not "generalizable," e.g., `CustID` is a primary key. (5) *Context-Specific*: This is a catch-all for columns whose type is hard to tell even for humans, e.g., `XYZ` has integers but is it really numeric (like `Age`) or categorical (like `Zipcode`)? To ascertain the type of such columns, data scientists typically need to manually check the application's data documentation.

2.3 Tasks for Usable-Categorical

While a *Usable-Categorical* column can be used directly, data scientists often seek to resolve two issues with its domain to boost target model accuracy: *missing value categories* and *duplicate categories*. For instance, we saw both"-999" and "unknown" for missing values and both "CA" and "California" for California in real datasets. One may want to discard missing value categories and instead use statistical techniques for handling missing values. One may also want to deduplicate categories to reduce domain size, which helps in the bias-variance tradeoff. Thus, we formalize two new data prep tasks as binary classification: **Task 1: Detect Anomalous Categories** to flag missing value categories and **Task 2: Category Deduplication** to flag pairs of categories that are duplicates. The column name and its domain are the raw features for both tasks. Task 2 is an instance of the entity matching problem in the data cleaning literature but with much less metadata for devising similarity scores; one could consider Siamese neural networks for this task.

2.4 Tasks for Usable-with-Extraction

Usable-with-Extraction columns require more processing to extract numeric and/or categorical features, e.g., `Income`. Figure 1 has "USD" prefixing a number, while `TimeSince`

has "months," "years," etc. suffixing numbers. Data scientists often write regular expressions or custom code to extract such values. While it is perhaps impossible to automate all such extractions, we identify three common tasks that can be cast as applied ML tasks.

Task 3: Embedded Feature Type Inference: What is the feature type embedded? Figure 1 shows our current taxonomy for embedded feature types. Dates and timestamps can be processed using standard DB techniques, while URLs and custom objects may require human intervention. One could consider character-level CNNs and RNNs for this task.

Task 4: Multiple Number Units Detection: Are the units of an embedded number the same? If not, we need to standardize the units, likely with human intervention and/or external knowledge bases about units. In Figure 1, `TimeSince` has multiple units. If yes, we get **Task 5: Embedded Number Extraction**: What is the embedded number? For instance, extract 1000 from "USD 1000." This can be seen as both a Seq2Seq task and a sequence-to-regression task. An encoder-decoder CNN/RNN may fit this task. One could also consider joint multi-task learning for Tasks 4 and 5.

Task 6: List Domain Extraction: Some columns have lists in a string separated by commas, space, semicolons, etc. Data scientists typically write custom code to *extract the domain* of the list values and use the domain to get new numeric/categorical features for the target model. This is a complex task that converts a sequence to a *set of sequences* representing domain entries. Once could consider more complex neural architectures for this task.

Other Featurization Routines. In our current scope, we leave other standard featurization routines for custom processing to the user. For instance, to process a full English sentence in a *Usable-with-Extraction* column, data scientists may want to use bag-of-words, n-grams, or embeddings like Word2Vec or Doc2Vec. Such feature engineering decisions are orthogonal to our focus and are often application-specific.

3 RESEARCH QUESTIONS AND OPTIONS

We now discuss a few major research questions in tackling the applied ML tasks we listed.

3.1 Metrics and Featurization

The accuracy metrics for Tasks 0 to 4 are standard, but for Tasks 5 and 6, we may need to define new metrics. For Task 5, edit distance and/or squared loss are candidates with differing results, e.g., "12" is closer to "$12.99" under the latter but not the former although edit distance helps sequence extractors. Task 6 has a complex structured prediction output, which may need a complex loss function (ideally, still differentiable) and multiple accuracy metrics. Even the featurization of the raw column is an open question, since the ML models for our tasks also need numeric, categorical, or string features. Several options exist: obtain n-grams or embeddings from column names and sample values, get summary statistics, and so on. Characterizing which of these features matter the most is also part of our research, since such featurization matters for both accuracy and inference latency at deployment time.

3.2 Creating Large Labeled Datasets

This is our central research challenge. To the best of our knowledge, there are no large benchmark labeled datasets for any of our 6 data prep tasks. So far, we have collected 360 CSV data files from Kaggle, UCI repository, etc., adding up to 9000 columns. Manual labeling for each task each could yield best accuracy but it is highly time-consuming and expensive. We plan to try 3 alternative approaches: crowdsourcing, active learning, and weak supervision.

Crowdsourcing labels is common in ML practice, but we face a major quality issue: most crowd workers are lay users, not data scientists who "get" data prep. In fact, our pilot run for crowdsourcing labels for Task 0 on the FigureEight platform resulted in too much noise even with 5 labels per example. Thus, how to structure crowd labeling questions better is an open research question. Active learning with a data scientist in the loop is another option we plan to explore. But a key disadvantage here is that we need to fix the task's ML model beforehand. Finally, weak supervision is a promising approach here, since it is often possible to write small labeling functions (LFs) to encode structural heuristics and dictionary lookups for some tasks. A denoising framework like Snorkel [6] can potentially help boost accuracy over the LFs' outputs. We also plan to try Snuba-on-Snorkel [10] to automate the production of LFs for some classification tasks. But an open challenge is that Snorkel current does not support complex prediction outputs like in Tasks 5 and 6.

3.3 Creating Human-in-the-loop Tools

Our work cannot end at getting ML models for our tasks. To complete the loop,we need to integrate them for inference in popular data prep ecosystems. There are two main kinds of tools: programmatic (e.g., R, Pandas, and TFDV) and visual (e.g., Excel and Trifacta). Each presents its own set of interesting implementation challenges. For the former, we plan to introduce simple APIs to plug our trained ML models. For the latter, it is an open research question as to how to create appropriate interface mechanisms that can exploit both our ML models' predictions and human-in-the-loop correction capabilities. For instance, the user could "guide" an ensemble of ML models based on column semantics or specific column values they see. Looking even further out, we can integrate ML models with programming-by-example

	TF-DV		Pandas		LogReg		RandForest	
	Num	N-Num	Num	N-Num	Num	N-Num	Num	N-Num
Precision	0.5117	0.9876	0.5418	0.9382	0.9331	0.9450	0.9722	0.9360
Recall	0.9915	0.4166	0.9502	0.4849	0.9093	0.9598	0.8909	0.9843
Accuracy	0.6359		0.6667		0.9394		0.9508	

Figure 2: Test Accuracy Results.

and program synthesis approaches, especially for Tasks 5 and 6 that require value extraction. This requires resolving ambiguity in the program search space and defining new ranking schemes aimed at reducing manual extraction effort.

4 CASE STUDY: ML SCHEMA INFERENCE

Data Labeling. We manually labeled 9000 columns from the data files we collected into the one of five classes of Task 0. This process took about 10 man-weeks across 4 months.

Featurization. Based on the two raw features (column name and values), we extract several hand-crafted features to train classical ML models. Our feature set is diverse: n-grams from column name, summary statistics (mean, %ge NaNs, etc.), castability as number, length of a random sample value, etc.

Experimental Setup. We perform 5-fold nested CV with a random quarter of the train fold used for hyper-parameter tuning. We compare logistic regression and RandomForest trained on our data against TF-DV and Pandas. TF-DV can infer only 2 types of features in our vocabulary: numeric or otherwise. Pandas can only infer syntactic types: int, float and object. Thus, we report the results on a binarization of our 5-class vocabulary: numeric (Num) and all non-numeric (N-Num). Figure 2 presents the test accuracy results.

Initial Results. We see a massive lift of 30% in accuracy for our approach against both TF-DV and Pandas. Interestingly, TF-DV and Pandas have high recall on numeric features but very low precision. This is because their rule-based heuristics are syntactic, leading them to wrongly classify many categorical features such as `ZipCode` as numeric. Our models have slightly lower recall on numeric features but much higher precision and overall accuracy.

5 THE ML DATA PREP ZOO

We announce the ML Data Prep Zoo, a living public repository (on GitHub) of labeled data for ML data prep tasks [1]. We will release all datasets we create as CSV files. We will also release our trained ML models in Python for the defined tasks. Our first release will be for Task 0, with the base features being the column name, summary statistics, and 5 random sample values. Our trained models will include logistic regression, RandomForest, kernel SVM, and a character-level CNN. The Zoo will also tabulate the accuracy of the baselines and our models on each task. We invite contributions from the research community to augment these datasets, create new data for the other tasks, and/or define new tasks along with their own labeled data and models. We also plan to have leaderboards for public competitions on the hosted datasets with multiple accuracy and runtime metrics, inspired by the ImageNet competition. We invite researchers to use our data to create better featurization and models to semi-automate ML data prep tasks.

6 RELATED WORK

Data Prep and Cleaning. TFDV [2] is a tool for managing ML-related data in TensorFlow Extended. It uses conservative rule-based heuristics to infer ML schema from column statistics. Our ML-based approach raises accuracy of ML schema inference substantially. DataLinter is a rule-based tool to inspect a data file and flag possible data quality issues to the user [3]. It still requires users to perform data transformations manually, which makes it orthogonal to our focus. There is growing work on reducing data cleaning effort using ML properties (e.g., [5]). Our work is part of this growing direction but our work specifically targets data prep tasks and casts them as applied ML tasks.

AutoML Platforms. Existing AutoML platforms such as Einstein [8] and AutoWeka [4] focus mainly on model selection, not ML-based ML data prep. Thus, the models we produce can enhance such platforms. OpenML [9] is an open-source platform for ML users to share and compare models, data, and analysis workflows. Our focus on *creating* high-quality labeled datasets for semi-automating ML data prep tasks is thus complementary. Our artifacts can be contributed to OpenML for spurring more research on end-to-end AutoML platforms. We could also get more analysis workflows from OpenML to enhance our work in the future.

REFERENCES

[1] Accessed March 15, 2018. The ML Data Prep Zoo Repository. https://github.com/pvn25/ML-Data-Prep-Zoo.

[2] Denis Baylor et al. 2017. Tfx: A tensorflow-based production-scale machine learning platform. In *SIGKDD*.

[3] Nick Hynes et al. 2017. The data linter: Lightweight, automated sanity checking for ml data sets. In *NIPS MLSys Workshop*.

[4] Lars Kotthoff et al. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *JMLR* (2017).

[5] Sanjay Krishnan et al. 2016. Activeclean: An interactive data cleaning framework for modern machine learning. In *SIGMOD*.

[6] Alexander Ratner et al. 2017. Snorkel: Rapid training data creation with weak supervision. *PVLDB* (2017).

[7] https://www.kaggle.com/surveys/2017. Accessed February 15, 2018. 2017 Kaggle survey on data science.

[8] https://www.salesforce.com/video/1776007. Accessed February 15, 2018. SalesForce Einstein AutoML.

[9] Joaquin Vanschoren et al. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter* (2014).

[10] Paroma Varma et al. 2018. Snuba: automating weak supervision to label training data. *PVLDB* (2018).