Towards Benchmarking Feature Type Inference for AutoML Platforms

Vraj Shah
University of California, San Diego
vps002@eng.ucsd.edu

Jonathan Lacanlale
California State University,Northridge
jonathan.lacanlale.608@my.csun.edu

Premanand Kumar
University of California, San Diego
p8kumar@eng.ucsd.edu

Kevin Yang
University of California, San Diego
khy009@eng.ucsd.edu

Arun Kumar
University of California, San Diego
arunkk@eng.ucsd.edu

ABSTRACT

The paradigm of AutoML has created an opportunity to enable ML for the masses. While many works have looked into the automated model selection or hyper-parameter search in AutoML, little work has studied how good automated data preparation (prep) is. Formalizing data prep tasks and creating benchmark labeled datasets can help objectively validate and improve AutoML platforms. In this work, we objectively benchmark a critical data prep task: *ML feature type inference*. The semantic gap between attribute types (e.g., strings, numbers) in databases/files and ML feature types (e.g., *Numeric*, *Categorical*) necessitates type inference. We formalize and standardize this task by creating the first ever benchmark labeled dataset. Our dataset has 9921 examples and a 9-class label vocabulary. Our labeled data also offers an alternative approach to automate this task than existing rule-based or syntax-based approaches: use ML itself to predict feature types. We collate a benchmark suite of 30 classification and regression tasks to assess the importance of type inference for downstream models. Empirical comparison on our labeled data shows that an ML-based approach delivers a lift of an average 14% and up to 38% in accuracy for identifying feature types compared to prominent industrial tools. Our downstream benchmark suite reveals that the ML-based approach outperforms existing industrial-strength tools for 47 out of 60 downstream models. We release our labeled dataset, models, and downstream benchmarks in a public repository with a leaderboard. They are available for download from https://adalabucsd.github.io/sortinghat.

1 INTRODUCTION

Surveys of data science practitioners show that Machine Learning (ML) over structured data is gaining popularity [38]. Several suitable data preparation (prep) steps are performed before building any ML model on such data. This makes data prep particularly challenging and laborious because it involves many diverse tasks such as inferring feature types, performing feature transformations, and cleaning feature values.

With the promise of automating the end-to-end ML workflow, including data prep, cloud vendors have released AutoML platforms such as Google's Cloud AutoML [6] and Salesforce's Einstein [8] that build ML models on millions of datasets from thousands of small-and-medium enterprises automatically. There exist open-source tools such as TransmogrifAI in Einstein [10], Tensorflow Data Validation (TFDV) in TensorFlow Extended [15], and AutoGluon from AWS [17] that automate many data prep tasks for structured data. But the effectiveness of their data prep automation is not known because there are no objective benchmarks for evaluation. Benchmark tasks and datasets for ML data prep will help advance the science of AutoML platforms by enabling answers to key questions: *How good is data prep in such AutoML tools? How can one do better? How does data prep accuracy affect downstream ML model accuracy?*

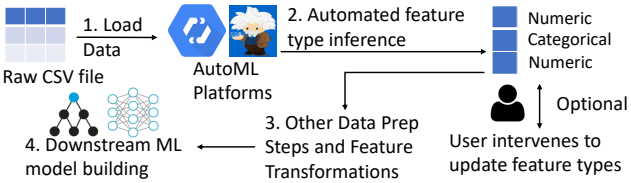


Figure 1: Typical workflow in AutoML platforms.

Our Focus. We take a step towards answering such questions by initiating work on benchmarking the accuracy of data prep steps systematically. In this paper, we focus on the very first and a critical data prep step for ML over structured data: *ML feature type inference*. We formalize and standardize this task by creating a benchmark labeled dataset. This will enable an objective progress measurement, akin to ImageNet's role in vision [35], except we go further because data prep is still an ill-defined space. Note that, data prep has many other important steps too. But it is impractical to study all steps in-depth in one paper; thus, we leave other steps to future work.

Problem: ML Feature Type Inference. Features could be *Numeric*, *Categorical*,or something else, as shown in Figure 1. Depending upon the inferred types, suitable data transformation steps may be applied, and then the downstream model is built. For instance, if a column is inferred with type *Timestamp*, then several useful features such as day, month, and year can be extracted automatically to build the downstream model. Thus, the accuracy of ML feature type inference is critical for the downstream model's accuracy because the AutoML platform can then deal with features accordingly. Even more importantly, the predictions are more *interpretable* with accurate feature types.

Importance. Our conversations with AutoML platform engineers at Google and SalesForce revealed that their tools are used on over tens of thousands of datasets, adding up to millions of features in production settings. Forcing users to manually annotate features can lead to a tedious, slow, and error-prone process that also violates the promise of end-to-end AutoML. Thus, AutoML platform engineers prefer ever more accurate automation of type inference.

CustID	Gender	Age	ZipCode	XYZ	Income	HireDate	Churn
1501	'F'	25	92092	005	'USD 15000'	'05/01/1992'	'Yes'
1704	'M'	34	78712	003	'25384'	'12/09/2008'	'No'

Figure 2: A simplified *Customers* data for churn prediction.

Challenge. We explain why feature type inference is hard to automate for existing rule-based or syntax-based systems. Datasets are typically loaded from RDBMSs, data lakes, or filesystems as flat CSV files into the AutoML platforms. Thus, there exists a *semantic gap between feature types for ML and attribute types* in databases/files. The latter tells us the syntactic datatypes of columns such as integer, real, or string. This semantic gap means reading syntax as semantics often leads to nonsensical results.

Example. Consider the dataset for a common ML task, customer churn prediction in Figure 2. We immediately see two major issues caused by the semantic gap. (1) Attributes such as *HireDate*, *Gender*, *Income*, and *Churn* are stored as strings, but not all of them are useful as *Categorical* features. For instance, *Income* is actually *Numeric* but some of its values have a string prefix. (2) Attributes such as *CustID*, *Age*, *ZipCode*, and *XYZ* are stored as integers, but only *Age* is useful as *Numeric*. *CustID* is unique for every customer, hence it can not be generalized for ML. Inspecting only the column *XYZ*, it is difficult to decide if the feature is *Numeric* or *Categorical*. *ZipCode* is *Categorical*, even though it is stored as integers. In fact, this issue is ubiquitous in real-world datasets, since categories are often encoded as integers, e.g., item code, state code, etc.

We ran TFDV on a dataset as above and found the following. It wrongly calls many *Categorical* features with integer values as *Numeric*, e.g., *ZipCode*. This can cause the downstream model to produce garbage results. Moreover, *Income* is inferred as *Categorical* even though it has numbers embedded. Such issues can lead to loss of information and can potentially reduce the accuracy of the model, or even cause it to fail in some scenarios. Thus, it is critical to objectively measure not only the accuracy of the type inference task but also its impact on the downstream model's accuracy.

Our Labeled Dataset and Label Vocabulary. Creating labeled data for the task requires a common formalized label vocabulary, which is important to create because the dichotomy of *Numeric* vs. *Categorical* is not usually enough for categorizing feature types of raw columns. For instance, column *HireDate* in Figure 2 stores *Date* type values. Thus, we need more classes. We surveyed existing AutoML data prep tools and collected them into a common and practically useful set of labels that are already widely used: ones that require some type of processing before being used as a feature, ones that are non-generalizable, and the ones that are "hard" to judge purely automatically. *We gather and hand-label the very first large meta-dataset for benchmarking feature type inference.* Our dataset has 9921 columns from 1240 real data files. Our labeling process took about 90 man-hours across 5 months.

Approaches to Type Inference. There are open-source tools such as Pandas [31], TransmogrifAI [10], TFDV [15], and AutoGluon [17] that automate this task. They all happen to be either rule-based or syntax-based. In contrast to prior approaches, our labeled dataset also presents an alternative approach to type inference: use ML itself to automate this task. We cast ML feature type inference as a multi-class classification problem and use ML models to bridge the semantic gap. We extract signals from raw data files that a typical data scientist may look at to identify the feature type. We summarize the signals in a feature set, which we use to build standard ML models on our labeled data. We empirically compare the ML-based approach enabled by our labeled data and existing public tools on our labeled *test* dataset.

Downstream Benchmark Suite. To understand the impact of the accuracy of ML feature type inference task on the downstream models, we create a *downstream benchmark*: 30 curated real-world datasets containing classification and regression tasks from diverse application domains such as healthcare, retail, sports, etc. The benchmark enables us to answer two key questions: (1) How does wrong type inference affect downstream performance? (2) How accurate are the downstream models delivered by the prior tools and the ML-based approach using our labeled data relative to performance with true feature types?

Empirical Evaluation and Analysis. An empirical comparison of different approaches on our labeled data shows that the ML-based approach delivers a lift of an average 14% and up to 38% in accuracy compared to existing tools for identifying feature types. We then evaluate and compare different ML models on our dataset. Overall, Random Forest outperforms the other models and achieves the best 9-class accuracy of 92.6%. We perform an ablation study on our ML models to characterize what types of features are useful.

Our empirical evaluation on the downstream benchmark suite shows that an ML-based approach using our labeled data delivers the most accurate downstream model against the prior tools for 47 out of 60 downstream models. In addition, we find that the wrong types inferred by existing tools often lead to a significant decrease in the downstream model's accuracy relative to their true accuracy. For instance, Pandas underperforms over truth in 45 out of 60 cases. Furthermore, we notice that the current label vocabulary of many tools is not sufficient for building an accurate downstream model. Finally, we release a repository containing our labeled dataset, trained ML models, downstream benchmarks, and announce a leaderboard for community contributions.

In summary, our work makes four key contributions.

1. Formalization of a key data prep task. We formalize and standardize the type inference task with our benchmark labeled dataset and a readily practically useful 9-class label vocabulary.

2. Utility of our labeled dataset. Using the benchmark dataset we created, we show that off-the-shelf ML models with standard featurization can outperform the industrial-strength public tools.

3. Downstream benchmark suite. The curated benchmark offers evidence that the downstream model's performance can benefit by accurately determining feature types. Moreover, we show that an ML-based approach using our labeled dataset is even valuable to produce accurate downstream models.

4. Real-world impact. Google is collaborating with us to adopt the best performing models on our labeled dataset into TDFV to improve its inference of *Categorical* type. We release a public competition on our labeled dataset to invite contributions to create/augment datasets, better featurization schemes, and models.

Fitness for EAB track: To the best of our knowledge, this is the first paper to present a benchmark dataset for the task of feature type inference. We perform an empirical comparison of existing tools and ML-based methods on our data. We also present a downstream benchmark suite and an empirical comparison of different approaches on it. Thus, we believe this paper is a good fit for EAB.

2 BACKGROUND

2.1 ML Terms and Concepts

We explain the ML terms and concepts relevant to this work intuitively and refer the interested readers to [23, 32] for a complete background. We focus on supervised ML models that require a dataset with *labeled* examples to learn their *parameters*. A trained model's prediction *error* (or *accuracy*) is measured using a *test dataset* not used for training. The test error has three components: *bias*, *variance*, and noise [37]. Bias quantifies the error occurring due to assumptions made by the model representing its *complexity*. Variance quantifies the error resulting due to changes in the training data. Thus, a simpler model with few parameters has high bias and low variance, while a complex model with large number of parameters has a higher variance but a lower bias; this is the *bias-variance trade-off*. It is used to quantify *generalization* ability of the model given by the difference between *test* and *train* error.

2.2 Assumptions and Scope

We focus on relational data, which is typically stored with schemas in RDBMSs or as "schema-light" files (CSV, JSON, etc.) on data lakes and filesystems. Either way, we assume the dataset is logically a single table with all column names available. Several data prep steps must be applied to build ML models on such data. We focus on a major step, the ML feature type inference. We leave other data prep steps to future work. Note that our focus is *not* on feature engineering over prepared data. Also, to avoid ambiguity, we call the ML model to be trained on the prepared data the "downstream model." For example, one might load a customer table to train a downstream model for predicting customer churn.

3 OUR DATASET

This section discusses our efforts in creating the labeled dataset. We discuss how we design the label vocabulary, the data sources, the signals we extract from the columns that enable us to inspect the columns succinctly, and the labelling process.

3.1 Label Vocabulary

Most ML models ultimately operate over only 2 (final) feature types: *Numeric* (continuous set) and *Categorical* (discrete set). Thus, each example (or column) has to be labelled as either of the two classes. However, we find that this bifurcation is not enough. This is because many other column types such as *Date*, *URL*, and *Primary Keys* are inevitable in the raw data file. Moreover, we find that the data file may not contain enough information to determine the feature type of a column, even for humans, e.g., column *XYZ* in Figure 2. Thus, we need more classes. We surveyed how the existing open source data prep tools such as Google's TFDV [15], TransmogrifAI in Salesforce Einstein [10], and AutoGluon from Amazon AWS [17] approach type inference and perform type-specific feature transformations. Figure 3 shows the feature type vocabulary of these tools. Inspired by this, we distill a common and practically useful set of labels for our vocabulary. We discuss the labels below.

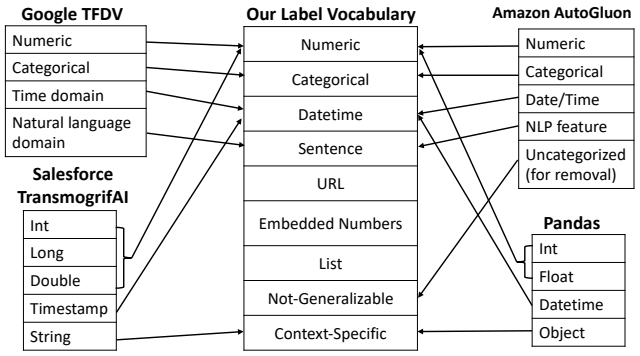


Figure 3: Feature type vocabulary mapping of TFDV, Pandas, TransmogrifAI, and AutoGluon to our vocabulary

(1) Numeric. These attributes are quantitative in nature and can directly be utilized as a *Numeric* feature for the downstream ML model. For instance, *Age* is *Numeric*, while ID attributes such as *CustID* or integers representing encodings of discrete levels are not. Note that all numbers can always be represented as categories by discretizing them. Thus, it is indeed possible to give numbers as a *Categorical* feature. However, most ML models benefit by operating on numbers directly because they can create infinite feature spaces. In contrast, a discrete set of categories is only a finite space. There is a loss of information going from numeric to a discrete category.

(2) Categorical. These attributes contain qualitative values that can directly be utilized as *Categorical* features for the downstream ML model. There are two major sub-classes: nominal and ordinal. Ordinal features have a notion of ordering among its values, while nominal do not. For instance, *Year* is ordinal, while *ZipCode* is nominal. Names and coded real-world entities from a known finite domain set are also *Categorical*. One often needs to alter the syntax of *Categorical* features for the downstream model, e.g., one-hot encoding in Scikit-learn or explicitly cast as a "factor" variable in R.

(3) Datetime. This class represents attributes containing date or timestamp values, e.g., *"7/11/2018"*, and *"21hrs:15min:3sec."* One may choose to extract custom features, either *Numeric* or *Categorical* or both through standard featurization routines. For instance, the month of the year can be *Categorical*, while time can be *Numeric*. Note that, such feature engineering decisions are not focus of this work since they are typically application-specific.

(4) Sentence. This class represents attributes containing textual values with semantic meaning. For instance, a passage of text may provide rich semantic information for a sentiment analysis application. One may choose to extract custom features, either *Numeric* or *Categorical*, or both through standard featurization routines. For instance, the AutoML platform developer can route such columns to an *n*-gram featurization routine or a routine to get Word2Vec embeddings from an English sentence for the downstream model. Again, we leave such downstream feature engineering decisions that come after type inference to the AutoML platform developer.

(5) URL. This class is for attributes whose values follow the URL standards [13]. This requires that the attribute values begin with a protocol followed by a sub-domain and a domain name. Any following information such as a file path is optional.

(6) Embedded Number. This class denotes attributes with "messy" syntax that preclude their direct use as *Numeric* or *Categorical* features. Thus, they require some form of processing before being used as as features. For instance, a number may be present along with string(s) denoting a measurement unit (*"30 Mhz" or "USD 45"*) and/or special characters (*"5,00,000"*). In all cases, a number is typically extracted and the units are standardized (if applicable). One would typically use regular expressions or custom Python/R scripts for such extraction, e.g., converting *"USD 45"* to 45.

(7) List. These attributes contain a list of items separated by a delimiter. One may write custom scripts to extract the domain of the list values and then get new features from the list domain for the downstream model.

(8) Not-Generalizable. An attribute in this class is a primary key in the table or has (almost) no informative values to be useful as a feature. Similarly, a column with only one unique value in the whole table offers no discriminative power and is thus useless. Such attributes are most unlikely to be used as features for the downstream model because they are not "generalizable." For example, *CustID* belongs to this class, since every future customer will have a new *CustID*. It is quite unlikely that one can get any useful features from it. Note that an attribute categorized as *Not-Generalizable* does not mean that it can never be useful for the downstream model. One may obtain some features from such attributes through more custom processing or domain knowledge. On the other hand, even though attributes such as *Income* and *Date* may have all unique values in their columns, they are still generalizable. Thus, they belong to *Embedded Numbers* and *Datetime* respectively since it is highly likely that one can extract useful features from them.

(9) Custom-Specific. This class is a catch-all for attributes that require human intervention either to determine their feature types and/or to inspect their values to build custom featurization routines. The following examples illustrate this class. (1) Attributes wherein the data file does not have enough information even for a human to judge its feature type. Such columns typically have meaningless names, e.g., *XYZ* in Figure 2. Judging the feature type would require manually tracing down the provenance of how this column came to be using external "data dictionaries" maintained by the application or speaking to the data creator. (2) Attributes whose values require manual inspection for extracting useful features, e.g., JSON objects, geo-locations, addresses, or other complex objects that contain information dump about the data.

Our 9-class label vocabulary, while limited, is already practically useful for AutoML platforms. The label vocabulary can also give other insights to an AutoML platform developer. For instance, they could look for tables to join when faced with a large-domain *Categorical* feature such as *ZipCode*. They could route attributes marked as *Embedded Numbers* or *Datetime* to suitable Python/R scripts. Moreover, they could dispatch the columns that are marked *Not-Generalizable* for any missing values or errors in data entry to appropriate libraries. Finally, they could prompt for user intervention on only the columns that are marked *Context-Specific*. This can reduce user time spent on annotation significantly.

3.2 Data Sources

We gather 1240 CSV data files from sources such as Kaggle and UCI ML repository. Each column of the CSV file is just one example for our task. We obtain 9921 examples from all data files. Note that we do not always use all the columns from a single data file for labeling. We explain this in Section 3.4. Kaggle and UCI ML are the largest public data sources that are closest to real-world datasets. However, we note a caveat that the files on Kaggle and UCI ML may have undergone some pre-processing. It is almost impossible for researchers to get access to large numbers of truly "in-the-wild" data from enterprises and other organizations and make them publicly available due to legal restrictions. We hope this paper starts a conversation around enhancing such benchmark datasets. We are already working with industry collaborators towards this goal of deploying the models in-the-wild. But the crux of our point in this paper is this: even on data files from Kaggle and UCI, existing open-source and industrial tools yield relatively poor accuracy compared to the ML models trained on our data (Section 5.2). Thus, we believe our work is a promising start towards objectively evaluating AutoML platforms.

3.3 Base Featurization

To identify the feature type of a raw column, a human data scientist may look at the column name, some sample values in the column, and even descriptive stats about the column. For instance, just by reading the attribute name, *ZipCode*, an interpretable string, a human can tell its feature type is *Categorical*. Thus, we represent the columns in a more concise way such that it emulates what a typical data scientist may look at to determine the feature type. We call this step Base Featurization. We extract the following base features for every column in the raw data file.

(1) Column name. We extract the column name as it can give crucial semantic clues for the feature type.

(2) Column values. A human would typically inspect some values in the column to make sure they make sense. For instance, values with decimal points are likely to mean *Numeric* features, while values with delimiters are likely lists. Thus, we extract 5 randomly sampled *distinct* attribute values from the column.

(3) Descriptive statistics. Finally, a human would look at some descriptive stats about the column. For instance, if the human finds that all values in the column are NaNs, then they might classify the column as *Not-Generalizable*. Based on this observation, we extract 25 descriptive statistics for a column such as percentage of distinct values, percentage of NaNs, mean, standard deviation, minimum value, maximum value, and an average number of whitespace-separated tokens. We provide the complete list of these 25 features in the appendix.

Each column in the raw data file is an example in the new base featurized file and we manually label every example of the base featurized file. The base featurization step also helps to deliver an ML-based approach to type inference (Section 4.3).

3.4 Labelling Process

We first use base featurized columns from 360 source files to label them in one of the nine classes. But, we find that they only contain a small handful of examples for the classes: *URL*, *List*, *Sentence*, *Embedded Number*, and *Datetime*. Thus, we use an additional 880 source data files to only label the examples for the under-represented classes. In addition, we find that many data files have a series of column names such as *xyz1*, *xyz2*, and so on. Thus, we drop the columns with repeating series of names.

To reduce the cognitive load of labelling, we follow the following process. Initially, we manually label 500 examples. We then use Random Forest with 100 estimators to perform 5-fold nested cross-validation (CV). The model achieves a classification accuracy of around 74% on the test set (average across 5 folds). We use this model to predict a class label on all of the 9921 examples. We then group all the examples by these predicted labels and inspect all of them manually. Such grouping helps reduce the cognitive load caused by class context switches during labeling. *The labeling process took about 90 man-hours across 5 months.*

We also tried to crowdsource labels on the FigureEight platform but abandoned this effort because the label quality was too low across two trial runs. We suspect high noise arises because this task is too technically nuanced for lay crowd workers relative to popular crowdsourcing tasks like image recognition. Devising better crowdsourcing schemes for our task with lower label noise is an avenue for future work. We summarize the results of our crowdsourcing effort in the appendix.

3.5 Data Statistics

The distribution of class labels in our labeled dataset is: *Numeric* (36.6%), *Categorical* (23.3%), *Datetime* (7%), *Sentence* (3.9%), *URL* (1.5%), *Embedded Number* (5.7%), *List* (2.4%), *Not-Generalizable* (10.6%), and *Custom-Specific* (8.9%). We provide a complete breakdown of the cumulative distribution by class for different descriptive stats in the appendix.

4 APPROACHES COMPARED

In this section, we discuss the different approaches to type inference. We first discuss existing open-source tools that all happen to be either rule-based or syntax-based. We then briefly discuss an intuitive rule-based baseline to check if a set of rules can accurately capture our labeled dataset. Finally, we explain how our labeled dataset is used to build ML models.

4.1 Existing Tools

Figure 3 shows the feature type vocabulary of these tools and how they map to our label vocabulary.

Tensorflow Data Validation (TFDV). TFDV is a tool to analyze and transform ML data in TensorFlow Extended (TFX) pipeline [15]. TFDV uses conservative heuristics to infer ML feature types such as numeric, categorical, time or date domain, or natural language text from the descriptive statistics about the column. The users can then review the inferred feature types and can update them manually to capture any domain knowledge about the data that the heuristics might have missed.

Pandas. Pandas is a Python library that provides tools for data analysis and data transformations. It infers syntactic types such as integer, float, or object [31]. It also provides a utility function that can check the column for the datetime type.

TransmogrifAI. This is an AutoML library for structured data in Salesforce's AutoML platform called Einstein [10]. TransmogrifAI supports rudimentary automatic feature type inference over primitive types such as Integer, Long, Double, Timestamp, and String. It also has an extensive vocabulary for feature types such as email, phone numbers, zipcodes, etc. However, users have to manually specify these feature types for their data.

AutoGluon-Tabular. AutoGluon is an end-to-end AutoML framework from AWS [17]. It classifies each column into numeric, categorical, date/time, text, or columns that needs to be discarded because they can't be classified into any of the classes.

4.2 Rule-based Baseline

We develop a rule-based baseline approach to validate if a set of rules can accurately represent our labeled dataset. We write 11 rules to capture all the classes using a flowchart-like structure. As an example, if % of NaN values in the column are greater than 99.99% then we mark it as *Not-Generalizable*. We describe our rule-based approach in-depth in the appendix. We find that it is highly cumbersome and perhaps even infeasible to hand-craft a perfect rule-based classifier.

4.3 ML-based Approach using our Data

As shown in Figure 4, we use our labeled data to build standard ML models. Base Featurization is a common step for all ML models. Some ML models cannot operate on the raw characters of attribute names or sample values. Thus, we extract hand-crafted feature sets from the attribute names and sample values. We then train several classical ML models, k-NN with a distance function tuned for our task, and a CNN. Finally, the pre-trained model is used to infer feature types for columns in an "unseen" CSV file. At the scale of AutoML platforms where there are potentially millions of columns, human intervention can be costly and slow. The models output predictions and the corresponding confidence scores for each class. Thus, an ML-based approach allows users to intervene to prioritize their effort towards *Context-Specific* types or columns with low confidence scores that may need more human attention.

4.3.1 Feature Extraction. The attributes with similar names can likely belong to the same class. For instance, both attributes *temperature_jan* and *temperature_feb* are *Numeric*. Similarly, knowing that the sequence of characters are numbers followed by a "/," can give an indication of *Datetime*. Based on these intuitions, we extract an n-gram feature set from the attribute names and sample values.

Notation. We denote the descriptive stats by $\mathbf{X}_{stats}$, the attribute name by $\mathbf{X}_{name}$, and the randomly sampled attribute values by $\mathbf{X}_{sample}$ (first sampled value is referred to as $\mathbf{X}_{sample1}$ and similarly for other values). We leverage the commonly used bigram feature set on the attribute name (denoted by $\mathbf{X2}_{name}$) and sample value (denoted by $\mathbf{X2}_{sample}$).

4.3.2 Classical ML models. We consider classical models: Logistic Regression, RBF-SVM, and Random Forest. Note that they cannot operate on raw characters of attribute names or sample values.

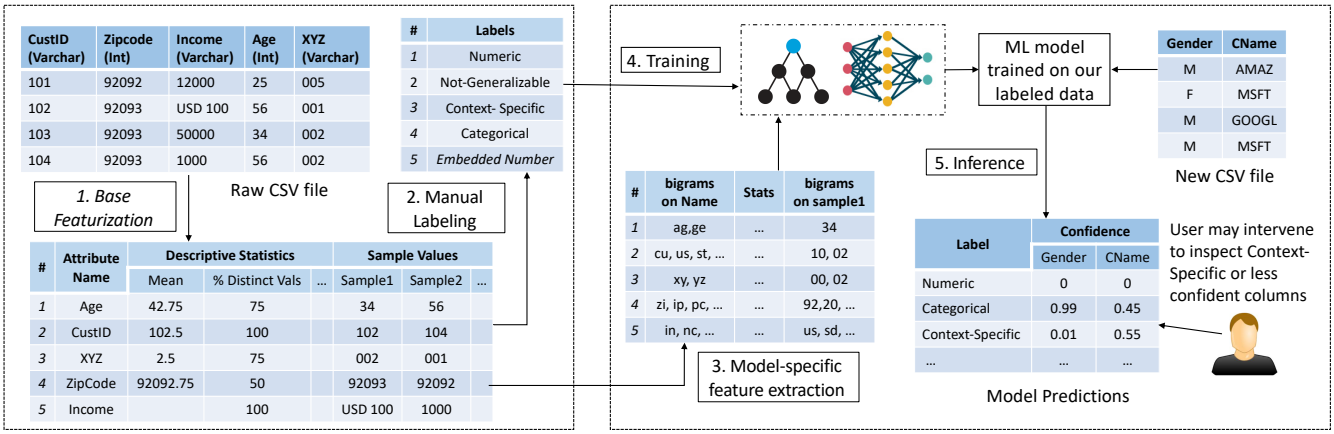


Figure 4: Workflow showing our labeling process and how our data is used for ML-based feature type inference.

Thus, we use features: $\mathbf{X}_{stats}$, $\mathbf{X2}_{name}$, $\mathbf{X2}_{sample1}$, and $\mathbf{X2}_{sample2}$. For scale-sensitive models such as RBF-SVM and logistic regression, we standardize $\mathbf{X}_{stats}$ to have mean 0 and standard deviation 1.

4.3.3 Nearest Neighbor. Most implementations of k-NN use a simple Euclidean distance. But, we can adapt the distance function for the task at hand to do better. Thus, we define the weighted distance function as:

$$d = ED(X_{name}) + \gamma \cdot EC(X_{stats})$$

Here, ED (resp. EC) is the edit distance (resp. euclidean distance) between X_{name} (resp. X_{stats}) of a test example and a training example. γ is the parameter that needs to be tuned during training.

4.3.4 CNN. Inspired by the success of CNN on short text classification tasks [43, 44], we leverage a character-level CNN for our task. We present the architecture and layers of CNN in the appendix. The network takes attribute name, descriptive stats, and sample values as input and outputs the class from the label vocabulary. The attribute name and descriptive stats are fed into the CNN module consisting of 3 cascading layers: 2 1-D CNN and a global max-pooling layer. We concatenate all CNN modules with descriptive stats and feed them to an MLP. The whole network can be trained end-to-end using backpropagation.

5 EMPIRICAL STUDY AND ANALYSIS

We now empirically compare the industrial open source tools and ML models on the accuracy of type inference. This is the *very first empirical comparison* of this sort of these tools, thanks to our new benchmark labeled dataset. The headline result is that our ML models substantially surpass these prior tools on test accuracy.

5.1 Methodology and Setup

Methodolody. We partition our labeled dataset into a train and held-out test set with 80:20 ratio. We perform 5-fold nested cross-validation of the train set, with a random fourth of the examples in a training fold being used for validation during hyper-parameter tuning. We use a standard grid search for hyper-parameter tuning. The grids are described in the appendix. We also did a 5-fold leave-data file out cross-validation to "stress-test" the ML models for new data files. The raw data files were split into 60:20:20 train, validation, and test partitions where each partition has all columns of a particular data file. Thus, the test partition has columns of the raw data files that the model has not seen before. The trends of the leave-data file out approach are similar to the former approach; so, we defer its details to the appendix.

Experimental Setup. We use CloudLab [16] with custom OpenStack profile running Ubuntu 18.04 with 10 Intel Xeon cores and 192GB of RAM. For TFDV, Transmogrifai, AutoGluon, and Pandas, we use version number 0.22.2, 0.7.0, 0.0.11, and 0.25.3 respectively.

Metrics. Our key metric is prediction accuracy, both for the 9-class task and class-specific binarizations. We also report full confusion matrices in the appendix.

5.2 Comparison of All Approaches

We compare ML models trained on our dataset against the open-source tools on our labeled data. Figure 3 showed the feature type vocabulary of these tools and how they map to our label vocabulary. Since none of these tools support our full 9-class vocabulary, we report the results on binarization of our vocabulary: *Numeric* vs. all Non-Numeric, *Categorical* vs. all Non-Categorical, and similarly for other classes.

Results. Table 1 presents the precision, recall, and overall 2 x 2 diagonal accuracy results of all tools on our benchmark labeled test set. We also report the performance of the models trained on our benchmark labeled train set. We present the results in-depth below.

(1) We see that the ML models achieve significantly higher accuracy than all industrial tools across the board for all feature types. For instance, we see a lift of 28% and 14% in accuracy in predicting *Categorical* compared to TFDV and AutoGluon respectively. Of all approaches, Random Forest achieves the best accuracy in inferring the feature types.

(2) Interestingly, all the existing tools have a high recall on *Numeric* but very low precision. This is because their heuristics are syntactic, which leads them to wrongly classify many *Categorical* features such as *ZipCode* as *Numeric*. The ML models have a slightly lower recall on *Numeric*. This is because with many features thrown, they get slightly confused and could wrongly predict a *Numeric* type as non-numeric. But, the ML models have much higher precision and high overall accuracy.

Feature Type	Metric	Open-source Industrial Tools				Baseline	Models trained on our data		
		TFDV	Pandas	TransmogrifAI	AutoGluon	Rule-based	Log Reg	CNN	Rand Forest
Numeric	Precision	0.657	0.614	0.605	0.646	0.773	0.909	0.929	0.934
	Recall	1	1	1	1	0.946	0.943	0.941	0.984
	Accuracy	0.814	0.776	0.767	0.805	0.882	0.946	0.953	**0.97**
Categorical	Precision	0.396	-	-	0.667	0.577	0.808	0.846	0.913
	Recall	0.652			0.534	0.457	0.884	0.928	0.943
	Accuracy	0.691			0.831	0.798	0.925	0.945	**0.966**
Datetime	Precision	0.985	0.956	1	1	0.559	0.951	0.925	0.945
	Recall	0.475	0.915	0.454	0.844	0.135	0.972	0.965	0.972
	Accuracy	0.962	0.991	0.961	0.989	0.931	0.994	0.992	**0.994**
Sentence	Precision	0.472	-	-	0.516	1	0.913	0.725	0.865
	Recall	0.457			0.902	0.043	0.793	0.804	0.902
	Accuracy	0.951			0.956	0.956	0.987	0.977	**0.989**
Not-Generalizable	Precision	-	-	-	0.465	0.216	0.732	0.81	0.934
	Recall				0.53	0.507	0.732	0.66	0.86
	Accuracy				0.883	0.747	0.947	0.937	**0.978**
Context-Specific	Precision	-	0.08	0.074	-	0.211	0.747	0.741	0.859
	Recall		0.295	0.295		0.195	0.621	0.663	0.705
	Accuracy		0.609	0.582		0.853	0.944	0.946	**0.961**

Table 1: Binarized class-specific accuracy of different approaches on our benchmark labeled held-out test dataset. The bold fonts highlight the most accurate approach/model per class.

(3) Heuristics for identifying *Datetime* by all the existing tools have high precision, even higher than the ML models. However, their rules do not capture many *Datetime* type instances (e.g., an attribute named *BirthDate* "19980112"); thus, they have a much lower recall.

(4) The heuristic rules of AutoGluon and TFDV are largely dependent upon the number of words in a string for accurately inferring *Sentence* type. Thus, a column with most of its values having a large number of words will likely get inferred as *Sentence* by these tools. However, a *Categorical* or *Context-Specific* column (e.g., containing JSON object) can satisfy the criteria provided by the rules. Thus, AutoGluon and TFDV have low precision on *Sentence*. On the other hand, the ML-based approaches have much higher precision.

Other Commercial Tools. There exist other commercial tools that also automate the ML feature type inference task such as Google AutoML Tables [5], DataRobot [2], and Trifacta [11]. However, since these systems are closed source, we do not know how these tools work. It is also hard to evaluate their accuracy because: (1) DataRobot has no public/free trial version of their platform. We got no response to our demo request. (2) AutoML Tables and Trifacta only offer GUI-based usage where users must upload the raw CSV files manually to identify the feature types. Both these tools do not provide any programmatic way for evaluation. So, we cannot evaluate their accuracy automatically. We manually uploaded 5 CSV files from our raw data. All 15 categoricals encoded as integers were (wrongly) classified as numeric by both tools. So, they will likely have the same issues as TFDV, AutoGluon, and TransmogrifAI.

5.3 Comparison of ML-based Approaches

Rule-based Baseline. The 9-class classification accuracy on the held-out test set is only *0.54*. We observe that this approach achieves 95% and 46% recall in classifying *Numeric* and *Categorical* respectively. The recall for *Categorical* is low because a number encoded as a category is wrongly classified as *Numeric*. Admittedly, our rules are not exhaustive and one can always come up with more rules to improve the accuracy. However, writing rules for every little corner case is excruciating and will likely never be comprehensive. We present the confusion matrix in the appendix.

Classical ML Models. Table 2 presents the 9-class accuracy results of the classical ML models using different feature sets. We present the 5-fold held-out train and validation accuracy in the appendix. For logistic regression, we see that the descriptive stats alone are not enough, as it achieves an accuracy of just 69% on the held-out test set. But, for RBF-SVM and Random Forest, the accuracy with stats alone is already 82% and 91% respectively. Incorporating 2-gram features of the attribute name into logistic regression leads to a whopping 15% lift in accuracy. Random Forest achieves an impressive 93% accuracy using just 2-gram feature set along with descriptive stats.

	X_{stats}	X^*_{name}	$X^*_{sample1}$	X_{stats}, X^*_{name}	$X_{stats}, X^*_{sample1}$	$X^*_{name}, X^*_{sample1}$	$X^*_{sample1}, X^*_{sample2}$	$X_{stats}, X^*_{name}, X^*_{sample1}$	$X_{stats}, X^*_{name}, X^*_{sample1}, X^*_{sample2}$
Logistic Regression	0.6862	0.7293	0.6603	0.8428	0.7763	0.8043	0.7144	0.8578	**0.8643**
RBF-SVM	0.8213	0.777	0.6521	0.8724	0.7845	0.8159	0.7131	**0.8761**	0.8712
Random Forest	0.9121	0.7785	0.6657	**0.9259**	0.8956	0.8346	0.7374	0.9216	0.9096
CNN	0.6809	0.8019	0.6805	0.8692	0.7965	0.8655	0.7763	**0.8788**	0.8701
k-NN	0.8605	0.7839	-	**0.8796**	-	-	-	-	-

Table 2: Full 9-class test accuracy of the ML models trained on our data with different feature sets. $X^*_{name}, X^*_{sample1}, X^*_{sample2}$ denote bigram features ($X2_{name}, X2_{sample1}, X2_{sample2}$) for classical ML models and raw character-level features ($X_{name}, X_{sample1}, X_{sample2}$) for CNN and k-NN. The bold fonts highlight the most accurate feature set for that model.

Adding bigram features of a random sample value lifts the accuracy further by 2% for logistic regression. However, as the complexity of the model increases, the bigrams on a sample value do not provide a boost in accuracy. Also, adding more sample values does not give any rise in accuracy, except for logistic regression. Overall, Random Forest achieves the best 9-class accuracy of 93% using bigrams on the attribute name along with descriptive statistics.

CNN. Table 2 also shows the CNN accuracy. We see that with just X_{name}, the accuracy is already 82%. The descriptive stats lift the accuracy further by 8%. We find that sample values are not that useful, yielding only a minor lift.

Nearest Neighbor. We observe that with only Euclidean distance on descriptive statistics, the accuracy is already 86% on the held-out test set. With only edit distance on attribute name, the accuracy is 78%. Finally, with our weighted edit distance function from Section 4.4, k-NN achieves a high 88% accuracy.

5.4 Analysis of Errors

We now explain the behavior of the best performing Random Forest on our held-out test dataset (shortened henceforth as "OurRF") by inspecting the raw datatype of the column values. We categorize the data type into integers, floats, negative numbers, strings, strings with one token, and strings with more than one space-separated token. Table 3 shows the confusion matrix of the predicted class by OurRF vs actual data type of the attribute value. Table 4 shows examples of columns and the corresponding prediction made by OurRF. We intuitively explain the errors by class below.

Numeric. We see that when the actual label is *Numeric* (Table 3 (A1)), OurRF is less likely to misclassify an attribute whose values are floats or negative numbers compared to integers. We observe that with integers, OurRF gets most confused with *Context-Specific* class, e.g., *s1p1c2area* (Table 4 example(A)). This is possibly because of the non-sensical attribute name.

Categorical. As shown in Table 3 (A2), when the sample values are strings with number of tokens greater than 1, there is more chance for OurRF to misclassify *Categorical* as *Sentence* or *Context-Specific* (Table 4 example(B)). In contrast, for one-token strings, OurRF is more accurate in predicting *Categorical*.

Not-Generalizable. From Table 3(A8), we notice that OurRF often confuse *Not-Generalizable* with *Categorical*. For instance, *q19TalToolResumeScreen* (Table 4 example(G)) is *Not-Generalizable*, because its domain contains only 2 values: "NULL!" and "Resume screening." However, OurRF treats "NULL!" as a separate category. Thus, OurRF is lacking in its semantic understanding ability of sample values.

Context-Specific. We find that our model performs worst in accurately inferring this type, with a recall of just 71%. Integers are most commonly misclassified with *Numeric* (Table 4 example(H)). Again, it seems that OurRF is missing the human-level intuition of accurately identifying the attributes with meaningless names.

Other types. We find that our model achieves high precision and recall in inferring other types such as *Datetime* and *URL*. In addition, *List* types are often confused with *Embedded Number* (Table 4 example(C)) even though there is no number available for extraction. This can be due to few available training examples for *List* type.

5.5 Prediction Runtimes

We evaluate the running time of ML models in the online phase, i.e, to make predictions on a new column. This involves base featurization, model-specific feature extraction (only needed for the classical models), and inference time. The measurements were made on the test set and averaged. All the models finish in under 0.2 sec per column. For the classical models, the additional feature extraction dominates overall runtime. Since SVM and k-NN are distance-based methods, they have the highest runtime. Overall, CNN is the fastest on prediction. We present the time breakdown in the appendix.

6 DOWNSTREAM BENCHMARK SUITE

To complete the loop on type inference, we now empirically study if doing feature type inference accurately is essential for downstream model accuracy. Thus, we verify if there are cases where doing wrong type inference may improve, reduce, or match the downstream accuracy relative to true feature types. From Section 5.3, we saw that type inference accuracy is highest for the Random Forest (OurRF) among all ML-based approaches. Thus, we compare the OurRF against the industrial and open source tools on a suite of downstream tasks we collected and curated.

6.1 Datasets

The impact of type inference is dependent on the dataset and the downstream prediction task. Since there are unboundedly many

	(A1) Numeric			(A2) Categorical			(A3) DT	(A4) ST	(A5) URL	(A6) EN	(A7) List	(A8) Not-Generalizable		(A9) Context-Specific		
	Integer	Float	Negative Number	Number	String (> 1 token)	String (1 token)	String	String	String	String	String	Number	String	Number	String	**Precision**
Numeric	369	327	212	12								1	2	34		**93.4**
Categorical	2	1		209	99	125	2	3	2	5	1	3	10	8	4	**91.3**
Datetime							137			1			6	1		**94.5**
Sentence					4			83			3		4		2	**86.5**
URL									30				1			**96.8**
Embedded Number							2			92	5					**92.9**
List											43					**100**
Not - Generalizable	1	1				1		3				91	93	1	6	**93.4**
Context-Specific	6			5	4			3		1		3		109	21	**85.9**
Recall	**97.6**	**99.4**	**100**	**92.5**	**92.5**	**99.2**	**97.2**	**90.2**	**93.8**	**92.9**	**82.7**	**92.9**	**83.8**	**71.2**	**63.6**	

Table 3: Breakdown of Random Forest's prediction for different types of column values on the held-out test set. Confusion matrices (Predicted class on the row vs. Actual type of the attribute on the column) when the ground-truth label is (A1) *Numeric*: *NU*, (A2) *Categorical*: *CA*, (A3) *Datetime*: *DT*, (A4) *Sentence*: *ST*, (A5) *Embedded Number*: *EN*, (A6) *URL*, (A7) *List*: *LST*, (A8) *Not-Generalizable*: *NG*, and (A9) *Context-Specific*: *CS*. Blank cells denote zero predictions.

#	Attribute Name	Sample Value	Total Values	% Distinct Values	% NaNs	Label	RF Prediction
A	s1p1c2area	50	9597	3.6	45.2	*NU*	*CS*
B	Tenure Status	Own house, rent lot	41544	0.02	0	*CA*	*ST*
C	End	March 4, 1797	45	97.8	2.2	*DT*	*EN*
D	Name	Battle of Riverrun	38	100	0	*ST*	*NG*
E	%White	18.90%	192	58.9	0	*EN*	*CA*
F	Countries	ru; uk; mx	1359	32.9	46.3	*LST*	*EN*
G	q19TalToolResumeScreen	#NULL!	25090	0.008	6	*NG*	*CA*
H	Livshrmd	151	9597	1.17	42.3	*CS*	*NU*
I	Name	Technic	614	65.5	0	*CS*	*NG*

Table 4: Examples of errors made by RandomForest.

datasets and downstream tasks, for the sake of tractability we got 30 "unseen" datasets from Kaggle, UCI ML repository, and OpenML [40] for evaluation. Since classification tasks are more common in practice, we got 25 datasets for such tasks, and 5 for regression tasks. We ensure representation of various combinations of feature types with many different data types (*ints*, *floats*, *string*, *dates*, *timestamps*, and even *primary keys*). We did not cherry-pick a dataset to particularly suit one approach over another. Overall, we have 566 columns across 30 downstream datasets. We manually label all the columns with their true feature type. The datasets and their source details are available on our Github repo [4].

6.2 Models and Metrics

In terms of downstream model evaluation, we present both extremes of bias-variance tradeoff [19]: L2-regularized Logistic regression (high bias, low variance) for classification, L2-regularized Linear regression (high bias, low variance) for regression, and Random Forest (low bias, high variance) for both classification and regression. Thus, we have 60 downstream models in total. We use the accuracy metric scaled to 100 for the classification tasks and root mean squared error (RMSE) metric for the regression tasks.

6.3 Tools compared

We compare Pandas, TFDV, AutoGluon, and OurRF, relative to the truth on 30 downstream datasets. We map the feature types inferred by these tools to our label vocabulary as per Figure 3. Columns that are inferred *Numeric* are retained as is, *Categorical* columns are one-hot encoded, *Sentence* columns are routed through TF-IDF [33] and character-level bigrams, *URL*s are specially processed through a word-level bigrams, *Not-Generalizable* columns are dropped, and the rest of the types are featurized with bigrams. After featurization, we use the same methodology as Section 5.1 for evaluation. Note that, one can plug-in any alternate featurization scheme to derive more useful features. However, such feature engineering decisions can be application-specific and are not the focus of this work.

6.4 Results

6.4.1 Type Inference Results. Table 6 (A) shows the type inference accuracy of all tools on the downstream datasets. We see that OurRF can correctly infer the feature types for 516 out of 566 columns in these 30 datasets. Pandas has a seemingly high accuracy of 90% but note the low coverage of columns by its vocabulary, which makes it benefit from high recall. It cannot predict on the other columns at all. The accuracy of TFDV and AutoGluon is much lower than OurRF; their coverage is also slightly lower than OurRF.

6.4.2 Downstream Model Performance. Table 5 presents the massive end-to-end comparison of downstream models built with feature types inferred by Pandas, AutoGluon, TFDV, and OurRF relative to the true feature types. Table 6 (B) offers summary statistics on how the tools perform relative to the ground truth and other tools. We find that, for a given dataset and a downstream model, OurRF performs worse than the best performing tool for only 13 out of 60 downstream models. Moreover, relative to the truth, OurRF underperforms for only 20 downstream models. In contrast, Pandas, TFDV, and AutoGluon underperform for significantly more models: 44, 35, and 35 respectively. We explain the results in-depth below.

(A) Feature Types	Raw Attribute Types	Dataset	|A|	|Y|	Logistic Regression					Random Forest				
					Truth	PD	TFDV	AGL	OurRF	Truth	PD	TFDV	AGL	OurRF
NU	Int, Float	Cancer	9	2	60.8	+0	+0	+0	+0	66.7	+0	+0	+0	+0
	Int	Mfeat	216	10	92.5	+0	+0	+0	-2.7	91.8	+0	+0	+0	-2.3
CA	String	Nursery	8	5	92.8	-0.9	+0	+0	+0	98.2	-3.9	+0	+0	+0
	String	Audiology	69	24	73	-1.3	+0	-1.3	+0	72.2	-0.9	+0	-1.3	+0
	Int	Hayes	4	3	74.1	-14.1	-14.1	-14.1	+0	78.5	-14.1	-14.1	-14.1	+0
	Int	Supreme	7	2	99.3	-14.5	-17.1	-14.5	+0	99.4	+0	+0	+0	+0
	Int, String	Flares	10	2	90.8	+0	+0	+0	+0	89.2	+0.3	+0.3	+0.3	+0
	Int, String	Kropt	6	18	39.4	-6.9	-6.9	-6.9	+0	68.8	-3.4	-3.4	-3.4	+0
	Int, String	Boxing	3	2	80.7	-24.4	-25.2	-25.2	-34.1	78.5	-17	-11.9	-11.9	-28.9
NU + CA	Int, String	Flags	28	2	68.2	-6.2	-3.6	-6.7	-4.1*	75.9	-1	-2.6	-2.6	-3.1*
	Int,Float,String	Diggle	8	2	99.9	+0	+0	+0	-5.8	99.9	+0	+0	+0	+0
	Int, Float	Hearts	13	2	84.9	-0.7	-1.6	-0.7	+0	86.2	-1.3	-3	-1.3	+0
	Int, Float	Sleuth	10	2	68.9	-3.3	-3.3	-3.3	+0	76.7	+0	+0	+0	+0
CA + NG	Int, String	Apnea2	3	2	92	-6.7	-0.6	-0.6	-0.6	90.1	-2.3	-0.8	-0.8	-0.8
NU + CA + ST	Int, String	Auto-MPG	8	3	89.1	-4.8	-8.6	-8.6	-15.9	95.2	+0.5	-18.9	-18.9	-20.5
NU + CA + EN	Int,Float,String	Churn	19	2	79.1	-0.7	+0.1	-0.1	+0.2	78.7	-0.2	-0.9	-0.8	-0.3
NU + DT + EN	Int, Float, String, Date	NYC	6	15	55.8	+0	-0.1	-0.3	-0.3	67.6	+0	+0.5	+0.8	+0.8
ST	String	BBC	1	5	97.1	-6.9	+0	+0	+0	96.3	-13.1	+0	+0	+0
DT + ST	String, Date	Articles	3	2	98.8	-2.1	+0	+0	+0	99.0	-3.2	+0	+0	+0
NU+CA+ST+NG	Int,String,PK	Clothing	10	5	66.7	-9.2	-9.1	-9.2	+0	64.2	-2.2	-4.9	-2.6	+0
NU + DT + NG	Int, String, Time, PK	IOT	4	2	83.8	-0.3	+0	+0	+3.6*	93.8	-1.4	+0	+0	+0*
NG + CA	Int,String, PK	Zoo	17	5	75.6	-13.4	-11.1	-8.9	-2.2	77.8	-15.6	-8.9	-6.7	-4.4
NU+CA+EN+NG	Int,Float,String	PBCseq	18	2	68.6	-1.3	+0.5	+0.5	+6.2*	73	-1.2	-0.1	-0.1	+2.2*
NU + CA + LST + NG + CS	Int, Float, String, PK	Pokemon	40	36	65.84	-52.2	-52.4	-52.6	-0.6	88.1	-3.9	-3.2	+0	+0
NU + CA + DT + URL + NG + CS	Int,Float,Date, String, Time	President	26	57	39.5	-7.9	-7.9	-8	-0.9	81.7	-29.4	-23.1	-28.8	-2.1

(B) Feature Types	Raw Attribute Types	Dataset	|A|	Linear Regression – L2 Regularization					Random Forest				
				Truth	PD	TFDV	AGL	OurRF	Truth	PD	TFDV	AGL	OurRF
CA	Int	MBA	2	0.363	+0.05	+0.05	+0.05	-0	0.384	+0.09	+0.08	+0.09	-0
NU + CA	Int	Vineyard	3	2.97	+2	+2	+2	-0	2.7	+0.37	+0.37	+0.37	-0
	Int, String	Apnea	3	2206.2	+62.5	-0	-0	-0	1355.7	+1972.7	-0	-0	-0
DT	Date	Accident	1	466	-0	+384.6	-0	-0	589.7	-0	+474.8	-0	-0
NU + CA + EN + NG	Int, String	Car Fuel	11	11.3	-0.09	+0.16	+0.14	+0.01*	11.7	+0.33	+1.1	+0.9	+0.03*

Table 5: Accuracy comparison of downstream models using inferred types from Random Forest trained on our labeled data (OurRF) against Pandas (PD), TFDV, and AutoGluon (AGL), relative to accuracy with true feature types. Datasets involve (A) Classification tasks with accuracy metric (B) Regression tasks with RMSE metric. *Numeric*(*NU*), *Categorical*(*CA*), *Datetime*(*DT*), *Sentence*(*ST*),*Not-Generalizable* (*NG*), *Embedded Number* (*EN*), *URL*, *List* (*LST*), and *Context-Specific* (*CS*) are feature types. |A| is the number of columns/attributes in that dataset. |Y| is the number of target classes. PK denote primary keys. * denotes the cases where OurRF prediction is either *EN* or *CS*, where user intervention can help improve model accuracy or generalization.

(A)	Pandas	TFDV	AutoGluon	OurRF
Column Coverage	300	535	553	566
Type inference accuracy given coverage	90.3%	75%	71.4%	91.2%

(B)	Logistic Regression				Random Forest			
	PD	TFDV	AGL	OurRF	PD	TFDV	AGL	OurRF
Underperform truth	23	18	19	11	21	17	16	9
Match truth	6	10	10	16	7	11	12	19
Outperform truth	1	2	1	3	2	2	2	2
Best performing tool for a dataset	9	11	10	23	10	14	16	24

Table 6: (A) Type Inference accuracy on downstream datasets. (B) Number of downstream datasets where tools underperform, match, or outperform the ground truth downstream performance or the best performing tool. OurRF is the Random Forest for type inference trained on our data. LR denotes downstream linear model (Logistic/Linear regression) and RF denotes downstream Random Forest.

1. Why does wrong type inference hurt downstream accuracy?
Table 5 shows that wrong type inference almost always leads to a drop in accuracy compared to the accuracy with true feature types. Moreover, the amount of drop depends upon how many feature types are wrongly classified and how predictive those features are for the target. For instance, wrong type inference leads AutoGluon and TFDV to underperform on 35 out of 60 downstream models. This led to a reduction of an average 7% and up to 52% in accuracy compared to the ground truth-based model. Due to space constraints, we only explain the two most common patterns of how wrong type inference affected downstream accuracy below.

(a) *MFeat* has 216 *Numeric* integer columns, presenting a best-case scenario for prior tools as they have the highest possible recall in inferring *Numeric*. Thus, they classify all columns correctly. However, OurRF confuses 7 of them with *Categorical*, possibly because of their low domain sizes, thus leading to a drop in accuracy. We verified that this is the primary reason why OurRF underperforms truth and prior tools on datasets like *Auto-MPG* and *Diggle*.

(b) On *Zoo*, out of 4 *Not-Generalizable* columns, 1 column is erroneously predicted as *Categorical* by OurRF. Thus, using a feature that offers no discriminative power leads to a drop in accuracy compared to the ground truth. On the other hand, AutoGluon classifies all 4 of them incorrectly. Other tools like TFDV and Pandas do not even support *Not-Generalizable* in their vocabulary. Thus, the drop in accuracy is much larger for the prior tools. This underscores the importance of identifying *Not-Generalizable* columns correctly. We observe the same pattern across many datasets like *Pokemon*, *President*, and *Car Fuel*.

2. Why does wrong type inference of *integer Categorical* often not hurt downstream Random Forest?
Although the categories encoded as integers in *Supreme*, *Flags*, *Sleuth*, and *Vineyard* are misclassified by Pandas, AutoGluon, and TFDV, the accuracy of Random Forest either does not drop or drops only marginally. This is because the *Categorical* features in these datasets are either ordinal and/or have binary domain size. Random Forest has zero bias and thus can potentially represent all categories by doing splits on integers. Linear models, which have lower VC-dimension, cannot do this. Thus, the linear models often see much higher accuracy with OurRF than prior tools.

3. How can OurRF exploit user intervention to help raise accuracy?
(a) On *Flags*, a *Categorical* feature was erroneously predicted as *Context-Specific* type by OurRF. If a human intervenes to inspect such columns, then the accuracy of the downstream model can be further improved.

(b) *Car Fuel* has two *Embedded Number* columns. Although they are predicted correctly by OurRF, a human can intervene to extract their values to use them as *Numeric* instead of the current bigramization in our benchmark. Thus, a user-in-the-loop approach can further boost downstream model accuracy.

4. Why is outperforming truth not necessarily beneficial?
(a) A *Not-Generalizable* unique identifier column denoting the "case number" on *PBCseq* is predicted as *Numeric* by OurRF. Even though we notice a significant lift in accuracy compared to the ground truth, this is not necessarily beneficial in the deployment setting, where every newly conducted study will have a new case number. Thus, it is very unlikely that the downstream model will generalize.

(b) The other dataset where we observe a significant lift in accuracy relative to the truth is *IOT*, where a *Numeric* column called "temp" (denoting temperature) is classified as *Context-Specific*. This again may not be desired because interpretability can be a concern in this application. Predictions are more explainable when using temperature data as *Numeric* feature than bigrams. Thus, a human-in-the-loop approach for handling *Context-Specific* prediction can help resolve this issue.

6.5 Summary

Overall, we find that OurRF achieves a high accuracy of 91.2% for inferring feature types on 30 unseen datasets from Kaggle, UCI ML repo, and OpenML. Moreover, we find that wrong feature type inference almost always leads to an accuracy drop for the downstream model relative to the ground truth, except for the Random Forest on ordinal and/or binary domain *Categorical*. More importantly, our labeled dataset is valuable to build an accurate downstream model because even standard ML models like Random Forest trained on our labeled data achieves the highest accuracy against existing tools for 47 out of 60 downstream models. In addition, with some human intervention for *Context-Specific* and *Embedded Number* types, the accuracy of the downstream model can be further improved.

7 DISCUSSION

7.1 Public Release and Leaderboard

We have released a public repository on GitHub with our entire labeled data for the ML feature type inference task [4]. We also release the pre-trained ML models: k-NN, logistic regression, RBF-SVM, Random Forest, and the CNN. The repository tabulates the precision, recall, and accuracy of all models and existing open-source approaches. The repository includes a leaderboard for public competition on the hosted dataset with 9-class classification accuracy and per-class precision, recall, and binarization accuracy being the metric. We release the downstream benchmark suite containing

30 datasets and the associated code for running the benchmark. Also, we release the raw 1240 CSV files and we invite researchers and practitioners to use our datasets and contribute to augmenting them and creating better featurizations and models.

7.2 Takeaways

7.2.1 For Practitioners. We make all the models and featurization routines available for use by wrapping them under functions in a Python library [4]. The ML models can be integrated for feature type inference into existing data prep environments. For visual tools such as Excel and Trifacta [11], designing new user-in-the-loop interfaces that account for both model's prediction and human's judgement remains an open research question. *We are currently in the process of integrating our pre-trained models with TFDV in collaborations with Google engineers to improve its inference of Categorical and Numeric feature types.*

7.2.2 For Researchers. We see three main avenues of improvement for researchers wanting to improve accuracy: better features, better models, and/or getting more labeled data.

First, designing features that can perfectly capture human-level reasoning is an open research question. We found that descriptive statistics and attribute names are very useful for prediction. But, raw attribute values have only marginal utility. Thus, one can consider designing better featurization routines for them. Capturing more semantic knowledge of attributes with an alternative neural architecture is another open problem. Finally, based on our analysis in Section 5.4, one potential way to increase the accuracy is to create more labeled data in categories of examples where ML models get confused, e.g., for *List* type. Weak supervision and denoising with Snorkel [34] and/or Snuba [41] is one potential mechanism to amplify labeled datasets and teach the ML models to learn better.

8 RELATED WORK

AutoML Platforms. Several AutoML tools such as AutoWeka [39] and Auto-sklearn [18] have an automated search process for model selection, allowing users to spend no effort for algorithm selection or hyper-parameter search. However, these AutoML systems do not automate data prep tasks. AutoML platforms such as Einstein AutoML [8], AutoML Tables [5], and AutoGluon [17] do automate some data prep tasks. However, how good their existing automation schemes are is not well-understood. We believe there is a pressing need to formalize data prep tasks and create benchmark labeled datasets for evaluating and comparing AutoML platforms on such tasks. The ML models trained on our labeled dataset can be integrated into such AutoML platforms to improve their accuracy on type inference, as we are currently doing with TFDV. In addition, other platforms for Machine Learning such as Airbnb's Zipline [1], Uber's Michelangelo [12], Facebook's FBLearner Flow [3], and commercial AutoML platforms such as H20.AI [7] and DataRobot [2] are complementary to our focus and they can also benefit by adopting the models trained on our labeled data.

ML Data Prep and Cleaning. Sherlock [25] is a distantly-supervised deep-learning-based tool that identifies 78 semantic types such as *Person*, *Code*, *Publisher*, *Religion*, etc. for automated schema matching and discovery. But such semantic types are not directly usable for AutoML because the same semantic type can span different ML feature types, e.g., Code can be *Categorical* or *Not-Generalizable*, with different implications for downstream modeling. This is by design because the application motivations are different: Sherlock is aimed at BI tool users to browse attributes more easily, not AutoML platform users. So, our work is complementary. Our focus is also on a novel benchmark dataset, not novel models. AutoType [42] synthesize type detection logic for semantic types such as *EAN Code*, *Swift Code*, etc. But it too is complementary and not directly usable for AutoML just like Sherlock above.

DataLinter is a rule-based tool that inspects a data file and raises potential data quality issues as warnings to the user [26]. However, ML feature type inference must be done manually. Many works study program synthesis-based approaches [21, 22, 24, 27] and/or visual interfaces [11] to reduce manual data transformation grunt work in data prep. There is also much work on reducing data validation and cleaning effort (e.g., [28, 29, 36]). Our work further this general direction on reducing manual effort but it is complementary to all these prior works: our paper is the first to formalize and benchmark ML feature type inference in AutoML platforms.

Database Schema Inference. DB schema inference has been explored in some prior work. Google's BigQuery does syntactic schema detection when loading data from external data warehouses [9]. [14] infers a schema from JSON datasets by performing *map* and *reduce* operations using pre-defined rules. But DB schema inference task is syntactic. For instance, the attribute type with integer values has to be identified as an integer. In contrast, with ML type inference the attributes with type integer can be *Categorical*.

Benchmarks. OpenML AutoML Benchmark focuses on understanding the automation of model selection and hyper-parameter search components of the ML workflow [20]. However, they do not cover any data prep steps. CleanML benchmark focuses on studying the effect of data cleaning operations on downstream models [30]. However, they do not handle the feature type inference task. Thus, both benchmarks are orthogonal to our work.

Data/Model Repositories. OpenML [40] is an open-source collaborative repository for ML practitioners and researchers to share their models, datasets, and workflows for reuse and discussion. Our labeled datasets can be made available to the OpenML community to invite more contributions for augmenting the current labeled dataset and for building more sophisticated models. Hence, our work is complementary to OpenML.

REFERENCES

[1] Accessed July 18, 2020. Airbnb Zipline https://conferences.oreilly.com/strata/strata-ny-2018/public/schedule/detail/68114.
[2] Accessed July 18, 2020. DataRobot https://www.datarobot.com/.
[3] Accessed July 18, 2020. Facebook's FBLearner Flow https://engineering.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/.
[4] Accessed July 18, 2020. Github Repository for ML Feature Type Inference https://github.com/pvn25/MLDataPrepZoo/tree/master/MLFeatureTypeInference.
[5] Accessed July 18, 2020. Google AutoML Tables https://cloud.google.com/automl-tables.
[6] Accessed July 18, 2020. Google Cloud AutoML https://cloud.google.com/automl/.
[7] Accessed July 18, 2020. H2o.AI https://www.h2o.ai/.
[8] Accessed July 18, 2020. SalesForce Einstein AutoML https://www.salesforce.com/video/1776007.
[9] Accessed July 18, 2020. Schema Detection BigQuery https://cloud.google.com/bigquery/docs/schema-detect.
[10] Accessed July 18, 2020. TransmogrifAI: Automated machine learning for structured data https://transmogrif.ai/.

[11] Accessed July 18, 2020. Trifacta: Data Wrangling Tools & Software https://www.trifacta.com/.
[12] Accessed July 18, 2020. Uber Michelangelo https://eng.uber.com/michelangelo/.
[13] Accessed July 18, 2020. URL Standards. https://url.spec.whatwg.org/.
[14] Mohamed-Amine Baazizi et al. 2017. Schema inference for massive JSON datasets. In *Extending Database Technology (EDBT)*.
[15] Denis Baylor et al. 2017. Tfx: A tensorflow-based production-scale machine learning platform. In *SIGKDD*. ACM.
[16] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19
[17] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *arXiv preprint arXiv:2003.06505* (2020).
[18] Matthias Feurer et al. 2015. Efficient and robust automated machine learning. In *Advances in neural information processing systems*.
[19] Jerome Friedman et al. 2001. *The elements of statistical learning*. Springer.
[20] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An open source AutoML benchmark. *arXiv preprint arXiv:1907.00909* (2019).
[21] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *ACM Sigplan Notices*, Vol. 46. ACM, 317–330.
[22] Sumit Gulwani et al. 2012. Spreadsheet data manipulation using examples. *ACM* (2012).
[23] Trevor Hastie, R Tibshirani, and J Friedman. 2001. *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. Springer-Verlag.
[24] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek Narasayya, and Surajit Chaudhuri. 2018. Transform-data-by-example (TDE) an extensible search engine for data transformations. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1165–1177.
[25] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1500–1508.
[26] Nick Hynes et al. 2017. The data linter: Lightweight, automated sanity checking for ml data sets. In *NIPS MLSys Workshop*.
[27] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and Hosagrahar V Jagadish. 2017. Foofah: A programming-by-example system for synthesizing data transformation programs. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1607–1610.
[28] Sanjay Krishnan et al. 2016. Activeclean: An interactive data cleaning framework for modern machine learning. In *SIGMOD*.
[29] Sanjay Krishnan et al. 2017. Boostclean: Automated error detection and repair for machine learning. *arXiv preprint* (2017).
[30] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2019. CleanML: A Benchmark for Joint Data Cleaning and Machine Learning [Experiments and Analysis]. *arXiv preprint arXiv:1904.09483* (2019).
[31] Wes McKinney. 2011. pandas: a foundational Python library for data analysis and statistics. *Python for High Performance and Scientific Computing* 14 (2011).
[32] Tom M. Mitchell. 1997. *Machine Learning*. McGraw Hill.
[33] Juan Ramos et al. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, Vol. 242. New Jersey, USA, 133–142.
[34] Alexander Ratner et al. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment* (2017).
[35] Olga Russakovsky et al. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* (2015). https://doi.org/10.1007/s11263-015-0816-y
[36] Sebastian Schelter et al. 2018. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment* (2018).
[37] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
[38] Survey. Accessed July 18, 2020. 2017 Kaggle survey on data science. https://www.kaggle.com/surveys/2017.
[39] Chris Thornton et al. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *ACM SIGKDD*. ACM.
[40] Joaquin Vanschoren et al. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter* (2014).
[41] Paroma Varma and Christopher Ré. 2018. Snuba: Automating weak supervision to label training data. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, Vol. 12. NIH Public Access, 223.
[42] Cong Yan and Yeye He. 2018. Synthesizing type-detection logic for rich semantic data types using open-source code. In *Proceedings of the 2018 International Conference on Management of Data*. 35–50.
[43] Xiang Zhang et al. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*.
[44] Xiang Zhang et al. 2015. Text understanding from scratch. *arXiv* (2015).

APPENDIX

A METHODOLOGY

For all the classical ML models, we use the Scikit-learn library in Python. For CNN, we use the popular Python library Keras on Tensorflow. We use a standard grid search for hyper-parameter tuning, with the grids described in detail below.

Logistic Regression: There is only one regularization parameter to tune: *C*. Larger the value of C, lower is the regularization strength, hence increasing the complexity of the model. The grid for C is set as $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100, 10^3\}$.

RBF-SVM: The two hyper-parameters to tune are C and γ. The C parameter represents the penalty for misclassifying a data point. Higher the C, larger is the penalty for misclassification. The $\gamma > 0$ parameter represents the bandwidth in the Gaussian kernel. The grid is set as follows: $C \in \{10^{-1}, 1, 10, 100, 10^3\}$ and $\gamma \in \{10^{-4}, 10^{-3}, 0.01, 0.1, 1, 10\}$.

Random Forest: There are two hyper-parameters to tune: *NumEstimator* and *MaxDepth*. *NumEstimator* is the number of trees in the forest. *MaxDepth* is the maximum depth of the tree. The grid is set as follows: *NumEstimator* $\in \{5, 25, 50, 75, 100\}$ and *MaxDepth* $\in \{5, 10, 25, 50, 100\}$.

k-Nearest Neighbor: The hyper-parameter to tune are the number of neighbors to consider (k) and the weight parameter in our distance function (γ). We use all integer values from 1 to 10 for k. The grid for γ is set as $\{10^{-3}, 0.01, 0.1, 1, 10, 100, 10^3\}$.

CNN Model: We tune *EmbedDim*, *numfilters* and *filtersize* of each Conv1D layer. The MLP has 2 hidden layers and we tune the number of *neurons* in each layer. The grid is set as follows: *EmbedDim* $\in \{64, 128, 256\}$, *numfilters* $\in \{32, 64, 128\}$, *filtersize* $\in \{2\}$, and *neurons* $\in \{250, 500, 1000\}$. In order to regularize, we use dropout with a probability from the grid: {0.25}. Rectified linear unit (ReLU) is used as the activation function. We use the Adam stochastic gradient optimization algorithm to update the network weights. We use its default parameters.

B CROWDSOURCING EFFORTS

We tried to crowdsource labels for our dataset on the FigureEight platform but abandoned this effort because the label quality was too low across two trial runs. In our pilot run, we used a concise label vocabulary with 5 classes: *Numeric*, *Categorical*, *Needs-Extraction*, *Not-Generalizable*, and *Context-Specific*. *Needs-Extraction* includes the classes: *Datetime*, *Sentence*, *URL*, *Embedded Number*, and *List*. In the first run, we got 5 workers each for 100 examples; in the second, 7 each for 415. The "golden" dataset were the 500 examples we labeled manually. We listed several rules and guidelines and provided many examples for worker training. But in the end, we found the results too noisy to be useful: in the first run, 4% of examples had 4 unique labels, 27% had 3, and 69% had 2; in the second run, these were 5%, 21%, and 49%. Majority voting gave the wrong answer in half of the examples we randomly checked. We suspect

Model		X_{stats}	X^*_{name}	$X^*_{sample1}$	X_{stats}, X^*_{name}	$X_{stats}, X^*_{sample1}$	$X^*_{name}, X^*_{sample1}$	$X^*_{sample1}, X^*_{sample2}$	$X_{stats}, X^*_{name}, X^*_{sample1}$	$X_{stats}, X^*_{name}, X^*_{sample1}, X^*_{sample2}$
Logistic Regression	Train	0.6954	0.8553	0.7139	0.9135	0.8288	0.9236	0.7975	0.9471	0.9571
	Validation	0.6927	0.7438	0.6551	0.8477	0.7743	0.8226	0.7117	0.8668	0.8749
	Test	0.6862	0.7293	0.6603	0.8428	0.7763	0.8043	0.7144	0.8578	**0.8643**
RBF-SVM	Train	0.892	0.9114	0.7133	0.9475	0.8779	0.9166	0.8392	0.9598	0.9605
	Validation	0.8203	0.7768	0.6529	0.8691	0.7847	0.8308	0.718	0.8822	0.8780
	Test	0.8213	0.7785	0.6521	0.8724	0.7845	0.8159	0.713	**0.8761**	0.8712
Random Forest	Train	0.9771	0.9168	0.7404	0.9817	0.9734	0.9447	0.8406	0.9803	0.9787
	Validation	0.9114	0.775	0.6604	0.9236	0.8938	0.837	0.7342	0.9195	0.9162
	Test	0.9121	0.777	0.6657	**0.9259**	0.8956	0.8346	0.7374	0.9216	0.9096
CNN	Train	0.7077	0.9545	0.7433	0.9846	0.8798	0.9855	0.8588	0.9727	0.9891
	Validation	0.7016	0.8167	0.6863	0.8768	0.7966	0.8892	0.7903	0.89	0.8821
	Test	0.6808	0.8019	0.6805	0.8692	0.7965	0.8655	0.7763	**0.8788**	0.8701
k-NN	Validation	0.8728	0.8002	-	0.8889	-	-	-	-	-
	Test	0.8605	0.7839	-	**0.8796**	-	-	-	-	-

Table 7: Full 9-class train, validation, and test accuracy of the ML models trained on our data with different feature sets. $X^*_{name}, X^*_{sample1}, X^*_{sample2}$ denote bigram features ($X2_{name}, X2_{sample1}, X2_{sample2}$) for classical ML models and raw character-level features ($X_{name}, X_{sample1}, X_{sample2}$) for CNN and k-NN. The bold fonts highlight the most accurate feature set for that model.

Descriptive Stats
Total number of values
Number of nans and % of nans
Number of unique values and % of unique values
Mean and std deviation of the column values, word count, stopword count, char count, whitespace count, and delimiter count
Min and max value of the column
Regular expression check for the presence of url, email, sequence of delimiters, and list on the 5 sample values
Pandas timestamp check on 5 sample values

Table 8: List of descriptive statistics features

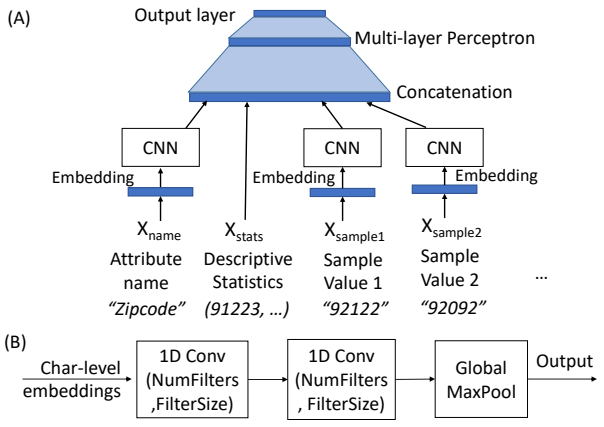


Figure 5: (A) The end-to-end architecture of our deep neural network. (B) The CNN block's layers.

such high noise arises because this task is too technically nuanced for lay crowd workers relative to popular crowdsourcing tasks like image recognition. Devising better crowdsourcing schemes for our task with lower label noise is an avenue for future work.

C DATA STATISTICS

Figure 8 plots the cumulative distribution functions (CDF) of different descriptive statistics obtained by base featurization. Table 11 presents the mean, standard deviation and the maximum of the same descriptive statistics. We observe that *Numeric* attributes have longer names than others. Attribute values for *Sentence*, *URL*, and *List* as expected, have more number of characters and words than other classes. In addition, we observe that all sample values in *Numeric* and 80% of the sample values in *Categorical* are single token strings. Furthermore, we find that almost 90% of the attributes in *Categorical* have less than 1% unique values in its columns. Interestingly, *Not-Generalizable* have either very few unique values or only NaN values in their domain.

D DESCRIPTIVE STATISTICS FEATURES IN BASE FEATURIZATION

Table 8 present all the descriptive stats used for base featurization.

E CNN

Figure 5(A) shows the architecture of CNN model. The layers of CNN are shown in Figure 5(B). The network takes attribute name, descriptive stats, and sample values as input and outputs the class from the label vocabulary. The attribute name and sample values are first fed into an embedding layer. The embedding layer takes as

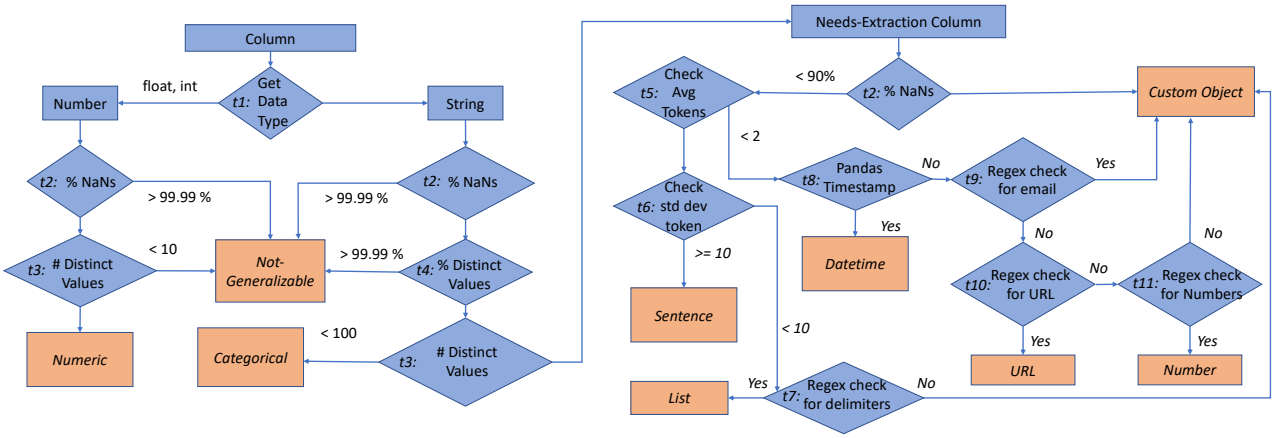


Figure 6: Flowchart of the rule-based baseline.

(A)	Numeric	Categorical	Datetime	Sentence	URL	Embedded Numbers	List	Not-Generalizable	Context-Specific
Numeric	669	0	0	0	0	1	0	37	0
Categorical	52	209	8	0	0	2	1	128	57
Datetime	4	7	19	0	0	0	1	90	20
Sentence	0	25	0	4	0	0	2	24	37
URL	0	12	0	0	8	0	0	6	6
Embedded Numbers	0	35	4	0	0	18	0	37	5
List	1	6	0	0	0	0	0	42	3
Not-Generalizable	35	59	0	0	0	2	0	109	10
Context-Specific	105	9	3	0	0	1	3	32	37

(B)	Numeric	Categorical	Datetime	Sentence	URL	Embedded Numbers	List	Not-Generalizable	Context-Specific
Numeric	696	3	0	0	0	0	0	2	6
Categorical	12	431	0	4	0	0	0	1	9
Datetime	0	2	137	0	0	2	0	0	0
Sentence	0	3	0	83	0	0	0	3	3
URL	0	2	0	0	30	0	0	0	0
Embedded Numbers	0	5	1	0	0	92	0	0	1
List	0	1	0	3	0	5	43	0	0
Not-Generalizable	3	13	6	4	1	0	0	185	3
Context-Specific	34	12	1	2	0	0	0	7	134

Table 9: Confusion matrices (actual class on row and predicted class on column) of (A) Rule-based baseline (B) Random Forest.

input a 3*D* tensor of shape (*NumSamples, SequenceLength, Vocabsize*). Each sample (attribute name or sample value) is represented as a sequence of one-hot encoded characters. *SequenceLength* represents the length of this character sequence and *Vocabsize* denotes

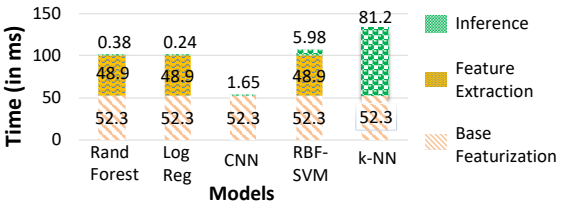


Figure 7: Comparison of prediction runtimes and breakdown for all models. Base Featurization is common for all models. Model-specific feature extraction is needed only for the 3 classical ML models.

the number of unique characters represented in corpus. The embedding layer maps characters to dense vectors and outputs a 3*D* tensor of shape (*NumSamples, SequenceLength, EmbedDim*), where *EmbedDim* represents the dimensionality of embedding space. The weights are initialized randomly and during training, the word vectors are tuned such that the embedding space exhibits a specialized structure for our task.

The resultant tensor from the embedding layers is fed into a CNN module, which consists of three cascading layers, 2 1-D Convolutions Neural Network, followed by a global max-pooling layer. The size of the filter (*FilterSize*) and number of filters (*NumFilters*) are tuned during training. We concatenate all CNN modules with descriptive statistics and feed them to a multi-layer perceptron on top. In the output layer, we use a softmax activation function that assigns a probability to each class of the label vocabulary. The whole network can be trained end-to-end using backpropagation.

F RULE-BASED BASELINE

Figure 6 shows the rule-based approach using a flowchart-like structure. The diamond-shaped nodes are the decision nodes that represents a "check" on the attribute. The final outcome is shown in orange rectangular boxes. Table 9 (A) shows the confusion matrix of the rule-based approach.

G EMPIRICAL STUDY

G.1 End-to-End Accuracy Results

Table 7 shows the train, cross-validation, and test accuracy results of all models trained on our dataset with 5-fold cross-validation methodology. Table 9 (B) shows the confusion matrix of the Random Forest.

Model		$[X_{stats}, X2_{name}]$
Logistic Regression	Train	0.9201
	Validation	0.8376
	Test	0.8411
RBF-SVM	Train	0.9612
	Validation	0.8554
	Test	0.8491
Random Forest	Train	0.9821
	Validation	0.9323
	Test	0.9199
k-NN	Validation	0.8537
	Test	0.8476

Table 10: 5-fold training, cross-validation, and held-out test accuracy of models with leave-datafile-out methodology. *k*-NN use our weighted edit distance function (Section 4.4).

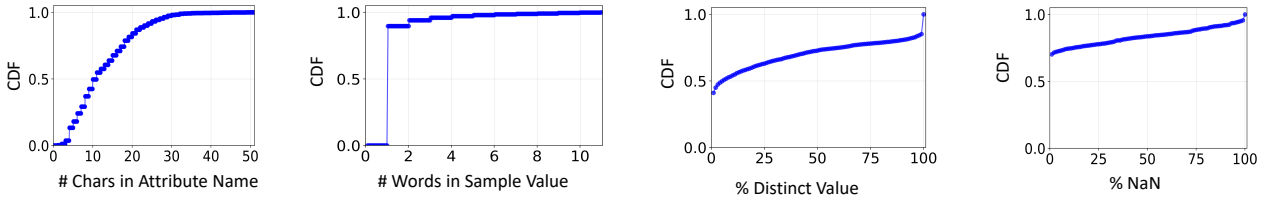
G.2 Leave-datafile-out methodology

We perform 5-fold leave-datafile-out cross validation to "stress-test" our models for new data files. In this methodology, the raw data files are split into 60:20:20 train, validation, and test partitions where each partition has columns of the same source data file. Thus, the test partition has columns of the raw data file that model has not seen before. Table 10 present the train cross-validation, and test accuracy results of the classical ML models and *k*-NN with this methodology on the 3-gram features from attribute name and descriptive stats. We observe that the results are comparable to what we found with *k*-fold cross-validation methodology.

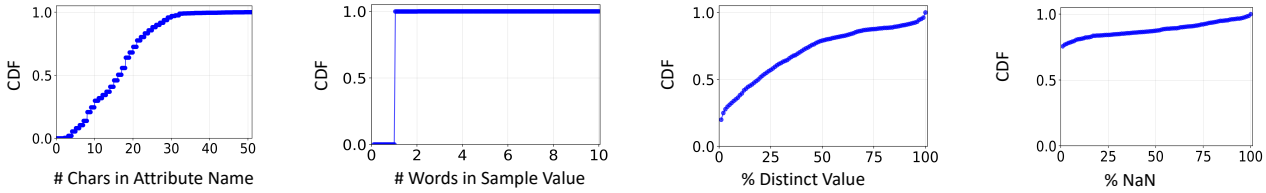
G.3 Prediction Runtimes

Figure 7 shows the time breakdown of base featurization, model-specific feature extraction time, and inference time of ML models.

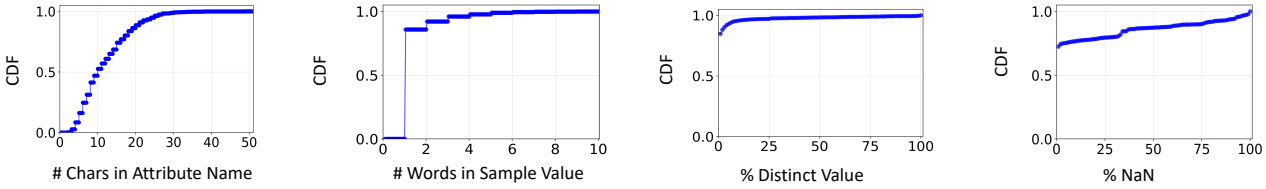
Overall



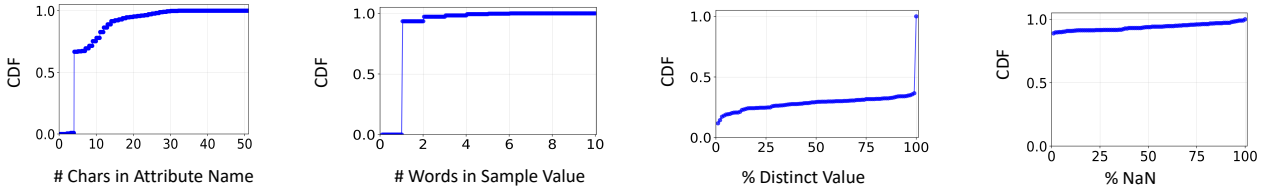
(1) Numeric



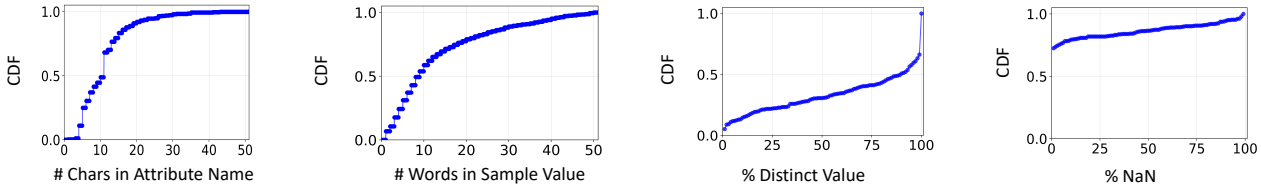
(2) Categorical



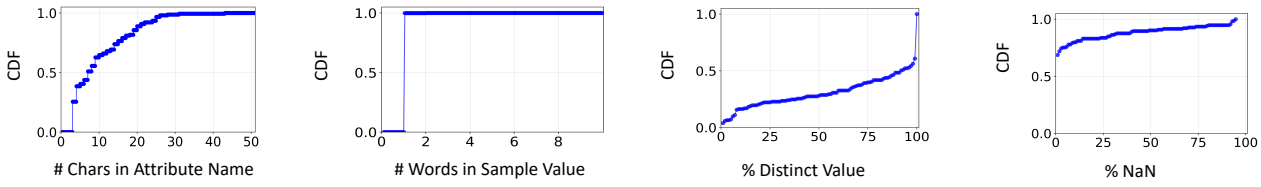
(3) Datetime



(4) Sentence



(5) URL



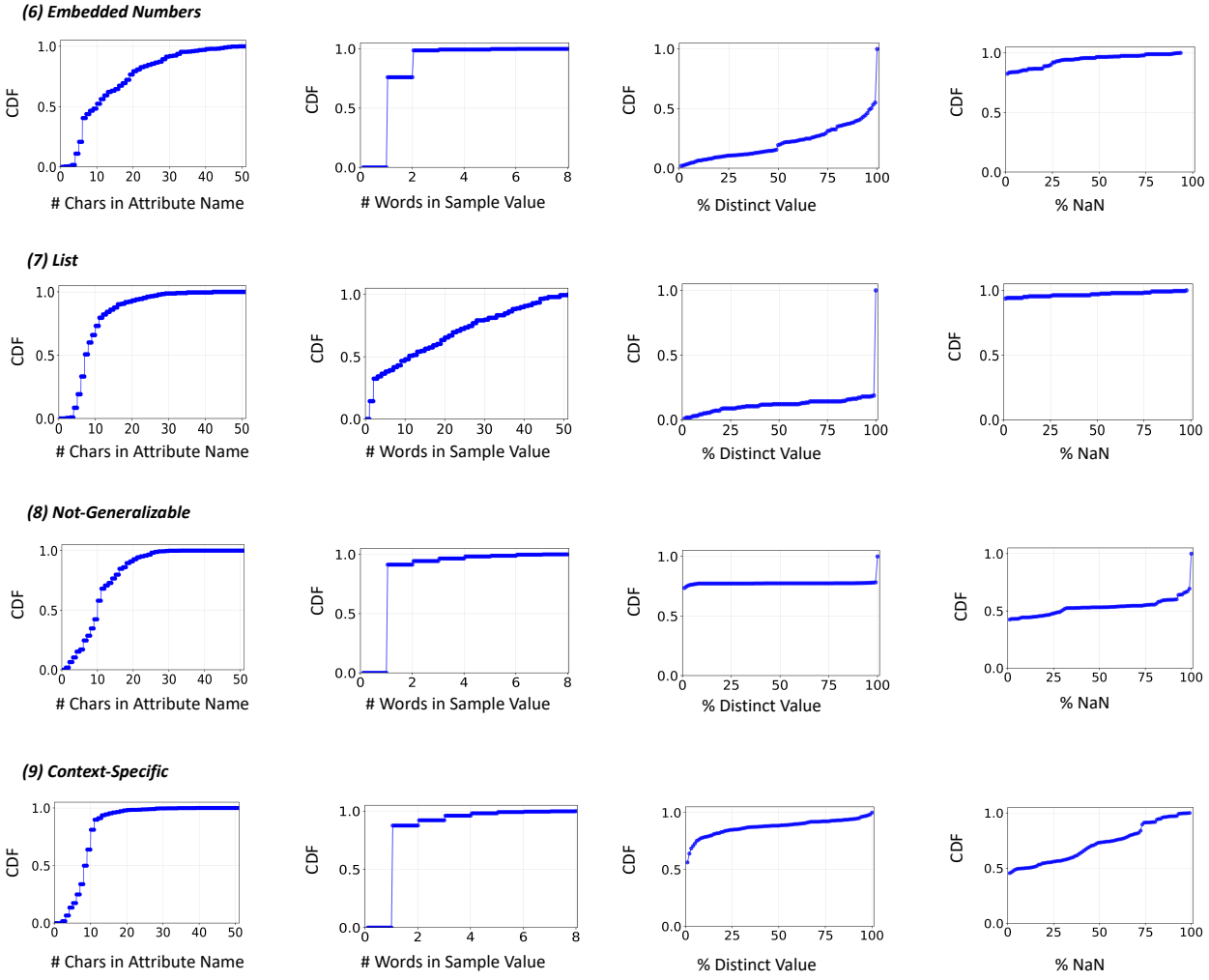


Figure 8: Cumulative distribution of different descriptive statistics features.

Statistics		Number of chars in Attribute Name	Number of chars in Sample Value	Number of words in Sample Value	Mean	% Distinct vals	% NaNs
Overall	Avg	13	25.7	3.5	1.5E+14	30.2	16.7
	Median	11	5	1	0	5.2	0
	Std Dev	9.8	242.3	38.3	9.9E+15	39.1	31.8
	Max	284	18584	3251	8.8E+17	100	100
Numeric	Avg	16.3	5.3	1	5.2E+10	29.1	11.9
	Median	16	5	1	1.7	19	0
	Std Dev	8.3	4.5	0.09	1.1E+12	30.9	27.1
	Max	91	398	7	5.6E+13	100	100
Categorical	Avg	11.9	6.4	1.4	3E+5	2.5	14.1
	Median	10	3	1	0	0.04	0
	Std Dev	7.2	10.9	1.2	6.5E+6	11.4	28.9
	Max	112	158	21	2.1E+8	100	100
Datetime	Avg	7.1	7.8	1.1	6.4E+9	72	6.2
	Median	4	6	1	0	100	0
	Std Dev	5.6	4.6	0.6	8.4E+10	41.7	20.7
	Max	32	119	18	1.5E+12	100	100
Sentence	Avg	11.8	282.3	44.4	0	67.8	13.6
	Median	11	62	10	0	89.9	0
	Std Dev	11	1134.5	186.5	0	37.4	29
	Max	132	18584	3251	0	100	99
URL	Avg	10	65.2	1.2	0	79	11.2
	Median	7	56	1	0	97	0
	Std Dev	7.7	56.9	0.9	0	30.2	25.2
	Max	43	632	7	0	100	94.8
Embedded Numbers	Avg	15.2	6.9	1.3	0	88.8	5.7
	Median	10	6	1	0	100	0
	Std Dev	14.3	2.7	0.5	0	27.5	15.9
	Max	116	41	6	0	100	93.5
List	Avg	14	296.2	26.7	0	88.8	3
	Median	8	191	18.5	0	100	0
	Std Dev	32	341.5	30.5	0	27.5	13.8
	Max	284	2417	203	0	100	97.7
Not-Generalizable	Avg	10.7	8.4	1.6	1.1E+10	22.7	47.2
	Median	10	3	1	0	0.02	29.2
	Std Dev	5.9	32.3	4.3	3E+11	41.6	46.4
	Max	33	689	89	9.8E+12	100	100
Context-Specific	Avg	8.9	14.2	1.7	1.7E+15	12.7	27.8
	Median	9	2	1	1.3	0.6	8.8
	Std Dev	5.1	101	5.5	3.3E+16	26.8	31.8
	Max	69	1964	134	8.8E+17	100	99.1

Table 11: Average, standard deviation, and maximum value of different descriptive statistics features.