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@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. Emerging industrial-scale cloud AutoML platforms aim to automate the end-to-end ML workflow. While many works have looked into automated feature engineering, model selection, or hyper-parameter search in AutoML, little work has studied a crucial step that serves as an entry point to this workflow: *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. In this work, we formalize and standardize this task by creating the first ever benchmark labeled dataset, which we use to objectively evaluate existing AutoML tools. 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.

1 INTRODUCTION

The paradigm of automated machine learning (AutoML) is beginning to help democratize machine learning for the masses [1]. Cloud vendors have released AutoML platforms such as Google's Cloud AutoML [2] and Salesforce's Einstein [3] that build ML models on millions of datasets from thousands of small-and-medium enterprises automatically. The central goal of these platforms is to get an accurate model for the prediction task while achieving maximum possible automation of the end-to-end ML workflow, especially on structured data, including data transformations and feature engineering, as well as model building and hyperparameter tuning. The automation of these steps has been intensively studied in the ML/data mining [1, 4] and database communities [5, 6]. However, a crucial gateway step to this whole workflow has received much less attention so far: *ML feature type inference*.

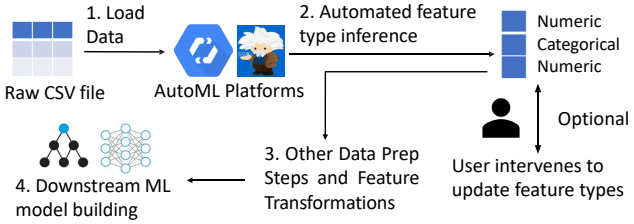


Figure 1: Typical workflow in AutoML platforms.

Datasets are typically loaded as files into the AutoML platforms. As Figure 1 illustrates, *ML feature type inference* is the very first step needed for ML over structured data. Features could be *Numeric*, *Categorical*, or something else, as shown in Figure 1. Determining the correct feature type is crucial for the whole workflow to work well: what data transformations to apply, how to extract features, and how to feed signals to the downstream models. For instance, if a column is inferred to be of type Timestamp, then several useful features such as day, month, and year are often extracted automatically for the downstream model. Thus, the accuracy of feature type inference is critical for the downstream model's accuracy, and in turn, the effectiveness of the entire ML platform.

Feature type inference is also performed automatically by many ML platforms, e.g. TransmogrifAI in Einstein [7], Tensorflow Data Validation (TFDV) in TensorFlow Extended [8], and AutoGluon from AWS [9]. But surprisingly, there is no objective evaluation to date of how good their automation of this task is. Thus, mistakes in their automated feature type inference can propagate and may degrade the workflow. For instance, consider what TFDV does on the illustrative dataset for a common ML task, customer churn prediction in Figure 2. 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.

One might ask: *Why cannot AutoML platform users manually verify their feature types?* From our conversations with AutoML platform engineers at Salesforce and Google, we learned that their AutoML tools are used on over tens of thousands of datasets, adding up to millions of features in production settings. Forcing users to

CustID	Gender	Salary	ZipCode	XYZ	Income	HireDate	Churn
1501	'F'	1500	92092	005	'USD 15000'	'05/01/1992'	'Yes'
1704	'M'	3400	78712	003	'25384'	'12/09/2008'	'No'

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

manually annotate features can lead to a tedious, slow, and error-prone process that also *violates the promise of automation.* Many domain users who may not have much ML expertise may not like the platform asking them to manually mark ML feature types. Thus, AutoML platform engineers prefer ever more accurate automation of this task. Clearly, this requires them to objectively measure the accuracy of their AutoML tool on the given task.

1.1 This Paper's Focus

Our Focus. We initiate work on benchmarking and objectively quantifying the task of *ML feature type inference* in existing open-source industrial-strength AutoML tools. 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 [10]. Moreover, this will help objectively evaluate and improve AutoML platforms by enabling answers to key questions: *How good are AutoML tools? How can one do better? How does the accuracy of type inference affect downstream ML model's accuracy?*

Challenge. We first 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 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. For instance, consider Figure 2 again. Attributes such as *CustID*, *Salary*, and *ZipCode* are stored as integers, but only *Salary* is useful as *Numeric. CustID* is unique for every customer, hence it can not be generalized for ML. *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.

Scope. Our focus is on relational/tabular data, which can be stored in any format (CSV, JSON, XML, etc.) and with any filesystems. Note that our focus is not to study any upstream processing steps that users might perform when they load their files into the AutoML tool. Also, our focus is not on feature engineering and transformation steps over the columns with the inferred types. We focus only on the ML feature type inference step. Admittedly, this is just one step in the entire end-to-end ML workflow, but we believe that studying this step in depth is critical to improve existing AutoML platforms, as we find that accurate type inference is critical for achieving high downstream model accuracy. Equally importantly, the predictions are more *interpretable* with accurate feature types.

1.2 Benchmark Comparisons

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 survey existing AutoML data prep tools and collect their feature type vocabulary into a common, practically useful set of labels that can be reused by any AutoML platform, as Figure 3 shows. *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 from sources such as Kaggle and UCI ML repository. Our labeling process took about 90 man-hours across 5 months.

Current Limitation. We admit that files on Kaggle and UCI ML repository may not be representative of the truly "in-the-wild" dataset as it may have undergone some pre-processing. But, it is impractical for researchers to get access to large numbers of publicly releasable data from enterprises and organizations due to legal restrictions. Thus, Kaggle and UCI are the closest sources we have to the real-world data. We believe that our exploratory work is the first step in the direction of objectively evaluating AutoML tools. We hope that this work starts a conversation around enhancing such benchmark datasets.

Approaches to Type Inference. There are open-source tools such as Pandas [11], TransmogrifAI [7], TFDV [8], and AutoGluon [9] 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.

Semantic Type Detection Tools. Recent tools such as Sherlock [12] and AutoType [13] perform column-level semantic type detection for automated data discovery and cleaning. The semantic type vocabulary of these tools is not directly usable for the AutoML setting because a semantic type can belong to multiple ML feature types. This is by design because the application motivations are different: semantic type detection tools are aimed at Business Intelligence (BI) tool users to browse attributes more easily, not AutoML users. Thus, it is complementary to our focus. To understand whether such tools can be ported to the AutoML setting, we use a rule-based approach to map Sherlock's semantic types to our vocabulary and evaluate it on our 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. 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. **A new benchmark task and dataset.** To the best of our knowledge, this is the first work to formalize and rigorously benchmark the task of ML feature type inference. We create the first large benchmark labeled datasets for this task with a readily practically useful 9-class label vocabulary.
2. **Benchmarking alternate tools and approaches.** Using our new data, we perform extensive empirical comparisons of open source and industrial (Auto)ML tools. Perhaps surprisingly, we find that even off-the-shelf ML models with standard featurization trained on our data significantly outperform all prior approaches.
3. **Downstream benchmark suite.** The curated benchmark offers evidence that the downstream model's performance can benefit by accurately determining feature types. We find that an ML model trained on our data for feature type inference often leads to more accurate downstream models than prior tools.
4. **Real-world impact.** Google collaborated with us to integrate our best performing ML models into TFDV to improve its inference of *Categorical* [14]. Google engineers are now reviewing it on internal benchmarks for adoption. AWS and OpenML [15] have also expressed interest in adopting our data and models for production use. Also, we release a public competition on our labeled dataset to invite contributions to create/augment datasets, better featurizations, and models.

2 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.

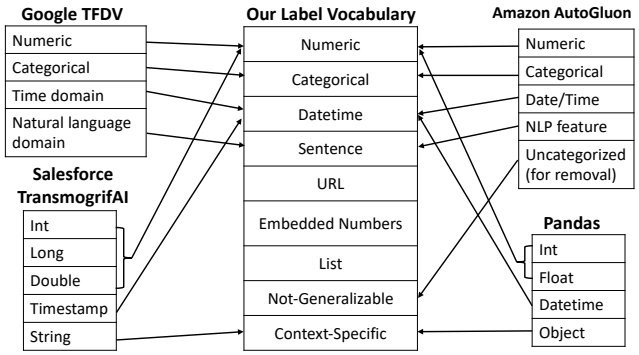


Figure 3: Feature type vocabulary mapping of TFDV, Pandas, TransmogrifAI, and AutoGluon to our vocabulary

2.1 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 [8], TransmogrifAI in Salesforce Einstein [7], and AutoGluon from Amazon AWS [9] 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.

(1) Numeric. These attributes are quantitative in nature and can directly be utilized as a *Numeric* feature for the downstream ML model. For instance, *Salary* is *Numeric*, while ID attributes such as *CustID* or integers representing encodings of discrete levels are not.

(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 [16]. 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 get new features 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. In contrast, 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) Context-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.

2.2 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 2.4. 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. 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 4.2). Thus, we believe our work is a promising start towards objectively evaluating AutoML platforms.

2.3 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. We choose 5 because we think it is a reasonable number for a human to understand the column and determine the feature type when doing manual labeling (Section 2.4). However, this number can very well be higher or lower. It can be even tuned when building an ML model or a heuristic. In fact, from the ablation study of the ML models built

on the base features, we find that even one or two sample values may be good enough to build an accurate model (Table 2).

(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*. Considering this, we extract 25 descriptive stats for a column such as the total number of values, the absolute number and % of NaNs relative to total values, the absolute number and % of distinct values relative to total values, mean, and standard deviation. We provide the complete list of these 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).

2.4 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. We extract these examples from additional sources as we did not want to create a heavily skewed class label distribution to get good confidence on all classes. Note that augmenting classes where the number of examples is under-represented is a common practice in the ML literature [17–20]. Since our benchmark contains multiple class-level accuracy metrics (discussed in Section 4.1), inspecting them can provide more confidence with the class predictions. Furthermore, 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 a 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 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. We summarize our crowdsourcing results in the Appendix.

2.5 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 *Context-Specific* (8.9%). We provide a complete breakdown of the cumulative distribution by class for different descriptive statistics in the Appendix.

3 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.

3.1 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 [8]. TFDV uses 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.

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 [11]. 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 [7]. 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 types for their data.

AutoGluon-Tabular. AutoGluon is an end-to-end AutoML framework from AWS [9]. 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.

Sherlock. Sherlock [12] is a distantly-supervised deep-learning-based tool that identifies 78 semantic types such as *Age*, *Code*, *Duration*, etc. But the semantic types are not directly usable for AutoML because the same semantic type can span different ML feature types. For instance, *Duration* type can be either *Numeric* (e.g., time elapsed in seconds), *Categorical* (e.g., time duration belonging to a discrete set), *Datetime* (e.g. the exact timestamp), or even *Sentence* (e.g., duration mentioned in words).

We find that out of 78 semantic types, 55 types can be uniquely mapped to one single class of our label vocabulary. The number of types that are mapped to 2, 3, and 4 classes of our label vocabulary are 18, 3, and 2, respectively. We release the mapping between Sherlock semantic types and our label vocabulary in the Appendix. We use a rule-based approach on top of Sherlock to identify one single feature type given a column. We discuss examples to illustrate how we map semantic types to our feature types in the Appendix.

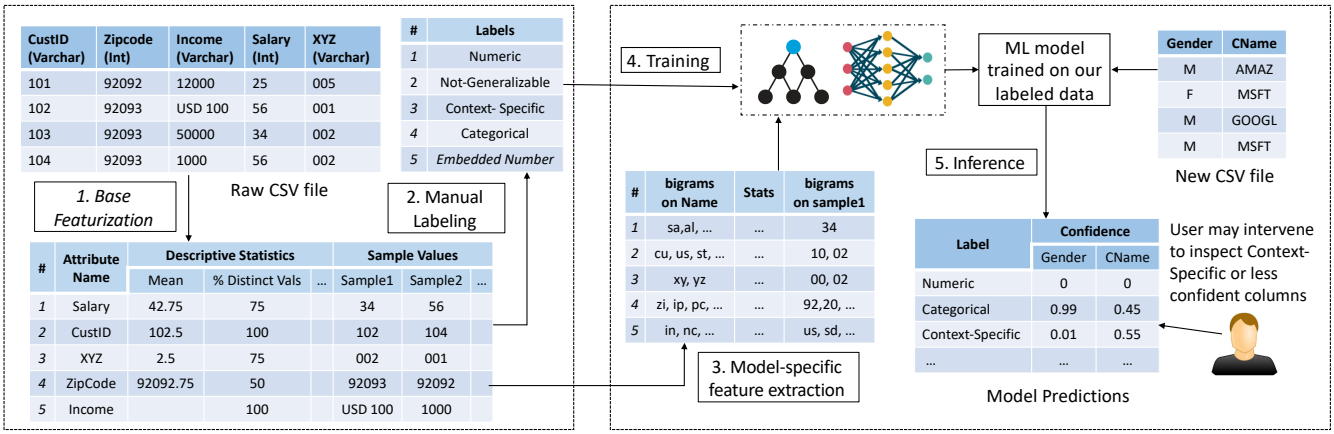


Figure 4: Workflow showing our labeling process and how our data is used for ML-based feature type inference.

3.2 Rule-based Baseline

We use this 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. We provide two examples below. (1) To identify *List*, non-empty sample values are matched with a regular expression based check of a series of characters separated by a type of delimiter such as ; | , etc. (2) If either of the % of NaNs or % of unique values in the column are greater than 99.99% then we mark it as *Not-Generalizable*. We describe the complete rule-based approach in the Appendix.

3.3 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.

3.3.1 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 randomly sampled attribute values by $\mathbf{X}_{sample}$ (first sampled value referred to as $\mathbf{X}_{sample1}$ and similarly for other values). We leverage the commonly used bigram features on the attribute name (denoted by $\mathbf{X2}_{name}$) and sample value ($\mathbf{X2}_{sample}$).

3.3.2 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. 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.

3.3.3 Nearest Neighbor. Most implementations of k-NN use a simple Euclidean distance. But, we can adapt the distance function for the task at hand by defining 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.

3.3.4 CNN. Inspired by the success of CNN on short text classification tasks [21, 22], we leverage a character-level CNN for our task. The network takes attribute name, descriptive stats, and sample values as input and outputs the class from the label vocabulary. We present the architecture and layers of CNN in the Appendix.

4 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.

4.1 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. We describe the grids 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

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.

Feature Type	Metric	Open-source Industrial Tools				Sherlock + Rules	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	0.599	0.773	0.909	0.929	0.934
	Recall	1	1	1	1	0.359	0.946	0.943	0.941	0.984
	Accuracy	0.814	0.776	0.767	0.805	0.683	0.882	0.946	0.953	**0.97**
Categorical	Precision	0.396	-	-	0.667	0.311	0.577	0.808	0.846	0.913
	Recall	0.652			0.534	0.707	0.457	0.884	0.928	0.943
	Accuracy	0.691			0.831	0.567	0.798	0.925	0.945	**0.966**
Datetime	Precision	0.985	0.956	1	1	0.89	0.559	0.951	0.925	0.945
	Recall	0.475	0.915	0.454	0.844	0.801	0.135	0.972	0.965	0.972
	Accuracy	0.962	0.991	0.961	0.989	0.979	0.931	0.994	0.992	**0.994**
Sentence	Precision	0.472	-	-	0.516	0.354	1	0.913	0.725	0.865
	Recall	0.457			0.902	0.554	0.043	0.793	0.804	0.902
	Accuracy	0.951			0.956	0.932	0.956	0.987	0.977	**0.989**
Not-Generalizable	Precision	-	-	-	0.465	0.692	0.216	0.732	0.81	0.934
	Recall				0.53	0.042	0.507	0.732	0.66	0.86
	Accuracy				0.883	0.893	0.747	0.947	0.937	**0.978**
Context-Specific	Precision	-	0.08	0.074	-	0.192	0.211	0.747	0.741	0.859
	Recall		0.295	0.295		0.168	0.195	0.621	0.663	0.705
	Accuracy		0.609	0.582		0.851	0.853	0.944	0.946	**0.961**

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 discuss its results in the Appendix.

Experimental Setup. We use CloudLab [23] 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 for the 9-class task. We also use class-specific binarization metrics such as precision, recall, F1 score, and confusion matrix.

4.2 Comparison of All Approaches

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

Results. Table 1 presents the precision, recall, and overall 2 x 2 diagonal accuracy results of all approaches on our benchmark labeled held-out test set[1]. We report F1 score and full confusion matrices in the Appendix. 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, a lift of 28% and 14% in accuracy in predicting *Categorical* compared to TFDV and AutoGluon respectively. Of all approaches, Random Forest achieves the highest accuracy in inferring the 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.

(3) Heuristics for identifying *Datetime* by all the existing tools have high precision, even higher than the ML models. However, their

[1] Since publication, we have released version 2 of our labeled dataset where 32 examples are relabeled after feedback on Github [24]. We find only minor changes in the results without altering any of our trends, conclusions, or takeaways discussed here. Note that our labeled dataset is a living public repository on Github which we anticipate to grow in the future. Please refer to our public repository for the up-to-date results.

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.

	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**	-	-	-	-	-

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 [25], DataRobot [26], and Trifacta [27]. 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. Since it is hard to draw any generalizable conclusion if these tools have the same issues as TFDV, AutoGluon, and TransmogrifAI, we leave it to future work to assert this more systematically.

4.3 Comparison of ML-based Approaches

Rule-based Baseline. The 9-class classification accuracy on the held-out test set is only 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 category encoded as a number 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.

Sherlock. Sherlock with a rule-based approach that maps their semantic types to our label vocabulary has an accuracy of just 42%. This is because their semantic type vocabulary is not suitable towards identifying ML feature types. The number of Sherlock semantic types (out of 78) that are mapped to ML feature types are: 14 to *Numeric*, 50 to *Categorical*, 4 to *Datetime*, 7 to *Sentence*, 11 to *Embedded Number*, 2 to *List* and *Not-Generalizable*, and 18 to *Context-Specific*. Since *Categorical* type occur most frequently, more examples in our labeled dataset are disproportionately confused with this feature type. For instance, many integer *Numeric* attributes are confused with semantic types that often contains discrete set of integers (such as *Credit* and *Class*). Interestingly, Sherlock has a high precision of 89% in identifying *Datetime* correctly, even with just 4 semantic type mapped to *Datetime*.

Classical ML Models. Table 2 presents the 9-class accuracy results of the classical ML models using different feature sets[1]. 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 bigrams of the attribute name into logistic regression leads to a whopping 15% lift in accuracy. However, 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 and Nearest Neighbor. Table 2 also shows the CNN and k-NN accuracy[1]. We see that with just X_{name}, the CNN 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. With k-NN, we observe that with only Euclidean distance on descriptive statistics, the accuracy is already at 86%. The edit distance on the attribute name approach achieves an accuracy of 78%. Finally, with our weighted edit distance function from Section 4.4, k-NN achieves a high 88% accuracy.

4.4 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. Table 3 shows

Table 3: Examples of errors made by RandomForest. *Numeric* (*NU*), *Categorical* (*CA*), *Datetime* (*DT*), *Sentence* (*ST*), *Not-Generalizable* (*NG*), *Embedded Number* (*EN*), *URL*, *List* (*LST*), and *Context-Specific* (*CS*) are feature types.

#	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*

examples of columns and the corresponding prediction made by OurRF. We present the full confusion matrix of the predicted class by OurRF vs actual data type of the attribute value in the Appendix. We intuitively explain the errors by class below.

***Numeric* and *Context-Specific*.** We see that OurRF is less likely to misclassify a *Numeric* 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 3 example(A)). This is possibly because of the non-sensical attribute name. Similarly, *Context-Specific* integers are most commonly misclassified with *Numeric* (Table 3 example(H)).

***Categorical* and *Not-Generalizable*.** When the sample values are strings with more than one token, OurRF is more likely to misclassify *Categorical* as *Sentence* or *Context-Specific* (Table 3 example(B)). *Not-Generalizable* types are often confused with *Categorical*. For instance, *q19TalTool-ResumeScreen* (Table 3 example(G)) has only 2 values in its domain: "NULL!" and "ResumeScreen." However, OurRF treats "NULL!" as a separate category. Thus, OurRF is lacking in its semantic understanding ability of sample values.

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 3 example(C)) even though there is no number available for extraction. This can be due to few available training examples for *List* type.

4.5 Prediction Runtimes and Extensions

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 classical models, the additional feature extraction dominates the overall runtime. Since SVM and k-NN are distance-based methods, they have the highest runtime. Overall, CNN is the fastest. We present the time breakdown in the Appendix.

Our benchmark and ML-based approach can be easily extended to support new additional types, including semantic types [12]. We showcase the the effort needed for this extension in the context of two semantic data types that are commonly used in BI applications: *Country* and *State*. We find that the overhead of supporting these additional types in terms of programming cost, feature engineering cost, and labeling cost is minimal to almost none. We present the complete discussion in the Appendix.

Table 4: (A) Type Inference accuracy on 30 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.

(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

5 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 4.3, 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.

5.1 Datasets

The impact of type inference is dependent on the dataset and the downstream prediction task. Since there are unboundedly many datasets and downstream tasks, for the sake of tractability we got 30 "unseen" datasets from Kaggle, UCI ML repository, and OpenML [15] for evaluation. Since classification tasks are more common in practice, we got 25 datasets for such tasks, and 5 for regression tasks. Table 5 presents the downstream datasets with descriptions such as their number of columns, target classes, and different feature types and attribute types they contain. 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 the Github repository [24].

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) 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*

5.2 Models and Metrics

In terms of downstream model evaluation, we present both extremes of bias-variance tradeoff [28]: 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 the root mean squared error (RMSE) metric for the regression tasks.

5.3 Tools compared

We compare Pandas (PD), TFDV, AutoGluon (AGL), 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 [29], *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 in Section 4.1 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.

5.4 Results

5.4.1 Type Inference Results. Table 4 (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.

5.4.2 Downstream Model Performance. Table 5 presents the 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 4 (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, OurRF underperforms the truth (perfect feature type predictions) for only 20 downstream models. In contrast, Pandas, TFDV, and AutoGluon underperform for significantly more models: 44, 35, and 35 respectively. We present the CDFs of the magnitude of the difference in downstream performance with different approaches compared to Truth in the Appendix. We explain the results in-depth below.

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 of 7% and up to 52% in accuracy compared to the ground truth-based model. We explain the common patterns of how wrong type inference affected downstream accuracy in the Appendix.

We empirically studied if the gap in downstream model accuracy caused by wrong type inference relative to the true types be bridged by giving multiple representations of the column at the same time to the downstream model. We studied this in the context of *Numeric* vs *Categorical* dichotomy of the integer columns of our downstream datasets. We again found that accurate inference of feature types is critical to building accurate downstream models. We present the complete evaluation and additional insights in the Appendix.

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 lift accuracy?
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. Thus, a user-in-the-loop can further improve downstream model. Moreover, such intervention can even help *Flags* where a *Categorical* feature was erroneously predicted as *Context-Specific* by OurRF.

4. Why is outperforming truth not necessarily beneficial?
On *IOT*, we observe the lift in accuracy due to a *Numeric* column called "temp" (denoting temperature) being classified as *Context-Specific*. This 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. We present more such cases in the Appendix.

5.5 Summary

Overall, OurRF achieves a high accuracy of 91.2% for inferring feature types on 30 unseen datasets from Kaggle, UCI ML repository, 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 achieve the highest accuracy against existing tools for 47 out of 60 downstream models.

6 DISCUSSION

6.1 Public Release and Leaderboard

We have released a public repository on GitHub with our entire labeled data for the ML feature type inference task [24]. 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.

6.2 Takeaways

6.2.1 For Practitioners. We make all the models and featurization routines available for use by wrapping them under functions in a Python library [24]. The ML models can be integrated for feature type inference into existing data prep environments. *We have already integrated our pre-trained models with TFDV to improve its inference of Categorical [14].* We are also planning to collaborate with AWS and OpenML on more such integration. We welcome inquiries from more practitioners interested in adopting or enhancing our benchmark. For visual tools such as Excel and Trifacta [27], designing new user-in-the-loop interfaces that account for both model's prediction and human's judgement remains an open research question.

6.2.2 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 stats and attribute names are most useful for prediction, while raw attribute values have only marginal utility. Thus, one can consider designing better featurization routines for them. Second, capturing more semantic knowledge of attributes with an alternative neural architecture is another open problem. Finally, based on our analysis in Section 4.4, 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 [30] and/or Snuba [31] is one potential mechanism to amplify labeled datasets and teach the ML models to learn better.

7 RELATED WORK

AutoML Platforms. Several AutoML tools such as AutoWeka [32] and Auto-sklearn [33] 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 the ML feature type inference task. Several tools perform automatic data transformation steps and generate a set of useful features given a dataset [34, 35]. However, Deep Feature Synthesis algorithm [35] assumes that the ML feature types are provided explicitly as input, while ExploreKit [34] operates on the syntactic types. Thus, such automatic feature engineering tools can benefit by leveraging the ML models trained on our labeled data.

Other end-to-end AutoML platforms such as Einstein AutoML [3], AutoML Tables [25], and AutoGluon [9] do automate the type inference task. We believe that the standardization of the task and our benchmark labeled dataset is valuable to objectively compare and improve their AutoML platforms. The ML models trained on our labeled dataset can be integrated into such AutoML platforms to improve their type inference accuracy. In addition, other ML platforms such as Airbnb's Zipline [36], Uber's Michelangelo [37], Facebook's FBLearner Flow [38], and commercial platforms such as H20.AI [39] and DataRobot [26] are complementary to our focus and they can also benefit by adopting models trained on our data.

ML Data Prep and Cleaning. Auto-Type [13] is a semantic type detection tool that synthesizes 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. DataLinter is a rule-based tool that inspects a data file and raises potential data quality issues as warnings to the user [40]. However, ML feature type inference must be done manually. Many works study program synthesis-based approaches [5, 41–43] and/or visual interfaces [27] to reduce manual data transformation grunt work in data prep. There is also much work on reducing data validation and cleaning effort (e.g., [6, 44, 45]). 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 [46]. [47] 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 [48]. However, they do not cover any data prep steps. CleanML benchmark focuses on studying the effect of data cleaning operations on downstream models [49]. However, they do not handle the feature type inference task. Thus, both benchmarks are orthogonal to our work.

Data/Model Repositories. OpenML [15] 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.

Acknowledgments. This work is supported in part by the National Science Foundation Convergence Accelerator grant under award number OIA-2040727 and gifts from Google and Amazon. We thank the members of UC San Diego's Database Lab, Alkis Polyzotis and Google TFDV team, Julian McAuley, Andreas Müller, Lawrence Saul, and Jingbo Shang for their feedback on this work.

REFERENCES

[1] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning - Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer, 2019.

[2] Google Cloud AutoML, https://cloud.google.com/automl/, Accessed March 22, 2021.

[3] Salesforce Einstein AutoML, https://www.salesforce.com/video/1776007, Accessed March 22, 2021.

[4] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An Open Source AutoML Benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

[5] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and Hosagrahar V Jagadish. Foofah: A Programming-By-Example System for Synthesizing Data Transformation Programs. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1607–1610, 2017.

[6] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, and Eugene Wu. BoostClean: Automated Error Detection and Repair for Machine Learning. *CoRR*, abs/1711.01299, 2017.

[7] TransmogrifAI: Automated Machine Learning for Structured Data, https://transmogrif.ai/, Accessed March 22, 2021.

[8] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1387–1395. ACM, 2017.

[9] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander J. Smola. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *CoRR*, abs/2003.06505, 2020.

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015.

[11] Wes McKinney. pandas: a Foundational Python Library for Data Analysis and Statistics. *Python for High Performance and Scientific Computing*, 14, 2011.

[12] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1500–1508, 2019.

[13] Cong Yan and Yeye He. Synthesizing Type-Detection Logic for Rich Semantic Data Types Using Open-Source Code. In *Proceedings of the 2018 International Conference on Management of Data*, pages 35–50, 2018.

[14] Vraj Shah, Kevin Yang, and Arun Kumar. Improving Feature Type Inference Accuracy of TFDV with SortingHat, Accessed March 22, 2021. https://adalabucsd.github.io/papers/TR_2020_TFDV.pdf.

[15] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luís Torgo. OpenML: networked science in machine learning. *SIGKDD Explor.*, 15(2):49–60, 2013.

[16] URL Standards, https://url.spec.whatwg.org, Accessed March 22, 2021.

[17] Christopher Ré, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. Overton: A Data System for Monitoring and Improving Machine-Learned Products. *arXiv preprint arXiv:1909.05372*, 2019.

[18] Vincent Chen, Sen Wu, Alexander J Ratner, Jen Weng, and Christopher Ré. Slice-based Learning: A Programming Model for Residual Learning in Critical Data Slices. In *Advances in neural information processing systems*, pages 9397–9407, 2019.

[19] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. Slice Finder: Automated Data Slicing for Model Validation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1550–1553. IEEE, 2019.

[20] Ki Hyun Tae and Steven Euijong Whang. Slice Tuner: A Selective Data Collection Framework for Accurate and Fair Machine Learning Models. *arXiv preprint arXiv:2003.04549*, 2020.

[21] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level Convolutional Networks for Text Classification. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 649–657, 2015.

[22] Xiang Zhang and Yann LeCun. Text Understanding from Scratch. *CoRR*, abs/1502.01710, 2015.

[23] 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. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[24] Github Repository for ML Feature Type Inference, https://github.com/pvn25/ML-Data-Prep-Zoo/tree/master/MLFeatureTypeInference, Accessed March 22, 2021.

[25] Google AutoML Tables, https://cloud.google.com/automl-tables, Accessed March 22, 2021.

[26] DataRobot, https://www.datarobot.com, Accessed March 22, 2021.

[27] Trifacta: Data Wrangling Tools & Software, https://www.trifacta.com/, Accessed March 22, 2021.

[28] Trevor Hastie, Jerome H. Friedman, and Robert Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2001.

[29] Juan Ramos. Using TF-IDF to Determine Word Relevance in Document Queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. New Jersey, USA, 2003.

[30] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid Training Data Creation with Weak Supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017.

[31] Paroma Varma and Christopher Ré. Snuba: Automating Weak Supervision to Label Training Data. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 12, page 223. NIH Public Access, 2018.

[32] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013.

[33] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and Robust Automated Machine Learning. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2962–2970, 2015.

[34] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. ExploreKit: Automatic Feature Generation and Selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984. IEEE, 2016.

[35] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pages 1–10. IEEE, 2015.

[36] Airbnb Zipline, https://conferences.oreilly.com/strata/strata-ny-2018/public/schedule/detail/68114, Accessed March 22, 2021.

[37] Uber Michelangelo, https://eng.uber.com/michelangelo/, Accessed March 22, 2021.

[38] Facebook's FBLearner Flow, https://engineering.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/, Accessed March 22, 2021.

[39] H2o.AI, https://www.h2o.ai/, Accessed March 22, 2021.

[40] Nick Hynes, D Sculley, and Michael Terry. The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets. In *NIPS MLSys Workshop*, 2017.

[41] Sumit Gulwani. Automating String Processing in Spreadsheets Using Input-Output Examples. In *ACM Sigplan Notices*, volume 46, pages 317–330. ACM, 2011.

[42] Sumit Gulwani, William R. Harris, and Rishabh Singh. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105, 2012.

[43] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek Narasayya, and Surajit Chaudhuri. Transform-Data-by-Example (TDE): An Extensible Search Engine for Data Transformations. *Proceedings of the VLDB Endowment*, 11(10):1165–1177, 2018.

[44] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Jiannan Wang, and Eugene Wu. ActiveClean: An Interactive Data Cleaning Framework For Modern Machine Learning. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 2117–2120. ACM, 2016.

[45] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. Automating Large-Scale Data Quality Verification. *Proceedings of the VLDB Endowment*, 11(12):1781–1794, 2018.

[46] Schema Detection BigQuery, https://cloud.google.com/bigquery/docs/schema-detect, Accessed March 22, 2021.

[47] Mohamed-Amine Baazizi, Houssem Ben Lahmar, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. Schema Inference for Massive JSON Datasets. In *Extending Database Technology (EDBT)*, 2017.

[48] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An Open Source AutoML Benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

[49] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. CleanML: A Benchmark for Joint Data Cleaning and Machine Learning [Experiments and Analysis]. *arXiv preprint arXiv:1904.09483*, 2019.

[50] Trevor Hastie, R Tibshirani, and J Friedman. *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. Springer-Verlag, 2001.

[51] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[52] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014.

APPENDIX

A BACKGROUND ON ML CONCEPTS

We explain the ML terms and concepts relevant to this work intuitively and refer the interested readers to [50, 51] 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 [52]. 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.

B 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.

C 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.

D DATA STATISTICS

Figure 10 plots the cumulative distribution functions (CDF) of different descriptive statistics obtained by base featurization. Table 18 presents the mean, standard deviation and the maximum of the same descriptive statistics. We observe that attribute values for *Sentence*, *URL*, and *List* have more characters and words than other classes. Also, all *Numeric* sample values and 80% of the *Categorical* sample values are single token strings. Furthermore, we find that almost 90% of the *Categorical* attributes have less than 1% unique values in its columns. Interestingly, 54% of *Not-Generalizable* have either one unique value or only NaN values in their domain.

E DESCRIPTIVE STATISTICS FEATURES IN BASE FEATURIZATION

Table 6 present all the descriptive stats used for base featurization.

Table 6: List of descriptive statistics features

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

F CNN

Figure 6 (A) shows the architecture of CNN model. The layers of CNN are shown in Figure 6 (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 input a 3D 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 the number of unique characters represented in corpus. The embedding layer maps characters to dense vectors and outputs a 3D tensor of shape (*NumSamples, SequenceLength, EmbedDim*), where

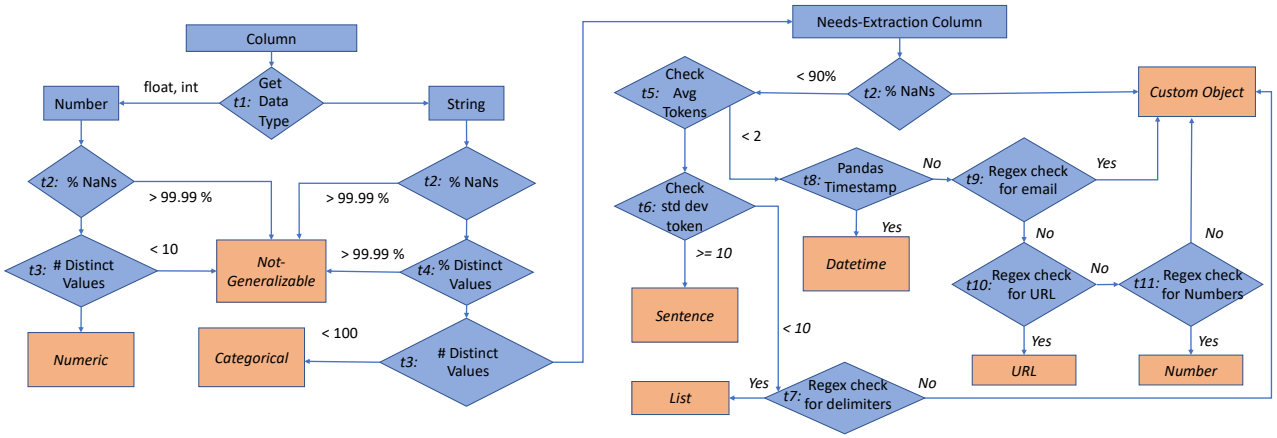


Figure 5: Flowchart of the rule-based baseline.

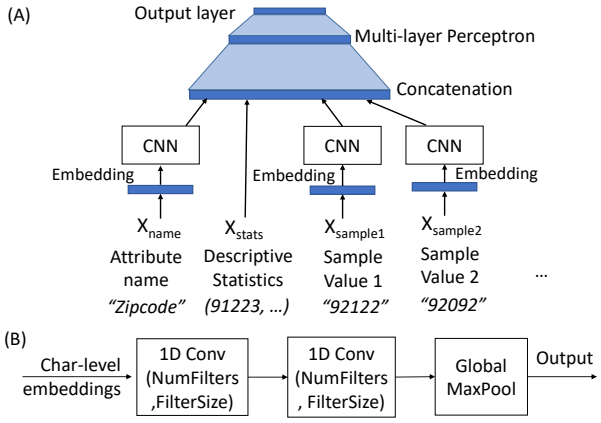


Figure 6: (A) The end-to-end architecture of our deep neural network. (B) The CNN block's layers.

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.

G RULE-BASED BASELINE

Figure 5 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 7: 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).

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

H SHERLOCK

We present the mapping between 78 Sherlock semantic types and our label vocabulary along with examples in Table 19. Note that a type from Sherlock vocabulary can map to multiple types from our label vocabulary. We use a rule-based approach to exclusively map a semantic type to automatically map it to our label vocabulary. We give two examples below to illustrate how we perform this mapping.

(1) To map *capacity*, we use the following rules in order. If the column has less than 20 unique values, then we label them as *Categorical*. We then check if we can cast the column to either *int* or *float* to label them as *Numeric*. Next, we check if the average number of space separated words is greater than 3 to map the column to *Sentence*. Finally, we use a regular expression to check if there are numbers followed or preceded by a set of commas and alphabets for *Embedded Number*, otherwise we map the column to *Categorical*.

Table 8: Binarized class-specific F1 score of different approaches on our benchmark labeled held-out test dataset. The bold fonts highlight the most accurate approach/model per class.

Feature Type	Open-source Industrial Tools				Sherlock + Rules	Baseline	Models trained on our data		
	TFDV	Pandas	TransmogrifAI	AutoGluon		Rule-based	Log Reg	CNN	Rand Forest
Numeric	0.793	0.761	0.754	0.785	0.449	0.851	0.926	0.935	**0.958**
Categorical	0.493	-	-	0.593	0.432	0.51	0.844	0.885	**0.928**
Datetime	0.641	0.935	0.624	0.915	0.843	0.217	**0.961**	0.945	0.958
Sentence	0.464	-	-	0.656	0.432	0.082	0.849	0.762	**0.883**
Not-Generalizable	-	-	-	0.495	0.079	0.303	0.732	0.727	**0.895**
Context-Specific	-	0.126	0.118	-	0.179	0.203	0.678	0.7	**0.774**

Table 9: 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.

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**	-	-	-	-	-

(2) For *duration*, we first check if the column has less than 20 unique values to map them to *Categorical*. Next, we check if we can cast the column to be either *int* or *float* to label them as *Numeric*. We perform a pandas timestamp check for *Datetime*. We check if the average number of space separated words is greater than 3 to map the column to *Sentence*, otherwise we map them to *Categorical*.

I EMPIRICAL STUDY

I.1 End-to-End Accuracy Results

Table 9 shows the train, cross-validation, and test accuracy results of all models trained on our dataset with 5-fold cross-validation methodology. Table 17 shows the confusion matrices of the rule-based approach, our Random Forest, and Sherlock.

I.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 7 present the train cross-validation, and

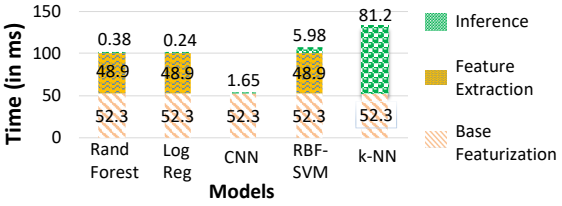


Figure 7: Comparison of prediction runtimes and breakdown for all models. Base Featurization is common for all models. Model-specific feature extraction is needed only for the 3 classical ML models.

Table 10: Number of examples of *Country* and *State* type in train and held-out test splits of our labeled dataset

	Train set	Held-out test set
Country	56	10
State	29	14

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.

I.3 Prediction Runtimes

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

I.4 Extensibility of our Benchmark

We discuss the extensibility of our benchmark and our ML-based approach in three parts as follows.

Part A. How extensible is our benchmark and ML-based approach to support new data types?

Setup. We conduct an experiment to understand the overhead of supporting additional types by expanding our 9-class label vocabulary. To illustrate the effort needed for this extension, we choose two semantic data types that are commonly used in business intelligence applications: *Country* and *State*. We extend our vocabulary to 10-classes by adding "enough" labeled examples of these two types one at a time. We then retrain our Random Forest on the extended vocabularies. We describe this process in detail with the following three steps.

(1) When creating our labeled dataset, we had annotated examples of *Country* and *State* type with *Categorical*, as they represent a discrete closed domain. Thus, we revisit our *Categorical* examples to identify *Country* and *State* types. Table 10 shows their number of examples in the train and held-out splits of our labeled data. We find that the number of examples is low to train Random Forest and draw any general conclusions for its predictions on both types.

(2) To get access to more labels, we leverage the distantly-supervised labeled data from the Sherlock data repository. This contains data columns with headers belonging to 78 semantic types [12], including *Country* and *State*. Appendix H shows these 78 types with examples. We first relabel the *Categorical* examples identified as *Country* in step 1 to a tenth class. We then add 100 randomly sampled (weakly) labeled examples for *Country* from Sherlock data repository to our held-out test set. Thus, our test split has a total of 110 *Country* examples. We add N ($N = 100$ or $N = 200$) randomly sampled *Country* examples from Sherlock data to our train set. We form two train splits where one has 100 more labeled examples for *Country* than the other. We do not alter any of the labeled examples for the rest of the 9-classes.

(3) We use the two train splits to build two Random Forest models operating on $\mathbf{X}_{stats}$ (25 descriptive stats) and $\mathbf{X2}_{sample1}$ (bigrams on the first sample value) feature sets on the 10-classes. We use the same methodology that we discussed in Section 4.1 for evaluation.

We repeat the steps 2 and 3 to extend our 9-classes with *State* and we again build two Random Forest models ($N = 100$ or $N = 200$) with the same feature set.

Results. Table 11 presents the accuracy results of the retrained Random Forest models with the extended 10-class vocabularies on the held-out test set. We find that precision and recall are already high with just 156 training examples for *Country* ($N = 100$ case). Random Forest makes most of the wrong predictions in accurately identifying abbreviations of the countries, e.g., AFG, ALB, etc. Many of such examples are wrongly predicted as *Categorical*. When Random Forest is retrained to support *State*, it achieves a recall of 78% with just 129 training examples. The recall is lower than that of *Country* because its domain is more complex since *State* includes examples of not just the United States, but also of many other countries. Moreover, we again notice that Random Forest makes wrong predictions in accurately identifying abbreviations, e.g, CA, AL, etc. *Categorical* again causes the most misclassifications, which are reduced by adding more training examples ($N = 200$ case), as we notice that recall and F1-score improve significantly relative to the $N = 100$ case.

Takeaways. We summarize the key takeaways from the above experiment below.

(1) Additional programming cost for feature type inference to support more types is negligible. We run the same scripts that we used to train the Random Forest model for our 9-classes.

(2) Labeling cost is marginal if one has access to semantic type datasets such as Sherlock data repository. Since a semantic type can be mapped to one or multiple ML feature types (Appendix H), one would need to revisit parts of our labeled data to change the labels of relevant examples. From Table 11, we find that the number of labels required to accurately identify semantic types is not too high. This is because such types are rich with information that is easier to detect with ML models. Moreover, adding more training labels will always help improve their recall. Note that there already exist complementary approaches and models for identifying semantic types. Thus, we explain why we chose not to label new examples of semantic types in Part C below.

Table 11: Accuracy of Random Forest trained using $(X_{stats}, X2_{sample1})$ feature set with the 10-class vocabulary (extending our vocabulary one at a time with either *Country* or *State*) on the held-out test set. 9-class accuracy of Random Forest with the same feature set was *0.896*.

		Adding Semantic type to our Vocabulary									
		Country					State				
		10-class Accuracy	Precision	Recall	F1 Score	Binarized Accuracy	10-class Accuracy	Precision	Recall	F1 Score	Binarized Accuracy
Adding N labeled examples to our training dataset	N=100	0.882	0.927	0.911	0.919	0.991	0.875	0.967	0.781	0.864	0.986
	N=200	0.881	0.892	0.972	0.93	0.992	0.875	0.942	0.851	0.894	0.988

Table 12: Ablation study of our feature set by dropping the three type-specific features one at a time from X_{stats} with Logistic Regression and Random Forest on our held-out test set. The bold font marks the cases where the corresponding type-related feature is dropped.

Logistic Regression										
Feature Set	9-class Accuracy	Datetime			URL			List		
		Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
[X_{stats}, $X2_{name}$, $X2_{sample1}$]	0.853	0.945	0.972	0.958	0.969	1	0.984	0.875	0.807	0.84
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – list-specific feature from X_{stats}	0.855	0.95	0.95	0.95	0.969	1	0.984	**0.913**	**0.807**	**0.857**
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – url-specific feature from X_{stats}	0.853	0.95	0.95	0.95	**0.967**	**0.906**	**0.936**	0.915	0.827	0.869
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – datetime-specific feature from X_{stats}	0.849	**0.936**	**0.936**	**0.936**	0.969	1	0.984	0.875	0.807	0.84
Random Forest										
[X_{stats}, $X2_{name}$, $X2_{sample1}$]	0.922	0.951	0.972	0.961	0.969	0.969	0.969	1	0.769	0.869
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – list-specific feature from X_{stats}	0.915	0.958	0.972	0.964	0.969	0.969	0.969	**0.951**	**0.75**	**0.839**
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – url-specific feature from X_{stats}	0.913	0.951	0.972	0.961	**0.969**	**0.969**	**0.969**	0.952	0.769	0.851
[X_{stats}, $X2_{name}$, $X2_{sample1}$] – datetime-specific feature from X_{stats}	0.914	**0.938**	**0.957**	**0.947**	0.969	0.969	0.969	0.951	0.75	0.839

(3) There exist negligible or no feature engineering cost. We do not add or remove any descriptive statistic feature (discussed in Appendix E) to support the additional types. Table 11 shows that we can get high accuracy for the new types even with the same feature set that we used for the 9-classes. Thus, our features provide meaningful signals even for the newly added classes. We provide more evidence of why our feature set is robust in Part B below.

Part B. Why our featurization is robust and broadly applicable to cover any semantic type?

We present the complete list of descriptive statistics that we use for our base featurization (discussed in Section 2.3) in Appendix E below. We designed three custom features to help our ML models identify *Datetime*, *URL*, and *List* types. These features include two regular expression-based boolean checks to identify *URL* and *List*, and one Pandas timestamp check to identify *Datetime*. For instance, the regular expression based boolean feature for *List* is used to identify if the string contains a series of characters separated by a delimiter such as ; | , etc. The rest of the descriptive statistics features do not apply exclusively to a particular class from our 9-classes, but they are generically applicable to cover any class.

Setup. To understand if our featurization is robust and broadly applicable, we drop the three custom type-specific features one at a time from X_{stats} (our descriptive statistics feature set). We retain other features $X2_{name}$ (bigrams on the column name) and $X2_{sample1}$ (bigrams on the first sample value) as it is. We then

Table 14: Accuracy results of Sherlock to identify semantic types on our labeled data. OurRF denotes our best performing Random Forest on our held-out test dataset.

	True Semantic Type		
	Country	State	Gender
#Examples in Test Set	10	14	6
#Examples predicted correctly by Sherlock	5	9	5
Recall of Sherlock	**50%**	**64.3%**	**83.3%**
#Examples predicted Categorical by OurRF	10	14	6
Given Categorical Prediction from Our Random Forest			
#Examples predicted correctly by Sherlock	5	9	5
Recall of Sherlock	**50%**	**64.3%**	**83.3%**

Table 13: Examples of *Categorical* type from our labeled dataset along with semantic type predicted by Sherlock. Our Random Forest predicts all of them as *Categorical*.

#	Attribute Name	Type predicted by Sherlock	Total Values	# Distinct Values	Sample 1	Sample 2
A	ad744	grades	9597	3	-99	0
B	ad7125	ranking	9597	3	0	1
C	applicant_race_name_1	type	466566	7	White	Asian

retrain our ML models and validate the held-out test accuracies of *Datetime*, *URL*, and *List* types. If the drop in accuracy of the three classes is significant, then it would imply the type-specific feature being predominantly important. On the other hand, if the drop is marginal then it would imply the robustness of the rest of the features to provide meaningful signals for these classes.

Results. Table 12 presents the ablation study results on the held-out test set with the metric being the 9-class accuracy, precision, and recall of the three classes. We choose model type family from both extremes of bias-variance tradeoff: Logistic Regression and Random Forest. We find that 9-class accuracy drops only negligibly when the three features are dropped one at a time for both models. Moreover, their precision and recall remain either unchanged or drops marginally with the newly trained model for almost all of the cases. The only cases where the drop appears significant is with Random Forest on the precision of *List* and Logistic Regression on recall of *URL*. This is because the held-out test set contains 32 and 52 examples of *URL* type respectively. Thus, misclassifying just two to three examples more leads to a 5-10% drop in precision/recall.

Takeaways. We conclude from this analysis that our featurization is robust and broadly applicable to cover any new class. This is because the signals that we extract from the raw column are column name (high-level signal), column values (low-level signals), and 25 descriptive stats (aggregate signals over the column). Thus, these standard features would provide meaningful signals even for a newly added class. Nevertheless, we do not claim that our featurization is comprehensive and unbeatable. We have publicly released raw data files as part of our benchmark and we invite contributions to devise new featurizations (see Section 6.1). Thus, our feature sets can be easily expanded.

Part C. Why it may be more feasible to leverage prior work such as Sherlock directly for identifying semantic types?

If one chooses to explore semantic types on top of our label vocabulary, then they can leverage complementary approaches such as Sherlock in conjunction with our ML models, rather than labeling new examples of such types for our models. To illustrate this, we now discuss how using Sherlock can complement our ML models to help identify fine-grained semantic types. We study this in the context of *Categorical* examples of our labeled data by leveraging the 78 class semantic vocabulary of Sherlock (Appendix H).

Table 13 shows three examples of *Categorical* type columns from our labeled data along with semantic type predicted by Sherlock. *Note that we don't need to distinguish the semantic type of these columns to be used by ML, because ultimately they would be used as a Categorical feature.* Admittedly, detecting semantic types can help one discover and leverage auxiliary datasets. However, this goal is orthogonal to our focus. We focus on identifying ML feature types that dictate how columns will be consumed by ML, especially in AutoML environments. Other goals such as schema matching, data discovery, and data exploration are orthogonal to our focus. This makes semantic types different and complementary to our ML feature types.

We first find that the notion of what exactly constitutes a semantic type in Sherlock is ambiguous. We give two examples below to illustrate why it is not possible to unambiguously identify the true semantic type for every single *Categorical* example.

(1) Consider example C in Table 13, which semantic type does `applicant_race_name_1` belongs to? Is it *class*, *category*, *classification*, or *type*? Sherlock predicts it as *type*, but its not possible for us to verify its true semantic type, even after inspecting the column description in the source metadata. However, our Random Forest predicts it as *Categorical*, which is how it will be used by ML.

(2) It is not trivial to determine the semantic type even for columns with non-sensical names such as `ad744` and `ad7125` (example A and B in Table 13). These columns are drawn from one source CSV file and they represent the same real-world entity ("Adaptation Climatic Variation") but for different seasons. Thus, they would belong to the same semantic type, but Sherlock gives random and different predictions for them. However, our Random Forest again predicts both of them as *Categorical*.

Considering this, we limit our analysis to a subset of *Categorical* columns from our labeled data where we can unambiguously identify their true semantic type from Sherlock's label vocabulary: *Country*, *State*, and *Gender*.

Results. Table 14 shows the number of examples of the three semantic types in our held-out test set. We showcase two approaches on these examples: (1) We run Sherlock independently. (2) We first get predictions from our best performing Random Forest (OurRF),

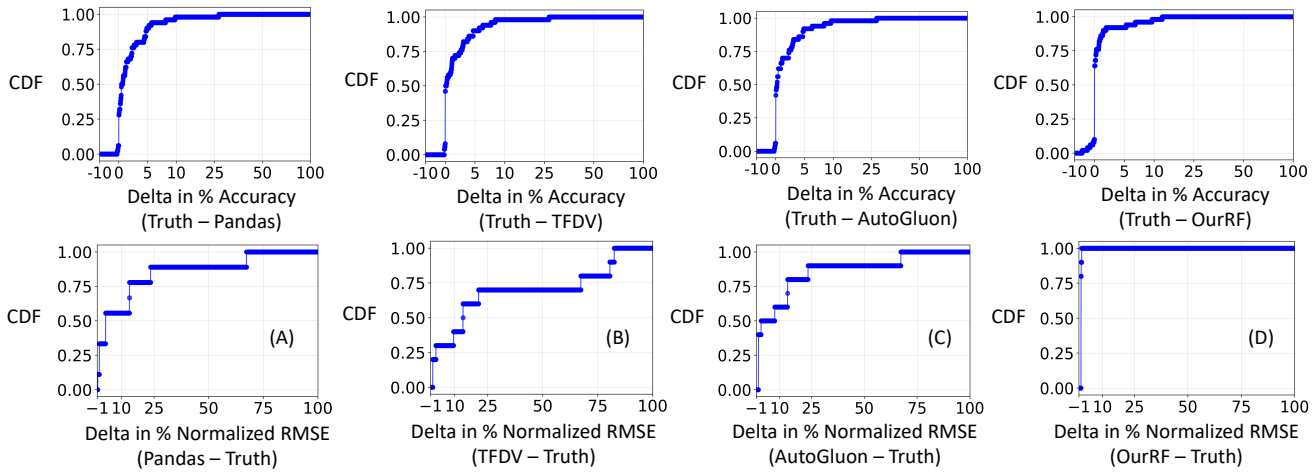


Figure 8: CDFs of the differences between downstream model performance with different approaches for feature type inference relative to performance with perfect feature type inference (Truth). The first and second row show CDFs on 50 downstream classification models and 10 regression models respectively. This includes both the downstream Linear model and downstream Random Forest model with (A) Pandas (B) TFDV (C) AutoGluon, and (D) Our best performing Random Forest (OurRF). For regression models, we normalize the RMSE relative to the corresponding true RMSE.

followed by running Sherlock on top of OurRF's *Catgeorical* predictions. Table 14 also presents the accuracy results of Sherlock on the three semantic types with both these approaches. We find that the recall of Sherlock with both approaches is identical on *Country*, *State*, and *Gender* type. Thus, Sherlock is truly complementary and it can be used independently or together with our ML models to identify the semantic types.

I.5 Downstream Benchmark Suite

I.5.1 Results. Table 5 shows the exact performance gap on the 30 downstream tasks, given the feature types from Random Forest vs. perfect predictions (denoted as "Truth"). Table 4 shows its result summary. For instance, on *Mfeat* dataset, the delta drop in percentage accuracy with our best performing Random Forest (OurRF) compared to the perfect prediction (Truth) on downstream Logistic Regression model is 2.7. As a second example, on *Car Fuel* dataset, the RMSE increase with OurRF compared to Truth on downstream Random Forest is 0.03.

Figure 8 presents the CDF of the magnitude of difference in downstream model performance with different ML feature type inference approaches compared to Truth. We find that the drop in percentage classification accuracy with OurRF relative to the Truth is less than 0.88 for 75% of the downstream models. In contrast, the 75th percentile delta drop in percentage accuracy for Pandas, TFDV, and AutoGluon relative to the Truth is 6.9, 7.7, and 6.9 respectively for the same downstream classification models. Moreover, the median increase in percentage normalized RMSE relative to truth is 0 with OurRF, in contrast to the median of 8.3, 13.7, 4.5 for Pandas, TFDV, and AutoGluon respectively. Thus, existing tools underperform the truth (perfect predictions of feature types) by a much higher magnitude than OurRF.

Overall, we find that OurRF underperforms the truth on 20 out of 60 downstream tasks because of wrong type inference. This shows that type inference is a critical task and improving its accuracy can directly improve the performance of the downstream model. We explain the results further below.

a. Why does wrong type inference hurt downstream accuracy? We 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, one 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. In contrast, AutoGluon classifies all of them incorrectly. Other tools like TFDV and Pandas do not even support *Not-Generalizable* in their vocabulary. Thus, the drop is much larger for the prior tools. We observe the same pattern across many datasets like *Pokemon*, *President*, and *Car Fuel*.

b. Why is outperforming truth not necessarily beneficial?

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.

I.5.2 Downstream Model Performance with Double Representation. In this section, we give multiple representations of the

Table 15: Number of downstream datasets (out of 25) where tools with both numeric and one-hot encoded representation of integer columns outperform the tool baseline with an exclusive type specific representation, underperform truth, OurRF, NewRF, or the best performing tool. NewRF is the Random Forest trained on our labeled dataset adapted to produce multiple feature representations.

	Logistic Regression				Random Forest			
	PD	TFDV	AGL	NewRF	PD	TFDV	AGL	NewRF
Underperform truth	20	18	18	13	20	19	16	13
Underperform tool baseline	3	1	3	6	11	10	8	5
Outperform tool baseline	11	7	8	7	11	8	7	7
Outperform tool baseline but worse than NewRF	6	2	4	-	9	5	5	-
Best performing tool for a dataset	6	8	7	18	5	6	7	18

column at the same time to the downstream model and study if this can help recover the gap in performance caused by wrong type inference. We study this in the context of *Numeric* vs *Categorical* dichotomy of the integer columns of our downstream datasets. Thus, with existing AutoML tools, instead of routing integer columns predicted as *Numeric* to an exclusive numerical representation and *Categorical* columns to an exclusive one-hot encoded representation, we route integer column to both representations regardless of the predicted feature type.

Note that representation of columns in multiple ways for the downstream models is completely orthogonal to whether correct feature types are known or not. Multiple representations can be performed even when the correct feature types are known. Thus, we adapt OurRF to support the double representation of integer columns by leveraging the class confidence probabilities of the prediction. We first use a threshold-based rule to check if the prediction made by OurRF is confident enough to route the column to an exclusive feature representation corresponding to the predicted type. We set the threshold to the probability value of 0.4, i.e., the confidence of the prediction should be at least twice the random guessing accuracy on the integer column. When the class confidence probability is below the threshold, we route the column to both *Numeric* and *Categorical* representation. We denote this adapted Random Forest with "NewRF."

Table 15 shows summary stats on how the tools supporting double representation of integer columns perform relative to the same tools supporting an exclusive type-specific representation on our 25 downstream benchmark datasets. We observe that the accuracy of the downstream models obtained with existing AutoML tools and NewRF do improve with the double feature representation on some datasets. We find that the amount of accuracy improvement is typically higher for the downstream linear model than the downstream Random Forest. e.g., an average 3.4% and up to 29.6% on the linear model and of an average 1.3% and up to 21% on Random Forest with TFDV. Despite this, existing AutoML tools underperform NewRF on most datasets with both downstream models. In addition, NewRF underperforms the truth on 26 downstream models. In contrast, Pandas, TFDV, and AutoGluon underperform for significantly more models: 40, 37, and 34 respectively. Overall, we notice that NewRF performs worse than the best performing tool for only 14 out of 50 downstream models. Thus, accurate inference of feature types is again critical to building accurate downstream models.

One can even consider exploring different subsets of feature representations for integer columns. However, this explodes exponentially in the number of columns, which will waste a lot more runtime and resources, while still having the same interpretability issues. Note that the discussion of how feature types should be represented is completely orthogonal to our focus. Knowing a feature type correctly will always provide more information, which can be used to build better featurization schemes.

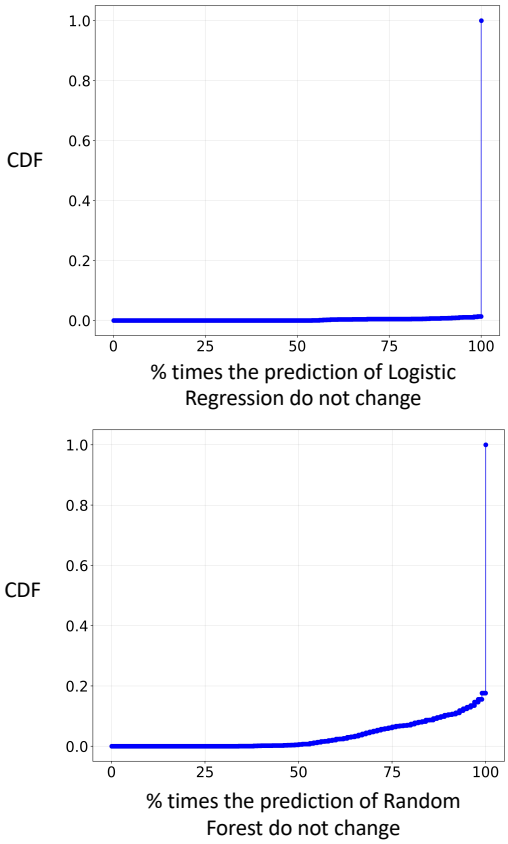


Figure 9: CDFs of the % of times the predictions of Logistic Regression and Random Forest do not change after perturbing columns from the held-out test data. Number of simulation runs are set to 100 for all held-out test set columns.

I.6 Robustness of ML models

We now analyse the robustness of our ML models' predictions to different samples of the column. Note that our base features including five descriptive stats are generated by leveraging five random sample values from the column. Thus, a different set of column values can potentially alter the prediction of our ML models.

We perform a Monte Carlo-style study where we randomly perturb every column from our held-out test dataset 100 times. In each run, we obtain the first five non-empty distinct sample values

Table 16: Summary statistics of the percentage of times the predictions of Logistic Regression and Random Forest do not change after perturbing columns from the held-out test data.

nth Percentile (over held-out test set)	% times the prediction of do not change	
	Logistic Regression	Random Forest
50	100	100
20	100	100
10	100	89
5	100	70
1	100	54
0.1	95	39
0.01	57	35

from the column. We then use our ML models trained on X_{stats}, $X2_{name}$, and $X2_{sample1}$ to get 100 predictions for every column of our held-out test data. We finally count the percentage of times (out of 100) the predictions of the ML models remain the same as the one on the original unperturbed column. Figure 9 presents the CDFs of these counts on the Logistic Regression and Random Forest models across the held-out test set. Table 16 shows the different nth percentiles of the % of times the predictions of the two models change after perturbation across the held-out test dataset. We find that both Logistic Regression and Random Forest are really robust to the randomness of sampling introduced by our base featurization process. For instance, on 5% of the test examples, the prediction of Random Forest changed only 30% of times. Moreover, we observe that Logistic Regression is more robust to variability in sample values than Random Forest.

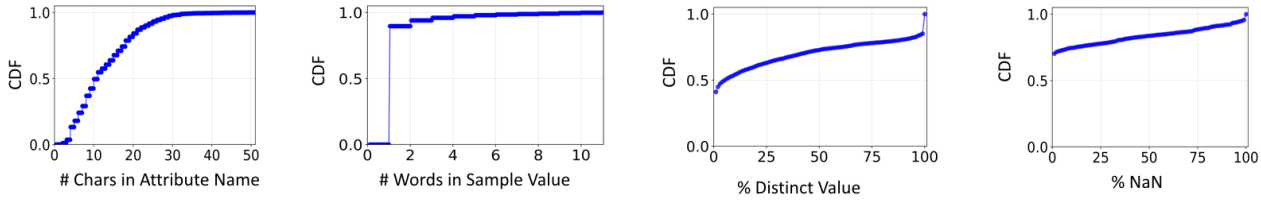
Table 17: Confusion matrices (actual class on row and predicted class on column) of (A) Rule-based baseline (B) Random Forest, and (C) Sherlock.

(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

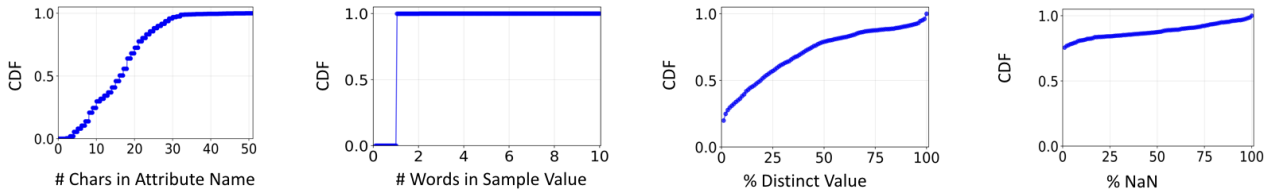
(B)	Numeric	Categorical	Datetime	Sentence	URL	Embedded Numbers	List	Not-Generalizable	Context-Specific
Numeric	696	3	0	0	0	0	0	2	6
Categorical	12	431	0	4	0	0	0	1	9
Datetime	0	2	137	0	0	2	0	0	0
Sentence	0	3	0	83	0	0	0	3	3
URL	0	2	0	0	30	0	0	0	0
Embedded Numbers	0	5	1	0	0	92	0	0	1
List	0	1	0	3	0	5	43	0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

(C)	Numeric	Categorical	Datetime	Sentence	URL	Embedded Numbers	List	Not-Generalizable	Context-Specific
Numeric	254	334	5	10	0	0	0	0	104
Categorical	63	323	1	62	0	5	0	0	3
Datetime	1	25	113	2	0	0	0	0	0
Sentence	0	31	0	51	0	2	0	0	8
URL	0	25	0	0	0	0	0	1	6
Embedded Numbers	1	59	0	1	0	36	0	2	0
List	0	17	0	24	0	1	0	1	9
Not-Generalizable	59	123	6	11	0	2	0	9	5
Context-Specific	46	101	2	5	0	4	0	0	32

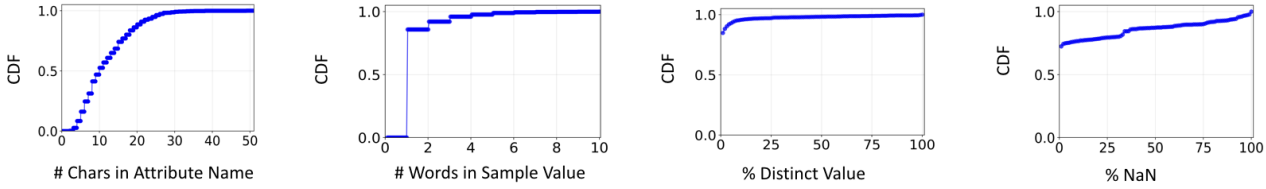
Overall



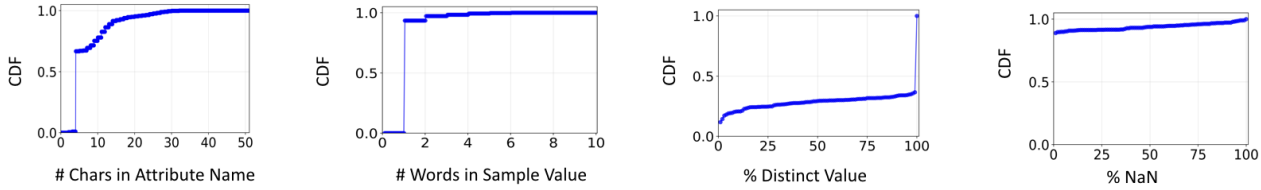
(1) Numeric



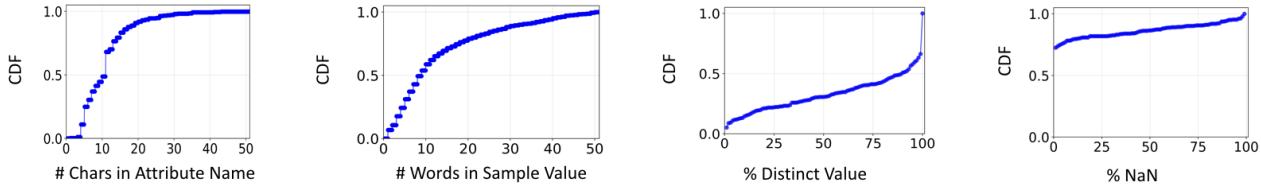
(2) Categorical



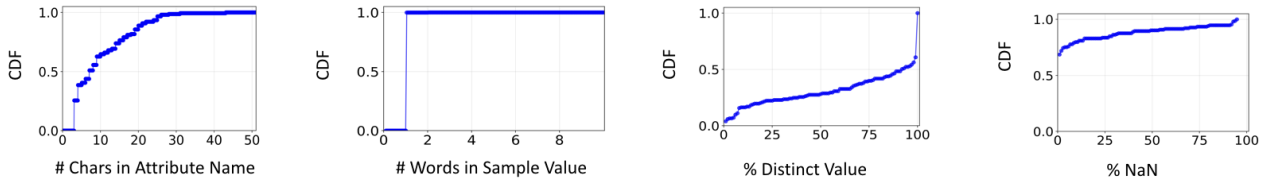
(3) Datetime



(4) Sentence



(5) URL



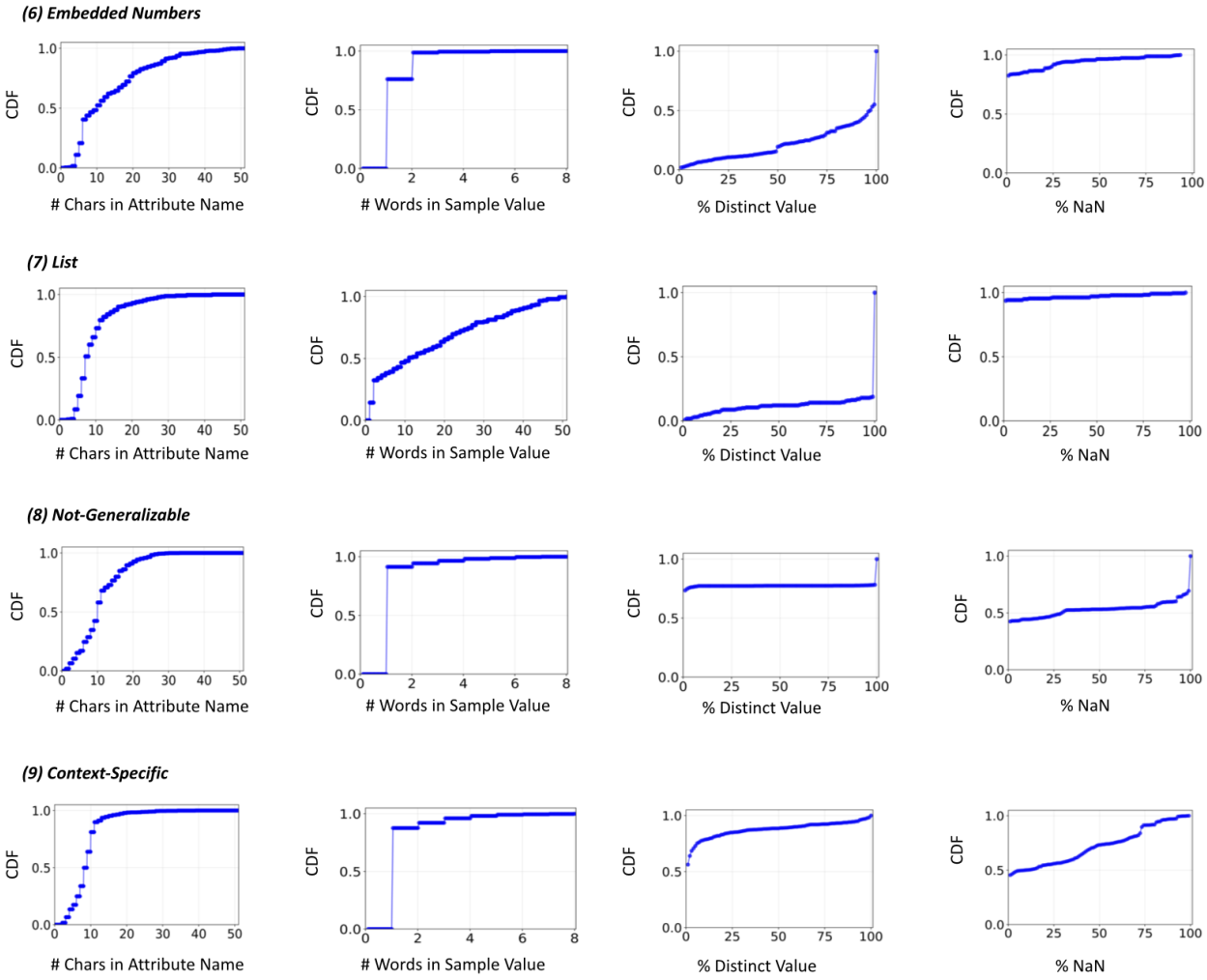


Figure 10: Cumulative distribution of different descriptive statistics features.

Table 18: Average, standard deviation, and maximum value 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 19: Mapping of Sherlock's 78 semantic types with our Label Vocabulary. The examples shown below are taken from Sherlock's data repository.

Sherlock Type	Example Values	Our Label
address	[184 New York Ave, …]	Context-Specific
affiliate	[DNC Services Corp, …]	Categorical
affiliation	[Government of Moldova, …]	Categorical
age	[27, 57, 85, …]	Numeric
	[12M, 18M, 24M, …]	Embedded Number
	['0/3M', '3/6M', '6/9M']	Categorical
album	[Need for Speed (Original Motion Picture Soundtrack), …]	Context-Specific
area	[512, 192, 128, …]	Numeric
	[United States, Alaska, Oregon, …]	Categorical
artist	[John Paul Joans, …]	Context-Specific
birth Date	[‘Jun 17, 1970’, …]	Datetime
birth Place	[‘Montreal, QC, CAN’, …]	Context-Specific
brand	[Walmart, Target, …]	Categorical
capacity	[52000, 50000, …]	Numeric
	['76,125', ‘76,225’, …]	Embedded Number
	['2 cup', '4.5 cup', '7 cup']	Categorical
	[Additional fuel oil required to fill all tanks to 95% (emergency), …]	Sentence
category	[Singlespeed Men, …]	Categorical
city	['Pasadena, CA, 91050’, …]	Context-Specific
class	['class 1', 'class 2', 'class 3', 'class 4', 'class 5']	Categorical
classification	['geograph', 'supplemental', 'supplemental']	Categorical
club	['SPM', 'SMS']	Categorical
code	['AS', 'FPAY']	Categorical
	[37, 38, 39, …]	Not-Generalizable
collection	['SS-Easy Books', 'Fiction, Adult - Non-Floating', ‘SS- Easy Books']	Categorical
	['CDs, Adult Musical’, …]	List
command	['Close Tab', 'Close Window’, …]	Categorical
	['show ip device tracking { all | interface interface-id | ip ip-address | mac imac-address}’, …]	Sentence
company	['General Motors’, …]	Context-Specific
component	['Unknown', 'Study view', 'Query']	Categorical
continent	[EU, AS]	Categorical
country	['Argentina', 'Australia’, …]	Categorical
county	['WSHGTN’, …]	Categorical
creator	['Andrew Jones’, …]	Context-Specific
credit	[0,1, 3, 6]	Categorical
currency	['Australian Dollar', 'Brazilian Real’, …]	Categorical
day	['Fri', 'Mon']	Categorical
	['2015-05-01', '2015-05-02’, …]	Datetime
depth	[62.5, 62.4, 62.9, …]	Numeric
	['35mm', '40mm’, …]	Embedded Number
description	['Automatically close braces, brackets and parentheses’, …]	Sentence
director	['Heidi Ewing', 'Rachel Grady']	Context-Specific
duration	[238, 276, …]	Numeric
	[184, 240, 360]	Categorical
	['3:21', '3:30', '3:40', '3:53', '3:12', '3:51']	Datetime
	[‘Rest of Season ? Projected stats for the rest of the season’, …]	Sentence
Education	['Ph.D.', 'MA', 'MA', 'B.A.']	Categorical

Sherlock Type	Example Values	Our Label
family	['Baptist', 'Pentecostal', ...]	Categorical
file Size	['4824', '5351', '5397', ...]	Numeric
	['1,276 kb', '232 kb']	Embedded Number
format	['AC', 'Classic Hits', 'Urban AC', ...]	Categorical
gender	['Female', 'Male']	Categorical
genre	['Action', 'Adult', ...]	Categorical
	['Poetry, Fiction, Creative Nonfiction', 'Fiction', ...]	List
Grades	['K - 5', 'K - 5', 'PK - 9']	Categorical
industry	['Communications', 'Fabricated Goods', ...]	Categorical
isbn	['isbn 0688156819', 'isbn 015205300X', 'isbn 1563898586']	Categorical
	['isbn 0307348245', 'isbn 188896314X', ...]	Not-Generalizable
jockey	['M J Odendaal', 'A Delpech', ...]	Context-Specific
language	['Japanese', 'French', 'Italian']	Categorical
location	['VBC Arena', 'East Hall', ...]	Context-Specific
manufacturer	['Discraft', 'Innova', 'Innova']	Categorical
name	['Matt Kemp', 'Prince Fielder', ...]	Context-Specific
nationality	['British', 'Hungarians', 'Germans']	Categorical
notes	['Details unclear, but probably worked by a Diesel Multiple Unit (DMU) of some description.', ...]	Sentence
operator	['Abellio London', 'London United', 'Abellio London']	Categorical
order	['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun']	Categorical
	[1,2,3,4,5, ...]	Context-Specific
organisation	['Ministry of Justice', "Her Majesty's Court Service (HMCS)", ...]	Context-Specific
origin	['Brazil', 'China', 'United States']	Categorical
Owner	['Mr P Naylor', 'Mr C A Howard', ...]	Context-Specific
Person	['Elizabeth Partridge', ...]	Context-Specific
plays	[90, 57, 33, ...]	Numeric
	['1,846', '1,506', ...]	Embedded Number
position	[8.0, 2.0, 3.0, 9.0, ...]	Numeric
	['Director', 'Chief Merchandising Officer', ...]	Categorical
product	['Norwegian Cod Liver Oil', 'Norwegian Cod Liver Oil Cherry', ...]	Context-Specific
publisher	['Activision', 'Bethesda Softworks', ...]	Context-Specific
range	['Last week', 'Last month', 'Last 3 months']	Categorical
	['15,000.00', '40,000.00', '140,000.00', ...]	Embedded Number
rank	[1, 2, 3]	Categorical
	['2,636,246', '3,407,650', '5,890,835']	Embedded Number
ranking	[27.8, 24.0, 23.8, ...]	Numeric
	[1, 5, 8]	Categorical
	['RB - #1', 'RB - #11', 'RB - #3', ...]	Embedded Number
region	['Chubu', 'Chugoku', ...]	Categorical
religion	['Catholic', 'Evangelical Protestant', ...]	Categorical
requirement	['See the Application Server Requirements.', ...]	Sentence
result	[7.43, 7.99, 12.68]	Numeric
	['Loss', 'Win']	Categorical
	['Blue mages are able to learn the abilities of monsters as spells.', ...]	Sentence
sales	[1456, 23, 981, ...]	Numeric
	['1,652', '6,190', '4,841', '3,005', '15,688']	Embedded Number
service	['', 'BLND', 'DEAF', 'WCHR', ...]	Categorical
sex	[M, F, ...]	Categorical
species	['Iris-setosa', ...]	Categorical
state	['MA', 'NY']	Categorical
status	['ONSCENE', 'DISPATCHED']	Categorical

Sherlock Type	Example Values	Our Label
symbol	['circle', 'square', 'diamond', 'cross']	Categorical
team	['HOU', 'CCW', 'CCW']	Categorical
team Name	['Designated Drinkers', 'Dream Killers', …]	Context-Specific
type	['range', 'eq_ref', 'eq_ref', 'eq_ref']	Categorical
weight	[32.3, 32.9]	Numeric
	['95 lbs.', '106 lbs.', …]	Embedded Number
year	[1972, 1979, 1968]	Categorical
	['May-07', 'May-08', 'May-09', '10-May']	Datetime