Towards Learning based Feature Type Inference

Vraj Shah Premanand Kumar Kevin Yang Arun Kumar
University of California, San Diego
{vps002,p8kumar,khy009,arunkk}@eng.ucsd.edu

ABSTRACT

The paradigm of AutoML has created an opportunity to enable ML for the masses. While many works have looked into automated model selection or hyper-parameter search in AutoML, little work has studied how good automated data prep is. Formalizing data prep tasks and creating benchmark labeled datasets can not only help automate them but can also help objectively validate and improve AutoML platforms. In this work, we automate and objectively benchmark a critical data prep task: *ML feature type inference*. Datasets are typically loaded as files into AutoML platforms, which creates a semantic gap between column data types (e.g., strings, numbers, etc.) and ML feature types (e.g., numeric or categorical). e.g., *ZipCode* stored as numbers are not *Numeric* features for ML. In contrast to existing rule-based or syntax-based approaches, we present the first supervised ML-based approach to type inference. We manually annotate and construct a labeled dataset with 9251 examples. We mimic human-level intuition behind labeling into the ML models by extracting relevant signals from the raw data files. Our empirical evaluation shows that our applied ML approach delivers a large 30% lift in identifying numeric features compared to existing prominent industrial and open source tools. In addition, with our approach the accuracy of the downstream model improves significantly, even more than 40% on some datasets. Google is collaborating with us to adopt our best performing models into TDFV, a production ML platform, to improve its automated type inference. Our labeled dataset and models are released in a public repository with a leaderboard. They are available for download from: `https://adalabucsd.github.io/sortinghat.html`.

1 INTRODUCTION

The paradigm of automated machine learning (AutoML) is beginning to help democratize machine learning for the masses [15]. Cloud vendors have released AutoML platforms such as Google's Cloud AutoML [11] and Salesforce's Einstein [22] that build ML models on millions of datasets from thousands of small-and-medium enterprises automatically. While automation of several components of the ML workflow such as model selection and hyper-parameter search are well studied [9, 15], little attention has been paid to understanding how good is the automated data prep, an equally important component of AutoML systems.

Data prep is particularly challenging on structured data, perhaps the most commonly analyzed form of data in practice [24]. The very first and a critical step in data prep for ML over structured data is *ML feature type inference*, as shown in Figure 1. Features could be *numeric*, *categorical*, or *something else*. Depending upon the inferred types, suitable data transformation steps may be applied, and then the downstream model is built. For instance, if a column is inferred with type *Timestamp*, then several useful features such as day, month, and year can be extracted automatically to build the downstream model. Thus, the accuracy of ML feature type inference is critical for the downstream model's accuracy because the AutoML platform can then deal with features accordingly.

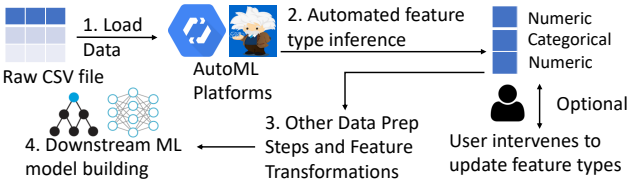

Figure 1: Typical workflow in the AutoML platforms.

There exist some open-source tools such as Tensorflow Data Validation (TFDV) in TensorFlow Extended [3] and TransmogrifAI in Einstein [26] that automate this task. However, they rely on ad hoc syntactical and rule-based pipelines and are thus often limited in accuracy. For instance, consider an illustrative dataset for a common ML task, customer churn prediction in Figure 2. TFDV and TransmogrifAI wrongly calls many *Categorical* features with integer values as *Numeric*, e.g., *ZipCode*. Moreover, *Income* is inferred as *Categorical* even though it has number embedded. Such issues can lead to loss of information and can potentially reduce the accuracy of the downstream model, or even cause it to fail in some scenarios.

Challenge: Semantic Gap and Scale of AutoML. Datasets are typically loaded from RDBMSs, data lakes, or filesystems as flat CSV or JSON files into the AutoML platforms. Automatic ML feature type inference is hard because of the *semantic gap between feature types and attribute types* in databases/files. The latter tells us the syntactic datatypes of columns such as integer, real, or string. This semantic gap often leads to nonsensical results. Moreover, conversations with AutoML platform engineers at Google and SalesForce revealed that their tools are used on over tens of thousands of datasets, adding up to millions of features in production settings. Forcing users to manually annotate features can lead to a tedious, slow, and error-prone process that also violates the promise of end-to-end AutoML. Thus, AutoML platform engineers prefer ever more accurate automation of feature type inference. Overall, it is a pressing problem to close the semantic gap and improve the accuracy of automated feature type inference in AutoML platforms.

Example. Consider Figure 2 again. We immediately see two major issues caused by the semantic gap. (1) Attributes such as *States*, *Gender*, *Income*, and *Churn* are stored as strings, but not all of them are useful as categorical features. For instance, *Income* is actually numeric but some of its values have a string prefix. (2) Attributes such as *CustID*, *Age*, *ZipCode*, and *XYZ* are stored as integers, but only *Age* is useful as numeric. *CustID* is unique for every customer, hence it can not be generalized for ML. Inspecting only the column *XYZ*, it is difficult to decide if the feature is numeric or categorical. *ZipCode* is categorical, even though it is stored as integers. In fact,

CustID	Gender	Age	ZipCode	XYZ	Income	States	Churn
1501	F	25	92092	005	USD 15000	CA/WA/NY	Yes
1704	M	34	78712	003	25384	NY/AL	No

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

this issue is ubiquitous in real-world datasets, since categories are often encoded as integers. e.g. item code, state code, etc.

Our Approach. In contrast to prior rule-based or syntax-based systems, we propose a systematic ML-based approach to semi-automate and objectively quantify the feature type inference task. The key limiting factor to achieve this nature of automation is not ML algorithmic advances, but availability of *large high quality labeled datasets*. For instance, the availability of the ImageNet dataset spurred several advances in computer vision today [21]. Creating labeled datasets and new ML models for this task will not only automate it but will also contribute to objectively benchmarking AutoML platforms. Of course, data prep has many other important steps too. We plan to formalize other ML data prep tasks and cast them as applied ML tasks in future work.

In this paper, we cast ML feature type inference as a multi-class classification problem and use ML models to bridge the semantic gap. We build a tool called SortingHat that uses our best performing ML models to automatically infer feature types from a raw CSV file. To make this possible, our work makes the following contributions.

1. Labeled Dataset. There exists no publicly available dataset for this task. Creating labeled data for our task is challenging because of two reasons. (1) An example for this task is an entire feature column in a raw data file. So, even a file with 1M records and 10 columns has only 10 examples! Hence, a lot of manual work needs to be done to collect raw data files. (2) There is often not enough information in just the data file to identify the class (*numeric* or *categorical*) correctly. e.g. column *XYZ*. Thus, we create one more class called "others" to capture the different variety of columns: ones with messy syntax, ones that are non-generalizable, and the ones that are "hard" to make a judgement. We manually label 9251 columns from the real data files we collected into one of the three classes. *The labeling process took about 75 man-hours across 4 months.*

2. Featurization and 3. ML models. To identify the feature type of a raw column, a human data scientist would look at the column name, some sample values in the column, and even descriptive stats about the column, e.g., number of NaNs or distinct values. For instance, just by reading the attribute name *ZipCode*, an interpretable string, a human can tell its feature type is categorical. We replicate this human-level intuition into ML models by extracting signals from the raw data files that a typical data scientist may look at. We summarize the signals in a feature set, which we use to build various ML models: logistic regression, support vector machine, Random Forest, k-nearest neighbor (k-NN), and character-level CNN.

Empirical evaluation and analysis. We first compare our models against existing public tools that all happen to be either rule-based or syntax-based: Python Pandas, TFDV, and TransmogrifAI. We found that our ML models deliver a 30% lift in accuracy compared to these tools for identifying numeric features among the attributes. We then evaluate and compare different ML models on our dataset. Overall, Random Forest outperforms the other models and achieves the best 3-class accuracy of 91.9%. The CNN and the k-NN perform comparatively with 91% accuracy. We perform an ablation study on our ML models to characterize what types of features are useful. We also analyze and intuitively explain the behavior of Random Forest and CNN by considering their predictions on different types of column values such as integers, float, dates, etc. We analyze the effect of the accuracy of ML feature type inference task on downstream models with 15 real-world datasets. Our results show that SortingHat delivers higher downstream model's accuracy, even up to 55% compared to prior tools. Finally, we release a repository containing our labeled dataset and trained ML models and announce a leaderboard for community contributions.

In summary, our work makes four key contributions as follows:
1. Formalization of a key data prep task. There exist no objective data/benchmarks to quantify how good is data prep automation on AutoML platforms. Feature type inference is the first and an unavoidable step in data prep. The key technical contribution of our paper lies in formalizing and standardizing this critical task as an ML prediction task and creating a benchmark labeled dataset. This will enable an objective progress measurement, akin to ImageNet's role in vision, except we go further because data prep is still an ill-defined space. We present the first supervised ML-based approach to type inference.

2. Utility of our labeled dataset. Using benchmark datasets we created, we show that even with off-the-shelf ML models and employing standard featurization routines, we outperform the state-of-the-art industrial-strength open-source tools such as TFDV and TranmogrifAI. The semantic gap makes type inference hard for most existing ML platforms since they use ad hoc rule-based approaches. Thus, they yield low accuracy. A key novelty of this work is that it shows for the first time that even standard ML models (trained on our data) can be more accurate for type inference.

3. Effect on downstream model. This work offers an initial evidence that downstream model accuracy can benefit from accurately determining the feature types. Our experiments with 15 real-world datasets show that SortingHat can lead to a significantly higher downstream model's accuracy than the existing open-source tools.

4. Real-world impact. Google is collaborating with us to adopt our best performing models into TDFV to improve their inference of categorical type. In addition, all of our code, models, and data are available publicly [1]. We release a public competition and leaderboard on our labeled dataset in order to invite contributions to create/augment datasets, better featurization schemes, and models.

2 APPROACH

2.1 Assumptions and Scope

We focus on relational data, perhaps the most commonly analyzed form of data in practice [24]. Such datasets are typically stored with schemas in database systems or as "schema-light" files (CSV, JSON, etc.) on data lakes and filesystems. Either way, we assume the dataset is logically a single table with all column names available. Several suitable data prep steps must be applied to build ML models on such data. We focus on a major step, the ML feature

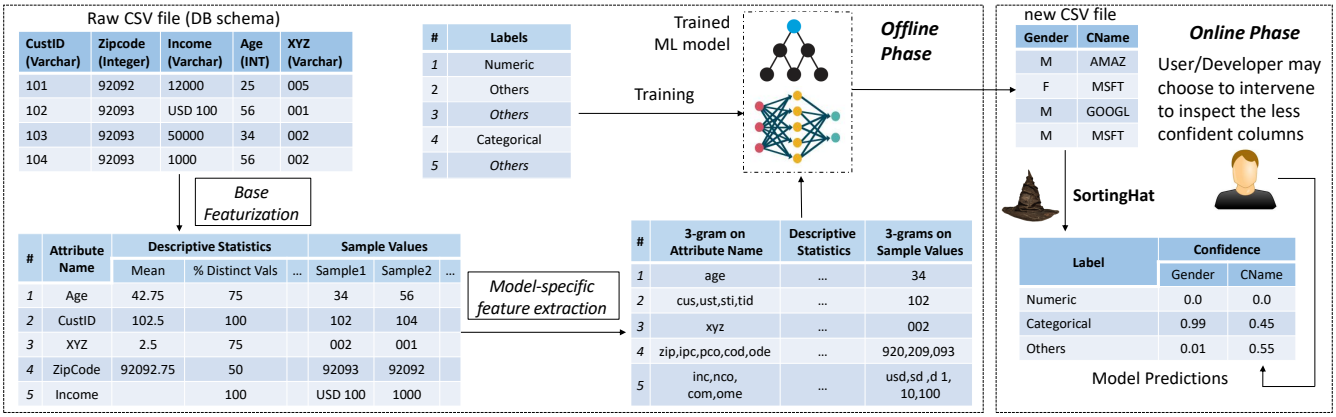

Figure 3: ML feature type inference workflow. In the offline phase, we extract 3 signals from the raw CSV file: attribute name, descriptive stats, and 5 sample values. We then extract hand-crafted *n*-gram feature set from attribute name and sample values. We finally use these feature sets to train ML models. In the online phase, SortingHat tool uses our pre-trained models to infer feature types for new CSV files.

type inference. We leave other data prep steps to future work. Note that our focus is *not* on feature engineering over prepared data. Also, to avoid ambiguity, we call the ML model(s) to be trained on the prepared data the "downstream model" or "target model." For example, one might load a customer table to train a target model for predicting customer churn.

2.2 Overview of Our Workflow

Figure 3 gives an end-to-end overview of the ML feature type inference workflow on a dataframe before target model training begins. Recall that we cast ML feature type inference as an applied ML classification task. To make this possible, we need the following.

Label Vocabulary. We find that usually just the dichotomy of numeric or categorical is not enough for categorizing the ML feature types. Hence, we add one more class to our label vocabulary. We explain this in depth in Section 3.1.

Labeled Dataset. There exists no publicly available dataset for this task. Thus, we manually annotate a large labeled dataset containing 9251 examples. Sections 3.2 to 3.4 cover this in depth.

Features. We transform each column in the raw CSV file into a feature vector containing the column name, descriptive stats, and 5 randomly sampled column values. We call this step Base Featurization. Section 3.3 explains this. 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 explain this step in Section 4.1.

ML models. We finally use our feature set on our labeled dataset to build ML models. Sections 4.3 to 4.5 compare them in depth.

The above steps are carried out once offline. In the online phase, the SortingHat tool uses our pre-trained ML model to infer feature types for columns in an "unseen" CSV file. Our trained ML models can help automate feature type inference in AutoML platforms and can help raise their overall performance. At the scale of AutoML platforms where there are potentially millions of columns, human intervention can be costly and slow. Our ML models also output confidence scores for each class. This allow users (or AutoML platform developers) to intervene to prioritize their effort towards features with low confidence scores that may need more human attention.

3 OUR DATASET

This section discusses our efforts in creating the labeled dataset: the label vocabulary, the data sources, the signals we extract from the columns, and the labelling process.

3.1 Label Vocabulary

Target models usually recognize only two feature types: *numeric* or *categorical*. Hence, each example (or attribute) has to be labelled as either of the two classes. But we find that there is often not enough information in just the data file to distinguish between the two classes correctly, even for humans. Thus, we add one more class to identify such attributes. We now explain each class and give examples below.

(1) Numeric. These attributes represents numeric values that are quantitative in nature and which can directly be utilized as a *numeric* feature for the target ML model. For instance, *Age* 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 target ML model. e.g. *ZipCode*, which is syntactically a number. Hence, one needs to alter its syntax slightly for a target model, e.g., convert it to string in Python or explicitly cast it as a "factor" variable in R.

(3) Others. This is a catch-all for columns whose type is hard to tell even for humans. Following examples illustrates this class.

(a) The column can have "messy" values that preclude their direct use as numeric or categorical features. For instance, *Income* in Figure 3 require some form of processing before being used as a numeric feature. More examples include text fields with semantic meaning, consisting of either sentences, URL, address, geo-location, list of items separated by a delimiter, or even a date or timestamp. One may choose to extract custom features, either numeric or categorical, or both through standard featurization routines. Note that, such feature engineering decisions are not focus of this work, since they are typically application-specific.

(b) A column can be a primary key in the table or has (almost) no informative values to be useful as a feature. e.g., *CustID*. Since every

future customer will have a new *CustID*, it is quite unlikely that one can get any useful features from it.

(c) A column can have meaningless names. For example, *XYZ* in Figure 3 has integer values but it is hard to decide its feature type without any additional information.

3.2 Data Sources

We obtain 360 CSV data files from sources such as Kaggle and UCI ML repository. In total, we have 9251 examples; note that an example for our task is a whole column in the raw data file. Kaggle and UCI ML are the largest 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 truly "in-the-wild" data from enterprises and other organizations and make them publicly available due to legal restrictions. We hope this paper start a conversation around enhancing such benchmark datasets. We are already working with industry collaborators towards this goald of deploying the models in-the-wild. But the crux of our point in this paper is, even on data files from Kaggle and UCI, existing open-source and industrial tools yield relatively poor accuracy compared to our ML models (Section 5.2). Thus, we believe our work is a promising start towards objectively evaluating AutoML platforms.

3.3 Base Featurization

To emulate the intuition of a human data scientist, we extract the following signals from the data files.

(1) Column name. The name of a column can give crucial semantic clues for the feature type. For instance, a human can tell that *ZipCode* is almost surely a categorical feature from just the name. Thus, we extract the column name from the first line of the CSV.

(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 feature, while values with commas are likely lists requiring extraction. Considering this, we extract 5 randomly sampled attribute values from the column.

(3) Descriptive statistics. Finally, a human would look at some descriptive statistics about the column. For instance, if the human finds that 99.99% of values in the column are NaNs, then they might classify the column as *Not-Generalizable*. Based on this observation, we extract several descriptive statistics for a column: percentage of distinct values, percentage of NaNs, mean, standard deviation, minimum value, maximum value, castability as number (e.g., "123" is a number embedded in string), extractability of number (e.g., "12 years" has a number that can be extracted using regular expressions), and average number of whitespace-separated tokens.

We convert the above signals to a feature set that will be used to build ML models. Each column in the raw CSV files is an example (or row) in the new base featurized file.

3.4 Labelling Process

We label all examples manually into one of the three classes. 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 9251 examples. We then group all the examples by these predicted labels and inspect all of them manually. Such grouping helps reduce cognitive load caused by class context switches during labeling. The labeling process took about 75 man-hours across 4 months.

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

3.5 Data Statistics

The distribution of class labels in our labeled dataset is: *Numeric* (39.4%), *Categorical* (22.9%), and *Others* (37.7%). We provide a complete breakdown of the cumulative distribution by class for different descriptive stats in the appendix.

4 APPROACHES COMPARED

In this section, we first discuss feature sets that we extract from the base featurized file, since some models do not operate at the raw character level. We briefly discuss an intuitive rule-based baseline. Finally, we discuss how we apply several classical ML models, k-NN with a distance function tuned for our task, and a CNN.

4.1 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*. Based on this intuition, we extract an n-gram feature set from the attribute names. Similarly, we extract n-gram feature set from the attribute values. This can be helpful because knowing that the sequence of the characters are numbers followed by a /, can give an indication of *date*, which belongs to *Others*.

Notation. We denote the descriptive stats by $\mathbf{X}_{stats}$, attribute name by $\mathbf{X}_{name}$ and the sampled attribute values by $\mathbf{X}_{sample}$ (first random attribute value is referred as $\mathbf{X}_{sample1}$ and similarly for other values). We leverage the commonly used bi-gram and tri-gram feature sets. We denote these feature sets on the attribute name by $\mathbf{X2}_{name}$ and $\mathbf{X3}_{name}$. Similarly, we denote the feature set on the attribute value by $\mathbf{X2}_{sample}$ and $\mathbf{X3}_{sample}$. $\mathbf{X}_{stats}$, $\mathbf{X}_{name}$, $\mathbf{X}_{sample}$, $\mathbf{X2}_{name}$, $\mathbf{X3}_{name}$, $\mathbf{X2}_{sample}$ and $\mathbf{X3}_{sample}$ are used as feature sets for the compared ML models.

4.2 Rule-based Baseline

We develop a rule-based approach that mimics the human thought process to arrive at the label. The rule-based model uses a flowchart-like structure where each internal node is a "check" on an attribute, each branch is the outcome of the check, and each leaf node represents a class label. We describe our rule based system in depth in the appendix. It is highly cumbersome and perhaps even infeasible to hand-craft a perfect rule-based classifier. Thus, we instead leverage ML models for this task.

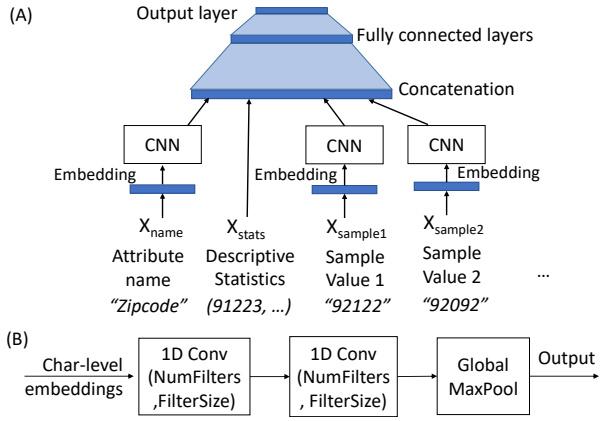

Figure 4: (A) The end-to-end architecture of our deep neural network. (B) The CNN block's layers.

4.3 Classical ML models

We consider classical ML models: logistic regression, RBF-SVM, and Random Forest. The features are: $\mathbf{X}_{stats}$, $\mathbf{X2}_{name}$, $\mathbf{X3}_{name}$, $\mathbf{X2}_{sample1}$, $\mathbf{X3}_{sample1}$, $\mathbf{X2}_{sample2}$ and $\mathbf{X3}_{sample2}$. Note that they cannot operate on raw characters of attribute names or sample values; thus, $\mathbf{X}_{name}$, $\mathbf{X}_{sample}$ are not used. For scale-sensitive ML models such as RBF-SVM and logistic regression, we standardize $\mathbf{X}_{stats}$ features to have mean 0 and standard deviation 1.

4.4 Nearest Neighbor

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

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

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

4.5 CNN

For short text classification tasks, character-level CNNs have achieved good accuracy [28, 29]. Inspired by this, we propose a neural model for our task as shown in Figure 4(A). The layers of CNN are shown in Figure 4(B). The network takes attribute name, descriptive statistics and sample values as input and gives the prediction from the label vocabulary as output.

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 *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 are 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 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.

5 EMPIRICAL STUDY AND ANALYSIS

We first discuss our methodology, setup, and metrics for evaluating the ML models. We then compare the ML models trained on our data against existing tools. We then present the accuracy results of all models trained on our dataset and intuitively explain the behavior of Random Forest and the CNN. We then present the prediction runtime of all models on a given example. Finally, we compare the downstream model's accuracy against existing tools.

5.1 Methodology, Setup and Metrics

Methodolody. We partition our labeled dataset into train and held-out test set with 80:20 ratio. We perform 5-fold nested cross-validation of the train set, with a random fourth of the examples in a training fold being used for validation during hyper-parameter tuning. We use a standard grid search for hyper-parameter tuning. The grids are described in the appendix. We also 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. We notice that the results are comparable to 5-fold nested cross-validation methodology. We present the results in the appendix.

Experimental Setup. We use CloudLab [20] with custom OpenStack profile running Ubuntu 16.10 with 10 Intel Xeon cores and 64GB of RAM. For TFDV, Transmogrifai, and Pandas, we use version number 0.21.0, 0.6.0, and 0.24.2 respectively.

Metrics. Our key metric is prediction accuracy, defined as the diagonal of the 3 x 3 confusion matrix. We also report the per-class accuracy and their confusion matrices.

5.2 Comparison with Prior Tools

We compare ML models trained on our dataset against 3 open-source and industrial tools: TFDV, Pandas, and TransmogrifAI. TFDV can infer only 2 types of features in our vocabulary: numeric or everything else. Pandas can only infer syntactic types: int, float, or object. TransmogrifAI can only infer primitive types such as Integer, Real or Text. Hence, we can not use our entire 3-class vocabulary for this comparison. Instead, we report the results on a binarization of our vocabulary: numeric (Num) vs. all non-numeric (Not-Num). Table 1 presents the precision, recall, and overall classification accuracy results on the test set.

Results. We notice a lift of ~30% in accuracy for our ML-based approach against existing tools. Interestingly, all the existing tools have high recall on numeric features but very low precision. This is because their rule-based heuristics are syntactic, which leads them to wrongly classify many categorical features such as *ZipCode* as numeric. Our models have slightly lower recall on numeric features. This is because when many features are thrown into an ML model,

	TFDV		Pandas		TransmogrifAI		Log Reg		RBF-SVM		k-NN		Random Forest		CNN	
	Num	Not-Num	Num	Not-Num	Num	Not-Num	Num	Not-Num	Num	Not-Num	Num	Not-Num	Num	Not-Num	Num	Not-Num
Precision	0.5412	0.9413	0.5418	0.9382	0.5130	0.9632	0.9331	0.9450	0.9324	0.9614	0.9583	0.9662	0.9722	0.9360	0.9445	0.9494
Recall	0.9490	0.5039	0.9502	0.4849	0.9711	0.4509	0.9093	0.9598	0.9377	0.9581	0.9447	0.9747	0.8909	0.9843	0.9164	0.9668
Accuracy	0.6737		0.6667		0.6451		0.9394		0.9512		0.9633		0.9508		0.9521	

Table 1: Held-out test accuracy comparison of the tools TFDV, Pandas, and TransmogrifAI with our ML models.

Model	$[X_{stats}]$	$[X2_{name}]$	$[X_{stats}, X2_{name}]$	$[X_{stats}, X2_{name}, X2_{sample1}]$	$[X_{stats}, X2_{name}, X2_{sample1}, X2_{sample2}]$	$[X3_{name}]$	$[X_{stats}, X3_{name}]$	$[X_{stats}, X3_{name}, X3_{sample1}]$	$[X_{stats}, X3_{name}, X3_{sample1}, X3_{sample2}]$
Logistic Regression	0.5288	0.7995	0.84	0.8512	0.8498	0.8479	0.8728	**0.8798**	0.8720
RBF-SVM	0.7681	0.8425	0.8871	0.88	0.8849	0.8431	0.8857	0.884	**0.9001**
Random Forest	0.8399	0.8367	0.9152	0.9126	0.9144	0.8001	**0.9187**	0.9126	0.9053

Table 2: 5-fold held-out test accuracy of classical ML models with different feature sets. The bold fonts marks the cases where we noticed highest held-out test accuracy for that model.

it gets slightly confused and could wrongly predict a numeric type as non-numeric. But, our ML models have much higher precision and high overall accuracy. Of all our ML models, the weighted k-NN achieves the best accuracy in identifying numeric type. Interestingly, k-NN also has the highest recall for numeric type among our ML models. On the other hand, Random Forest has the highest precision for predicting numeric type.

Other Commercial Tools. There exist other commerical tools that also automate the ML feature type inference task such as Google AutoML Tables [10], DataRobot [5], 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 on our demo request. (2) AutoML tables and Trifacta only offers a GUI-based interface where users must upload the raw CSV files manually to identify the feature types. Both these tools do not provide any programmatic way for evaluation. So, we cannot evaluate their accuracy automatically. We manually uploaded 5 CSV files from our raw data. All 15 categoricals encoded as integers were (wrongly) classified as numeric by both tools. So, they will likely have the same issues as TFDV and TransmogrifAI.

5.3 End-to-End Accuracy Results

Rule-based Heuristic. The overall 3-class classification accuracy on the held-out test set is *0.5754*. We observe that this approach achieves 97% and 37% recall in classifying *Numeric* and *Categorical* respectively. The recall for *Categorical* is low because an number encoded as a category is mistakenly classified as *Numeric*. We present the confusion matrix in the appendix.

Classical ML Models. Table 2 presents the 3-class accuracy results of the classical ML models using different feature sets on the held-out test set. For logistic regression, we see that the descriptive stats alone are not enough, as it achieves an accuracy of just 53%. But, for RBF-SVM and Random Forest, the accuracy with stats alone is already 77% and 84% respectively. Incorporating 2-gram features of the attribute name into Random Forest leads to an impressive 92% accuracy. 2-gram or 3-gram features of random sample values leads to only marginal gains. More complex features such as 3-gram leads to more significant gains in accuracy relative to bi-grams. Overall, Random Forest achieves the best 3-class accuracy of 92% using the 3-gram features on the attribute name along with descriptive stats.

We observe that with Random Forest, the recall for *Numeric* is 95% which is slightly lower than the rule-based heuristic. It does well in predicting *Categorical* and *Others* classes, achieving a recall of 86% and 93%, respectively. Many examples belonging to *Categorical* are confused with *Others*. We present the confusion matrix of all models in the appendix. We analyze the behavior of Random Forest in depth in Section 5.4.

Nearest Neighbor. From Table 3, we observe that with only Euclidean distance on descriptive statistics, the accuracy is already 78%. With only edit distance on attribute name the accuracy is 86%. Finally, with our weighted edit distance function from Section 4.4, k-NN achieves a high 91% accuracy, comparable to Random Forest.

CNN. Table 3 presents the accuracy of CNN on different feature sets. We see that with just X_{name}, the accuracy is already 82%. The descriptive statistics lift the accuracy further by 8%. We notice that sample values are not that useful here; they yield only minor lift in accuracy. We analyze its behavior in Section 5.4.

5.4 Analysis of Errors

We now explain the behavior of Random Forest and the CNN on our dataset by inspecting the raw datatype of the attribute values. We categorize the data type into integers, floats, negative numbers, sentences with one token, and sentences with more than one token. Table 4 (A) shows the confusion matrix of the predicted class by Random Forest and CNN vs actual datatype of the attribute value on the test set. Table 4 (B) shows examples of attributes and the corresponding prediction made by Random Forest and CNN. We intuitively explain the errors by class below.

Numeric. We see that when the actual label is *Numeric* (Table 4 (A1)), Random Forest and the CNN is less likely to misclassify an attribute whose values are floats or negative numbers compared to integers. We observe that with integers, Random Forest gets confused with *Others* class. For instance, *s3area* (Table 4 example(B1)) is predicted as *Others*. Looking at substring "area" in the attribute, humans have this intuition that the column is probably numeric. Although, Random Forest makes a wrong prediction, the CNN

Model	$[X_{stats}]$	$[X_{name}]$	$[X_{stats}, X_{name}]$	$[X_{sample1}]$	$[X_{name}, X_{sample1}]$	$[X_{stats}, X_{sample1}]$	$[X_{stats}, X_{name}, X_{sample1}]$	$[X_{stats}, X_{name}, X_{sample1}, X_{sample2}]$
CNN	0.7111	0.8186	0.9029	0.6636	0.8621	0.7702	0.9033	**0.9098**
k-NN	0.7768	0.8606	**0.9098**	N/A				

Table 3: 5-fold held-out test accuracy of CNN and k-NN model with different feature sets.

(A)	(A1) Numeric			(A2) Categorical			(A3) Others		
	Integers	Floats	Negative Numbers	Numbers	Sentence (length > 1)	Sentence (length = 1)	Numbers	Sentence (length > 1)	Sentence (length = 1)
Numeric	384 (372)	284 (285)	196 (195)	8 (16)	0 (0)	0 (3)	27 (32)	0 (0)	0 (4)
Categorical	5 (13)	2 (2)	0 (1)	369 (381)	61 (66)	108 (114)	26 (53)	10 (19)	9 (19)
Others	21 (25)	7 (6)	2 (2)	54 (34)	24 (19)	18 (9)	661 (629)	115 (106)	151 (137)

(B)	Attribute Name	Sample Value	Label	RF Prediction	CNN Prediction
B1	s3area	579	*NU*	*OT*	*NU*
B2	DataValue	30.7	*NU*	*OT*	*NU*
B3	job_title	IT Support Technician	*CA*	*OT*	*OT*
B4	SMOD_POPULATION	-9999	*CA*	*NU*	*CA*
B5	dhsclust	366	*OT*	*OT*	*NU*
B6	material	wax	*OT*	*OT*	*CA*

Table 4: (A) Breakdown of Random Forest (and CNN in parentheses) model's prediction for different types of attribute values on the held-out test set. Confusion matrices (Predicted class on the row vs. Actual type of the column value on the column) when the ground-truth label is (A1) *Numeric*, (A2) *Categorical*, and (A3) *Others*. (B) Examples for illustrating errors made by Random Forest (RF) and CNN. *NU* refers to *Numeric*, *CA* refers to *Categorical*, and *OT* refers to *Others*.

captures this human-like intuition. For other attributes involving sub-strings such as count, value, etc., we observe the same trend: the CNN captures this human-like intuition.

Categorical. As shown in Table 4 (A2), when the sample values are sentences with number of tokens greater than 1, there is more chance for Random Forest and CNN to misclassify *Categorical* as *Others* (Table 4 example(B3)). On the other hand, for one-token strings, both models are more accurate in predicting *Categorical*. Table 4 example(B4) shows an attribute "SMOD_POPULATION" which has values such as "-9999" as part of the encoded category. Random Forest classifies it as *Numeric*. Again, it seems that Random Forest, unlike CNN is missing the human-level intuition that an attribute can have semantic values embedded as categories.

Others. We notice that CNN often misclassifies attributes with non-sensical names and numerical values as *Numeric* (Table 4 example B5). In addition, attributes with only 1 unique value in the column are often misclassified as *Categorical* (Table 4 example B6).

5.5 Prediction Runtimes

We evaluate the running time of ML models in the online phase, i.e, to make prediction 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 1 sec. For the classical models, the additional feature extraction dominates overall runtime. Since SVM and k-NN are distance-based methods, they have the highest runtime. Overall, CNN does the fastest inference. We present the time breakdown in the appendix.

5.6 Downstream Model Accuracy Results

To understand the effect of accuracy of ML feature type inference on the downstream model's accuracy, we present an end-to-end comparison of SortingHat vs. Pandas and TFDV in Table 5.

Datasets. The results are heavily dependent on the dataset and the downstream prediction task. Since there are unboundedly many datasets and downstream tasks, for the sake of tractability we download 15 "unseen" datasets from Kaggle and UCI ML repository and use them for evaluation. *Cancer*, *Nursery*, *Balance*, *Flares*, and *Hearts*

Feature Types	Attribute Types	Dataset	Y	Logistic Regression			Random Forest		
				PD	TFDV	SH	PD	TFDV	SH
NU	Int,Float	Cancer	2	0.6	0.6	0.6	0.683	0.683	0.683
CA	String	Nursery	5	0.913	0.913	0.913	0.978	0.978	0.978
	Int	Balance	3	0.883	0.883	0.939	0.859	0.859	0.87
	String, Int	Flares	3	0.871	0.871	0.88	0.88	0.88	0.88
OT	Date, String	Articles	2	0.969	0.77	0.969	0.958	0.765	0.958
	String	Spam	2	0.986	0.887	0.986	0.979	0.887	0.979
NU + CA	Int	Hearts	2	0.879	0.885	0.918	0.885	0.879	0.902
	Int,Float, String	Churn	2	0.787	0.791	0.791	0.783	0.78	0.785
NU + OT	Int, Date, String	NYC	15	0.60	0.607	0.603	0.661	0.64	0.667
	Int, Date	Rain	2	0.999	0.57	0.999	0.999	0.57	0.999
CA + OT	Int, String	Zoo	5	0.867	0.924	0.933	0.905	0.924	0.933
	Int, Date, String	Industry	2	0.984	0.976	0.984	0.986	0.976	0.986
NU + CA + OT	Int, Date	House	2	0.571	0.571	0.809	0.814	0.42	0.82
	Int, Text, String, Float, Keys,Date	Avocado	2	0.733	0.92	0.798	0.998	0.998	0.985
		Airbnb	5	0.459	0.459	0.993	0.996	0.996	0.996

Table 5: Comparison of downstream model accuracy of SortingHat (SH) against Python Pandas (PD) and TFDV. *NU* refers to *Numeric*, *CA* refers to *Categorical*, and *OT* refers to *Others*. Y is the number of target classes.

are from UCI ML repository, while the rest are from Kaggle. Although the datasets are downloaded "randomly", we capture all possible combinations of feature types with many different combinations of attribute types (*ints*, *floats*, *string*, *dates*, and even *primary keys*). The datasets are available on our Github repo [1].

Models and tools compared. In terms of model evaluation, we present both extremes of bias-variance tradeoff [8]: Logistic Regression (high bias, low variance) and Random Forest (low bias, high variance). Pandas can only infer numeric vs. string, TFDV can infer numeric vs. categorical, and SortingHat can infer any class from our 3-class vocabulary. SortingHat internally uses the Random Forest model trained on our labeled dataset. Columns that are inferred categorical are one-hot encoded, columns that are inferred string

(by Pandas) or *Others* (by SortingHat) are featurized with 2-gram, and numeric columns are retained as it is. After featurization we use the same methodology as Section 5.1 for evaluation.

Results. We observe that SortingHat is able to correctly infer the feature types for 116 out of 129 columns in these 15 datasets. The accuracy of the downstream model with inferred types from SortingHat lead to significant improvements over TFDV and Pandas on 12 out of 15 datasets. For instance, on *House* and *Airbnb* datasets, logistic model with SortingHat leads to a lift of 23% and 53% lift in accuracy compared to that with TFDV and Pandas. Only on 1 dataset (*Avocado*), SortingHat performs worse than TFDV. This is because *Avocado* contains columns with non-sensical names such as 4046 and 4770 which are actually *Numeric* and very predictive of the target but treated as *Others* with a confidence score of 0.6 by SortingHat. If a human intervenes to inspect such low confident columns, then the accuracy can be further improved. On the remaining 2 datasets where there are either all numeric features (*Cancer*) or all categorical features with *strings* (*Nursery*), the accuracy of both models with all three tools are same.

Interestingly, although the categories encoded as integers in *Balance*, *Flares*, and *Hearts* are misclassified by Pandas and TFDV, the accuracy of Random Forest drops marginally. This is because Random Forest has zero bias and thus can potentially represent all categories by doing splits on integers. Logistic Regression, which has lower VC-dimension can not do this and hence, the lift in accuracy is significantly higher, almost 5% on *Balance* and 4% on *Hearts* compared to Pandas and TFDV. On *Articles* and *Spam*, where the dataset contains only string and date type features, TFDV misclassifies them as categorical. On the other hand, SortingHat is able to infer the feature types for both of them correctly. Thus, the lift in accuracy is significantly higher, almost 20% on *Articles* and 10% on *Spam*. We observe that treating date as categorical in contrast to taking bigrams leads to a significant information loss.

Overall, we find that SortingHat is accurate enough for inferring feature types for any unseen dataset from Kaggle and UCI ML repo. Most importantly, SortingHat can deliver significant improvements in accuracy of the downstream model compared to TFDV and Pandas. In addition, with some human intervention the accuracy of downstream model can be further improved.

5.7 Public Release and Leaderboard

We have released a public repository on GitHub with our entire labeled data for the ML feature type inference task [1]. We also release our pre-trained ML models: k-NN, logistic regression, RBF-SVM, Random Forest, and the CNN. The repository tabulates the accuracy of all the models. The repository includes a leaderboard for public competition on the hosted dataset with 3-class classification accuracy and per-class recall being the metric. In addition, we release the raw 360 CSV files and we invite researchers and practitioners to use our datasets and contribute to create better featurizations and models.

5.8 Discussion and Takeaways

For Practitioners: We make all of our models and featurization routines available for use by wrapping them under functions in a Python library [1]. Our trained ML models can be integrated for feature type inference into existing data prep environments. *We are currently in process of integrating our pre-trained models with TFDV in collaborations with Google engineers to improve its inference of categorical and numeric feature types.*

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 other features that can perfectly capture human-level reasoning is an open research question. We found that descriptive statistics and attribute names are very useful for prediction. But, attribute values are only marginally useful. Perhaps, one can consider designing better featurization routines for sample values. Although our current CNN captures several human-level intuitions, it sometimes fails in recognizing categories encoded as integers. Capturing more semantic knowledge of attributes in neural architecture is an open problem. Finally, based on our analysis in Section 5.4, one potential way to increase the accuracy is to create more labeled data in the categories of examples where our models get confused. For instance, we see that when strings such as "-999" are used as categories, almost all of our models treat them as numbers and predict *Numeric*. Weak supervision and denoising with Snorkel [19] is one potential mechanism to amplify labeled datasets and teach our models better. Finally, one can also design richer label vocabularies and train models on such fine-grained vocabulary.

6 RELATED WORK

AutoML Platforms. Several AutoML tools such as AutoWeka [25] and Auto-sklearn [7] have automated search process for model selection, allowing users to spend no effort for algorithm selection or hyper-paramter search. However, these AutoML systems do not automate data prep tasks in the ML workflow. AutoML platforms such as Einstein AutoML [22] and AutoML Tables [10] do automate some data prep tasks. However, how good their existing automation schemes are is not well-understood. We believe there is a pressing need to formalize data prep tasks and create benchmark labeled datasets for evaluating and comparing AutoML platforms on such tasks. Our comparison with TransmogrifAI and TFDV shows that they still fall short on accuracy. Our ML models can be integrated into such AutoML platforms to improve their accuracy, as we are currently doing with TFDV. In addition, other platforms for Machine Learning such as Airbnb's Zipline [30], Uber's Michelangelo [18], Facebook's FBLearner Flow [6], and commercial AutoML platforms such as H20.AI [14] and DataRobot [5] are complementary to our focus and they can also benefit by adopting our models.

ML Data Prep and Cleaning. ML feature type inference has been explored in some prior tools [3, 5, 10, 17, 26, 27]. TFDV is a rule-based tool that infers ML feature types from summary statistics about the column [3]. Pandas is a Python library that provides syntactic type inference [17]. TransmogrifAI is a library for data prep, feature engineering, and ML model building in Salesforces' Einstein AutoML [26]. It provides ML feature type inference over primitive types such as integer, real number, or text. AutoML Tables [10], DataRobot [5], and Trifacta [27] are commercial tools that do support the feature type inference. However, they do not offer any programmatic way of evaluation, thus we do not compare it

with our work. Compared to the open-source tools, our ML-based approach raises accuracy of ML feature type inference substantially.

DataLinter is a rule-based tool that inspects a data file and raises potential data quality issues as warnings to the user [16]. However, ML feature type inference must be done manually. Deequ is a tool for validating data quality where the integrity constraints specified by the user are interactively verified [23]. There are numerous tools such as Programming-by-example [12, 13] and visual interfaces [27] that allows users to perform data transformations tasks for data prep. However, they do not handle ML feature type inference. Hence, they are orthogonal to our focus.

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 [4]. [2] infer 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 type of the attribute with integer values has to be identified as an integer. In contrast, with ML feature type inference the attributes with type integer can be categorical.

REFERENCES

[1] Github repository for ml feature type inference https://github.com/pvn25/ML-Data-Prep-Zoo/tree/master/ML%20Feature%20Type%20Inference.
[2] M.-A. Baazizi et al. Schema inference for massive json datasets. In *Extending Database Technology (EDBT)*, 2017.
[3] D. Baylor et al. Tfx: A tensorflow-based production-scale machine learning platform. In *SIGKDD*. ACM, 2017.
[4] bigquery. Schema detection bigquery https://cloud.google.com/bigquery/docs/schema-detect.
[5] DataRobot. Datarobot https://www.datarobot.com/.
[6] FBLearner Flow. Facebook's fblearner flow https://engineering.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/.
[7] M. Feurer et al. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, 2015.
[8] J. Friedman et al. *The elements of statistical learning*. Springer, 2001.
[9] P. Gijsbers et al. An open source automl benchmark. *arXiv preprint*, 2019.
[10] google. Google automl tables https://cloud.google.com/automl-tables/docs/data-types.
[11] Google AutoML. Google cloud automl https://cloud.google.com/automl/.
[12] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *ACM Sigplan Notices*, volume 46, pages 317–330. ACM, 2011.
[13] S. Gulwani et al. Spreadsheet data manipulation using examples. *ACM*, 2012.
[14] H2o.AI. H2o.ai https://www.h2o.ai/.
[15] F. Hutter et al., editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018.
[16] N. Hynes et al. The data linter: Lightweight, automated sanity checking for ml data sets. In *NIPS MLSys Workshop*, 2017.
[17] W. McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14, 2011.
[18] Michelangelo. Uber michelangelo https://eng.uber.com/michelangelo/.
[19] A. Ratner et al. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 2017.
[20] R. Ricci et al. Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications. *the magazine of USENIX & SAGE*, 2014.
[21] O. Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015.
[22] SalesForce AutoML. Salesforce einstein automl https://www.salesforce.com/video/1776007.
[23] S. Schelter et al. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment*, 2018.
[24] Survey. 2017 kaggle survey on data science. https://www.kaggle.com/surveys/2017.
[25] C. Thornton et al. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *ACM SIGKDD*. ACM, 2013.
[26] Transmog. Transmogrifai: Automated machine learning for structured data https://transmogrif.ai/.
[27] Trifacta. Trifacta: Data wrangling tools software https://www.trifacta.com/.
[28] X. Zhang et al. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, 2015.
[29] X. Zhang et al. Text understanding from scratch. *arXiv*, 2015.
[30] Zipline. Airbnb zipline https://conferences.oreilly.com/strata/strata-ny-2018/public/schedule/detail/68114.

REPRODUCIBILITY APPENDIX

A METHODOLOGY

We partition our labeled dataset into train and heldout 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. 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* $\in \{64, 128, 256, 512\}$, *numfilters* $\in \{32, 64, 128, 256, 512\}$, *filtersize* $\in \{2, 3\}$, and *neurons* $\in \{250, 500, 1000\}$. In order to regularize, we use dropout with a probability from the grid: {0.25,0.5,0.75}. 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.

We also tried tuning the following knobs of the CNN, but it did not lead to any significant improvement in accuracy: MLP architecture with 3 hidden layers, L_1 and L_2 regularization from the set $\{10^{-4}, 10^{-3}, 0.01, 0.1\}$ and number of neurons from the set $\{5 \cdot 10^3, 10^4\}$.

APPENDIX

B CROWDSOURCING EFFORTS

We tried to crowdsource labels for our dataset on the FigureEight platform but abandoned this effort because the label quality was too low across two trial runs. In 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

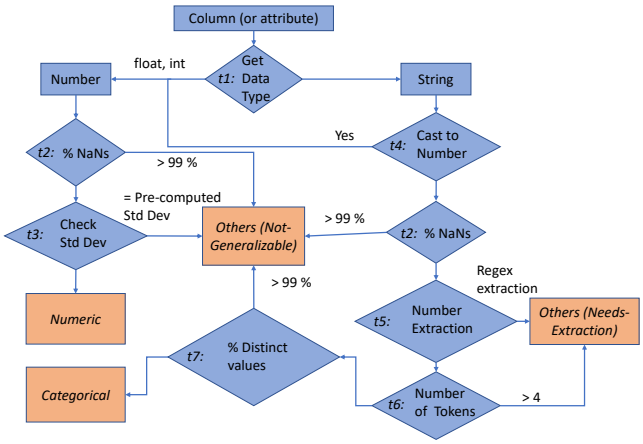

Figure 5: Flowchart of the rule based system. Diamond-shaped nodes are the decision nodes that represents a "check" on the attribute. The final outcome is shown in orange rectangular boxes.

and guidelines for the three classes and provided many examples for worker training. But in the end, we found the results too noisy to be useful: in the first run, 11% of examples had 3 unique labels and 41% had 2; in the second run, these were 6% and 49%. Majority voting gave the wrong answer in over 70% of the examples. 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.

C DATA STATISTICS

The distribution of class labels in our labeled dataset is given as: *Numeric* (39.4%), *Categorical* (22.9%), and *Others* (37.7%). Figure 7 plots the cumulative distribution functions (CDF) of different descriptive statistics obtained by base featurization. Table 6 reports the median for the same descriptive statistics. Figure 8 shows the CDF of the several descriptive statistics by class (A) *Numeric*, (B) *Categorical*, and (C) *Others*. Table 11 presents the mean, standard deviation and the maximum of the same descriptive statistics. We observe that *Numeric* attributes have longer names than others. Attribute values for *Others*, as expected, have more number of characters and words than other classes. In addition, we observe that all sample values in *Numeric* and 80% of the sample values in *Categorical* are single token strings. Furthermore, we find that almost 90% of the attributes in *Categorical* have less than 1% unique values in its columns.

	Numeric	Categorical	Others	Overall
# chars in Attribute Name	16	10	9	10
# chars in Sample Value	5	3	3	4
# words in Sample Value	1	1	1	1
% Distinct Vals	18.09	0.04	0.46	0.96
% NaNs	0	0	16.08	0

Table 6: Median of different Descriptive Statistics by class in the base featurized data file.

D CLASSICAL ML MODELS

The confidence of a model's prediction is defined as follows.

Logistic Regression. We use sigmoid function $(1/(1 + exp(-\theta^T \cdot x)))$ to determine the confidence of prediction for a given example. The parameter vector θ is learned during training.

RBF-SVM. We use Platt scaling $(1/(1 + exp(A * f(x) + B)))$ to calibrate the SVM to produce probabilities in addition to class predictions. $f(x)$ is the distance of a given example from the decision boundary generated by the SVM. A and B are the parameters learned through training.

Random Forest. The confidence score for class A is given by n_A/n, that is, the number of examples of class A (n_A) captured by the leaf node over the total number of examples (n) captured by that leaf during the training process. The confidence score of the whole forest for a particular class (say, class A) is calculated by taking average of the confidence score from the decision trees that classified the example as class A.

E RULE BASED BASELINE

We develop a rule based approach that mimics the human thought process to arrive at the label. The rule based model uses a flowchart-like structure as shown in Figure 5. Each internal node is a "check" on an attribute, each branch is the outcome of the check, and each leaf node represents a class label. We describe all the checks on the attribute below.

t1. We query the data type of 5 random sample values of an attribute using Python. We then take the mode of the returned data type. If the mode is `integer` or `float`, we mark it as a number; otherwise, we mark it as a `string`.

t2. For an attribute with a value marked as a number, we find the percentage of NaN values in its column. If this is greater than 99%, we classify it as *Others*.

t3. We find the standard deviation of the attribute values, denoted by *sd*. Denote the total number of attribute values as n and the standard deviation of integers from $1, 2, ..., n$ with *presd*. If *sd* is equal to *presd*, then the attribute is a serial number and we classify it as *Others*. Otherwise, we classify it as *Numeric*.

t4. For an attribute with a value marked as `string`, we check if we can cast the string into a number. We repeat this "castability" check (0 or 1) for 5 random sample values. We finally take mode of castability check result for the decision.

t5. We check if we can extract number from the marked string using regular expressions. We classify such attributes as *Others*.

t6. If the number of tokens in the sample value is large, then the sample value contains a text field that requires further processing to extract features. So, it is classified as *Others*.

t7. We count the percentage of distinct values present in an column. If this percentage is greater than 99%, then the attribute would not be able to generalize when used as feature for ML. We classify such attributes as *Others*; otherwise, we classify it as *Categorical*.

Model		$[X_{stats}]$	$[X2_{name}]$	$[X_{stats}, X2_{name}]$	$[X_{stats}, X2_{name}, X2_{sample1}]$	$[X_{stats}, X2_{name}, X2_{sample1}, X2_{sample2}]$	$[X3_{name}]$	$[X_{stats}, X3_{name}]$	$[X_{stats}, X3_{name}, X3_{sample1}]$	$[X_{stats}, X3_{name}, X3_{sample1}, X3_{sample2}]$
Logistic Regression	Train	0.5408	0.8788	0.9045	0.9255	0.9307	0.9243	0.9370	0.9532	0.9586
	Validation	0.5404	0.7968	0.8404	0.8552	0.8508	0.8312	0.8614	0.8651	0.8608
	Test	0.5288	0.7995	0.84	0.8512	0.8498	0.8479	0.8728	**0.8798**	0.8720
RBF-SVM	Train	0.8316	0.9362	0.9661	0.9503	0.9491	0.9189	0.9564	0.9532	0.9581
	Validation	0.7741	0.8391	0.8846	0.8708	0.8796	0.832	0.8766	0.8759	0.8884
	Test	0.7681	0.8425	0.8871	0.88	0.8849	0.8431	0.8857	0.884	**0.9001**
Random Forest	Train	0.9317	0.9327	0.9776	0.9754	0.9792	0.8718	0.9782	0.9754	0.9732
	Validation	0.8396	0.8396	0.9126	0.9034	0.9124	0.7965	0.9118	0.9034	0.8988
	Test	0.8399	0.8367	0.9152	0.9126	0.9144	0.8001	**0.9187**	0.9126	0.9053

Table 7: 5-fold training, cross-validation, and held-out test accuracy of classical ML models with different feature sets. The bold fonts marks the cases where we noticed highest held-out test accuracy for that model.

Model		$[X_{stats}]$	$[X_{name}]$	$[X_{stats}, X_{name}]$	$[X_{sample1}]$	$[X_{name}, X_{sample1}]$	$[X_{stats}, X_{sample1}]$	$[X_{stats}, X_{name}, X_{sample1}]$	$[X_{stats}, X_{name}, X_{sample1}, X_{sample2}]$
CNN	Train	0.7129	0.895	0.9926	0.7035	0.9333	0.844	0.9871	0.9932
	Validation	0.7062	0.8226	0.8976	0.6536	0.8639	0.7616	0.8892	0.898
	Test	0.7111	0.8186	0.9029	0.6636	0.8621	0.7702	0.9033	**0.9098**
k-NN	Validation	0.7632	0.8622	0.9077	N/A				
	Test	0.7768	0.8606	**0.9098**					

Table 8: Training, cross-validation and held-out test accuracy of CNN and k-NN model with different feature sets.

(A) Rule-based Heuristic	Numeric	Categorical	Others
Numeric	688	1	17
Categorical	239	161	31
Others	388	110	216

(B) Random Forest	Numeric	Categorical	Others
Numeric	669	8	29
Categorical	8	369	54
Others	27	26	661

(C) k-NN	Numeric	Categorical	Others
Numeric	669	10	27
Categorical	10	376	45
Others	24	51	639

(D) CNN	Numeric	Categorical	Others
Numeric	659	16	31
Categorical	16	381	34
Others	32	53	629

Table 9: Confusion matrices (actual class on the row and predicted class on the column) of (A) Rule-based heuristic (B) Random Forest model (C) k-NN model, and (D) CNN model.

F EMPIRICAL STUDY

F.1 End-to-End Accuracy Results

We present the train, cross-validation, and test accuracy results of all models trained on our dataset with 5-fold cross-validation methodology in Table 7 and Table 8. Table 9 shows the confusion matrices of all models.

F.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. Figure 10 present the train cross-validation, and test accuracy results of the classical ML models and k-NN with this methodology on the 3-gram features from attribute name and descriptive stats. We observe that the results are comparable to what we found with k-fold cross-validation methodology.

Model		$[X_{stats}, X3_{name}]$
Logistic Regression	Train	0.9504
	Validation	0.8777
	Test	0.8734
RBF-SVM	Train	0.9668
	Validation	0.8841
	Test	0.8849
Random Forest	Train	0.9999
	Validation	0.9241
	Test	0.9298
k-NN	Validation	0.8993
	Test	0.8971

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

F.3 Prediction Runtimes

We evaluate the running time of the ML models in the online phase, As shown in Figure 3, we must first perform Base Featurization of the column and then model-specific feature extraction. Base Featurization is a common step across all the models. Model-specific feature extraction is only needed for the classical models. Figure 6 shows the runtimes of all 5 models with breakdowns of Base Featurization, model-specific feature extraction time, and inference time. The measurements were made on the test set and averaged. All the models finish in under 1 sec. We see that for the classical models, the additional feature extraction dominates overall runtime. Since SVM and *k*-NN are distance-based methods, they require highest amount of time for inference. Overall, CNN does the fastest inference.

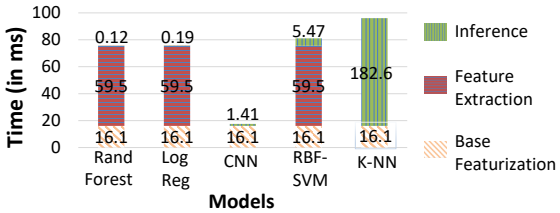

Figure 6: Comparison of prediction runtimes.

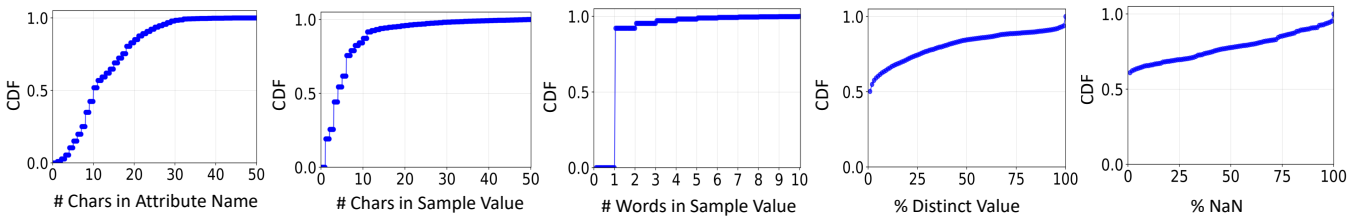

Figure 7: Cumulative distribution of descriptive statistics in the featurized data file.

Statistics	Overall			Numeric			Categorical			Others		
	Avg	Std Dev	MaxVal	Avg	Std Dev	MaxVal	Avg	Std Dev	MaxVal	Avg	Std Dev	MaxVal
Number of chars in Attribute Name	12.74	7.72	91	16.34	8.18	91	11.42	6.42	49	9.73	5.68	64
Number of chars in Sample Value	16.4	286.19	29.6K	5.98	5.39	398	6	10	150	33.4	464	29.6K
Number of words in Sample Value	2.66	46.67	4900	1	0.08	7	1.3	1.14	21	5.19	75.7	4905
Mean	1.65E+14	1.03E+16	8.8E+17	3.6E+10	1.03E+12	5.6E+13	2.5E+5	6.1E+6	2.05E+8	4.3E+14	1.7E+16	8.8E+17
Standard Deviation	3.34E+15	1.31E+17	5.4E+18	1.9E+12	9.9E+13	5.9E+15	2E+5	5.3E+6	2E+8	8.8E+15	2.1E+17	5.4E+18
% Distinct vals	19.3	31.58	100	28.8	31.1	100	2.4	11.8	100	19.5	35.7	100
% NaNs	22.4	34.7	100	12.6	27.8	99.97	18.7	32.5	99.99	34.6	38.5	100

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

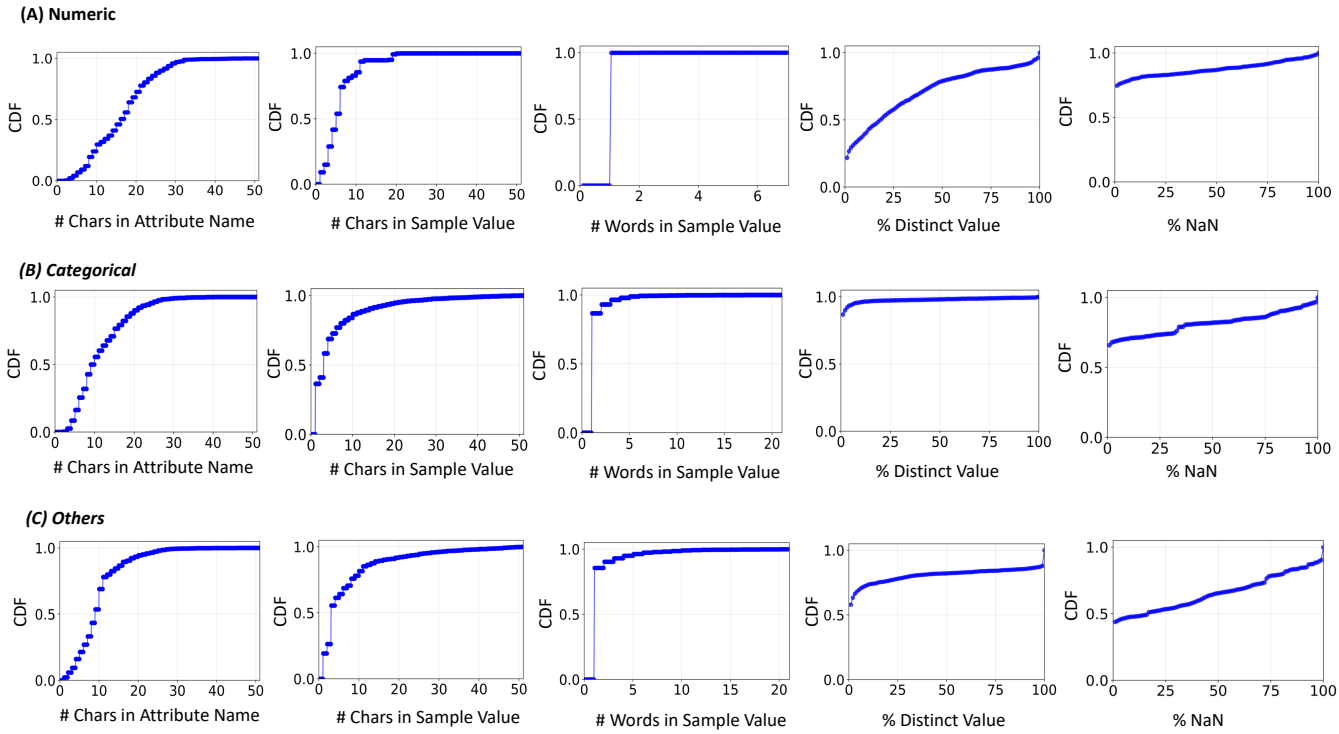

Figure 8: Cumulative distribution of different *descriptive statistics*.