Towards Semi-Automatic ML 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 tedious grunt work involved in data preparation is a major impediment for real-world ML applications which reduces the data scientist's productivity. It is also a roadblock to industrial-scale cloud AutoML workflows that build ML models for millions of datasets. In this work, we target a major task in data preparation: *ML feature type inference*. Datasets are typically exported from DBMSs into tools such as Python and R as CSV files. The semantic gap between attribute types (e.g., strings, numbers etc.) in a DBMS and feature types (e.g., numeric or categorical) in ML necessitates ML feature type inference. At scale, handling this task fully manually is slow, expensive, or even impossible. We formalize this task as a novel ML classification problem. We manually annotate and construct a labeled dataset with around 9000 examples. We mimic human-level intuition behind manual labelling into the ML models by extracting relevant signals and hand-crafted features from the raw CSV files. We present an extensive empirical analysis of several ML approaches on our dataset. Our results shows that our applied ML approach delivers an enormous 30% gain in identifying numeric attributes compared to existing rule-based or syntactic tools. We finally release a community-led repository with our labeled dataset and pre-trained models to invite further contributions. All of our code and datasets are available for download from `https://adalabucsd.github.io/sortinghat`.

1. INTRODUCTION

Several surveys of data science practitioners repeatedly show that most data scientists typically spend upto 80% of their time on data preparation (prep) and only 20% on real analytics [34, 33]. Data prep involves diverse tasks such as identifying feature types for ML, extracting, standardizing and cleaning feature values. Such tasks are mostly performed manually by data scientists in tools like Python and R, hence reducing their overall productivity. Furthermore, cloud vendors have released AutoML platforms such as Google's Cloud AutoML [32] and Salesforce's Einstein [35] that automates the end-to-end ML workflow including data prep. However, the formalization of different data prep steps in these platforms is ill-understood and there exist no objective benchmarks to evaluate them.

Automating specific data prep tasks and creating benchmark labeled datasets will not only help practitioners

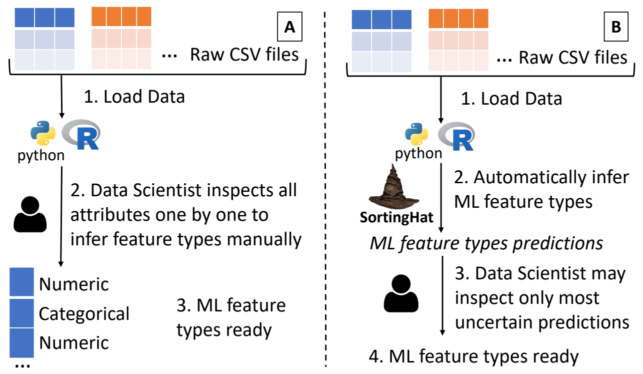

A
Raw CSV files
1. Load Data
python
2. Data Scientist inspects all attributes one by one to infer feature types manually
Numeric
Categorical
Numeric
3. ML feature types ready
B
Raw CSV files
1. Load Data
python
SortingHat
2. Automatically infer ML feature types
ML feature types predictions
3. Data Scientist may inspect only most uncertain predictions
4. ML feature types ready

Figure 1: (A) Current dominant approach to infer ML feature types. (B) Our proposed SortingHat tool that semi-automates the inference of ML feature types.

in reducing their grunt work but will also contribute to objectively benchmarking AutoML platforms. Considering this, in this paper, we focus on objectively quantifying a ubiquitous data prep step when applying ML over relational data: *ML Feature Type Inference.*

Problem: ML Feature Type Inference. Figure 1(A) shows the current dominant approach to infer ML feature types. Datasets are typically exported from DBMSs as flat CSV or JSON files into ML tools such as Python and R. Before ML, the data scientist has to manually decide upon the feature type of an attribute. Almost all ML models recognize only two types of features: *numeric* (a continuous set), or *categorical* (a discrete set) [19]. But the data files store only the names and values of the attribute, while the type of the attribute has to be inferred manually.

Challenge: Semantic Gap. This data prep task is hard to automate because of *the gap between the DB schema and the ML schema.* Analogous to attribute types and integrity constraints in a DB schema, the ML schema tells us the feature types and potential integrity constraints on the domains of the features. In this work, we only focus on feature type inference and leave inference of integrity constraints in the ML schema to future work. The key distinction between the DB schema and the ML schema is that the DB schema is primarily *syntactic*, while the ML schema is *semantic*. The DB schema tells us the datatype of an attribute such as integer, real, or string (`VARCHAR` in most DBMSs). On the other hand, the ML schema tells us the type of a feature such as *numeric* or *categorical*. The

Name	*CustID*	*Gender*	*Age*	*ZipCode*	*XYZ*	*Income*	*Churn*
VARCHAR (30)	*INTEGER*	*CHAR* (1)	*INTEGER*	*INTEGER*	*VARCHAR* (5)	*VARCHAR* (20)	*VARCHAR* (3)
Alice	1501	F	25	92092	005	USD 15000	Yes
Bob	1704	M	34	78712	003	25384	No

Figure 2: A simplified *Customers* dataset used for customer churn prediction.

semantic gap between DB and ML schemas means reading syntax as semantics often leads to nonsensical results. We explain this further with an example.

Example. Consider a simplified dataset for a common ML task, customer churn prediction in Figure 2.

We immediately see two major issues caused by the gap between the DB and ML schemas. First, attributes such as *Name*, *Gender*, *Income* and *Churn* are stored as strings, but not all of them are useful as categorical features for ML. For instance, *Income* is actually a numeric feature but some of its values have a string prefix. Second, attributes such as *CustID*, *Age*, *ZipCode* and *XYZ* are stored as integers, but only *Age* is useful as a numeric feature for ML. *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. A tool like Python Pandas will treat it as a numeric feature. Hence, a human has to manually convert to categorical feature for the ML model. This issue is ubiquitous in real-world datasets, since categories are often encoded as integers. e.g. item code, state code, etc.

Even worse, real-world datasets often have hundreds to thousands of columns. Asking a data scientist to spend even 1min to infer the feature type of a column could easily lead to hours, if not days, of pure grunt work! Also, real-world databases are seldom static because DB schemas evolve over time, which requires data scientists to manually infer the ML feature types every time. Overall, it is a pressing problem to close the gap between the DB and ML schemas and help reduce the human time spent on data prep for ML and/or improve the accuracy of AutoML systems.

Our Approach. To meet the above challenge, we apply a simple yet powerful philosophy: *using ML to semi-automate data prep for ML.* We cast the problem of ML feature type inference as an ML classification problem to exploit the ability of ML models to bride this semantic gap. As shown in Figure 1(B), we create a tool called SortingHat that uses our best performing ML models to semi-automatically infer the feature types from the raw CSV file. It gives a ranked list of attributes with a confidence score for its prediction. The data scientist may inspect only the attributes that are marked less confident by SortingHat. The key limiting factor to achieve this nature of automation is not ML algorithmic advances, but the availability of *large high quality labeled dataset.* For instance, availability of the ImageNet dataset has stirred several advances in computer vision today [11]. Considering this, we create the first labeled dataset for the task of ML feature type inference.

Label Vocabulary and Labeled Dataset. Creating labeled data for our task is challenging because of two reasons. First, each example in the labeled dataset is an entire feature column in a raw data file. A data file with 1M records and 10 columns will only give 10 examples! Hence, a lot of manual work needs to be done in order to collect raw data files. Second, there is usually not enough information in just the data file to identify the class (*numeric* or *categorical*) correctly. For instance, consider column *XYZ* in Figure 2. Is it *numeric* or *categorical*? This problem is non-trivial even for a human to make a judgement. Thus, we need more classes apart from just *Numeric* and *Categorical* class. We create 3 more classes that captures different variety of the columns: ones with messy syntax, ones that are non-generalizable and the ones that are "hard" to make a judgement. This intuitive 5-class prediction vocabulary will allow a human to quickly comprehend the semantics of an attribute. We manually label around 9000 columns from the real data files we collected into the one of the five classes.

Featurization and ML models. Given a raw data file, in order to identify the feature type, a human reader would look at the attribute (or column) name, some sample values in the column and even descriptive statistics about the column such as number of NaNs or number of distinct values. For instance, just by inspecting the attribute name such as *ZipCode*, an interpretable string, a human would know that the feature type is categorical. We replicate this human-level intuition into the ML models. We extract information from the raw data files that a human reader would look at: attribute name, sample values and descriptive statistics about the column. We summarize this information in a feature set, which we use to build popular ML models: logistic regression, support vector machine with radial basis kernel, Random Forest, k-nearest neighbor (k-NN) and character-level convolutional neural model.

Empirical evaluation and analysis. We first compare our models against existing rule-based or syntax-based approaches: Python Pandas [18], TensorFlow Data Validation (TFDV) [8], and Salesforce's TransmogrifAI [2]. We found that our ML models delivers a massive 30% lift in accuracy compared to these tools for identifying numeric features among the attributes. We then evaluate and compare different ML approaches on our dataset. Overall, we found that Random Forest model outperforms all ML models and achieves the best 5-class accuracy of 89.4%. The neural model and k-NN perform comparatively with 88.3% and 88.8% 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 neural model by considering their predictions on different types of column values such as integers, float, dates etc. Finally, we release a repository containing our labeled dataset and trained ML models and announce a competition for community-led contributions.

In summary, our work makes the following contributions:

- To the best of our knowledge, this is the first paper to formalize the ML feature type inference as an ML classification problem and create its label vocabulary.
- We create the first labeled dataset for the task of ML feature type inference.
- We compare several ML approaches on our dataset to understand how well they do in automating this task.
- We intuitively analyze the behaviour of Random Forest and the neural model on our dataset. Our analysis of errors gives several interesting findings useful for

further research on this work.

- We release a public competition and leaderboard on our labeled dataset in order to invite further contributions in this direction.

Outline. Section 2 presents background, prior work, assumptions, and scope. Section 3 present our overall Approach. Section 4 presents our manually labeled dataset. Section 5 presents several ML approaches used for comparison. Section 6 presents an in-depth experimental study and analysis of errors. We discuss the key takeaways for practitioners and researchers in Section 7 and finally conclude with related work in Section 8.

2. BACKGROUND

2.1 ML Terms and Concepts.

We explain the ML concepts, terms, and models intuitively and refer readers to [28, 20, 31] for more background.

Concepts. This work focuses solely on the classification task, which is an instance of supervised learning. It involves learning where a training dataset of correctly labeled examples is available. The ML classification models learn their parameters from the training dataset and then uses this learning to classify new unseen test examples. There exist different types of classification models: logistic regression, support vector machine (SVM), decision tree, Random Forest, and k-nearest neighbor (k-NN). The model's prediction accuracy is reported on the test dataset. k-fold cross-validation is a popular testing methodology where the labeled dataset is partitioned into k equal subsets, with k-1 subsets used for training and validation, and the remaining subset used for testing. The k accuracy results are finally averaged to obtain a single estimate.

Classical ML Models. Logistic regression is a linear classifier that finds a hyperplane to separate two classes. To distinguish multiple classes, "one-vs-rest" logistic regression trains a separate model for each class, predicting whether an example belongs to that class or not. SVM with radial basis kernel applies implicit transformations to the features to map them to a higher-dimensional space and use this to identify examples that help in separating classes. A decision tree classifies examples by learning a disjunction of conjunctive predicates. Random Forest is an ensemble model that learns multiple decision trees and predicts the mode of the classes given by individual trees. k-NN picks the plurality vote of k nearest training examples to a given test example.

CNNs. Convolution Neural Network (CNN) is a type of deep network that exploits spatial locality in data. CNNs contains layers of different types, each performing different transformation. A *Convolution* layer applies filters to the input in order to extract features (also called feature maps), where the filter weights are learned during training. A *ReLU* layer introduces non-linearity in CNN by applying an element wise operation and replaces all negative values in the feature maps by zero. A *Pooling* layer such as MaxPool and AvgPool reduces the dimensionality of each feature map while retaining the most important information. A *Fully Connected* layer is a traditional Multi Layer Perceptron (MLP) that has every neuron in a layer connected to every neuron in the next layer. A deep CNN stacks such layers multiple times.

2.2 Prior Art.

Tensorflow Data Validation (TFDV) is a tool to analyze and transform ML data in Tensorflow Extended (TFX) pipeline [8]. TFDV uses conservative heuristics to infer initial ML feature types from the descriptive statistics about the column. The users then review the inferred feature types and update them manually to capture any domain knowledge about the data that the heuristics might have missed. While this tool certain reduces manual effort in identifying ML feature types, it still requires users to manually go through individual descriptive statistics of all the columns, rather than the source data files. Pandas is a Python library that provides tools for data analysis and data transformations. It only infers syntactic types such as integer, float, or something else [18]. TransmogrifAI is an automated ML library for structured data in Salesforces' AutoML platform called Einstein [2]. TransmogrifAI supports rudimentary automatic feature type inference over primitive types such as Integer, Real and Text. It also has an extensive vocabulary for feature types such as email, phone numbers, zipcodes, etc. However, users have to manually specify these feature types for their data. Overall, all these prior art tools are limited in scope for feature type inference because they are primarily rule-based or syntactic.

2.3 Assumptions and Scope

Our current focus is on relational/tabular data, the most commonly analyzed form of data in practice [34]. Such datasets are typically stored with DB schemas in RDBMSs or data warehouses or as "schema-light" files (CSV, JSON, etc.) on data lakes and filesystems. Either way, we assume the dataset is a single table with all column names available. To build ML models on such data, the first thing most data scientists do is to load it into Python or R "dataframe." This is when the laborious process of preparing this dataframe for an ML training library (e.g., Scikit-learn) begins. This stage is the focus of our 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 "target model." For example, one might load a customer table to train a target model for predicting customer churn.

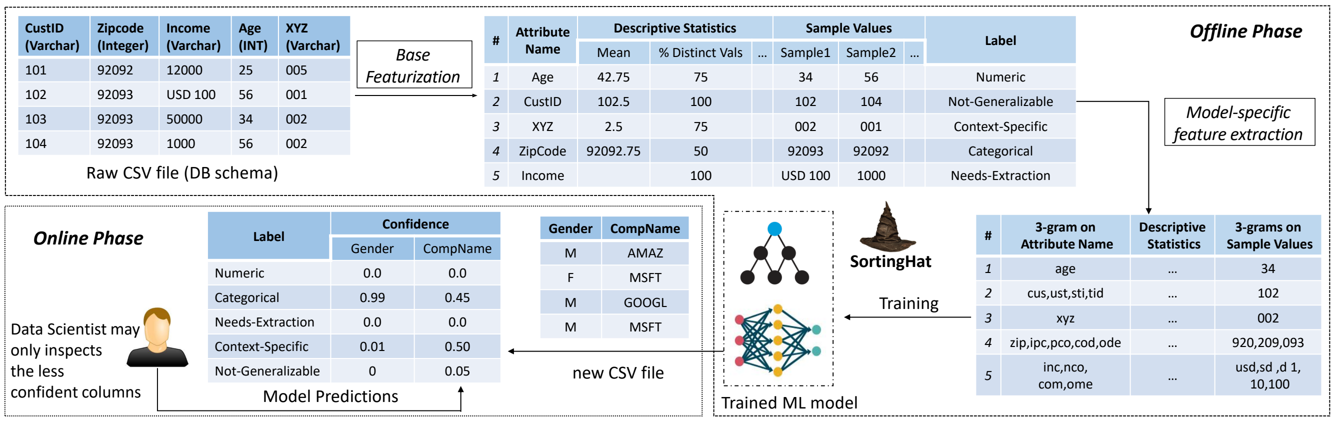
3. OVERVIEW OF OUR APPROACH

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 four components.

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 more classes to our label vocabulary to classify the attributes beyond just the numeric and categorical class. We explain this in depth in Section 4.1.

Labeled Dataset. We need a large labeled dataset for ML feature type inference. There exists no publicly available dataset for this task. Thus, we manually annotate a large labeled dataset containing more than 9000 examples. Section 4.2 to Section 4.4 covers this in depth.

Features. In order to identify signals that would be useful in the model building, we imitate the human intuition for ex-



CustID (Varchar)	Zipcode (Integer)	Income (Varchar)	Age (INT)	XYZ (Varchar)
101	92092	12000	25	005
102	92093	USD 100	56	001
103	92093	50000	34	002
104	92093	1000	56	002

#	Attribute Name	Descriptive Statistics			Sample Values			Label
		Mean	% Distinct Vals	...	Sample1	Sample2	...	
1	Age	42.75	75		34	56		Numeric
2	CustID	102.5	100		102	104		Not-Generalizable
3	XYZ	2.5	75		002	001		Context-Specific
4	ZipCode	92092.75	50		92093	92092		Categorical
5	Income		100		USD 100	1000		Needs-Extraction

#	3-gram on Attribute Name	Descriptive Statistics	3-grams on Sample Values
1	age	...	34
2	cus,ust,sti,tid	...	102
3	xyz	...	002
4	zip,ipc,pco,cod,ode	...	920,209,093
5	inc,nco, com,ome	...	usd,sd ,d 1, 10,100

Gender	CompName
M	AMAZ
F	MSFT
M	GOOGL
M	MSFT

Label	Confidence	
	Gender	CompName
Numeric	0.0	0.0
Categorical	0.99	0.45
Needs-Extraction	0.0	0.0
Context-Specific	0.01	0.50
Not-Generalizable	0	0.05

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

tracting features from the raw CSV file. We transform each column in the raw CSV file into a feature vector containing the column name, descriptive statistics about the column such as mean, standard deviation etc., and 5 randomly sampled column values. We call this step Base Featurization. We explain this step in depth in Section 4.3. Some ML models cannot operate on the raw characters of attribute names or sample values. Hence, we extract hand-crafted feature sets such as n-grams from the attribute names and sample values. We explain this step in depth in Section 5.

ML models. We finally use our feature set on our labeled dataset to build ML models. We compare several ML approaches in depth in Section 5.3 to Section 5.5.

The above steps are carried out once offline. In the online phase, a trained ML model can be used to infer feature types for columns in an "unseen" CSV file. We desire that the ML models should also output confidence scores for each class. Hence, this allows users to quickly dispose of easy features and prioritize their effort towards features with low confidence scores that need more human attention. Furthermore, our trained ML models can also help automate feature type inference in AutoML platforms and can help raise their overall performance.

4. OUR DATASET

This section discusses our efforts in creating the labeled dataset for the task of ML feature type inference: the label vocabulary, the data sources, the type of raw signals we extract from the columns, and the labelling process.

4.1 Label Vocabulary

The target model accepts only two classes: *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 need more classes. More importantly, there is a trade-off between having too many classes and not having enough labeled data versus too few classes to be useful in practice. We balance this trade-off by considering 5 intuitive classes: *Numeric*, *Categorical*, *Needs-Extraction*, *Not-Generalizable*, and *Context-Specific*. These classes represent the prediction vocabulary of our ML model. We now explain each class and give examples.

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*. ID attributes such as *CustID* or integers representing encodings of discrete levels does not belong to this class. Note that all numbers can always be represented as categories by discretizing them. Hence, it is indeed possible to give numbers as categorical feature. However, ML models benefit by operating on numbers directly because they will have infinite feature spaces. On the other hand, discrete set of categories is only a finite space. There is a loss of information going from numeric to a discrete category. Hence, a typical data scientist would give a numeric feature as a numeric feature and not discretize it to categories.

Categorical. These attributes contain qualitative values that can directly be utilized as *categorical* features for the target ML model. There are two major sub-classes of categorical features: nominal and ordinal. Ordinal attributes have a notion of ordering among its values, while nominal attributes do not have such a notion. For instance, *ShoeSize* is an ordinal attribute. While, *Zipcode* is a nominal attribute. Names and coded real-world entities from a known finite domain set are also categorical. For instance, *Gender* taking values from {M, F, O} is a categorical feature. Note that attributes such as *ShoeSize* and *ZipCode* are syntactically a number. Hence, we need 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.

Needs-Extraction. This class represents attributes with "messy" syntax that preclude its direct use as numeric or categorical feature. This class is orthogonal to the *Numeric* vs *Categorical* dichotomy because these attribute values require some form of processing before being used as features. The following examples illustrates this class.

(a) A number present along with string, denoting a measurement unit. e.g., *"30 Mhz"*, *"USD 45"* and *"500,000"*. In all cases, a number is typically extracted and the units are standardized (if applicable). A data scientist would typically write a regular expressions or custom Python/R scripts to extract usable feature values from such column, e.g., converting *"USD 45"* to 45.

(b) A text field with semantic meaning, consisting of either sentences, URL, address, geo-location, or even a list of items separated by a delimiter. The data scientist may choose to extract custom features, either numeric or categorical, or both through standard featurization routines. For instance, they can extract features such as n-grams, or even Word2Vec embeddings from an English sentence for the target model. Note that, such feature engineering decisions are not focus of this work, since they are mostly application-specific. We leave such downstream featurization routines for custom processing to the user.

(c) Date or time stamp, e.g., *"7/11/2018"*, and *"21hrs:15min:3sec."* The extracted features can be used as numeric or categorical, depending upon the feature extracted. For instance, month of the year can be categorical, while time can be numeric. Again, we leave such downstream feature engineering decisions to the user.

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 target 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 target model. One may obtain some feature from this attribute through more custom processing or domain knowledge. On the other hand, even though, attributes such as *Income* and *Date* has all unique values in its domain, they are generalizable. Thus, they belong to *Needs-Extraction* class, since it is highly likely that one can extract useful features from their domain.

Context-Specific. This class is for 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. For example, XYZ attribute has integer values but it is hard to decide if its feature type is numeric or categorical without any additional information. Clearly, answering this question 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.

We believe that our 5-class label vocabulary, while limited, is reasonable and useful. The label vocabulary can also give other insights to a data scientist. For instance, they could look for tables to join when faced with a large-domain *Categorical* feature such as *ZipCode*. They could offload attributes marked as *Needs-Extraction* to a software engineer for writing Python/R scripts. In addition, they could inspect the columns that are marked *Not-Generalizable* for any missing values or errors in data entry.

4.2 Data Sources

We obtain over 360 real datasets as CSV files from several sources such as Kaggle [3], UCI ML repository [4] and our prior work [25, 17]. Each attribute (or column) of the CSV file is just one example for our ML task. We obtain over 9000 such examples (or columns) from all data files put together.

4.3 Base Featurization

We replicate the human-level intuition into ML models by extracting the following information from the raw CSV files that a human reader would look at:

(1) Column name. Suppose, that a human would like to classify *ZipCode* into one of our five classes. Just by inspecting an interpretable name such as *ZipCode*, a human would know that the attribute would likely be *Categorical*. Hence, we extract the attribute name from the raw CSV file.

(2) Column values. A human would typically inspect some values in the column to make sure they make sense. For instance, if the human finds that some values are negative in the *ZipCode* column, then the column has to be treated appropriately for any missing values or data errors. Considering this, we extract 5 randomly sampled attribute values from the column.

(3) Descriptive statistics. Finally, the human would look at some descriptive statistics about the column. For instance, if the human finds that roughly 99.99% of the values in the column are NaNs, then the human would classify the *ZipCode* column as *Not-Generalizable*. Based on this observation, we extract several descriptive statistics for a column: percentage of distinct values out of total values, percentage of NaNs out of total values, 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 tokens in the values.

We summarize these extracted signals in a feature set that will be used to build ML models. Overall, as shown in Figure 3, we featurize the raw CSV files by extracting attribute name, descriptive statistics and 5 randomly sampled attribute values. Each column in the raw CSV files is an example (or row) in the new base featurized file. We have 9000 such examples.

4.4 Labelling Process

We followed the following process to reduce the cognitive load of labelling. Initially, we manually labelled 500 examples into a class. We then use Random Forest with 100 estimators to perform 5-fold nested CV. The model achieves a classification accuracy of around 74% on the test set (average across 5 folds). We use the trained model to predict a class label on all of the 9000 examples. This process allowed us to group together examples that were likely to belong to the same class label. Finally, we manually labelled all examples into one of the five classes. The labelling process took about 75 man-hours across 4 months. The complete labeled dataset is available on our project webpage [27].

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. 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 for the five 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, 4% of examples had 4 unique labels, 27% had 3, and 69% had 2; in the second run, these were 5%, 21%, and 42%. Majority voting gave the wrong answer in over half of the examples we randomly checked. We suspect such

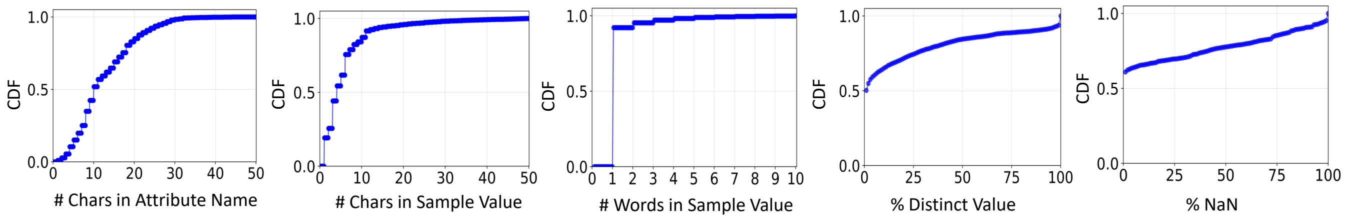


Figure 4: Cumulative distribution of descriptive statistics in the featurized data file.

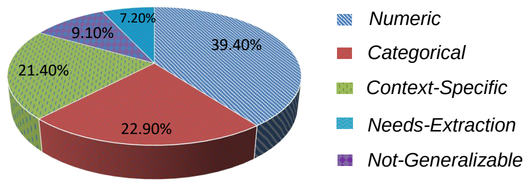


Figure 5: Distribution of 5-class labels on our labeled data.

	Numeric	Needs-Extraction	Categorical	Not-Generalizable	Context-Specific	Overall
# chars in Attribute Name	16	11	10	10	9	10
# chars in Sample Value	5	15	3	3	3	4
# words in Sample Value	1	2	1	1	1	1
% Distinct Vals	18.09	5.87	0.04	0.01	0.52	0.96
% NaNs	0	0.002	0	15.84	33.9	0

Table 1: Median of different Descriptive Statistics by class in the base featurized data file.

high noise arises because ML feature type inference 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 interesting avenue for future work.

4.5 Data Statistics

Figure 5 shows the distribution of class labels in our labeled dataset. We observe that majority of labels ($\sim$60%) are either *Numeric* or *Categorical*. While, *Needs-Extraction* and *Not-Generalizable* are less frequent in our dataset. Figure 4 plots the cumulative distribution functions (CDF) of different descriptive statistics obtained by base featurization. Table 1 reports the median for the same descriptive statistics. Due to space constraints, we provide a complete breakdown of the cumulative distribution by class in our technical report [26]. We observe that *Numeric* attributes have longer names than others. Attribute values for *Needs-Extraction*, as expected, have more number of characters and words than other classes. In addition, we observe that all sample values in *Numeric* and 80% of the sample values in *Categorical* are single token strings. Furthermore, we find that almost 90% of the attributes in *Categorical* have less than 1% unique values in its columns. Interestingly, *Not-Generalizable* have either very few unique values or only NaN values in their domain.

5. APPROACHES COMPARED

In this section, we initially discuss the type of features we extract from the base featurized file since some ML approaches do not operate at the raw character or word level. We then develop an intuitive rule-based approach as a baseline. Finally, we discuss how we apply several classical ML models, k-NN with a distance function tuned for our task and a convolution-based neural model.

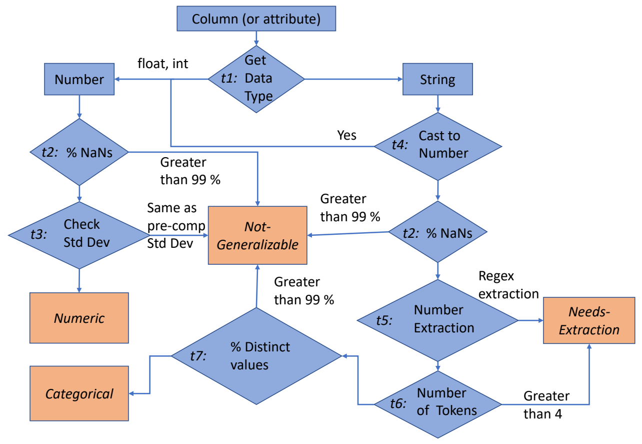


Figure 6: Flowchart of the rule based system. Diamond-shaped nodes are the decision nodes that represents a "check" on the attribute. The final outcome is shown in orange rectangular boxes.

5.1 Feature Extraction

We observe that attributes with similar names often belong to the same class. For instance, both attributes *temperature_in_jan* and *temperature_in_feb* are *Numeric*. Hence, a set of n consecutive characters extracted from both the words would be highly similar. Based on this intuition, we extract an n-gram feature set (set of all combinations of adjacent characters) 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* attribute which belongs to *Needs-Extraction* class. We use an one-hot encoding-based representation to obtain a feature vector where each feature indicates the presence or absence of that n-gram.

Notation. We denote the descriptive statistics by $\mathbf{X}_{stats}$, attribute name by $\mathbf{X}_{name}$ and a set of 5 randomly 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 the 2-gram and 3-gram features on the attribute name by $\mathbf{X2}_{name}$ and $\mathbf{X3}_{name}$ respectively. Similarly, we denote the 2-gram and 3-gram features 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.

5.2 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 as shown in Figure 6. 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. Note that *Context-Specific* is not included in the rule-based approach. We manually recognized this class when labelling, but automating this in rule is difficult. 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 *Not-Generalizable*.

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 *Not-Generalizable*. 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 *Needs-Extraction*.

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 *Needs-Extraction*.

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 *Not-Generalizable*; otherwise, we classify it as *Categorical*.

The above steps are not exhaustive, and it is indeed possible to craft more rules. But, it is highly cumbersome and perhaps even infeasible to hand-craft a perfect rule-based classifier. Thus, we instead leverage existing ML algorithms for this classification task.

5.3 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. 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 [21]. $f(x)$ 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.

5.4 Nearest Neighbor

k-NN is one of the oldest, yet powerful classifiers[10]. Its accuracy is influenced by two main factors: distance function used to determine the nearest neighbors and the number of neighbors used to classify a new example. 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:

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

In the above, 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. The confidence score for a class at prediction time is given by the number of neighbors out of k neighbors that voted for that class.

5.5 Neural Model

For text understanding tasks such as sentiment analysis and text classification, character-level CNNs have achieved state-of-the-art results [39, 38, 16]. Inspired by this, we propose a neural model for our task as shown in Figure 7(A). The layers of CNN are shown in Figure 7(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 the 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 the 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. The CNN architecture 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.

	TFDV		Pandas		TransmogrifAI		Log Reg		RBF-SVM		k-NN		Random Forest		Neural Model	
	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	0.5117	0.9876	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	0.9915	0.4166	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	0.6359		0.6667		0.6451		0.9394		0.9512		0.9633		0.9508		0.9521	

Table 2: 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	Train	0.5652	0.8802	0.8959	0.9233	0.9361	0.9004	0.9266	0.9496	0.9566
	Validation	0.5639	0.7522	0.8128	0.8284	0.8308	0.7953	0.8288	0.8408	0.8443
	Test	0.5482	0.7609	0.8120	0.8341	0.8415	0.7609	0.8422	0.8582	**0.8607**
RBF-SVM	Train	0.8204	0.9169	0.9348	0.9278	0.9382	0.8941	0.9274	0.9489	0.9513
	Validation	0.7400	0.7964	0.8482	0.8542	0.8532	0.7888	0.8497	0.8580	0.8612
	Test	0.7351	0.7999	0.8565	0.8695	0.8706	0.7976	0.8587	0.8696	**0.8742**
Random Forest	Train	0.9215	0.9155	0.9705	0.9616	0.9707	0.8770	0.9697	0.9643	0.9545
	Validation	0.8143	0.7997	0.8830	0.8715	0.8816	0.7704	0.8857	0.8761	0.8622
	Test	0.8118	0.7977	0.8923	0.8802	0.8871	0.7736	**0.8939**	0.8797	0.8683

Table 3: 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.

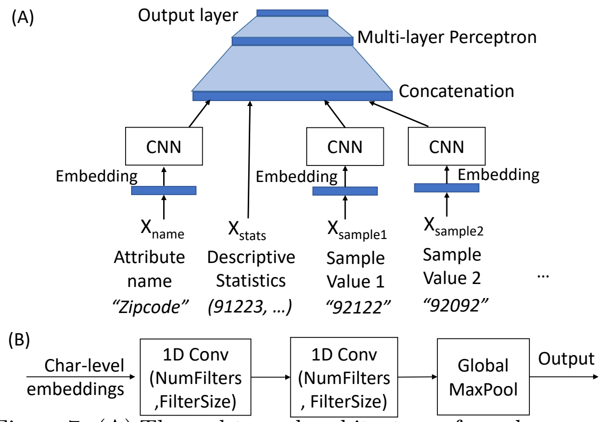


Figure 7: (A) The end-to-end architecture of our deep neural network. (B) The CNN block's layers.

6. 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 neural model. Finally, we present the prediction runtime of all models on a given example.

6.1 Methodology, Setup and Metrics

Methodolody. We partition our labeled dataset into train and held-out test set with 80:20 ratio. We then 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 the neural model, 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 $\{10^{-3}, 0.01, 0.1, 1, 10, 100, 10^3\}$.

Neural 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 neural model, 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 $\{5 \cdot 10^3, 10^4\}$.

Experimental Setup. All experiments were run on CloudLab [22]; we use a custom OpenStack profile running Ubuntu 16.10 with 10 Intel Xeon cores and 64GB of RAM.

Metrics. Our key metric is prediction accuracy, defined as the diagonal of the 5 x 5 confusion matrix. We also report the per-class accuracy and their confusion matrices.

6.2 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 5-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 2 presents the precision, recall and overall classification accuracy results on the test set.

Results. We notice a huge lift of $\sim$30% in accuracy for our 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, 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 among all approaches.

6.3 End-to-End Accuracy Results

Rule-based Heuristic. The overall 4-class classification accuracy of rule-based baseline heuristic on the held-out test set is *0.7160*. Table 5(A) shows its confusion matrix. Note that it excludes *Context-Specific* class. We observe that the rule-based approach achieves 97% accuracy in classifying examples belonging to *Numeric* class. This is because when presented with any number, it does not have enough rules to confuse it into labelling it as another class. On the other hand, on *Categorical* examples, it achieves only 37% accuracy. This is mainly because when it is presented with any number encoded as a category, it mistakenly classifies as *Numeric*. This approach achieves an high accuracy of 86% for *Not-Generalizable* as it is simpler to come up with rules that captures this class. Admittedly, our rules are not exhaustive and one can always come up with more rules to improve the performance of the approach. However, writing rules for every little corner case is excruciating and will likely never be comprehensive.

Classical ML Models. Table 3 presents the 5-class accuracy results of the classical ML models using different combinations of the feature sets. For logistic regression, we see that the descriptive statistics alone are not enough, as it achieves an accuracy of just 55% on the held-out test set. But, for RBF-SVM and Random Forest, we notice that the accuracy with descriptive statistics alone is already 74% and 81% respectively. Incorporating the 2-gram features of the attribute name into logistic regression leads to a whopping 26% lift in accuracy on the test set. Random Forest achieves a massive 89% accuracy using just 2-gram feature set along with descriptive statistics. This underscores the importance of 2-gram features on the attribute names.

Adding 2-gram features of a random sample value lifts the accuracy further by 2% for logistic regression. However, as the complexity of the model increases, the 2-gram features on a sample value do not provide boost in accuracy. Adding 2-gram features of more sample values does not give any rise in accuracy, except for logistic regression. More complex features such as 3-gram on logistic regression model leads to more significant gains in accuracy relative to the 2-gram features. However, for complex models such as RBF-SVM and Random Forest, we notice that the lift in accuracy with 3-gram features is only marginal relative to the 2-gram features. Overall, Random Forest achieves the best 5-class accuracy of 89% using the 3-gram features on the attribute name along with descriptive statistics.

Table 5(B) shows confusion matrix of Random Forest. We observe that it does well in predicting *Numeric* and *Categorical* classes, achieving a recall of 96% and 94%, respectively. On the other hand, recall for *Not-Generalizable* is only 74%, which is worse than the rule-based heuristic. This is because many examples belonging to *Not-Generalizable* are confused with *Categorical.* We also observe that Random Forest is skewed towards classifying many examples as *Categorical.* As a result, it has more chances of confusing any class with *Categorical.* We analyze the behavior of Random Forest in depth in Section 6.4.

Nearest Neighbor. Table 4 presents the 5-class accuracy of k-NN. We observe that when we use only Euclidean distance on descriptive statistics, the accuracy is already 74%. With only edit distance on attribute name the accuracy is 80%. Finally, with our weighted edit distance function from Section 5.4, k-NN achieves a massive 89% accuracy, comparable to Random Forest. Table 5(C) shows confusion matrix of k-NN. We see it does well in predicting *Numeric* and *Context-Specific*, achieving a recall of 96% and 86%, respectively. We also see it is skewed towards classifying many examples as *Context-Specific.* As a result, it has more chances of confusing any class with *Context-Specific.*

Neural Model. Table 4 presents the accuracy of neural model on different feature sets. We see that with just X_{name}, this model already achieves an accuracy of 81%. The descriptive statistics lift the accuracy further by 7%. We notice that sample values are not that useful here; they yield only minor lift in accuracy. Table 5(D) shows the confusion matrix. We see it does well in predicting *Needs-Extraction* and *Not-Generalizable* classes, achieving a recall of 84% and 83%, respectively. On the other hand, recall for *Categorical* is only 88%, which is worse than Random Forest. This is because neural model confuses many examples belonging to *Categorical* with *Numeric* class. We also see that neural model is not particularly skewed towards classifying examples towards any class; rather it distributes examples over different classeses proportionately, relative to other models.

6.4 Analysis of Errors

We now explain the behavior of Random Forest and the neural model on our dataset by inspecting the raw datatype of the attribute values. We categorize the data type into

Model		$[X_{stats}]$	$[X_{name}]$	$[X_{stats}, X_{name}]$	$[X_{sample1}]$	$[X_{name}, X_{sample1}]$	$[X_{stats}, X_{sample1}]$	$[X_{stats}, X_{name}, X_{sample1}]$	$[X_{stats}, X_{name}, X_{sample1}, X_{sample2}]$
Neural	Train	0.6885	0.9297	0.9692	0.6906	0.9622	0.8636	0.9854	0.9902
	Validation	0.6839	0.8088	0.8851	0.5868	0.8414	0.7466	0.8720	0.8715
	Test	0.6839	0.8138	0.8803	0.5927	0.8471	0.7560	0.8831	**0.8834**
k-NN	Validation	0.7345	0.8004	0.8841	N/A				
	Test	0.7380	0.8023	**0.8876**					

Table 4: Training, cross-validation and held-out test accuracy of Neural and k-NN model with different feature sets.

(A) Rule-based Heuristic	Numeric	Needs-Extraction	Categorical	Not-Generalizable	Context-Specific
Numeric	688	0	1	17	-
Needs-Extraction	34	43	60	6	-
Categorical	239	30	161	1	-
Not-Generalizable	8	2	13	144	-
Context- Specific	-	-	-	-	-

(B) Random Forest	Numeric	Needs-Extraction	Categorical	Not-Generalizable	Context-Specific
Numeric	676	1	10	1	18
Needs-Extraction	2	114	22	0	5
Categorical	5	11	403	2	10
Not-Generalizable	4	6	24	124	9
Context- Specific	31	7	27	2	337

(C) k-NN	Numeric	Needs-Extraction	Categorical	Not-Generalizable	Context-Specific
Numeric	675	1	10	3	17
Needs-Extraction	4	111	14	4	10
Categorical	11	20	379	9	12
Not-Generalizable	5	8	16	132	6
Context- Specific	17	7	23	11	346

(D) Neural Model	Numeric	Needs-Extraction	Categorical	Not-Generalizable	Context-Specific
Numeric	616	0	15	5	25
Needs-Extraction	5	120	8	2	8
Categorical	16	13	380	13	9
Not-Generalizable	3	5	14	139	6
Context- Specific	27	7	27	7	336

Table 5: 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) Neural model.

several categories such as integers, floats, negative numbers, dates, sentences with one token, and sentences with more than one token. Table 6 (resp. Table 7) shows the the confusion matrix of the predicted class by Random Forest (resp. Neural model) vs actual datatype of the attribute value on the test set. Table 8 shows examples of the attributes and the corresponding prediction made by Random Forest and the neural model. We explain the errors by class below.

Numeric. We see that when the actual label is *Numeric* (Table 6, 7(A)), Random Forest and the neural model 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 *Context-Specific* class. For instance, *s3area* (Table 8 example(A)) is predicted as *Context-Specific*. 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 neural model captures this human-like intuition. For other attributes involving sub-strings such as count, value, etc., we observe the same trend: the neural model captures this human-like intuition.

Needs-Extraction. As shown in Table 6 (B), Random Forest does well in predicting *Needs-Extraction* when the column values are strings with number of tokens greater than one. While, on one-token strings, Random Forest gets confused with *Categorical*. For instance, a one-token string such as "9years", as shown in table 8 example(C) is predicted as *Categorical*. Again, it seems that Random Forest, unlike the neural model, is missing the human-level intuition that an attribute can have values such as "years" or "months" embedded in a sample value.

Categorical. Table 6(C) showcases the same behaviour. We again observe that when the sample values are sentences with number of tokens greater than 1, there is more chance for Random Forest to misclassify *Categorical* as *Needs-Extraction*. On the other hand, for one-token strings, it does well for *Categorical*. Table 8 example(E) shows an attribute *"SMOD_POPULATION"* which has values such as "-9999" as part of the encoded category. Random Forest classifies it as *Numeric*. We observe that the neural model performs worse than Random Forest in predicting *Categorical* correctly. This is because when the number of unique values in a column relative to the domain size is extremely low (Table 8 example(G)), the neural model classifies such examples as *Not-Generalizable*. Moreover, the neural model misclassifies the categories encoded as integers to numeric type in some cases, as Table 8 example(H) shows.

Not-Generalizable and Context-Specific. From Table 6, 7(D), we notice that both the models often confuse *Not-Generalizable* with *Categorical*. Table 8 example(F) shows an attribute with only 2 sample values: "*NULL!*" and "*Others*". Hence, the actual ground-truth label is *Not-Generalizable*. But, Random Forest treats "*NULL!*" as a separate category. Furthermore, we see from Table 6, 7(E) that both the models often classify *Context-Specific* as *Numeric* when the values are integers. This is because they lack in their semantic understanding ability to accurately identify the attributes with meaningless names.

6.5 Prediction Runtimes

We evaluate the running time of the ML models in the online phase, i.e, for inference to make predictions on an attribute/column. As shown in Figure 3, we must first perform Base Featurization of the column and then model-specific feature extraction. The Base Featurization is a common step across all the models. Model-specific feature extraction is only needed for the classical ML models. Figure 8 show the runtime of all 5 models, with breakdowns of Base Featurization, model-specific feature extraction time, and inference time. The measurements were made on the test

	(A) Numeric			(B) Needs-Extraction			(C) Categorical			(D) Not-Generalizable			(E) Context-Specific		
	Integers	Floats	Negative Numbers	Sentence (length > 1)	Sentence (length = 1)	Dates	Numbers	Sentence (length > 1)	Sentence (length = 1)	Numbers	Sentence (length > 1)	Sentence (length = 1)	Numbers	Sentence (length > 1)	Sentence (length = 1)
Numeric	386	284	194	0	2	0	7	0	0	3	0	0	31	0	1
Needs-Extraction	0	0	0	70	44	23	0	10	1	1	3	4	0	5	2
Categorical	6	2	0	9	13	3	207	72	122	7	2	14	11	6	9
Not-Generalizable	1	0	0	1	0	0	0	1	2	49	8	68	2	0	0
Context-Specific	17	7	4	1	3	0	6	2	1	8	0	0	295	18	24

Table 6: Breakdown of Random Forest 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 (A) *Numeric*, (B) *Needs-Extraction*, (C) *Categorical*, (D) *Not-Generalizable*, and (E) *Context-Specific*.

	(A) Numeric			(B) Needs-Extraction			(C) Categorical			(D) Not-Generalizable			(E) Context-Specific		
	Integers	Floats	Negative Numbers	Sentence (length > 1)	Sentence (length = 1)	Dates	Numbers	Sentence (length > 1)	Sentence (length = 1)	Numbers	Sentence (length > 1)	Sentence (length = 1)	Numbers	Sentence (length > 1)	Sentence (length = 1)
Numeric	381	280	194	4	5	0	10	0	3	2	1	1	25	0	1
Needs-Extraction	0	0	0	65	47	26	0	6	0	0	3	1	0	3	1
Categorical	8	1	0	9	4	0	203	75	114	5	2	10	8	12	8
Not-Generalizable	3	0	0	0	0	0	2	2	8	57	6	74	11	0	1
Context-Specific	18	12	4	3	6	0	5	2	1	4	1	0	295	14	25

Table 7: Breakdown of neural model's prediction for different types of sample 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 (A) *Numeric*, (B) *Needs-Extraction*, (C) *Categorical*, (D) *Not-Generalizable*, and (E) *Context-Specific*.

#	Attribute Name	Sample Value	% Distinct Val	% NaNs	Label	RF Prediction	Neural Prediction
A	s3area	579	0.04	0.96	*NU*	*CS*	*NU*
B	waitlist_count	45	0.09	0	*NU*	*CA*	*NU*
C	q1AgeBeginCoding	9years	0.04	0	*NX*	*CA*	*NX*
D	job_title	IT Support Technician	85.3	0	*CA*	*NX*	*NX*
E	SMOD_POPULATION	-9999	0.1	0	*CA*	*NU*	*CA*
F	q17HirChaOther	#NULL!	0.008	30.24	*NG*	*CA*	*NG*
G	SchoolDist2	0	0.001	0.24	*CA*	*CA*	*NG*
H	outcome_month	5	0.04	0	*CA*	*CA*	*NU*

Table 8: Examples for illustrating errors made by Random Forest and neural model. *NU* refers to *Numeric*, *NX* refers to *Needs-Extraction*, *CA* refers to *Categorical*, *NG* refers to *Not-Generalizable*, and *CS* refers to *Context-Specific*.

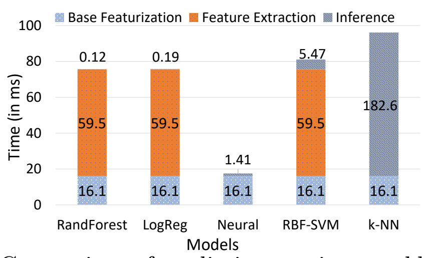


Figure 8: Comparison of prediction runtimes and breakdown for all models. Base Featurization is common for all models. Model-specific feature extraction is needed only for the 3 classical ML models.

set and averaged. All the models run in well under 1 sec. We also see that for the classical models, the additional feature extraction dominates overall runtime. Since SVM and k-NN are distance-based ML models, they require highest amount of time for inference. Overall, the neural model is the fastest in inferring the ML feature type.

6.6 Public Release and Leaderboard

We have released a live public repository on GitHub with our entire labeled data for the ML feature type inference task [5]. We have also released our pre-trained ML models in Python: k-NN, logistic regression, RBF-SVM, Random Forest, and the character-level neural model. The repository tabulates the accuracy of all these models. The repository includes a leaderboard for public competition on the hosted dataset with 5-class classification accuracy being the metric. In addition, we also release the raw 360 CSV files and we invite researchers and practitioners to use our datasets and contribute to create better featurizations and models.

7. DISCUSSION AND TAKEAWAYS

Tool Release: We make all of our models and featurization routines available for use by wrapping them under functions in a Python library. The Python package is available on our project webpage [27]. The data scientist can use the library directly by giving their raw CSV data files in form of a dataframe as input. The dataframe will be processed and featurized by our library functions, and for each attribute, a ranking of the classes from the label vocabulary based on confidence score will be given as output.

For Practitioners. Our trained ML models can be integrated for feature type inference into existing data prep environments. Programmatic tools such as TFDV and Pandas can leverage our models through APIs. Even, AutoML system developers can productionize our models to enhance their AutoML systems for data prep. For visual tools such as Excel and Trifacta [36], designing new user-in-the-loop interfaces that account for both model's prediction and human's judgement remains an open research question. We leave this integration to future work.

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. For instance, inspecting an attribute "*q17HirChaOther*" (example(F) from Table 8) with value "*NULL*", a human can tell that the value denotes a missing or corrupt entry and is not supposed to be taken as a category. Hence, designing a featurization scheme that can capture the semantic meaning of attribute values is an open research question.

Second, capturing more semantic knowledge of attributes in a neural architecture is an open research question. Although our current neural model captures several human-level intuitions, it sometimes fails in recognizing categories encoded as integers. For instance, looking at the attribute with name "month" and sample values from 1 to 12, humans know that the attribute is categorical feature.

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 model get confused. For instance, we see that when strings such as "-999" are used as categories, almost all ML models treat them as numbers and predict *Numeric*. Getting more labelled examples can potentially teach the model to learn better. In addition, we observe that attribute names are most useful for prediction. Hence, one can try getting more labeled attribute names without sample values. Finally, one can also design richer label vocabularies. For instance, we showed in Tables 6 and 7 a breakdown of Random Forest and neural model predictions on different types of sample values. One can consider training models end-to-end using such a fine-grained label vocabulary as well.

8. RELATED WORK

ML Data Prep and Cleaning. ML feature type inference has been explored in some prior tools [8, 2, 1, 18]. Tensorflow Data Validation is a rule-based tool that infers ML feature types from summary statistics about the column [8]. Pandas is a Python library that provides syntactic type inference [18]. TransmogrifAI is a library used for data prep, feature engineering, and ML model building in Salesforces' Einstein AutoML platform [2]. It provides ML feature type inference over primitive types such as integer, real number or text. AutoML Tables is another AutoML platform for structured data from Google [1]. It also suggests a feature type for each attribute of the imported CSV file. Since, it is available as a commercial tool, we do not have the availability to compare it with our work. Compared to existing open-source tools, our ML-based approach raises accuracy of ML feature type inference substantially by more than 30%.

DataLinter is a rule-based tool that inspects a data file and raises potential data quality issues as warnings to the user [15]. 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]. However, it does not handle ML feature type inference. Hence, they are orthogonal to our focus.

There are numerous tools that allows users to perform data transformations tasks for data prep. Trifacta [36] has visual data prep tools that allows users to perform tasks such as substring extraction, value standardization etc. with a human-in-the-loop. Programming-by-example (PBE) tools such as FlashFill [13, 14] and [29] allows users to perform syntactic string transformation tasks without the need to write a program by learning from input/output examples. These tools can be utilized to perform extraction of values that are embedded into strings, the *Needs-Extraction* class. However, they can run into scalability issues as the set of programs that are consistent with the examples would be huge. In any case, their goals are orthogonal to our focus.

Database Schema Inference. Inference for DB schema has been explored in some prior work. Googles BigQuery does syntactic schema detection when loading data from external data warehouses [6]. [7] infers a schema from JSON datasets by performing 2 operations: *map* operation to infer a data type for each value, and *reduce* operation that fuses inferred types using pre-defined rules. Thus, DB schema inference task is syntactic and rule-based. For instance, the type of the attribute with integer values has to be identified as an integer. In contrast, ML feature type inference is semantic and attributes with type integer can be categorical.

AutoML Platforms. Several AutoML systems such as AutoWeka [30] and Auto-sklearn [12] 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 any data prep tasks in the ML workflow. AutoML platforms such as Einstein AutoML [35] and AutoML Tables [1] 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 the data prep tasks and create benchmark labeled dataset 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 these AutoML platforms to improve their accuracy in the future.

Data/Model Repositories. OpenML [37] is an open-source collaborative repository for ML practitioners and researchers to share their models, datasets, and workflows for reuse and discussion. Our models and labeled datasets can be made available to OpenML community to invite more contributions on automated data prep tasks [24]. Hence, our work is complementary to OpenML.

9. REFERENCES

[1] Google automl tables, Accessed April 25, 2018. `https://cloud.google.com/automl-tables/docs/data-types`.

[2] Transmogrifai: Automated machine learning for structured data, Accessed April 25, 2018. `https://transmogrif.ai/`.

[3] Kaggle datasets, Accessed February 15, 2018. `https://www.kaggle.com/datasets`.

[4] Kaggle datasets, Accessed February 15, 2018. `https://archive.ics.uci.edu/ml/index.php`.

[5] The ml data prep zoo repository, Accessed March 15, 2018. `https://github.com/pvn25/ML-Data-Prep-Zoo`.

[6] Schema detection bigquery, Accessed March 15, 2018. `https://cloud.google.com/bigquery/docs/schema-detect`.

[7] M.-A. Baazizi, H. B. Lahmar, D. Colazzo, G. Ghelli, and C. Sartiani. Schema inference for massive json

datasets. In *Extending Database Technology (EDBT)*, 2017.

[8] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395. ACM, 2017.

[9] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[10] T. M. Cover, P. E. Hart, et al. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.

[13] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *ACM Sigplan Notices*, volume 46, pages 317–330. ACM, 2011.

[14] S. Gulwani, W. R. Harris, and R. Singh. Spreadsheet data manipulation using examples. *Communications of the ACM*, 55(8):97–105, 2012.

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

[16] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[17] A. Kumar, J. Naughton, J. M. Patel, and X. Zhu. To Join or Not to Join? Thinking Twice about Joins before Feature Selection. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, 2016.

[18] W. McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14, 2011.

[19] T. Mitchell, B. Buchanan, G. DeJong, T. Dietterich, P. Rosenbloom, and A. Waibel. Machine learning. *Annual review of computer science*, 4(1):417–433, 1990.

[20] T. M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[21] J. Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[22] R. Ricci, E. Eide, and C. Team. Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications. *; login:: the magazine of USENIX & SAGE*, 39(6):36–38, 2014.

[23] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment*, 11(12):1781–1794, 2018.

[24] V. Shah and A. Kumar. The ml data prep zoo: Towards semi-automatic data preparation for ml. In *Proceedings of the third Workshop on Data Management for End-to-End Machine Learning*, DEEM'19. ACM, 2019.

[25] V. Shah, A. Kumar, and X. Zhu. Are key-foreign key joins safe to avoid when learning high-capacity classifiers? *Proceedings of the VLDB Endowment*, 11(3):366–379, 2017.

[26] V. Shah, P. Kumar, K. Yang, and A. Kumar. SortingHat: Towards Semi-Automatic ML feature type inference (Technical Report). `https://adalabucsd.github.io/sortinghat.html`.

[27] V. Shah, P. Kumar, K. Yang, and A. Kumar. Towards Semi-Automatic ML feature type Inference. `https://adalabucsd.github.io/sortinghat.html`.

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

[29] R. Singh and S. Gulwani. Transforming spreadsheet data types using examples. In *Acm Sigplan Notices*, volume 51, pages 343–356. ACM, 2016.

[30] C. Thornton, F. Hutter, H. H. Hoos, and K. 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.

[31] H. Trevor, T. Robert, and F. JH. The elements of statistical learning: data mining, inference, and prediction, 2009.

[32] `https://cloud.google.com/automl/`. Google cloud automl, Accessed May 6, 2018.

[33] `https://www.figure-eight.com/figure-eight-2018-data-scientist-report`. Crowdflower 2018 data science report., 2019.

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

[35] `https://www.salesforce.com/video/1776007`. Salesforce einstein automl, Accessed February 15, 2018.

[36] `https://www.trifacta.com/`. Trifacta: Data wrangling tools software, Accessed April 27, 2018.

[37] J. Vanschoren et al. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 2014.

[38] X. Zhang and Y. LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

[39] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

APPENDIX

A. DATA STATISTICS

Figure 10 shows the CDF of the several descriptive statistics by class (A) *Numeric*, (B) *Needs-Extraction*, (C) *Categorical*, (D) *Not-Generalizable*, and (E) *Context-Specific*. Table 9 presents the mean, standard deviation and the maximum of the same descriptive statistics.

B. RANDOM FOREST: EXPLANATIONS

We quantitatively explain the relevance of different features using Gini importance for the random forest model [9]. Figure 9 shows the normalized importance of different features from X_{stats}. Higher the value, the more important is the feature and more is the feature used in internal feature selection and partitioning in the decision trees. We observe that features like % distinct values, standard deviation, % NaNs and length of sample value are used repeatedly for partitioning inside decision trees. On the other hand, custom features like castability and extractability are less relevant for making prediction.

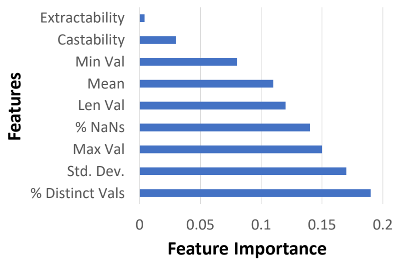


Figure 9: Gini feature importance score of the Random Forest model. y-axis shows different features from X_{stats} and x-axis shows the normalized importance score.

Statistics	Overall			Numeric			Needs-Extraction			Categorical			Not-Generalizable			Context-Specific		
	Avg	Std Dev	MaxVal	Avg	Std Dev	MaxVal	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	11.65	9.35	43	11.42	6.42	49	11.03	6.16	33	8.49	4.56	64
Number of chars in Sample Value	16.4	286.19	29.6K	5.98	5.39	398	152	1064	29.6K	6	10	150	7.96	33.3	689	5.22	7.36	141
Number of words in Sample Value	2.66	46.67	4900	1	0.08	7	22.4	174	4900	1.3	1.14	21	1.5	4.4	89	1.13	0.75	21
Mean	1.65E+14	1.03E+16	8.8E+17	3.6E+10	1.03E+12	5.6E+13	2.7E+12	2.4E+11	4.6E+13	2.5E+5	6.1E+6	2.05E+8	1.2E+10	3.2E+11	9.7E+12	7.7E+14	2.2E+16	8.8E+17
Standard Deviation	3.34E+15	1.31E+17	5.4E+18	1.9E+12	9.9E+13	5.9E+15	1.3E+13	2E+14	4.8E+15	2E+5	5.3E+6	2E+8	1E+9	2.2E+10	4.9E+11	1.6E+16	2.8E+17	5.4E+18
% Distinct vals	19.3	31.58	100	28.8	31.1	100	32.2	39.2	100	2.4	11.8	100	24.1	42.6	100	13.1	28.8	100
% NaNs	22.4	34.7	100	12.6	27.8	99.97	22.8	35.9	99.98	18.7	32.5	99.99	41.1	45.2	100	35.5	34.9	99.98

Table 9: Average, standard deviation, and maximum value of different *descriptive statistics*.

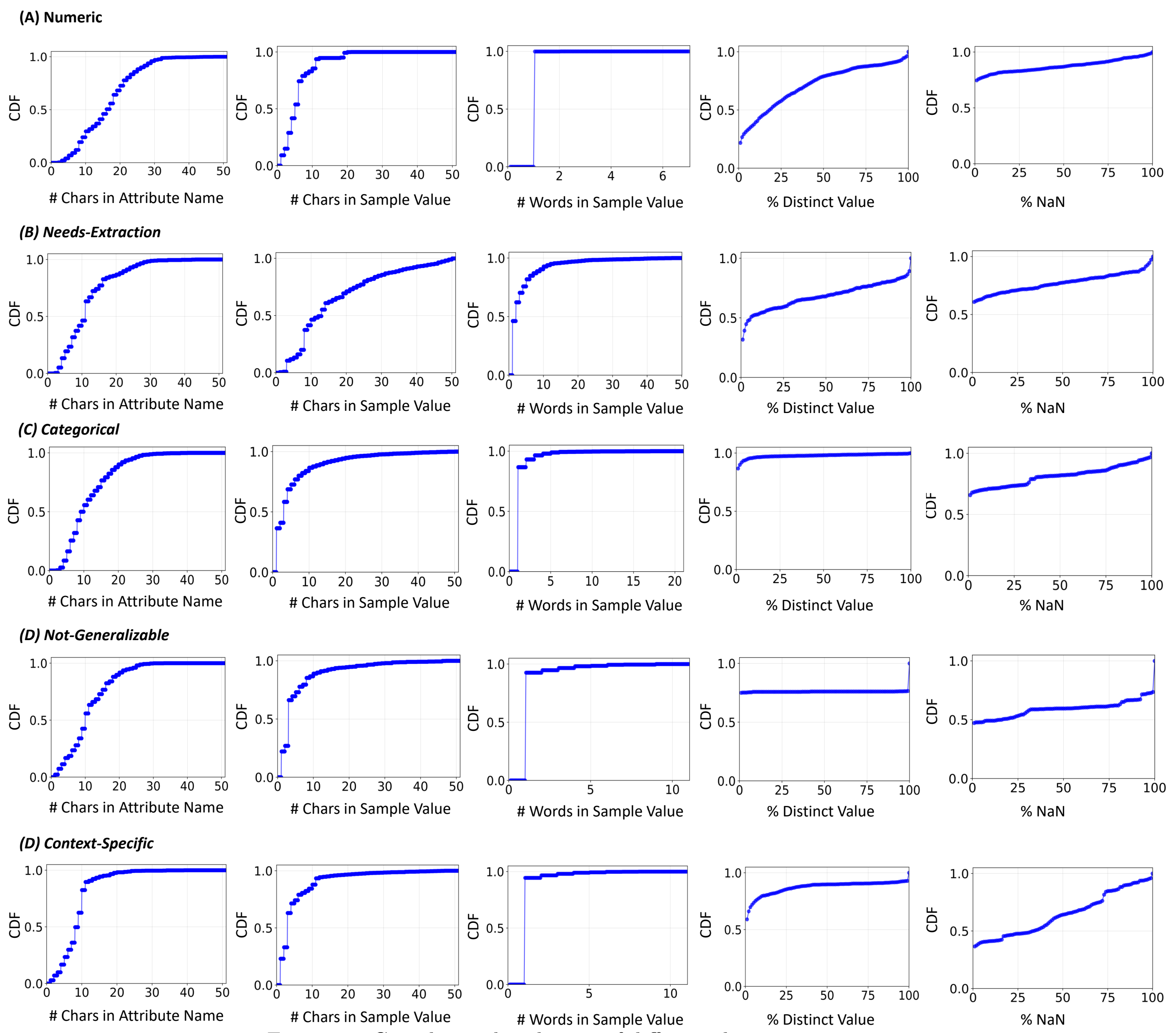


Figure 10: Cumulative distribution of different *descriptive statistics*.