Scaling Feature Type Inference

ANDREW ZHOU* and ERIC MANIRASITH*, University of California San Diego, USA

Additional Key Words and Phrases: datasets, neural networks, replication, data preparation, AutoML

1 INTRODUCTION

The field of Data Science and Machine Learning has seen intense growth over the past decade, specifically in automation. With the introduction of automating the entire ML pipeline now anyone can perform basic machine learning task. To ease the cost of adoption and for industry use, end-to-end automated machine learning (AutoML) systems have been released in the form of commercial cloud services including Google's Cloud AutoMl and Saleforce's Einstein. Hence, in the current industry AutoML is very successful in feature engineering, model selection, and hyper-parameter tuning. However, other steps in the pipeline such as feature type inference has been largely ignored in AutoML.

Nevertheless, manually cleaning the data and performing feature type inference takes the most amount of time for ML engineers and data scientists alike, because of datasets having millions of features in real life production settings. For AutoML feature type inference to be ready for industry scale it has to be automated, but still accurate. To solve this, Shah et al. has created a ML Data Prep Zoo containing a labelled benchmark dataset of features found in the wild, along with source code and pre-trained models for automated feature type inference.

2 BACKGROUND

The current rise of AutoML has led to many new tools for the automation of the Machine Learning pipeline. There are industry tools that do automate the cleaning/feature extraction of Machine Learning, such as TransmogrifAI in Einstein, Tensorflow Data Validation (TFDV) in TensorFlow Extended, and AutoGluon from AWS. However, most industry AutoML platforms fail to accurately automate data cleaning/feature extraction part of the Machine learning pipeline. These industry tools fail to have any objective evaluation measure to compare how accurate or inaccurate they can be. As a result, Shah et al. have created a ML Data Prep Zoo that is able to automate "data cleaning" or specifically feature type inference with consistent accuracy. With the use of a labelled benchmark data set of features found in the wild, Shah et al. have been able to conclude that using the Random Forest model results in the greatest accuracy with each feature type. Therefore, with renew interest in new AutoML techniques there is a push to create scalable platforms for industry use.

3 METHODS

3.1 Task

Project SortingHat has proven that automated feature type inference can be accurately done. This can be seen with the Random Forest model achieving a high accuracy of 91.2 when inferring feature types according to Shah et al.. Therefore, we intend to scale this automation, so that it may be industry ready.

Figure 1 shows the entire workflow of ML-based feature type inference. The first step or base featurization takes on average the longest amount of time during auto feature type inference and as a result the whole workflow is bottle necked at this step. This is because base featurization is the only step in the workflow that iterates through the whole dataframe including every row and column. Therefore, base featurization has a time complexity of $O(Row * Column)$,

*Both authors contributed equally to this research.

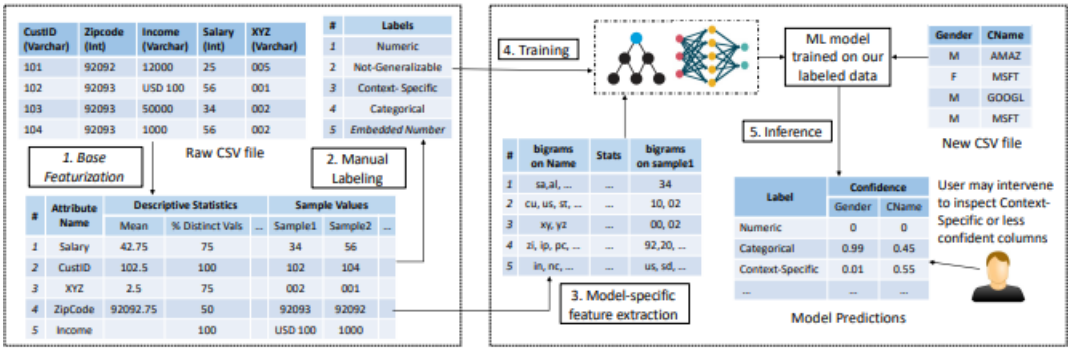

Fig. 1. Workflow showing how our data is used for ML-based feature type inference. (Recreation of Figure 4 from Shah et al.)

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

Fig. 2. All the descriptive stats used for base featurization. (Recreation of Table 6 from Shah et al.)

while the actual inference (step 5) itself only has a time complexity of $O(Column)$. The base featurization step itself is when we take the raw csv and extract specified features to use in training the model, including: descriptive statistics (figure 2) and five sample values. Our task then is to scale the base featurization.

3.2 PySpark

To scale the base featurization step, we reimplement the SortingHat base featurization algorithm in PySpark. Like the Python code, we assume that the input PySpark DataFrame is read with initial datatype inference. In Pandas, this is done by default; in PySpark, the `inferSchema` argument of `DataFrameReader.csv()` must be set to `true`.

Unfortunately, the PySpark DataFrame's data model does not support efficient sampling, as operations crossing the Python/JVM barrier incur significant overhead. As such, we step down to the lower level RDD abstraction, which allows for mapping across partitions, reducing this overhead from $O(NM)$ to $O(M)$, where N and M are respectively the number of rows per partition and the number of partitions.

In addition to reducing

We plan to first create a hash table for all columns in the raw csv for easy access. Then we plan to "map" our data set into a collection of (key, value) pairs and reduce over all collections to extract certain features from the now distributed data frame. When iterating through specific columns we plan to utilize the hash table we made earlier in the map function.

3.3 Experiments

After the implementation of PySpark API in base featurization we intend to perform experiments on preprocessing time. Specifically, we intend to document the preprocessing time of using the PySpark port of base featurization and checking if it scales. For the experiment we intend to use a csv that initally is 1 gb in size then increasing the size of the csv by times 2 Next, we would mesasure the preprocessing time it takes for each csv and check for scalability/parallelism.

4 RESULTS

5 CONCLUSION

REFERENCES

6 APPENDIX

6.1 proposal-statement

Over the course of the past few years, the release and general availability of industrial scale AutoML systems have allowed for simplified "black box" creation of Machine Learning models. Although the field of feature extraction – converting facets of data into a form suitable for model input – is well represented in the literature, the important step of feature type inference has only recently seen rigorous attention in this context. Previous study into this topic shows that the state of the art in open AutoML systems perform poorly on this task, and causal testing on a Model Prep Zoo shows that this significantly penalizes model accuracy. For our replication project, we reproduced select experimental findings from the ADA lab at the University of California San Diego. We benchmarked AutoML feature type inference using trained ML models and existing open source libraries and tools by evaluating the effect on downstream inference accuracy. We also reproduced runtime performance results of several of the models. Our results in this reproduction largely align with that of the original papers. In our experiments, we observed that the Random Forest model outperformed industrial AutoML tools and all other classical ML models when comparing type inference accuracy. However, the original implementation used in the paper lacks scalability. For our project, we propose investigating the scalability of various type inference techniques. Specifically, we intend to investigate the preprocessing time and accuracy trade-off in such systems. We also intend to investigate probabilistic methods that do not require full-dataset iteration: the current limit on scalability lies in featurization, not in model evaluation. To ensure that our experiments take place in a realistic industrial environment, we will port the base featurization routines used by these models to the PySpark runtime. As a result, we will be able to port the entire model to the PySpark MLlib APIs, which in turn will allow for running the entire process at cluster-level scale. Based on private correspondence, this specific undertaking has industrial interest. Thus, our project has two primary and one secondary output objectives. First, we intend to produce a novel scalable implementation of state-of-the-art feature type inference, and release this as an open source library artifact. Second, we intend to produce a technical report explaining our implementation methods and techniques, as well as experimental evidence to back up our result. Finally, as the task of setting up a spark cluster is nontrivial, we intend to release an open source tool for doing so on commodity-grade server platforms as a secondary artifact based on one author's previous work.