Improving Downstream Model Performance of AutoGluon With SortingHat

Jin Dai
University of California, San Diego
jidai@ucsd.edu

Xin Yu
University of California, San Diego
xiy264@ucsd.edu

ABSTRACT

The emergence of AutoML is democratizing ML for the mases. While many AutoML platforms put their focus on automating feature extraction, model selection, etc., little has studied the gateway step of feature type inference. The semantic gap between the source data's attribute types stored in databases and other file systems and the actual feature types consumable by the ML pipeline necessitates this study. Besides, modern AutoML tools like AutoGluon by Amazon still heavily rely on rule-based heuristics to automate feature type inference, which may lead to wrong feature types being used and thus hurting the downstream model performance. In this work, we aim to boost the downstream model performance of AutoGluon by integrating a ML-based feature type inference engine that can generate more accurate feature types over tabular data. We integrate SortingHat with AutoGluon-Tabular version 0.3.1 codebase, and leverage a downstream benchmark suite of 15 classification tasks and 5 regression tasks provided in the SortingHat project to assess the impact of feature type inference on downstream models. Preliminary analysis shows that AutoGluon powered by SortingHat feature type inference generates models that perform closer to the ground truth based on the current implementation. We release our integration code and the benchmarking pipeline as public repositories through Github.

1 INTRODUCTION

As machine learning (ML) techniques are growing into sophistication to solve increasingly complex problems, the emergence of Automated Machine Learning (AutoML) is helping democratize machine learning for the masses [7]. Industry AutoML platforms and tools aim to simplify the work for the experts and lower the barrier for the novices by automating the entire end-to-end ML flow. Such platforms and libraries put their focus mainly on data transformation, feature engineering, model selection and hyperparameter tuning. However, the step of feature type inference – the entry point to the majority of ML workflows over structured data – has received far less attention from the AutoML community [9].

The study of automated feature type inference is crucial in applying ML on large-scale tabular datasets. Typically, source data reside in databases or data lakes until queried where the attribute types (e.g. numbers, strings) do not embed any semantic meanings. Whereas in a ML workflow, these attribute types have to be converted to semantic types (e.g. numeric, categorical) that best describe the features for downstream model training. Although one straightforward way is to hand-label each feature types beforehand, it may require tedious, slow human efforts and is prone to human error[9]. Therefore, effective feature type inference becomes necessary to bridge this semantic gap in between for any end-to-end AutoML system.

Popular AutoML tools including AutoGluon from Amazon Web Services (AWS)[6] still heavily rely on rule-based heuristics to infer unknown feature types. However, this can result in less satisfying prediction accuracy and lead to wrong feature types being passed onto subsequent ETL jobs and training pipelines. Previous studies by the ADA [9][10] have shown that inadequate feature types can propagate to the downstream training pipeline and potentially hurt the output model's performance due to lack of useful information or inclusion of redundant features with no prediction value.

What is AutoGluon? - AutoGluon is an AutoML library developed by AWS that enables effective image classification, object detection, text classification, and tabular data prediction with little to no prior experience in machine learning [8]. It automates the end-to-end ML work flow from automatically preparing the dataset, trying different machine learning approaches to generating the best performing models with fine-tuned hyperparameters through only a few lines of code. Under the hood, AutoGluon supports multiple widely-adopted ML techniques including XGBoost, LightGBM, Neural Networks, Random Forest, and etc. AutoGluon can also combine the results of individual models to deliver high-quality predictions through model bagging and ensembling by default [8].

What is SortingHat? - SortingHat is a ML-driven solution to effectively solve the feature type inference problem. It can extract meaningful column-wise attributes of a tabular dataset such as the number of distinct values, the standard deviation of the values, whether the column values are strings, whether the values contain delimiters, random value samples, and etc. It subsequently feed those attributes as features to a trained random forest classifier that labels each column with the right feature type. SortingHat currently supports 9 feature types, including Numeric, Categorical, Datetime, Sentence, URL, Embedded Numbers, List, Not-Generalizable and Context-Specific. We refer the reader to the technical report of SortingHat for more detailed explanations [9].

Objective - The objective of our work is to boost the downstream model performance by bringing ML-based feature type inference to AutoML.

Scope - Since our focus is on relational/tabular data, the scope of our work mainly focus on the integration of SortingHat with

UCSD CSE234 '21, December 09, 2021, La Jolla, CA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

AutoGluon's Tabular Predictor. We want to understand the impact of replacing AutoGluon's rule-based feature type inference with SortingHat's ML-driven approach on the downstream model training by setting up a downstream benchmarking pipeline. For the current phase, we mainly analyze 3 downstream models that AutoGluon-Tabular supports: Random Forest, XGBoost and Nural Net with FastAI backend. We also disable AutoGluon's model bagging and ensembling features to achieve straightforward comparison between the prediction accuracy of individual models.

2 SORTINGHAT INTEGRATION WITH AUTOGLUON

In this section, we discuss the implementation-level details of the SortingHat-AutoGluon integration. We first explain our efforts on preparing SortingHat as a working Python package for installation and import. Secondly, we examine the gap between SortingHat's feature type vocabulary and AutoGluon's and present how we map feature types inferred by SortingHat to AutoGluon supported types. Finally, we illustrate the high level logic flow of AutoGluon's data pre-processing pipeline and discuss the best way to plug the SortingHat library into AutoGluon's codebase. In this work, we used AutoGluon version 0.3.1, which was the latest stable version when this report was written.

2.1 Preparing SortingHat as a Package

SortingHat was released as a public library and open-sourced through Github [2]. However, it was no longer under active maintenance at the beginning of our project and hence could not be directly installed per the README instructions. We forked the original repository and applied patches on the top, including the removal of unused dependencies and fixes for model binary loading/packaging and NLTK resource downloading [3]. With these patches, we could install and import SortingHat as a standalone python package from the forked repo.

2.2 Feature Type Vocabulary Mapping

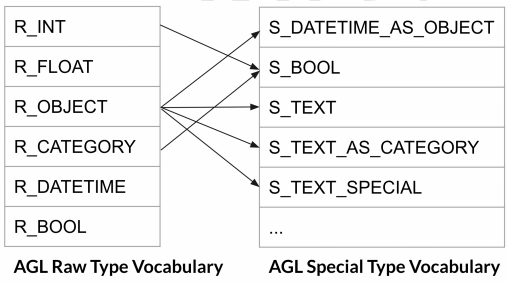

Figure 1: AutoGluon's raw and special feature type vocabularies

The design of the feature type vocabulary mapping is crucial in SortingHat-AutoGluon integration because there is no clear one-to-one relationship between SortingHat inferred feature types and AutoGluon supported feature types. More interestingly, AutoGluon proides two sets of feature type vocabularies: a raw type vocabulary and a special type vocabulary. The raw type vocabulary is a set of value types of corresponding tabular columns in the training data. Such raw types do not embed much semantic meaning but are still important for the special type inference through AutoGluon's subsequent data processing pipeline. The special type vocabulary provides better descriptions for features that are semantically useful and plays a decisive role in how the features are transformed by the heuristic rules. Figure 1 illustrates the two type vocabularies and the arrows indicate what special types can be derived by AutoGluon's data processing rules from each raw type. Below is a detailed description of the special types that are useful for tabular prediction [4].

S_DATETIME_AS_OBJECT: feature is a datetime in object form (string dates), which can be later converted to datetime object via `pd.to_datetime()`

S_BOOL: feature has been converted to int8 raw dtype with only 0 and 1 values that is semantically equivalent to R_BOOL

S_TEXT: feature is an object type that contains text information that can be utilized in natural language processing

S_TEXT_AS_CATEGORY: feature is a categorical that was originally text information

S_TEXT_SPECIAL: feature is a generated feature based off of a text feature but is not an ngram. Examples include character count, word count, symbol count, etc

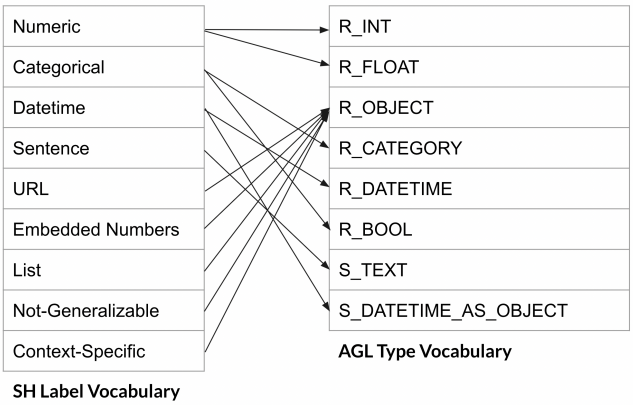

Figure 2: Mapping that illustrates the conversion from SortingHat inferred feature types to AutoGluon supported feature types

Mapping from SortingHat types to AutoGluon raw types is straightforward. This can be effectively done by cross-referring the raw column types of the training data (e.g. mapping from Numeric to R_INT or R_FLOAT). However, after careful study of AutoGluon's code, we realize that AutoGluon's special type inference heuristics are tightly coupled with its data processing and transformation logic. In addition, the inference of special types has a dependency on the columns' raw types. Such design makes it difficult to modulize and replace the feature inference rules as a whole. As a middle ground, we decide to map SortingHat types to AutoGluon raw and special types as much as possible. W would still need to partially rely on AutoGluon's data processing rules to complete the inference of the rest of the feature types. Figure 2 shows how we decide to

map the SortingHat inferred types to AutoGluon supported raw and special types. We present AutoGluon's data pre-processing flow and justify our decision in the next section.

2.3 AutoGluon Data Pre-Processing Flow

One of AutoGluon Tabular Predictor's most important user interface is the `TabularPredictor.fit()` method. The official documentation [1] lists a detailed description of all method variables it requires and supports. The `fit()` method accepts an optional `feature_metadata` parameter that embeds the feature type information and is used in various inner logic in feature pre-processing. If `feature_metadata` is not explicitly provided, AutoGluon will use a set of rule-based heuristics to automatically infer the feature types based on the properties of the given training data and construct a `FeatureMetadata` object for subsequent use. Figure 3 illustrates a rough overview of how the data pre-processing pipeline of AutoGluon-Tabular works internally.

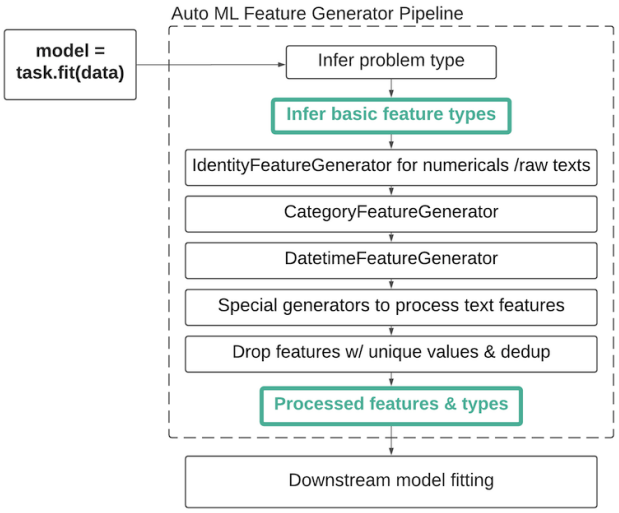

Figure 3: AutoGluon's Data Pre-Processing Logic Flow

After a user runs the `fit()` method, AutoGluon-Tabular first infers what type of prediction problem this is (binary, multi-class classification or regression) base on the values present in the label column. The following pipeline consists of three main stages. AutoGluon firstly infers the basic feature types using heuristics. It then passes the training data along with the inferred basic types onto a chain of rule-based feature generators to handle more complex features like Categorical features, Datetime features and Text features. Finally AutoGluon outputs the transformed features with the final feature types that are consumed by the downstream model fitting. Note that AutoGluon does not provide an explicit Uncategorized feature type to mark fields with presumably little predictive value (e.g. UserIDs) that will be dropped from the data. Instead, it checks whether a column consists of non-numeric, non-repeating values and drop the marked ones as part of `DropUniqueFeatureGenerator`.

As the feature type inference and data transformation logic are tightly coupled in the feature generator chain, we decide that the point in `TabularPredictor.fit()` where basic feature type inference happens could be the best place to plug in the SortingHat library and the aforementioned mapping logic, while still being able to benefit from the subsequent unified data processing and transformation flow.

Also for the ease of downstream benchmarking pipeline setup, we modified the `TabularPredictor.fit()` method's signature to include an additional `use_metadata_engine` parameter. Whenever we invoke the `fit()` method with `use_metadata_engine=True`, AutoGluon will invoke our SortingHat metadata engine for the preliminary feature type inference and then continue with the subsequent data processing and downstream model fitting.

3 EMPIRICAL STUDY

3.1 Datasets

We use 20 datasets from the SortingHat downstream benchmark suite that hasat least one integer column. They contains feature types including Numeric (NU), Categorical (CA), Datetime (DT), Sentence (ST), Not-Generalizable (NG), Embedded Number (EN), URL, List (LST), and Context-Specific (CS). In total, there are 19 classification tasks and 4 regression tasks. The datasets and their source details are available on our Github repo.

3.2 Downstream Benchmarking Pipeline

First, from the metadata file, we load the name of the target column we want to predict and the TruthVector to compute the ground truth in each dataset for further comparison. Then we classify the datasets based on their features types so that we can run the datasets with the same feature types in a batch to reduce human effort. In the pipeline, we build three routes for the ground truth, AGL (Autogluon), and AGL+SH (Autogulon powered by SortingHat) to see their performance separately. In terms of downstream model evaluation, the three downstream models we selected are RandomForest, XGBoost and NN-FastAI. Thus, we have 60 downstream models in total. We use the accuracy metric for the classification tasks and root mean squared error (RMSE) metric for the regression tasks following the same approach as [10].

For the parameters in Autogluon Tabluar Predictor, we set *eval_metric = 'accuracy'/'root_mean_squared_error'* corresponding to the problem type, and *presets= 'best_quality'*, and *num_stack_levels=0.* The first one is the metric we want to use to evaluate the performance of the downstream models. We pick accuracy as the metric for the classification tasks and root mean square error for the regression tasks. The parameter *presets= 'best_quality'* is to make Autogluon to achieve the best quality of the model. The trade-off of the parameter is the runtime of each model may increase. Since Autogluon has a automated feature which creates a weight ensemble model, the parameter *num_stack_levels=0* disable this feature so that we can make model-wise comparison.

3.3 Results

3.3.1 Classification. Table 1,2 and 3 present a comprehensive comparison between the performance of AutoGluon downstream models using feature types inferred from the ground truth, AutoGluon's default heuristics and the integrated SortingHat library. We also plot the all the results of our classification tasks into column charts

Table 1: Downstream Model Performance On Classification Tasks Using Random Forest

Feature Types	Raw Attibute Types	Dataset	Size	Truth	AGL	AGL + SH
NU	Int, Float	Cancer	116	0.7917	+0	+0
	Int	Mfeat	2000	0.9675	+0	+0
CA	String	Nursery	12960	0.9981	+0	+0
	Int	Hayes	132	0.7778	-0.0371	+0
	Int	Supreme	4052	0.984	+0.0024	+0.0024
	Int, String	Flares	323	0.9077	+0	+0
	Int, String	Kropt	28056	0.8466	+0.001	+0.001
	Int, String	Boxing	132	0.6296	-0.037	-0.037
CA + NG	Int, String	Apnea2	475	0.9368	+0	+0
NU+CA	Int, String	Flags	194	0.6667	+0.0256	+0.0256
	Int, Float, String	Diggle	310	1	+0	+0
	Int, Float	Hearts	304	0.7869	-0.0164	+0
	Int, Float	Sleuth	87	0.7222	+0	+0
NU + CA + ST	Int, String	Auto-MPG	392	0.8228	+0	-0.0008
NU + CA + LST + NG + CS	Int, Float, String, PK	Pokemon	801	0.9193	-0.0125	+0

Table 2: Downstream Model Performance On Classification Tasks Using XGBoost

Feature Types	Raw Attibute Types	Dataset	Size	Truth	AGL	AGL + SH
NU	Int, Float	Cancer	116	0.7083	+0	+0
	Int	Mfeat	2000	0.96	+0	+0
CA	String	Nursery	12960	0.9996	+0	+0
	Int	Hayes	132	0.7407	+0.0371	+0
	Int	Supreme	4052	0.9901	+0	-0.0012
	Int, String	Flares	323	0.9231	+0	+0
	Int, String	Kropt	28056	0.8815	+0.0102	+0
	Int, String	Boxing	132	0.6667	+0.1852	+0.1852
CA + NG	Int, String	Apnea2	475	0.9263	+0.0105	+0
NU+CA	Int, String	Flags	194	0.6923	-0.0513	-0.0513
	Int, Float, String	Diggle	310	1	+0	+0
	Int, Float	Hearts	304	0.8033	+0.0328	+0
	Int, Float	Sleuth	87	0.7778	+0	+0
NU + CA + ST	Int, String	Auto-MPG	392	0.7595	+0.0253	+0
NU + CA + LST + NG + CS	Int, Float, String, PK	Pokemon	801	0.9503	+0.0124	-0.0062

as in Figure 4,5,6 in the appendix for easier reference.

In Figure 4 where random forest is used as our downstream model, we observe that AGL + SH has an overall better performance than AGL alone. In all the datasets, AGL is underperforming the ground truth in 4 datasets: Hayes, Boxing, Hearts and Pokemon. But AGL + SH is underperfroming the truth only in one dataset, Boxing. In this case, the addition of SortingHat improves the performance of AutoGluon downstream models.

In Figure 5, using XGBoost as the downstream model, we observe some overfitting issues where AGL or AGL + SH achieve higher accuracy than the ground Truth. We think this issue is mainly by the limitation of XGBoost's support on categorical data. Although both AGL and AGL + SH have overfitting issues, the performance of AGL + SH is still closer to the ground truth, which means it performed better than using AGL alone. In this case, we think sometimes SH can alleviate the overfitting problems when using AGL.

In Figure 6, when using NN-FastAI as our downstream model, observe similar issues as in Figure 1 and 2, underperformance and overfitting still exist. AGL is underperforming the truth in 4 datasets: Hayes, Supreme, Kropt and Hearts, while AGL + SH is underperforming the truth in only 2 sets: Hayes and Pokemon. When comes to overfitting , AGL + SH still achieve a score closer to the ground truth. In conclusion, we think AGL + SH has a overall better performance in the classification tasks we performed in the 16 datasets we picked.

3.3.2 Regression. We also performed regression tasks using a separate set of 5 benchmarking datasets. From Table 4, 5 and 6, we can see that AGL + SH still has a overall better performance than AGL. In our data, in most of the time, AGL + SH is one performed closer to the ground, except using NN-FastAI on the Accident dataset.

Table 3: Downstream Model Performance On Classification Tasks Using NN-FastAI

Feature Types	Raw Attibute Types	Dataset	Size	Truth	AGL	AGL + SH
NU	Int, Float	Cancer	116	0.7083	+0	+0
	Int	Mfeat	2000	0.9725	+0.0025	+0.005
CA	String	Nursery	12960	0.9996	+0	+0
	Int	Hayes	132	0.8148	-0.1481	-0.1111
	Int	Supreme	4052	0.9901	-0.0074	+0
	Int, String	Flares	323	0.8769	+0	+0.0308
	Int, String	Kropt	28056	0.8792	+0.8113	-0.005
	Int, String	Boxing	132	0.6667	+0.7407	-0.037
CA + NG	Int, String	Apnea2	475	0.8842	+0	+0
NU+CA	Int, String	Flags	194	0.5897	+0.6154	+0
	Int, Float, String	Diggle	310	0.9677	+0	+0
	Int, Float	Hearts	304	0.7869	-0.7705	+0.0328
	Int, Float	Sleuth	87	0.6111	+0.7778	+0.1111
NU + CA + ST	Int, String	Auto-MPG	392	0.7215	+0.8101	+0.0127
NU + CA + LST + NG + CS	Int, Float, String, PK	Pokemon	801	0.913	+0.9689	-0.0124

Table 4: Downstream Model Performance On Regression Tasks Using Random Forest

Feature Types	Raw Attibute Types	Dataset	Size	Truth	AGL	AGL + SH
CA	Int	MBA	61	0.2561	+0.0292	+0.0292
NU+CA	Int	Vineyard	468	2.2587	-0.001	-0
	Int, String	Apnea	450	815.9176	-0.0443	-0
NU+CA+EN+NG	Int, String	Car Fuel	388	14.0802	-0.0956	+0.0073
DT	Int, String	Accidents	72	839.6767	-0	-0

Table 5: Downstream Model Performance On Regression Tasks Using XGBoost

Feature Types	Raw Attibute Types	Dataset	Size	Truth	AGL	AGL + SH
CA	Int	MBA	61	0.2516	-0.0156	-0.0156
NU+CA	Int	Vineyard	468	2.0331	+0.1992	+0.1018
	Int, String	Apnea	450	856.5938	-309.4784	-0
NU+CA+EN+NG	Int, String	Car Fuel	388	13.7149	-0.8832	-0.523
DT	Int, String	Accidents	72	1081.9608	-0	-0

Table 6: Downstream Model Performance On Regression Tasks Using NN-FastAI

Feature Types	Raw Attibute Types	Dataset	Size	Truth	AGL	AGL + SH
CA	Int	MBA	61	0.2403	-0.0179	-0.006
NU+CA	Int	Vineyard	468	4.6884	-1.5325	+0.273
	Int, String	Apnea	450	2721.346	-109.816	-29.8213
NU+CA+EN+NG	Int, String	Car Fuel	388	12.6666	-0.2459	-0.3484
DT	Int, String	Accidents	72	1201.8549	+33.3102	+2.3629

However, because we have very limited data for the regression tasks, we may need further study and more data to support our thought in the regression part.

4 PUBLIC RELEASE

We have published our modified SortingHat and AutoGluon as separate repositories on Github. We have also released a public repository on GitHub with our downstream benchmark pipeline [5]. The README file in the pipeline repository contains links to

our modified SortingHat and AutoGluon repos, along with links to the original repos. The README also provides guidance on how to install the dependencies and how to perform a sample run of our pipeline.

5 TAKEAWAYS

(1) From all of the data and performance analysis in our work, we observe that overall AutoGluon with SortingHat generates downstream models that perform closer to the ground truth in both the classification and regression tasks. Specifically, AutoGluon can benefit from ML-based feature type inference by SortingHat in the classification tasks.
(2) AutoGluon may have overfitting issues as we observed in the classification tasks using XGBoost. In this case, adding SortingHat to replace AutoGluon's rule-based feature type inference may alleviate this issue because AutoGluon with SortingHat performs closer to the ground truth even in the overfitting situation.
(3) On the other hand, the feature inference and data processing are tightly coupled in AutoGluon's data processing and transformation pipeline. We believe writing SortingHat specific data processing logic may yield better empirical results for continued study.
(4) Lastly, we think there are some limitation in the datasets in out study. Some datasets has only hundreds of records, which may also lead to overfitting in the downstream models for unseen data. We believe our study could benefit from larger datasets or data augmentation.

REFERENCES

[1] [n. d.]. *API Documentation Of AutoGluon Tabular Predictor's fit() Method.* Retrieved December 09, 2021 from https://auto.gluon.ai/stable/api/autogluon.task.html#autogluon.tabular.TabularPredictor.fit
[2] 2020. Github Repository for ML Feature Type Inference. https://github.com/pvn25/ML-Data-Prep-Zoo
[3] 2020. Patches For preparing SortingHat as a Working Python Package. shorturl.at/cwAM0
[4] 2021. AutoGluon's Types Class. https://github.com/fabulousdj/autogluon/blob/master/core/src/autogluon/core/features/types.py
[5] 2021. Github Repository for AutoGluon-Tabular Downstream Benchmarking Pipleine. https://github.com/yxx0726/AGL_Tabular_ML_Pipline
[6] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505* (2020).
[7] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. Automated Machine Learning - Methods, Systems, Challenges. 219 (2019).
[8] Shashank Prasanna. 2020. *Machine learning with AutoGluon, an open source AutoML library.* https://aws.amazon.com/cn/blogs/opensource/machine-learning-with-autogluon-an-open-source-automl-library/
[9] Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar. 2021. Towards Benchmarking Feature Type Inference for AutoML Platforms. In *Proceedings of the 2021 International Conference on Management of Data.* 1584–1596.
[10] Vraj Shah, Kevin Yang, and Arun Kumar. 2020. *Improving Feature Type Inference Accuracy of TFDV with SortingHat.* Technical Report.

A APPENDIX

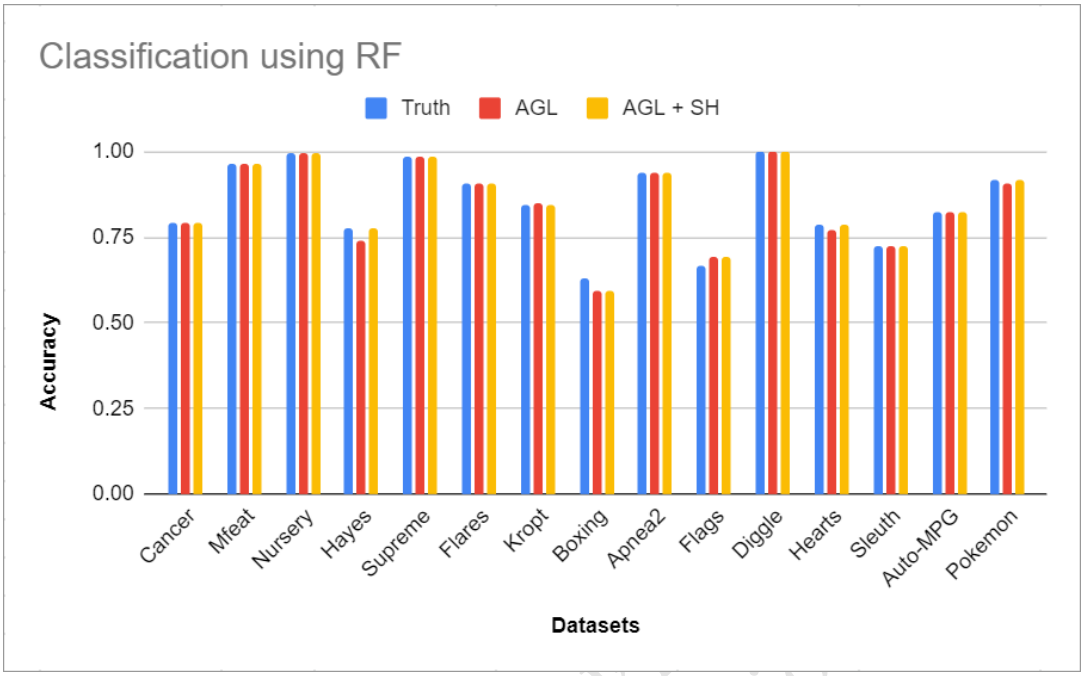

Figure 4: Performance of classification tasks using Random forest

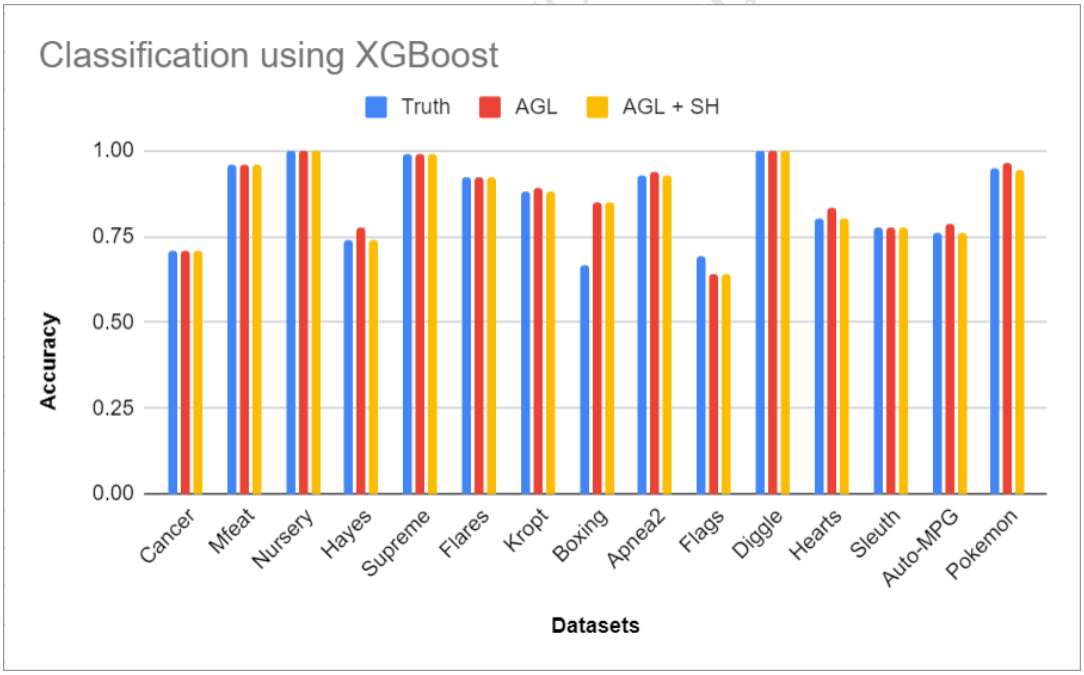

Figure 5: Performance of classification tasks using XGBoost

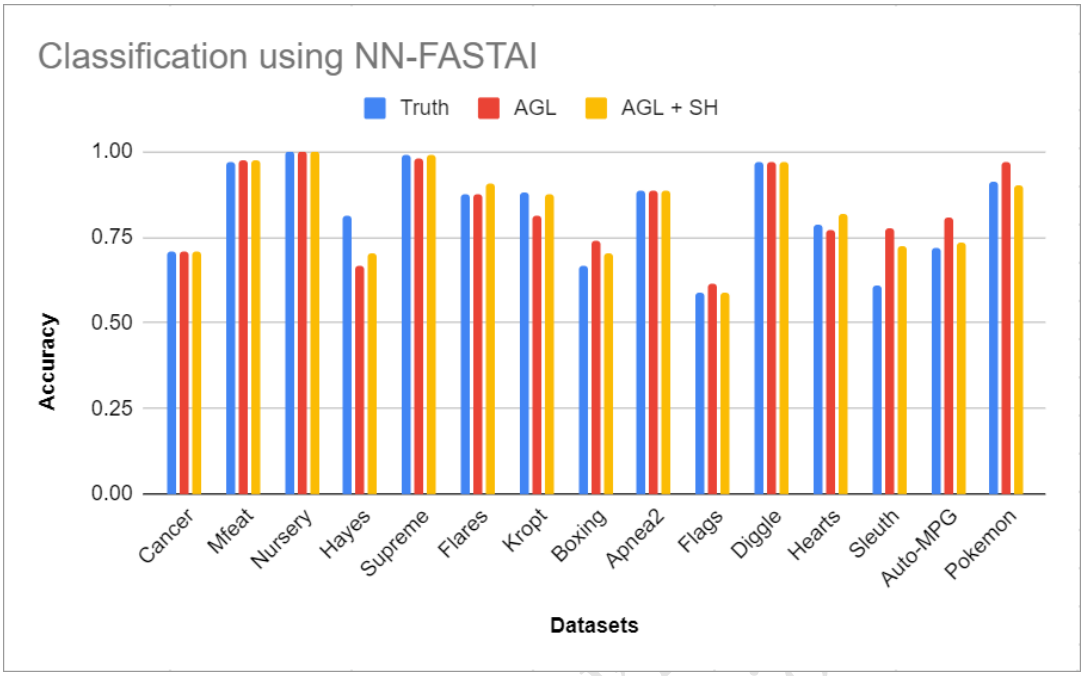

Figure 6: Performance of classification tasks using NN-FastAI