Bringing ML-based Feature Type Inference to OpenML

Ryan Tran
rhtran@ucsd.edu
UCSD

Victor Zhu
vzhu@ucsd.edu
UCSD

ABSTRACT

OpenML is a democratized ML platform for crowdsourcing ML datasets and models. Users uploading tabular data as Pandas dataframes can make use of the platform's automatic feature type inference. This can be especially useful for datasets with a large number of columns for which it is too time-consuming to annotate by hand. But as previous work under Project SortingHat has shown, inferring the feature type of columns correctly can have a huge impact on downstream model performance. We show that using an ML-based approach to feature type inference, as detailed in previous Project SortingHat work, yields improvements in classification accuracy of categorical and numeric columns, as well as expands the feature type vocabulary when compared to OpenML's current approach. As OpenML currently only supports using a limited set of feature types, we created mappings from the original 9 SortingHat features to the OpenML types and we release this library as a Python package on PyPi to facilitate further use.

1 BACKGROUND / PRELIMINARIES

1.1 OpenML

OpenML is an open-source ML platform for sharing data, tasks, workflows, models, and results [5]. Users can upload and share their datasets and models to the platform. One feature of OpenML is their support for automatic data type inference for Pandas Dataframes. This is useful when there are many columns (like in a large crowd-sourced ML platform) and removes a lot of the manual labeling burden. A lot of data out in the wild is in relational databases which may or may not be properly labeled and so having this step of data preparation automated is a huge time-saver. However, though this is very useful, it is not very powerful and prone to misclassifications for more complicated types. For example, it can misclassify zip codes as `integers` instead of `categorical`, and timestamps as `integers` instead of datetime.

Another aspect of OpenML is the deduplication of data cleaning efforts. Users can restructure data for specific tasks and upload it to OpenML so that other users do not have to and can refocus their efforts on other tasks. Automatic feature type inference here is helpful as well, as once inferred and verified by humans, other users can utilize the datasets without worry and train better performing models which can then be cross-validated with models trained by others.

1.2 SortingHat

SortingHat is a library for ML-based feature type inference [4]. It uses the column name, sample values, and descriptive statistics to predict the feature type for each column. Properly labeled feature types have been shown to improve downstream model performance [4] which is an excellent reason for OpenML to utilize SortingHat.

1.2.1 Base Featurization. Base featurization is performed on the given dataset. First, a bigram feature set is computed on the attribute names and some sample values. The inspiration for this comes from humans. We naturally look at the attribute name along with some number of sample values to manually label attributes. The next step is calculating a number of descriptive statistics. 25 total descriptive stats are calculated for each column, including the total number of values, the number of unique values, the percentage of NaNs, the mean, and the standard deviation.

1.2.2 Feature Type Prediction. The featurized dataset is then fed into a random forest model which outputs the predicted feature type for each column.

During their research, Shah et al. 2021 [4] tested a number of model architectures including classical ML models, kNearest Neighbor, and convolutional neural nets. Overall, the random forest classical ML model performed the best on the key metric prediction accuracy.

2 ADAPTING FOR OPENML

2.1 SortingHat Feature Type Taxonomy

9 Original SortingHat Feature Types.

- Numeric Represents attributes that are quantitative in nature and can directly be utilized as Numeric for downstream ML models. e.g. 7, 10.5, 42.0
- Categorical Represents attributes that are qualitative in nature that can directly be utilized as Categorical for downstream ML models. e.g. Cat, Male, 92093 (Zip Code)
- Datetime Represents attributes containing date or timestamp values. e.g. 2021-12-02, 20220222
- Sentence Represents attributes containing textual values with semantic meaning. e.g. "So when in tears, the love of years, is wasted like the snow"
- URL Represents attributes whose values follow the URL standard. e.g. www.openml.org
- Embedded Number Represents attributes with "messy" syntax that require some preprocessing before their direct use as Numeric or Categorical. e.g. 30 Mhz, USD 45
- List Represents attributes that contain a list of items separated by a delimiter. e.g. 0 \t 1 \t 2 , car;plane;bicyle
- Not-generalizable Represents attributes that has no or almost no informative values as a feature for downstream models. e.g. Primary key in table, only one unique value in the column
- Context-specific Represents attributes that require human intervention either for feature type determining or custom featurization routines.

2.2 OpenML Attribute Type Taxonomy

OpenML's attribute types are loosely based on the Attribute-Relation File Format (ARFF) [1].

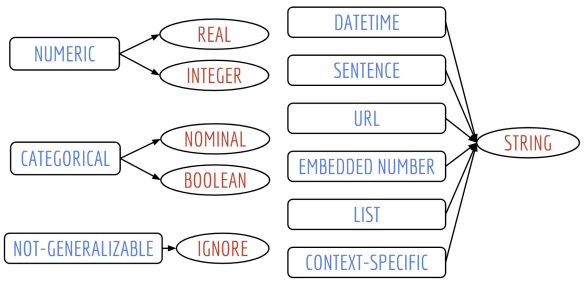
- Real Represents attributes that contain real values. e.g. 123.45, 777.77
- Integer Represents attributes that contain integer values. e.g. 0, 7, 42
- Boolean Represents attributes that contain boolean values. e.g. 0, 1, true, false
- Nominal Represents attributes that contain categorical values. e,g, animal, zip code
  - Stored in (COL_NAME, [POSSIBLE_TYPES]) format.
- String Represents attributes that contain arbitrary textual values. e.g. "hello there", "general kenobi"
- Ignore Represents attributes that can be safely ignored by OpenML.

2.3 Discussions with OpenML

We discussed with OpenML on how they wanted to utilize SortingHat and the main use cases. The first major use case would be in the user uploading dataset story. A user wants to upload their dataset with lots of columns to OpenML, but does not want to manually label each feature. Here, SortingHat can be utilized to label each feature automatically for the user. Since they already use Pandas inference for this, they requested a drop-in replacement here so that they don't have to modify much code. Another is how SortingHat is integrated with OpenML. They decided they wanted the SortingHat mappings to OpenML to be its own library that they can simply include as an optional dependency for users. As such, we created a PyPi package which exposed the required functionality for them that they can simply include and use.

2.4 Type Mapping

Figure 1: SortingHat to OpenML types



In Figure 1, you can see the mappings from SortingHat to OpenML types. Numeric is mapped to Real and Integer types. Categorical is mapped to Nominal and Boolean types. Datetime, Sentence, URL, Embedded Number, List, and Context-specific are all mapped to String. Even though Datetime is a supported type in ARFF, OpenML itself does not support it and so we map Datetime to the next best thing, String. Not-generalizable are mapped to Ignore as they are not useful in downstream model tasks.

2.5 Technical Hurdles

2.5.1 SortingHat Dependencies. One issue we faced in the beginning was that the dependency list in `SortingHatLib` was not directly usable. It held many conflicting dependencies along with a large number of irrelevant ones. We spent a large portion of time culling not only unnecessary dependencies, but also unnecessary code to simplify our workflow.

2.5.2 Model File Size. The original pickled random forest model was 114 megabytes large and so would not fit in a standard PyPi repository. To reduce the file size, we compressed it using bzip2 [2] before packaging. This reduced the filesize to 7.4 megabytes, only 6.5% of the original 114 mb. When a user downloads and uses our library, we simply first decompress the pickle model before loading it.

2.5.3 Downloading the Benchmark Dataset. Shah et al. 2021 [4] make their feature type benchmark dataset public available [1]. This includes a link to download the raw CSV files. Unfortunately, at the time of writing, this link is incomplete. Readers wishing to reproduce our study may need to download one of the CSV files separately [2] and add it to their data folder. Note that we obtained this missing CSV file by examining the metadata information [3] for the benchmark dataset.

3 IMPLEMENTATION

Shah et al. 2021 [4] make their best-performing model publicly available on Github [4]. We ship this model with our package and use it to power our ML-based feature type inference. In later sections during comparison with OpenML's feature type inference, we will simply refer to this model as SortingHat. We borrow code from the official library repo to perform the featurization of the columns as described in [4]. The featurized columns are then passed to the loaded model, which outputs its predictions as one of the 9 SortingHat types.

3.1 Public API

We expose three public functions in our API: `get_sortinghat_types`, `get_expanded_feature_types`, and `get_feature_types_as_arff`. The first function simply returns feature type predictions in the original SortingHat vocabulary. The second two are functions specially requested by OpenML that perform mappings from the SortingHat vocabulary as alluded to in earlier sections.

3.1.1 Expanded Feature Types. In our discussions with OpenML, they described to us their use cases for delineating floating point numbers from integers. Similarly, they requested recognizing `boolean` columns separately from more general `categorical` columns. The function `get_expanded_feature_types` was designed with these requests in mind. To separate `numeric` columns into `REAL` and `INTEGER`, we use the function `pandas.api.types.is_integer_dtype`.

[1] https://github.com/pvn25/ML-Data-Prep-Zoo/tree/master/MLFeatureTypeInference/Benchmark-Labeled-Data
[2] https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe
[3] https://github.com/pvn25/ML-Data-Prep-Zoo/blob/master/MLFeatureTypeInference/Benchmark-Labeled-Data/Metadata/meta_data.csv
[4] https://github.com/pvn25/ML-Data-Prep-Zoo/tree/master/MLFeatureTypeInference/Pre-trained%20Models

If a column is marked as `numeric` by SortingHat, then it is mapped to `INTEGER` if the function returns true, otherwise it is mapped to `REAL`. Although we ourselves are now using a Pandas function to infer types, note that this problem setting is much easier. Because we already know a column to be of `numeric` type, we can apply this simple data type check to differentiate two types of `numeric` data. This is more reliable than using the simple data type check on data that could be of any feature type.

Similarly, we use `pd.api.types.is_bool_dtype` to separate `boolean` columns from other `categorical` ones. If SortingHat marks a column as `categorical`, we perform this check and return a column as `boolean` instead of `categorical` if the function returns true.

3.1.2 ARFF Types. As discussed in earlier sections, OpenML currently categorizes columns using ARFF attribute types. To minimize friction in the integration of SortingHat with OpenML, we design our function `get_feature_types_as_arff` to return the feature types in the format OpenML is already expecting. Specifically, their code expects a list of tuples, with the first element being the column name and the second being the predicted type. For categorical columns, however, they expected a tuple with the first element being the column name and the second being a list of the unique values that column contains.

To support these features we treat categorical columns slightly differently. If SortingHat marks a column as `categorical`, we cast it to the `category` Pandas data type so that we can extract the column's unique values. Note that we separate `boolean` columns from `categorical` as before. In the special case that a categorical column is detected as boolean, we return `True` and `False` as that column's unique values.

The rest of the SortingHat types are mapped as illustrated in Figure 1. As mentioned ealier, the ARFF format does support the `datetime` type but OpenML does not for the time being, so we map detected `datetime` columns to `STRING`. Additionally, `IGNORE` is not an ARFF type. When users create an `OpenMLDataset`, they can specify an optional parameter `ignore_attribute` which signals to OpenML that that column should not be used in modeling. This is intended to identify identifiers and indexes, so it is directly analogous to the `notgeneralizable` type that SortingHat recognizes.

Finally we also note that the function `get_feature_types_as_arff` also returns the SortingHat types for each column as its second return value. This is at the request of OpenML, who noted that they may in the future find uses for columns tagged with their SortingHat types.

3.2 Example Usage

Below we demonstrate usage of our package. Note that `sortinghatinf` can take any tabular dataset loaded as a Pandas dataframe as input, but we use datasets loaded from OpenML here as an example.

```
!pip install sortinghatinf
!pip install openml

import sortinghatinf
import openml
```

Table 1: Comparison of our attempt to reproduce the results of Shah et al. 2021 on the benchmark dataset against the results reported in their paper.

Feature Type	Metric	Reproduced	Shah et al. 2021
Numeric	Precision	0.921	0.934
	Recall	0.989	0.984
	Accuracy	0.966	**0.97**
Categorical	Precision	0.899	0.913
	Recall	0.873	0.943
	Accuracy	0.948	**0.966**
Datetime	Precision	0.938	0.945
	Recall	0.957	0.972
	Accuracy	0.992	**0.994**
Sentence	Precision	0.851	0.865
	Recall	0.870	0.902
	Accuracy	0.987	**0.989**
Not-Generalizable	Precision	0.797	0.934
	Recall	0.842	0.86
	Accuracy	0.960	**0.978**
Context-Specific	Precision	0.867	0.859
	Recall	0.670	0.705
	Accuracy	0.960	**0.961**

```
# Load the Blood Transfusion Dataset
# from OpenML.
data = openml.datasets.get_dataset(
  dataset_id=31)
# Loads as Pandas dataframe
X, _, _, _ = data.get_data()

# Infer the SortingHat and ARFF
# feature types.
infer_arff, infer_sh = sortinghatinf
            .get_feature_types_as_arff(X)
print(infer_arff)
# [('checking_status', ['<0', '0<=X<200', ...]),
# ('duration', 'INTEGER'), ...]
print(infer_sh)
# ['categorical', 'numeric', ...]
```

3.3 Reproducing SortingHat Benchmarks

To check the correctness of our implementation, we evaluate our package on the SortingHat benchmark dataset. We report our results in Table 1 under the column *Reproduced*. As we can see, the performance is still quite good, but noticeably worse than the results reported by Shah et al. 2021 [4], especially on `categorical` and `non-generalizable` columns. Being that we are using the best model from [4] as provided by Shah et al. 2021, we do not have an explanation for this degradation in performance and have reached out to the authors to understand the issue.

4 EXPERIMENTAL EVALUATION

To confirm the motivations for our project, we compare the feature type inference of OpenML against SortingHat. We evaluate the performance of both approaches on a labeled benchmark dataset as well as examine their agreement/disagreement on unlabeled datasets.

4.1 Methodology

Currently, OpenML offers automatic type inference to users uploading their datasets as Pandas dataframes only. Pandas is an open-source software library written for Python that aides in data analysis and manipulation [3]. OpenML's type inference works by first calling `pandas.api.types.infer_dtype` on columns of the input dataframe. This function outputs a string corresponding to a data type. From this output, OpenML then maps `integer` to `INTEGER`, `floating` to `REAL`, `string` to `STRING`, and encodes both `boolean` and `categorical` as the list of unique values data in that column assumes.

To compute the performance of OpenML on the benchmark dataset, we need to map these OpenML classes back to SortingHat classes. The `INTEGER` and `REAL` classes are mapped to `numeric`. We combine the `boolean` and `categorical` labels into simply `categorical`. Finally, we ignore columns marked as `string` by OpenML since there is not a straightforward way to separate general strings into the remaining 7 SortingHat classes. As a result, we are only able to evaluate on the `numeric` and `categorical` columns of the benchmark test set, and we report results on only these two classes here.

4.2 Benchmark Dataset

The SortingHat benchmark dataset is a collection of 9921 columns from 1240 CSV files downloaded from Kaggle and the UCI ML Repository [4]. Each column has been labeled according to the 9-class feature type vocabulary as described in Section 3.1. A random 20% of the dataset has been set aside as a test set, and we evaluate on the subset of this test set corresponding to the classes `numeric` and `categorical`. This results in a test set of 707 `numeric` columns and 457 `categorical` columns.

Table 2 shows the results of evaluating both OpenML feature type inference and SortingHat on the benchmark dataset. We can see that on `numeric` columns, SortingHat handily outperforms OpenML. OpenML's precision and recall numbers suggest that their feature type inference over-predicts numeric columns. This aligns with our intuition that categorical feature types represented as numbers, such as zip code, will be mis-classified as `numeric` by a simple data type check, such as the function `pandas.api.types.infer_dtype`.

Moreover we can also see from Table 2 that OpenML's feature type inference has an extremely low recall score on categorical columns. The reason for this is `pandas.api.types.infer_dtype`. The columns of a Pandas dataframe have a field `dtype` for denoting their data type. Among Pandas' data types is the type `category`. `pandas.api.types.infer_dtype` will only classify a column as categorical if it has this `category` data type. However, when dataframes are loaded from file, columns with this data type are usually never created: it is not straightforward to determine which columns in a file are categorical after all! Therefore, it is expected that OpenML, which uses this Pandas function under the hood, will severely underpredict the `categorical` class.

Table 2: Comparison of OpenML feature type inference against SortingHat on the benchmark dataset.

Feature Type	Metric	OpenML	SortingHat
Numeric	Precision	0.615	0.921
	Recall	1.000	0.989
	Accuracy	0.777	0.966
Categorical	Precision	1.000	0.899
	Recall	0.007	0.873
	Accuracy	0.771	0.948

4.3 Unlabeled Datasets

OpenML hosts thousands of easily accessible tabular datasets uploaded by users on their website, providing us with the perfect opportunity to further compare OpenML type inference with SortingHat on datasets that are more "in-the-wild". We ran OpenML and SortingHat type inference on 50 datasets available on OpenML, and we report the results as two experiments.

4.3.1 OpenMLDataset object. For our first experiment we use the `openml-python` API to download datasets and load them as Pandas dataframes. This was achieved with the following API calls.

```
task = openml.tasks.get_task(id)
data = openml.datasets.get_dataset(task.dataset_id)
X, _, _, _ = data.get_data()
```

Following this, we pass the loaded Pandas dataframe to OpenML and SortingHat feature type inference and compare the results. The statistics are shown in Table 3. From these datasets, we predicted on a total of 1924 columns. Of these, 1788 saw the same predictions being made by OpenML and SortingHat. The majority of these agreements were `numeric` with 939 columns, followed by `categorical` with 853 columns. Interestingly, this means that for around 93% of the total columns, both SortingHat and OpenML agreed on the predicted types. One implication of this is that for the sample datasets we used, the vast majority of the columns were numeric and categorical. This is not too surprising, as Shah et al. 2021 [4] noted a similar finding: The original source files they used for constructing the benchmark dataset contained very few examples for the URL, List, Sentence, Embedded Number, and Datetime classes. To prevent a heavily skewed class label distribution, they augmented the under-represented classes in this dataset from other sources, although numeric and categorical still heavily made up the distribution in the end.

More peculiar however, is the high amount of agreement on categorical columns. From Table 2 we can see that OpenML's feature type inference has an extremely low recall score for categorical columns. Therefore, we expect OpenML to miss a lot more categorical columns and to have far less agreement with SortingHat on categorical columns. This highlights a difference between the SortingHat benchmark dataset and the sample datasets loaded from OpenML. Digging into this deeper, we found that the datasets returned by OpenML's API have a far higher proportion of columns

Table 3: Pandas vs. SortingHat Inference on OpenML datasets.

	Matching Output	Total	% Relative to Total
Numeric	939	-	48.80
Categorical	853	-	44.33
Total	1788	1924	92.93

Table 4: Pandas vs. SortingHat Inference on OpenML datasets.

	Matching Output	Total	% Relative to Total
Numeric	1118	-	57.45
Categorical	0	-	0.00
Total	1118	1946	57.45

with the `categorical` data type than those from the SortingHat benchmark dataset, making it much easier for the OpenML system to classify those columns as categorical. The reason for this difference was in how the dataframes were loaded: OpenML explicitly stores a list of which columns are categorical alongside the data in specialized file formats. Then when the dataset is requested, the column metadata is loaded along with the data and is used to cast the appropriate columns to `categorical` data type. The Pandas dataframes for the SortingHat benchmark dataset are by contrast loaded from CSV files, which do not store this additional column metadata. To confirm that these asymmetrical loading schemes were responsible for this anomaly, we conduct another experiment on the same 50 OpenML datasets.

4.3.2 Loaded as CSV. For this experiment, we visited the OpenML webpage for all 50 datasets we had used in the previous experiment and downloaded the corresponding CSV files. Instead of loading these datasets through the OpenML API, we read them into Pandas dataframes directly from these CSV files. We compute the same statistics as before and report them in Table 4. The difference in results between these two experiments is clear: OpenML no longer agrees with SortingHat on categorical columns at all, likely predicting close to nothing as `categorical`. This suggests that without the extra information from the column metadata hiding in the column data type field, OpenML fails to recognize categorical columns "in-the-wild". This is important because many tabular datasets in the real world are stored in CSVs, not in special file formats, and the feature type of their columns are completely unknown. Being able to robustly determine the feature type of a column in the absence of metadata is precisely the use case of SortingHat, and it is clear from these experiments that OpenML's type inference system was not designed for that.

The large drop in `categorical` column performance between Table 3 and Table 4 also implies that most of the columns from these OpenML datasets we sampled were not annotated automatically, but did in fact receive human help. Most likely, the majority of these datasets were uploaded with their feature types hand-labeled. As described at the beginning of this paper, SortingHat and this project were designed to help eliminate this exact type of manual labor. Looking at this another way, we might view the dataframes loaded from the OpenML API as being approximately ground truth for categorical columns. This makes the first experiment an unfair evaluation of OpenML but it also highlights the effectiveness of SortingHat: SortingHat achieves high similarity with OpenML, which is leveraging ground truth labels in some capacity.

Note that here the total number of columns is 1946 instead of 1924 as it was in the first experiment. This is simply due to the fact that in the first experiment, we only ran feature type inference on the feature columns for each dataset. Depending on each dataset's respective downstream task, some columns may have been designated as target columns for prediction. The CSV files, on the other hand, make no distinction between target and feature columns. For this second experiment we evaluate on all columns for simplicity. We expect that this does not have a significant impact on the results since these target columns make up a small proportion of the total columns.

We also observe an increase in the agreement on the `numeric` class moving from the first experiment to the second. We theorize that the absence of the column metadata denoting categorical columns in the second experiment has led the OpenML feature type inference to predict more columns as `numeric`, resulting in more opportunities for agreement with SortingHat.

Table 5: Example Column Predictions.
1: Cylinder Bands (33) 2: Heart Statlog (53) 3: Anneal (2)
#: Dataset Name (task_id)

Col Name	$Timestamp^1$	$FastBldSgr^2$	$ProductType^3$
	19910108	0	C
	19910109	1	C
	19910110	0	C
	19910111	0	C
	19910112	1	C
	...	...	...
OpenML	Categorical	Integer	Categorical
SortingHat	String (Date)	Boolean (Cat)	Ignore (Not-Gen)

4.4 Examples

Figure 5 shows a few manually verified examples from OpenML datasets where SortingHat accurately predicts the feature type that OpenML does not. The mismatches are heavily skewed towards the non-numeric and non-categorical types. We can see some common examples of misclassified types. Timestamp here is pulled from the Cylinder Bands dataset (task_id=33) and is miscategorized by OpenML as `categorical` when it should be `datetime`. Fasting Blood Sugar is pulled from the Heart Statlog dataset (53) and is miscategorized by OpenML as `integer` when it should be `boolean`. Product Type here is pulled from the Anneal dataset (2) and is miscategorized as `categorical` although it should be `not-generalizable`. Although Product Type would normally be `categorical`, in this case it is `not-generalizable` as the entire column only contains C's, leading to a non-informative column. These 3 examples show a variety of common failure scenarios that OpenML does not catch. SortingHat is powerful enough to catch many of these automatically.

5 FUTURE WORK

Currently, we are waiting on OpenML to integrate `sortinghatinf` into their pipelines and to provide any additional feedback. One modification that could be implemented in the future is a finer type mapping when OpenML decides to support more of the SortingHat types. Currently Datetime, Sentence, URL, Embedded Number, List, Context-specific all map to String. Datetime could be mapped to the ARFF type Datetime and Sentence mapped to some kind of Text type. This is essentially up to OpenML on how they want to proceed. We can also contribute `sortinghatinf` to Pandas as an alternative to their `pandas.api.types.infer_dtypes` function.

Future work could also explore extending `sortinghatinf` to accept other data structures like Numpy arrays. Currently, the base featurization code depends on Pandas dataframes, but this isn't necessary and could be implemented for other data structures. Let's say for instance that a user has many Numpy arrays with data. For them to currently utilize `sortinghatinf`, they would first have to convert the Numpy arrays to Pandas dataframes and then attach their corresponding attribute names before running the `sortinghatinf` functions. If they could directly pass in the Numpy arrays along with a file of their attribute names, this could save a lot of time and manual pre-processing for the user. Of course, a survey would need to be done to see what data structures the majority of users keep their data in before we add logic to accept them.

Another avenue for future work could be to explore improving feature-type inference by including information from the rest of the dataset. Currently, each column is evaluated individually by the model, but in the real world each column is almost definitely dependent on other columns within its dataset. Anecdotally, when we look at a column's name and sample values to determine its feature type, we usually already have a prior context of what the dataset contains from the dataset name and the feature types of other columns. Research should be done to determine the possible benefits that using this information could have, especially on the under-represented feature types.

6 CONCLUSION

We have packaged a model demonstrated to be effective for improving feature type inference "in the wild" and verified its implementation. We release this project publicly as a PyPi package[5] for ease of use, and we have demonstrated potential performance improvements to be gained from the integration of `sortinghatinf` into OpenML's codebase. For the majority of datasets already sitting on OpenML servers, Pandas inference works about as well as SortingHat due to the abundance of already labeled Categorical attributes. However, for future uploaded datasets, `sortinghatinf` can streamline the feature type inference pipeline and reduce the manual labeling workload. This is significant for the quantity of future datasets that is expected to be uploaded on the crowd-sourced OpenML platform over its lifetime, and this will ultimately translate into a performance boost for downstream models trained on these datasets.

[5] https://pypi.org/project/sortinghatinf/

7 REFERENCES

ACKNOWLEDGMENTS

Special thanks to Prof. Arun Kumar for his guidance and feedback throughout the project and to OpenML for our discussion on functionality and integration.

REFERENCES

[1] [n. d.]. Attribute-Relation File Format. https://waikato.github.io/weka-wiki/formats_and_processing/arff_stable/.
[2] [n. d.]. BZ2 Compression. https://docs.python.org/3/library/bz2.html/.
[3] Wes Mckinney. 2011. pandas: a Foundational Python Library for Data Analysis and Statistics. *Python High Performance Science Computer* (01 2011).
[4] 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* (Virtual Event, China) *(SIGMOD/PODS '21)*. Association for Computing Machinery, New York, NY, USA, 1584–1596. https://doi.org/10.1145/3448016.3457274
[5] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013), 49–60. https://doi.org/10.1145/2641190.2641198