PINEAPPLE CODE

A4 Report

Winter 2021 CSCD01 A4

Bryan Liu | Eric Li | Trey Robinson | Tahasun Tarannum | Cherise Bryan | Muizz Ahmed

Table of Contents

Issue Investigation

Issue # 14253

Link to scikit-learn: https://github.com/scikit-learn/scikit-learn/issues/14253

Type: New feature

Description: We want to allow the user to specify an estimator to be used for a specific column in IterativeImputer.

Estimate: 34 (based on scrum poker)

Preface: Datasets can contain missing values which are typically encoded as np.nan. An imputer can be used to infer the missing values with a given estimator.

Solution Proposal: Create an interface for IterativeImputer that allows users to specify an estimator they want to use to impute a specific column. We will do this by adding a parameter 'estimators' which will be a list of tuples of the form (name, estimator, columns), where 'name' allows the estimator and its parameters to be set using set_params(), 'estimator' is an estimator that supports fit() and transform(), and 'columns' is an integer which indexes the data columns or a string which specifies the column name. During the imputation process when we impute each column, we will reference the 'estimators' tuple list to decide which estimator to use to apply the imputation with.

Files to be changed: We will need to change _iterative.py to edit the IterativeImputer class for the interface as well as adding the new parameter to the class. We will need to also add unit tests to the impute/tests/tests_impute.py file to ensure that our changes do not break existing code.

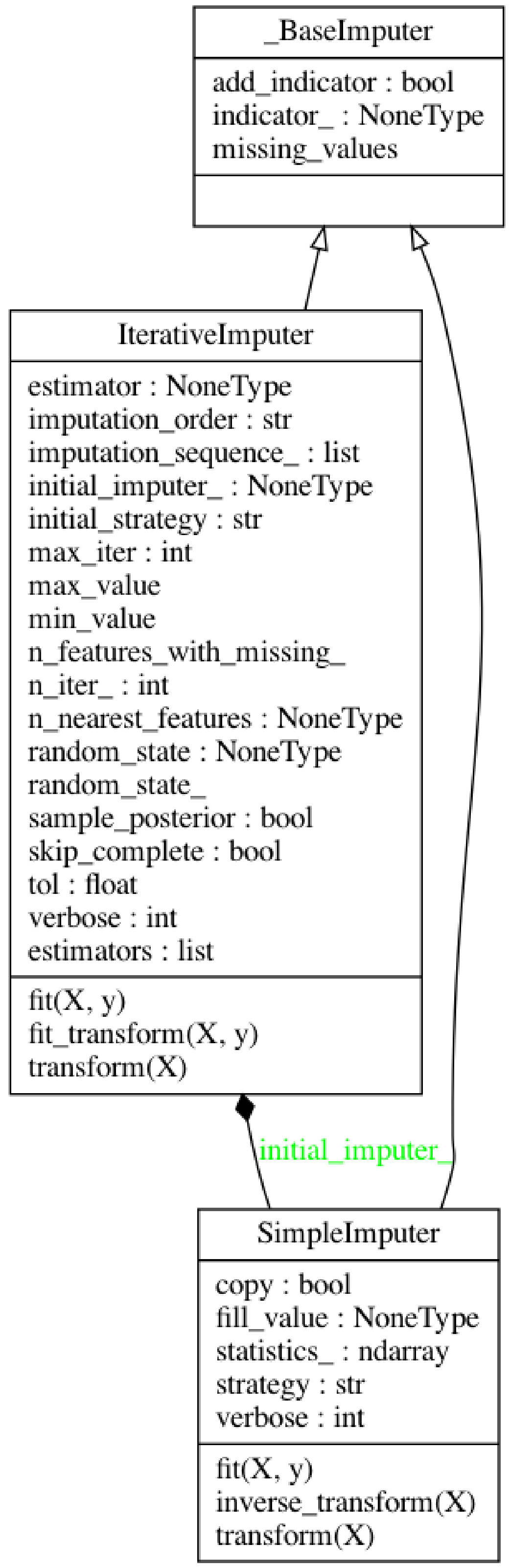

Figure 1: UML diagram for module 'impute'

Issue # 18553

Link to scikit-learn: https://github.com/scikit-learn/scikit-learn/issues/18553

Type: New feature

Description: Add “most_frequent” drop method to OneHotEncoder

Estimate: 21

Preface: In Encoders, there is a parameter in the constructor that the user can specify to determine how the OneHotEncoder object will drop columns in the transformed dataset. Currently there are ‘first’, ‘array’ and ’binary’ methodologies; however it was requested that a ‘most_frequent’ methodology be implemented for situations when the most frequent column should be dropped

Solution Proposal: Add an option ‘most_frequent’ to the ‘drop’ parameter in the constructor of the OneHotEncoder class. If this option is enabled, the object should keep track of how frequently each column appears in the transformed dataset in a list. After the transformation is done, simply drop the column that appears the most frequently in the dataset.

Files to be changed: We will need to edit the preprocessing/_encoders.py file to add the new most_frequent drop option to the OneHotEncoder class. We will also need to add unit tests to preprocessing/tests/test_encoders.py to ensure that the behaviour is correctly implemented and works under all situations.

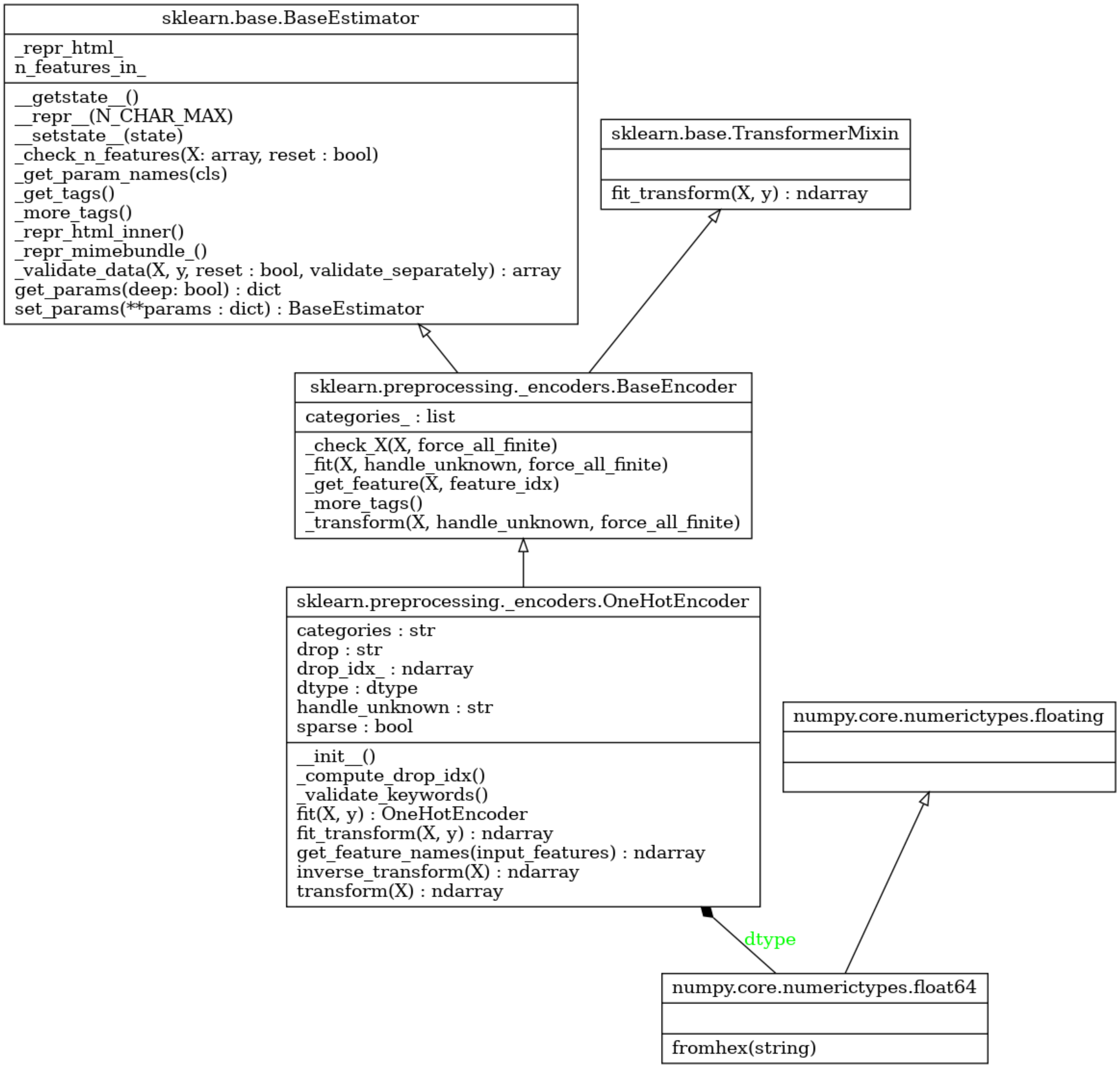
sklearn.base.BaseEstimator
_repr_html_
n_features_in_
__getstate__()
__repr__(N_CHAR_MAX)
__setstate__(state)
_check_n_features(X: array, reset : bool)
_get_param_names(cls)
_get_tags()
_more_tags()
_repr_html_inner()
_repr_mimebundle_()
_validate_data(X, y, reset : bool, validate_separately) : array
get_params(deep: bool) : dict
set_params(**params : dict) : BaseEstimator
sklearn.base.TransformerMixin
fit_transform(X, y) : ndarray
sklearn.preprocessing._encoders.BaseEncoder
categories_ : list
_check_X(X, force_all_finite)
_fit(X, handle_unknown, force_all_finite)
_get_feature(X, feature_idx)
_more_tags()
_transform(X, handle_unknown, force_all_finite)
sklearn.preprocessing._encoders.OneHotEncoder
categories : str
drop : str
drop_idx_ : ndarray
dtype : dtype
handle_unknown : str
sparse : bool
__init__()
_compute_drop_idx()
_validate_keywords()
fit(X, y) : OneHotEncoder
fit_transform(X, y) : ndarray
get_feature_names(input_features) : ndarray
inverse_transform(X) : ndarray
transform(X) : ndarray
numpy.core.numerictypes.floating
dtype
numpy.core.numerictypes.float64
fromhex(string)

Figure 2: UML diagram for ‘OneHotEncoder

Implementation Decision

We have decided to work on issue #18553 due to several factors:

- At first we investigated issue #14253 which was related to the the 'impute' module which we were familiar with, but upon further investigation, the feature was deemed too difficult to be done in the amount of time we were given
- We also wanted to investigate other areas of the scikit-learn library so that we would have a broader understanding of how users preprocess data prior to entering the data into estimators
- The issue we chose required us to do additional research on OneHotEncoders, examine the existing class and it's inner workings, and will require substantial development while also maintaining a realistic timeline for us to complete in time

Customer Acceptance Tests

Test for drop='most_frequent': *test_one_hot_encoder_drop_equals_most_frequent()*

1) Have a dataset with two categorical features one with unique categories and the other with one category appearing twice.
2) Initialize OneHotEncoder with drop='most_frequent' and sparse='False'.
3) Then call fit_transform on the encoder.
4) The expected outcome is the first category removed from the first feature and the most frequent category removed from the second feature.

Modified Tests

- *test_one_hot_encoder_inverse(sparse_, drop)*, *test_one_hot_encoder_drop_reset(drop, reset_drop)*, *test_one_hot_encoder_feature_names_drop(drop, expected_names)*
  - Added parameters for drop option 'most_frequent and appropriate expected outcome in the test.
- *test_categories(density, drop)*
  - Added parameters for drop option 'most_frequent in the test
  - Added branch in if statement to test for 'most_frequent' drop option
  - Since the test data set contains all unique categorical values, drop will remove the first category of each feature. We evaluate this by checking the drop indexes of categories are equal with the expected and actual outcome.

User Guide

Refer to the documentation for the preprocessing to see our changes to the existing user guide. We added documentation for the OneHotEncoder and explained it's behaviour.