A3 Report

Scikit-learn

Table of Contents

Analysis of 2 Issues

Bug: PowerTransformer Divide by Zero in Log (Issue 14959):

Analysis of Issue

As indicated by the title, this issue occurs in PowerTransformer, specifically in the `fit_transform` method. In the issue description, we are given a use case where the divide by zero warning occurs. Upon further investigation, using the warning message listed in the issue report: “`loglike = -n_samples / 2 * np.log(x_trans.var())`” we located that the source of this warning was the `_neg_log_likelihood` method. In this method the value of x_trans.var() could potentially return 0, as it does in this use case. Thus when the method takes the log of `x_trans.var()` (line 3242) it returns -INF. Thus it returns 0 values in the returned array as opposed to the correctly fitted data.

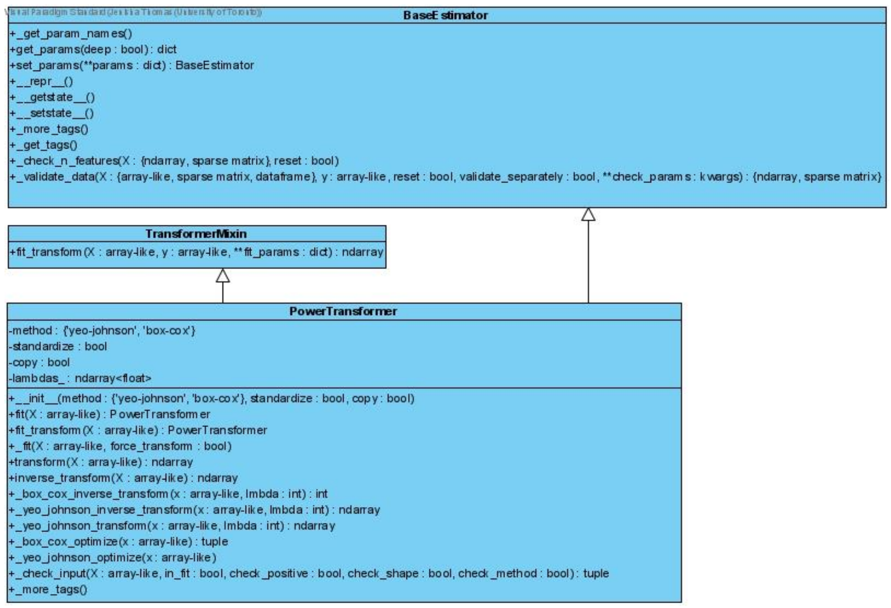
Affected areas in codebase

The files this issue concerns are the following:

- /sklearn/preprocessing/data.py

Proposed solutions and designs

A potential fix for this issue would be a simple check on the variance of the transform values of x. If the variance is 0, we simply return INF for the log likelihood. Since this change simply alters the values return based on a condition, the overall architecture of the class remains unchanged, and the PowerTransformer class would still be represented with the following UML:



BaseEstimator
+_get_param_names()
+get_params(deep : bool) : dict
+set_params(**params : dict) : BaseEstimator
+__repr__()
+__getstate__()
+__setstate__()
+_more_tags()
+_get_tags()
+_check_n_features(X : {ndarray, sparse matrix}, reset : bool)
+_validate_data(X : {array-like, sparse matrix, dataframe}, y : array-like, reset : bool, validate_separately : bool, **check_params : kwargs) : {ndarray, sparse matrix}
TransformerMixin
+fit_transform (X : array-like, y : array-like, **fit_params : dict) : ndarray
PowerTransformer
-method : {'yeo-johnson', 'box-cox'}
-standardize : bool
-copy : bool
-lambdas_ : ndarray<float>
+__init__(method : {'yeo-johnson', 'box-cox'}, standardize : bool, copy : bool)
+fit(X : array-like) : PowerTransformer
+fit_transform (X : array-like) : PowerTransformer
+_fit(X : array-like, force_transform : bool)
+transform(X : array-like) : ndarray
+inverse_transform(X : array-like) : ndarray
+_box_cox_inverse_transform (x : array-like, lmbda : int) : int
+_yeo_johnson_inverse_transform(x : array-like, lmbda : int) : ndarray
+_yeo_johnson_transform(x : array-like, lmbda : int) : ndarray
+_box_cox_optimize(x : array-like) : tuple
+_yeo_johnson_optimize(x : array-like)
+_check_input(X : array-like, in_fit : bool, check_positive : bool, check_shape : bool, check_method : bool) : tuple
+_more_tags()

Enhancement: Missing features removal with SimpleImputer (Issue 16426):

Analysis of Issue

As stated by the issue report, the SimpleImputer removes any feature that has the value of np.nan for all training samples, without notifying the user. This entails a variety of risks that can be caused by the change in shape of the dataset. From the issue report, we identified that the main class we would be working with is the SimpleImputer class, found in impute/_base.py. The methods of concern are the fit and transform methods.

Affected areas in codebase

- /sklearn/impute/_base.py
- /sklearn/impute/_knn.py
- /sklearn/impute/_iterative.py

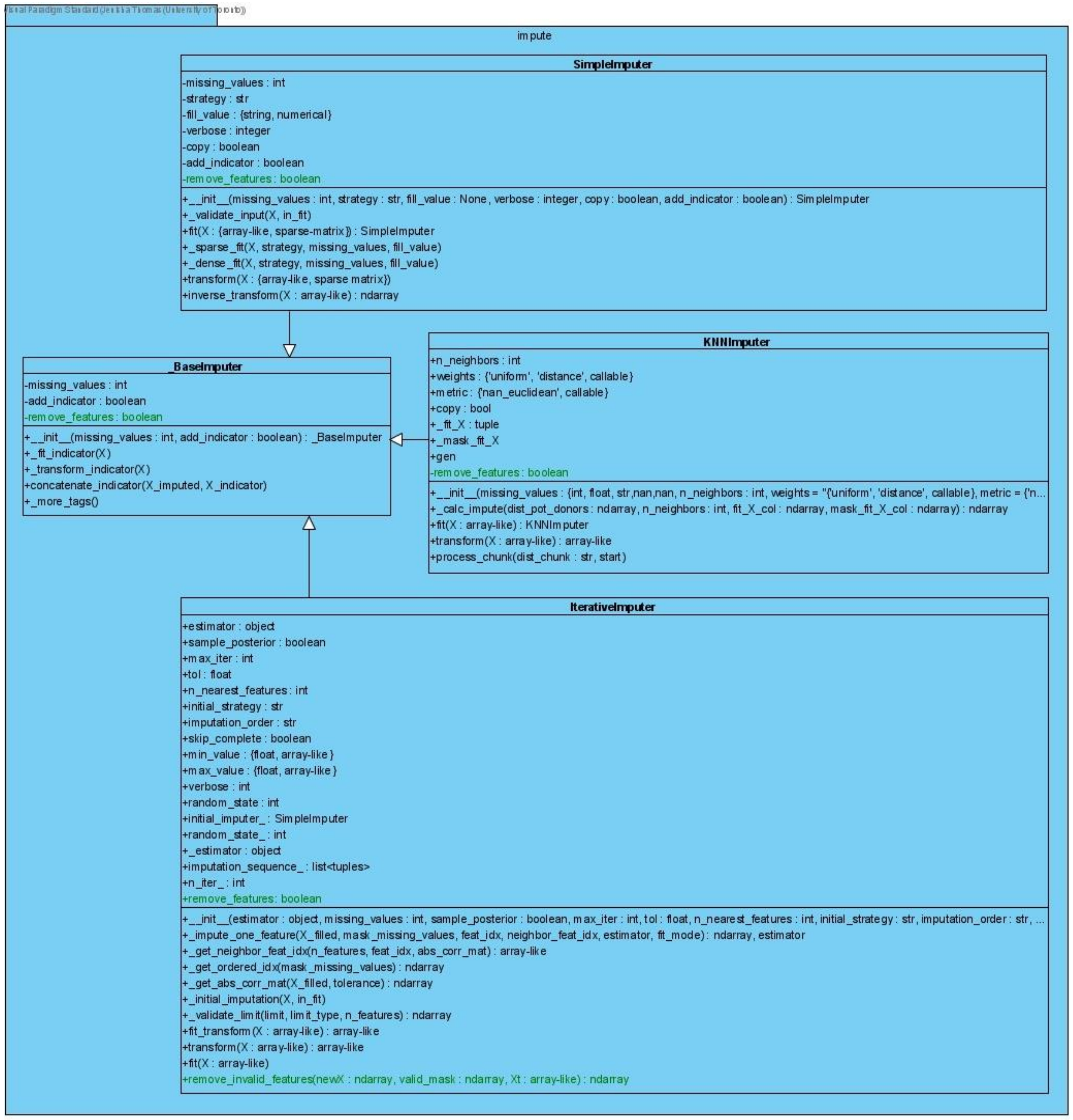
Proposed solutions and designs

There are 3 possible implementations that can be used to resolve this issue.

1. Add a boolean parameter to each of the Imputer classes that will be used to indicate whether NaN columns will be removed or not. This allows the user to decide on how to handle the missing values, so they will be entirely aware of what is happening to their data.
2. Raise a ValueError when there is a NaN column present in the data. Although this is better than having the program silently remove the column, since the user will be aware of the issue, it is also unfavorable as this will halt the execution and cause problems further down the line
3. Switch the imputation strategy to constant, when a NaN column is present. By doing this we can easily replace the missing values with a constant value, preventing any issues that may be caused by having NaN values in calculations. However, the statistics of the data may become inaccurate because of this as the constant value may be an outlier for the dataset.

After analyzing the pros and cons of each possible solution, we decided that option 1; adding a new parameter would be the best resolution to the issue.

Our proposed solution of adding an additional parameter to indicate whether NaN columns are to be kept result in a minor change in the architecture of the Imputer classes. For all of the four imputer classes we added a new attribute `remove_features` which is a boolean value used to indicate whether NaN columns are to be removed. Alongside this, for the `IterativeImputer` class we introduced the function `remove_invalid_features`, used to remove the missing/invalid features when `remove_features` is true. These changes are indicated in green in the UML below:


Visual Paradigm Standard(Jessica Thomas(University of Toronto))
impute
SimpleImputer
-missing_values : int
-strategy : str
-fill_value : {string, numerical}
-verbose : integer
-copy : boolean
-add_indicator : boolean
-remove_features : boolean
+__init__(missing_values : int, strategy : str, fill_value : None, verbose : integer, copy : boolean, add_indicator : boolean) : SimpleImputer
+_validate_input(X, in_fit)
+fit(X : {array-like, sparse-matrix}) : SimpleImputer
+_sparse_fit(X, strategy, missing_values, fill_value)
+_dense_fit(X, strategy, missing_values, fill_value)
+transform(X : {array-like, sparse matrix})
+inverse_transform(X : array-like) : ndarray
_BaseImputer
-missing_values : int
-add_indicator : boolean
-remove_features : boolean
+__init__(missing_values : int, add_indicator : boolean) : _BaseImputer
+_fit_indicator(X)
+_transform_indicator(X)
+concatenate_indicator(X_imputed, X_indicator)
+_more_tags()
KNNImputer
+n_neighbors : int
+weights : {'uniform', 'distance', callable}
+metric : {'nan_euclidean', callable}
+copy : bool
+_fit_X : tuple
+_mask_fit_X
+gen
-remove_features : boolean
+__init__(missing_values : {int, float, str,nan,nan, n_neighbors : int, weights = "{'uniform', 'distance', callable}, metric = {'n...
+_calc_impute(dist_pot_donors : ndarray, n_neighbors : int, fit_X_col : ndarray, mask_fit_X_col : ndarray) : ndarray
+fit(X : array-like) : KNNImputer
+transform(X : array-like) : array-like
+process_chunk(dist_chunk : str, start)
IterativeImputer
+estimator : object
+sample_posterior : boolean
+max_iter : int
+tol : float
+n_nearest_features : int
+initial_strategy : str
+imputation_order : str
+skip_complete : boolean
+min_value : {float, array-like}
+max_value : {float, array-like}
+verbose : int
+random_state : int
+initial_imputer_ : SimpleImputer
+random_state_ : int
+_estimator : object
+imputation_sequence_ : list<tuples>
+n_iter_ : int
+remove_features: boolean
+__init__(estimator : object, missing_values : int, sample_posterior : boolean, max_iter : int, tol : float, n_nearest_features : int, initial_strategy : str, imputation_order : str, ...
+_impute_one_feature(X_filled, mask_missing_values, feat_idx, neighbor_feat_idx, estimator, fit_mode) : ndarray, estimator
+_get_neighbor_feat_idx(n_features, feat_idx, abs_corr_mat) : array-like
+_get_ordered_idx(mask_missing_values) : ndarray
+_get_abs_corr_mat(X_filled, tolerance) : ndarray
+_initial_imputation(X, in_fit)
+_validate_limit(limit, limit_type, n_features) : ndarray
+fit_transform(X : array-like) : array-like
+transform(X : array-like) : array-like
+fit(X : array-like)
+remove_invalid_features(newX : ndarray, valid_mask : ndarray, Xt : array-like) : ndarray

Imputer Enhancement Implementation

User Guide

Since we made some changes to the Imputer classes, the user guide had to be updated to reflect our changes. The three pages that had to be updated were `SimpleImputer`, `KNNImputer` and `IterativeImputer`. In these pages, we included a description for the extra parameter we added called `remove_features`. This parameter is boolean type, which will help determine how we handle cases with NaN value columns when imputing a matrix. For the `IterativeImputer`, we added an extra function which removed NaN values when the remove_features parameter is true. Since this was a new function that was introduced, added that into the user guide as well.

Links to the user guides for each of the Imputer Classes:

- `SimpleImputer`: https://simple-imputer-userguide.herokuapp.com/
- `KNNImputer`: https://knn-imputer-userguide.herokuapp.com/
- `IterativeImputer`: https://iterative-imputer-userguide.herokuapp.com/

** Since only the user guide source files are included for the classes modified, not all of the internal links in the updated user guides work

User Acceptance Tests

For SimpleImputer:

1. Create a matrix X with dimension m x n where, m can be equal to n and one of the column has NaN values
2. Create a matrix Y with dimension m x n where m can be equal to n (same size and shape as X)
3. Provide the statement `impute = SimpleImputer(remove_features=False)` along with fitting X using `impute.fit(X)` and transform matrix Y by `impute.transform(Y)`.
4. The expected outcome of this function will be that the matrix Y will not lose the corresponding column of X that contains NaN values.

For KNNImputer:

1. Create a matrix X with dimension m x n where, m can be equal to n and one of the column has NaN values
2. Create a matrix Y with dimension m x n where m can be equal to n (same size and shape as X)
3. Provide the statement `impute = KNNImputer(remove_features=False)` along with fitting X using `impute.fit(X)` and transform matrix Y by `impute.transform(Y)`.
4. The expected outcome of this function will be that the matrix Y will not lose the corresponding column of X that contains NaN values.

For IterativeImputer:

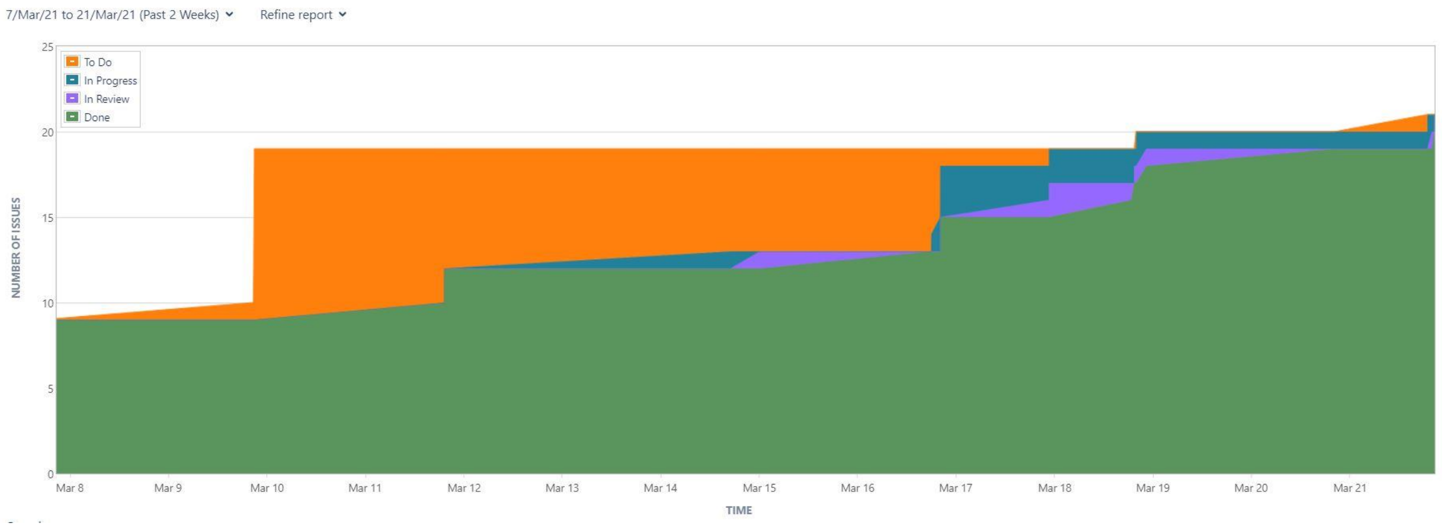
1. Create a matrix X with dimension m x n where, m can be equal to n and one of the column has NaN values
2. Create a matrix Y with dimension m x n where m can be equal to n (same size and shape as X)
3. Provide the statement `impute = IterativeImputer(remove_features=False)` along with fitting X using `impute.fit(X)` and transform matrix Y by `impute.transform(Y)`.
4. The expected outcome of this function will be that the matrix Y will not lose the corresponding column of X that contains NaN values.

Unit Testing Instructions

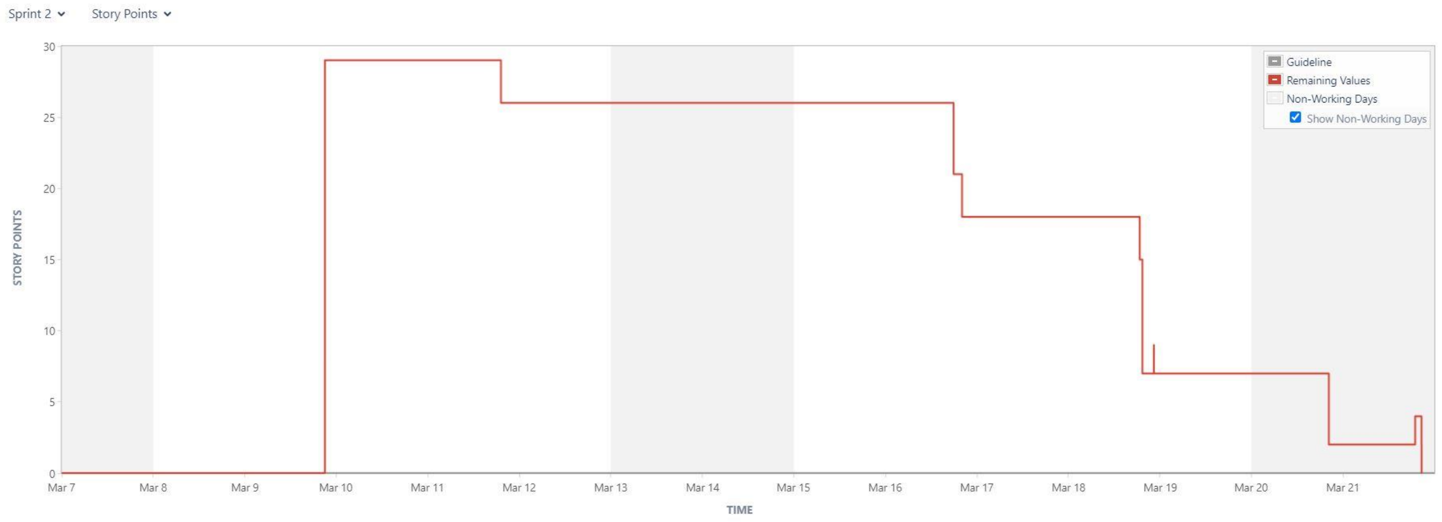
The cases can be run by using the command “`pytest 16426-testsuite.py`” in your terminal opened in the `a3` directory after successfully installing scikit-learn.

Development Process

JIRA



Cumulative Flow Diagram



Burndown Chart