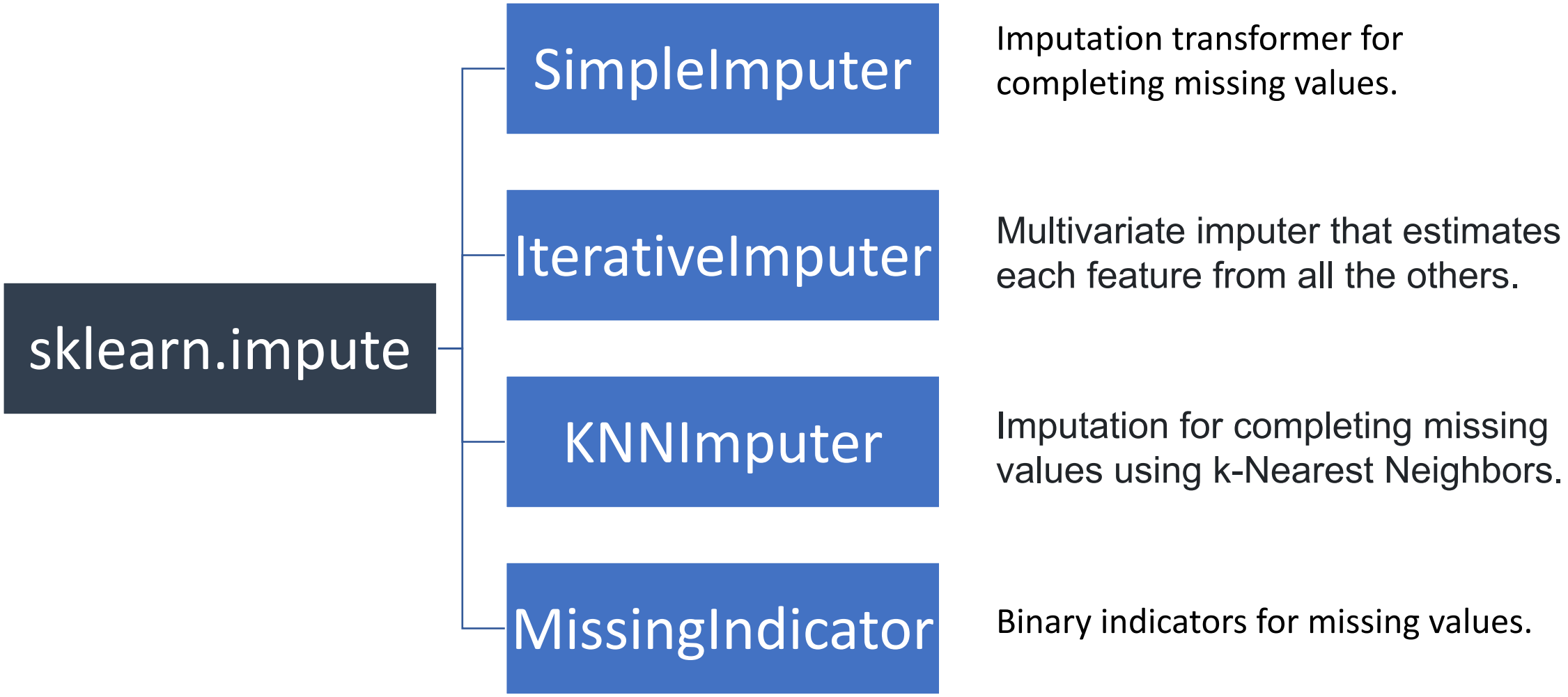
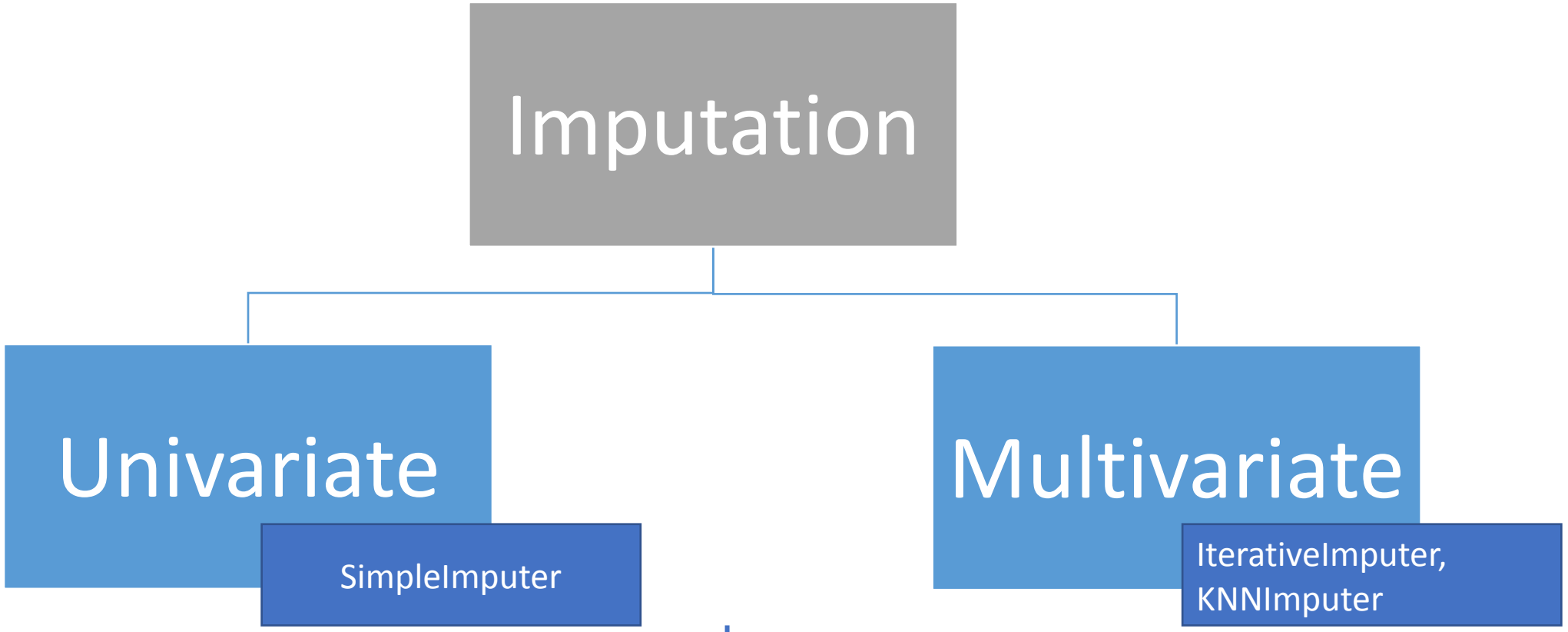
sklearn.impute

Transformers for missing value imputation

Lyahnovich Kirill, 2021

What's inside this module?





Imputes values in the i-th feature dimension using only non-missing values **in that feature dimension**

The **entire set of available feature dimensions** may be used to estimate the missing values

All imputers implement methods:

fit(X[, y])	Fit the imputer on X.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
inverse_transform(X)	Convert the data back to the original representation.
set_params(**params)	Set the parameters of this estimator.
transform(X)	Impute all missing values in X.

Simple Imputer

class sklearn.impute.SimpleImputer

```
SimpleImputer(
missing_values=nan
,
strategy='mean',
 fill_value=None,
 verbose=0,
 copy=True,
 add_indicator=False
)
```

The placeholder for the missing values

The imputation strategy: 'constant', 'mean', 'median' or 'most_frequent'

Needed if strategy is 'constant'

Controls the verbosity of the imputer.

If True, a copy of data will be created.

If True, a **MissingIndicator** transform will stack onto output of the imputer's transform.

Example

[4]

```
import numpy as np
from sklearn.impute import SimpleImputer
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
imp_mean.fit([[7, 2, 3],
              [4, np.nan, 6],
              [10, 5, 9]])

SimpleImputer()
X =  [[np.nan, 2, 3],
      [4, np.nan, 6],
      [10, np.nan, 9]]
print(imp_mean.transform(X))
```

```
[[ 7.   2.   3. ]
 [ 4.   3.5  6. ]
 [10.   3.5  9. ]]
```

Iterative Imputer

class sklearn.impute.IterativeImputer

A strategy for imputing missing values by modeling each feature with missing values as a function of other features in a round-robin fashion.

At each step, a feature column is designated as output y and the other feature columns are treated as inputs X. A regressor is fit on (X, y) for known y. Then, the regressor is used to predict the missing values of y. This is done for each feature in an iterative fashion, and then is repeated for max_iter imputation rounds. The results of the final imputation round are returned.

Iterative Imputer

class sklearn.impute.IterativeImputer

```
IterativeImputer(
 estimator=None
 missing_values=nan,
 initial_strategy='mean',
 n_nearest_features=None,
 verbose=0,
 imputation_order='ascending',
 random_state=None
....
many other settings
)
```

The estimator to use at each step of the round-robin imputation. default=BayesianRidge()

The placeholder for the missing values

How to initialize missing data : 'constant', 'mean', 'median' or 'most_frequent'

Number of other features to use to estimate the missing values of each feature column. If None, all features will be used.

Controls the verbosity of the imputer.

The order in which the features will be imputed. Possible values: **"ascending", "descending", "roman", "arabic", "random"**

The seed of the pseudo random number generator to use.

Example

[1]

```
import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
imp_mean = IterativeImputer(random_state=0)
imp_mean.fit([[7, 2, 3],
              [4, np.nan, 6],
              [10, 5, 9]])
IterativeImputer(random_state=0)
X = [[np.nan, 2, 3],
    [4, np.nan, 6],
    [10, np.nan, 9]]
imp_mean.transform(X)
```

```
array([[ 6.95847623,  2.        ,  3.        ],
       [ 4.        ,  2.6000004 ,  6.        ],
       [10.        ,  4.99999933,  9.        ]])
```

k-Nearest Neighbors Imputer

class sklearn.impute.KNNImputer

```
KNNImputer(
 missing_values=nan,
 n_neighbors=5,
 weights='uniform’,
 metric='nan_euclidean’,
 copy=True,
 add_indicator=False
)
```

The placeholder for the missing values

Number of neighboring samples to use for imputation.

Weight function used in prediction. Possible values: ‘uniform’ , ‘distance’ or user-defined function

Distance metric for searching neighbors. Possible values: ‘nan_euclidean’, or user-defined function

If True, a copy of data will be created.

If True, a **MissingIndicator** transform will stack onto output of the imputer’s transform.

Example

[12]

```
import numpy as np
from sklearn.impute import KNNImputer
X = [[1, 2, np.nan], [3, 4, 3], [np.nan, 6, 5], [8, 8, 7]]
imputer = KNNImputer(n_neighbors=2)
imputer.fit_transform(X)
```

```
array([[1. , 2. , 4. ],
       [3. , 4. , 3. ],
       [5.5, 6. , 5. ],
       [8. , 8. , 7. ]])
```

Marking imputed values

class sklearn.impute.MissingIndicator

The MissingIndicator transformer is useful to transform a dataset into corresponding binary matrix indicating the presence of missing values in the dataset. This transformation is useful in conjunction with imputation.

```
MissingIndicator(
 missing_values=nan,
 features='missing-only',
 sparse='auto',
 error_on_new=True’,
)
```

The placeholder for the missing values

Whether the imputer mask should represent all or a subset of features. Could be ‘missing-only’ or ‘all’

Whether the imputer mask format should be sparse or dense. True, False or ‘auto’

If True, transform will raise an error when there are features with missing values in transform that have no missing values in fit. This is applicable only when features='missing-only'

Example

[21]

```
import numpy as np
from sklearn.impute import MissingIndicator
X = np.array([[-1, -1, 1, 3],
              [4, -1, 0, -1],
              [8, -1, 1, 0]])
indicator = MissingIndicator(missing_values=-1)
mask_missing_values_only = indicator.fit_transform(X)
mask_missing_values_only
```

```
array([[ True,  True, False],
       [False,  True,  True],
       [False,  True, False]])
```

[22]

```
indicator = MissingIndicator(missing_values=-1, features="all")
mask_all = indicator.fit_transform(X)
mask_all
```

```
array([[ True,  True, False, False],
       [False,  True, False,  True],
       [False,  True, False, False]])
```