Train In Data

Imputation with Scikit-learn

Scikit-learn supports basic imputation

- Mean / median imputation
- Arbitrary number imputation
- Frequent category imputation
- Arbitrary category imputation, i.e., “missing”
- Missing indicators

SimpleImputer

sklearn.impute.SimpleImputer

class `sklearn.impute.SimpleImputer`(*, *missing_values=nan*, *strategy='mean'*, *fill_value=None*, *verbose='deprecated'*, *copy=True*, *add_indicator=False*, *keep_empty_features=False*) [source]

Univariate imputer for completing missing values with simple strategies.

Replace missing values using a descriptive statistic (e.g. mean, median, or most frequent) along each column, or using a constant value.

Read more in the User Guide.

New in version 0.20: `SimpleImputer` replaces the previous `sklearn.preprocessing.Imputer` estimator which is now removed.

Parameters:

missing_values : *int, float, str, np.nan, None or pandas.NA, default=np.nan*

The placeholder for the missing values. All occurrences of `missing_values` will be imputed. For pandas' dataframes with nullable integer dtypes with missing values, `missing_values` can be set to either `np.nan` or `pd.NA`.

strategy : *str, default='mean'*

The imputation strategy.

fit() and transform()

- fit() ➔ learns parameters from train set
- transform() ➔ transforms data
- set_output() ➔ to return dataframes

SimpleImputer - mean

```
imp_mean = SimpleImputer(strategy='mean')

imp_mean.fit(X_train)

X_train = imp_mean.transform(X_train)
X_test = imp_mean.transform(X_test)
```

SimpleImputer - median

```
imp_mean = SimpleImputer(strategy='median')

imp_mean.fit(X_train)

X_train = imp_mean.transform(X_train)
X_test = imp_mean.transform(X_test)
```

SimpleImputer - mode

```
imp_mean = SimpleImputer(
        strategy='most_frequent')


imp_mean.fit(X_train)


X_train = imp_mean.transform(X_train)
X_test = imp_mean.transform(X_test)
```

SimpleImputer - arbitrary number

```
imp_mean = SimpleImputer(
        strategy='constant',
        fill_value=99,
)

imp_mean.fit(X_train)

X_train = imp_mean.transform(X_train)
X_test = imp_mean.transform(X_test)
```

SimpleImputer - arbitrary category

```
imp_mean = SimpleImputer(
        strategy='constant',
        fill_value='missing',
)

imp_mean.fit(X_train)

X_train = imp_mean.transform(X_train)
X_test = imp_mean.transform(X_test)
```

SimpleImputer - missing indicators

```
imp_mean = SimpleImputer(
        strategy='mean',
        add_indicator=True,
)

imp_mean.fit(X_train)

X_train = imp_mean.transform(X_train)
X_test = imp_mean.transform(X_test)
```

SimpleImputer - advantages

- Stores the learned parameters: mean, median, mode
- Fast computation (uses NumPy under the hood)
- Single class – multiple imputation methods

SimpleImputer - limitations

- Imputes entire dataframe or array
- Returns a NumPy array instead of a pandas dataframe
- Can return dataframe if we specify output type beforehand.

ColumnTransformer - subset of features

sklearn.compose.ColumnTransformer

class `sklearn.compose.`**`ColumnTransformer`**(*transformers*, *, *remainder='drop'*, *sparse_threshold=0.3*, *n_jobs=None*, *transformer_weights=None*, *verbose=False*, *verbose_feature_names_out=True*) [source]

Applies transformers to columns of an array or pandas DataFrame.

This estimator allows different columns or column subsets of the input to be transformed separately and the features generated by each transformer will be concatenated to form a single feature space. This is useful for heterogeneous or columnar data, to combine several feature extraction mechanisms or transformations into a single transformer.

Read more in the User Guide.

New in version 0.20.

Parameters:

transformers : *list of tuples*

List of (name, transformer, columns) tuples specifying the transformer objects to be applied to subsets of the data.

name : *str*

Like in Pipeline and FeatureUnion, this allows the transformer and its parameters to be set using `set_params` and searched in grid search.

ColumnTransformer

```
ct = ColumnTransformer(
    [(“num_imp",
      SimpleImputer(strategy=‘mean'),
      [“Income”, “Age”]),

     (“cat_imp",
      SimpleImputer(strategy=‘most_frequent'),
      [“Color”, “Make”]),
      ],
)
```

ColumnTransformer - advantages

- Keep working within the Scikit-learn API – full compatibility
- Fast computation (uses NumPy under the hood)

ColumnTransformer - limitations

- Changes variable names
- Difficult to set up when modifying different variable subsets with different methods in sequence

Accompanying Jupyter Notebook

- Jupyter Notebooks in **sklearn folder**
- Demo of different imputation methods

Train In Data

THANK YOU

www.trainindata.com