5-handling-missing-data

October 20, 2024

1 Handling Missing Data

Handling missing data is a crucial step in the data preprocessing phase of machine learning projects. Missing values can arise from various reasons, such as incomplete data collection, sensor errors, or data entry issues. If not handled properly, missing data can significantly degrade model performance or cause errors during training and testing.

Why Handling Missing Data is Important:

1. Maintaining Data Integrity: Missing values can skew analysis results, leading to incorrect conclusions and poor model performance.
2. Ensuring Model Robustness: Most machine learning models (especially algorithms like linear regression, logistic regression, or SVM) cannot handle missing values directly, so preprocessing is necessary to ensure smooth execution.
3. Preserving Dataset Size: Rather than discarding records with missing values, handling them appropriately allows you to use as much of your data as possible.

Different Techniques for Handling Missing Data:

1. Removing Missing Data:
    - Dropping Rows: If the number of rows with missing values is small, you can drop those rows.
    - Dropping Columns: If an entire column has too many missing values (e.g., more than 50%), you may choose to drop that feature.
2. Imputation (Filling Missing Data):
    - Mean/Median/Mode Imputation: Replace missing values with the mean, median, or mode of the non-missing values in that column.
    - Forward/Backward Fill: Use previous or next values to fill missing entries (mainly used in time-series data).
    - Predictive Imputation: Use a machine learning model to predict the missing values based on other features.
    - K-Nearest Neighbors (KNN) Imputation: Use KNN to find similar data points and fill missing values based on the nearest neighbors.
3. Flagging/Indicator Variables:

- Create a new binary feature indicating whether the value in the original column was missing or not. This approach allows the model to know which data points had missing values.

4. Using Algorithms that Handle Missing Data:
   - Some machine learning algorithms like decision trees and random forests can handle missing values internally without requiring imputation.

Practical Considerations:

1. Nature of Missing Data: Before choosing a method, understand the nature of missing data. Is it Missing Completely at Random (MCAR), Missing at Random (MAR), or Not Missing at Random (NMAR)? Different techniques are better suited for different types of missingness.
2. Preserving Relationships: While imputation preserves data, it may introduce bias or distort relationships between features, especially if the missing data is not random..
3. Multiple Imputation: For more advanced scenarios, techniques like Multiple Imputation (e.g., MICE - Multiple Imputation by Chained Equations) are used to generate multiple datasets with different imputations, combining their results to reduce imputation bias.

Summary:

- Removing Missing Data: Drop rows or columns with missing values if the missing proportion is small.
- Imputation: Replace missing values with mean, median, mode, or more advanced techniques like KNN or predictive imputation.
- Flagging: Create binary flags for missing values to allow the model to capture this information.
- Advanced Algorithms: Some algorithms like decision trees and XGBoost handle missing values naturally without preprocessing.

[1]:
```
import pandas as pd
import numpy as np

miles = pd.DataFrame({'farthest_run_mi': [50,62, np.nan,100,26,13,31,50]})
miles
```

[1]:

	farthest_run_mi
0	50.0
1	62.0
2	NaN
3	100.0
4	26.0
5	13.0
6	31.0
7	50.0

[2]:
```
miles.isna().sum()
```

[2]:
```
farthest_run_mi    1
dtype: int64
```

[3]:
```
from sklearn.impute import SimpleImputer
imp_mean = SimpleImputer(strategy='mean')
imp_mean.fit_transform(miles)
```

[3]:
```
array([[ 50.        ],
       [ 62.        ],
       [ 47.42857143],
       [100.        ],
       [ 26.        ],
       [ 13.        ],
       [ 31.        ],
       [ 50.        ]])
```

[4]:
```
imp_median = SimpleImputer(strategy='median')
imp_median.fit_transform(miles)
```

[4]:
```
array([[ 50.],
       [ 62.],
       [ 50.],
       [100.],
       [ 26.],
       [ 13.],
       [ 31.],
       [ 50.]])
```

[5]:
```
imp_mode = SimpleImputer(strategy='most_frequent')
imp_mode.fit_transform(miles)
```

[5]:
```
array([[ 50.],
       [ 62.],
       [ 50.],
       [100.],
       [ 26.],
       [ 13.],
       [ 31.],
       [ 50.]])
```

[6]:
```
imp_constant = SimpleImputer(strategy='constant', fill_value=13)
imp_constant.fit_transform(miles)
```

[6]:
```
array([[ 50.],
       [ 62.],
       [ 13.],
       [100.],
       [ 26.],
```

```
       [ 13.],
       [ 31.],
       [ 50.]])
```

[7]:
```
names = pd.DataFrame({'names':['Ryan', 'Nolan', 'Honus', 'Wagner', np.nan,␣
 ↪'Ruth']})
names
```

[7]:
```
    names
0    Ryan
1   Nolan
2   Honus
3  Wagner
4     NaN
5    Ruth
```

[8]:
```
imp_constant_cat = SimpleImputer(strategy='constant', fill_value='babe')
imp_constant_cat.fit_transform(names)
```

[8]:
```
array([['Ryan'],
       ['Nolan'],
       ['Honus'],
       ['Wagner'],
       ['babe'],
       ['Ruth']], dtype=object)
```

[9]:
```
imp_mean_marked = SimpleImputer(strategy='mean', add_indicator= True)
imp_mean_marked.fit_transform(miles)
```

[9]:
```
array([[ 50.        ,   0.        ],
       [ 62.        ,   0.        ],
       [ 47.42857143,   1.        ],
       [100.        ,   0.        ],
       [ 26.        ,   0.        ],
       [ 13.        ,   0.        ],
       [ 31.        ,   0.        ],
       [ 50.        ,   0.        ]])
```

[13]:
```
new_df = pd.DataFrame({
    'Name': ['Ryan', 'Nolan', 'Walter', 'Honus', 'Christy', np.nan, 'Napoleon',␣
 ↪'Tris'],
    'farthest_run_mi': [50,62, np.nan,100,26,13,31,50]
})
new_df
```

[13]:
```
       Name  farthest_run_mi
0      Ryan             50.0
```

```
1     Nolan            62.0
2    Walter             NaN
3     Honus           100.0
4   Christy            26.0
5       NaN            13.0
6  Napoleon            31.0
7      Tris            50.0
```

[14]:
```
from sklearn.compose import make_column_transformer
ct = make_column_transformer(
    (imp_constant_cat, ['Name']),
    (imp_mean, ['farthest_run_mi']),
    remainder='drop'
)
ct.set_output(transform='pandas')
```

[14]:
```
ColumnTransformer(transformers=[('simpleimputer-1',
                                 SimpleImputer(fill_value='babe',
                                               strategy='constant'),
                                 ['Name']),
                                ('simpleimputer-2', SimpleImputer(),
                                 ['farthest_run_mi'])])
```

[15]:
```
df_pandas = ct.fit_transform(new_df)
df_pandas
```

[15]:
```
  simpleimputer-1__Name  simpleimputer-2__farthest_run_mi
0                  Ryan                         50.000000
1                 Nolan                         62.000000
2                Walter                         47.428571
3                 Honus                        100.000000
4               Christy                         26.000000
5                  babe                         13.000000
6              Napoleon                         31.000000
7                  Tris                         50.000000
```