Dhirubhai Ambani Institute of Information and Communication Technology

Exploratory Data Analysis, 2024

Missingno package

Group Number : 22

Group Members :
Bhavya Boda (202203067)
Krishil Jayswal (202203040)
Aniket Pandey (202411001)

Course Instructor:
Gopinath Panda

Missingno Package

1. Introduction

scikit-learn (sklearn) is a powerful and widely used library in Python for machine learning, known for its simplicity and efficiency. Built on essential scientific libraries like NumPy and SciPy, it provides a comprehensive toolkit for data mining and analysis. Users can access a wide variety of algorithms for classification, regression, clustering, and dimensionality reduction, along with robust preprocessing tools for scaling, encoding, and handling missing values. The library also features convenient model evaluation functions and cross-validation techniques, making it easier to assess model performance.

2. Steps Undertaken

In the colab file, we followed a systematic approach to explore the features of sklearn's preprocessing and modeling capabilities. The steps undertaken are as follows:

1. **Data Loading and Exploration**: We began by loading the dataset and performing initial exploratory data analysis (EDA) to understand its structure, check for missing values, and identify relevant features.
2. **Data Preprocessing**: This involved handling missing values using SimpleImputer from sklearn to ensure data quality. We also applied scaling and encoding techniques to prepare the data for modeling.
3. **Model Building**: We constructed multiple regression models using a pipeline, including Linear Regression and Random Forest Regressor, to establish a framework for training and evaluation.
4. **Model Training and Evaluation**: Each model was trained on the training dataset, and predictions were made on the test dataset. Performance metrics such as Mean Absolute Error (MAE) and R-squared score were calculated to evaluate model performance.
5. **Performance Comparison**: Finally, we compared the performance of the different models to determine which was the most effective based on the metrics calculated.

This structured approach enabled a comprehensive understanding of the impact of preprocessing on model performance and highlighted the capabilities of sklearn in the machine learning workflow. Now we will start to address each of these steps in detail.

3. Data Loading and Exploration

The initial step in this practical lab involved loading the dataset and performing a basic exploration to understand its structure and content. We utilized the pandas library, which is well-known for its data manipulation capabilities. The following operations were carried out during this phase:

- pd.read_csv(): This function was used to load the dataset from a CSV file into a pandas DataFrame, allowing us to handle the data in a tabular format.
- df.info(): We used this method to get a concise summary of the dataset, including the number of non-null entries, data types of each column, and memory usage. This helped in identifying columns with missing values and understanding the dataset's overall structure.
- df.head(): To preview the first few rows of the dataset, this method was used. It provided a quick glance at the data, helping us confirm that the data was loaded correctly.
- df.columns and df.shape: These operations were used to check the names of the columns and the dimensions of the dataset (i.e., the number of rows and columns). This was important for understanding the scale of the data.
- df.isnull().sum(): This function was employed to detect missing values across all columns, which is a crucial step before proceeding to data cleaning. Knowing the extent of missing data helped in planning the preprocessing phase.

Through these operations, we were able to gain an initial understanding of the dataset, including its structure, data types, and any missing or incomplete data that required further attention. This foundational step ensured a clear overview of the dataset, setting the stage for subsequent data cleaning and preparation.

4. Data Preprocessing

Data preprocessing is a critical step to ensure the dataset is clean, consistent, and suitable for training machine learning models. In this practical lab, we utilized various features from the `scikit-learn` package to handle missing values and transform categorical data into a numerical format. The following methods were employed:

4.1 Handling Missing Values: `SimpleImputer`

To address missing values in the dataset, we used `SimpleImputer`. This tool provides strategies for imputing missing values, ensuring that no information is lost due to incomplete data. For numerical data, missing values were replaced with the mean of the column, while for categorical data, the most frequent value was used. Below are the key parameters of `SimpleImputer`:

```
from sklearn.impute import SimpleImputer
```

Parameters:

- `missing_values`: Specifies the placeholder for missing values. Default is `np.nan`.
- `strategy`: Determines the imputation strategy. Options include:
  - `"mean"` (default): Replace missing values using the mean along the column (used for numerical data).
  - `"most_frequent"`: Replace missing values using the most frequent value (used for categorical data).
  - `"median"`: Replace missing values using the median along the column.
  - `"constant"`: Replace missing values with a constant value, specified by the `fill_value` parameter.

4.2 Encoding Categorical Variables: `OneHotEncoder`

To convert categorical variables into a numerical format, we used `OneHotEncoder`. This technique transforms each unique category within a feature into a set of binary indicators (0s and 1s), enabling the models to process these features effectively. Key parameters include:

```
from sklearn.preprocessing import OneHotEncoder
```

Parameters:

- `sparse`: If set to `True` (default), the encoder returns sparse matrices. If set to `False`, it returns dense matrices.
- `handle_unknown`: Determines the behavior when an unknown category is encountered. Options include `'error'` (default) or `'ignore'`.

4.3 Standardizing Numerical Features: `StandardScaler`

To ensure that numerical features have a mean of 0 and a standard deviation of 1, we used `StandardScaler`. This scaling is important for many machine learning algorithms that are sensitive to the magnitude of the input features. Key parameters include:

```
from sklearn.preprocessing import StandardScaler
```

Parameters:

- `with_mean`: If set to `True` (default), it centers the data before scaling by subtracting the mean.
- `with_std`: If set to `True` (default), it scales the data to unit variance (dividing by the standard deviation).

4.4 Column Transformation: `ColumnTransformer`

`ColumnTransformer` from `sklearn.compose` is a utility that allows us to apply different preprocessing pipelines to different subsets of features in the dataset. It is especially useful when handling datasets that have both numerical and categorical features, as it enables us to preprocess them separately and combine the results seamlessly.

```
from sklearn.compose import ColumnTransformer
```

Parameters:

- `transformers`: A list of (name, transformer, columns) tuples specifying which transformer to apply to which columns. The `transformer` can be any transformer object like `Pipeline`, `SimpleImputer`, or `StandardScaler`.
- `remainder`: Specifies how to handle the remaining columns not explicitly selected. Options include `'drop'` (default) or `'passthrough'` to include them unchanged.

We used `ColumnTransformer` to apply separate pipelines to numerical and categorical data. This allowed us to impute missing values, scale numerical data, and encode categorical data using a single, unified step.

4.5 Creating a Preprocessing Pipeline: `Pipeline`

We utilized the `Pipeline` feature from `scikit-learn` to streamline the preprocessing workflow. The pipeline allowed us to combine various preprocessing steps into a single, cohesive operation, ensuring the same transformations were consistently applied to both training and testing datasets. Key parameters of `Pipeline` include:

```
from sklearn.pipeline import Pipeline
```

Parameters:

- `steps`: A list of tuples where each tuple contains the name of a step and an estimator (e.g., `('imputer', SimpleImputer())`).
- `memory`: Optional parameter to cache the transformation steps.

These preprocessing techniques ensured that our data was well-prepared for the modeling process. By organizing these steps into a pipeline, we made the process efficient, reproducible, and easy to apply across different datasets.

5. Model Building Using Pipelines

After preprocessing the data, the next step was to build machine learning models to predict the target variable. We used `Pipeline` from `scikit-learn` to combine preprocessing steps and model training into a single, cohesive workflow. This approach allowed us to streamline the process and ensure that the same transformations were applied consistently across training and testing datasets.We used two models for our report which are discussed as follows.

5.1 Linear Regression Model

The first model we implemented was `LinearRegression`, which aims to predict a continuous target variable by fitting a linear equation to the input features. Linear regression is simple, interpretable, and effective for datasets with linear relationships.

```
from sklearn.linear_model import LinearRegression
```

- `fit_intercept`: Whether to calculate the intercept for this model. If set to `False`, no intercept will be used in calculations.
- `normalize`: If `True`, the regressors will be normalized before fitting. Deprecated in newer versions.

5.2 Random Forest Regressor

To capture more complex, non-linear patterns in the data, we also used `RandomForestRegressor`. This model is an ensemble learning method that combines multiple decision trees to provide robust predictions.

```
from sklearn.ensemble import RandomForestRegressor
```

- `n_estimators`: The number of trees in the forest. A higher number of trees improves performance but may increase computation time.
- `max_depth`: The maximum depth of the trees. Limiting depth can prevent overfitting.
- `random_state`: Ensures reproducibility by fixing the random seed.
- `n_jobs`: The number of cores used for parallel processing. Setting `-1` uses all available cores.

5.3 Follow Up

After building all the above models, next comes the training part. We trained these modesl and calculated some standard benchmark scores which is discussed in next section.

6. Model Training and Evaluation

In this section, we describe the steps taken to train our machine learning models and evaluate their performance using appropriate metrics. The primary steps involved in this process include splitting the dataset into training and testing sets, training the models, and assessing their performance using metrics like Mean Absolute Error (MAE) and R-squared (R^2) score.

6.1 Train-Test Split

To ensure that the model is capable of generalizing well to unseen data, we utilized the `train_test_split` function from `sklearn.model_selection`. This function randomly divides the dataset into two subsets: a training set, which is used to train the model, and a testing set, which is used to evaluate the model's performance.

```
from sklearn.model_selection import train_test_split
```

Parameters:

- `X`: Feature dataset (input variables).
- `y`: Target variable (output variable).
- `test_size`: Proportion of the dataset to include in the test split (default is 0.25, i.e., 25%).
- `random_state`: Controls the shuffling applied to the data before splitting. Passing an integer will ensure reproducibility.

6.2 Model Training

After splitting the dataset, we trained multiple models, including Linear Regression and Random Forest. Each model was fitted to the training data using the `fit` method, allowing them to learn patterns and relationships within the data.

```
model.fit(X_train, y_train)
```

The `fit` method takes in the training features `X_train` and the target variable `y_train` to train the model.

6.3 Performance Evaluation

To evaluate the performance of the trained models, we employed two key metrics:

- **Mean Absolute Error (MAE)**: This metric quantifies the average magnitude of the errors in a set of predictions, without considering their direction. It is calculated as follows:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  where y_i represents the true values and $\hat{y}_i$ represents the predicted values.

```
from sklearn.metrics import mean_absolute_error
```

  Parameters:

  - `y_true`: Ground truth (correct) target values.
  - `y_pred`: Estimated target values from the model.

- **R-squared Score (R^2)**: This statistic indicates how well the independent variables explain the variability of the dependent variable. An R^2 score of 1 indicates perfect prediction, while a score of 0 indicates that the model does not explain any of the variability in the target variable. The formula for R^2 is given by:

$$R^2 = 1 - \frac{\text{SS}_{res}}{\text{SS}_{tot}}$$

  where SS_{res} is the sum of squares of residuals and SS_{tot} is the total sum of squares.

```
from sklearn.metrics import r2_score
```

  Parameters:

  - `y_true`: Ground truth (correct) target values.
  - `y_pred`: Estimated target values from the model.

By utilizing these evaluation metrics, we were able to gauge the performance of our models effectively and make informed decisions regarding their suitability for our predictive tasks.

7. Model Performance Comparison and Conclusion

In this section, we compare the performance of the two models—Linear Regression and Random Forest—using the evaluation metrics Mean Absolute Error (MAE) and R-squared score (R^2). The results obtained are summarized below:

7.1 Model Performance Metrics

- **Linear Regression:**
  - Mean Absolute Error: 976.50
  - R-squared Score: 0.9747
  - Mean Absolute Error / Mean Percentage: 7.17%
  - Median Absolute Error / Median Percentage: 8.07%
- **Random Forest:**
  - Mean Absolute Error: 971.76
  - R-squared Score: 0.9747
  - Mean Absolute Error / Mean Percentage: 7.14%
  - Median Absolute Error / Median Percentage: 8.03%

Model Predictions Visualization: Predicted vs Actual Values

To further understand the performance of our models, we plotted the predicted values against the actual values from the test dataset. This visualization helps in assessing how well the models are capturing the underlying trends in the data. A perfect model would result in all points lying on the 45-degree line, indicating that predicted values match the actual values perfectly.

The scatter plot provides insights into the model's performance, revealing any patterns of bias or systematic errors in the predictions. Analyzing the spread of the points in relation to the diagonal line can help identify areas where the model performs well and where it may need improvement.

For creating this visualization, we utilized the `matplotlib` library, which is a powerful tool for creating static, animated, and interactive visualizations in Python.

7.3 Comparative Analysis

Both models exhibit similar performance in terms of the R-squared score, indicating that they explain a comparable amount of variance in the target variable. However, when it comes to Mean Absolute Error (MAE), the Random Forest model slightly outperforms the Linear Regression model, achieving a lower MAE of 971.76 compared to Linear Regression's MAE of 976.50. This demonstrates that the Random Forest model has a better predictive accuracy.

The percentage metrics for both models also reveal similar trends, with Random Forest showing slightly better mean and median absolute error percentages (7.14% and 8.03%, respectively) compared to Linear Regression (7.17% and 8.07%).

7.3 Conclusion

The analysis indicates that both models perform well, with Random Forest demonstrating marginally better performance than Linear Regression in terms of predictive accuracy. However, it is essential to note that the high performance of the Random Forest model comes at the cost of increased computational intensity. This means that while Random Forest may yield more accurate predictions, it requires more computational resources and time for training compared to Linear Regression.

In summary, the choice of model may depend on the specific requirements of the application, including the need for computational efficiency versus the need for accuracy in predictions.

8. Github Link

These complete process of performing each steps are available on github. Click the below link button to access the github link.

Access Link