Optilearn.optimal.OptimalDataSelector

08 November 2024 13:44

OptimalDataSelector

Overview

The OptimalDataSelector function aims to find the best data combination for predictive models by optimizing accuracy with different data sizes. It helps in selecting the most effective subset of data by utilizing various parameters like boosting, scaling, and the choice between classification or regression tasks. It also allows for saving and loading previously computed data combinations to enhance the process.

Parameters

1. **predictor** (array-like):
    - The independent variables or features that are used to predict the target variable. This is required for both classification and regression tasks.
2. **target** (array-like):
    - The dependent variable or target that you want to predict. This is needed for both classification and regression tasks.
3. **combination** (int, optional, default=10):
    - The number of data combinations to evaluate. Increasing this number will test more potential data combinations for optimal model performance. The default value is 10.
4. **random_state** (int, optional, default=0):
    - The seed for random number generation, ensuring reproducibility of results. A fixed integer seed will allow consistent splits across different runs.
5. **train_size** (float, optional, default=0.7):
    - The proportion of the data to be used for training the model. By default, 70% of the data is used for training and the remaining 30% for testing.
6. **bs_problem** (str, optional, default='class'):
    - Specifies the type of problem being solved:
        - 'class' for classification.
        - 'reg' for regression.
7. **time_forecast** (bool, optional, default=True):
    - Flag indicating whether to print the computation time. If set to True, the function will output the time taken for the calculations.
8. **scaling** (str, optional, default=None):
    - Determines how the data should be scaled before processing:
        - 'normal' scales the data to a [0,1] range.
        - 'st_normal' applies standard normalization (zero mean, unit variance).
        - If None, no scaling is applied.
9. **acc_forecast** (bool, optional, default=True):
    - Flag indicating whether to print the probable accuracy after selecting the best data combination. If True, accuracy will be printed.
10. **boost** (str, optional, default=None):
    - Flag to control boosting behavior:
        - None means the function will prioritize stable combinations without considering time complexity.
        - 'prim' aims for a balance between stability and time efficiency.
        - 'max' prioritizes time efficiency over stability.
11. **randomness** (bool, optional, default=True):
    - Flag to control the randomization of the data combinations:
        - True will allow random mixing of the data.
        - False will retain sequential order for data splitting.
12. **load_path** (str, optional, default=None):
    - File path to a previously saved data combination structure. This allows the function to recall and reuse the data split and combination without recalculating them.
13. **active_checkpoint** (bool, optional, default=False):
    - If True, the function will prompt the user to save the combination of data and will track every combination for future use. If False, the function will run without prompting for saving data.

Returns

- **x_train** (array-like):
    - The training set for the independent variables.
- **x_test** (array-like):
    - The testing set for the independent variables.
- **y_train** (array-like):
    - The training set for the target variable.
- **y_test** (array-like):
    - The testing set for the target variable.

Notes

- **Classification Problems**:
    - If bs_problem is set to 'class', the function will treat the task as a classification problem.
- **Regression Problems**:
    - If bs_problem is set to 'reg', the function will treat the task as a regression problem.
- **Accuracy Calculation**:
    - The function calculates the accuracy (or relevant metric for regression) for each combination and returns the combination that results in the highest accuracy.
- **Computation Time**:
    - If time_forecast is True, the function will print the computation time in minutes.
- **Scaling**:
    - The function supports data scaling using the scaling parameter. It scales the data if a valid scaling option ('normal' or 'st_normal') is provided.
- **Boosting**:
    - If boost is set to 'None', the function prioritizes stability of data combinations, which might take more time.
    - If boost is set to 'prim', a balance between stability and time efficiency is sought.
    - If boost is set to 'max', the function prioritizes time efficiency over data stability.
- **Randomization**:
    - If randomness is set to True, the function will randomly select data combinations. If False, the function will use sequential data combinations.
- **Reusing Data Combinations**:
    - If load_path is provided, the function will load a previously saved data combination structure and avoid recalculating combinations.
- **Checkpointing**:
    - If active_checkpoint is set to True, the function will prompt the user to save the data combination structure. This allows future use of the exact same data combinations.

Performance Considerations

- **Large Datasets**:
    - Be cautious when setting high values for combination with large datasets, as this may significantly increase the computation time.
- **Boosting**:
    - The boost parameter allows you to optimize the balance between stability and time-saving:
        - None > 'prim' > 'max' for stability.
        - 'max' > 'prim' > None for time-saving.

Examples

1. **Loading a previously saved combination**:

```
x_train, x_test, y_train, y_test = OptimalDataSelector(load_path="<path_to_saved_comb
```

2. **Classification Problem with Data Scaling**:

```
x_train, x_test, y_train, y_test = OptimalDataSelector(
    predictor, target, combination=5, random_state=42, train_size=0.8,
    bs_problem='class', scaling='normal', boost='max', randomness=False
)
```

3. **Regression Problem with Boosting**:

```
x_train, x_test, y_train, y_test = OptimalDataSelector(
    predictor, target, combination=10, random_state=0, train_size=0.7,
    bs_problem='reg', scaling='st_normal', boost='prim', randomness=True
)
```

Additional Considerations

- **Memory and Performance**:
  - The function may consume significant memory and time for very large datasets with high combinations. Therefore, adjust the combination parameter according to the available resources.