Optilearn.sequential.timeseries_split

08 November 2024 14:16

timeseries_split

Description:
The timeseries_split function is designed to split a time series dataset into training and testing sets, considering previous data points for prediction. It provides options for scaling the data, specifying the number of dimensions for the input sequences, and returning the indices of the training and testing data, especially useful for time-based indexing.

Parameters:

- **data** (pd.DataFrame or pd.Series):
  The input time series data that will be split into training and testing datasets. This can either be a DataFrame (with multiple columns) or a Series (for a single-column time series). The specified column (via column) will be extracted for use.
- **column** (str):
  The name of the column in the data to be used as the time series. This parameter is required when using a DataFrame to specify which column contains the time series data.
- **n_previous_values** (int):
  The number of previous time steps to consider as input features for prediction. For example, if n_previous_values = 30, the function will use the last 30 data points to predict the next value.
- **train_size** (float, optional, default=0.8):
  The proportion of the data that will be used for the training set. The default value is 0.8, meaning 80% of the data will be used for training, and the remaining 20% will be used for testing.
- **scaling** (str, optional, default=None):
  The type of scaling to be applied to the time series data:
  - 'normal': Applies standard normalization (scaling the data to a 0-1 range).
  - 'st_normal': Applies standardization (scaling the data to have a mean of 0 and a standard deviation of 1).
  - If None, no scaling is applied.
- **n_dimension** (int, optional, default=3):
  The number of dimensions for the input data:
  - 2: Returns 2D input sequences, which are suitable for simpler models that expect input data in a 2D form.
  - 3: Returns 3D input sequences, typically used for deep learning models like LSTMs or GRUs that expect sequential data with multiple time steps.
- **return_index** (str, optional, default=None):
  Specifies whether to return the indices of the training and testing data, which is especially useful when working with time-based data (like dates or timestamps). The options are:
  - 'training': Returns the indices for the training set.
  - 'validation': Returns the indices for the testing set.
  - 'both': Returns both training and testing indices.
  - None: Does not return any indices.

Returns:

- **tuple**:
  The function returns a tuple containing the split data and labels, which will differ based on the value of n_dimension:
  - If n_dimension = 2:
    - x_train (numpy.ndarray): 2D array of training input sequences.
    - x_test (numpy.ndarray): 2D array of testing input sequences.
    - y_train (numpy.ndarray): 1D array of training output values (the value to predict).
    - y_test (numpy.ndarray): 1D array of testing output values (the true value to compare against).
  - If n_dimension = 3:
    - x_train (numpy.ndarray): 3D array of training input sequences.
    - x_test (numpy.ndarray): 3D array of testing input sequences.
    - y_train (numpy.ndarray): 1D array of training output values.
    - y_test (numpy.ndarray): 1D array of testing output values.
      If return_index is specified, the function will also return the corresponding indices as:
  - train_indices: Indices of the training data.
  - test_indices: Indices of the testing data.

Example:

```
x_train, x_test, y_train, y_test, train_indices, test_indices = timeseries_split(
    data=data, column='column_name', n_previous_values=30, n_dimension=3, return_index='bo
```

In this example:

- data: The time series data in a DataFrame or Series.
- 'column_name': The column from which the time series will be extracted.
- n_previous_values=30: Uses the previous 30 time steps to predict the next value.
- n_dimension=3: Returns 3D input sequences for deep learning models.
- return_index='both': Returns both the training and testing indices.

Notes:

- The function works by sliding a window of size n_previous_values across the time series, using the previous n_previous_values as input to predict the next value.
- If scaling is applied (scaling), the data will be scaled before splitting into training and testing sets. This is particularly useful when working with neural networks or other models sensitive to the scale of the data.
- The split is done in a way that respects the time series order, ensuring that the training data comes from earlier time steps than the testing data.

Tutorials:

- **Query-1**: *How to handle missing values in time series before using timeseries_split?*
  Answer:
  Missing values in time series data can be handled by methods such as forward-fill, backward-fill, or interpolation. These methods can be applied using pandas' fillna() function before splitting the data using timeseries_split.
  Example:

  ```
  data['column_name'] = data['column_name'].fillna(method='ffill')
  ```

- **Query-2**: *What is the best value for n_previous_values?*
  Answer:
  The best value for n_previous_values depends on the nature of your data and the model you're using. In general, the larger the window, the more context the model has for making predictions, but it also increases the complexity. Experiment with different values to find the optimal size for your data.
- **Query-3**: *Can I use n_dimension=2 with a deep learning model?*

Answer:
While you can use n_dimension=2 with a deep learning model, models like LSTMs or GRUs typically expect 3D data (with time steps as the third dimension). If you are using a simpler model like linear regression or decision trees, n_dimension=2 should work well.

- **Query-4**: *What happens if I set return_index to 'both'?*
  Answer:
  When return_index is set to 'both', the function will return both the indices of the training and testing data, which can be useful for tracking the corresponding time points or dates of the split data.
- **Query-5**: *What is the effect of setting train_size to a value like 0.5?*
  Answer:
  Setting train_size=0.5 will split the data into equal halves, with 50% of the data used for training and 50% for testing. You can adjust this ratio depending on how much data you want to allocate to the training and testing sets. Typically, values between 0.7 and 0.9 are used for training.