Script

Got It!

1. Cross-validating timeseries data

We'll now discuss some of the basics of cross-validation, as well as how they relate to fitting timeseries data.

2. Cross validation with scikit-learn

Scikit-learn has many classes for cross-validation. For reference, here is the standard way to use each one of them.

3. Cross validation types: KFold

The most common form of cross-validation is k-fold cross-validation. In this case, data is split into K subsets of equal size. In each iteration, a single subset is left out as the validation set. Here we show how to initialize a k-fold iterator with scikit-learn.

4. Visualizing model predictions

Always visualize your model's behavior during cross-validation. Here, we first show how to plot the indices of the validation set for each iteration. Next, we visualize the predictions generated in this loop. This helps us perform sanity checks on the process.

5. Visualizing KFold CV behavior

Looking at the generated plot, we first see that the validation indices are chunked in "blocks" or "folds". This is the default cross-validation behavior. On the bottom we see three timeseries that were predicted, one for each iteration. They have the same general structure as time series data, which is a good sanity check.

6. A note on shuffling your data

Many cross-validation iterators let you randomly shuffle the data. This may be appropriate if your data is independent and identically distributed, but timeseries data is usually not i.i.d. Here we use the "ShuffleSplit" cross-validation iterator, which randomly permutes the data labels in each iteration. Let's see what our visualization looks like when using shuffled data.

7. Visualizing shuffled CV behavior

Here we can see from the top plot that the validation indices are all over the place, not in chunks like before. That's because we used a cross-validation object that shuffles the data. Below we see that the output data no longer "looks" like timeseries, because the temporal structure of the data has been destroyed. If the data is shuffled, it means that some information about the training set now exists in the validation set, and you can no longer trust the score of your model.

8. Using the time series CV iterator

Finally, we'll cover another cross-validation iterator that can help deal with these problems. There is one cross-validation technique that is meant particularly for time series data. This approach *always* uses data from the past to predict timepoints in the future. Through CV iterations, a larger amount of training data is used to predict the next block of validation data, corresponding to the fact that more time has passed. This more closely mimics the data collection and prediction process in the real world.

9. Visualizing time series cross validation iterators

We can use this cross-validation approach with the TimeSeriesSplit object in scikit-learn. Here we iterate through the object and plot the training data in blue and the validation data in red. Let's visualize this behavior.

10. Visualizing the TimeSeriesSplit cross validation iterator

Here we see how the TimeSeriesSplit CV object behaves. Each row is an iteration of the cross validation. In red we see the validation indices for that iteration. As you can see, the training data always comes before the validation data. This ensures that only the past is always used to predict the future.

11. Custom scoring functions in scikit-learn

You can also create custom scorers in scikit-learn. They must all take an estimator object, and X and y arrays as inputs, and output a single number, representing the score calculated after generating model predictions. You can use whatever function you like to create this score.

12. A custom correlation function for scikit-learn

Here we define our custom correlation function for scikit-learn. It generates model predictions, then uses numpy's corrcoef function to output a correlation matrix. We take a single value from this matrix, since there are only two variables in it, and return the value.