1. Stationarity and stability

In this lesson, we'll cover how to quantify variability in our models and how this relates to time series data.

2. Stationarity

A stationary signal is one that does not change its statistical properties over time. It has the same mean, standard deviation, and general trends. A non-stationary signal does change its properties over time. Each of these has important implications for how to fit your model.

3. Examples of stationary and non-stationary data

Here's an example of a stationary and a non-stationary signal. On the top, we can see that the signal generally does not change its structure. Its variability is constant throughout time. On the bottom, we see a signal that is highly non-stationary. Its variance and trends change over time. Almost all real world data are non-stationary. In fact, these two plots are of the same data, but in different ranges of time.

4. Model stability

Most models have an implicit assumption that the relationship between inputs and outputs is static. If this relationship changes (because the data is not stationary), then the model will generate predictions using an outdated relationship between inputs and outputs. How can we quantify and correct for this?

5. Cross validation to quantify parameter stability

One approach is to use cross-validation, which yields a set of model coefficients per iteration. We can quantify the variability of these coefficients across iterations. If a model's coefficients vary widely between cross-validation splits, there's a good chance the data is non-stationary (or noisy).

6. Bootstrapping the mean

Bootstrapping is a way to estimate the confidence in the mean of a collection of numbers. To perform a bootstrap for the mean, take many random samples (with replacement) from your collection of numbers and calculate the mean of each. Now, calculate lower/upper percentiles for this list. The lower and upper percentiles represent the variability of the mean.

7. Bootstrapping the mean

Here's an example using scikit-learn and numpy. Use the resample function in scikit-learn to take a random sample of coefficients, then use numpy to calculate the mean for each coefficient in the sample and store it in an array. Then, we calculate the 2-point-5 and 97-point-5 percentile of the results to calculate lower and upper bounds for each coefficient. This is called a 95% confidence interval.

8. Plotting the bootstrapped coefficients

Here we plot the lower and upper bounds of the 95% confidence intervals we calculated. This gives us an idea for the variability of the mean across all cross-validation iterations.

9. Assessing model performance stability

It's also common to quantify the stability of a model's predictive power across cross-validation folds. If you're using the TimeSeriesSplit object mentioned before, then you can visualize this as a timeseries.

10. Model performance over time

In this example, we'll use the cross_val_score function, along with the TimeSeriesSplit iterator, to calculate the predictive power of the model over cross-validation splits. We first create a small scoring function that can be passed to cross_val_score. Next we use a list comprehension to find the date of the beginning of each validation block. Finally, we collect the scores and convert them into a Pandas Series.

11. Visualizing model scores as a timeseries

Because the cross-validation splits happen linearly over time, we can visualize the results as a timeseries. If we see large changes in the predictive power of a model at one moment in time, it could be because the statistics of the data have changed. Here we create a rolling mean of our cross-validation scores and plot it with matplotlib.

12. Visualizing model scores

We can see the scores of our model across validation sets, which means over time. There is a clear dip in the middle, probably because the statistics of the data changed. What can we do about this?

13. Fixed windows with time series cross-validation

One option is to restrict the size of the training window. This ensures that only the latest datapoints are used in training. We can control this with the max_train_size parameter.

14. Non-stationary signals

Re-visiting our visualization from before, we see that restricting the training window slightly improves the dip in performance in the middle of our validation data.