Time Series: 6th lesson – Forecasting With Machine Learning

Defining the forecasting task:

There are two things to establish before designing a forecasting model.

1} What information is available at the time, a forecast is made as features

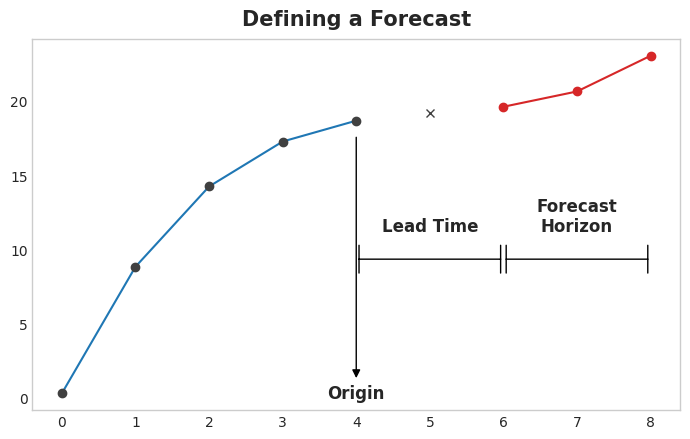
2} The time period during which forecasted values required as targets

Forecast origin:

The time at which forecast is made. Practically, users might consider the forecast origin to be the last time for which acquired training data for the time being predicted. Everything up to the origin can be used to create features.

Forecast horizon:

The time for which forecast is made. Programmers often describe a forecast by the number of time steps in its horizon, e.g. “1-step” forecast or “5-step” forecast. The forecast horizon also describes a target.



The time between the origin and the horizon is the lead time (or sometimes *latency*) of the forecast. A forecast's lead time is described by the number of steps from origin to horizon: a "1-step ahead" or "3-step ahead" forecast. In practice, it may be necessary for a forecast to begin multiple steps ahead of the origin because of delays in data acquisition or processing.

Preparing data for forecasting:

In order to forecast time series with ML algorithms, we need to transform the series into a DataFrame, we can use with those algorithms (unless, of course, you are only using deterministic features like trend and seasonality).

Each row in a DataFrame represents a single forecast. The time index of the row is the first time in the forecast horizon, but we arrange values for the entire horizon in the same row. For multistep forecasts, this means we are requiring a model to produce multiple outputs, one for each step.

import numpy as np

import pandas as pd

N = 20

ts = pd.Series(

np.arange(N),

index=pd.period\_range(start='2010', freq='A', periods=N, name='Year'),

dtype=pd.Int8Dtype,

)

*# Lag features*

X = pd.DataFrame({

'y\_lag\_2': ts.shift(2),

'y\_lag\_3': ts.shift(3),

'y\_lag\_4': ts.shift(4),

'y\_lag\_5': ts.shift(5),

'y\_lag\_6': ts.shift(6),

})

*# Multistep targets*

y = pd.DataFrame({

'y\_step\_3': ts.shift(-2),

'y\_step\_2': ts.shift(-1),

'y\_step\_1': ts,

})

data = pd.concat({'Targets': y, 'Features': X}, axis=1)

data.head(10).style.set\_properties(['Targets'], \*\*{'background-color': 'LavenderBlush'}) \

.set\_properties(['Features'], \*\*{'background-color': 'Lavender'})

Targets Features

y\_step\_3 y\_step\_2 y\_step\_1 y\_lag\_2 y\_lag\_3 y\_lag\_4 y\_lag\_5 y\_lag\_6

Year

2010 2 1 0 nan nan nan nan nan

2011 3 2 1 nan nan nan nan nan

2012 4 3 2 0 nan nan nan nan

2013 5 4 3 1 0 nan nan nan

2014 6 5 4 2 1 0 nan nan

2015 7 6 5 3 2 1 0 nan

2016 8 7 6 4 3 2 1 0

2017 9 8 7 5 4 3 2 1

2018 10 9 8 6 5 4 3 2

2019 11 10 9 7 6 5 4 3

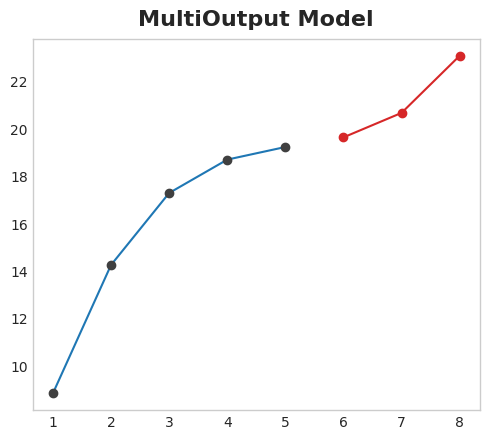
The above illustrates how a dataset would be prepared similar to the Defining a Forecast figure: a three-step forecasting task with a two-step lead time using five lag features. The original time series is y\_step\_1. The missing values we could either fill-in or drop.

Multistep forecasting strategies:

There are a number of strategies for producing the multiple target steps required for a forecast.We'll outline four common strategies, each with strengths and weaknesses.

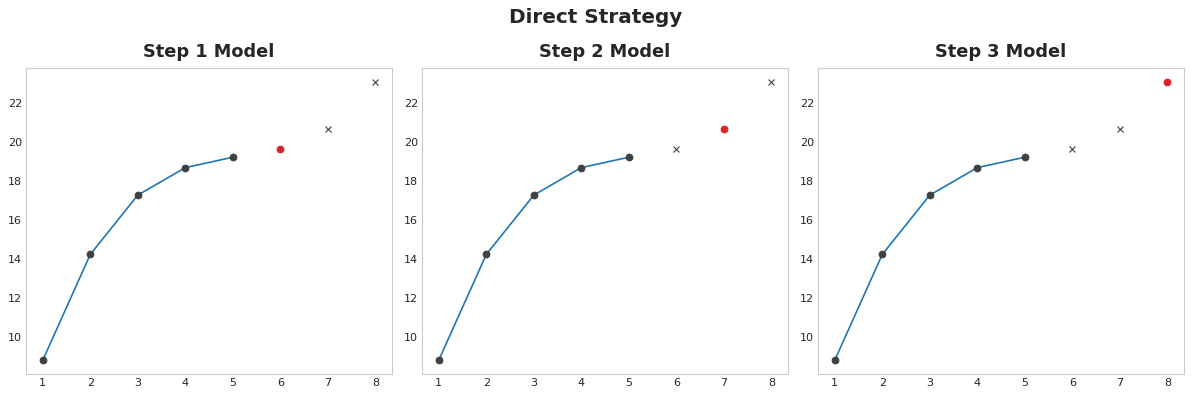
* Multi-output model

Use a model that produces multiple outputs naturally. Linear regression and neural networks can both produce multiple outputs. This strategy is simple and efficient, but not possible for every algorithm you might want to use. XGBoost can't do this, for instance.



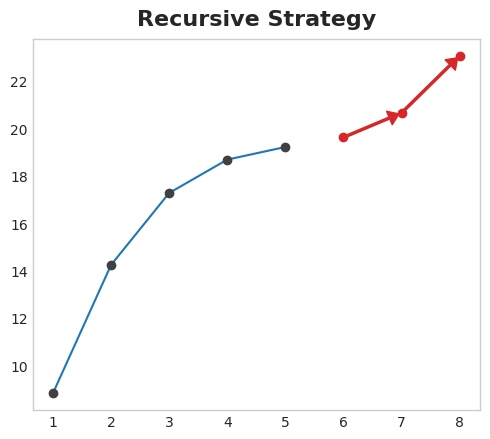
* Direct strategy

Train a separate model for each step in the horizon: one model forecasts 1-step ahead, another 2-steps ahead, and so on. Forecasting 1-step ahead is a different problem than 2-steps ahead (and so on), so it can help to have a different model make forecasts for each step. The downside is that training lots of models can be computationally expensive.



* Recursive strategy

Train a single one-step model and use its forecasts to update the lag features for the next step. With the recursive method, we feed a model's 1-step forecast back in to that same model to use as a lag feature for the next forecasting step. We only need to train one model, but since errors will propagate from step to step, forecasts can be inaccurate for long horizons.



* Direct-recursive strategy

A combination of the direct and recursive strategies: train a model for each step and use forecasts from previous steps as new lag features. Step by step, each model gets an additional lag input. Since each model always has an up-to-date set of lag features, the DirRec strategy can capture serial dependence better than Direct, but it can also suffer from error propagation like Recursive.

