Using MLJ

Lesson 1: Basics

Authors: Anthony Blaom

Introducing the “Using MLJ” lessons

MLJ is a Machine Learning toolbox written for Julia.

This series of video lessons focuses on how to **use** MLJ.

The lessons are not a replacement for a formal machine learning course, but can be used in conjunction with such a course.

The lessons have been developed by **PumasAI**, and are used in the *AI for Drug Development Course* offered by **PumasAI** and its partner **SOPHAS**.

Introducing the “Using MLJ” lessons

Series prerequisites

We assume you already know some Julia. At the very least:

- You are confident interacting with Julia using the **REPL**.
- You know how to install new Julia packages and how to manage Julia **package environments**.

There are lots of online resources for getting up to speed with Julia.

Any machine learning practice depends on a solid grounding in Statistics and Linear Algebra.

Lesson 1 Goals

We want to:

1. Get to know the basic MLJ objects: **models**, **machines**
2. Understand the basic **fit/predict** workflow
3. Know how to estimate model **performance**:
   - by hand
   - with advanced tools (`evaluate`)
4. Understand **scientific types** and their role in data preparation (`scitype`, `schema`)

Lesson 1 Prerequisites

1. Familiarity with the concept of **supervised learning**
2. Understand the distinction between **hyper-parameters** and **learned parameters** in learning.
3. Previous exposure to the idea of **cross-validation**

Getting more help

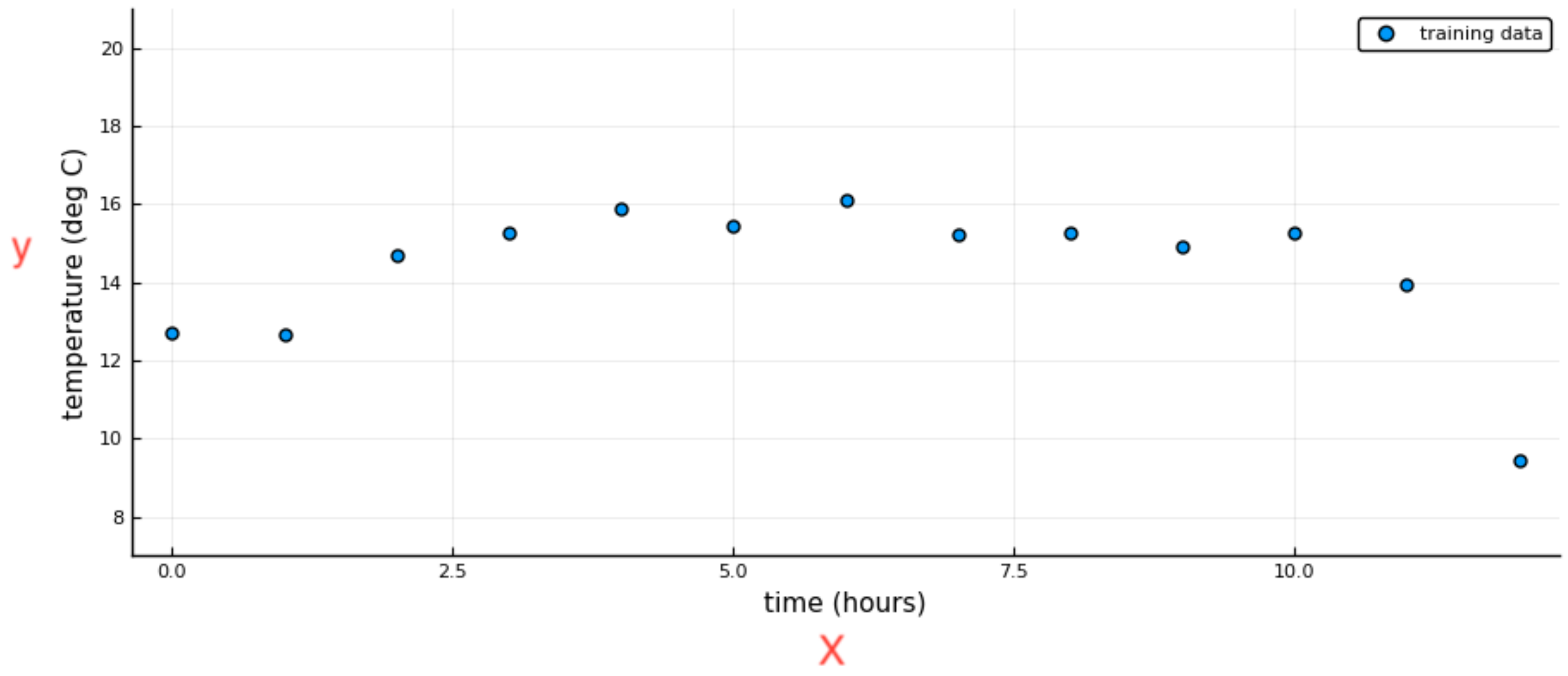
The **Resources** page linked below contains:

- Slides for this presentation
- Julia code for the demos
- Links to general MLJ learning resources

https://github.com/JuliaAI/MLJ.jl/tree/dev/examples/using_mlj

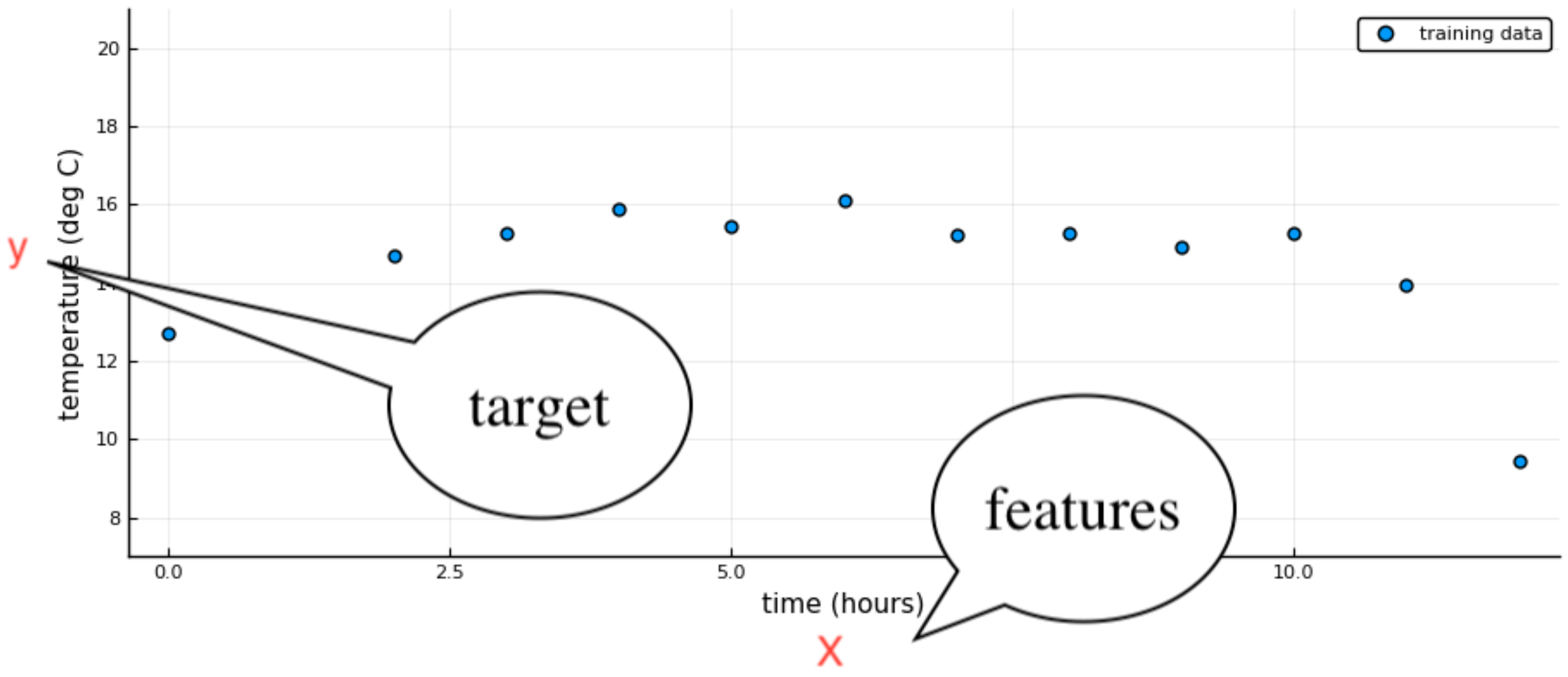
Supervised learning

You are given **hourly** temperature readings in a room:



Supervised learning

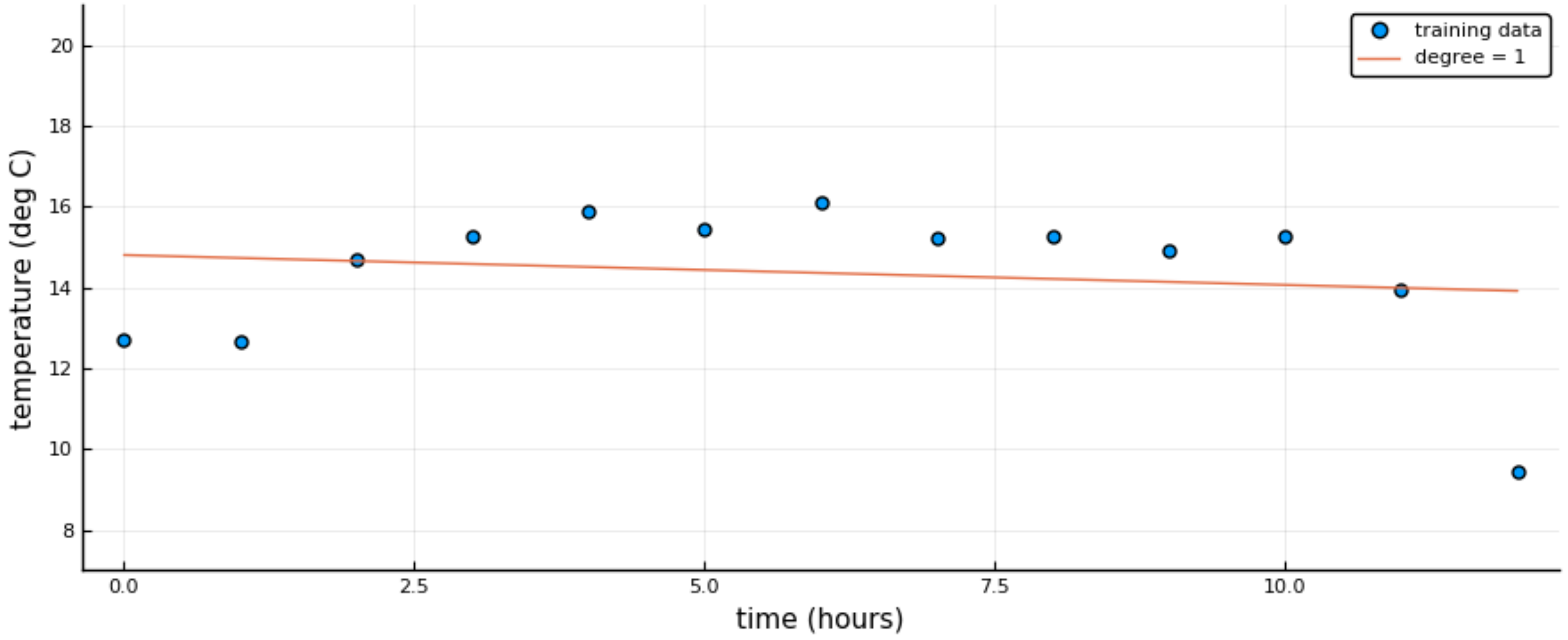
You are given **hourly** temperature readings in a room:



Your mission: Predict the temperature at any time.

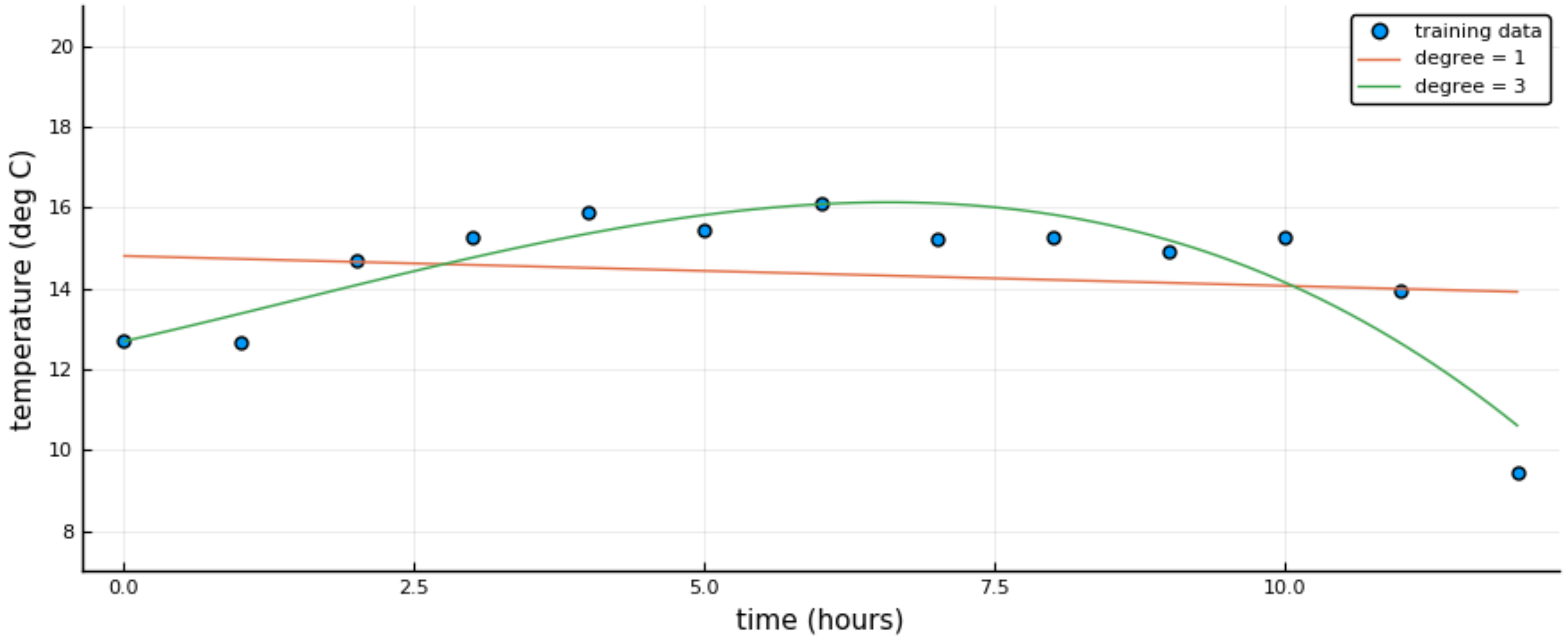
Supervised learning

You try linear regression:



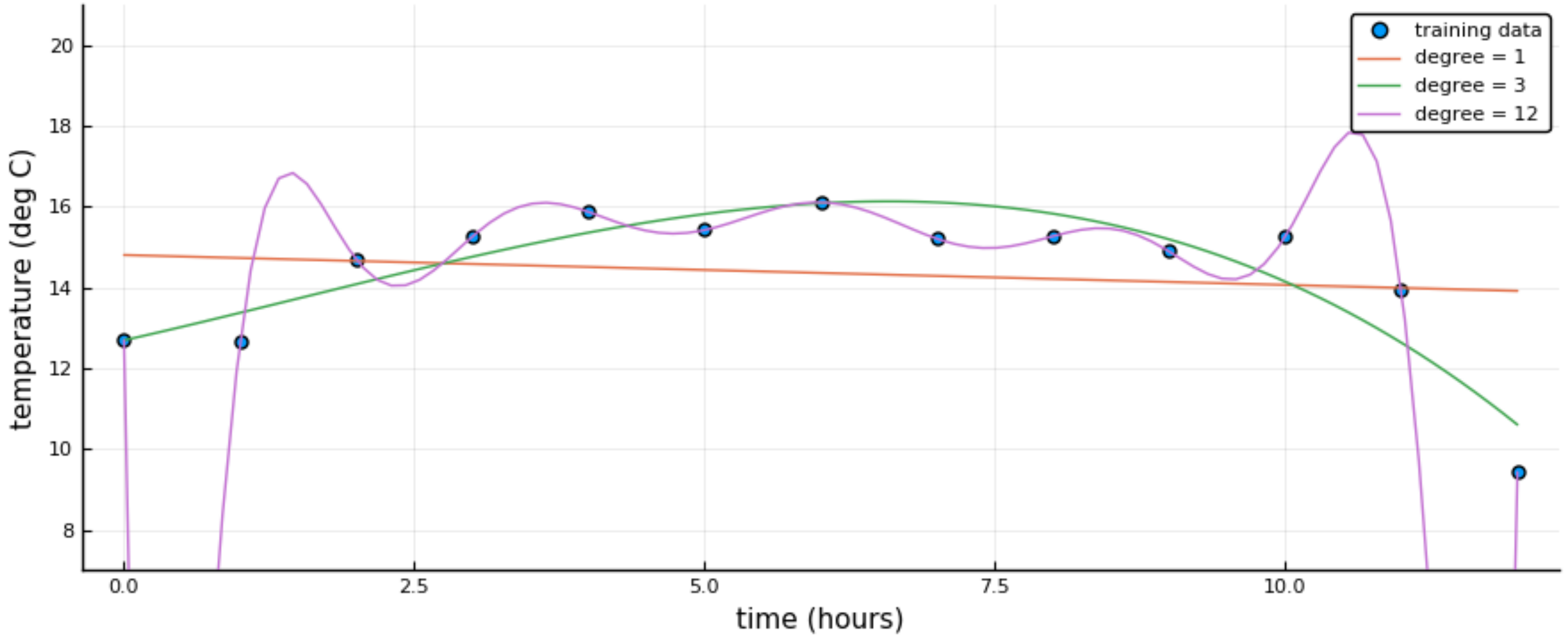
Supervised learning

You try fitting a cubic:



Supervised learning

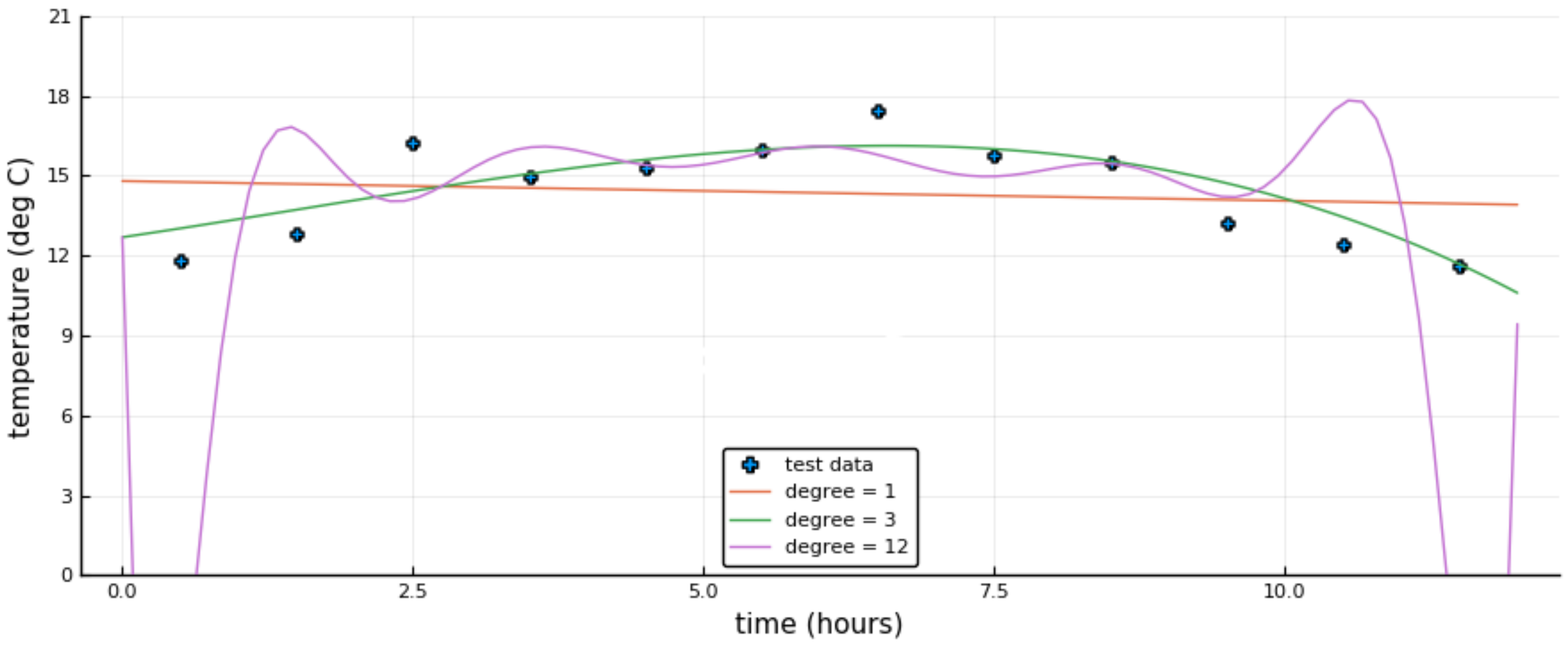
How about a 12th order polynomial?



Fits all the data perfectly!

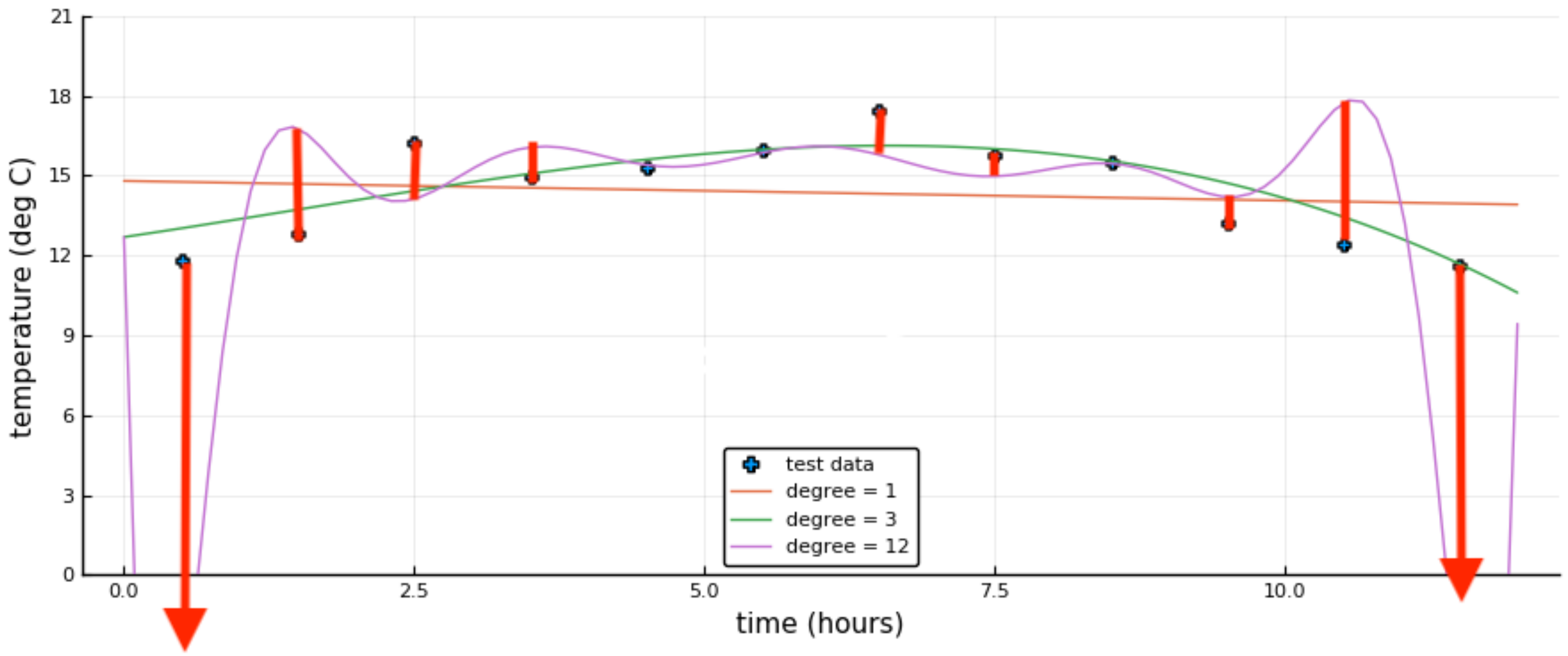
Performance evaluation

Lucky for me, I held back readings on the **half-hour** to test your skills.



Performance evaluation

Luck for me, I held back readings on the **half-hour** to test your skills.



Adding the lengths of the red lines, we get a **performance estimate** (mean absolute error).

What’s a model?

In MLJ, a **model** is the specification of a learning algorithm and its hyper-parameters (specified before learning begins).

Examples:

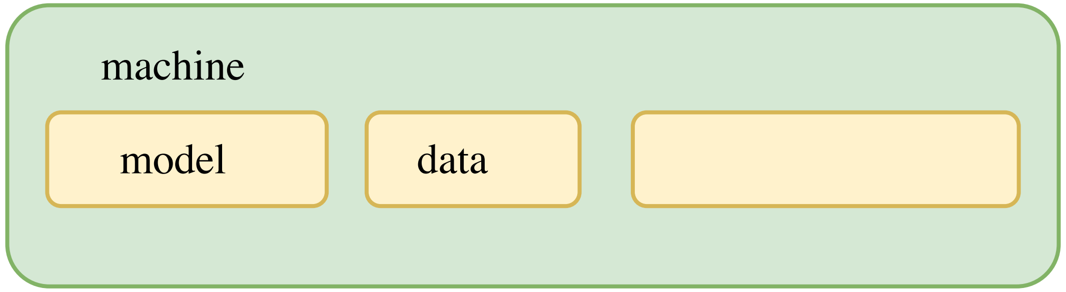
- Polynomial (by least squares) of degree 5.
- A decision tree classifier with maximum depth 3.
- A ridge regressor with L2 regularization of 0.1.

A **model** does *not* include learned parameters (**fitted_params** in MLJ docs).

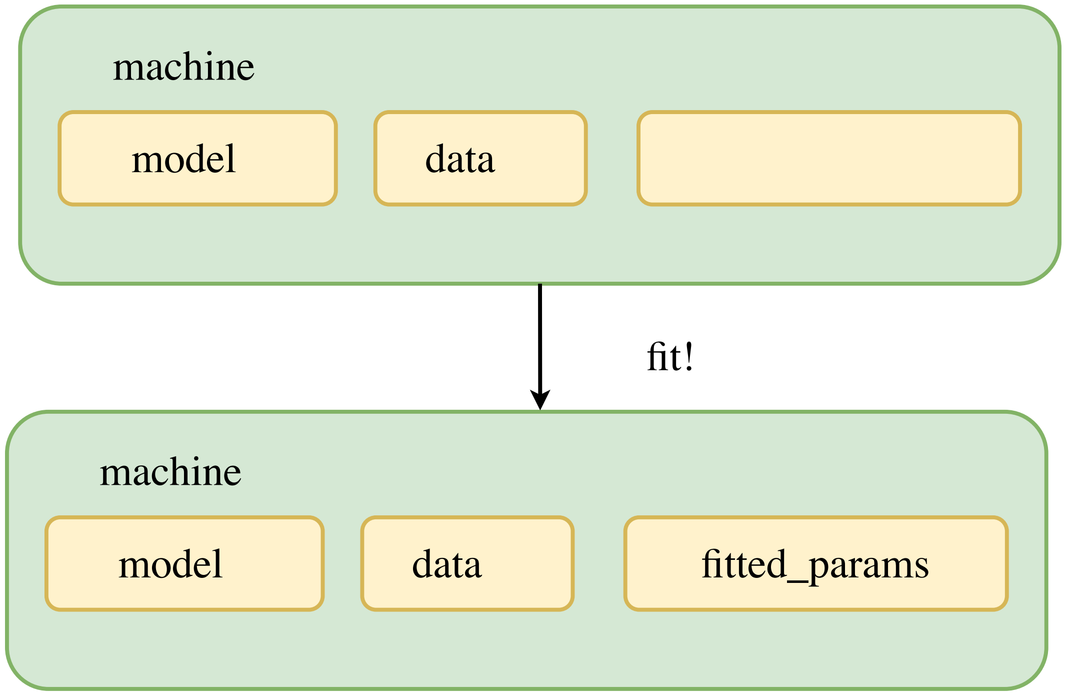
Warning: Outside of MLJ, a “model” might include learned parameters.

Machines

A machine starts life as a marriage of a **model** and **data**:

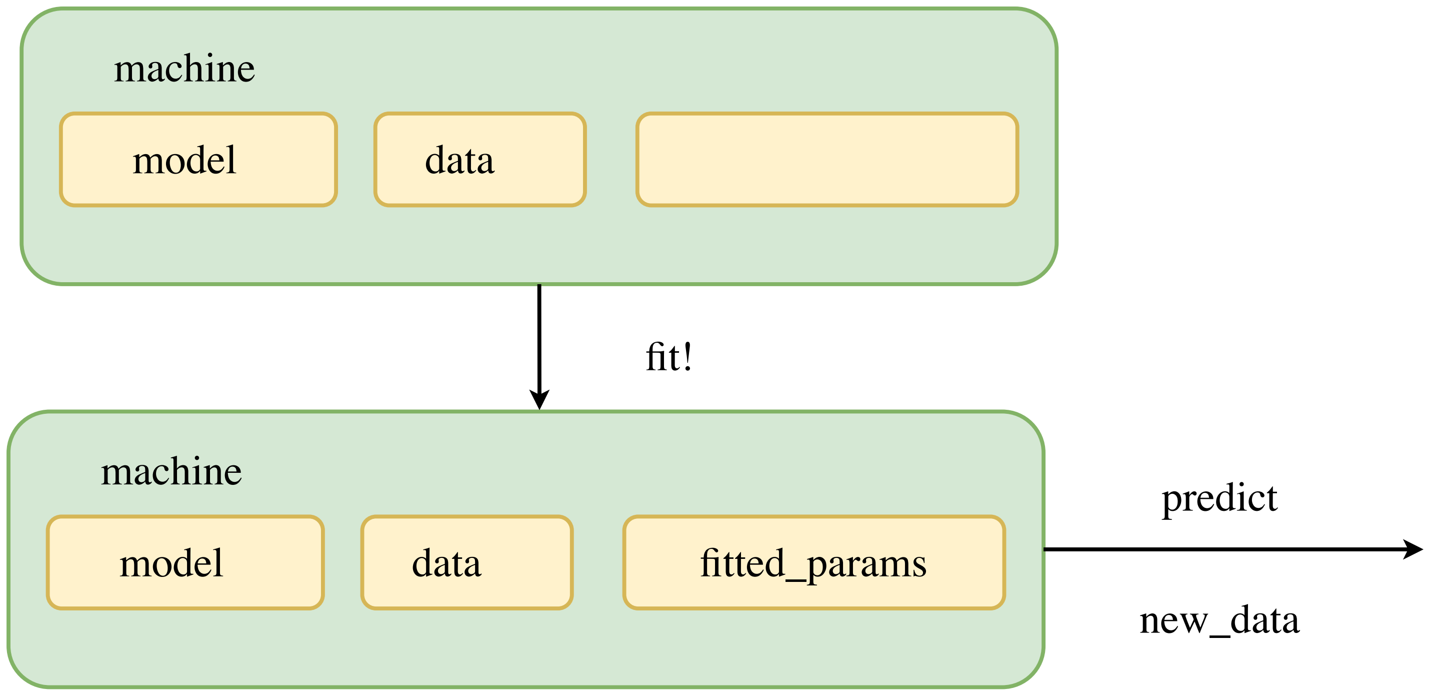


Life cycle of a machine

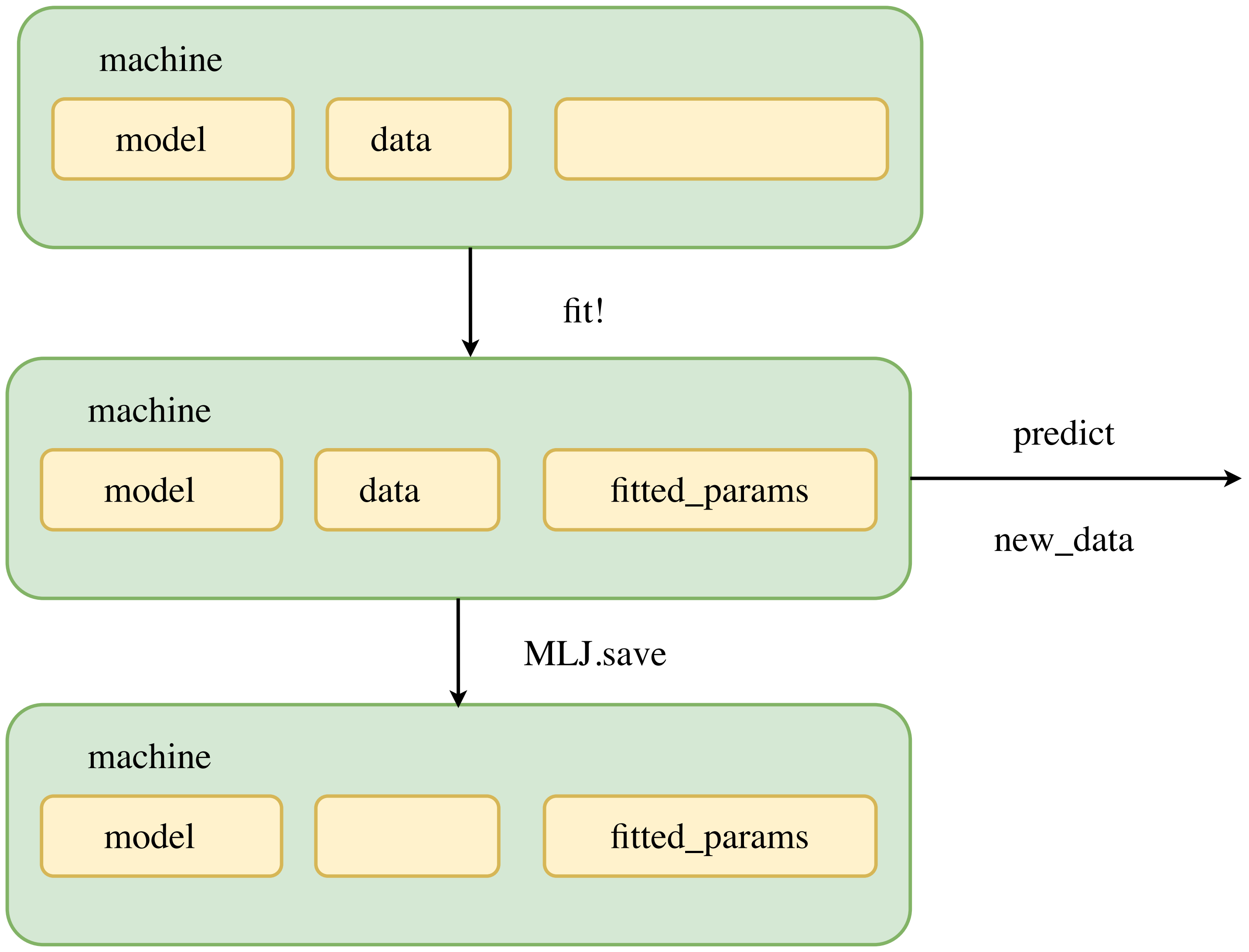


Aside: In `fit` we can specify `rows=...` to say which `data` observations to train on. So `data` could be *all* available training data. Or, you might exclude a holdout test set.

Life cycle of a machine



Life cycle of a machine



Why machines?

The machine syntax anticipates an advanced MLJ feature called **learning networks**, in which complex MLJ workflows are “exported” as new standalone models.

Learning networks are outside of the scope of this lesson.

Live coding demonstration

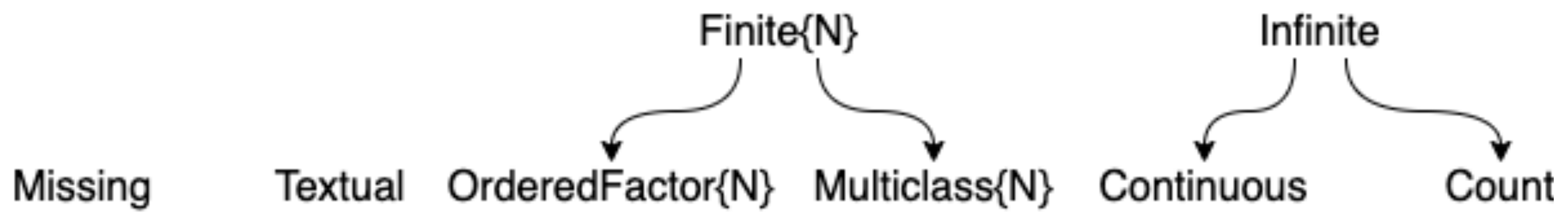
We now demonstrate a **regression** task in which we:

- Split data into `X` and `y`
- Split observation indices into `test` and `train`
- Choose a regression model
- Estimate the performance by hand
- Use a shortcut
- Estimate using cross-validation

Scientific types

Machine types: indicate how data is represented on the machine: `Float64`, `String`, etc.

Scitypes: indicate how data will be *interpreted*:



MLJ provides:

- `scitype(object)`: to see how MLJ models will interpret `object`, as it is currently encoded (use `schema` instead for tables)
- `coerce`: to change the machine encoding to match the desired interpretation (scitype)

More live coding

We now demonstrate a **classification** task in which we:

- Coerce data scitypes to ensure correct interpretation by MLJ
- Horizontally and vertically split data as before
- Demonstrate **one-hot encoding** to get `Continuous` input scitypes
- Estimate performance by hand and with `evaluate`.