PHYS 139/239: Machine Learning in Physics

Lecture 1: Introduction

Javier Duarte — January 10, 2023

Welcome to PHYS 139/239

- Fill out the pre-course survey: https://forms.gle/GPLwE5QKeYApiui4A
- Let's review the syllabus:
  - jduarte.physcis.ucsd.edu/phys139_239/syllabus.pdf
- Instructor: Javier Duarte (jduarte@ucsd.edu), office hours: TuTh 2:00-3:00pm (right after class) in MHA 5513 and on Zoom
- TA: Xiaoche Wang (xiw067@ucsd.edu), Office hours TBD
- Learning outcomes:
  - Find, explore, select, and preprocess scientific data
  - Choose and design machine learning models
  - Evaluate model performance and compare to standard benchmarks
  - Debug machine learning workflows
  - Relate model inputs and outputs to underlying physics concepts
  - Collaborate with peers to tackle complex, realistic problems
  - Present findings

Assignment breakdown

- 50% Homework
- 10% Participation in class/via Slack and completion of exit tickets
- 20% Midterm: Written proposal for group project
- 20% Final: Written group project summary, presentation, self-evaluation, and code

Homework

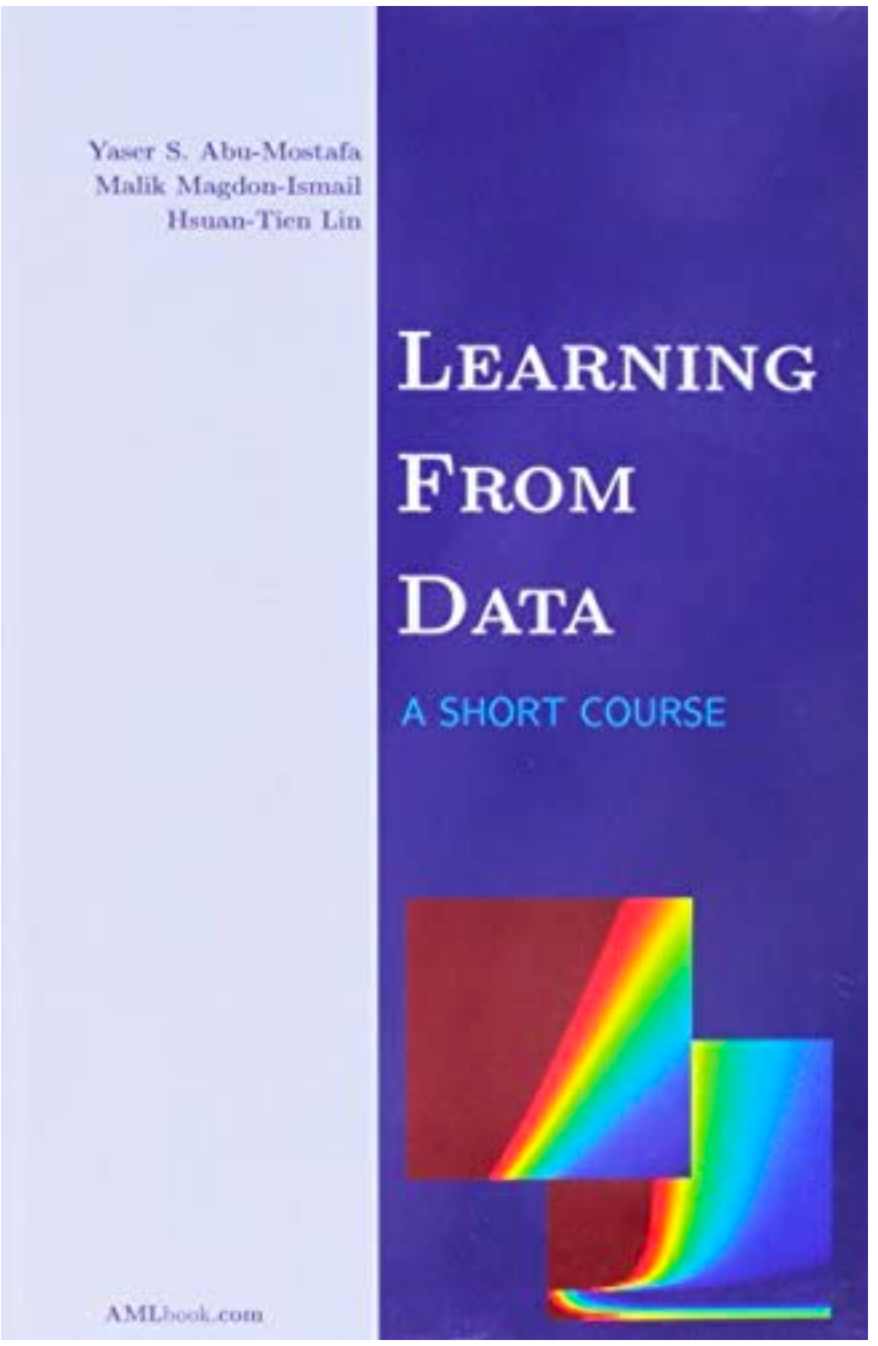
- Half of grade will be from turning in draft Fridays at 5:00pm
  - Graded on effort (on all problems)
  - Solution released shortly afterward
- Half of grade will be from turning in complete/revised solution Wednesdays at 5:00pm
  - Graded on correctness and effort (on all problems)
- Report (pdf file) uploaded to Gradescope
- Code (zip file) uploaded to Canvas
- First homework will be released tomorrow, Wednesday 1/11

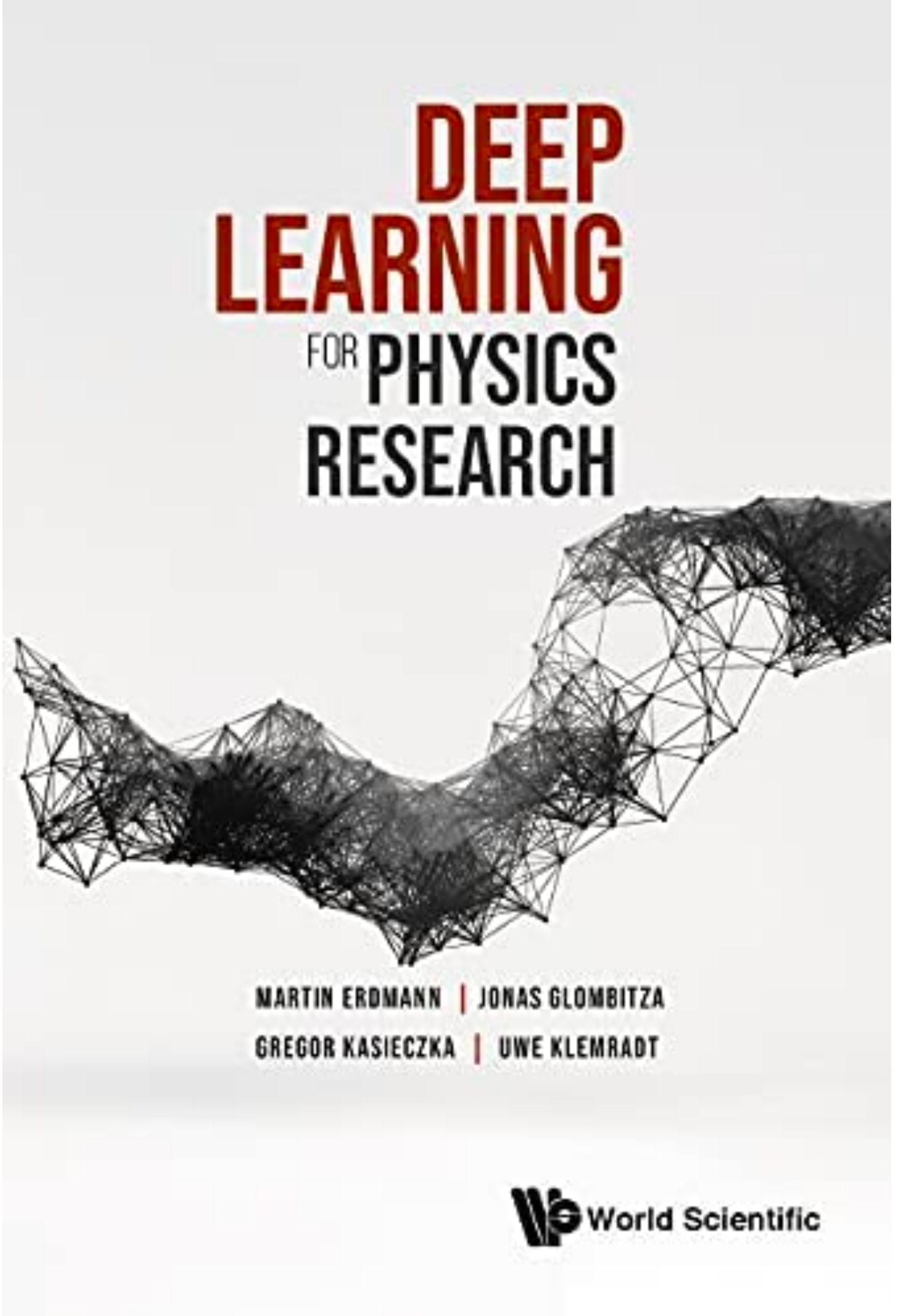
Midterm + final project

- Final project (due Finals Week): Reproduce or extend an existing, published ML in physics paper in groups of ~4
    - Some suggested articles and datasets found in the syllabus
    - But feel free to get creative!
    - Deliverables:
      (1) 4-page paper describing methods and results,
      (2) code (in public GitHub repository),
      (3) 20-minute presentation delivered by group during finals week, and
      (4) self and peer evaluations for group contributions
- Midterm (due Week 7): 2-page project proposal for instructors to check and make sure it's feasible, etc.,

Recommended reading

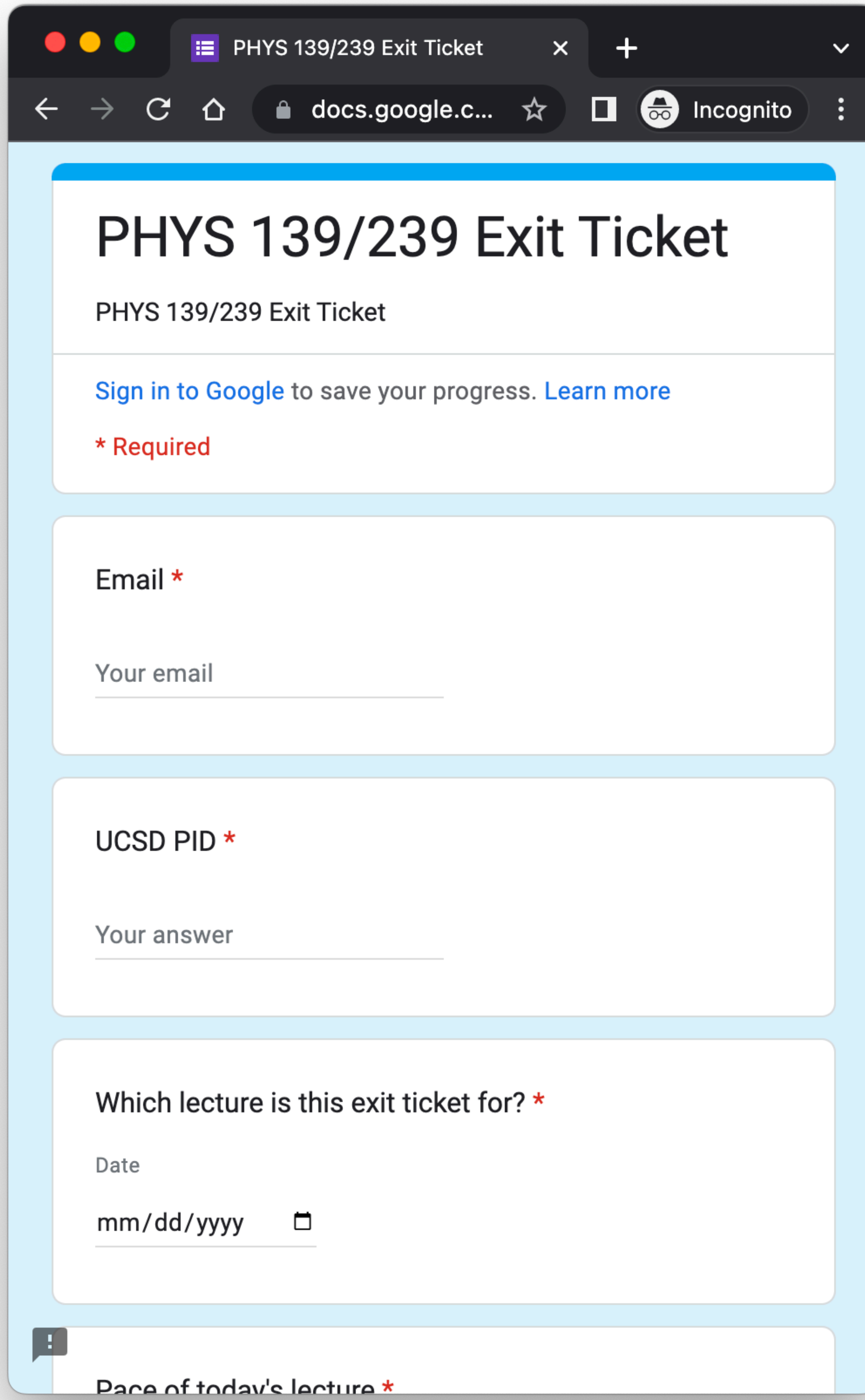
- No required textbook, but if you're having trouble following lectures, or haven't seen some of the introductory material before, there are some recommended (many free!) textbooks are in the syllabus
  - For early lectures, recommend: amlbook.com
  - For hands-on Keras-based portions, recommend: deeplearningphysics.org





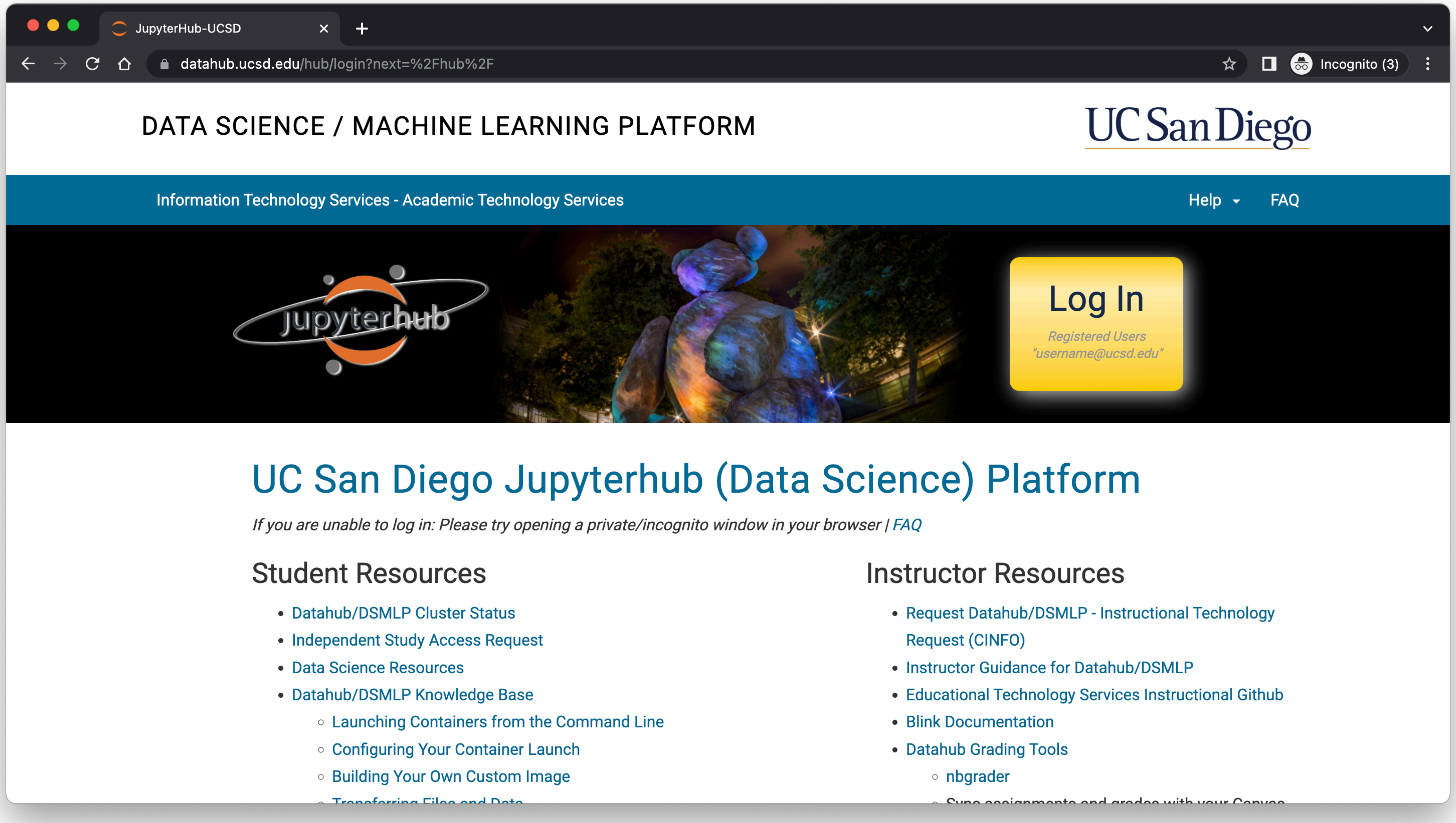
Exit tickets

- Exit tickets: https://forms.gle/4DmG5SjBUEM5pe6U8
  - Designed to see how you felt about the lecture, what you took away, whether you have any further questions or feedback
  - Filling it out will go toward the 10% participation score



DataHub

- We will use DataHub for in-class hands-on portions
  - Recommend to use it for homework, final project, etc.
- Address: datahub.ucsd.edu
- Similar to public, free services Google Colab, but with access to better CPUs and GPUs and run by UCSD
- Provides a “Jupyter notebook” interface (Python-based but interactive coding like MATLAB/Mathematica)



Slack

- Join the Slack workspace for the course: https://join.slack.com/t/ucsdphys139/shared_invite/zt-110gwd4lx-pZBsltfcxhbOD5BV6afVDA

- Tutorial: https://slack.com/help/categories/360000049063

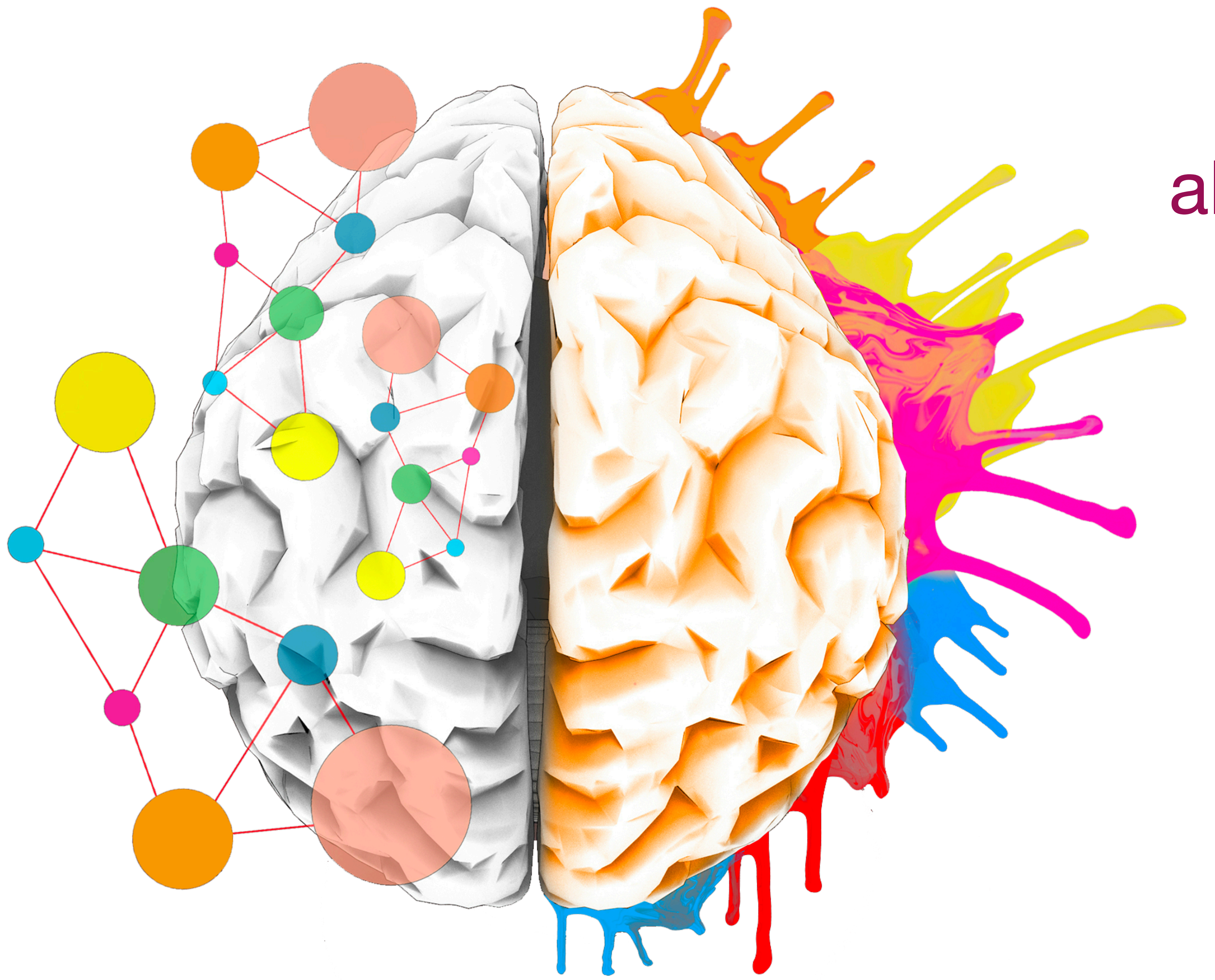
- Feel free to create channels to collaborate with others, etc.

Course overview

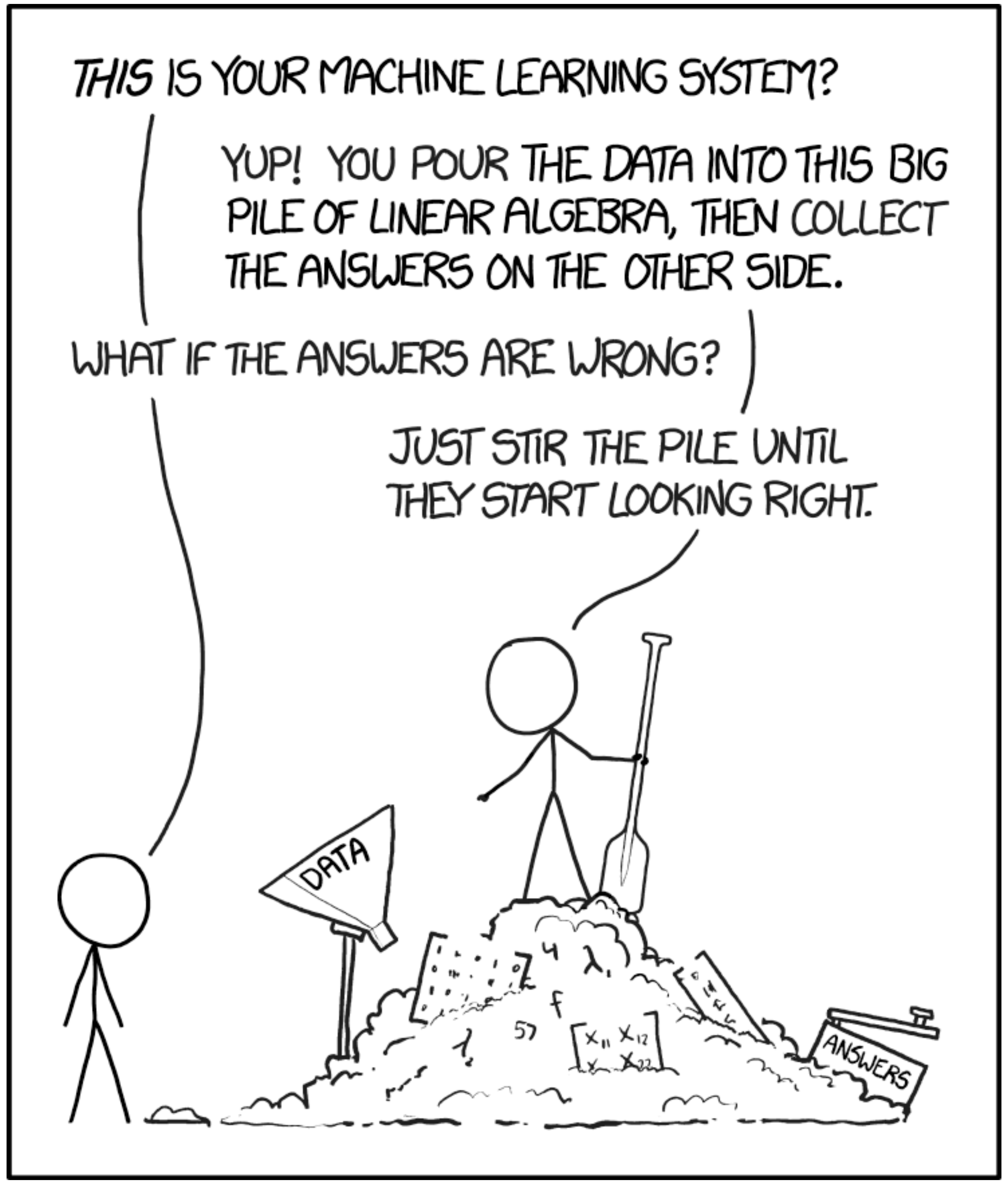
- Supervised learning
  - (Boosted) decision trees — tabular data
  - (Deep) neural networks — tabular data
  - Convolutional neural networks — image-like data
  - Graph neural networks — graph-like data and point clouds
- Unsupervised learning
  - (Variational) autoencoders for anomaly detection
- Model compression
- Special topics via guest lectures (TBC)
  - Equivariant models
  - Generative models
  - Reinforcement learning
  - Explainability
  - Uncertainty

What is machine learning?

- Science and art of learning automatically from data and experience



Also, a lot of calculus, linear algebra, statistics, group theory, …

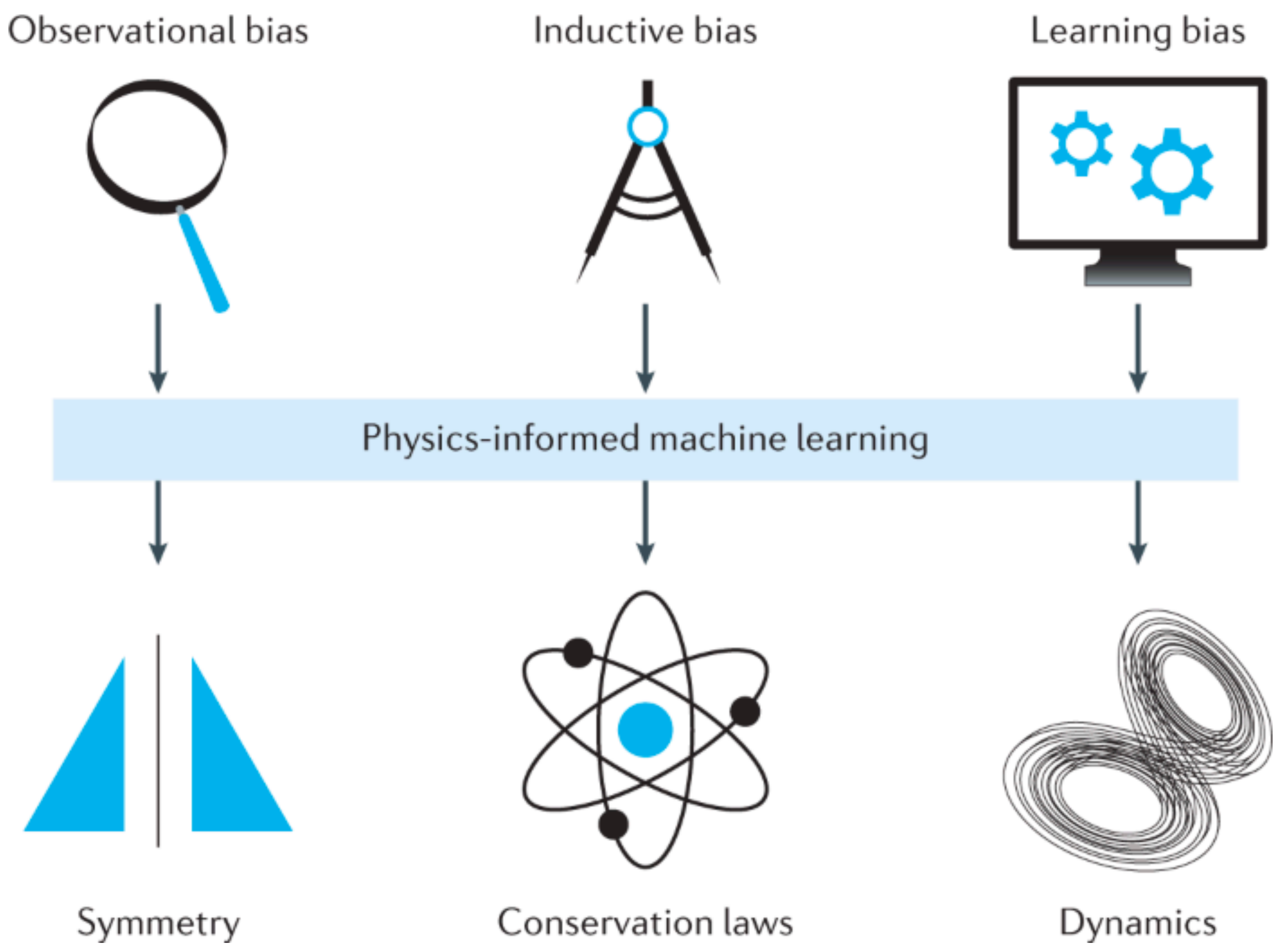


- Large overlap with data mining:
  - ML focuses on algorithms, DM on discovering patterns

Machine learning in physics

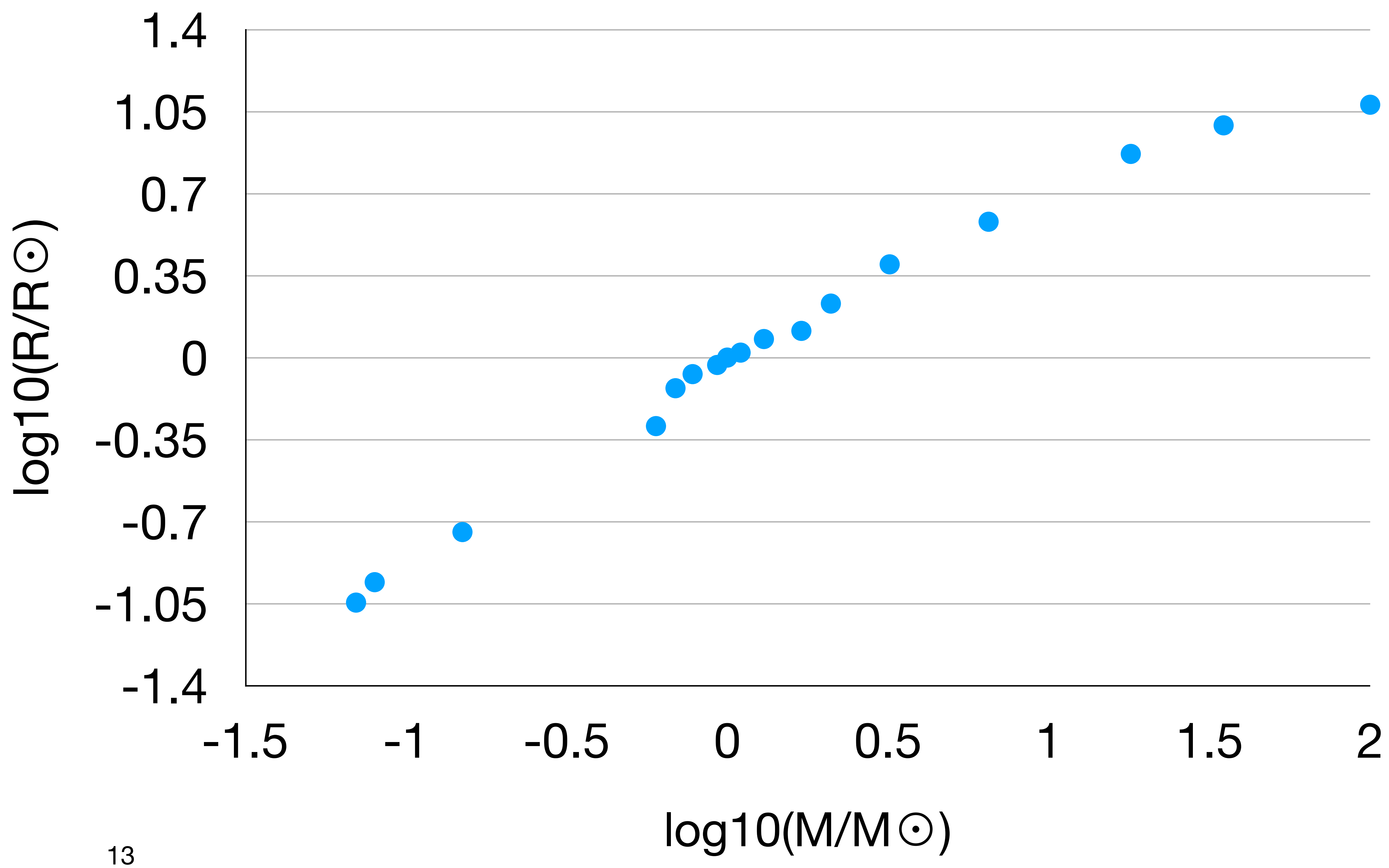
NeurIPS 2021 Tutorial:
neurips.cc/virtual/2021/tutorial/21896
Physics meets ML:
physicsmeetsml.org
NeurIPS ML4PS Workshop:
ml4physicalsciences.github.io
ICLR Physics4ML Workshop:
physics4ml.github.io

- Two interrelated themes
  - ML for physics research
    - ML applied to physics data, which may be unique or different from typical data used for ML
      - e.g. physics data can be “noisy” but in well characterized ways related to sensors
  - Physics for ML research
    - Physics-based algorithms, embedded symmetries, physical inductive bias
- Lots of overlapping ideas!



Supervised learning

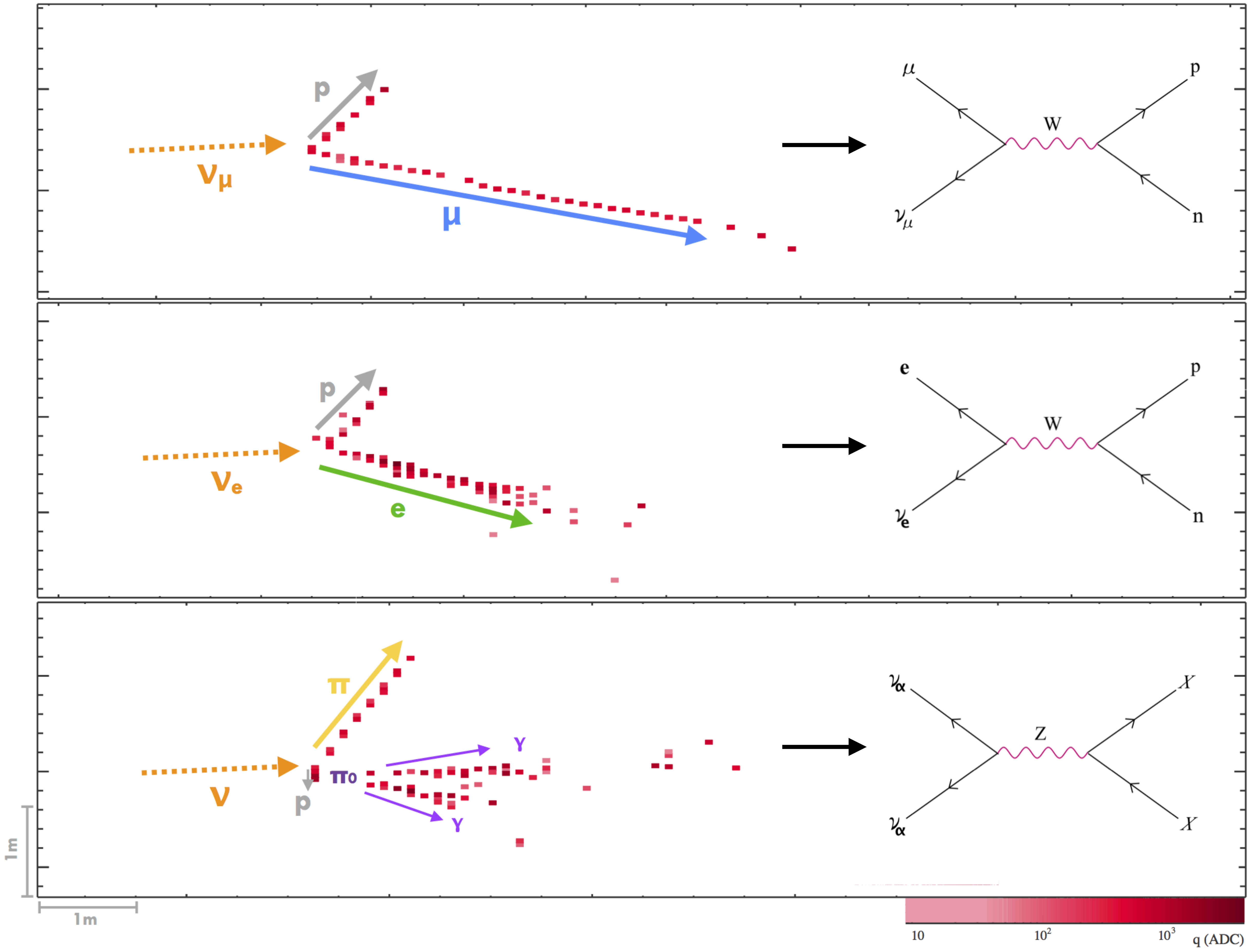
- Learn a function $f : X \to Y$ from an input space X (observations) to an output space Y (targets), using a set of labeled examples$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$.
- Example 1: Predict stellar radius given stellar mass



Supervised learning

- Learn a function $f : X \to Y$ from an input space X (observations) to an output space Y (targets), using a set of labeled examples$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$.

- Example 2: Classify images of neutrino interactions



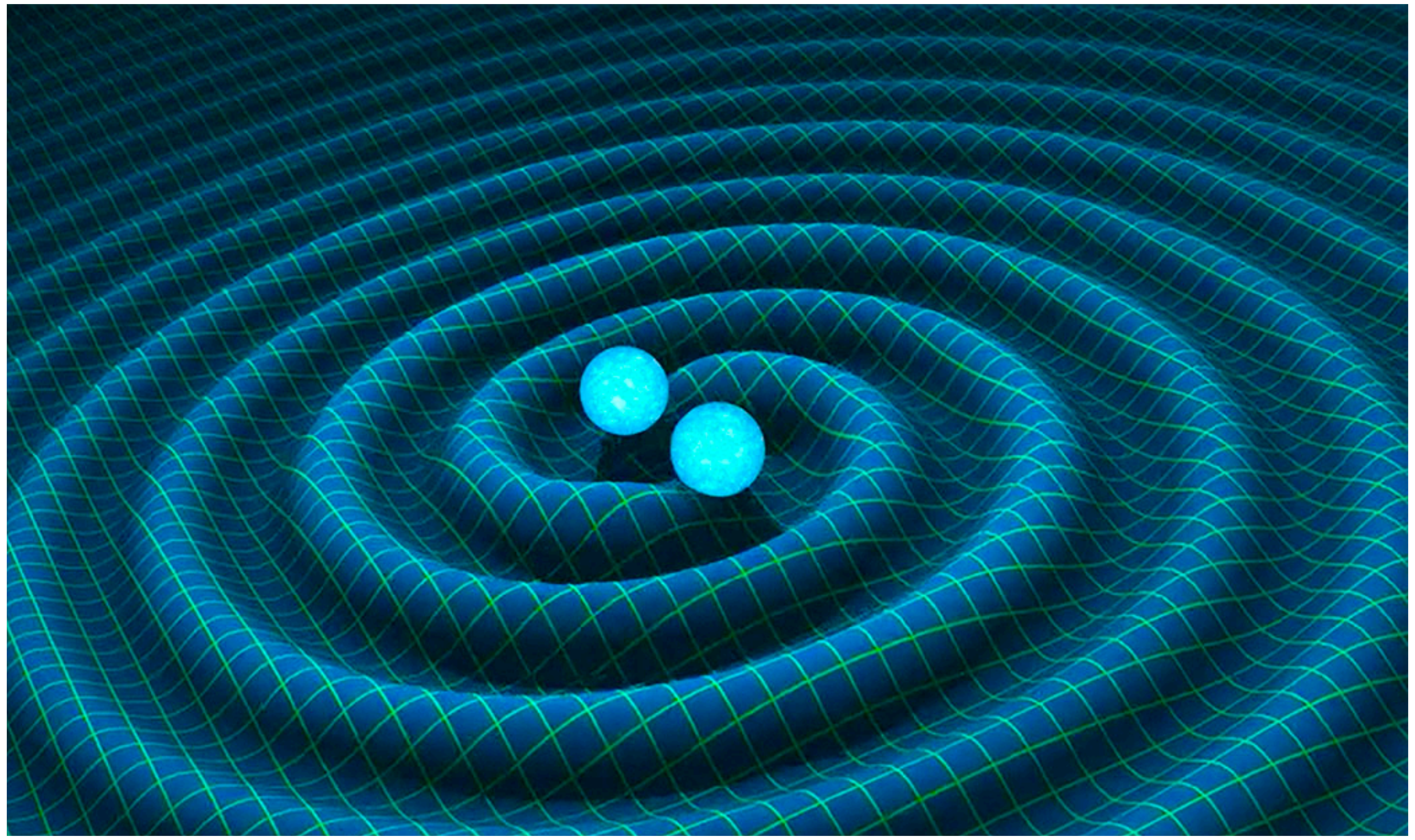
arXiv:1604.01444

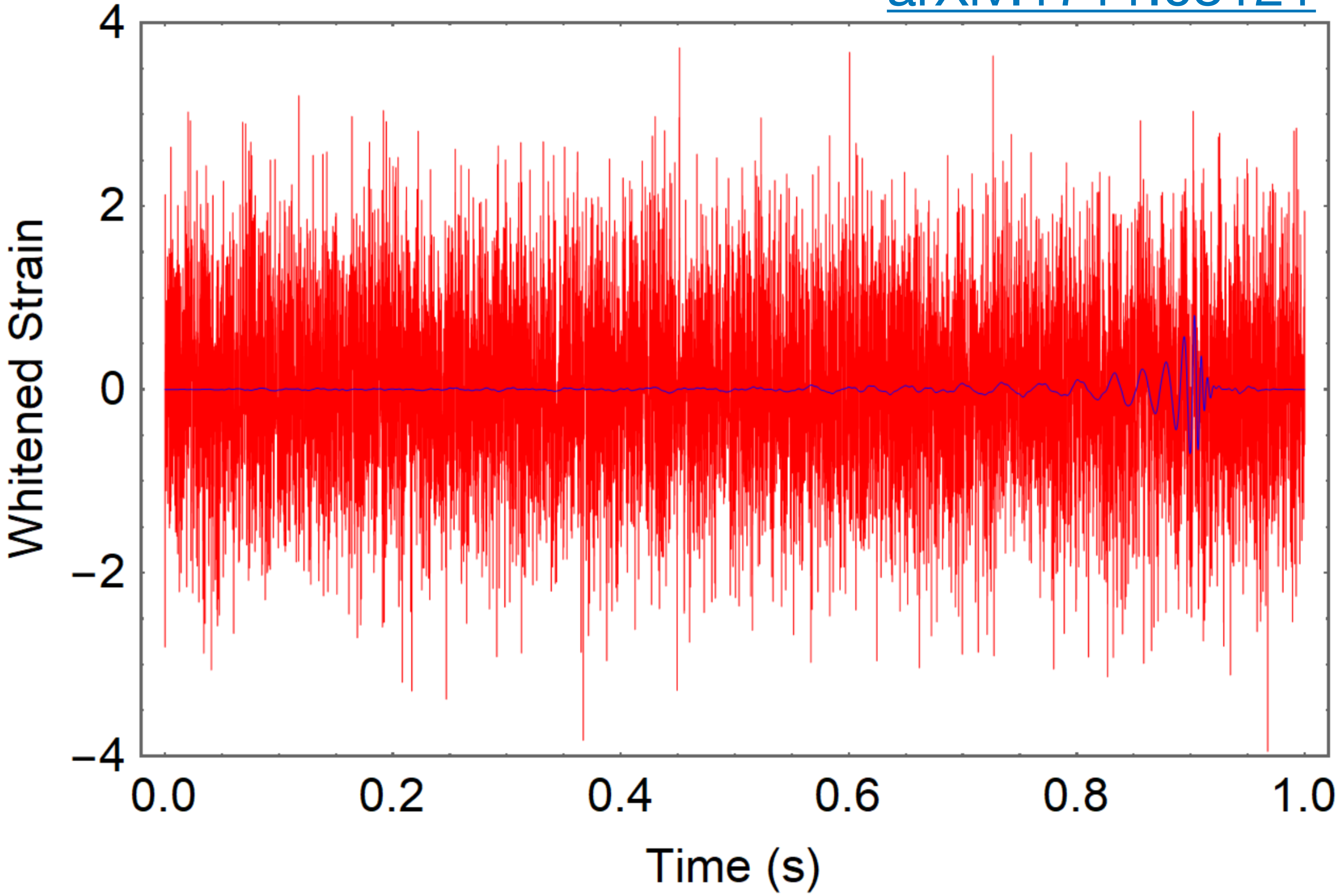
Supervised learning

- Learn a function $f : X \to Y$ from an input space X (observations) to an output space Y (targets), using a set of labeled examples$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$.

- Example 3: Reduce noise in a time-series trace to identify a gravitational wave signal

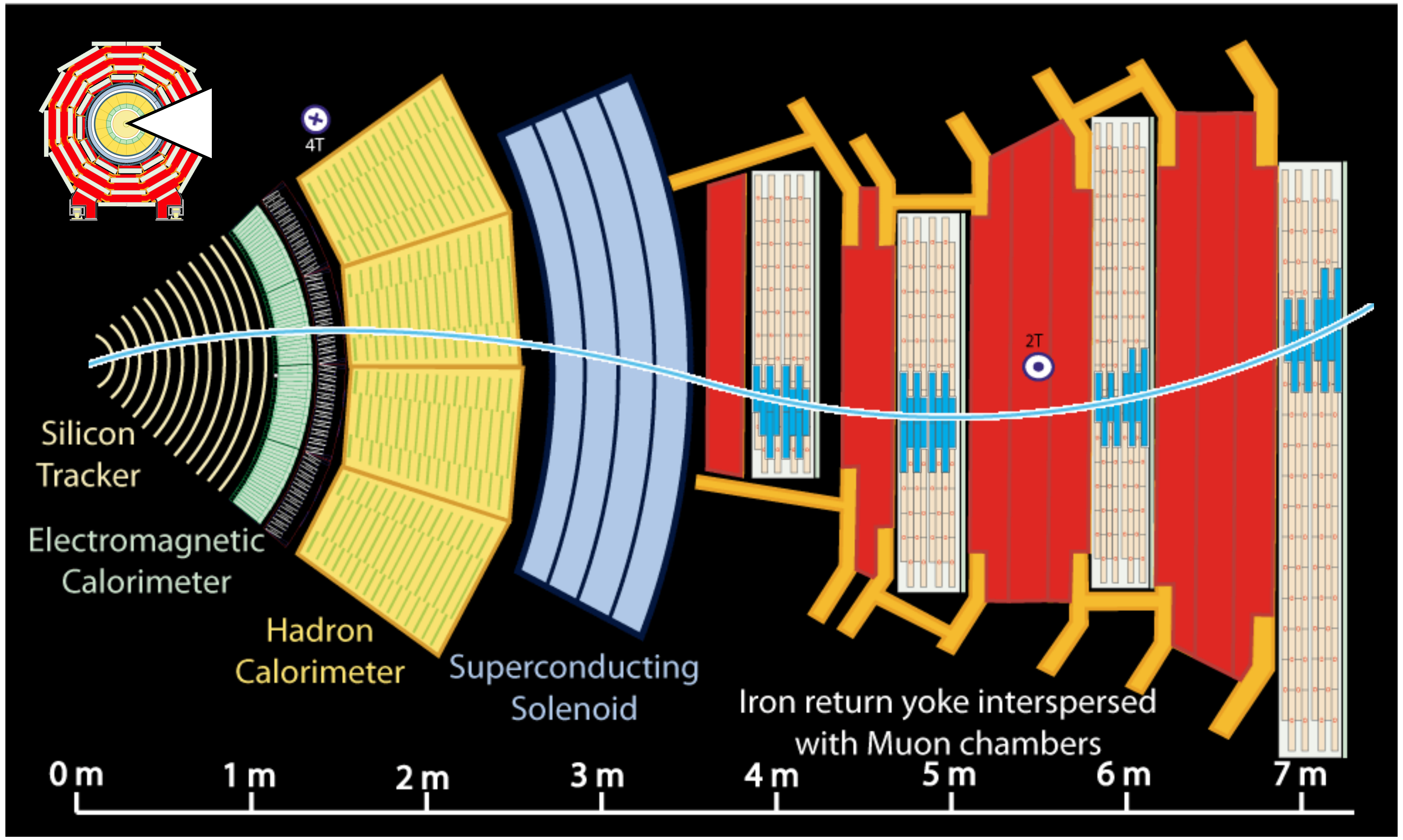
[arXiv:1711.03121](https://arxiv.org/abs/1711.03121)





Supervised learning

- Learn a function $f : X \to Y$ from an input space X (observations) to an output space Y (targets), using a set of labeled examples$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$.
- Example 4: Estimate particle momentum, charge, type, etc. from detector hits

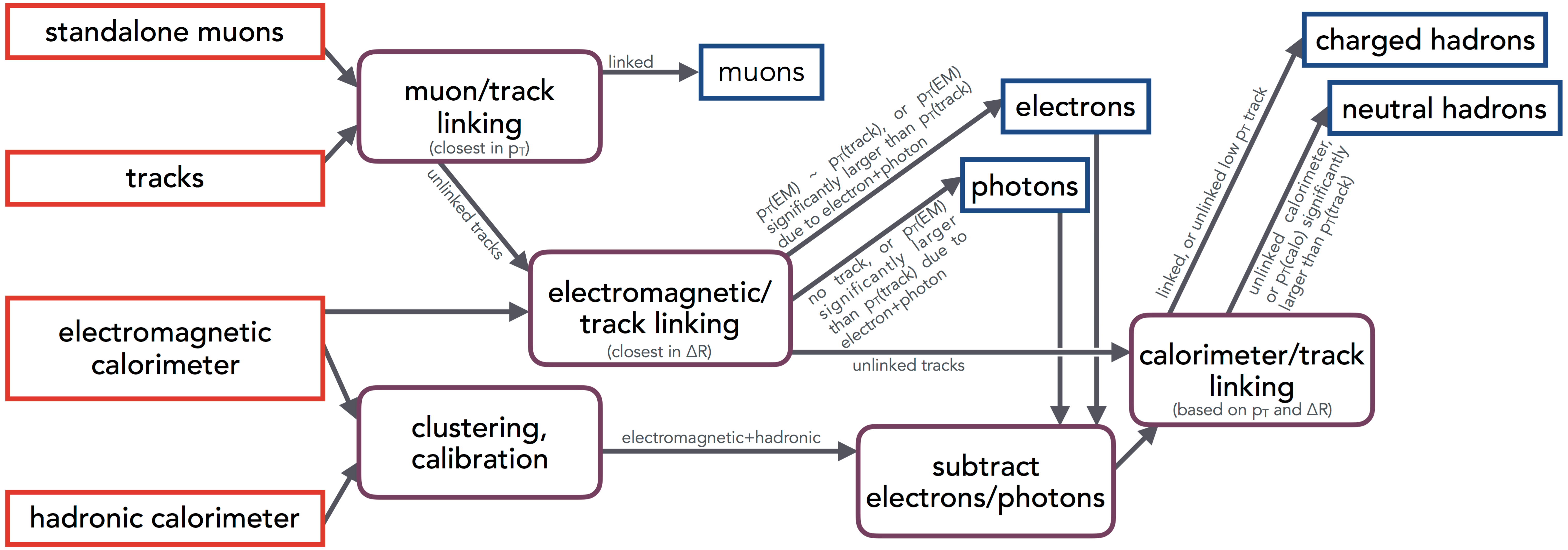


$$\to \begin{pmatrix} E \\ p_x \\ p_y \\ p_z \end{pmatrix}, q, \text{type}, p_{\text{pileup}}, \ldots$$

arXiv:2101.08578

Why are these problems hard?

- Expert-engineered solutions are
  - Complicated to write and maintain
  - Require decades of domain knowledge (physics, engineering, …)
    - But, they are interpretable/understandable (to experts)
  - Until recently, they are the standard (“baseline”) in physics experiments

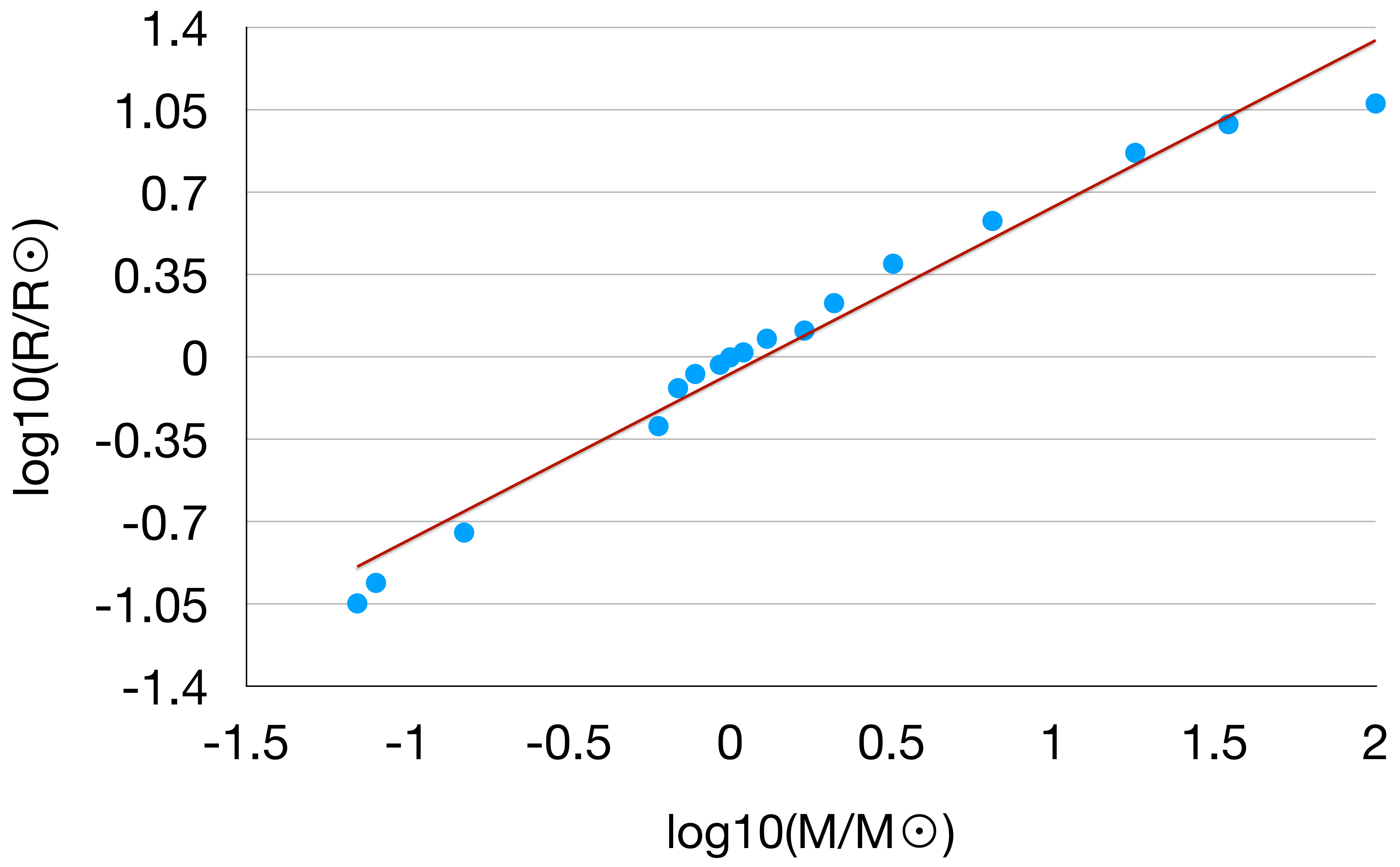


Highly simplified flow chart of particle reconstruction in CMS [arXiv:1808.02094]

- Problem might inherently require data (variations in detectors or over time, etc.)

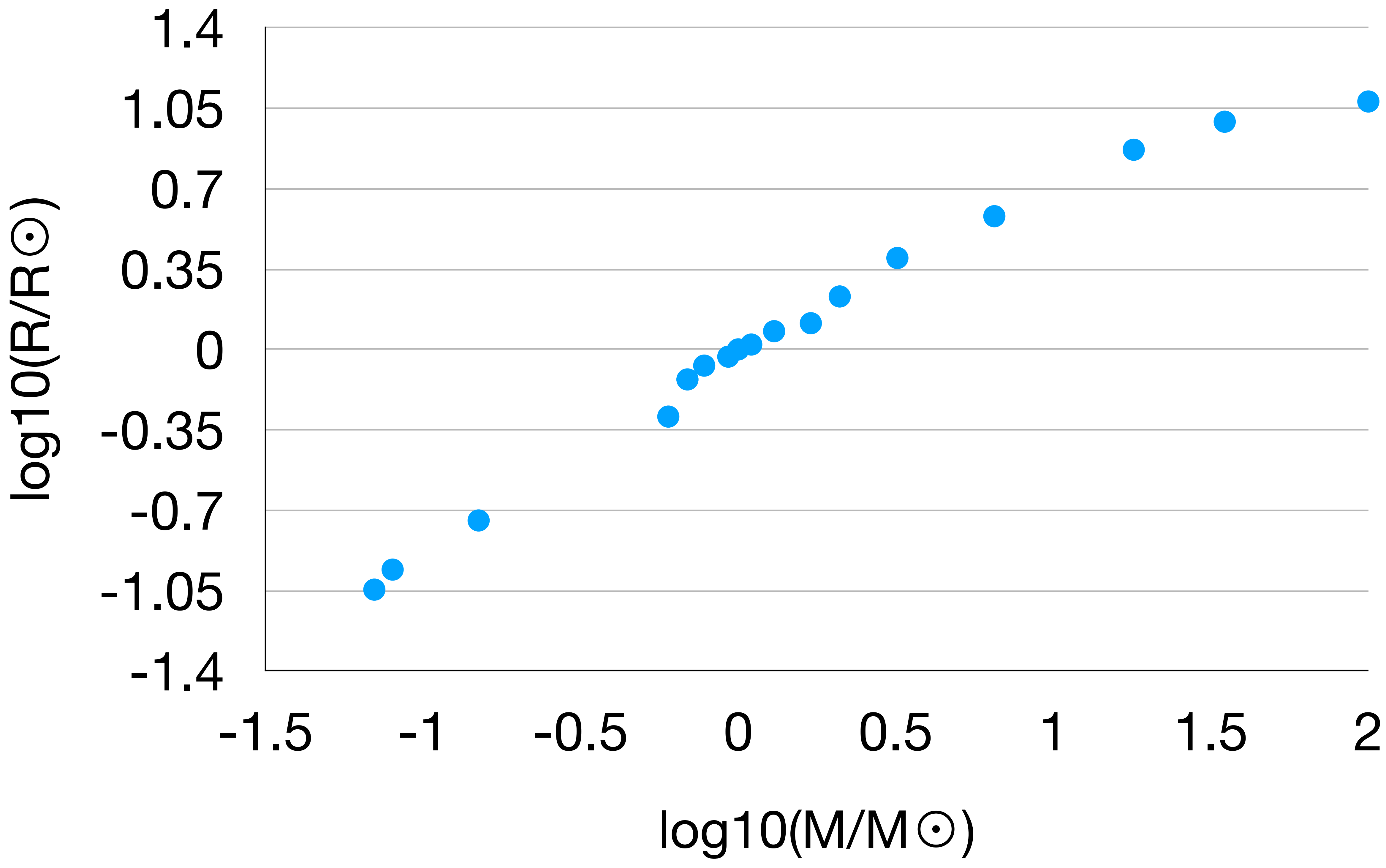
Machine learning as an alternative approach

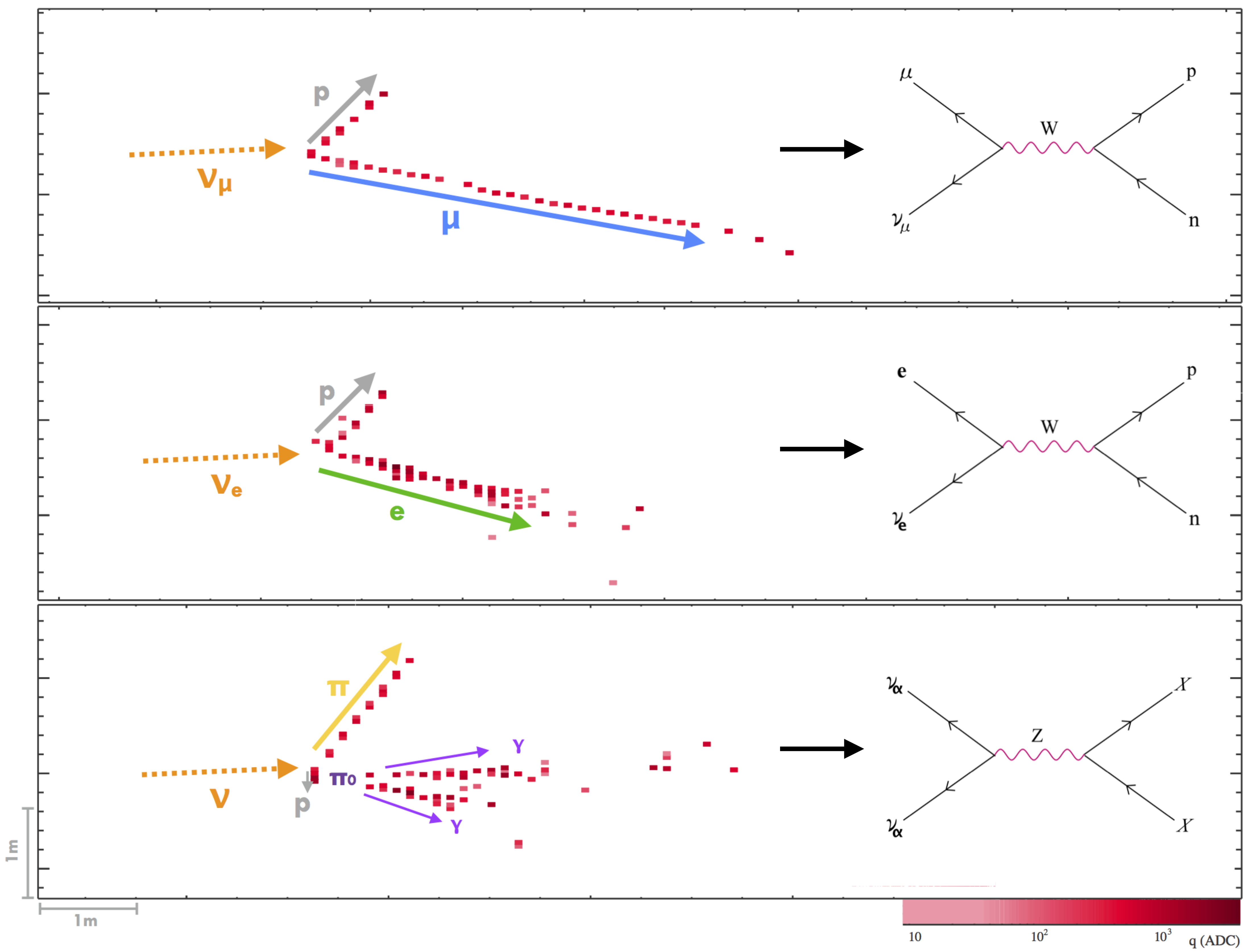
- Collect a **labeled** training set (supervision)
  - Often requires **simulation** where the "ground truth" is known



- Train a model using a learning algorithm (find patterns in the data)

Types of supervised learning algorithms

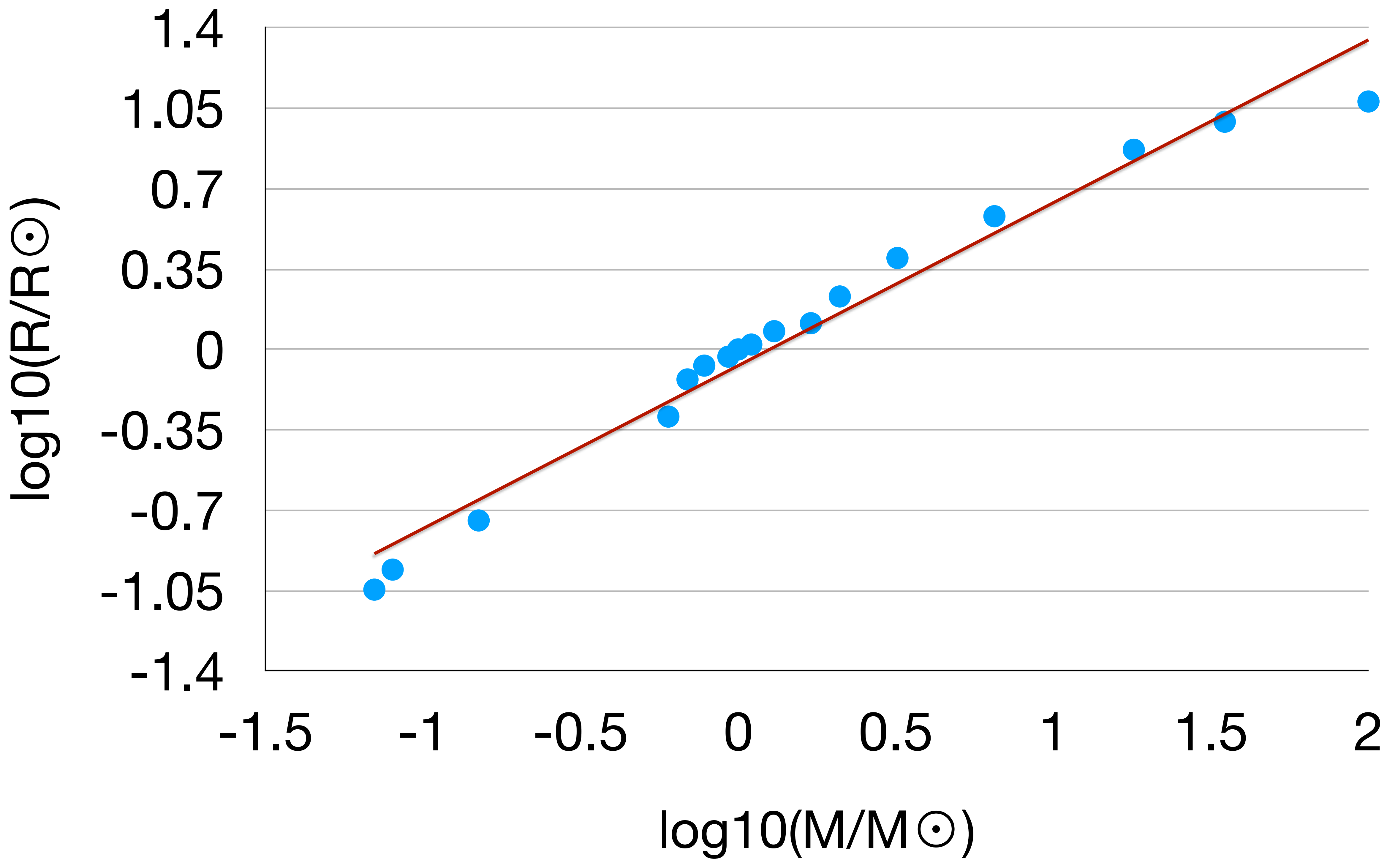




- Regression: predict real values $y \in Y = \mathbb{R}$ or $\mathbb{R}^n$
- + more (e.g., object detection)
- Classification: predict a class $y \in Y = \{0,1,\ldots,n-1\}$ from a fixed finite set

Linear regression

$$\log_{10}(R/R_\odot) = w \log_{10}(M/M_\odot) + b$$



- Let's try to fit a straight line:

  $f(x \mid w, b) = wx + b$ (linear model)

- More generally, if $x \in \mathbb{R}^D$:

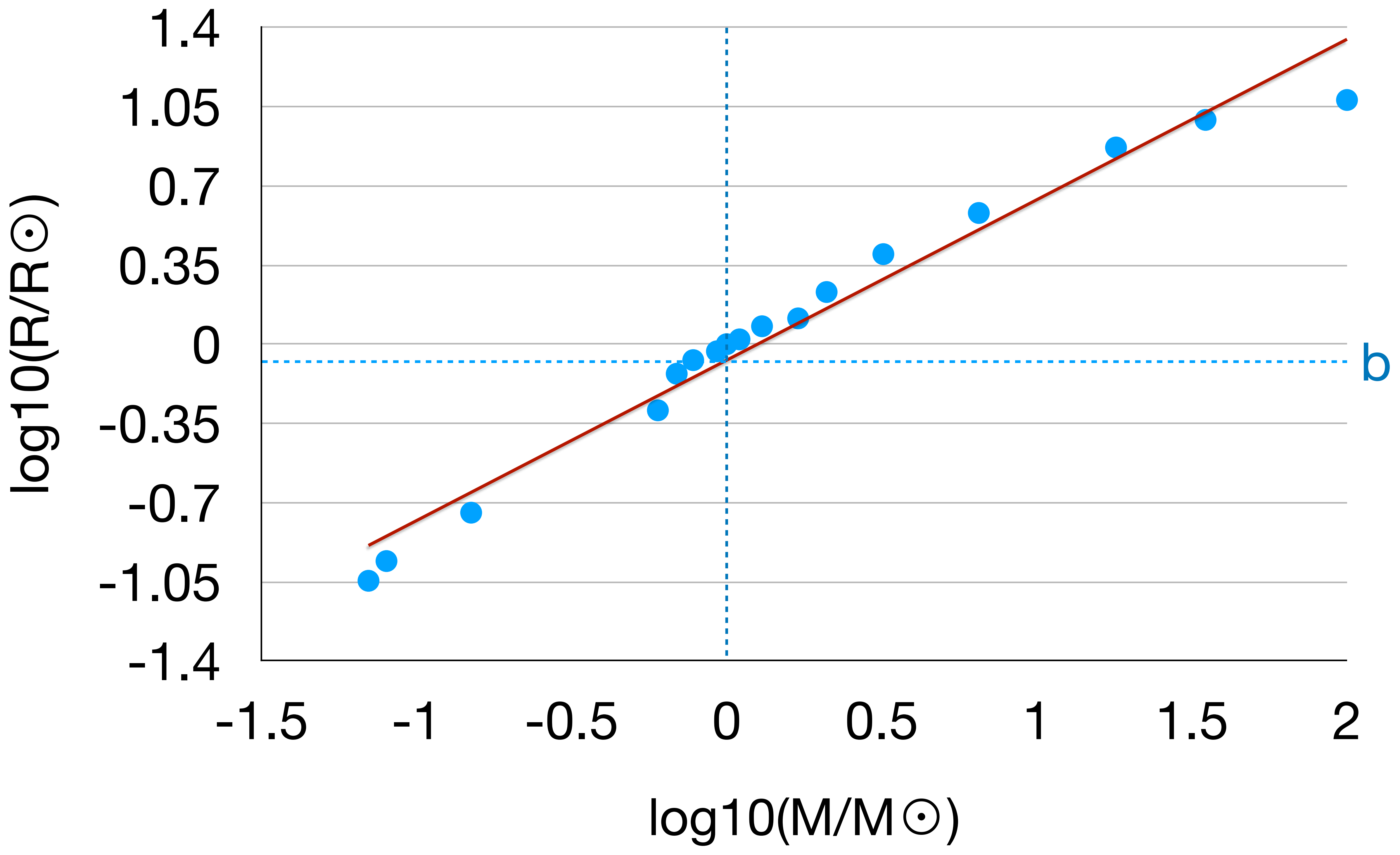
  $f(x \mid w, b) = w^\intercal x + b \quad (w \in \mathbb{R}^D)$

- Example: $x = (x^{(1)}, x^{(2)})$ where

  $x^{(1)}$ = mass and
  $x^{(2)}$ = luminosity of star

  $\Rightarrow f(x \mid w, b) = w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + b$

Linear regression



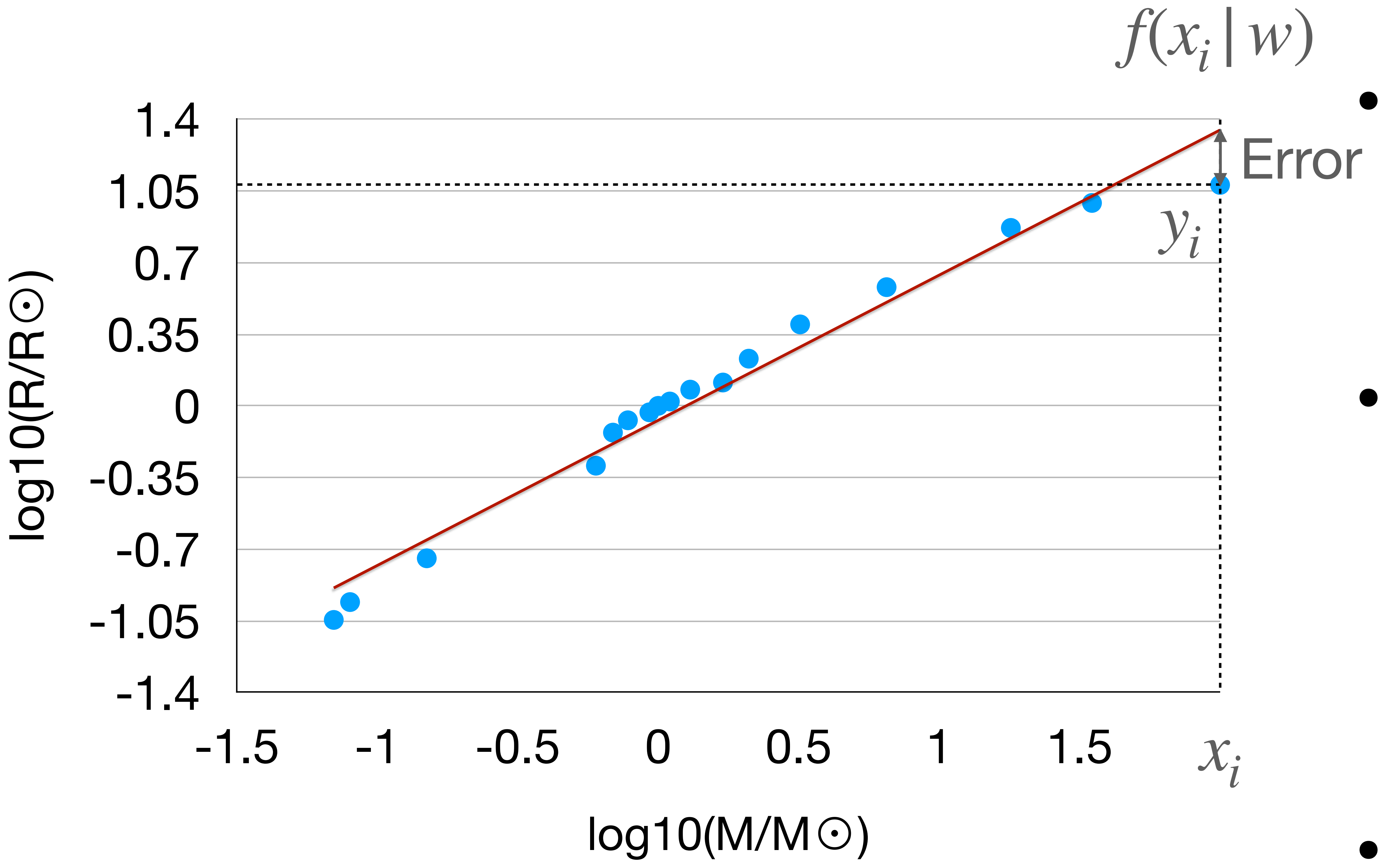
- Linear model (WLOG):

$$f(x \mid w) = w^\intercal x \quad (w \in \mathbb{R}^{D+1})$$

- We can add a “dummy feature” $x^{(0)} = 1$ to all input data x so that $w^{(0)}$ acts as bias:

$$f(x \mid w) = \underbrace{w^{(0)}}_{b} x^{(0)} + w^{(1)} x^{(1)} + \cdots + w^{(D)} x^{(D)}$$

Linear regression



- Linear model:

  $$f(x | w) = w^{\intercal} x \quad (w \in \mathbb{R}^{D+1})$$

- How do we select the parameters w?
  - We want $y_i \approx f(x_i | w)$

- Squared loss: $L(y, y') = (y - y')^2$

  (Least squares)

Learning objective: $\arg\min_w \sum_{i=1}^{N} L(y_i, f(x_i | w)) = \arg\min_w \sum_{i=1}^{N} (y_i - w^{\intercal} x_i)^2$

Optimizing the learning objective

Learning objective: $\arg\min_{w} \sum_{i=1}^{N} L(y_i, f(x_i \,|\, w)) = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^\intercal x_i)^2$

- Quadratic function of w can be minimized by setting the gradient equal to 0:

$$\frac{\partial}{\partial w^{(j)}} \sum_{i=1}^{N} (y_i - w^\intercal x_i)^2 = -2 \sum_{i=1}^{N} (y_i - w^\intercal w_i) x_i^{(j)} = 0$$

- Closed-form solution in terms of the "design matrix" $X_{ij} = x_i^{(j)}$ and the column vector Y consisting of the targets y_i:

$$w = (X^\intercal X)^{-1} X^\intercal Y$$

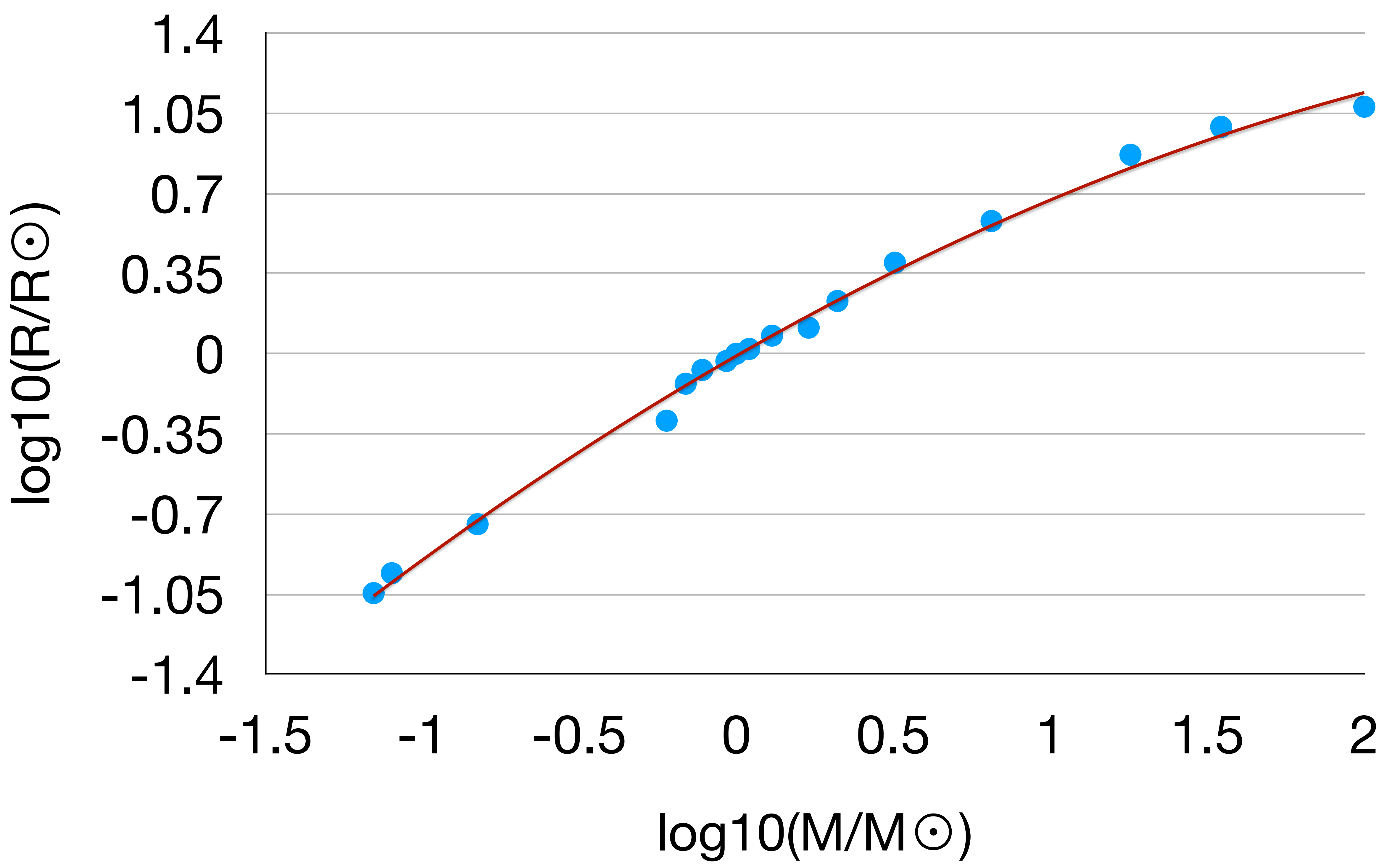
(but if the dataset is very large, then it may be not feasible to use this closed-form solution)

Getting more out of linear models

features / embedding of x

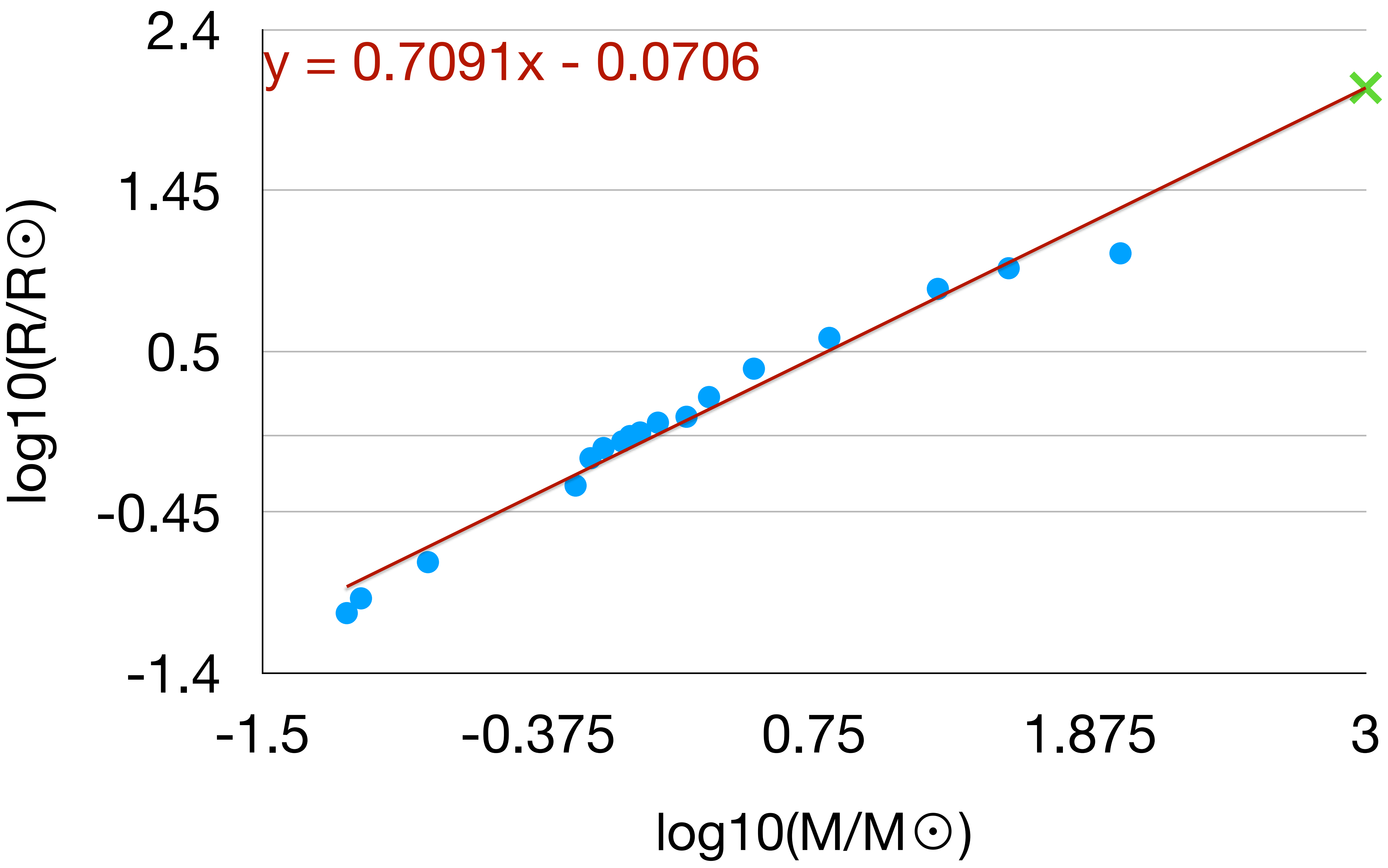
- Replace our input vector x with some $\phi(x)$ to make our model more expressive
- For example, if $\phi(x) = (1, x, x^2)$ then our model becomes:

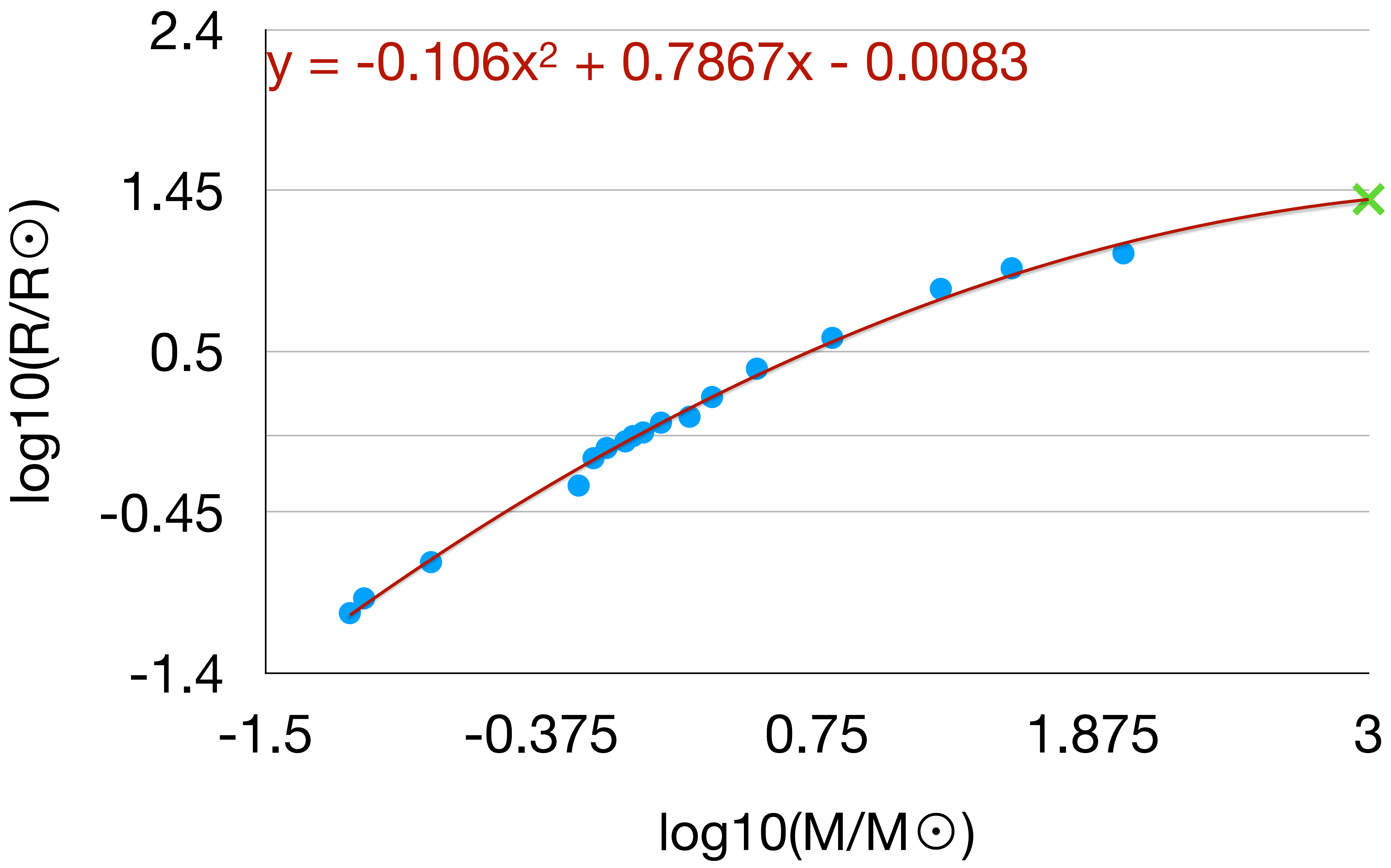
$$f(x \mid w) = w^\intercal \phi(x) = w_0 + w_1 x + w_2 x^2$$



- The model is still **linear** in the parameters w!
- More expressive than a line $w_0 + w_1 x$, so the fit is better (i.e., training error is lower)

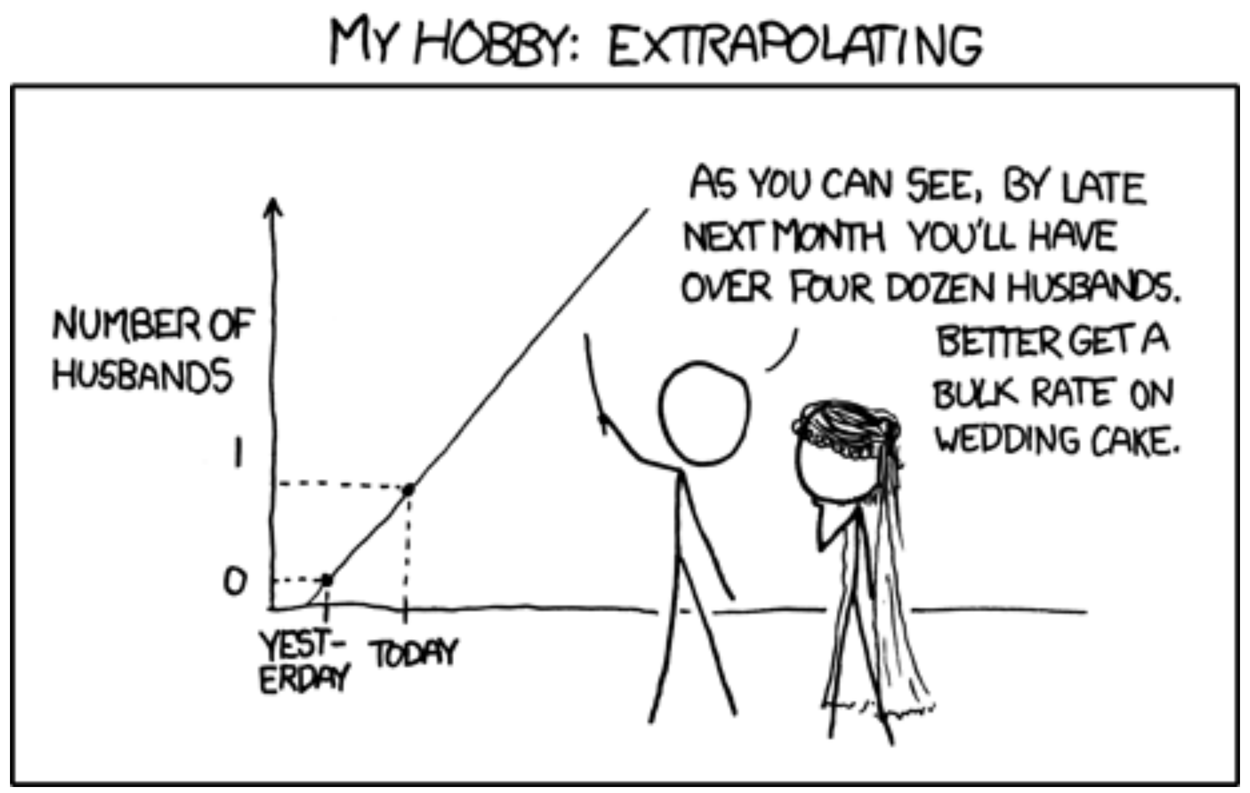
Different models extrapolate differently





- Both models fit the training data well

Extrapolation is very different!

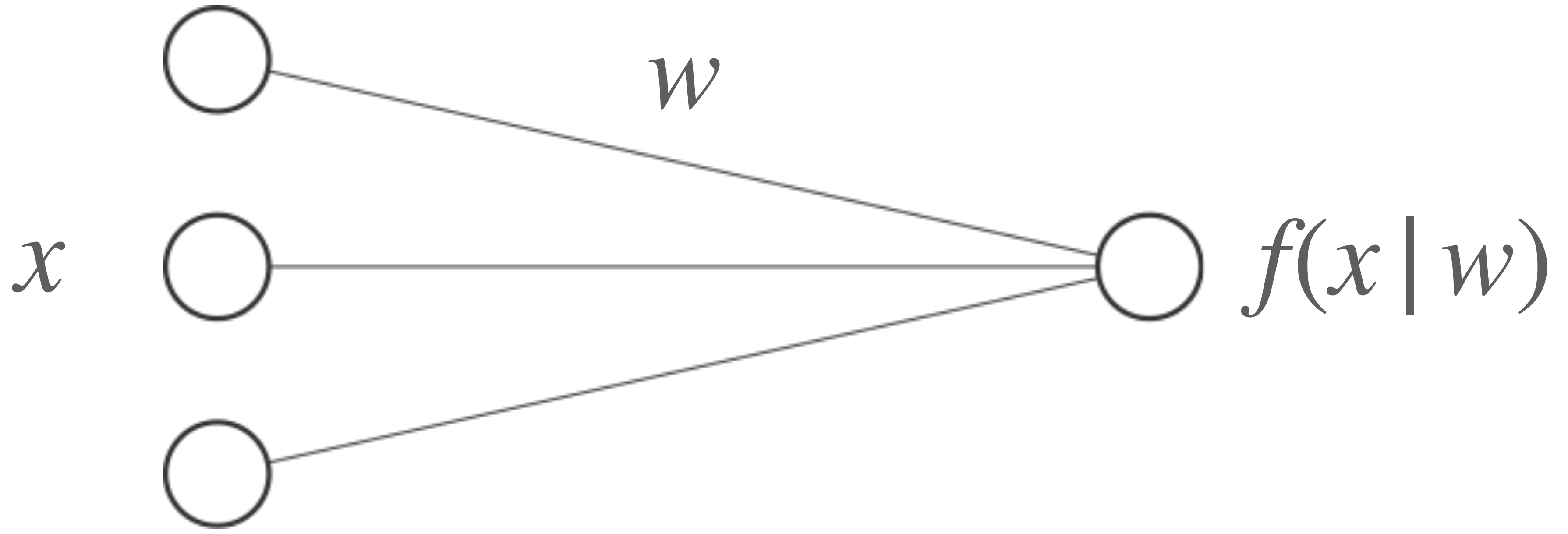


- What do they predict for a star 1,000 times more massive than the sun ($\log_{10}(M/M_\odot) = 3$)?
  - First model: $R = 114R_\odot$; Second model: $R = 25R_\odot$

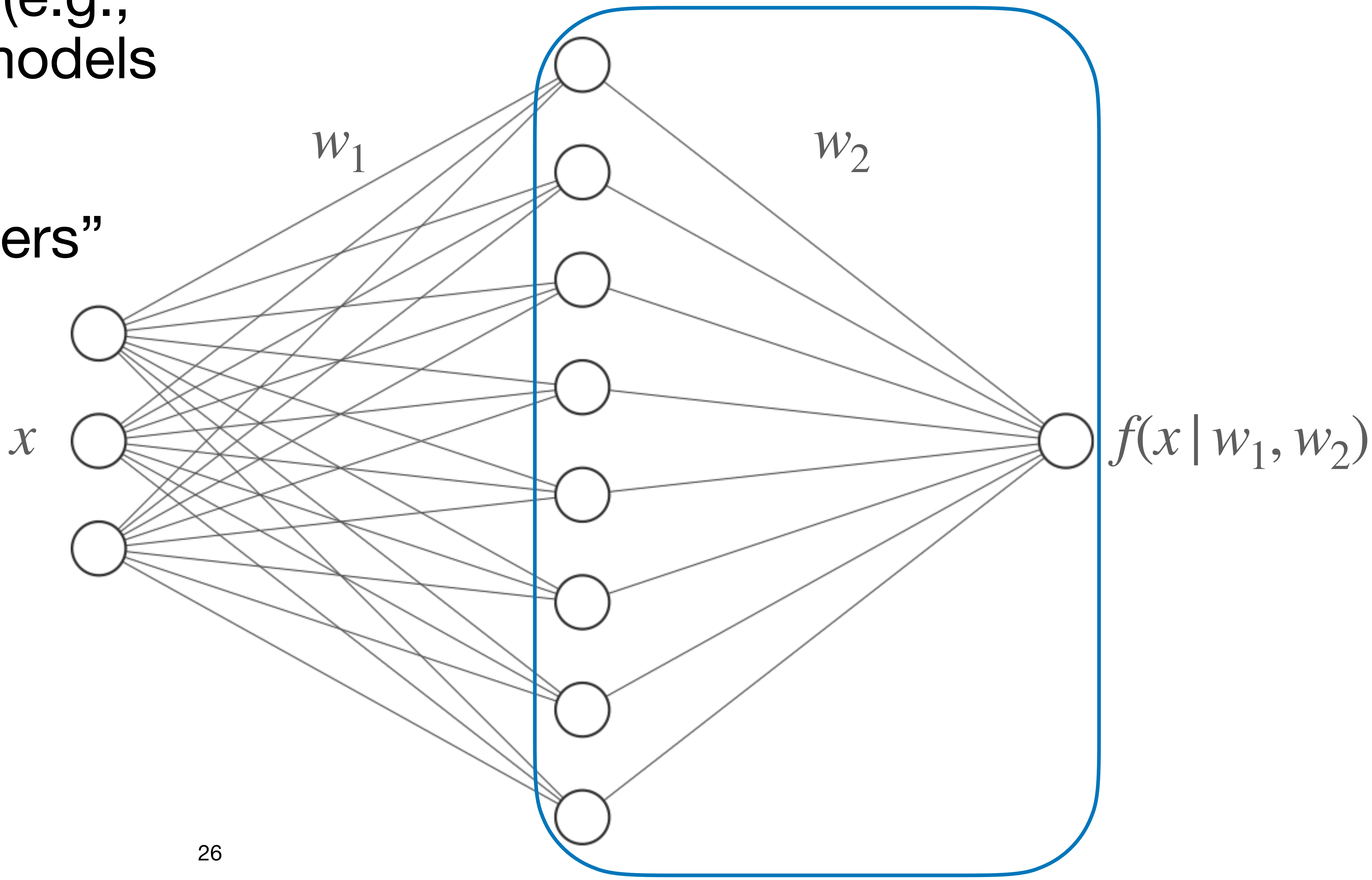
Linear models: workhorse of machine learning

- Linear models on top of good features can yield excellent results
- More complex model classes (e.g., neural networks) have linear models as their basic building block
  - NNs are “automatic featurizers”

Neural network: linear model after inputs are mapped to features through a nonlinear transformation
$f(x \mid w_1, w_2) = w_2^\intercal \sigma(w_1^\intercal x)$



Linear model: $f(x \mid w) = w^\intercal x$



Supervised learning pipeline (so far)

- Training dataset: $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ where $x \in \mathbb{R}^D$ and $y \in \mathbb{R}$
- Model / hypothesis class: $f(x \mid w) = w^\intercal x$ (linear models)
- Loss function: $L(y, y') = (y - y')^2$ (squared loss)

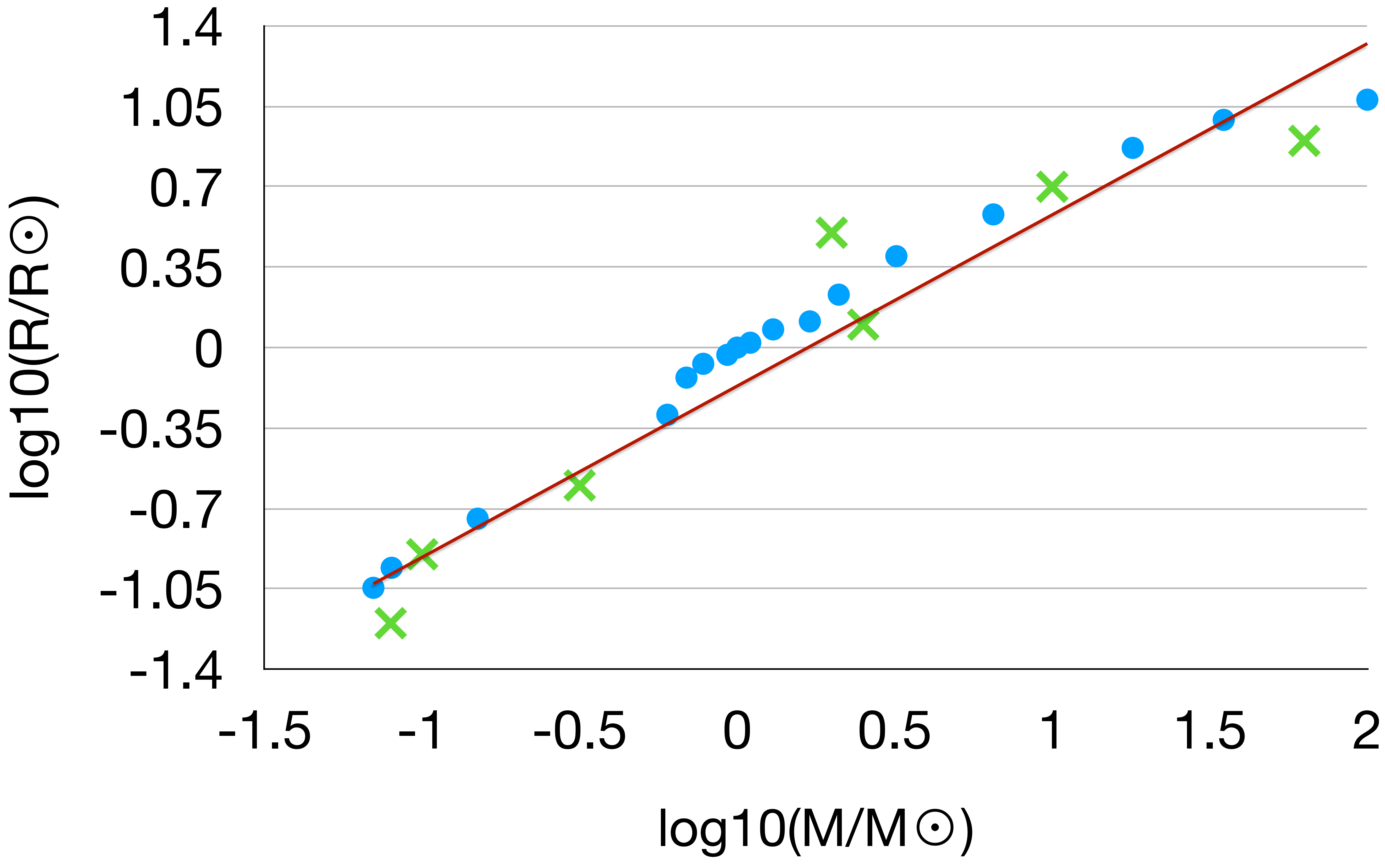
or $\phi(x)$ instead of x

- The three ingredients above define the learning objective:

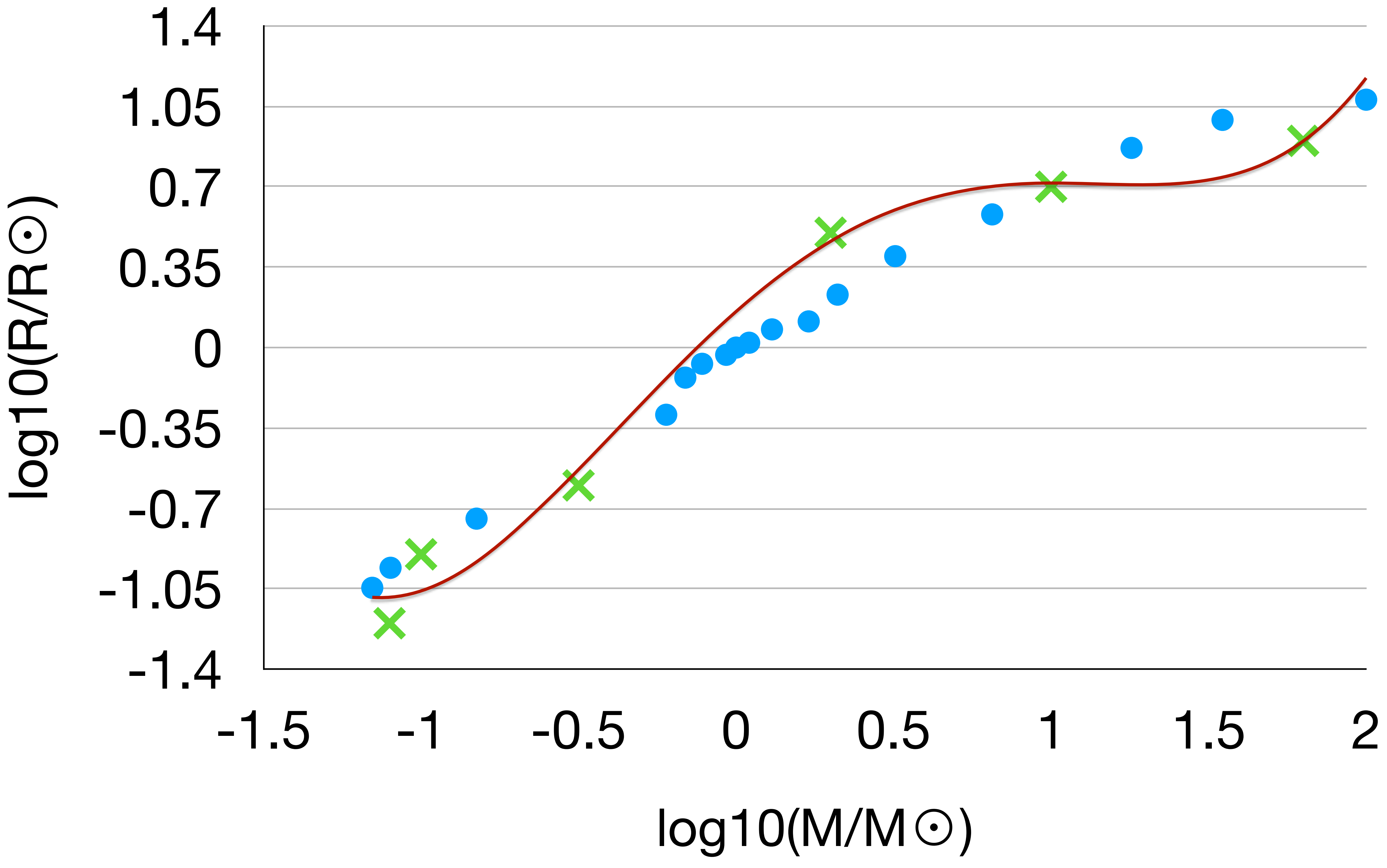
$$\arg\min_w \sum_{i=1}^{N} L(y_i, f(x_i \mid w))$$

But does your model generalize?

- Fitting the training dataset perfectly (error = 0) does not necessarily mean the model will work well on new test data!



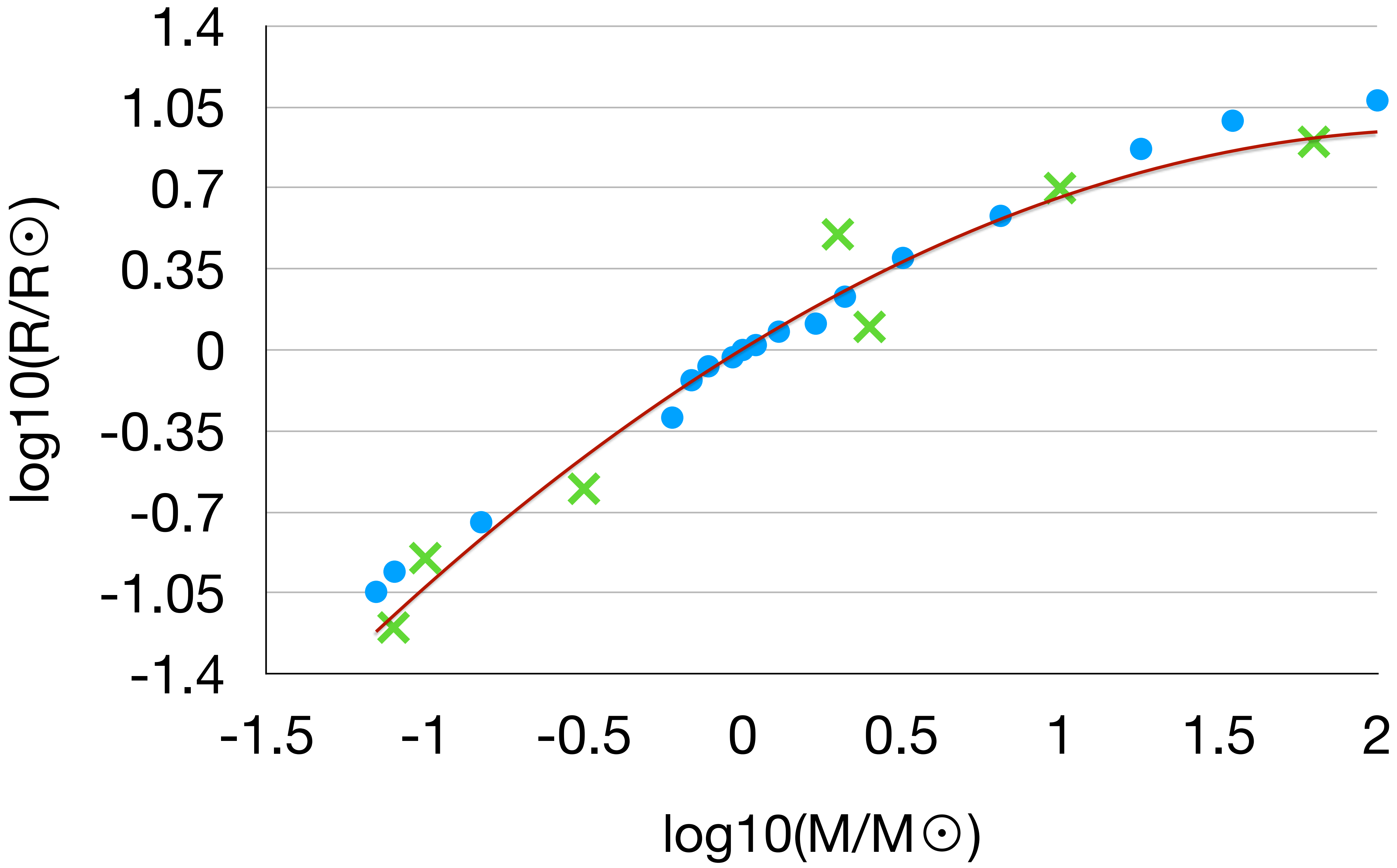
Linear fit: ok on both training and testing



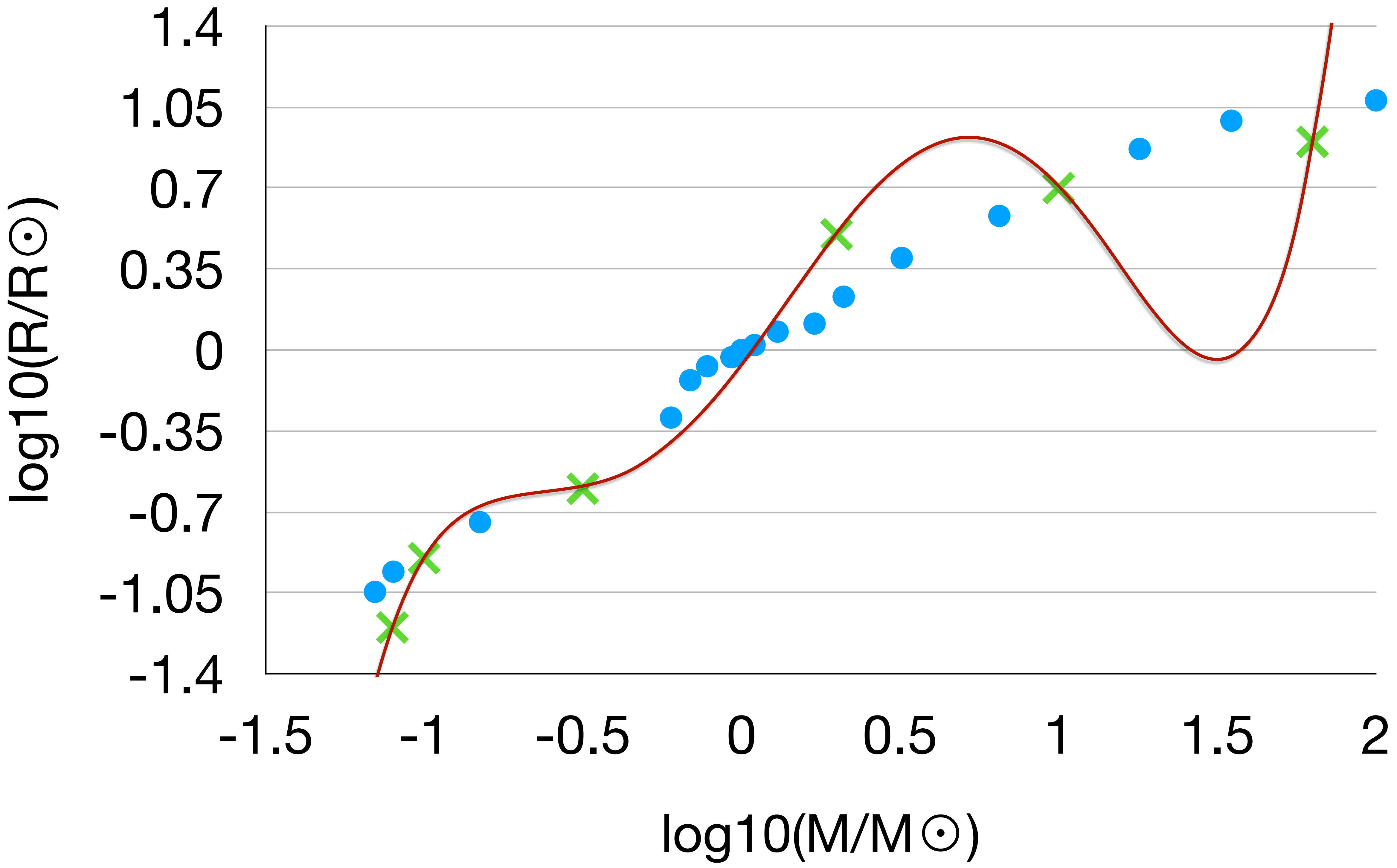
Polynomial fit (degree 4): excellent on training, bad on testing

But does your model generalize?

- Fitting the training dataset perfectly (error = 0) does not necessarily mean the model will work well on new test data!



Polynomial fit (degree 2): good on both training and testing



Polynomial fit (degree 5): perfect on training, catastrophic on testing

Test error

- Assuming that:
  - There is a "true" probability distribution $P(x, y)$ over all possible data (**unknown to us**!)
  - Each training data point (x_i, y_i) is sampled independently and identically distributed (i.i.d.) from $P(x, y)$
- Then a trained model $f(x \mid w)$ has a test error:

$$L_P(f) = \mathbb{E}_{(x,y)\sim P(x,y)}\left[L(y, f(x \mid w))\right]$$

(Prediction loss on all possible stars)

Expectation $\mathbb{E}_{z\sim P(z)}\left[g(z)\right]$: average value of $g(z)$ when z is sampled from the probability distribution $P(z)$

  - The training error is generally smaller than the test error
  - Overfitting: test error $\gg$ training error
  - Underfitting: training and test error are similar and both are high

Expected test error

- We can consider the optimal set of model parameters w_S as a function of the training dataset S

$$w_S = \arg\min_{w} \sum_{(x,y)\in S} L(y, f(x \mid w))$$

- S consists of N i.i.d. samples from $P(x, y)$ so the parameters w_S are random variables

- Expected test error:

$$\mathbb{E}\left[L_P(f(x \mid w_S)\right] = \mathbb{E}_S \mathbb{E}_{(x,y)\sim P(x,y)}\left[L(y, f(x \mid w_S))\right]$$

(fix model class f, loss function L, size of S)

Bias-variance tradeoff

- If L is the mean-squared-error loss, we can decompose the expected test error:

$$\mathbb{E}\left[L_P(f(x\,|\,w_S)\right] = \mathbb{E}_S\mathbb{E}_{(x,y)\sim P(x,y)}\left[L(y, f(x\,|\,w_S))\right]$$

$$= \mathbb{E}_{(x,y)\sim P(x,y)}\left[\underbrace{\mathbb{E}_S\left[(f(x\,|\,w_S) - F(x))^2\right]}_{\text{Variance}} + \underbrace{(F(x) - y)^2}_{\text{(Squared) bias}}\right]$$

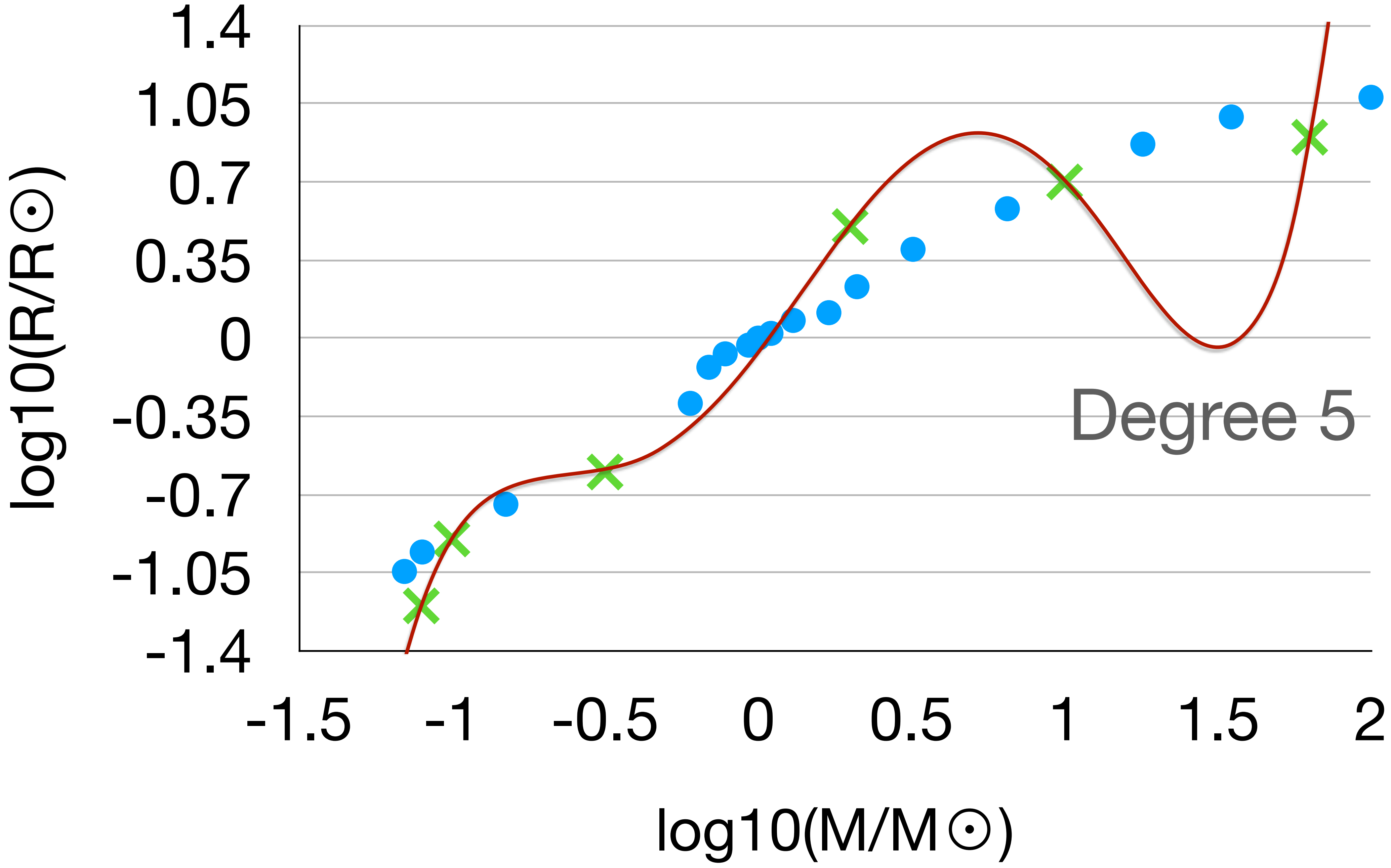
- where $F(x) = \mathbb{E}_S\left[f(x\,|\,w_S)\right]$ is the average prediction of our model over different possible training datasets

- Variance: difference in predictions when training on different datasets

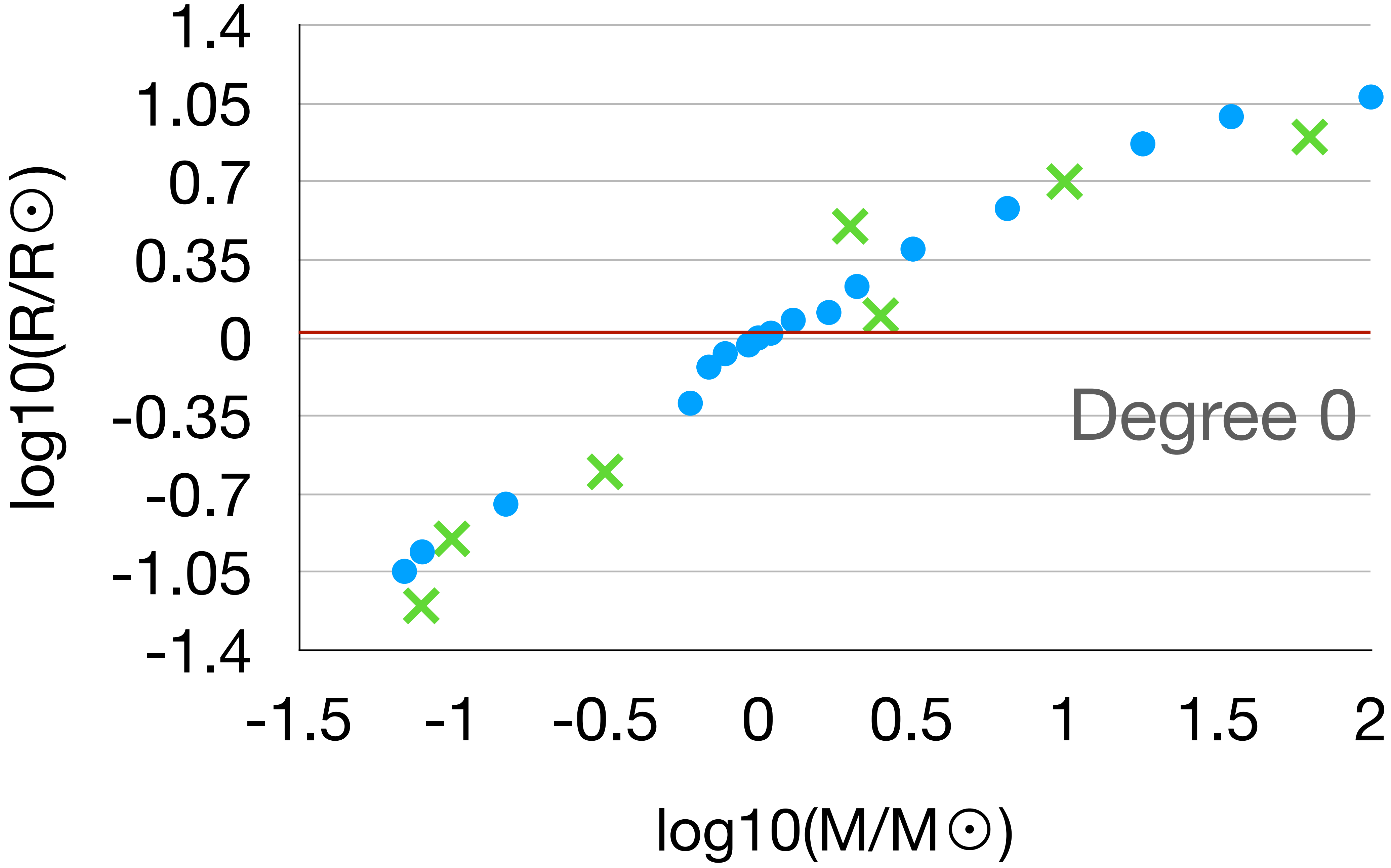
- Bias: difference from ground truth

Overfitting vs. underfitting

- Overfitting implies high variance (unstable model class)
    - Variance increases with model complexity
    - Variance decreases with more training data



- Underfitting implies high bias
    - Even with no variance, model class has high error
    - Underfitting happens whenever model complexity is too low



Model selection

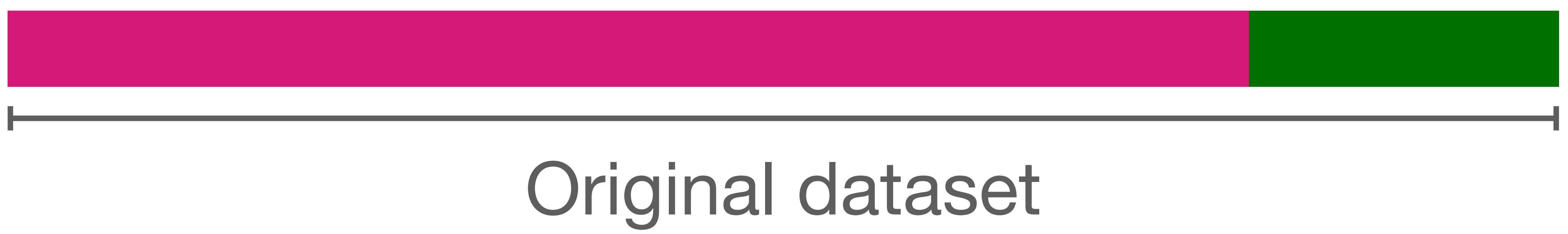
- We only have a finite training dataset
- We cannot measure the true test error
- Simple model classes underfit
- Complex model classes overfit

Bias-variance tradeoff

(but not so straightforward for deep neural networks!)

- **Goal**: Select the model class with the lowest test error

Validation set



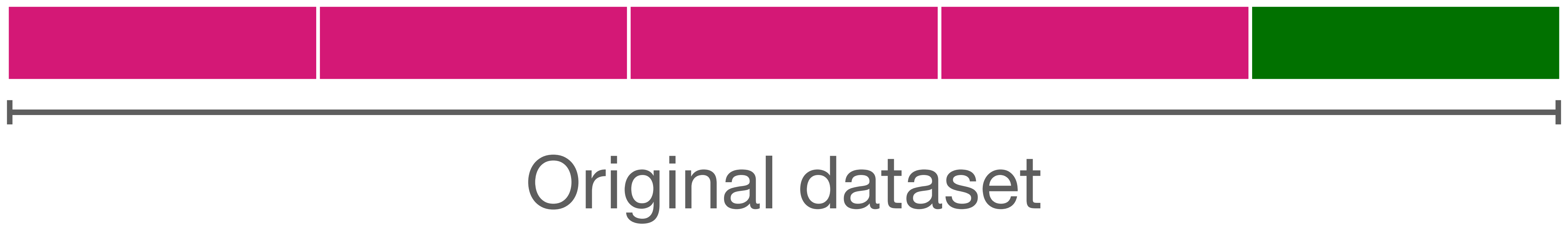
- Split the original dataset into a training and validation set
- Train model on the training set
- Evaluate on the validation set to estimate the test error
- Select the model class that gives the lowest estimated error
- Optionally, re-train the selected model class on the whole dataset (training + validation)

- **Issue**: we would like both training and validation sets to be as large as possible (so that the estimate is better), but they must not overlap!

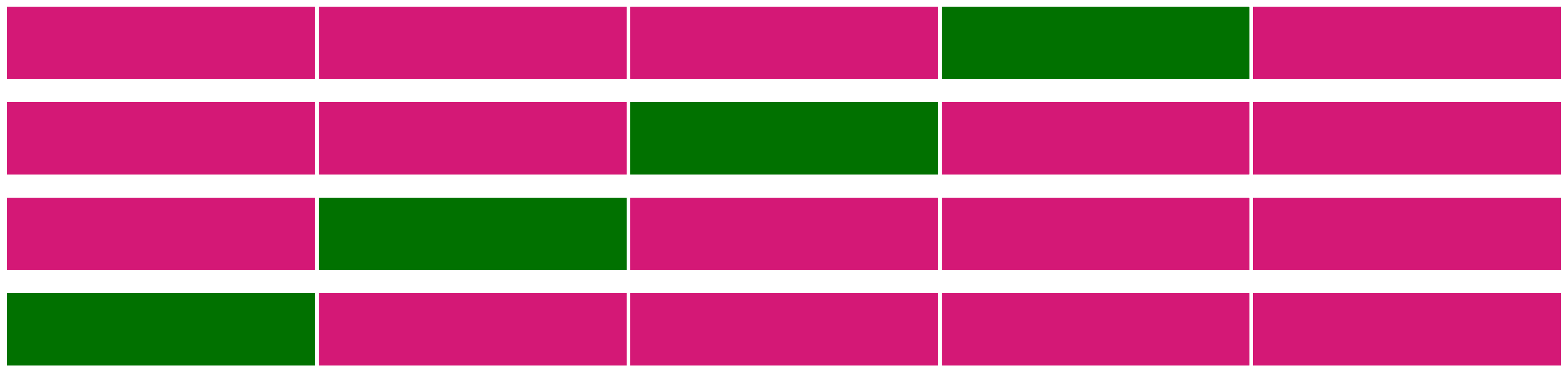
k-fold cross-validation

- Split the original dataset into k equal parts (e.g, $k = 5$)

- Train on the $k - 1$ parts and validate on the remaining one



- Repeat for every choice of the $k - 1$ parts and average the validation errors



- **Advantage**: use all data as validation to improve the estimate of the test error, at the cost of more computation (k trainings)

Supervised learning pipeline

- Training dataset: $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ where $x \in \mathbb{R}^D$ and $y \in \mathbb{R}$
- Model / hypothesis class: $f(x \,|\, w) = w^\intercal x$ (linear models)

  or $\phi(x)$ instead of x

- Loss function: $L(y, y') = (y - y')^2$ (squared loss)
- Optimization algorithm to minimize the learning objective:

$$\arg\min_{w} \sum_{i=1}^{N} L(y_i, f(x_i \,|\, w))$$

- Cross validation and model selection:

- Testing and deployment

  Important: if a testing set is available, never use it to make decisions on the model!

Next time

- Perceptron learning algorithm
- Hands-on introduction to Jupyter and DataHub