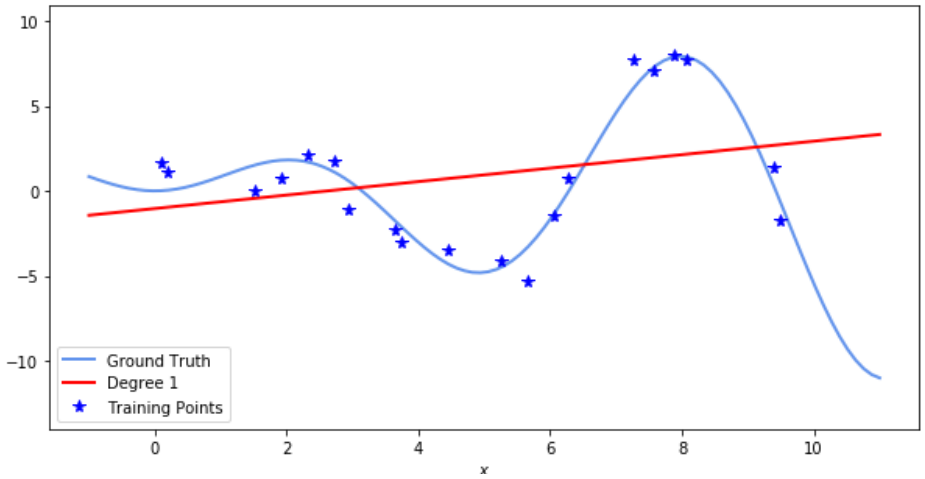
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

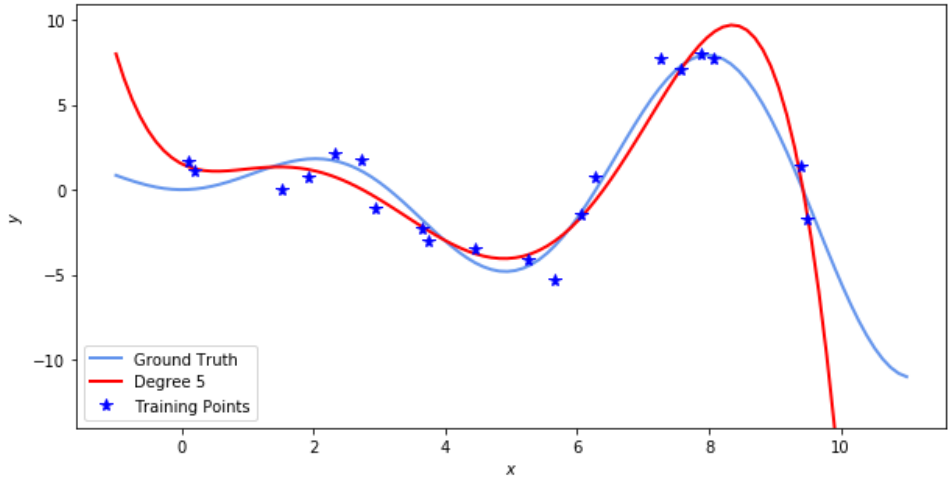
Introduction to Machine Learning

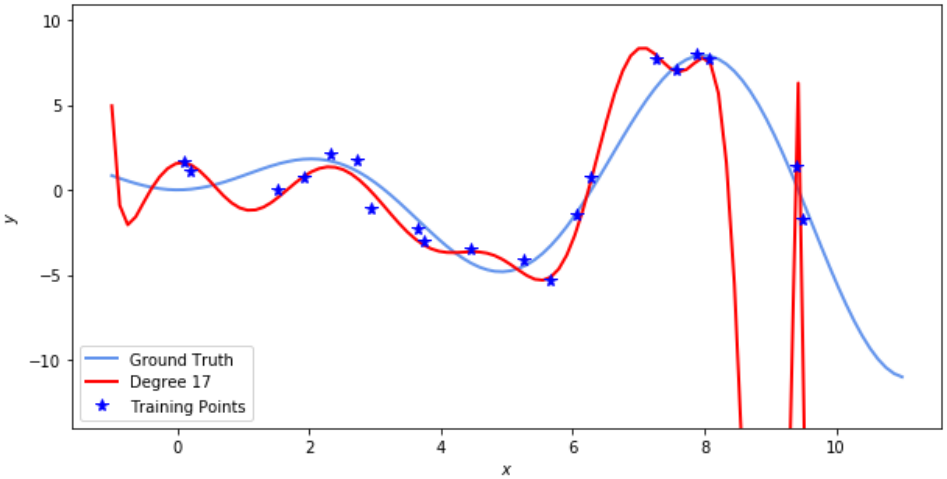
Model Validation and Selection

Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)



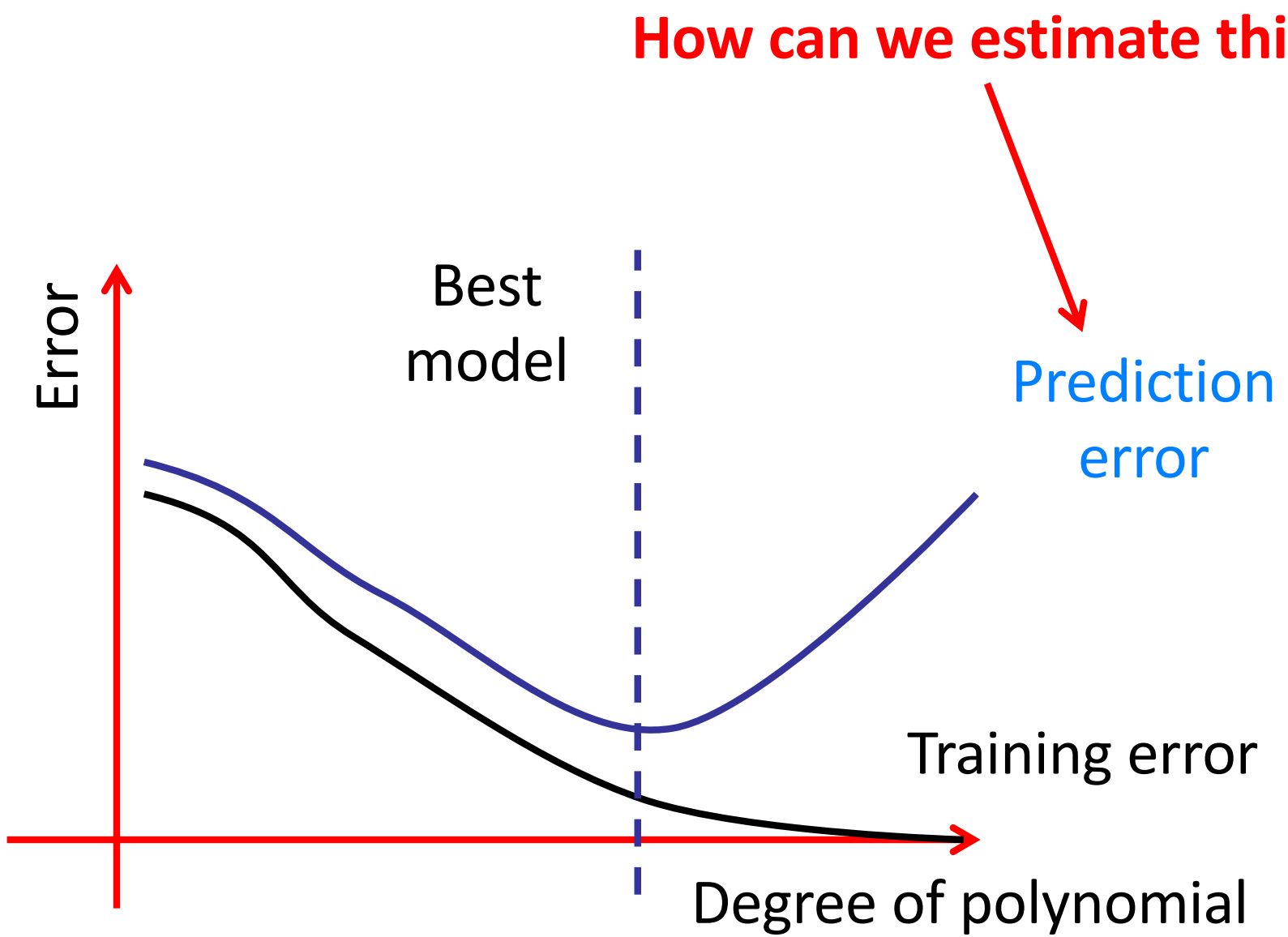
Underfitting





Overfitting

Model selection for linear regression with polynomials



Achieving generalization

- Fundamental assumption: Our data set is generated independently and identically distributed (iid) from some unknown distribution *P*

$$(\mathbf{x}_i, y_i) \sim P(\mathbf{X}, Y)$$

- Our goal is to minimize the *expected error (true risk)* under *P*

$$R(\mathbf{w}) = \int P(\mathbf{x}, y)(y - \mathbf{w}^T\mathbf{x})^2 d\mathbf{x}dy$$

$$= \mathbb{E}_{\mathbf{x},y}[(y - \mathbf{w}^T\mathbf{x})^2]$$

Evaluating predictive performance

- Training error (empirical risk) systematically underestimates true risk

$$\mathbb{E}_D\left[\hat{R}_D(\hat{\mathbf{w}}_D)\right] < \mathbb{E}_D\left[R(\hat{\mathbf{w}}_D)\right]$$

- Using an independent test set avoids this bias

$$\mathbb{E}_{D_{train},D_{test}}\left[\hat{R}_{D_{test}}(\hat{\mathbf{w}}_{D_{train}})\right] = \mathbb{E}_{D_{train}}\left[R(\hat{\mathbf{w}}_{D_{train}})\right]$$

Side note on iid assumption

- When is iid assumption invalid?
  - Time series data
  - Spatially correlated data
  - Correlated noise
  - …
- Often, can still use machine learning, but one has to be careful in interpreting results.
- Most important: Choose train/test to assess the desired generalization

First attempt: Evaluation for model selection

- Obtain training and test data D_{train} and D_{test}
- Fit each candidate model (e.g., degree *m* of polynomial)

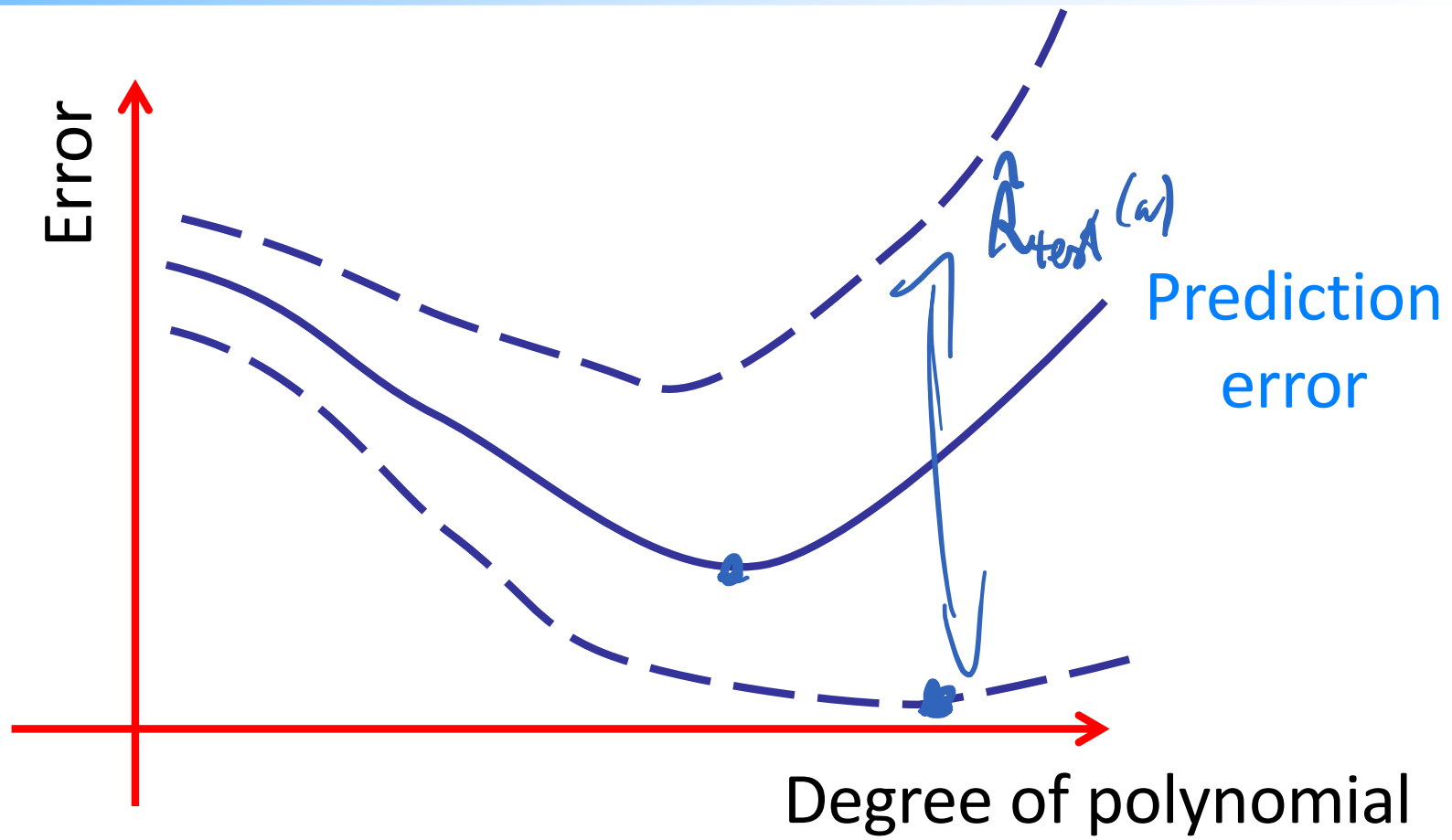
$$\hat{\mathbf{w}}_m = \underset{\mathbf{w}:\text{degree}(\mathbf{w})\leq m}{\text{argmin}} \hat{R}_{\text{train}}(\mathbf{w})$$

- Pick one that does best on test set:

$$\hat{m} = \underset{m}{\text{argmin}}\, \hat{R}_{\text{test}}(\hat{\mathbf{w}}_m)$$

- *Do you see a problem?*

Overfitting to *test* set



- Test error is itself random! Variance usually increases for more complex models
- Optimizing for *single* test set creates bias

Solution: Pick multiple test sets!

- **Key idea**: Instead of using a single test set, use multiple test sets and average to decrease variance!
- **Dilemma**:
  Any data I use for testing I can‘t use for training

➔ Using multiple independent test sets is expensive and wasteful

Evaluation for model selection

- For each candidate model *m* (e.g., polynomial degree) repeat the following procedure for i = 1:k
  - Split the same data set into training and validation set

$$D = D_{\text{train}}^{(i)} \uplus D_{\text{val}}^{(i)}$$

  - Train model

$$\hat{\mathbf{w}}_{im} = \arg\min_{\mathbf{w}} \hat{R}_{\text{train}}^{(i)}(\mathbf{w})$$

  - Estimate error

$$\hat{R}_m^{(i)} = \hat{R}_{\text{val}}^{(i)}(\hat{\mathbf{w}}_i)$$

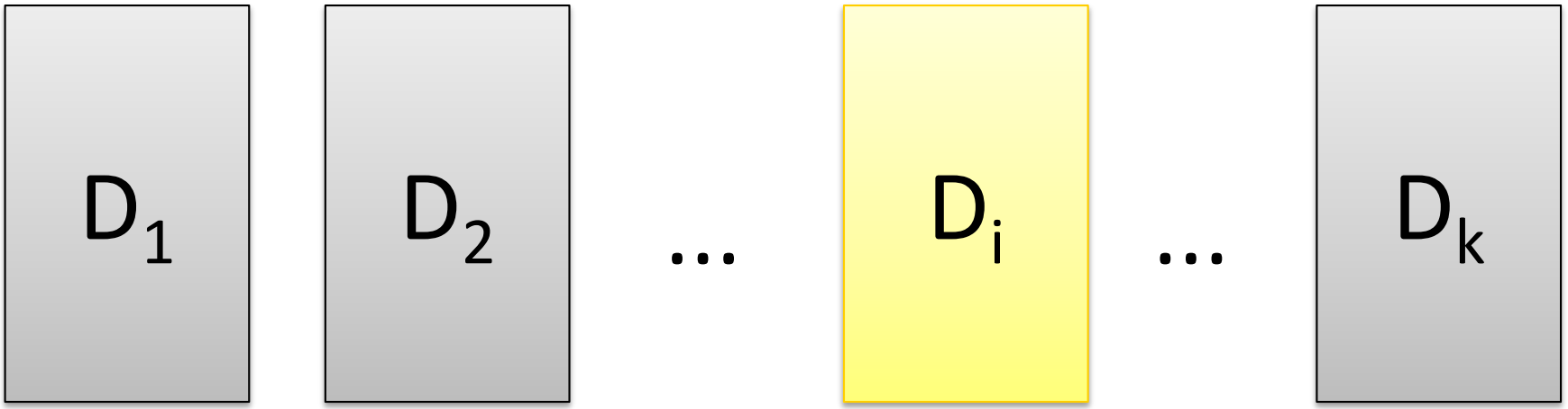
- Select model:

$$\hat{m} = \operatorname{argmin}_m \frac{1}{k} \sum_{i=1}^{k} \hat{R}_m^{(i)}$$

How should we do the splitting?

- Randomly (Monte Carlo cross-validation)
  - Pick training set of given size uniformly at random
  - Validate on remaining points
  - Estimate prediction error by averaging the validation error over multiple random trials
- k-fold cross-validation (➔ default choice)
  - Partition the data into *k* „folds“
  - Train on *(k-1)* folds, evaluating on remaining fold
  - Estimate prediction error by averaging the validation error obtained while varying the validation fold

k-fold cross-validation



Accuracy of cross-validation

- Cross-validation error estimate is very nearly unbiased for large enough *k*
- *Show demo*

Cross-validation

- How large should we pick k?
- Too small
  - ➔ Risk of overfitting to test set
  - ➔ Using too little data for training
  - ➔ risk of underfitting to training set
- Too large
  - In general, better performance! k=n is perfectly fine (called leave-one-out cross-validation, LOOCV)
  - Higher computational complexity
- In practice, k=5 or k=10 is often used and works well

Best practice for evaluating supervised learning

- Split data set into training and test set
- Never look at test set when fitting the model.
  For example, use *k*-fold cross-validation on training set
- Report final accuracy on test set
  (but never optimize on test set)!

- **Caveat**: This only works if the data is i.i.d.
- Be careful, for example, if there are temporal trends or other dependencies

Model selection more generally

- For polynomial regression, model complexity is naturally controlled by the degree
- In general, there may not be an ordering of the features that aligns with complexity
  - E.g., how should we order words in the bag-of-words model?
  - Collection of nonlinear feature transformations

$$x \mapsto \log(x + c)$$

$$x \mapsto x^{\alpha}$$

$$x \mapsto \sin(ax + b)$$

$x \longmapsto \phi(x) = [x_1, x_2, x_1^2, \sin(x_2), \sqrt{x_3}]$

- Now model complexity is no longer naturally „ordered“

Demo: Overfitting → Large Weights

Regularization

- If we only seek to minimize our loss (optimize data fit) can get very complex models (large weights)

- Solution?

- Regularization!
  Encourage small weights via penalty functions (regularizers)

Ridge regression

- Regularized optimization problem:

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda ||\mathbf{w}||_2^2$$

$\lambda \geq 0$

$= \sum_{j=1}^{d} w_j^2$

- Can optimize using gradient descent, or still find analytical solution:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

$\uparrow \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix} \in \mathbb{R}^{d}$

- Note that now the **scale of x** matters!

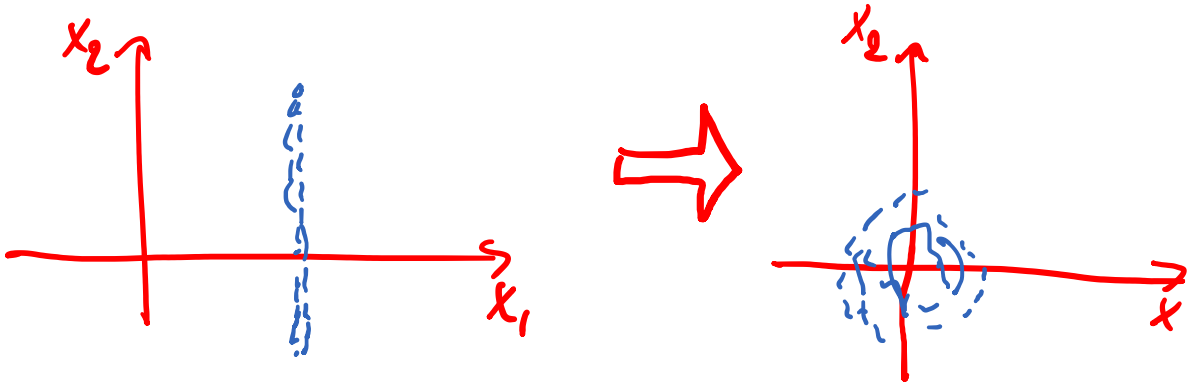
Renormalizing data: Standardization

- Ensure that each feature has zero mean and unit variance

$$\tilde{x}_{i,j} = (x_{i,j} - \hat{\mu}_j)/\hat{\sigma}_j$$

- Hereby $x_{i,j}$ is the value of the j-th feature of the i-th data point

$$\hat{\mu}_j = \frac{1}{n}\sum_{i=1}^{n} x_{i,j} \qquad \hat{\sigma}_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_{i,j} - \hat{\mu}_j)^2$$



Gradient descent for ridge regression

$$\nabla_w \left(\underbrace{\frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2}_{\hat{R}(w)} + \lambda \|w\|_2^2 \right) = \nabla_w \hat{R}(w) + \lambda \nabla_w \|w\|_2^2$$

$$\nabla_w \|w\|_2^2 = \nabla_w (w^T w) = 2w$$

$$w_{t+1} \leftarrow w_t - \eta_t \left(\nabla \hat{R}(w) + 2\lambda w_t \right) = (1 - 2\lambda\eta_t) w_t - \eta_t \nabla \hat{R}(w)$$

Demo: Regularization

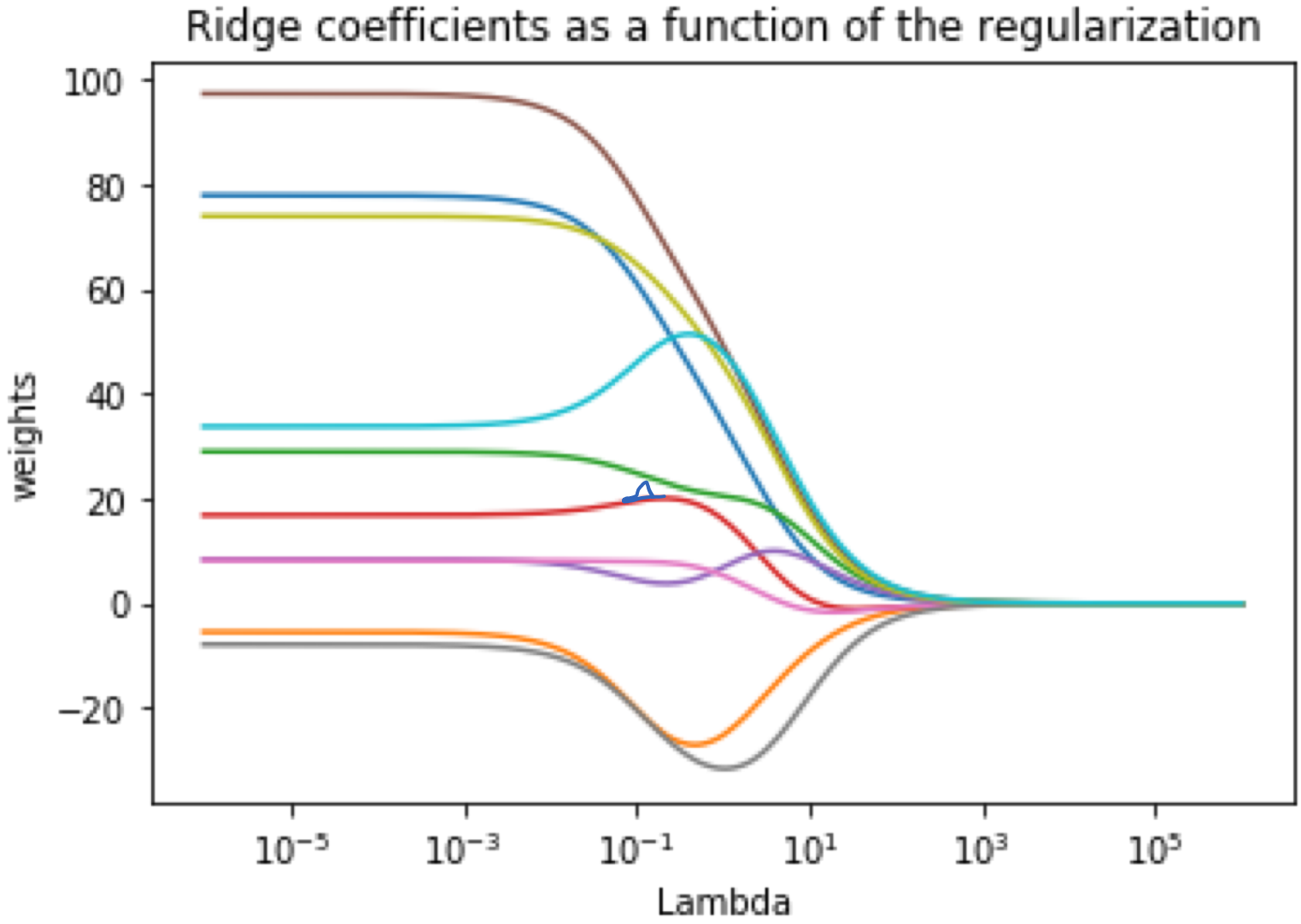
How to choose regularization parameter?

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda ||\mathbf{w}||_2^2$$

- Cross-validation!
- Typically pick λ logarithmically spaced:

$\lambda \in \{10^{-5}, 10^{-4}, \ldots 10^{4}, 10^{5}\}$

Regularization path



Outlook: Fundamental tradeoff in ML

- Need to trade loss (goodness of fit) and simplicity
- A lot of supervised learning problems can be written in this way:

$$\min_{\mathbf{w}} \hat{R}(\mathbf{w}) + \lambda C(\mathbf{w})$$

- Can control complexity by varying regularization parameter λ
- Many other types of regularizers exist and are very useful (more later in this class)

What you need to know

- Linear regression as model and optimization problem
  - How do you solve it?
  - Closed form vs gradient descent
  - Can represent non-linear functions using basis functions
- Model validation
  - Resampling; Cross-validation
- Model selection for regression
  - Comparing different models via cross-validation
- Regularization
  - Adding penalty function to control magnitude of weights
  - Choose regularization parameter via cross-validation