UCLA

CS M146: Introduction to Machine Learning
Generalization

Aditya Grover

UCLA

The instructor gratefully acknowledges Sriram Sankararaman (UCLA) and Andrew Ng (Stanford) for some of the materials and organization used in these slides, and many others who made their course materials freely available online.

https://aditya-grover.github.io/ @adityagrover_

Recap: Linear Regression

3 Steps:

- Represent hypothesis class

  Linear functions: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x}$ (linear in both $\boldsymbol{\theta}$ and $\boldsymbol{x}$)

- Define a loss function

  Squared error loss: $J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right)^2$

- Optimize loss to find best hypothesis

  Gradient descent: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{1}{n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) \boldsymbol{x}^{(i)}$

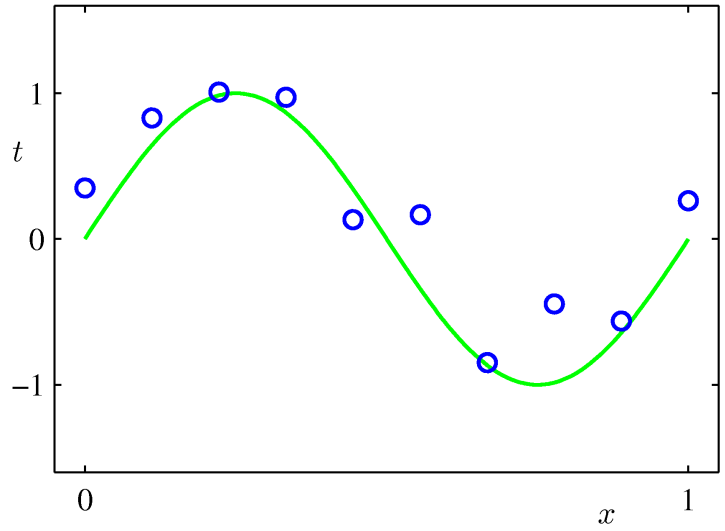
  Normal equations: $\widehat{\boldsymbol{\theta}} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & \dots & x_d^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Extending Linear Regression

- Many functions of real-world interest pose a non-linear relationship between features x and labels y

  E.g., housing rents as a function of the month/year [higher in summer, lower in winter]



Extending Linear Regression

- So far: hypothesis is linear in both $\boldsymbol{\theta}$ and $\boldsymbol{x}$

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x}$$

- Conventionally, the term **linear** in linear regression is reserved for the relationship between $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ and $\boldsymbol{\theta}$
- Possible inputs for linear regression:
  - Original features $\boldsymbol{x}$ e.g., real-valued, categorical encodings, mixture
  - Non-linear transformations of original features, e.g, $\boldsymbol{x^2}, \log \boldsymbol{x}, \sqrt{\boldsymbol{x}}$ (applied elementwise)
  - Non-linear interactions between original features, e.g., $x_3 = x_1^2 x_2$

Linear Basis Function Models

- Generalized class of linear hypothesis

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \phi(\boldsymbol{x}) = \sum_{j=0}^{k} \theta_j \phi_j(\boldsymbol{x})$$

- $\phi(\boldsymbol{x})$: $\mathbb{R}^d \to \mathbb{R}^k$ is a k-dimensional basis with parameters $\boldsymbol{\theta} \in \mathbb{R}^k$
  - Typically, $\phi_0(\boldsymbol{x}) = 1$ so θ_0 acts as a bias as usual
  - k can be different from $d + 1$. E.g., polynomial regression ($d = 1, k = 4$)

$$\phi(x) = [1, x, x^2, x^3]$$

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

- Everything else same as before, replace $\boldsymbol{x}$ by $\phi(\boldsymbol{x})$.

Extending Linear Regression

3 Steps:

- Represent **hypothesis class and inputs**

  Linear functions: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \phi(\boldsymbol{x})$

- Define a loss function

  Squared error loss: $J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right)^2$

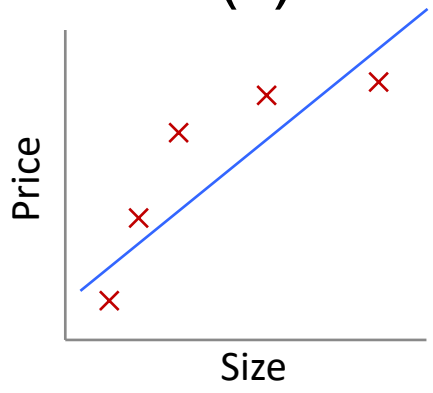
- Optimize loss to find best hypothesis

  Gradient descent: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{1}{n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) \phi(\boldsymbol{x}^{(i)})$

  Normal equations: $\widehat{\boldsymbol{\theta}} = (\phi(\boldsymbol{X})^T \phi(\boldsymbol{X}))^{-1} \phi(\boldsymbol{X})^T \boldsymbol{y}$

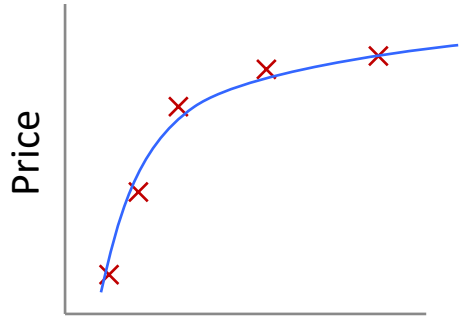
More Complex Is Not Always Better

(a)



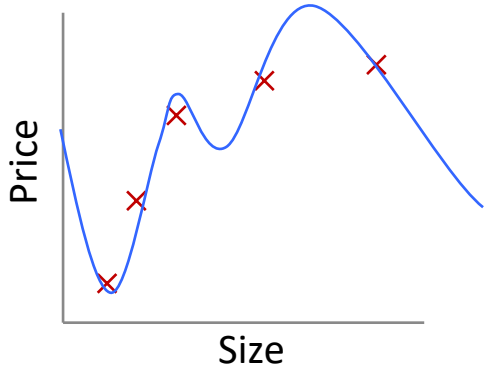
$\theta_0 + \theta_1 x$

(b)



$\theta_0 + \theta_1 x + \theta_2 x^2$

(c)

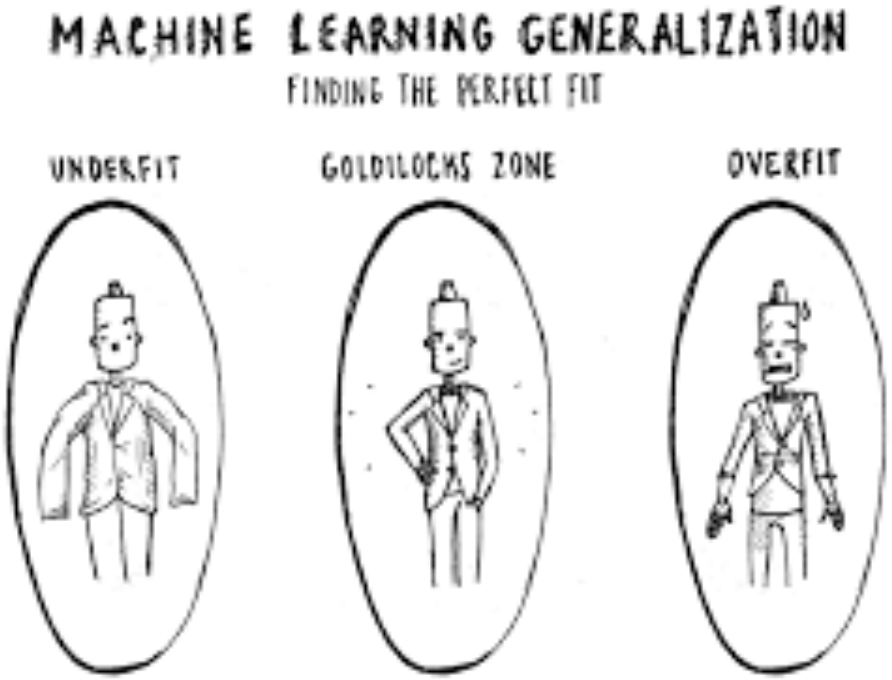


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Which of these curves is the best fit?

- For training data, (c) is best – zero training loss!
- For test data, cannot say. Depends on the (unknown) true hypothesis
  - If training dataset is noisy and true hypothesis is linear, (a) could be best
  - With no noise, both (b, c) obtain ~zero training loss and could be best

Generalization



Generalization

- The goal of learning is to **generalize** to making good predictions on **unseen** test instances

- **Empirical risk minimization (ERM):** define loss on training instances

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - \ y^{(i)}\right)^2$$

- Does ERM solution generalize to unseen $\boldsymbol{x}$?

Generalization

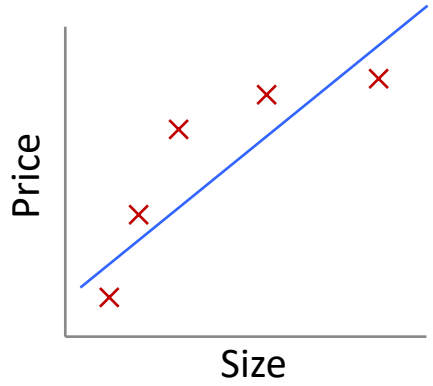
- When (if) and how does the ERM solution generalize on test data?
- One of the big mysteries of ML!

- **Theoretically:** Depends on hypothesis class, data size, learning algorithm. Studied in **learning theory** (out of scope)
- **Empirically:** Can assess via **validation data** (this lecture)
- **Algorithmically:** Can strengthen via **regularization** (this lecture)

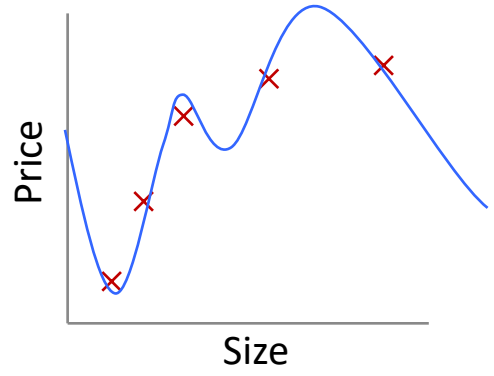
Lack of Generalization

- Two reasons why a ML model may not work in the real world
  - **Underfitting**: Hypothesis is not very **expressive** / **complex**
  - **Overfitting**: Hypothesis is too complex for given dataset
- Today's lecture:
  - What is hypothesis complexity?
  - How to detect underfitting/overfitting? **Validation**
  - How to control overfitting? **Regularization**

Hypothesis Complexity



$\theta_0 + \theta_1 x$

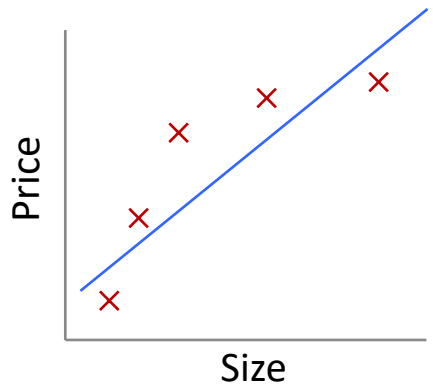


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

- Which is more complex: deg 1 or deg 4 polynomials?
- Hypothesis complexity: Notion of representative ability of a hypothesis class $\mathcal{H}$
- Very hard to define formally!
  - Many attempts in learning theory. All have flaws.
- In certain contexts, we will rely on intuitive proxies. E.g., degree of polynomial in $\boldsymbol{x}$ for linear regression

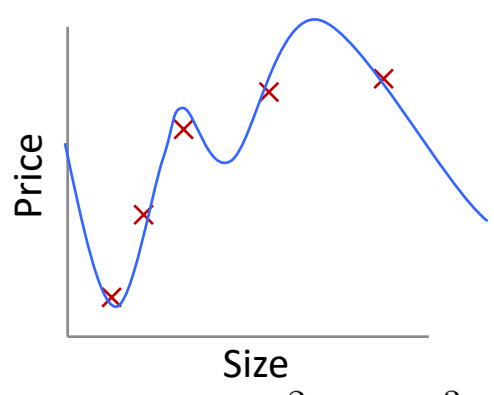
Model Selection

h_1



$\theta_0 + \theta_1 x$

h_2

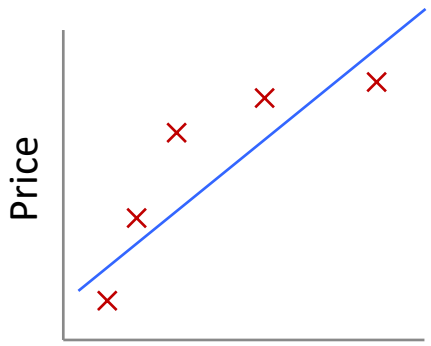


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

- Say we have two hypothesis, h_1 and h_2
- **Model selection:** How do we pick the best one to use for future predictions on some unseen test data?
- Can we use train loss for model selection?
- No! An n degree polynomial can easily get 0 train loss on a dataset of size n

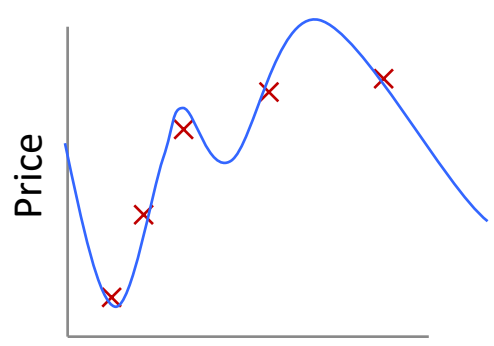
Model Selection

h_1



$\theta_0 + \theta_1 x$

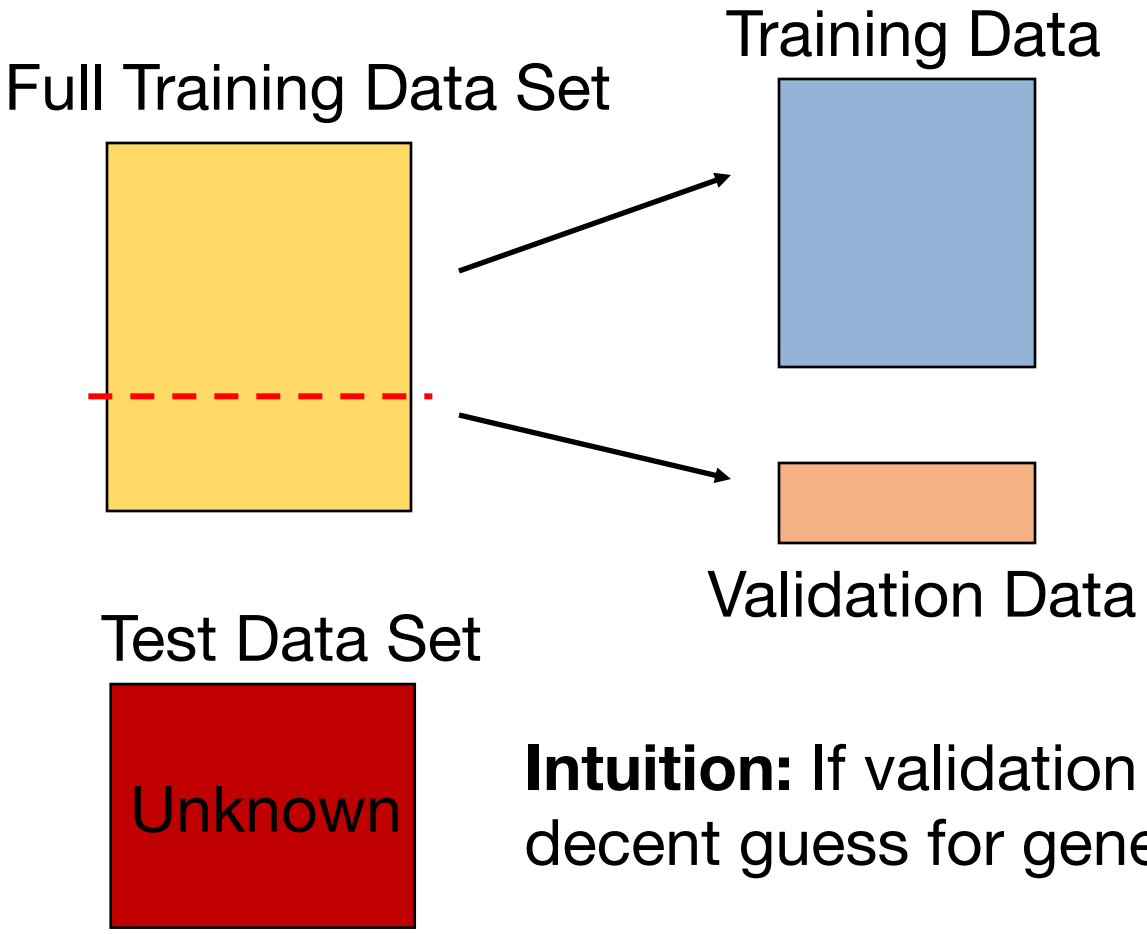
h_2



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

- Ideally, want to choose based on test accuracy…
- But we don't observe test data ☹
- Can we pretend to have test data?

Training, Validation, Test Data



Idea:
Train each model on a subset of full training data...

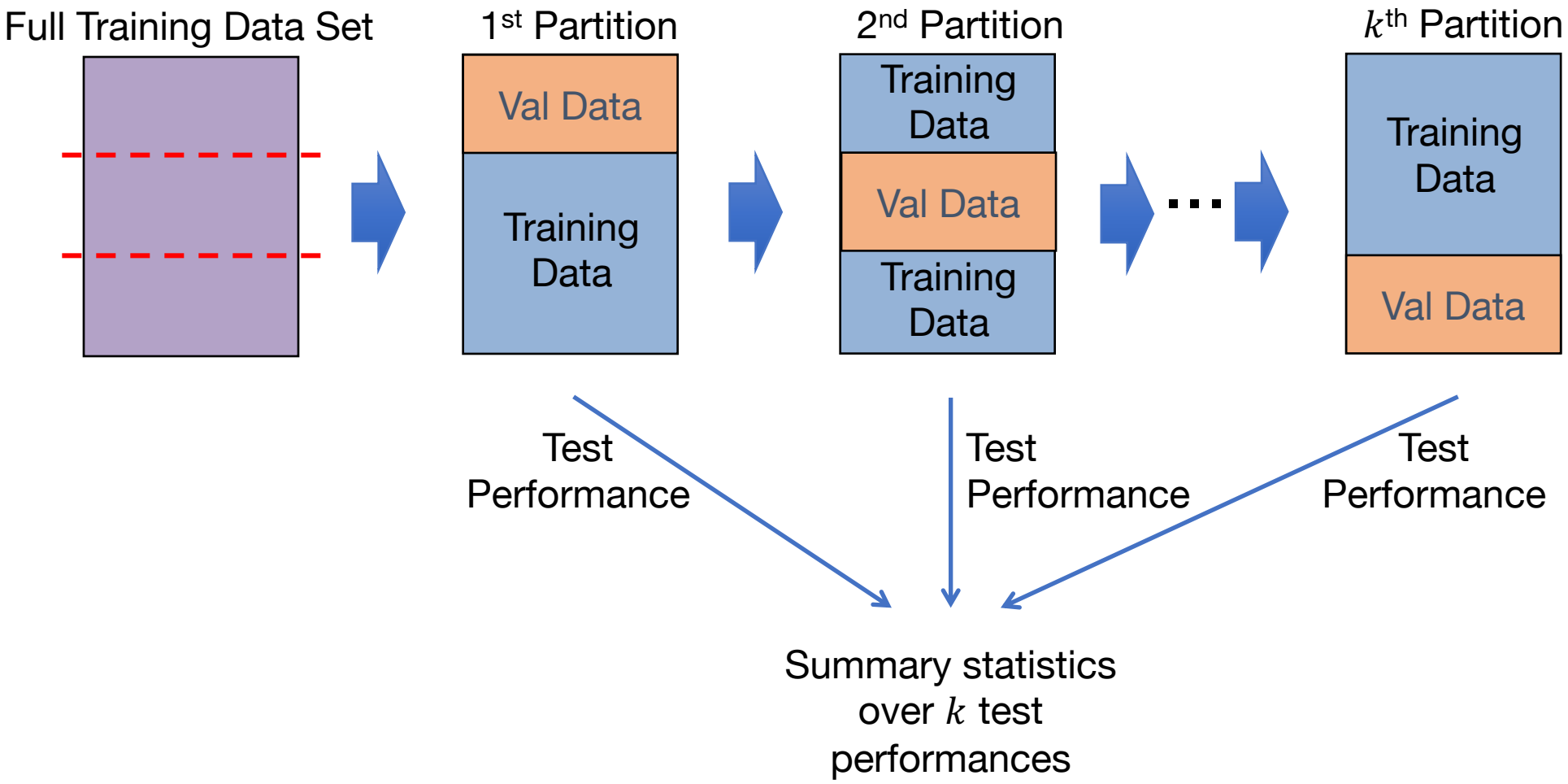
...and then evaluate each model's accuracy on the held-out data a.k.a. **validation/dev data**

Intuition: If validation data ~ test data, then we have a decent guess for generalization

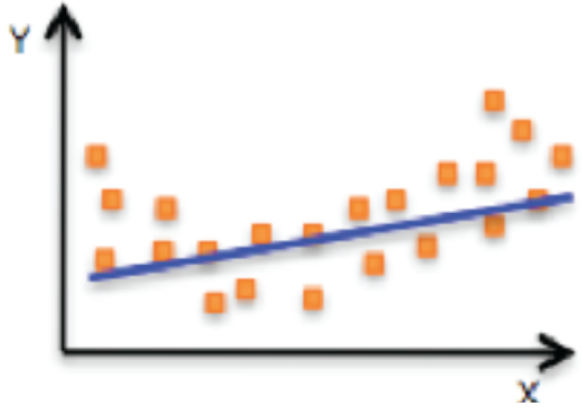
k-fold Cross Validation

- Why just choose one particular "split" of the data?
  - In principle, we should do this multiple times since performance may be different for each split
- **k-fold cross validation (CV)** (e.g., $k = 10$)
  - Randomly partition full data set of n instances into k disjoint subsets ("folds")
  - For fold $i = 1,2, \ldots, k$
    - Choose fold i as the test set
    - Train model on the other $k - 1$ folds and evaluate accuracy on fold i
  - Compute average statistics over k folds, or choose best of the k models
- "leave-one-out CV": $k = n$.

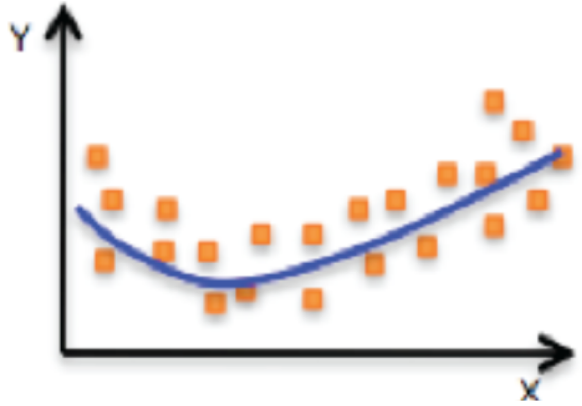
Example: 3-Fold CV



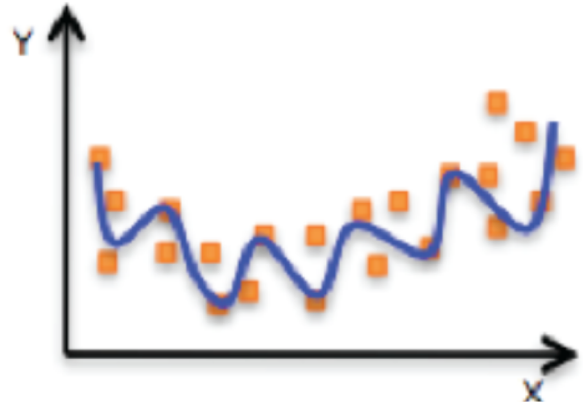
Detecting Underfitting and Overfitting



Case 1



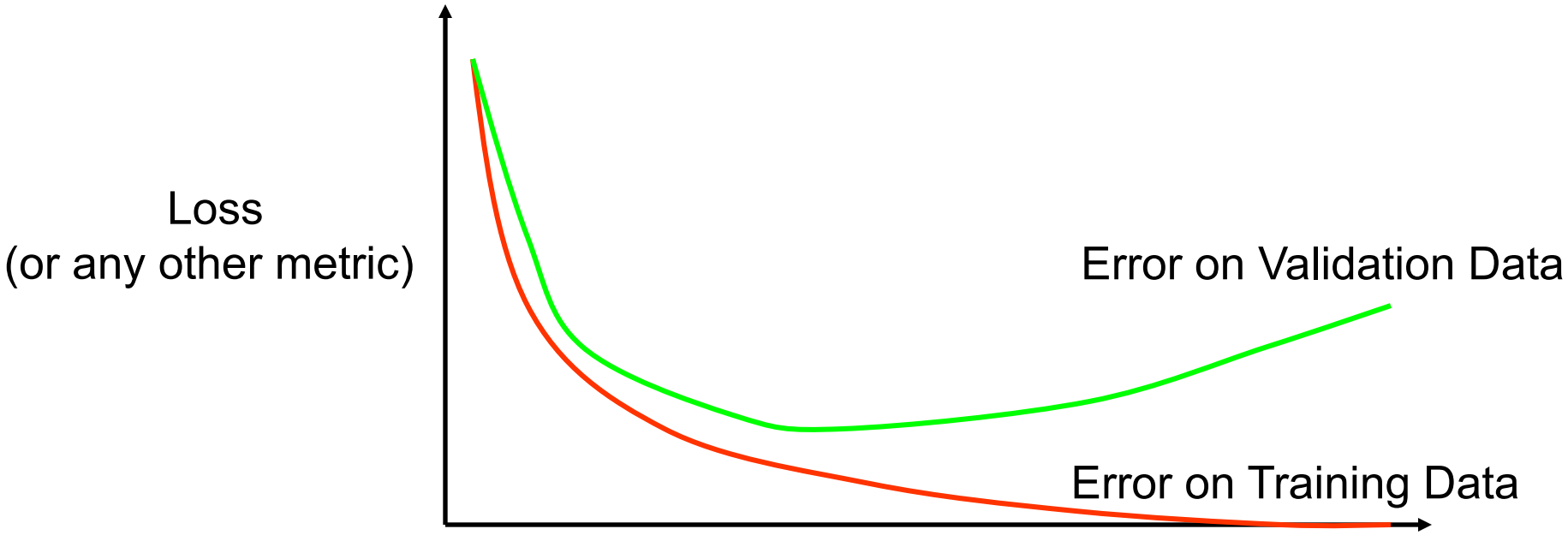
Case 2

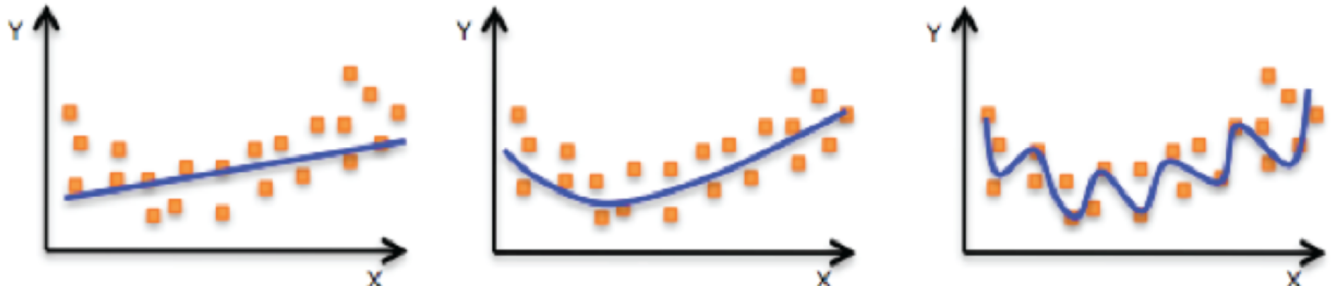


Case 3

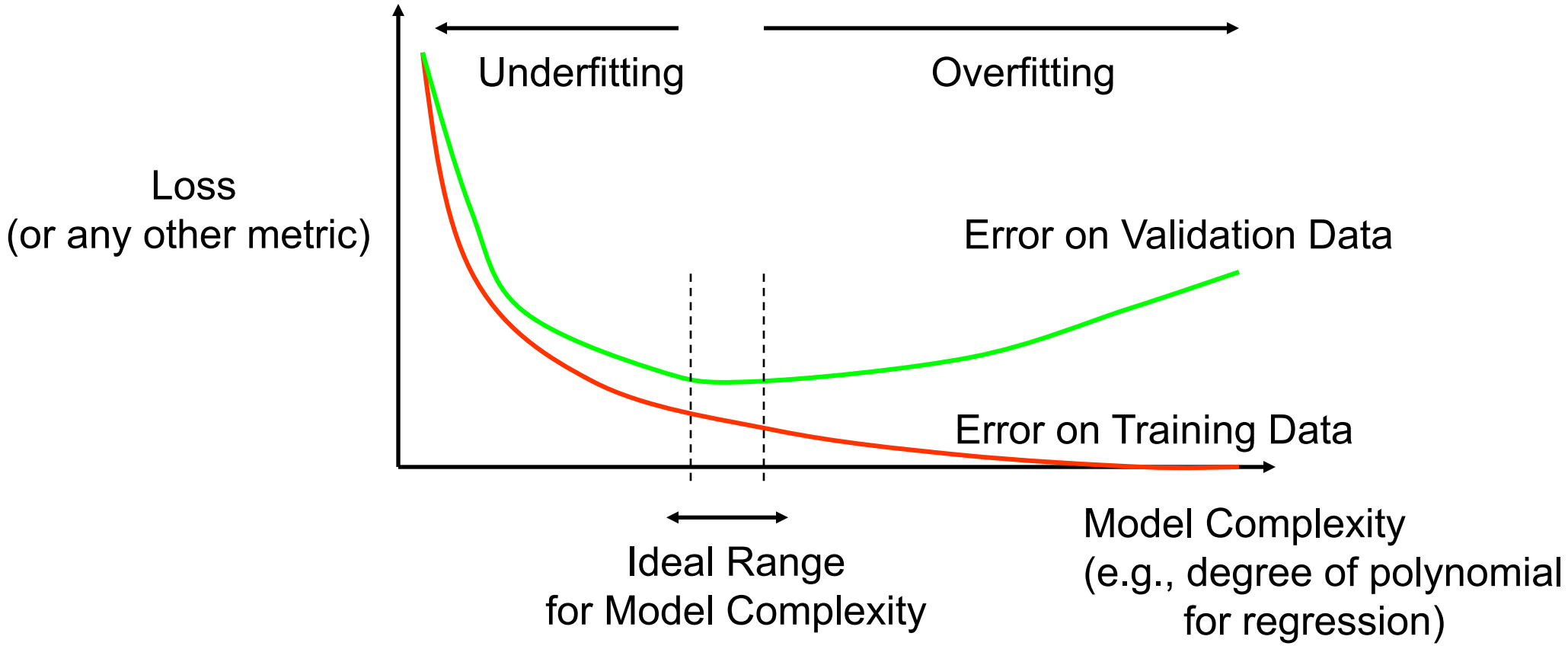
	Case 1	Case 2	Case 3
Training loss:	High	Medium	Low
Validation loss:	High	Medium	High
	Underfitting	**Good fit**	**Overfitting**

Detecting Underfitting and Overfitting





Detecting Underfitting and Overfitting



Underfitting and overfitting show very different behaviors on training and validation data

Regularization

Regularization

- Addressing overfitting:
  - eliminating features
  - getting more data

- **Loss Regularization:** A method for preventing overfitting by automatically controlling the complexity of the learned hypothesis

- **Idea**: penalize large values of θ_j during optimization
  - Can incorporate into the loss function
  - Works well when we have a lot of features, each that contributes a bit to predicting the label

Regularization

- Regularized linear regression objective function

$$J(\boldsymbol{\theta}) = \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2}_{\text{model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{regularization}}$$

- λ is the regularization hyperparameter ($\lambda \geq 0$)
- Note: no need for regularization on the bias θ_0

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- Note that $\sum_{j=1}^{d}\theta_j^2 = \|\boldsymbol{\theta}_{1:d}\|_2^2$
  - This is the squared magnitude (or $\ell_2$norm) of the parameter vector!
  - Also called **ℓ_2- regularized regression** or **ridge regression**
- We can also think of this as:

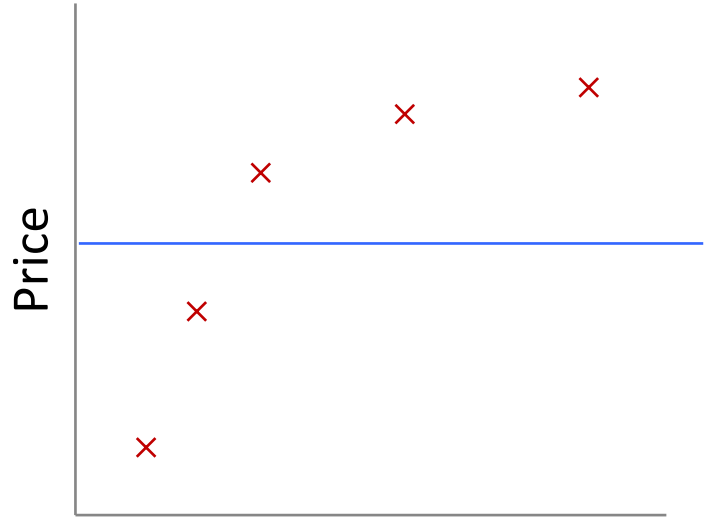
$$\sum_{j=1}^{d}\left(\theta_j - 0\right)^2 = \|\boldsymbol{\theta}_{1:d} - \mathbf{0}\|_2^2$$

  - **ℓ_2** regularization pulls parameter vector towards the origin

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

0 0 0 0

Regularized Linear Regression

- Loss Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- Fit by solving

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- Gradient updates

$\frac{\partial}{\partial\theta_0}J(\theta)$

$$\theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$\frac{\partial}{\partial\theta_j}J(\theta)$

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} - \alpha\lambda\theta_j$$

regularization

Regularized Linear Regression

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

$$\theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} - \alpha\lambda\theta_j$$

- We can rewrite the gradient step as:

$$\theta_j \leftarrow \theta_j\left(1 - \alpha\lambda\right) - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Regularized Linear Regression

- To incorporate regularization into the closed form solution:

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^\intercal \boldsymbol{X} \cdot \right)^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$$

- Can derive this the same way as before, by solving $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = 0$
- Can prove that for $\lambda > 0$ inverse exists in the equation above

Regularized Linear Regression

- To incorporate regularization into the closed form solution:

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^\intercal \boldsymbol{X} + \lambda \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \right)^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$$

- Can derive this the same way as before, by solving $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = 0$
- Can prove that for $\lambda > 0$ inverse exists in the equation above

Hyperparameters

Hyperparameters

- **Hyperparameters:** Additional unknowns (different from model parameters $\boldsymbol{\theta}$) for improving learning
- Two kinds of hyperparameters
  - **Model hyperparameters** influence the representation. E.g.,
    - Hypothesis class $\mathcal{H}$ (e.g., perceptron vs. logistic regression)
    - Basis function ϕ
  - **Algorithmic hyperparameters** influence the model training. E.g.,
    - Learning rate α
    - Regularization coefficient λ
    - Batch size B

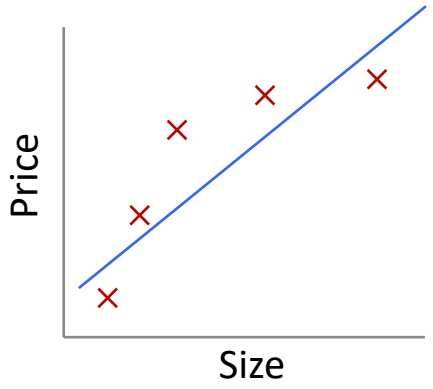
Model Hyperparameters

Hypothesis Class

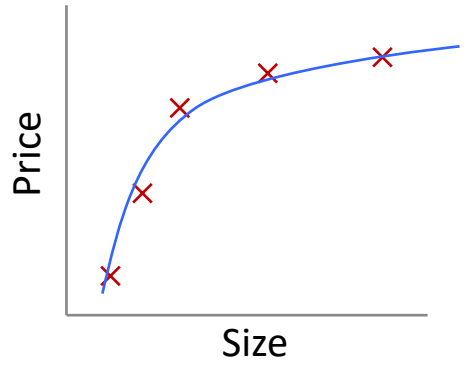
$$\hat{y} = \log(\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3)$$

$$\hat{y} = \frac{\theta_0}{1 + \theta_1^{(x-\theta_2)}}$$

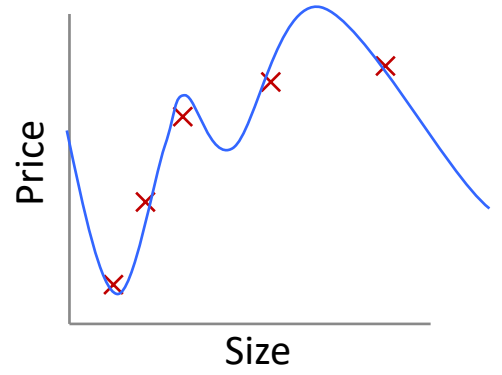
Basis Functions



$\theta_0 + \theta_1 x$

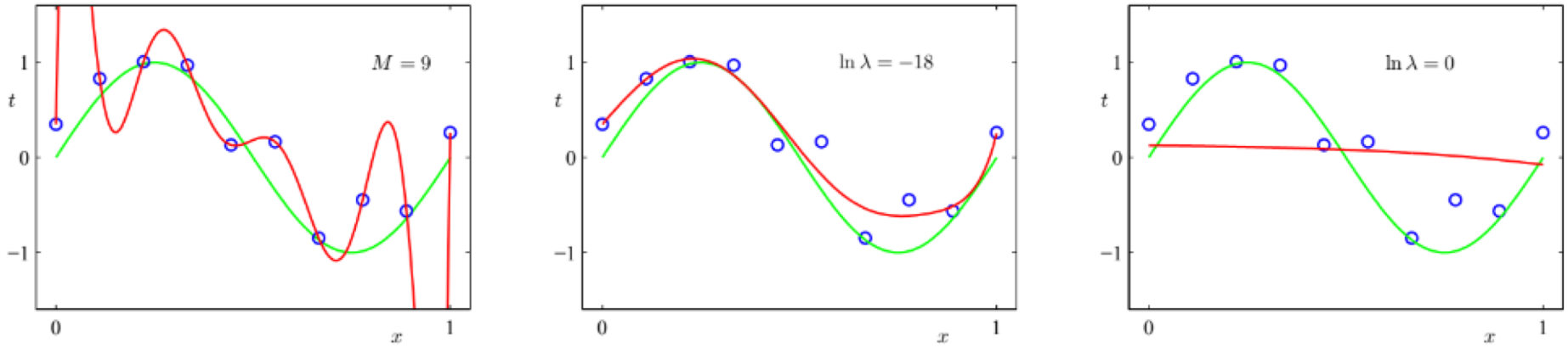


$\theta_0 + \theta_1 x + \theta_2 x^2$

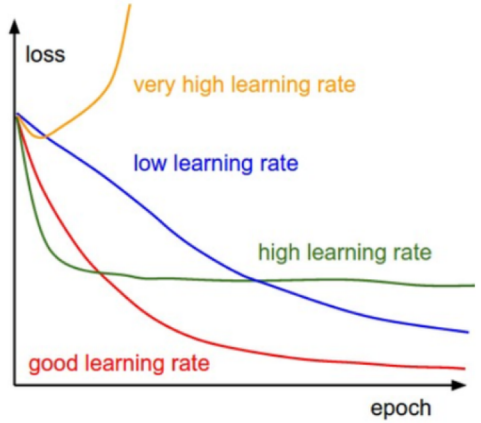


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

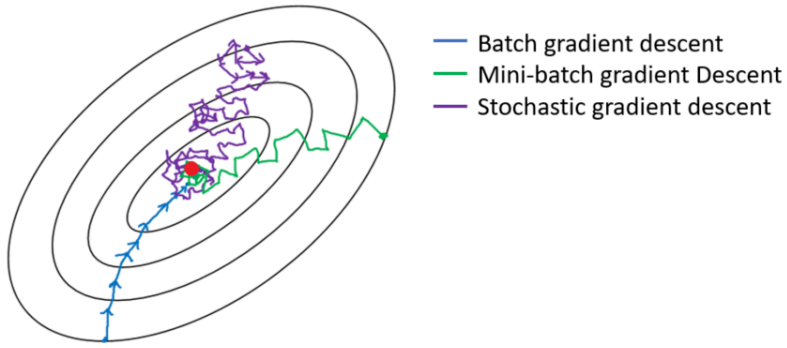
Algorithm Hyperparameters



Regularization coefficient



Learning Rate



Batch size

Setting Hyperparameters

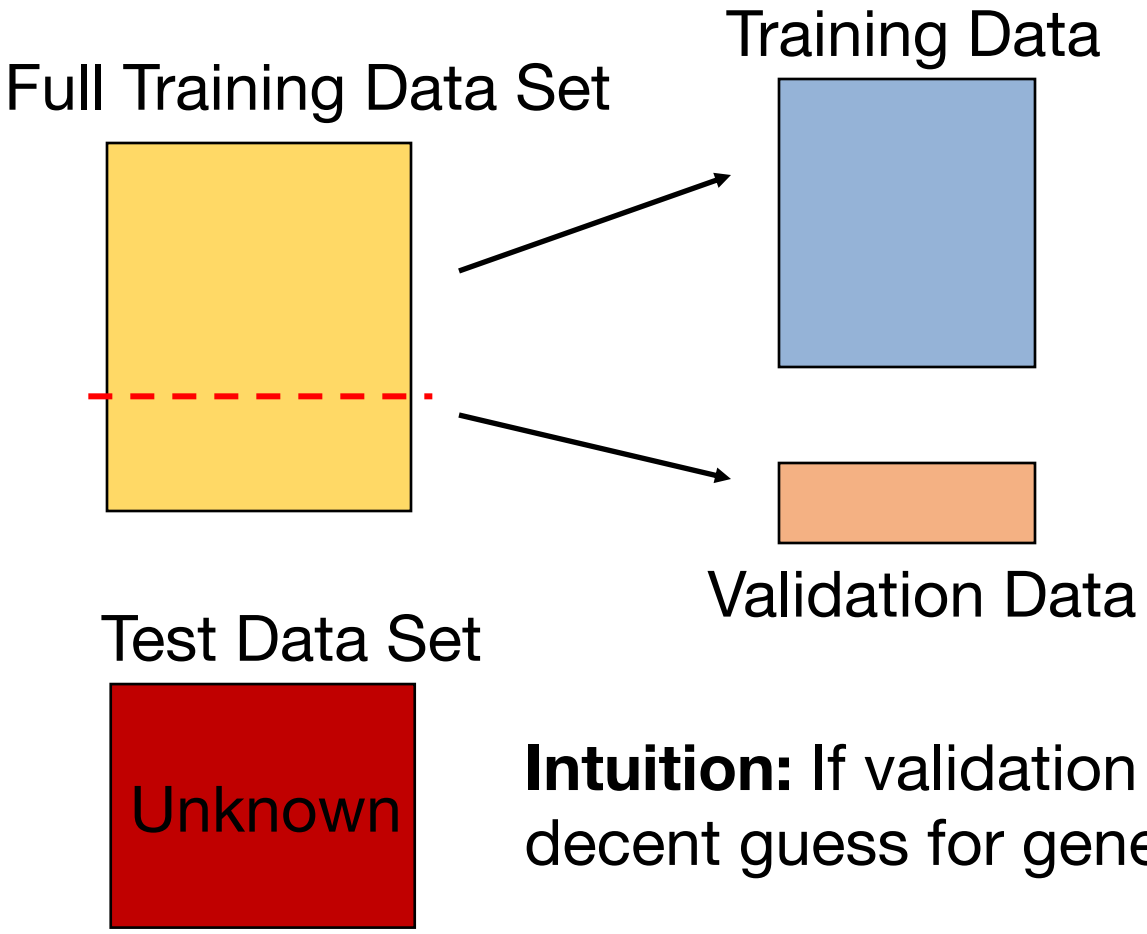
- Hyperparameters are critical for any ML model. How do we optimize them?
- Gradient descent is not always feasible:
  - Gradients may not exist. E.g., hypothesis class
  - Gradient descent might be themselves using hyperparameters. E.g., learning rate
- **Model selection:** Best hyperparameters are ones that help generalize. Use validation loss to choose best ones/

Model Selection

- Say we have two classifiers, *C1* and *C2*, and want to choose the best one to use for future predictions
- Can we use training accuracy to choose between them?
- No!

  E.g., an n degree polynomial can easily get 0 training loss on a dataset of size n but may not be best
- Instead, choose based on test accuracy...
- …but we don't have test data
- Can we pretend to have test data?

Training, Validation, Test Data



Idea:
Train each model on a subset of full training data...

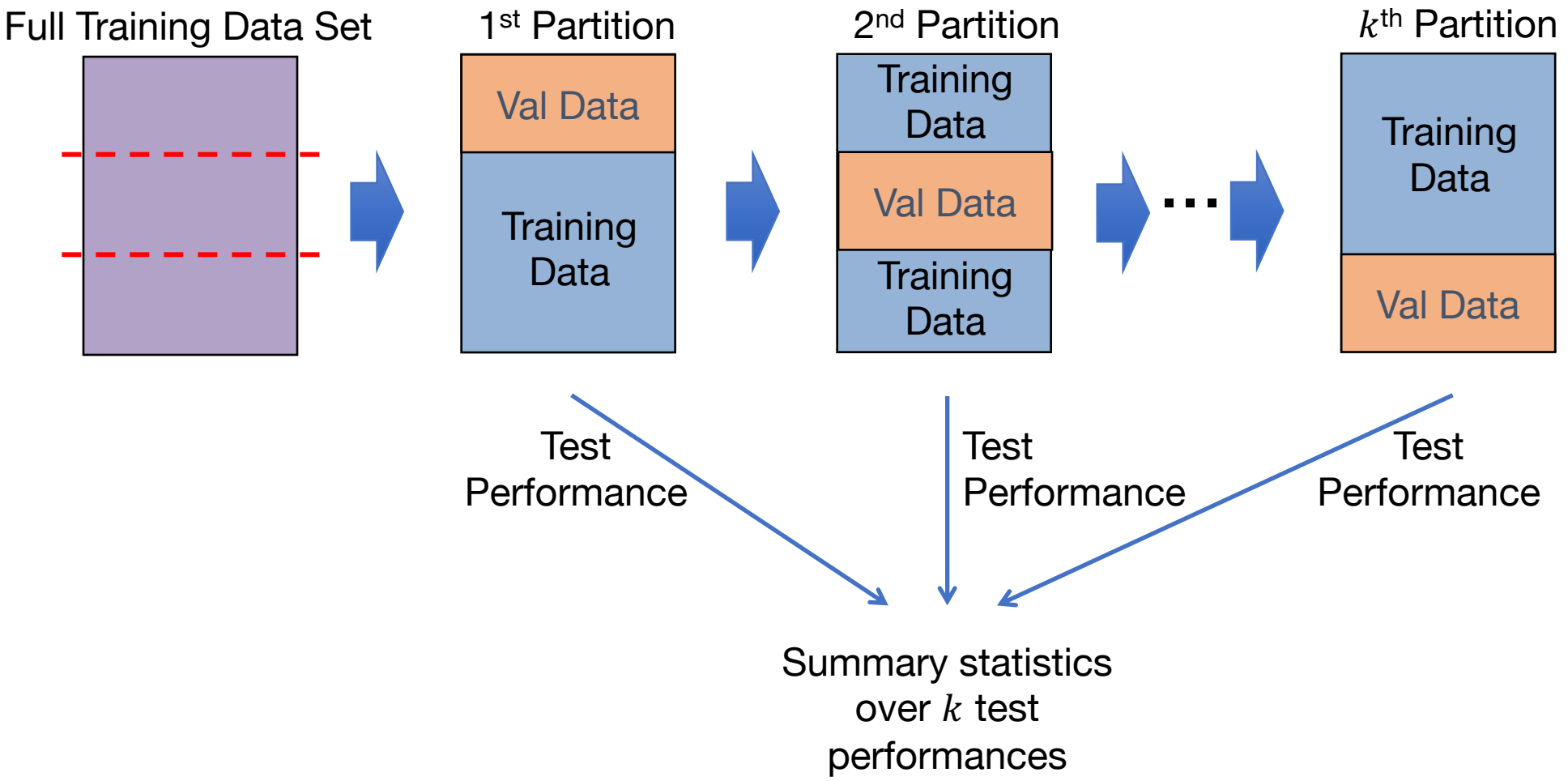
...and then evaluate each model's accuracy on the **validation data** a.k.a. held-out data

Intuition: If validation data ~ test data, then we have a decent guess for generalization

k-Fold Cross-Validation

- Why just choose one particular "split" of the data?
  - In principle, we should do this multiple times since performance may be different for each split
- **k-Fold Cross-Validation** (e.g., $k = 10$)
  - randomly partition full data set of n instances into k disjoint subsets (each roughly of size n/k)
  - Choose each fold in turn as the test set; train model on the other folds and evaluate
  - Compute statistics over k test performances, or choose best of the k models
  - Can also do "leave-one-out CV" where $k = n$

Example: 3-Fold CV



Summary

Linear Basis Function Regression

- Extends ordinary linear regression to (non-linear) complex transformations of input features

Generalization

- More complex hypothesis leads to overfitting. Less complex hypothesis can result in underfitting
- Validation splits help evaluate generalization

Loss regularization

- Aids generalization by showing preference for *simple* hypothesis (e.g., small ℓ_2 norm)

Hyperparameters

- Important design decisions for generalization. Set via validation dataset.