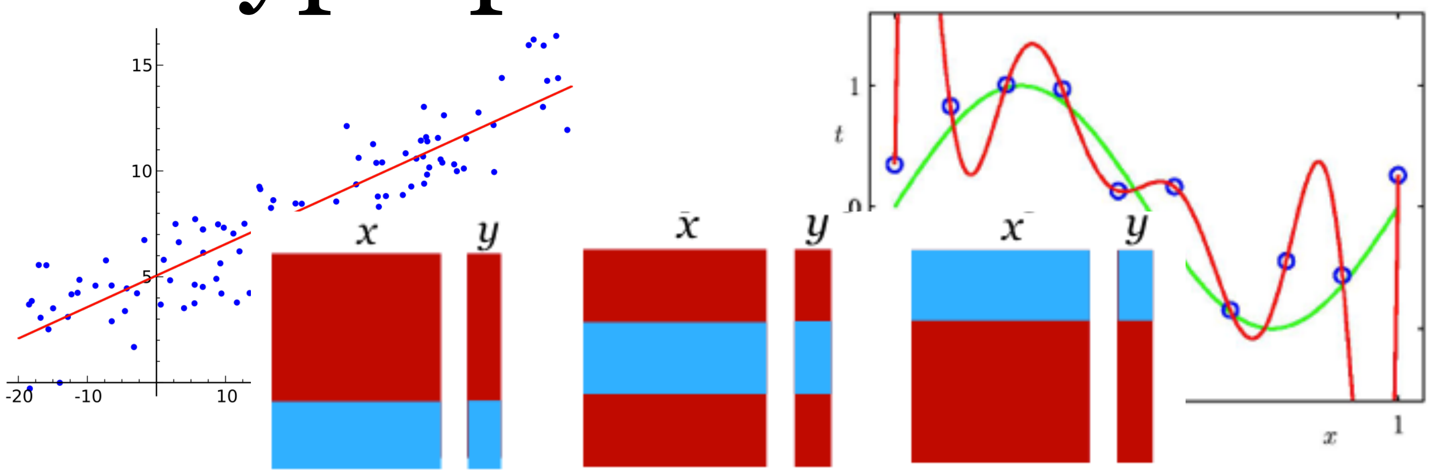
Tufts COMP 135: Introduction to Machine Learning

https://www.cs.tufts.edu/comp/135/2020f/

Linear Regression with Polynomial Features, Cross Validation, and Hyperparameter Selection



Many slides attributable to:
Erik Sudderth (UCI)
Finale Doshi-Velez (Harvard)
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Prof. Mike Hughes

Objectives for Today (day 04)

- Regression with transformations of features
  - Especially, polynomial features

- Ways to estimate generalization error
  - Fixed Validation Set
  - K-fold Cross Validation

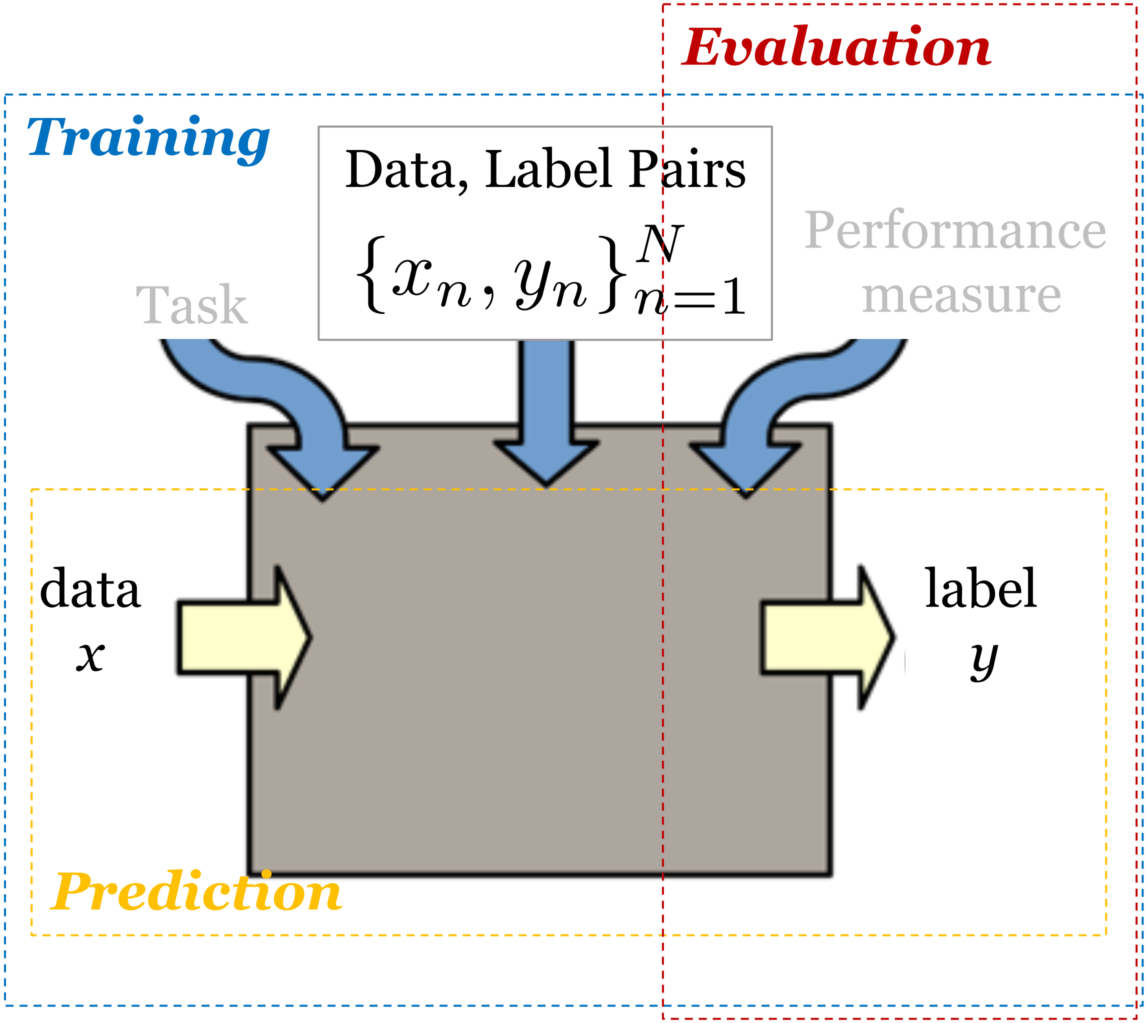
- Hyperparameter Selection

What will we learn?

Supervised Learning

Unsupervised Learning

Reinforcement Learning



Task: Regression

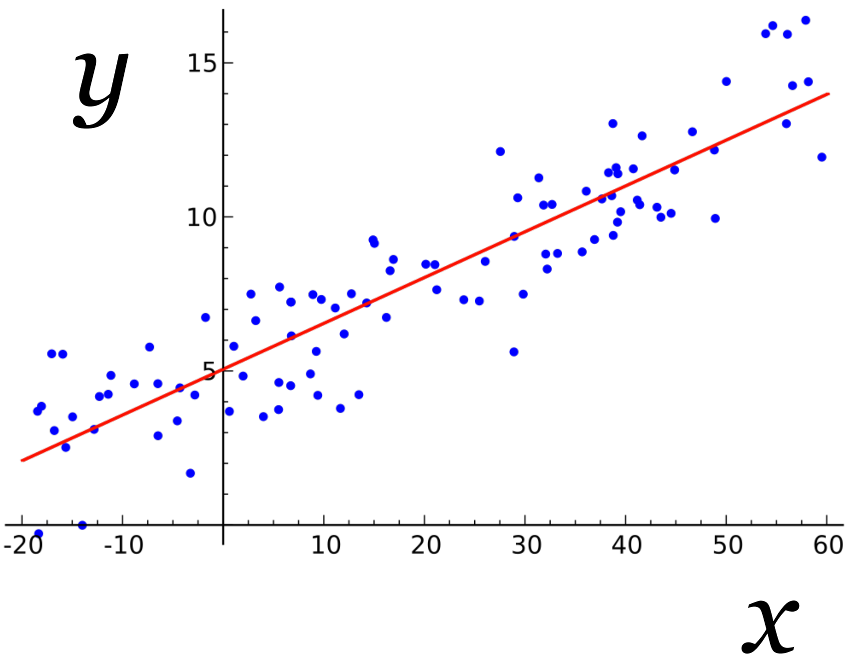
Supervised Learning

regression

Unsupervised Learning

Reinforcement Learning

y is a numeric variable
e.g. sales in $$



Review: Linear Regression

Optimization problem: “Least Squares”

$$\min_{w,b} \sum_{n=1}^{N} \Big(y_n - \hat{y}(x_n, w, b) \Big)^2$$

Exact formula for optimal values of w, b exist!

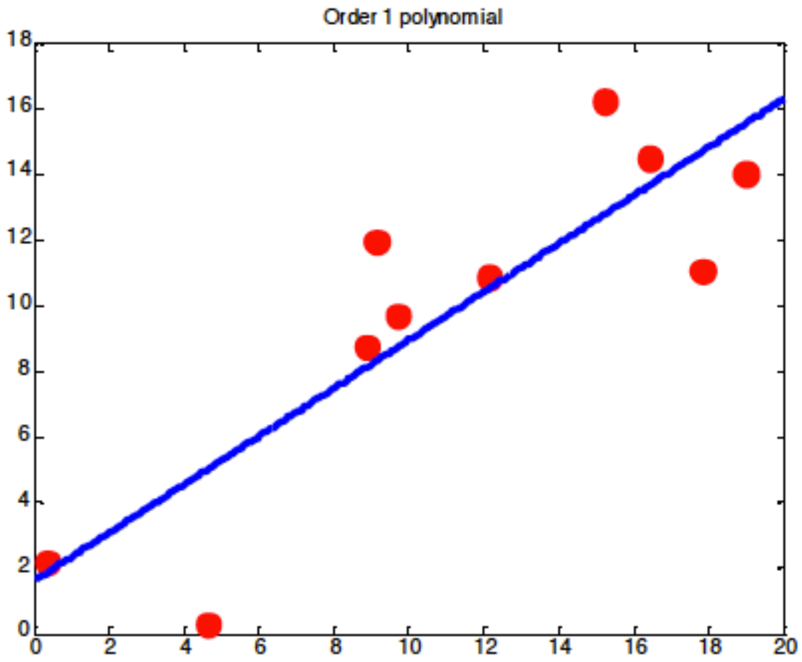
$$\tilde{X} = \begin{bmatrix} x_{11} & \dots & x_{1F} & 1 \\ x_{21} & \dots & x_{2F} & 1 \\ & & \dots & \\ x_{N1} & \dots & x_{NF} & 1 \end{bmatrix}$$

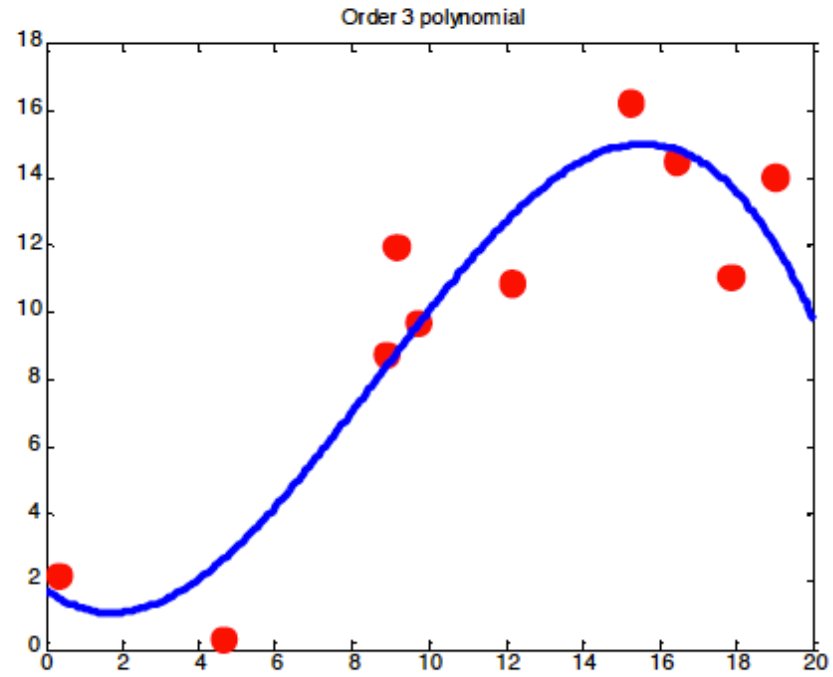
$$[w_1 \dots w_F \;\; b]^T = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y$$

Math works in 1D and for many dimensions

Transformations of Features

Fitting a line isn't always ideal





(c) Alexander Ihler

Can fit **linear** functions to **nonlinear** features

A nonlinear function of x:

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

Can be written as a linear function of $\phi(x_i) = [1 \ x_i \ x_i^2 \ x_i^3]$

$$\hat{y}(x_i) = \sum_{g=1}^{4} \theta_g \phi_g(x_i) = \theta^T \phi(x_i)$$

"Linear regression" means linear in the parameters (weights, biases)

Features can be arbitrary transforms of raw data

What feature transform to use?

- Anything that works for your data!
  - sin / cos for periodic data
  - polynomials for high-order dependencies

$$\phi(x_i) = [1 \; x_i \; x_i^2 \ldots]$$

  - interactions between feature dimensions

$$\phi(x_i) = [1 \quad x_{i1}x_{i2} \quad x_{i3}x_{i4} \ldots]$$

  - Many other choices possible

Linear Regression with Transformed Features

$$\phi(x_i) = [1 \ \phi_1(x_i) \ \phi_2(x_i) \ldots \phi_{G-1}(x_i)]$$

$$\hat{y}(x_i) = \theta^T \phi(x_i)$$

Optimization problem: "Least Squares"

$$\min_\theta \sum_{n=1}^N (y_n - \theta^T \phi(x_i))^2$$

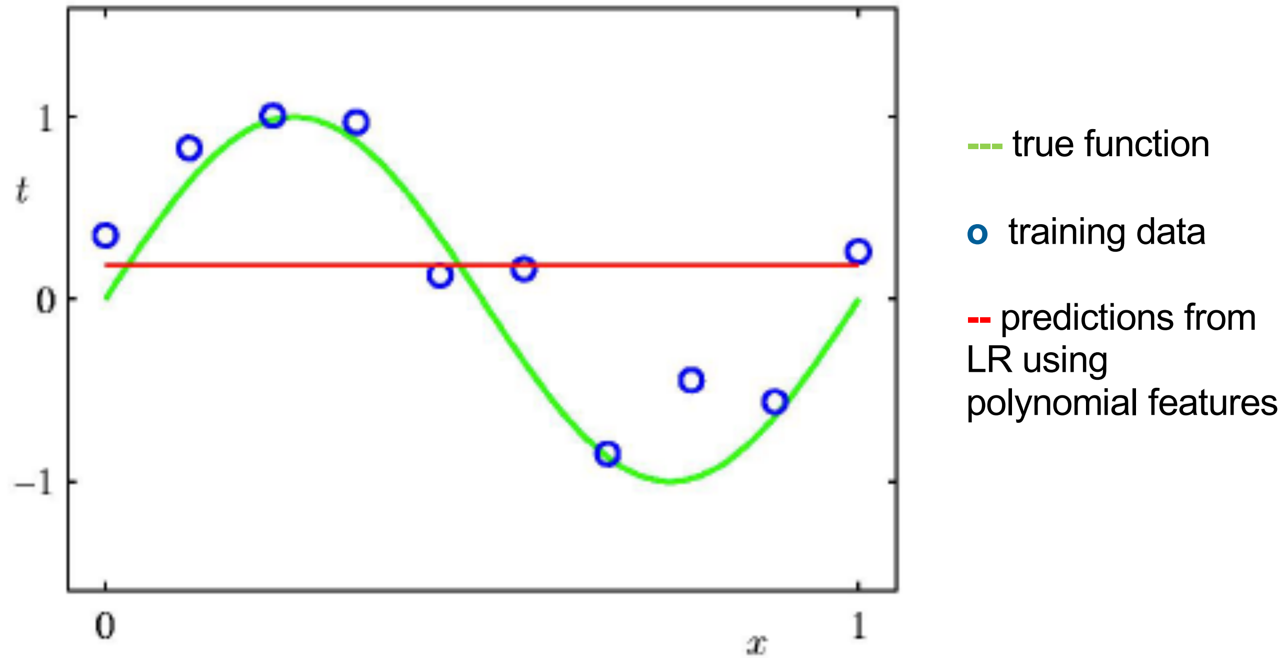
Exact solution:

$$\Phi = \begin{bmatrix} 1 & \phi_1(x_1) & \ldots & \phi_{G-1}(x_1) \\ 1 & \phi_1(x_2) & \ldots & \phi_{G-1}(x_2) \\ \vdots & & \ddots & \\ 1 & \phi_1(x_N) & \ldots & \phi_{G-1}(x_N) \end{bmatrix}$$

N x G matrix

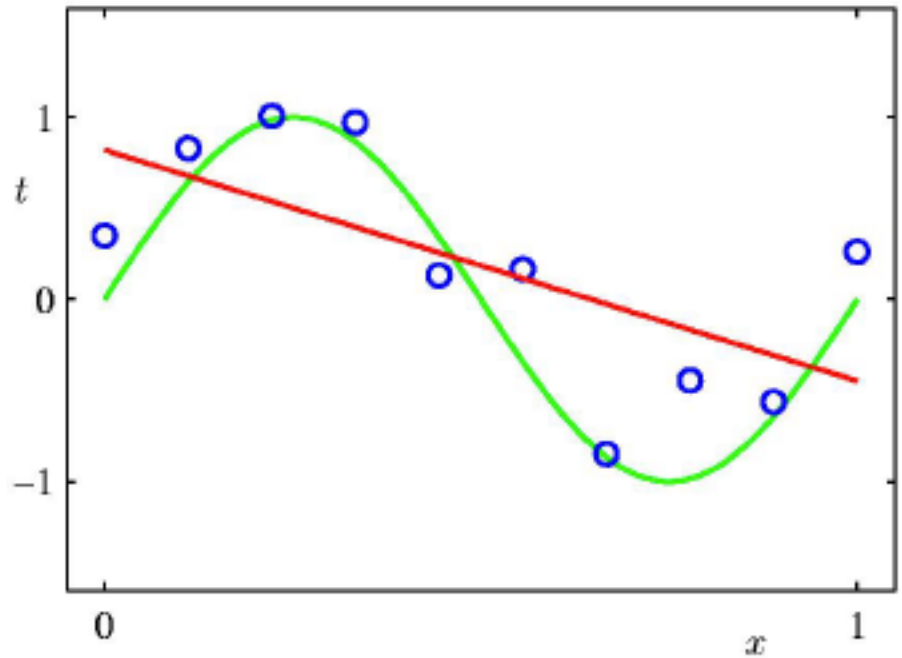
$$\theta^* = (\Phi^T \Phi)^{-1} \Phi^T y$$

0^{th} degree polynomial features



Credit: Slides from course by Prof. Erik Sudderth (UCI)

1^{st} degree polynomial features



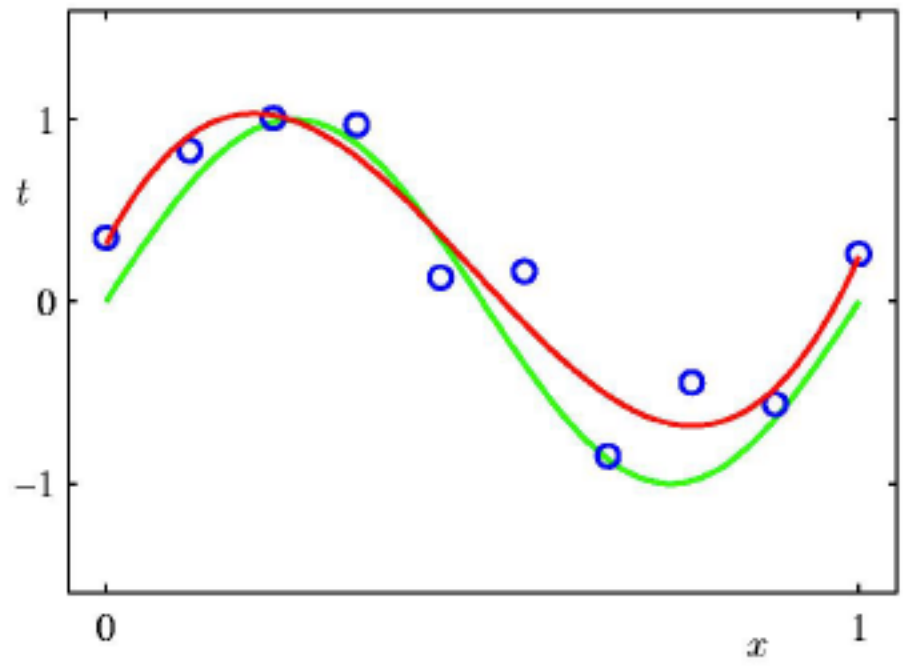
--- true function

o training data

-- predictions from LR using polynomial features

Credit: Slides from course by Prof. Erik Sudderth (UCI)

3rd degree polynomial features



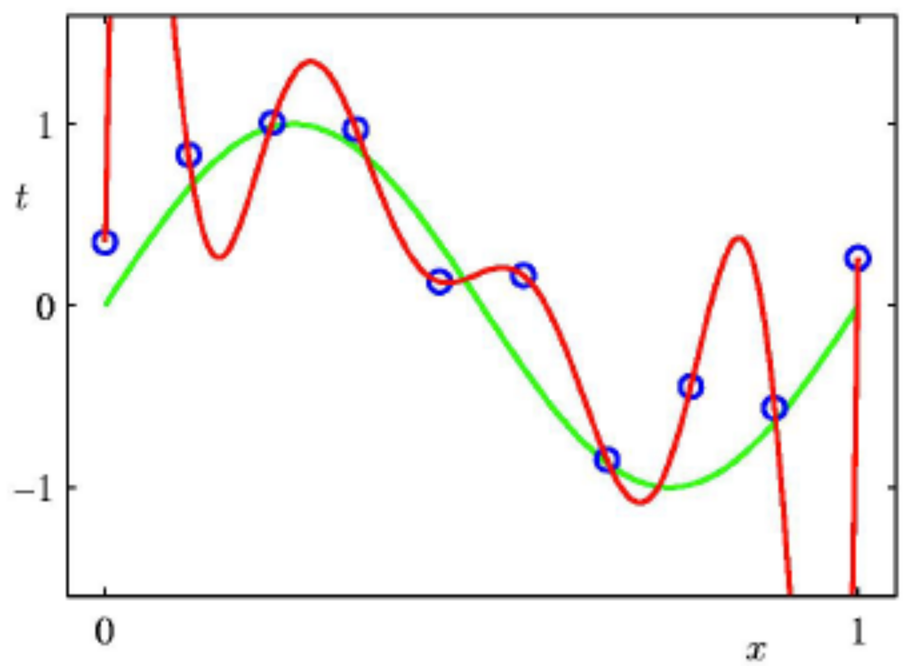
--- true function

o training data

-- predictions from LR using polynomial features

Credit: Slides from course by Prof. Erik Sudderth (UCI)

9^{th} degree polynomial features



--- true function

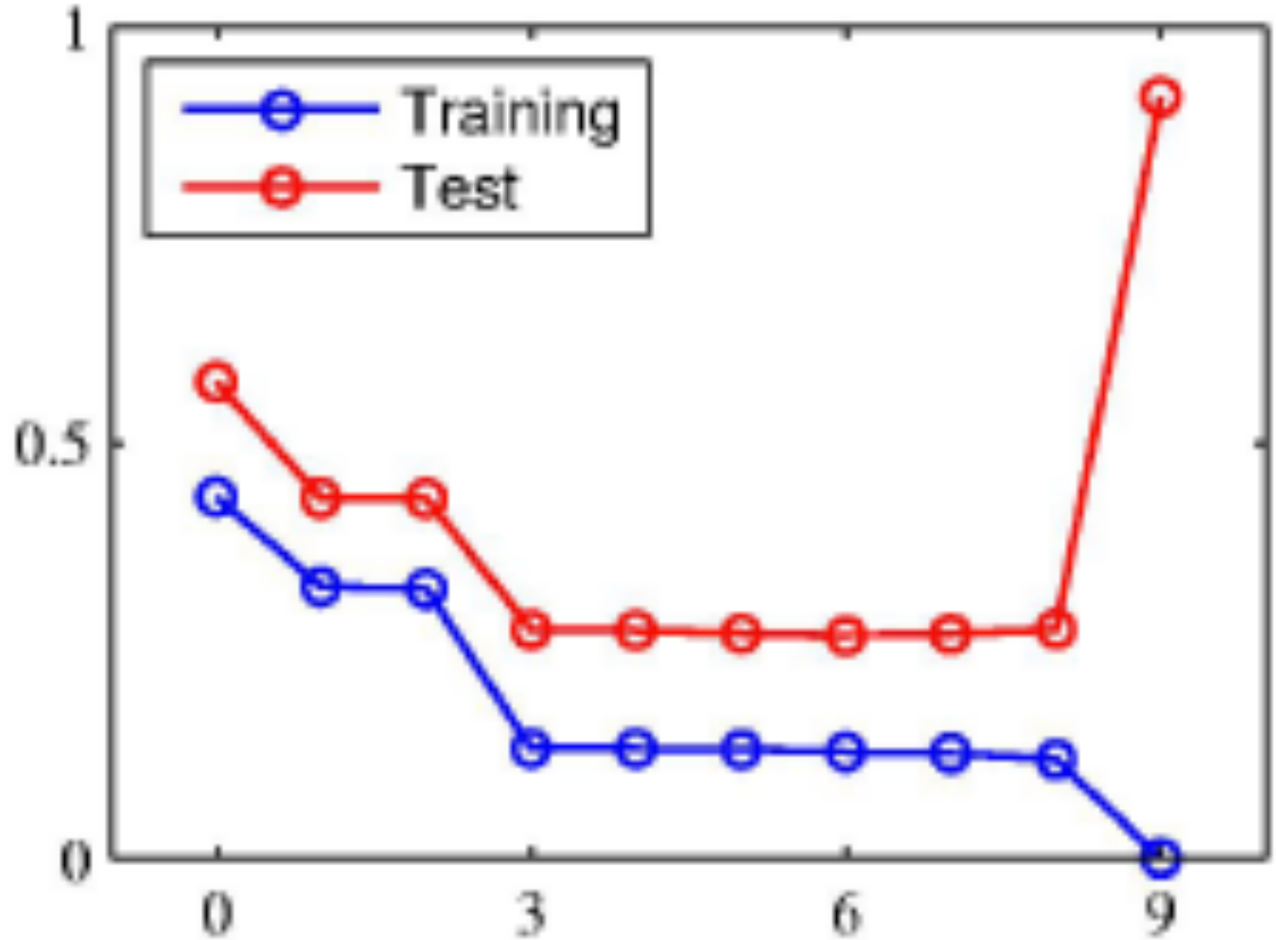
o training data

-- predictions from LR using polynomial features

Credit: Slides from course by Prof. Erik Sudderth (UCI)

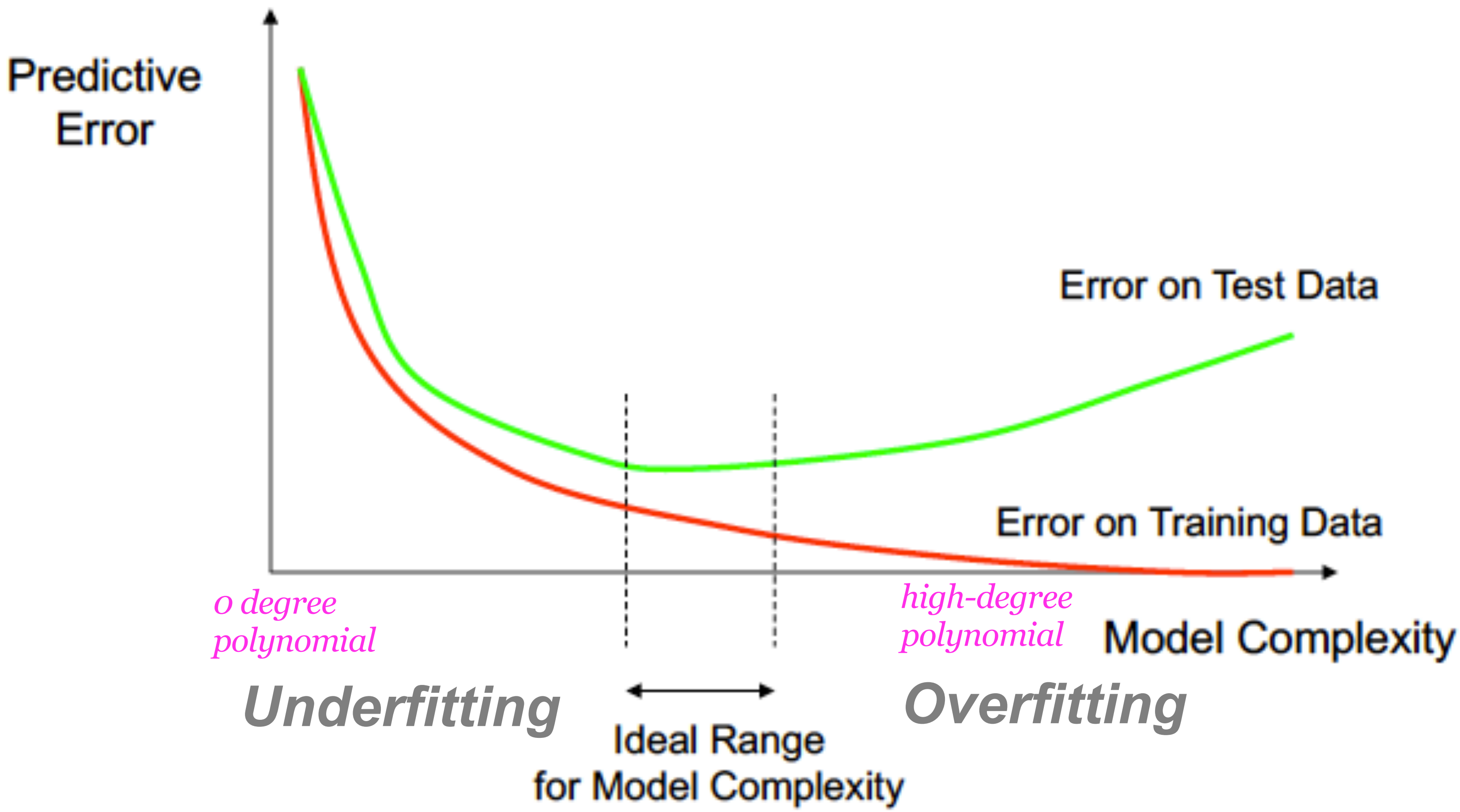
Error vs Degree

mean
squared
error



polynomial degree

Error vs Model Complexity



What to do about underfitting?

Increase model complexity (add more features!)

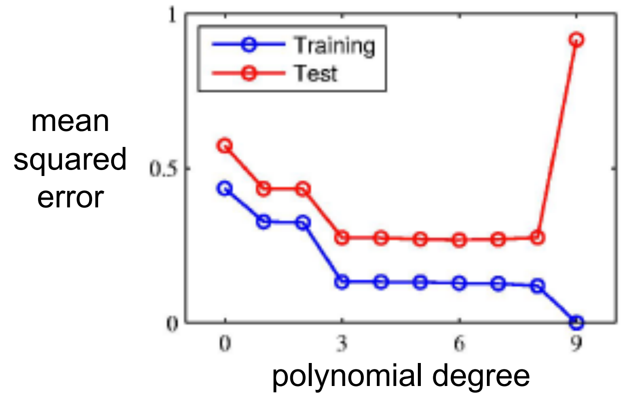
What to do about overfitting?

Select among several complexity levels the one that *generalizes* best (today)

Control complexity with a penalty in training objective (next class)

Hyperparameter Selection

Selection problem: What polynomial degree to use?



If we picked lowest training error, we'd select a 9-degree polynomial

If we picked lowest test error, we'd select a 3 or 4 degree polynomial

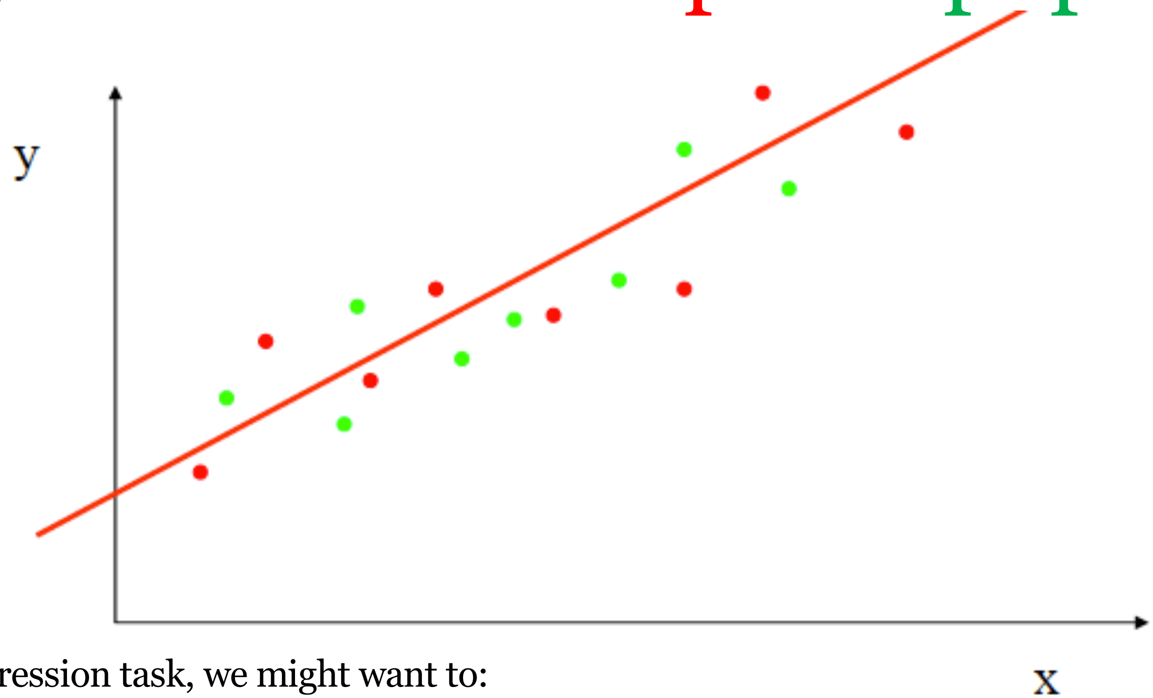
"**Parameter**" (e.g. weight values in linear regression)

a numerical variable controlling quality of fit that we can effectively estimate by minimizing error on training set

"**Hyperparameter**" (e.g. degree of polynomial features)

a numerical variable controlling model complexity / quality of fit whose value we cannot effectively estimate from the training set

Goal of regression (supervised ML) is to **generalize**: sample to population

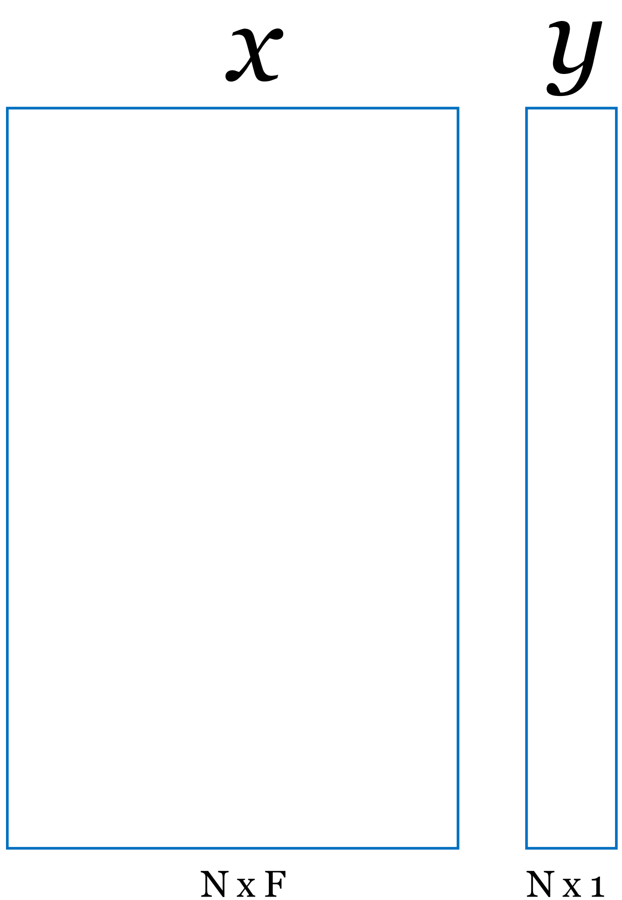


For any regression task, we might want to:
- Train a model (estimate parameters)
  - Requires calling `fit` on a *training* labeled dataset
- Select hyperparameters (e.g. which degree of polynomial?)
  - Requires evaluating predictions on a *validation* labeled dataset
- Report its ability on data it has never seen before ("generalization error" or "test error")
  - Requires comparing predictions to a *test* labeled dataset

Should ALWAYS use different labeled datasets to do each of these things!

Two Ways to Measure Generalization Error
- Fixed Validation Set
- Cross-Validation

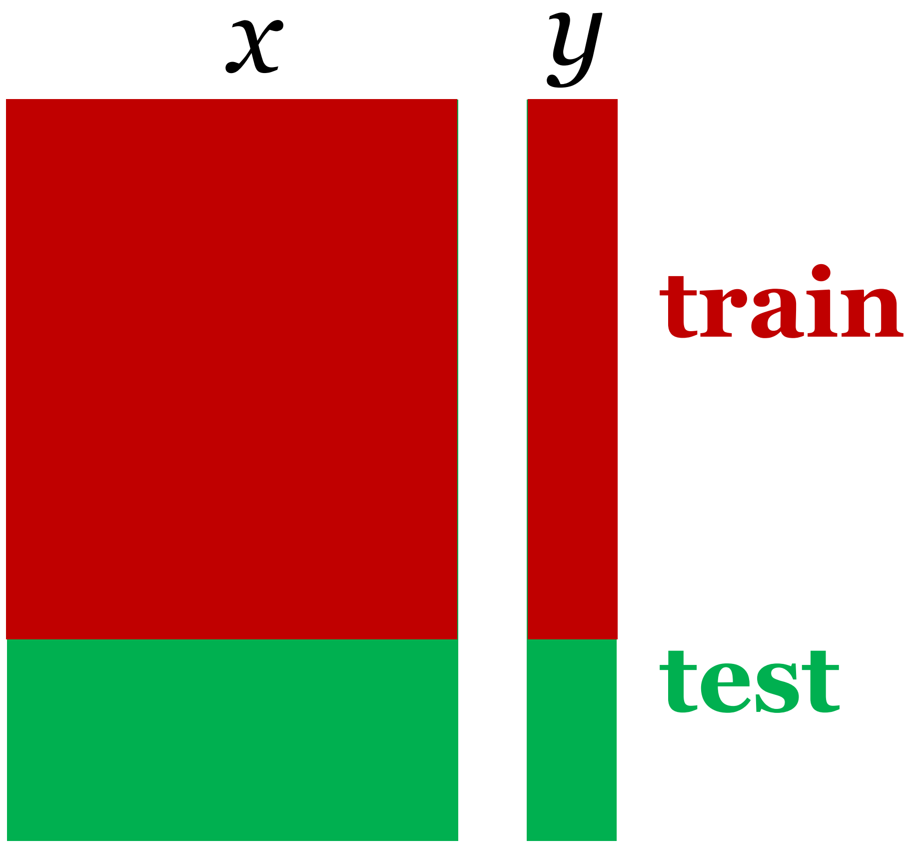
Labeled dataset



Each row represents one example

Assume rows are arranged "uniformly at random" (order doesn't matter)

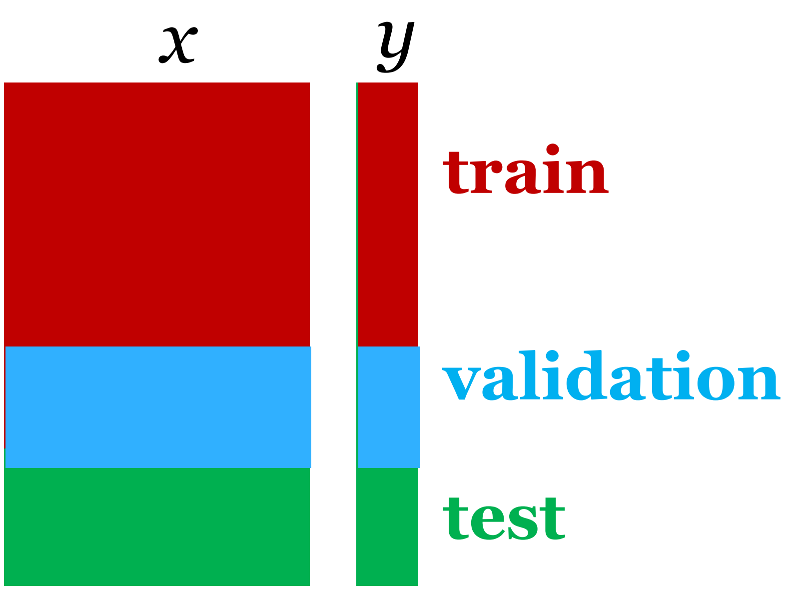
Split into train and test



Selection via Fixed Validation Set

Option: Fit on train, select on **validation**

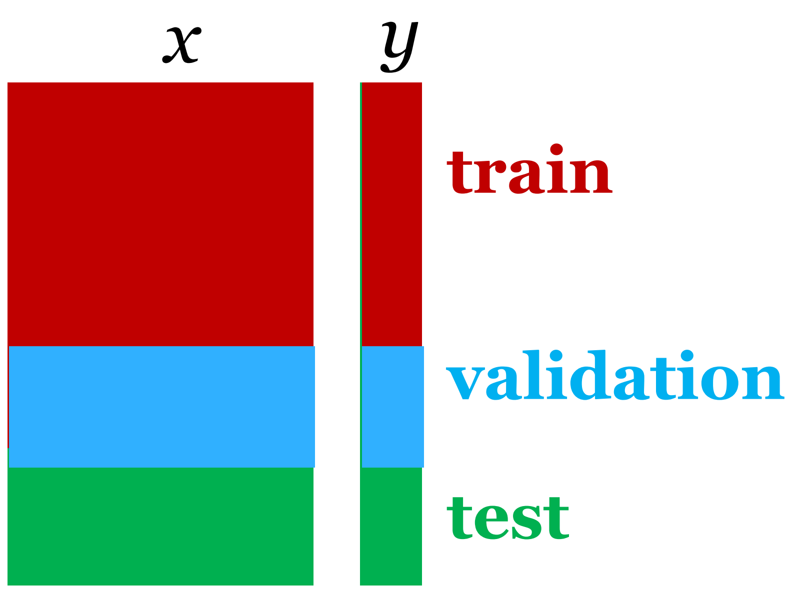
1) Fit each model to **training** data
2) Evaluate each model on **validation** data
3) Select model with lowest **validation** error
4) Report error on **test** set



Selection via Fixed Validation Set

Option: Fit on train, select on **validation**

1) Fit each model to **training** data
2) Evaluate each model on **validation** data
3) Select model with lowest **validation** error
4) Report error on **test** set



Concerns

- What sizes to pick?
- Will train be too small?
- Is validation set used effectively? (only to evaluate predictions?)

For small datasets, randomness in validation split **will impact selection**

Single random split *10 other random splits*

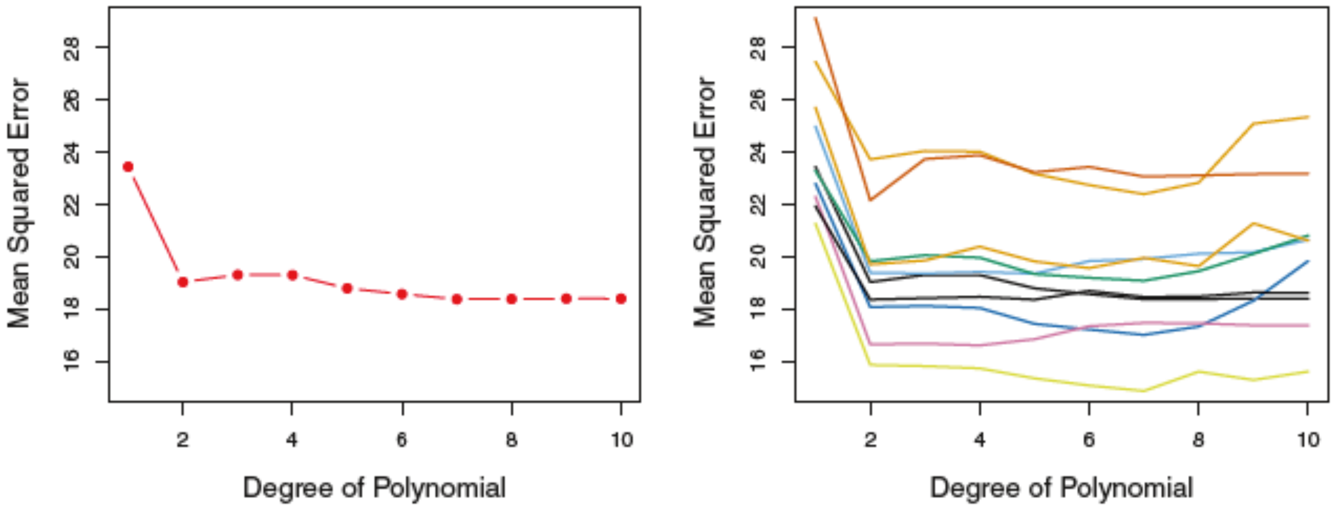
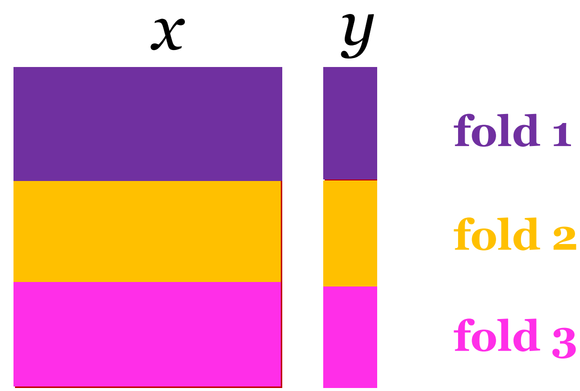


FIGURE 5.2. *The validation set approach was used on the* Auto *data set in order to estimate the test error that results from predicting* mpg *using polynomial functions of* horsepower. Left: *Validation error estimates for a single split into training and validation data sets.* Right: *The validation method was repeated ten times, each time using a different random split of the observations into a training set and a validation set. This illustrates the variability in the estimated test MSE that results from this approach.*

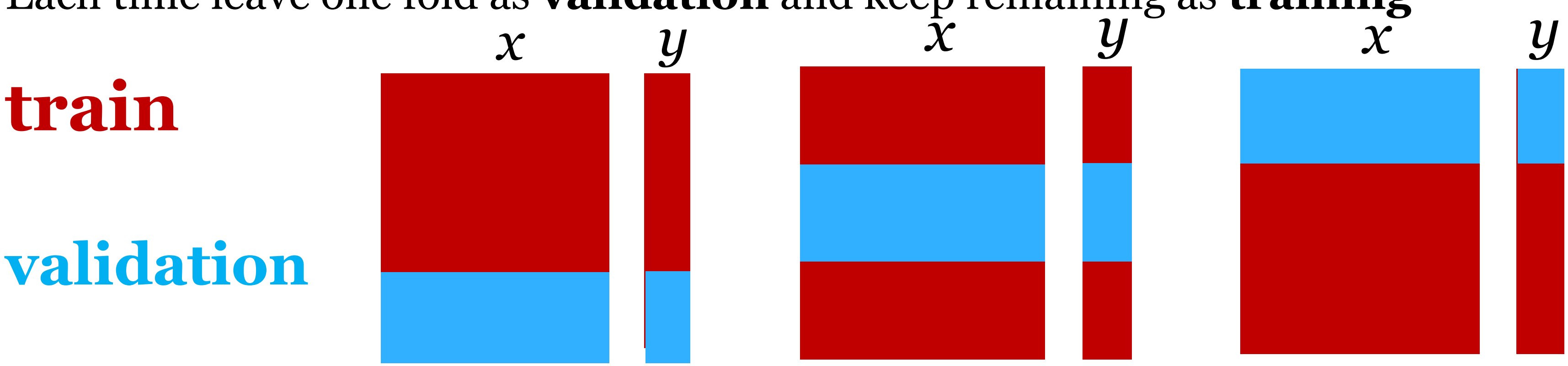
Credit: ISL Textbook, Chapter 5

3-fold Cross Validation

Divide labeled dataset into 3 even-sized parts



Fit model 3 independent times.
Each time leave one fold as **validation** and keep remaining as **training**



Heldout error estimate: average of the validation error across all 3 fits

K-fold CV: How many folds *K*?

- Can do as low as 2 fold
- Can do as high as N-1 folds ("Leave one out")
- Usual rule of thumb: 5-fold or 10-fold CV

- Computation runtime **scales linearly** with *K*
  - Larger K also means each fit uses more train data, so each fit might take longer too

- Each fit is independent and **parallelizable**

Estimating Heldout Error with Cross Validation

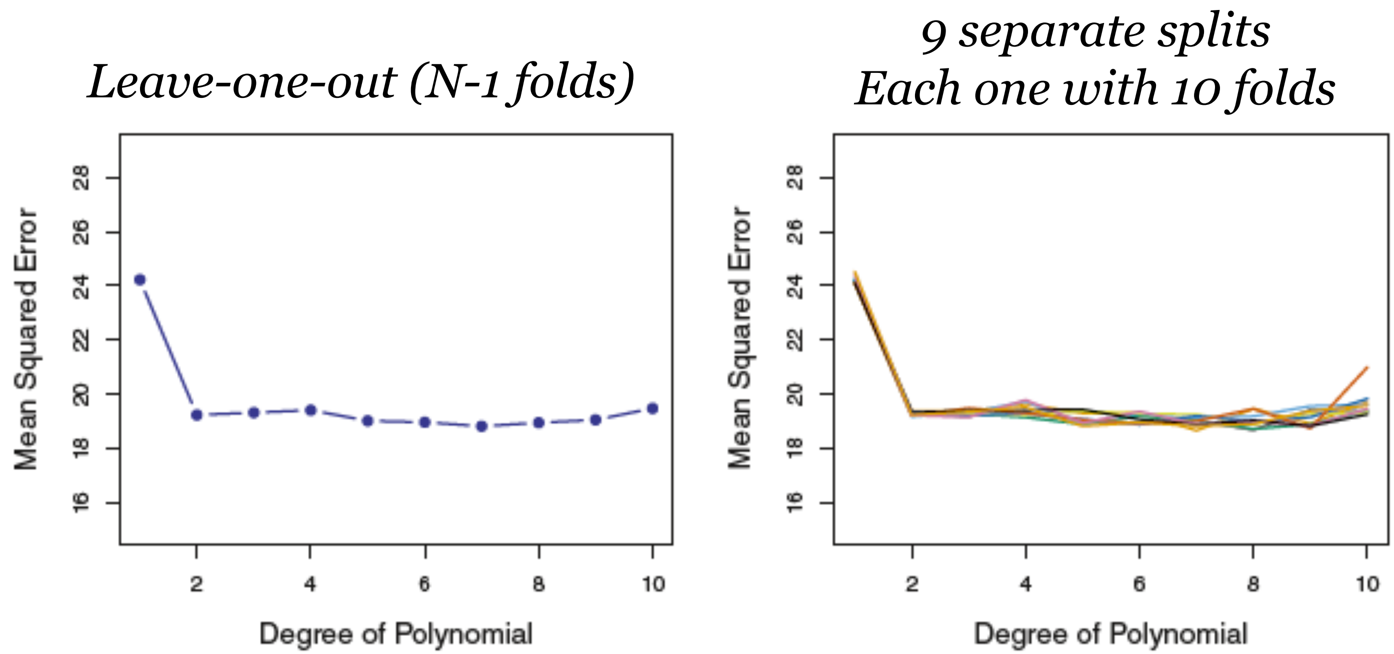


FIGURE 5.4. *Cross-validation was used on the* `Auto` *data set in order to estimate the test error that results from predicting* `mpg` *using polynomial functions of* `horsepower`. Left: *The LOOCV error curve.* Right: 10-*fold CV was run nine separate times, each with a different random split of the data into ten parts. The figure shows the nine slightly different CV error curves.*

Credit: ISL Textbook, Chapter 5