Unit 3
Model-Order Selection

Prof. Phil Schniter

The Ohio State University

ECE 4300: Introduction to Machine Learning, Sp20

Learning objectives

- Understand the problem of model-order selection
- Visually identify underfitting and overfitting in a scatterplot
- Understand the need to partition data into training and testing subsets
- Understand the K-fold cross-validation process
  - Use it to assess the test error for a given model
  - Use it to select the model order
- Understand the concepts of bias, variance, and irreducible error
  - Know how to compute each from synthetically generated data
  - Understand the bias-variance tradeoff

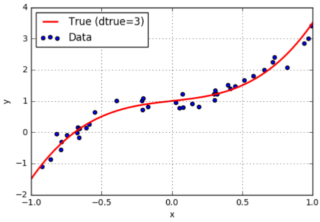
Outline

Polynomial regression

- Recall polynomial regression from last lecture
- Given data $\{(x_i, y_i)\}_{i=1}^n$, model target y as

$$y \approx \beta_0 + \beta_1 x + \cdots \beta_d x^d$$

  - model parameters $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_d]^\mathsf{T}$
  - d is the degree of the polynomial
  - given d, can fit $\boldsymbol{\beta}$ using least-squares (multiple linear regression with $x_j \triangleq x^j$)

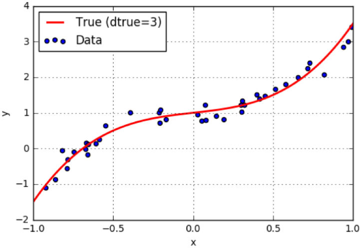


- Question: Can we select d from the data?
  - An instance of "model-order selection"

Example with synthetic data

- We consider synthetic data generated using a noisy polynomial model
- $\{x_i\}$: 40 samples uniformly distributed in interval $[-1,1]$
- $\{y_i\}$: generated as $y_i = f(x_i) + \epsilon_i$
  - $f(x) = \beta_0 + \beta_1 x + \cdots \beta_d x^d$ with $d=3$ for some "true" coefficients $\{\beta_j\}_{j=0}^3$
  - noise $\epsilon_i \sim \mathcal{N}(0,\sigma^2)$ & independent over i
- Synthetic data is useful for analysis and experimentation
  - We know the "ground truth"
  - Thus we can assess the performance of various estimators

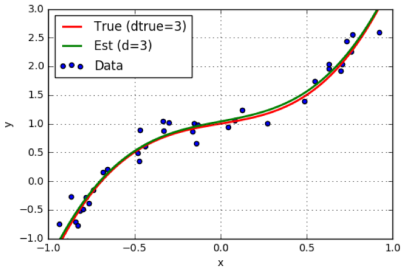
```
# Import useful polynomial library
import numpy.polynomial.polynomial as poly

# True model parameters
beta = np.array([1,0.5,0,2])   # coefficients
wstd = 0.2                     # noise
dtrue = len(beta)-1            # true poly degree

# Independent data
nsamp = 40
xdat = np.random.uniform(-1,1,nsamp)

# Polynomial
y0 = poly.polyval(xdat,beta)
ydat = y0 + np.random.normal(0,wstd,nsamp)
```



Fitting with the true model order

- Could implement multiple linear regression with $x_j \triangleq x^j$
- Shortcut: numpy.polynomial package

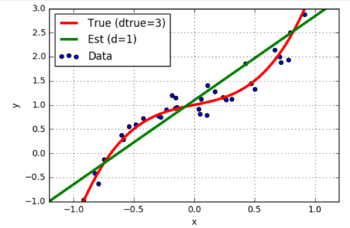
```
d = 3
beta_hat = poly.polyfit(xdat,ydat,d)

# Plot true and estimated function
xp = np.linspace(-1,1,100)
yp = poly.polyval(xp,beta)
yp_hat = poly.polyval(xp,beta_hat)
plt.xlim(-1,1)
plt.ylim(-1,3)
plt.plot(xp,yp,'r-',linewidth=2)
plt.plot(xp,yp_hat,'g-',linewidth=2)

# Plot data
plt.scatter(xdat,ydat)
plt.legend(['True (dtrue=3)', 'Est (d=3)', 'Data'], loc='upper left')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
```



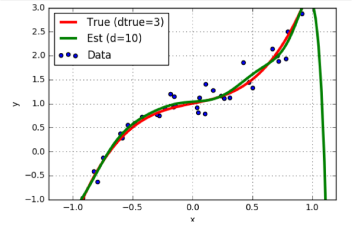
- First, let's assume the true polynomial order, $d = 3$, is known
  - Get a very good fit!

Fitting with the wrong model order

$d = 1$: "underfitting"

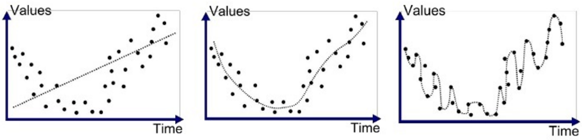


$d = 10$: "overfitting"



another illustration:

https://medium.com/greyatom



Underfitted Good Fit/Robust Overfitted

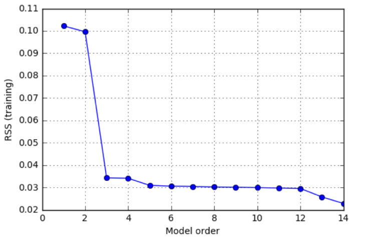
Is there a way to estimate the true d from the data $\{(x_i, y_i)\}_{i=1}^{n}$?

Select model-order that minimizes RSS?

Simple idea:

- For each hypothesized model order d...
  - Compute LS coefficients $\boldsymbol{\beta}_{\mathsf{ls}} \in \mathbb{R}^{d+1}$
  - Predict the targets: $\widehat{y}_i = \boldsymbol{\beta}_{\mathsf{ls}}^{\mathsf{T}} \boldsymbol{x}_i$
  - Compute $\mathrm{RSS}(d) = \sum_{i=1}^{n} (y_i - \widehat{y}_i)^2$

  Finally, pick the d that minimizes $\mathrm{RSS}(d)$
- This doesn't work!
  - $\mathrm{RSS}(d)$ monotonically decreases with d
  - Suggests to choose d as large as possible
  - Leads to overfitting
- What is the problem?



Overfitting

Here is why we can't use training RSS to select model order:

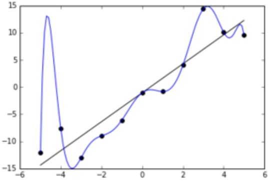
- In effect, we are choosing from a nested set of models

$$\begin{aligned} \boldsymbol{\beta} &= [\beta_0, \beta_1, 0, 0, 0, \dots]^{\mathsf{T}} && \text{when } d = 1 \\ \boldsymbol{\beta} &= [\beta_0, \beta_1, \beta_2, 0, 0, \dots]^{\mathsf{T}} && \text{when } d = 2 \\ \boldsymbol{\beta} &= [\beta_0, \beta_1, \beta_2, \beta_3, 0, \dots]^{\mathsf{T}} && \text{when } d = 3 \\ &\vdots && \vdots \end{aligned}$$

  and thus each model is a special case of the next model.

- The training RSS *can get no worse* as the model becomes more general, and so we will always choose the most general/complex model
- When d is large, $\widehat{y}_i$ models the training noise ϵ_i
  - This is the main characteristic of overfitting!
- When $d \geq n-1$, the training $\text{RSS}(d)$ is zero:

$$\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}_{\mathsf{ls}}\|^2 = 0$$



https://en.wikipedia.org/wiki/Overfitting

Outline

- Motivating Example: Polynomial Degree Selection
- Cross-validation
- The Bias-Variance Tradeoff
- From Model-Order Selection to Feature Selection

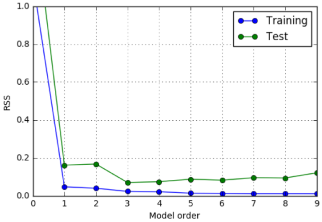
Cross-validation

Main idea:

Evaluate performance on "test data" that is independent of the training data

- Simplest version: Partition total dataset into two subsets:
  - n_{train} training samples: $\{(\boldsymbol{x}_{\text{train},i}, y_{\text{train},i})\}_{i=1}^{n_{\text{train}}}$
  - $n_{\text{test}} = n - n_{\text{train}}$ test samples: $\{(\boldsymbol{x}_{\text{test},i}, y_{\text{test},i})\}_{i=1}^{n_{\text{test}}}$
- Then, for each hypothesized model-order d...
  - Compute LS coefficients $\boldsymbol{\beta}_{\text{ls}}$ from training data
  - Predict the test targets: $\widehat{y}_{\text{test},i} = \boldsymbol{\beta}_{\text{ls}}^{\mathsf{T}} \boldsymbol{x}_{\text{test},i}$
  - Compute $\text{RSS}_{\text{test}}(d) = \sum_{i=1}^{n_{\text{test}}} (y_{\text{test},i} - \widehat{y}_{\text{test},i})^2$

  Finally, choose the d that minimizes $\text{RSS}_{\text{test}}(d)$



Finds true ↑ model-order $d = 3$!

K-fold cross-validation

More robust versions:

- K-fold cross validation
  - Split data into K folds
  - Use $K-1$ for training & 1 for test
  - Repeat for each of K possible test sets
  - Typically use $K = 5$ or 10
  - Expensive: requires K parameter fits
  - Good approx of true performance!
- Leave-one-out cross validation (LOOCV)
  - Extreme case where $K = n$ (i.e., test set contains 1 sample!)
  - Most accurate, but most expensive
  - Advantageous when n is small

https://medium.com/@sebastiannorena

K-fold cross-validation with sklearn

- CV approach:
  - Outer loop: over K folds
  - Inner loop: over D model-orders
  - Compute test $\mathrm{RSS}_{d,k}$ for each order d & fold k
  - Average test errors across K folds to get $\overline{\mathrm{RSS}}_d \triangleq \frac{1}{K}\sum_{k=1}^{K} \mathrm{RSS}_{d,k}$
  - Choose d giving smallest $\overline{\mathrm{RSS}}_d$
- Use sklearn's KFold method to generate index sets for the folds!

```
# Create a k-fold object
k = 10
kfo = sklearn.model_selection.KFold(n_splits=k,shuffle=True)

# Model orders to be tested
dtest = np.arange(0,10)
nd = len(dtest)

RSSts = np.zeros((nd,k))

# Loop over the folds
for isplit, Ind in enumerate(kfo.split(xdat)): # enumerate r

    # Get the training data in the split
    Itr, Its = Ind
    #kfo.split( ) produced Ind, which contains a pair of ind
    xtr = xdat[Itr]
    ytr = ydat[Itr]
    xts = xdat[Its]
    yts = ydat[Its]

    # Loop over the model order
    for it, d in enumerate(dtest):

        # Fit data on training data
        beta_hat = poly.polyfit(xtr,ytr,d)

        # Measure RSS on test data
        yhat = poly.polyval(xts,beta_hat)
        RSSts[it,isplit] = np.mean((yhat-yts)**2)
```

Accuracy of K-fold cross-validation

- Problem: $\overline{\text{RSS}}_d = \frac{1}{K}\sum_{k=1}^{K} \text{RSS}_{d,k}$ may inaccurately estimate RSS_d when K is small

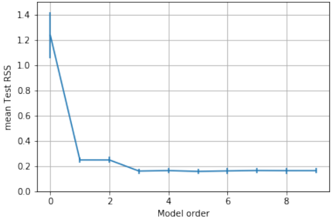
- Can measure the estimation accuracy of $\overline{\text{RSS}}_d$ using the standard error (SE):

$$\text{SE}_d \triangleq \frac{\text{std}(\text{RSS}_d)}{\sqrt{K}}, \text{ where}$$

$$\text{std}(\text{RSS}_d) = \sqrt{\frac{1}{K-1}\sum_{k=1}^{K}(\text{RSS}_{d,k} - \overline{\text{RSS}}_d)^2}$$

- Above, $\frac{1}{K-1}$ gives an "unbiased" variance estimate

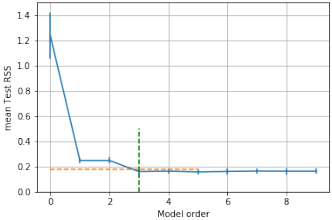
line shows $\overline{\text{RSS}}_d$, error bars show SE_d:

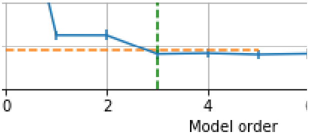



```
RSS_mean = np.mean(RSSts,axis=1) #note mean is taken over the
RSS_se   = np.std(RSSts,axis=1)/np.sqrt(k-1)
plt.errorbar(dtest, RSS_mean, yerr=RSS_se, fmt='-')
plt.ylim(0,1.5)
plt.xlabel('Model order')
plt.ylabel('mean Test RSS')
plt.grid()
```

The one-standard-error rule

- Previously, said to choose d minimizing $\overline{\mathrm{RSS}}_d$
  - But this sometimes overfits true model-order!
- Better approach: one-standard-error (OSE) rule
  - Use *simplest* model giving $\overline{\mathrm{RSS}}_d$ within one SE of minimum
- Detailed procedure:
  - Set $d_{\min} = \arg\min_d \overline{\mathrm{RSS}}_d$
  - Set $\overline{\mathrm{RSS}}_{\mathsf{tgt}} = \overline{\mathrm{RSS}}_{d_{\min}} + \mathrm{SE}_{d_{\min}}$
  - Find smallest d such that $\overline{\mathrm{RSS}}_d \leq \overline{\mathrm{RSS}}_{\mathsf{tgt}}$
- In example on right: $d_{\min} = 5$, and OSE selects $d = 3$, which is the true model-order





Outline

- Motivating Example: Polynomial Degree Selection
- Cross-validation
- The Bias-Variance Tradeoff
- From Model-Order Selection to Feature Selection

Statistical learning theory

- With degree-d polynomial regression, we saw that
  - choosing d too small causes underfitting
  - choosing d too large causes overfitting
  - d can be optimized by minimizing $\mathrm{RSS}_{\textsf{test}}$ through cross-validation
- This is special case of a more general concept:
  - a model that is too simple leads to underfitting
  - a model that is too complex leads to overfitting
  - model complexity can be optimized by minimizing mean-squared error, $\mathrm{MSE}_{\widehat{y}}$
- From a theoretical perspective,
  - analyzing $\mathrm{MSE}_{\widehat{y}}$ leads to the bias-variance equation
  - minimizing $\mathrm{MSE}_{\widehat{y}}$ involves a tradeoff between bias and variance

Statistical model

Setup for theoretical analysis...

- True model: $\boxed{y = f(\boldsymbol{x}) + \epsilon}$ for $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$
  - ϵ is a Gaussian random variable with mean zero and variance σ_ϵ^2
  - test ϵ assumed to be independent of trained $\boldsymbol{\beta}$
- Prediction: $\boxed{\widehat{y} = \widehat{f}(\boldsymbol{x}; \boldsymbol{\beta})}$ for some <u>random</u> $\boldsymbol{\beta}$
  - $\boldsymbol{\beta}$ was designed from some training data
  - $\boldsymbol{\beta}$ is random because the training data is random (i.e., random $\{\boldsymbol{x}_i\}$ and $\{\epsilon_i\}$)
  - The test feature-vector $\boldsymbol{x}$ is deterministic, and may be outside training set
- Mean-squared error on $\widehat{y}$ for given $\boldsymbol{x}$: $\boxed{\mathrm{MSE}_{\widehat{y}}(\boldsymbol{x}) \triangleq \mathbb{E}\left\{(y - \widehat{y})^2\right\}}$
  - $\mathbb{E}\{\cdot\}$ denotes statistical expectation: "the average value"
  - Here, the expectation is taken over test ϵ and trained $\boldsymbol{\beta}$

Expectation

- Intuitively, expectation $\mathbb{E}\{\cdot\}$ means "the average value"
- Formally, for any function $f(\cdot)$ and random variable $a \in \mathbb{R}$,

$$\mathbb{E}\{f(a)\} = \int_{-\infty}^{\infty} f(a)\, p(a)\, \mathrm{d}a \quad \text{for probability density function (pdf) } p(a)$$

  Vector-valued random variables (e.g., $\boldsymbol{a} \in \mathbb{R}^M$) can be handled similarly

- We will avoid formalities for now and focus on two key properties.
  For any functions $f(\cdot)$ & $g(\cdot)$, and random variables a & b:
  - $\mathbb{E}\{c + d\, f(a)\} = c + d\, \mathbb{E}\{f(a)\}$ for deterministic c, d (linearity)
  - $\mathbb{E}\{f(a)g(b)\} = \mathbb{E}\{f(a)\}\, \mathbb{E}\{g(b)\}$ when a and b are independent: $p(a, b) = p(a)p(b)$

Bias-variance formula

We now derive the bias-variance formula:

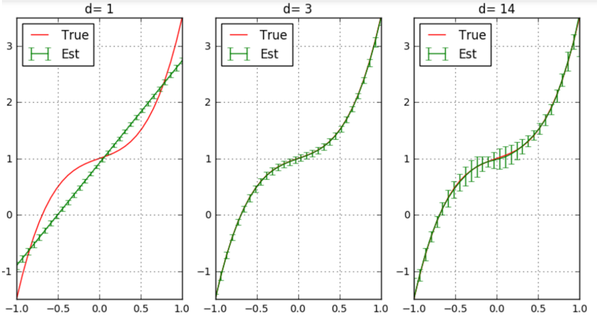
$$
\begin{aligned}
\mathrm{MSE}_{\widehat{y}}(\boldsymbol{x}) &\triangleq \mathbb{E}\left\{(y-\widehat{y})^2\right\} = \mathbb{E}\left\{\left(\epsilon + f(\boldsymbol{x}) - \widehat{f}(\boldsymbol{x};\boldsymbol{\beta})\right)^2\right\} \\
&= \mathbb{E}\left\{\epsilon^2 + 2\epsilon\big(f(\boldsymbol{x}) - \widehat{f}(\boldsymbol{x};\boldsymbol{\beta})\big) + \big(f(\boldsymbol{x}) - \widehat{f}(\boldsymbol{x};\boldsymbol{\beta})\big)^2\right\} \\
&= \mathbb{E}\{\epsilon^2\} + \cancelto{0}{2\mathbb{E}\{\epsilon\}}\,\mathbb{E}\left\{\big(f(\boldsymbol{x}) - \widehat{f}(\boldsymbol{x};\boldsymbol{\beta})\right\} + \mathbb{E}\left\{\big(f(\boldsymbol{x}) - \widehat{f}(\boldsymbol{x};\boldsymbol{\beta})\big)^2\right\} \quad \text{via linearity and independence of } \epsilon \text{ \& } \boldsymbol{\beta} \\
&= \sigma_\epsilon^2 + \mathbb{E}\left\{\big(f(\boldsymbol{x}) - \mathbb{E}[\widehat{f}(\boldsymbol{x};\boldsymbol{\beta})] + \mathbb{E}[\widehat{f}(\boldsymbol{x};\boldsymbol{\beta})] - \widehat{f}(\boldsymbol{x};\boldsymbol{\beta})\big)^2\right\} \\
&= \sigma_\epsilon^2 + \mathbb{E}\left\{\big(f(\boldsymbol{x}) - \mathbb{E}[\widehat{f}(\boldsymbol{x};\boldsymbol{\beta})]\big)^2\right\} + \mathbb{E}\left\{\big(\mathbb{E}[\widehat{f}(\boldsymbol{x};\boldsymbol{\beta})] - \widehat{f}(\boldsymbol{x};\boldsymbol{\beta})\big)^2\right\} \\
&\qquad + 2\big(f(\boldsymbol{x}) - \mathbb{E}[\widehat{f}(\boldsymbol{x};\boldsymbol{\beta})]\big)\cancelto{0}{\mathbb{E}\left\{\mathbb{E}[\widehat{f}(\boldsymbol{x};\boldsymbol{\beta})] - \widehat{f}(\boldsymbol{x};\boldsymbol{\beta})\right\}} \\
&= \underbrace{\sigma_\epsilon^2}_{\text{irreducable error}} + \Big(\underbrace{f(\boldsymbol{x}) - \mathbb{E}[\widehat{f}(\boldsymbol{x};\boldsymbol{\beta})]}_{\mathrm{bias}_{\widehat{y}}(\boldsymbol{x})}\Big)^2 + \underbrace{\mathbb{E}\left\{\big(\widehat{f}(\boldsymbol{x};\boldsymbol{\beta}) - \mathbb{E}[\widehat{f}(\boldsymbol{x};\boldsymbol{\beta})]\big)^2\right\}}_{\mathrm{var}_{\widehat{y}}(\boldsymbol{x})}
\end{aligned}
$$

Could furthermore average over the test features $\boldsymbol{x}$ to get

$$\mathrm{MSE}_{\widehat{y}} \triangleq \mathbb{E}\left\{\mathrm{MSE}_{\widehat{y}}(\boldsymbol{x})\right\} = \sigma_\epsilon^2 + \mathbb{E}\{\mathrm{bias}_{\widehat{y}}(\boldsymbol{x})^2\} + \mathbb{E}\{\mathrm{var}_{\widehat{y}}(\boldsymbol{x})\}$$

A bias-variance experiment for polynomial models

- Polynomial demo
- Red curve: $f(x)$
- Solid green curve: mean of $\widehat{y} = \widehat{f}(x;\boldsymbol{\beta})$ over 100 trials
  - bias = gap between red & green curves
- Green error-bars: ± 1 std of $\widehat{y} = \widehat{f}(x;\boldsymbol{\beta})$ over 100 trials
  - variance = std^2



high bias, low variance | no bias, medium variance | no bias, high variance

The bias-variance formula for linear models

- Consider noisy linear training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$:
  - $y_i = f(\boldsymbol{x}_i) + \epsilon_i$ with $f(\boldsymbol{x}_i) = \beta_0 + \sum_{j=1}^{d_{\textsf{true}}} \beta_j x_{ij}$ and $\epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2)$

  and the d-term linear regression model:
  - $\widehat{y} = \widehat{f}(\boldsymbol{x}; \boldsymbol{\beta}_{\textsf{ls}}) = \beta_{\textsf{ls},0} + \sum_{j=1}^{d} \beta_{\textsf{ls},j} x_j$
- Result 1: If $n < d+1$, then $\boldsymbol{\beta}_{\textsf{ls}}$ is not unique, so linear regression undefined
- Result 2: If $n \geq d+1$ and $d < d_{\textsf{true}}$, then $\widehat{y}$ is biased due to underfitting
- Result 3: If $n \geq d+1$ and $d \geq d_{\textsf{true}}$, then $\widehat{y}$ is unbiased, i.e.,

$$\mathbb{E}\{\widehat{y}\} = y_{\textsf{noiseless}} \quad \text{or} \quad \mathbb{E}\{\widehat{f}(\boldsymbol{x}; \boldsymbol{\beta}_{\textsf{ls}})\} = f(\boldsymbol{x})$$

- Result 4: If $n \gg d$ and $d \geq d_{\textsf{true}}$ and $\boldsymbol{x}$ has same distribution as $\{\boldsymbol{x}_i\}$,

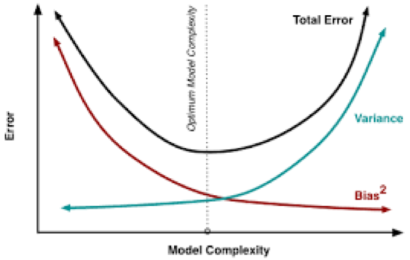
$$\boxed{\mathbb{E}\{\text{var}_{\widehat{y}}(\boldsymbol{x})\} = \frac{d+1}{n}\sigma_\epsilon^2} \quad \text{so} \quad \begin{cases} \text{variance increases linearly with \# parameters} \\ \text{variance decreases inversely with \# data samples} \end{cases}$$

Details in book: Hastie, Tibshirani, Friedman, *The Elements of Statistical Learning*

The bias-variance tradeoff for general models

- Saw two examples of how $\text{MSE}_{\widehat{y}}$ changes with model complexity:
  - polynomials: polynomial degree $= d$
  - linear regression: # features
- Similar trends hold for general models!
  - There exists a tradeoff between bias and variance
- The optimal model complexity depends on
  - the true model complexity (which affects bias)
  - the number of training samples (which affects variance)

$$\text{MSE}_{\widehat{y}} = \sigma_\epsilon^2 + \mathbb{E}\{\text{bias}_{\widehat{y}}(\boldsymbol{x})^2\} + \mathbb{E}\{\text{var}_{\widehat{y}}(\boldsymbol{x})\}$$



$\Leftarrow$	$\Rightarrow$
simpler models	richer models
less parameters	more parameters
underfitting	overfitting

http://scott.fortmann-roe.com/docs/BiasVariance.html

Outline

- Motivating Example: Polynomial Degree Selection
- Cross-validation
- The Bias-Variance Tradeoff
- From Model-Order Selection to Feature Selection

Feature selection: A generalization of model-order selection

- So far, we discussed model-*order* selection (e.g., polynomial degree d)
  - Select between several models, each with a different complexity
  - For example, $$\begin{aligned} y &\approx \beta_0 + \beta_1 x_1 \\ y &\approx \beta_0 + \beta_1 x_1 + \beta_2 x_2 \\ y &\approx \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \end{aligned}$$
- More generally, given d total features $\{x_j\}_{j=1}^d$, we might wonder which subset of features works best for predicting y
  - Called "feature selection"
  - Given d features (plus intercept), there are $2^d - 1$ possible non-trivial subsets
- How do we choose the best subset?
  - Can use cross-validation to choose between models
  - but need to manage computational complexity...
- Discussed further in the next unit ...

Learning objectives

- Understand the problem of model-order selection
- Visually identify underfitting and overfitting in a scatterplot
- Understand the need to partition data into training and testing subsets
- Understand the K-fold cross-validation process
  - Use it to assess the test error for a given model
  - Use it to select model order
- Understand the concepts of bias, variance, and irreducible error
  - Know how to compute each from synthetically generated data
  - Understand the bias-variance tradeoff