Lecture 4
Model Order Selection

EE-UY 4423: INTRODUCTION TO MACHINE LEARNING

PROF. SUNDEEP RANGAN

Learning Objectives

- ❑ Identify the order of a linear model
- ❑ Visually identify overfitting and underfitting in a scatterplot
- ❑ Compute bias and variance in linear models
  - Function of the model order, noise statistics, input distribution, training samples
- ❑ Write a program to perform cross-validation to select an optimal model order

Outline

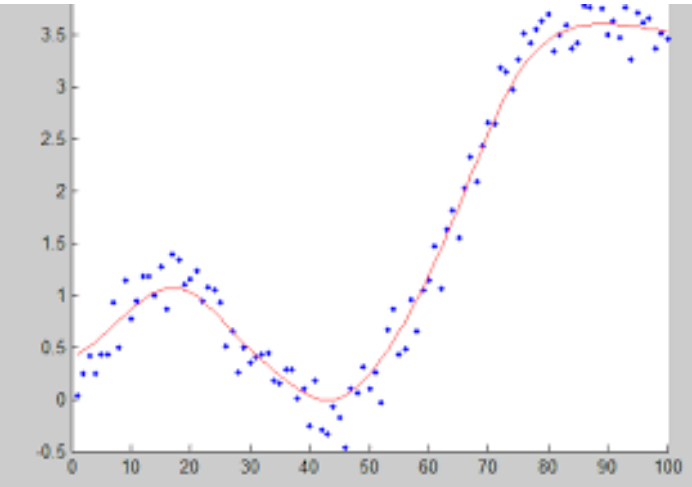
- Motivating Example: What polynomial degree should a model use?
- Bias and variance in linear models
- Cross-validation

Polynomial Fitting

- Last lecture: polynomial regression
- Given data $(x_i, y_i), i = 1, \dots, N$
- Learn a polynomial relationship:

$$y = \beta_0 + \beta_1 x + \cdots + \beta_d x^d + \epsilon$$

  - d = degree of polynomial. Called model order
  - $\boldsymbol{\beta} = (\beta_0, \cdots, \beta_d)$ = coefficient vector
- Given d, can find $\boldsymbol{\beta}$ via least squares
- How do we select d from data?
- This problem is called model order selection.

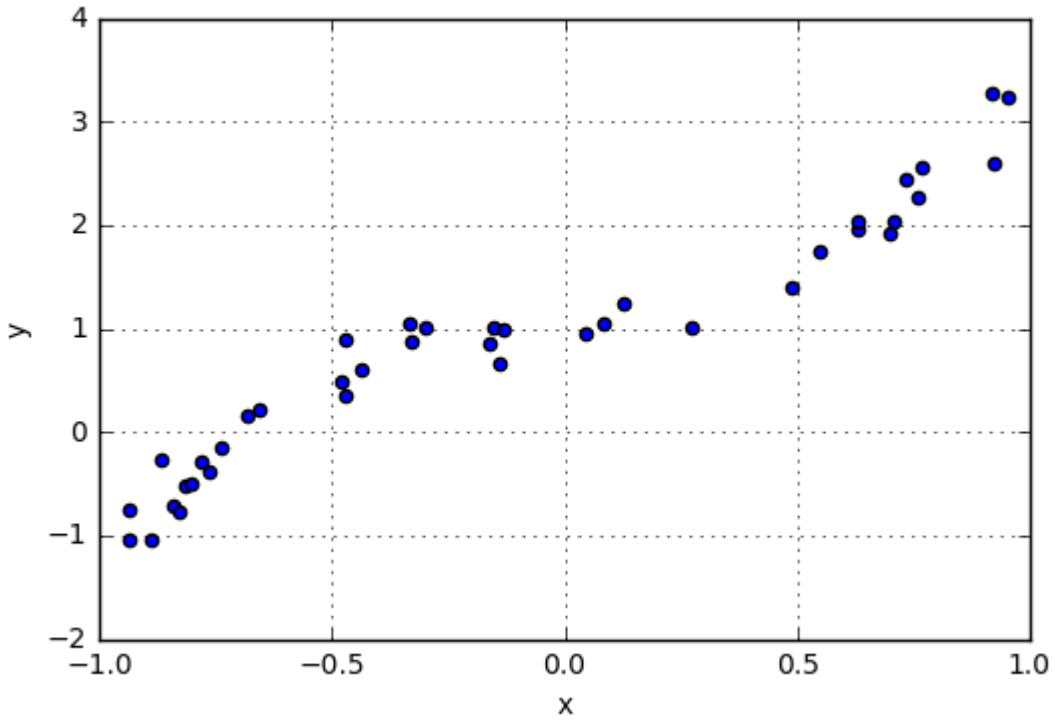


Example Question

- You are given some data.
- Want to fit a model: $y \approx f(x)$
- Decide to use a polynomial:

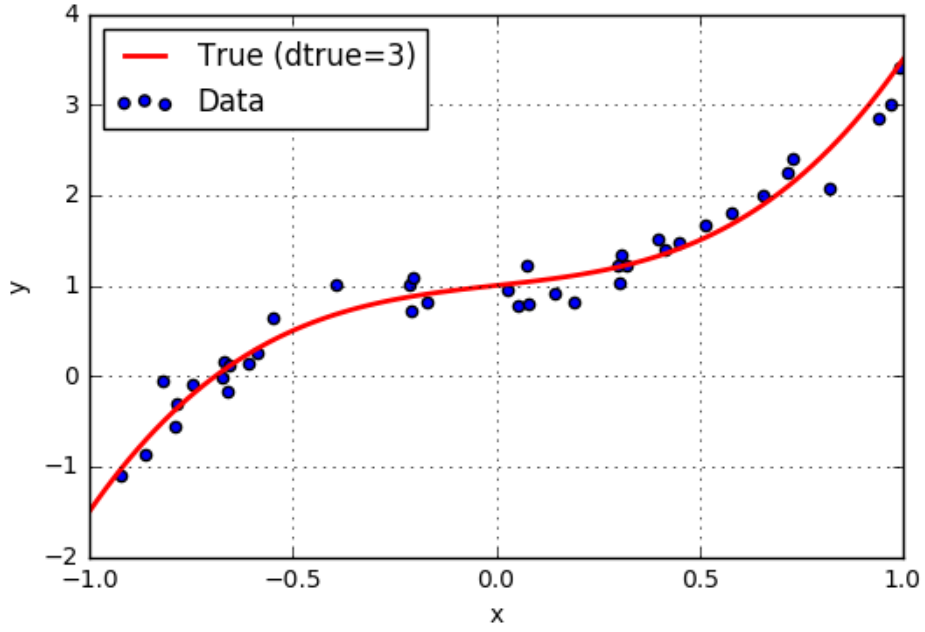
$$f(x) = \beta_0 + \beta_1 x + \cdots + \beta_d x^d$$

- What model order d should we use?
- Thoughts?



Synthetic Data

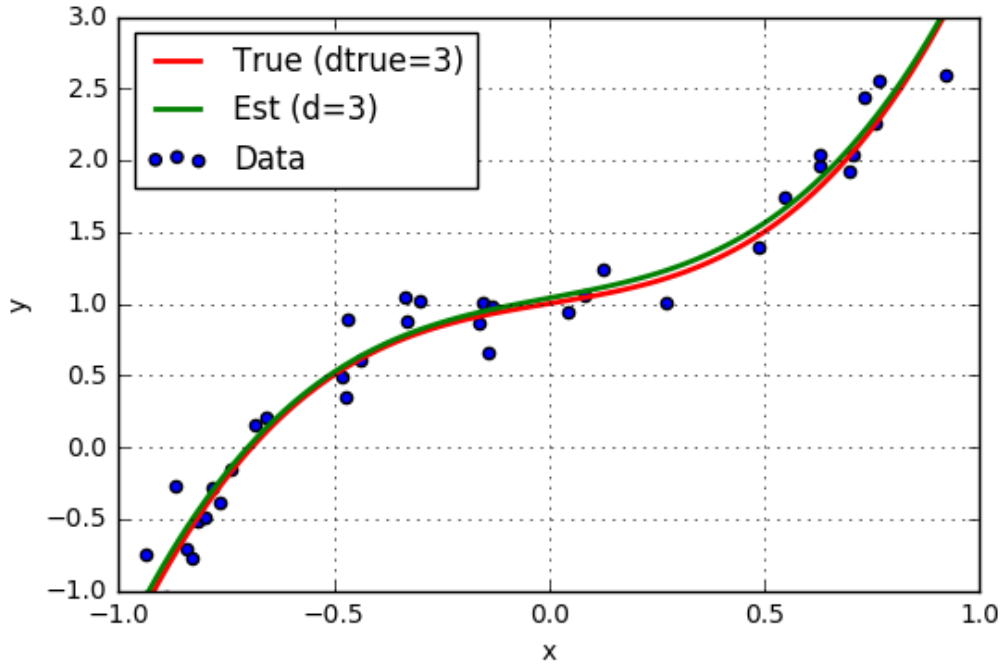
- Previous example is synthetic data
- x_i: 40 samples uniform in [-1,1]
- $y = f(x) + \epsilon$,
  - $f(x) = \beta_0 + \beta_1 x + \cdots + \beta_d x^d$ = "true relation"
  - $d = 3,\ \epsilon \sim N(0, \sigma^2)$
- Synthetic data useful for analysis
  - Know "ground truth"
  - Can measure performance 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 True Model Order

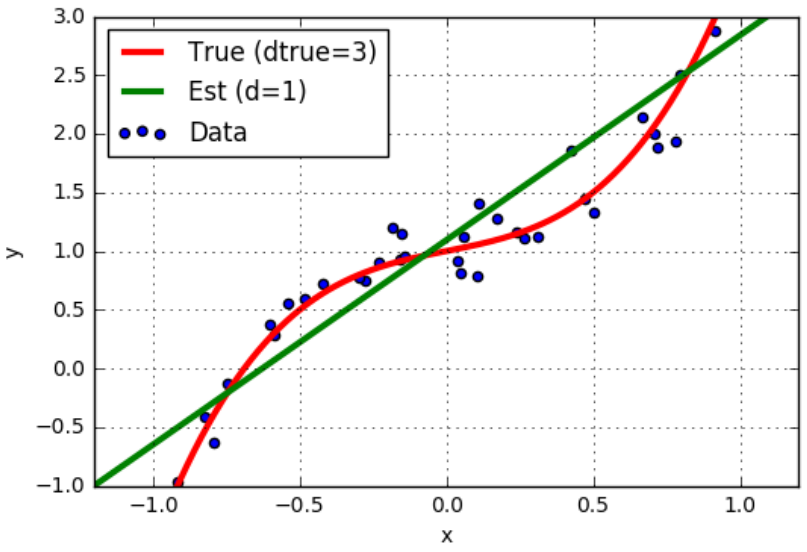
- Suppose true polynomial order, d=3, is known
- Use linear regression
  - numpy.polynomial package
- Get very good fit



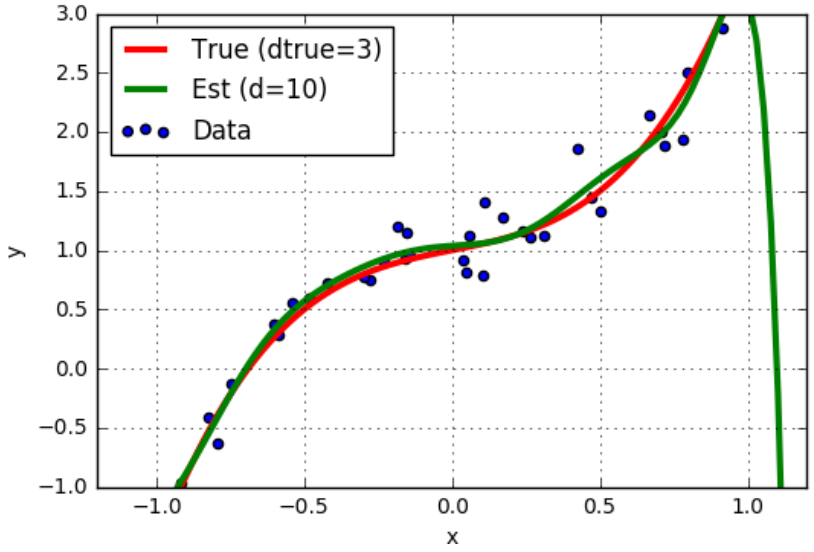

```
d = 3
beta_hat = poly.polyfit(xdat,ydat,d)

# 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')
```

But, True Model Order not Known

- Suppose we guess the wrong model order?

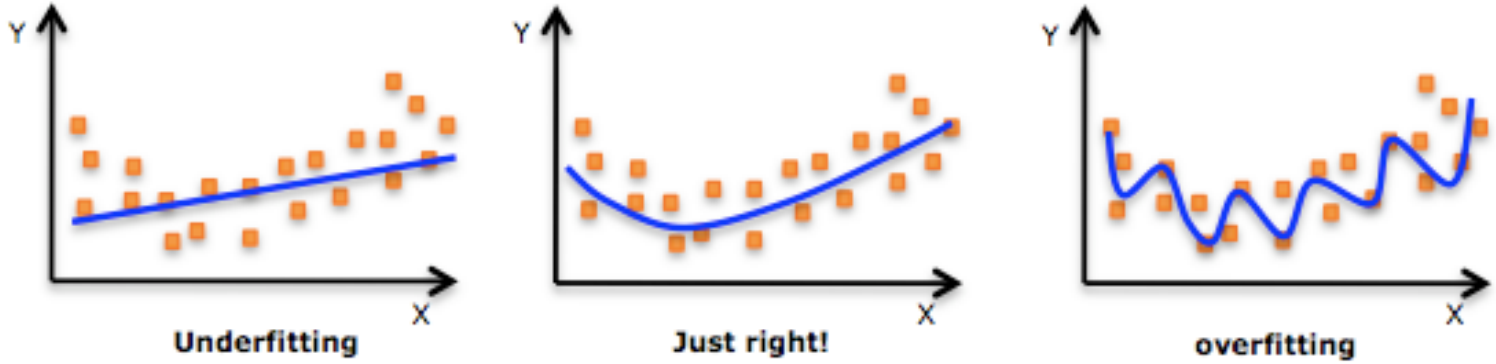


d=1 “Underfitting”



d=10 “Overfitting”

How Can You Tell from Data?



- Is there a way to tell what is the correct model order to use?
- Must use the data. Do not have access to the true d?
- What happens if we guess:
  - d too big?
  - d too small?

Using RSS on Training Data?

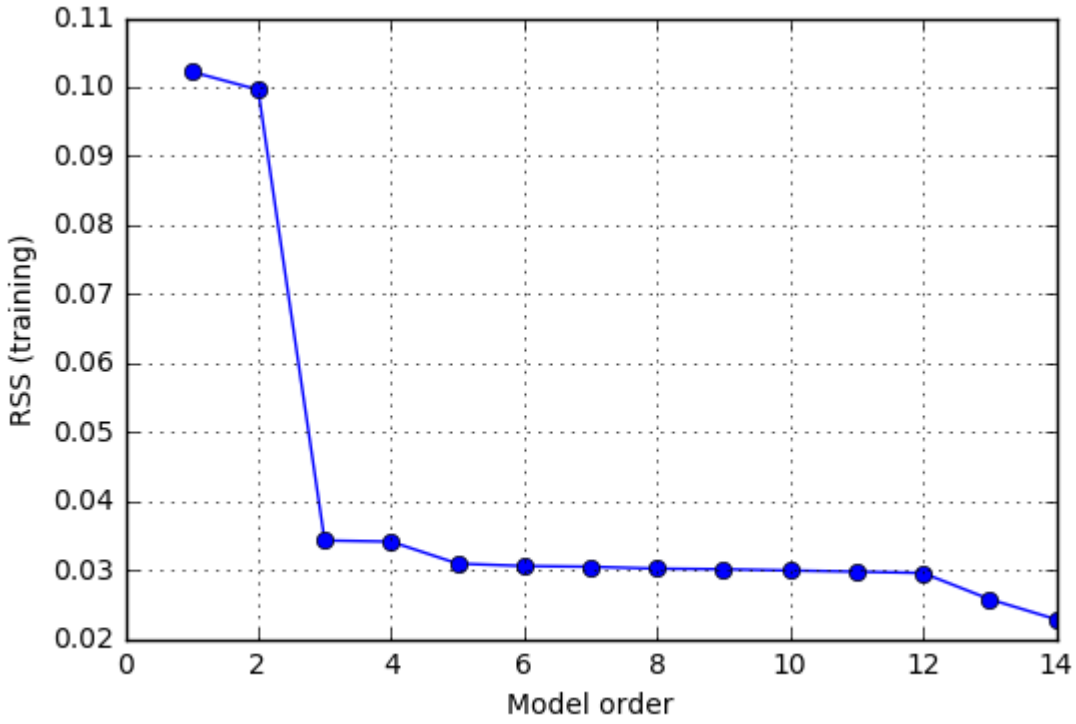
- Simple (but bad) idea:
  - For each model order, d, find estimate $\widehat{\boldsymbol{\beta}}$
  - Compute predicted values on training data

$$\hat{y}_i = \widehat{\boldsymbol{\beta}}^T \boldsymbol{x}_i$$

  - Compute RSS

$$RSS(d) = \sum_i (y_i - \hat{y}_i)^2$$

  - Find d with lowest RSS
- This doesn't work
  - $RSS(d)$ is always decreasing (Question: Why?)
  - Minimizing $RSS(d)$ will pick d as large as possible
  - Leads to overfitting
- What went wrong?
- How do we do better?



Outline

- Motivating Example: What polynomial degree should a model use?
- Bias and variance in linear models
- Cross-validation

Review: Transform Linear Models

- Consider linear estimation under a transformed linear model:

$$y = f(\boldsymbol{x}) + \epsilon, \qquad f(\boldsymbol{x}) = \sum_{i=1}^{p} \phi_j(\boldsymbol{x})\beta_j = \phi(\boldsymbol{x})^T \boldsymbol{\beta}$$

  - $f(\boldsymbol{x})$ = linear function to learn, $\phi_j(\boldsymbol{x})$= transformed features
  - ϵ = "noise" = model error.

- Example transforms:
  - Standard regression $\phi(\boldsymbol{x}) = [1, x_1, \dots, x_k]^T$ (k original features, $k+1$ transformed features)
  - Polynomial regression: $\phi(x) = \left[1, x, \dots, x^d\right]^T$ (1 original feature, $d+1$ transformed features)

Review: Least Squares Formula

- Get training data $(\boldsymbol{x}_i, y_i), i = 1, \dots, N$
- Define matrix of transformed features on training data:

$$A = \begin{bmatrix} \phi(\boldsymbol{x}_1)^T \\ \vdots \\ \phi(\boldsymbol{x}_N)^T \end{bmatrix} = \underbrace{\begin{bmatrix} \phi_1(\boldsymbol{x}_1) & \cdots & \phi_p(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_N) & \cdots & \phi_p(\boldsymbol{x}_N) \end{bmatrix}}_{\text{Transformed features}}, \quad \text{Samples} \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} \quad \text{Parameters}$$

- Matrix model: $\boldsymbol{y} = \boldsymbol{A\beta} + \boldsymbol{\epsilon}$
- Least-squares fit: $\widehat{\boldsymbol{\beta}} = \left(\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}\right)^{-1}\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{y}$
- Prediction on a new sample: $\hat{y} = \hat{f}(\boldsymbol{x}) = \phi(\boldsymbol{x})^T\widehat{\boldsymbol{\beta}}$
  - $\hat{f}(\boldsymbol{x})$ is the estimated function

The Model and True Function

- To study effect of undermodeling, suppose that
  - Estimator assumes linear model: $\hat{y} = \hat{f}(\boldsymbol{x}) = \phi(\boldsymbol{x})^T \widehat{\boldsymbol{\beta}}$
  - But, data has actual relation: $y = f_0(x) + \epsilon,\ \epsilon \sim N(0, \sigma_\epsilon^2)$
- Function $f_0(x)$ may not be exactly within linear model class. Example:
  - Model $\hat{f}(x) = \beta_0 + \beta_1 x + \beta_2 x^2$ = second order polynomial
  - True function, $f_0(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$ = cubic polynomial
- Basic questions:
  - How well does multiple linear regression perform?
  - When does it work well and when does it fail?
- Will provide answers in terms of true function $f_0(x)$, model class and noise σ_ϵ^2

Mean Squared Error

- Consider following method to evaluate performance of estimator.
- Training stage:
  - Get data $(\boldsymbol{x}_i, y_i), i = 1, \dots, N$
  - Learn parameters $\widehat{\boldsymbol{\beta}}$
- Take a new test sample x
  - Generally different from training data
- Define test mean squared error:

$$\mathrm{MSE}(\hat{y}) \coloneqq E(y - \hat{y})^2$$

  - Measures average error in predicting response on new samples.
  - Assume true function $f_0(x)$, inputs training data x_i and test sample x are fixed
  - Randomness is only due to noise in training and test.

Irreducible and Reducible MSE

- Recall true relation: $y = f_0(x) + \epsilon,\ \epsilon \sim N(0, \sigma_\epsilon^2)$, Estimated relation $\hat{y} = \hat{f}(x)$
  - $f_0(x)$ = "true" functional relation between x and y
- Since ϵ is independent of training data (proof on board):

$$MSE(\hat{y}) = E(y - \hat{y})^2 = \sigma_\epsilon^2 + E\left(f_0(x) - \hat{f}(x)\right)^2$$

- MSE has two components
- Irrreducible component: σ_ϵ^2. Due to inability of x to fully explain y
  - Cannot be reduced with any amount of training
- Reducible / trainable component: MSE on the functions

$$\mathrm{MSE}\left(\hat{f}(x)\right) = E\left(f_0(x) - \hat{f}(x)\right)^2$$

  - We will see this component can be reduced with sufficient rich model and enough training.

Bias Variance Formula

- Bias: $\text{Bias}\left(\hat{f}(x)\right) = f_0(x) - E\left(\hat{f}(x)\right)$
  - We will see that the bias is large when there is under-modeling
- Variance: $\text{Var}\left(\hat{f}(x)\right) = E\left(\hat{f}(x) - E(\hat{f}(x))\right)^2$
  - Will see this is large when the over-modeling
- Bias-Variance Formula: The trainable component of the MSE is given by

$$\text{MSE}\left(\hat{f}(x)\right) = \text{Bias}^2\left(\hat{f}(x)\right) + \text{Var}\left(\hat{f}(x)\right)$$

  - Proof on board
- Total MSE has three components:

$$\text{MSE}(\hat{y}) = \text{Bias}^2\left(\hat{f}(x)\right) + \text{Var}\left(\hat{f}(x)\right) + \sigma_\epsilon^2$$

Variance in the Linear Model

- Recall: $\hat{f}(x) = \phi(x)^T \hat{\beta}$
- Variance in the trainable component: $\mathrm{Var}\left(\hat{f}(x)\right) = \frac{\sigma_\epsilon^2}{N} \phi(x)^T R_{AA}^{-1} \phi(x)$
  - Decreases with number of samples N
  - R_{AA} is the sample correlation matrix of the transformed data:

$$R_{AA} = \frac{1}{N} A^T A = \frac{1}{N} \sum_{i=1}^{N} \phi(x_i)\phi(x_i)^T$$

- Proof requires a little bit more of linear algebra and will be skipped.
- For linear model, variance does not depend on true function $f_0(x)$
  - Only depends on noise level σ_ϵ^2 and locations of training data $x_i, i = 1, \dots, N$
- When N is large and x_i are i.i.d. $R_{AA} \to E(\phi(x_i)\phi(x_i)^T)$

Variance and Model Order

- Suppose that test sample x drawn from training samples.
- Then, average variance is:

$$\frac{1}{\mathrm{N}}\sum_{i=1}^{N} Var\left(\hat{f}(x_i)\right) = \frac{\sigma_\epsilon^2 p}{N}$$

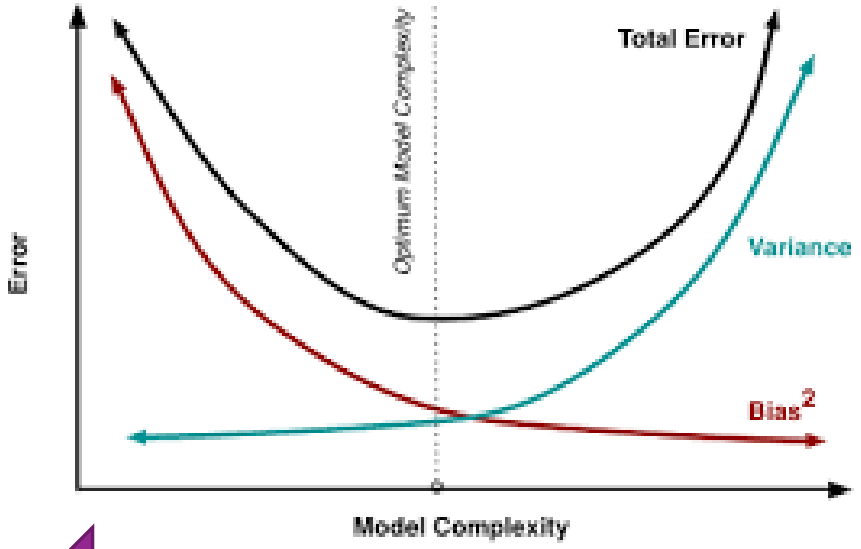
  - Proof on board

- Conclusions:
  - Variance increases with number of parameters p and noise variance σ_ϵ^2
  - Decreases with number of samples N
  - Learning more complex models (higher p) requires more data (higher N)
- Variance may be much higher for test data points not distributed like training data

Bias in the Linear Model

- Recall $\text{Bias}\left(\hat{f}(x)\right) = f_0(x) - \bar{f}(x), \quad \bar{f}(x) = E(\hat{f}(x))$
  - $\bar{f}(x)$ = expected value of the estimate given training inputs $x_i, i = 1, \ldots, N$
- No undermodeling occurs when:
  - True function is $f_0(\boldsymbol{x}) = \phi(x)^T\beta$ for some true parameter β
- Theorem: When there is no undermodeling, there is no bias
  - $E(\hat{\beta}) = \beta$
  - At any test sample x, $\bar{f}(x) = E(\hat{f}(x)) = f_0(x)$. Hence $\text{Bias}(x) = 0$
- Bias occurs when:
  - True $f_0(x)$ is not fit exactly by a linear model
  - Ex: Model uses assumes a 2-nd order poly, but $f(x)$ is higher order
  - Occurs from under-modeling

Bias-Variance Tradeoff



Simpler models
Less parameters
Under-fitting

Richer models
More parameters
Over-fitting

- Optimal model order depends on:
  - Amount of samples available
  - Underlying complexity of the relation

Outline

- ❑ Motivating Example: What polynomial degree should a model use?
- ❑ Bias and variance in linear models
- ❑ Cross-validation

Cross Validation

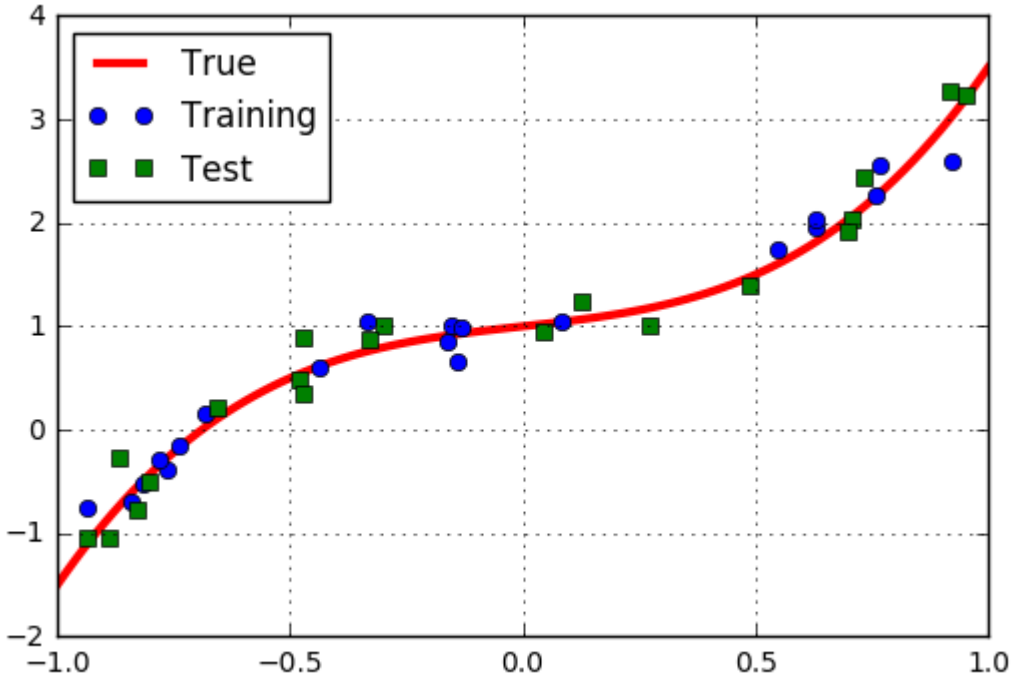
- Concept: Need to test fit on data independent of training data
- Divide data into two sets:
  - N_{train} training samples, N_{test} test samples
- For each model order, p, learn parameters $\hat{\beta}$ from training samples
- Measure RSS on test samples.

$$RSS_{test}(p) = \sum_{i \in \text{test}} (\hat{y}_i - y_i)^2$$

- Select model order p that minimizes $RSS_{test}(p)$

Polynomial Example: Training Test Split

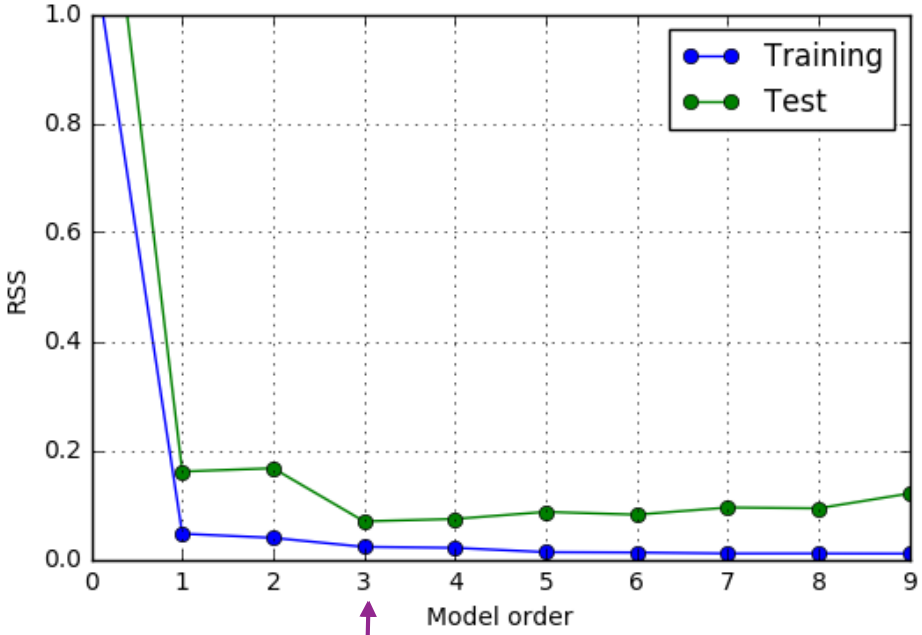
- Example: Split data into 20 samples for training, 20 for test




```
# Number of samples for training and test
ntr = nsamp // 2
nts = nsamp - ntr

# Training
xtr = xdat[:ntr]
ytr = ydat[:ntr]

# Test
xts = xdat[ntr:]
yts = ydat[ntr:]
```

Finding the Model Order

- Estimated optimal model order = 3



RSS test minimized at $d = 3$
RSS training always decreases

```
dtest = np.array(range(0,10))
RSStest = []
RSStr = []
for d in dtest:

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

    # Measure RSS on training data
    # This is not necessary, but we do it just to show the training error
    yhat = poly.polyval(xtr,beta_hat)
    RSSd = np.mean((yhat-ytr)**2)
    RSStr.append(RSSd)

    # Measure RSS on test data
    yhat = poly.polyval(xts,beta_hat)
    RSSd = np.mean((yhat-yts)**2)
    RSStest.append(RSSd)

plt.plot(dtest,RSStr,'bo-')
plt.plot(dtest,RSStest,'go-')
plt.xlabel('Model order')
plt.ylabel('RSS')
plt.grid()
plt.ylim(0,1)
plt.legend(['Training','Test'],loc='upper right')
```

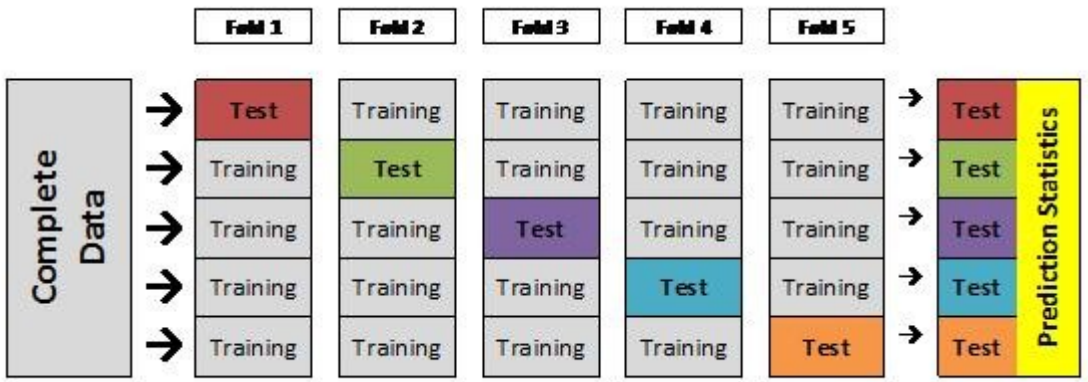
Problems with Simple Train/Test Split

- Test error could vary depending on samples selected
- Only use limited number of samples for training
- Problems particularly bad for data with limited number of samples

K-Fold Cross Validation

- K-fold cross validation
  - Divide data into K parts
  - Use $K-1$ parts for training. Use remaining for test.
  - Average over the K test choices
  - More accurate, but requires K fits of parameters

- Leave one out cross validation (LOOCV)
  - Take $K = N$ so one sample is left out.
  - Most accurate, but requires N model fittings



From http://blog.goldenhelix.com/goldenadmin/cross-validation-for-genomic-prediction-in-svs/

Polynomial Example

- Use sklearn Kfold object
- Loop
  - Outer loop: Over K folds
  - Inner loop: Over model order
  - Measure test error in each fold and order
  - Can be time-consuming

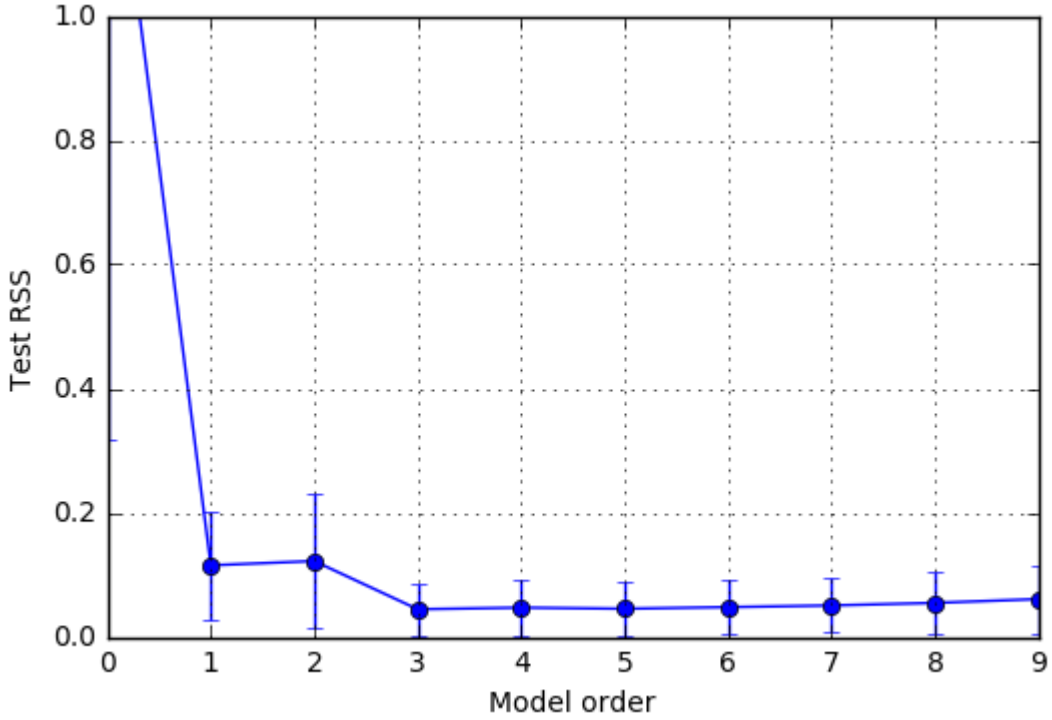
```
# Create a k-fold object
nfold = 20
kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)

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

# Loop over the folds
RSSts = np.zeros((nd,nfold))
for isplit, Ind in enumerate(kf.split(xdat)):

    # Get the training data in the split
    Itr, Its = Ind
    xtr = xdat[Itr]
    ytr = ydat[Itr]
    xts = xdat[Its]
    yts = ydat[Its]

    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)
```

Polynomial Example Plotting

- For each model order d
  - Compute mean test RSS
  - Compute std deviation test RSS
  - Average over the k folds

```
RSS_mean = np.mean(RSSts,axis=1)
RSS_std  = np.std(RSSts,axis=1)
plt.errorbar(dtest, RSS_mean, yerr=RSS_std, fmt='o-')
plt.ylim(0,1)
plt.xlabel('Model order')
plt.ylabel('Test RSS')
plt.grid()
```

- Simple model selection
  - Select d with lowest mean test RSS
- For this example
  - Estimate model order = 3



One Standard Deviation Rule

- Previous slide: Select d to minimize mse_mean[d]
- Problem: Often over-predicts model order
- One standard deviation rule
  - Use simplest model within one standard dev. of minimum
- Detailed procedure:
  - Find d0 to minimize mse_mean[d]
  - Set mse_tgt = mse_mean[d0] + mse_std[d0]
  - Find dopt minimize d s.t. mse_mean[d] <= mse_tgt