Lecture 4
Model Order Selection

EE-UY 4563/EL-GY 9123: INTRODUCTION TO MACHINE LEARNING

PROF. SUNDEEP RANGAN

Learning Objectives

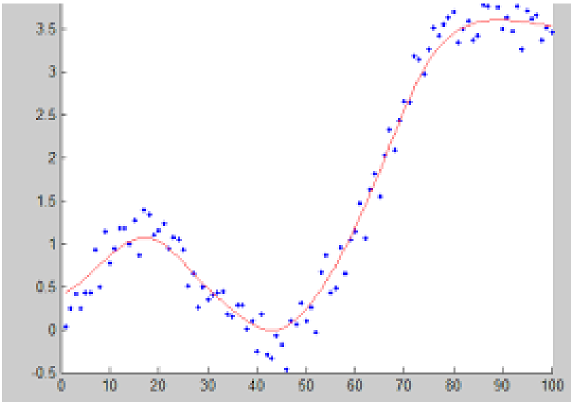
- ❑Identify the order of a linear model
- ❑Visually identify overfitting and underfitting in a scatterplot
- ❑Determine if there is under-modeling for a given true function and model class
- ❑Compute the irreducible error for a model
- ❑Compute bias in a model class for the case of no noise
  - Computing variance is more advanced and not considered here
- ❑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
- Cross-validation

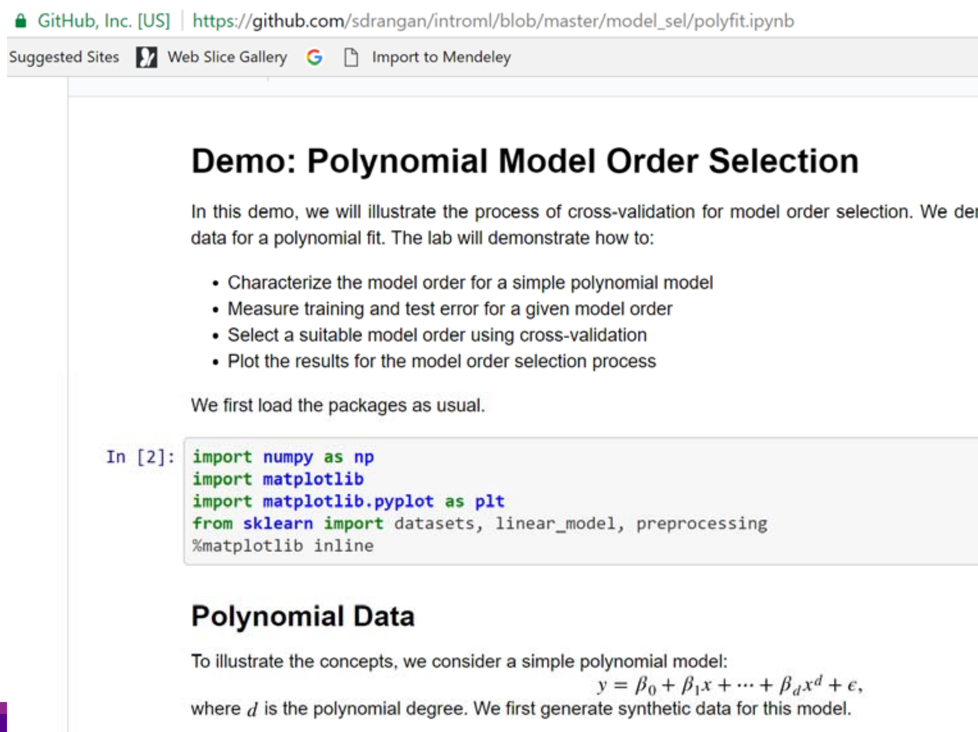
Polynomial Fitting

- Last lecture: polynomial regression
- Given data $(x_i, y_i), i = 1, \ldots, N$
- 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.



Demo on Github

- Demo on github: https://github.com/sdrangan/introml/blob/master/model_sel/polyfit.ipynb



GitHub, Inc. [US] | https://github.com/sdrangan/introml/blob/master/model_sel/polyfit.ipynb

Suggested Sites Web Slice Gallery Import to Mendeley

Demo: Polynomial Model Order Selection

In this demo, we will illustrate the process of cross-validation for model order selection. We der
data for a polynomial fit. The lab will demonstrate how to:

- Characterize the model order for a simple polynomial model
- Measure training and test error for a given model order
- Select a suitable model order using cross-validation
- Plot the results for the model order selection process

We first load the packages as usual.

In [2]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model, preprocessing
%matplotlib inline
```

Polynomial Data

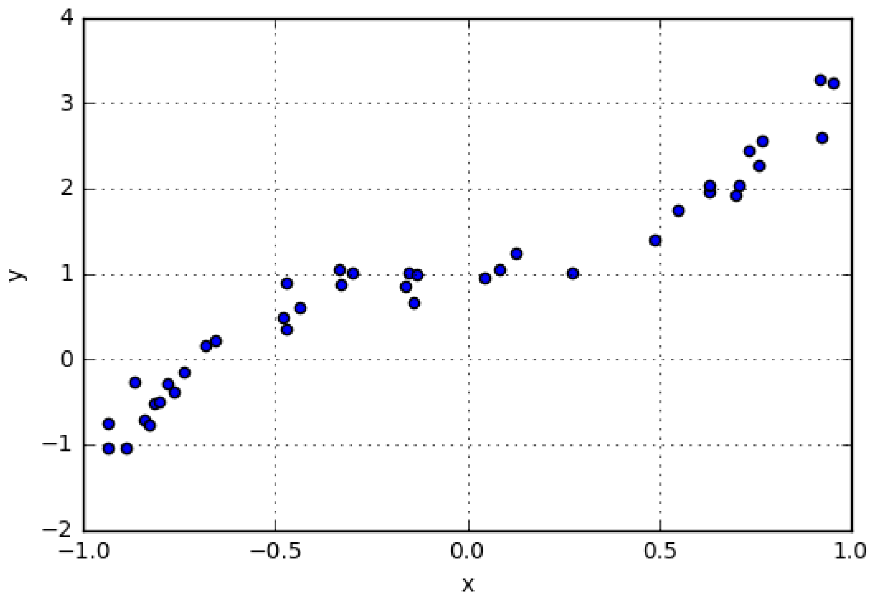
To illustrate the concepts, we consider a simple polynomial model:

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

where d is the polynomial degree. We first generate synthetic data for this model.

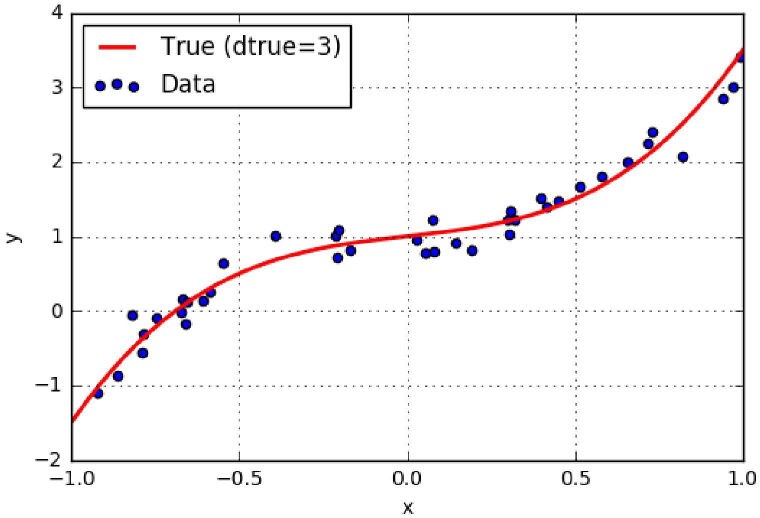
Example Question

- 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

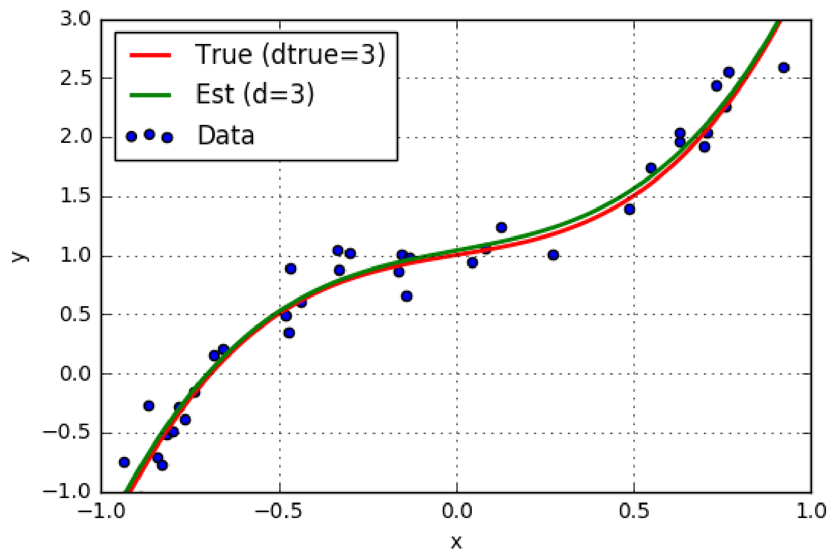
- Previous example is synthetic data
- x_i: 40 samples uniform in [-1,1]
- $y = f(x) + \epsilon,$
  - $f(x) = \beta_0 + \beta_1 x + \cdots + \beta_d x^d$ = "true relation"
  - $d = 3,\ \epsilon \sim N(0, \sigma^2)$
- Synthetic data useful for analysis
  - Know "ground truth"
  - Can measure performance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

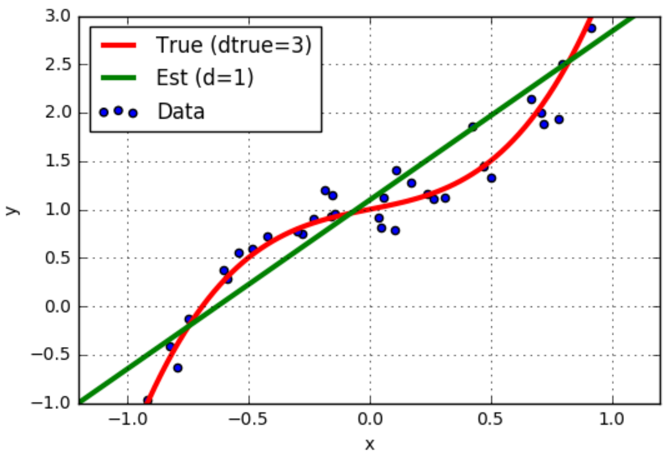
- Suppose true polynomial order, d=3, is known
- Use linear regression
  - numpy.polynomial package
- Get very good fit



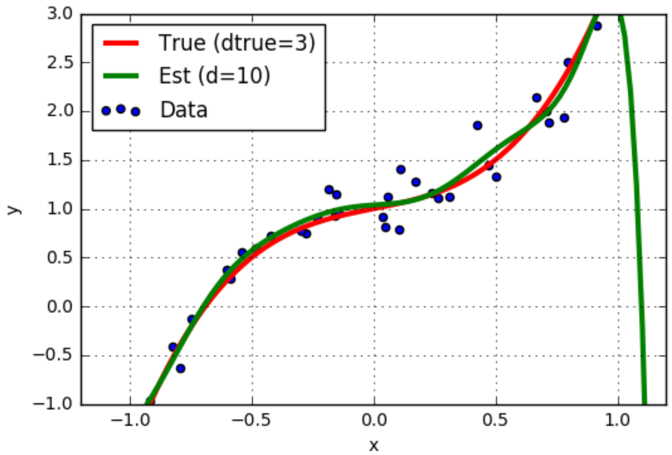

```
d = 3
beta_hat = poly.polyfit(xdat,ydat,d)

# 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?

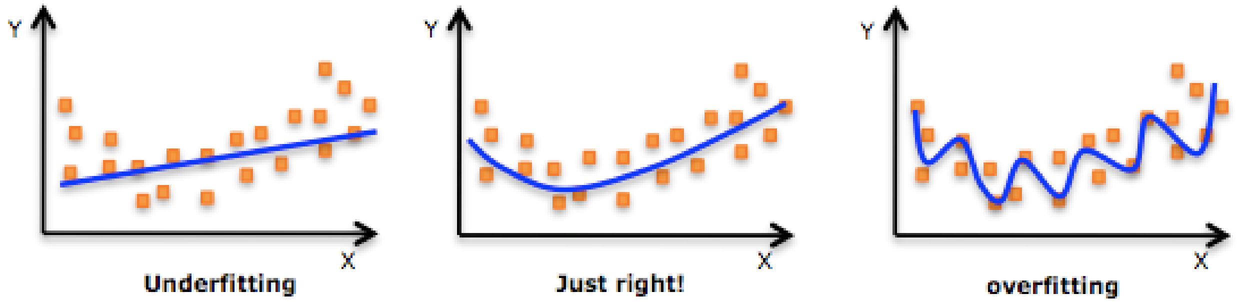


d=1 “Underfitting”



d=10 “Overfitting”

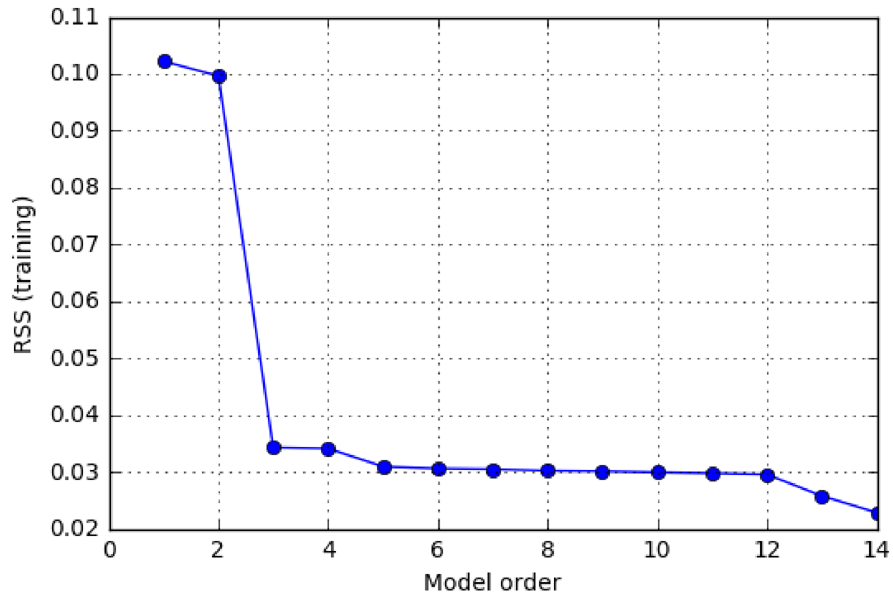
How Can You Tell from Data?



- ❑ 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 $\hat{\boldsymbol{\beta}}$
  - Compute predicted values on training data
$$\hat{y}_i = \hat{\boldsymbol{\beta}}^T \boldsymbol{x}_i$$
  - 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
- Cross-validation

Model Class

- Consider general estimation problem
  - Given data (x_i, y_i) want to learn a functional relation: $y \approx \hat{y} = f(x)$
- Model class: The set of possible estimates:

$$\hat{y} = f(\boldsymbol{x}, \boldsymbol{\beta})$$

  - Set is parametrized by $\boldsymbol{\beta}$
- Many possible examples:
  - Linear model: $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$
  - Polynomial model: $\hat{y} = \beta_0 + \beta_1 x + \cdots + \beta_k x^k$
  - Nonlinear: $\hat{y} = \beta_0 + \beta_1 e^{-\beta_2 x} + \beta_3 e^{-\beta_4 x}$
  - …

Model Class and True Function

- Analysis set-up:
  - Learning algorithm assumes a model class: $\hat{y} = f(\boldsymbol{x}, \boldsymbol{\beta})$
  - But, data has true relation: $y = f_0(x) + \epsilon,\ \epsilon \sim N(0, \sigma_\epsilon^2)$

- Will quantify three key effects:
  - Irreducible error
  - Under-modeling
  - Over-fitting

Output Mean Squared Error

- To evaluate prediction error suppose we are given:
  - A parameter estimate $\widehat{\boldsymbol{\beta}}$ (computed from the learning algorithm)
  - A test point $\boldsymbol{x}_{test}$
  - Test point is generally different from training samples.
- Predicted value: $\hat{y} = f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)$
- Actual value: $y = f_0(\boldsymbol{x}_{test}) + \boldsymbol{\epsilon}$
- Output mean squared error:

$$MSE_y\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right) \coloneqq E[y - \hat{y}]^2$$

  - Expectation is over noise $\boldsymbol{\epsilon}$ on the test sample.

Irreducible Error

- Rewrite output MSE:

$$MSE_y(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}) := E[y - \hat{y}]^2 = E\left[f_0(\boldsymbol{x}_{test}) + \epsilon - f(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}})\right]^2$$

- Since noise on test sample is independent of $\widehat{\boldsymbol{\beta}}$ and $\boldsymbol{x}_{test}$:

$$MSE_y(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}) := \left[f_0(\boldsymbol{x}_{test}) - f(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}})\right]^2 + \mathrm{E}(\epsilon^2) = \left[f_0(\boldsymbol{x}_{test}) - f(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}})\right]^2 + \sigma_\epsilon^2$$

- Define irreducible error: σ_ϵ^2
  - Lower bound on $MSE_y(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}) \geq \sigma_\epsilon^2$
  - Fundamental limit on ability to predict y
  - Occurs since y is influenced by other factors than $\boldsymbol{x}$

Under-Modeling

- Definition: A true function $f_0(\boldsymbol{x})$ is in the model class $\hat{y} = f(\boldsymbol{x}, \boldsymbol{\beta})$ if:

$$f_0(\boldsymbol{x}) = f(\boldsymbol{x}, \boldsymbol{\beta}_0) \ \text{ for all } \boldsymbol{x}$$

for some parameter $\boldsymbol{\beta}_0$.
  - $\boldsymbol{\beta}_0$ called the true parameter

- Under-modeling: When $f_0(\boldsymbol{x})$ is not in the model class

Sample Question

- For each pair, state if the true function is in the model class or not
  - That is, is there under-modeling or not?
  - If true function is in the model class, state the true parameter

- Examples:
  - True function: $f_0(x) = 2 + 3x$ Model class: $f(x, \beta) = \beta_0 + \beta_1 x + \beta_2 x^2$
  - True function: $f_0(x) = 2 + 3x + 4x^2$ Model class: $f(x, \beta) = \beta_0 + \beta_1 x$
  - True function: $f_0(x) = \sin(2\pi(5)x + 7)$ Model class: $f(x, \beta) = \beta_0 \sin(2\pi(5)x) + \beta_1 \cos(2\pi(5)x)$
  - True function: $f_0(x) = \sin(2\pi(8)x + 7)$ Model class: $f(x, \beta) = \beta_0 \sin(2\pi(5)x) + \beta_1 \cos(2\pi(5)x)$

- Solutions in class

Analysis of Under-Modeling: Noise-Free Case

- Assume true relation has no noise: $y = f_0(\boldsymbol{x})$
  - Can model noise, but requires more probability theory
- Get training data: $(\boldsymbol{x}_i, y_i), i = 1, \ldots, n$
- Fit model parameter from least-squares:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(y_i - f(\boldsymbol{x}_i, \boldsymbol{\beta})\right)^2 = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(f_0(\boldsymbol{x}_i) - f(\boldsymbol{x}_i, \boldsymbol{\beta})\right)^2$$

- Conclusions: With no noise
  - Fitting finds best least squares fit of the true functions in the model class
  - If there is a unique true parameter, then $\hat{\boldsymbol{\beta}} = \boldsymbol{\beta}_0$. Estimator identifies correct parameter

Bias: Noise-Free Case

- Let $\boldsymbol{x}_{test}$ = some test point
  - Can be different from the training data set
- Definition: When there is no noise, the bias at a test point $\boldsymbol{x}_{test}$ is:

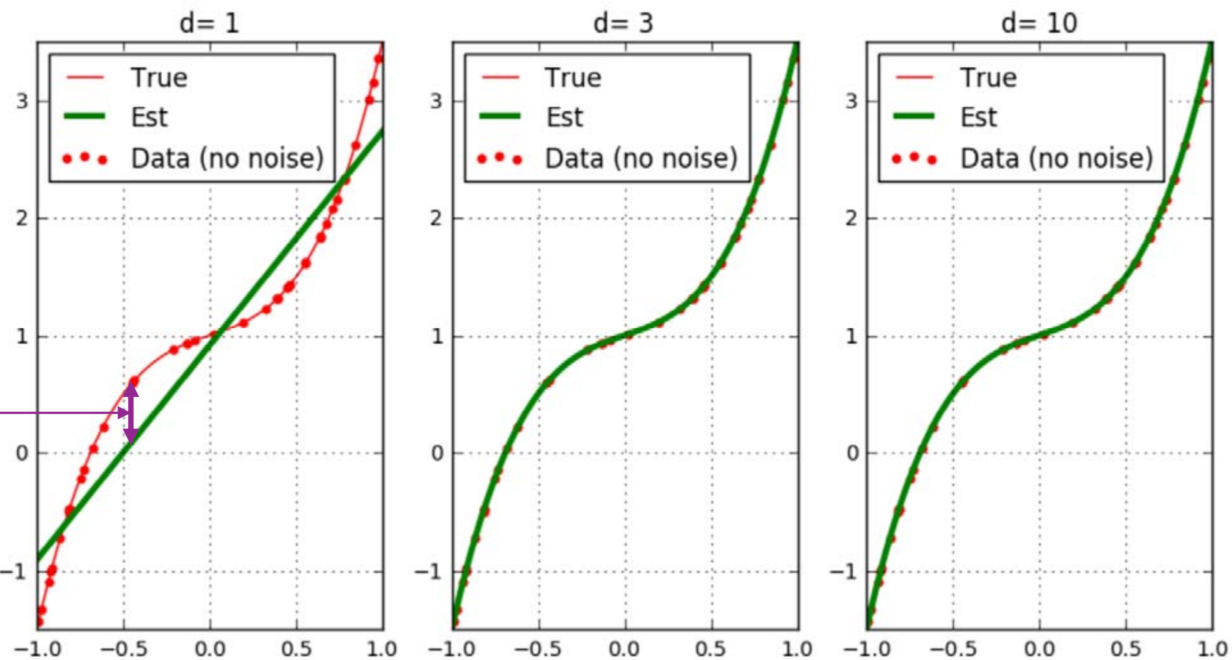
$$Bias(\boldsymbol{x}_{test}) := f_0(\boldsymbol{x}_{test}) - f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)$$

- Measures difference true and estimated relation in absence of noise
- Previous analysis shows:
  - Bias is small when true function is close to model class
  - When there is no under-modeling, $Bias(\boldsymbol{x}_{test}) = 0$ and true parameter found.

Bias Visualized

- Polynomial example
  - $d_{true} = 3$
- No noise in data

Bias(x)



Model has bias

No bias

No bias

Analysis with Noise (Advanced)

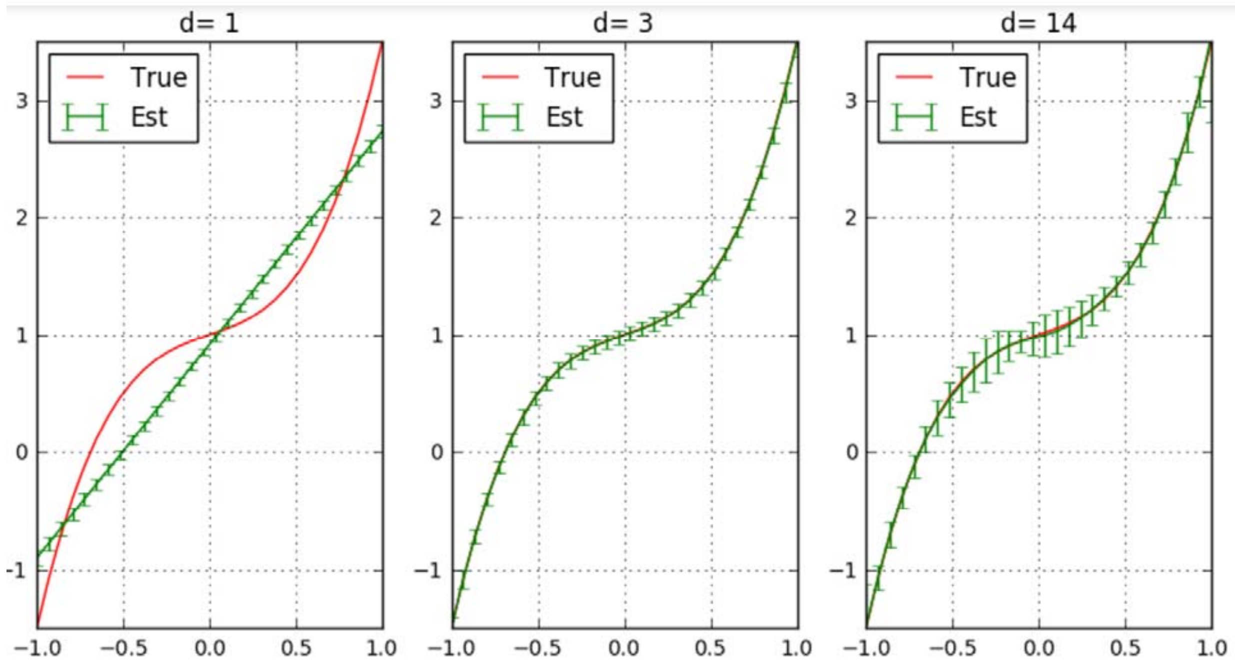
- Now assume noise: $y = f_0(\boldsymbol{x}) + \epsilon, \epsilon \sim N(0, \sigma_\epsilon^2)$
- Get training data: $(\boldsymbol{x}_i, y_i), i = 1, \dots, n$
- Fit a parameter:

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(y_i - f(\boldsymbol{x}_i, \boldsymbol{\beta})\right)^2$$

  - $\widehat{\boldsymbol{\beta}}$ will be random.
  - Depends on particular noise realization.
- Take a new test point $\boldsymbol{x}_{test}$ (not random)
- Compute mean and variance of estimated function $f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)$
- Define:
  - Bias: Difference of true function from mean estimate
  - Variance: Variance of estimate around its mean

Bias and Variance Illustrated

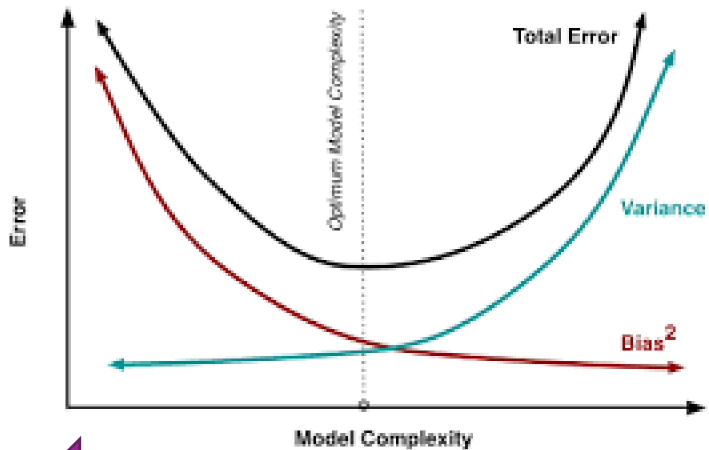
- ❑Polynomial ex
- ❑Mean and std dev of estimated functions
- ❑100 trials



Low variance,
High bias

High variance,
Zero bias

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

Bias-Variance in Linear Models

- Computing bias and variance with noise is beyond this class
  - Take the probability or detection and estimation class!
  - Not hard, but need a little more work

- This class: Only compute bias in the noise-free case in this class

- We state some results from probability without proof

Results for Linear Models (No proof)

- Suppose model is linear with $n =$ num samples, p = num parameters
- Result 1: When $n < p$, linear estimate is not unique
  - Need at least as many samples as parameters
- Now assume that $n \geq p$ and parameter estimate is unique
- Result 2: When there is no under-modeling, estimate is unbiased
  - Mean estimate will always match "true" parameter.
- Result 3: For $n \gg p$ and test point drawn from same distribution as training data:

$$Var(\boldsymbol{x}_{test}) = \frac{p}{n}\sigma_\epsilon^2$$

  - Variance increases linearly with number of parameters and inversely with number of samples

Bias-Variance Formula (Advanced)

- Consider test point $\boldsymbol{x}_{test}$
- Bias: $Bias(\boldsymbol{x}_{test}) := f_0(\boldsymbol{x}_{test}) - E\left[f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)\right]$
  - Represents under-modeling
- Variance: $Var(\boldsymbol{x}_{test}) \coloneqq E\left[E\left[f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)\right] - f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)\right]^2$
  - Represents effect of noise
- Mean-squared error: $MSE(\boldsymbol{x}_{test}) \coloneqq E\left[f_0(\boldsymbol{x}_{test}) - f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)\right]^2$
- Bias-Variance formula

$$\mathrm{MSE}(\boldsymbol{x}_{test}) = \mathrm{Bias}^2(\boldsymbol{x}_{test}) + \mathrm{Var}(\boldsymbol{x}_{test})$$

  - See proof in text

Outline

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

Cross Validation

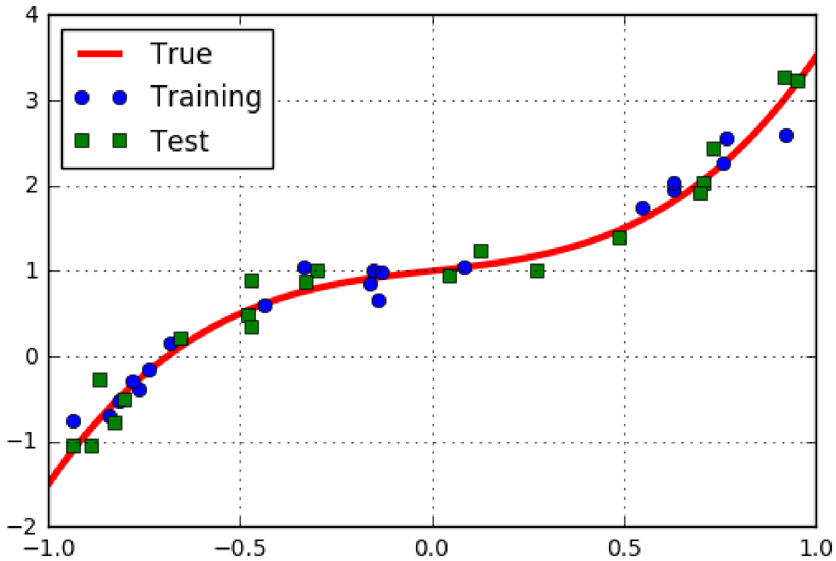
- Concept: Need to test fit on data independent of training data
- Divide data into two sets:
  - N_{train} training samples, N_{test} test samples
- For each model order, p, learn parameters $\hat{\beta}$ from training samples
- Measure RSS on test samples.

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

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

Polynomial Example: Training Test Split

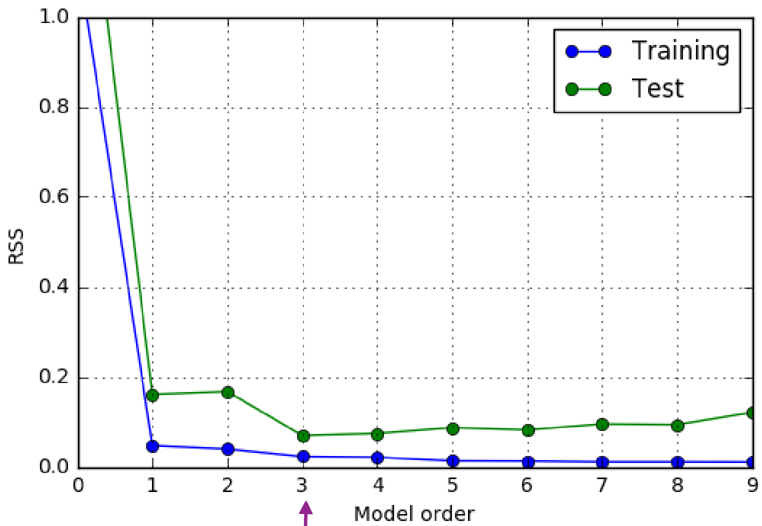
- Example: Split data into 20 samples for training, 20 for test




```python
# 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 significantly 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)
```

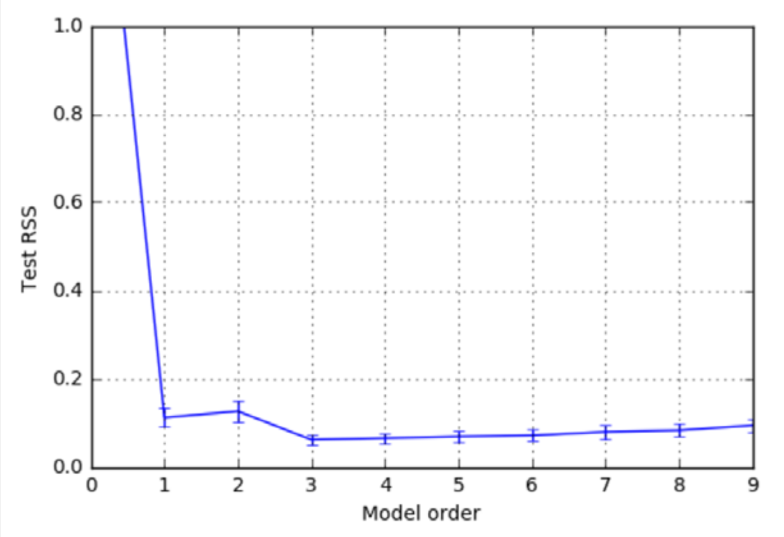
Polynomial Example CV Results

- For each model order d
  - Compute mean test RSS
  - Compute std error (SE) of test RSS
  - SE = std dev / $\sqrt{K-1}$
  - Mean and SE computed over the K folds

- Simple model selection
  - Select d with lowest mean test RSS

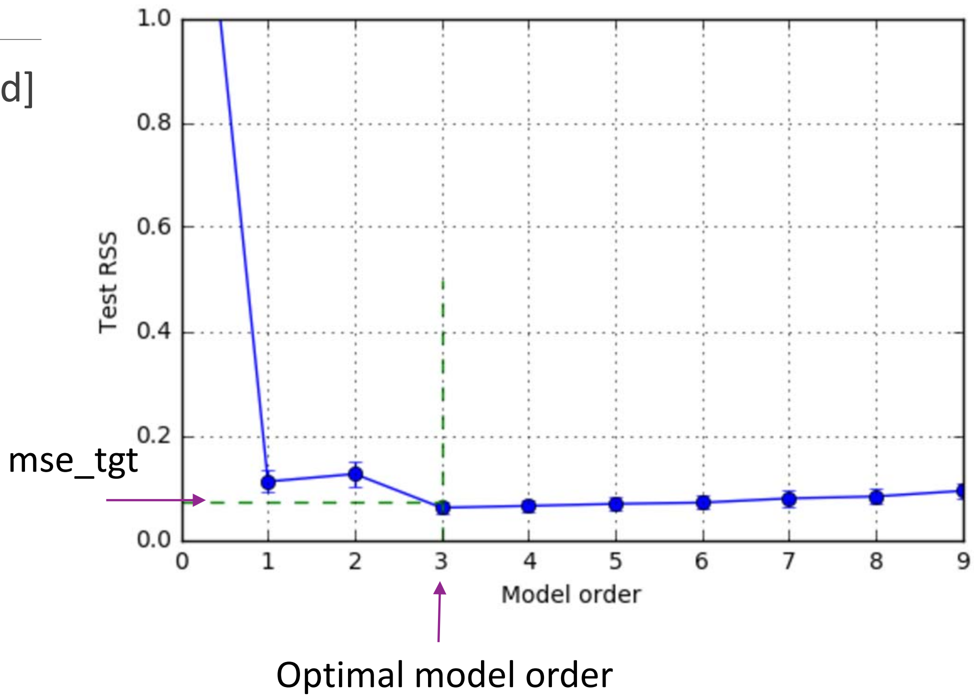
- For this example
  - Estimate model order = 3

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



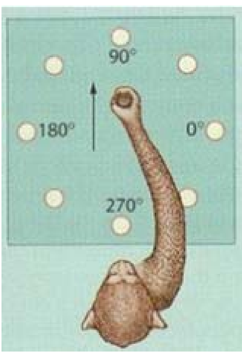
One Standard Error Rule

- ❑Previous slide: Select d to minimize mse_mean[d]
- ❑Problem: Often over-predicts model order
- ❑One standard deviation rule
  - Use simplest model within one SE 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

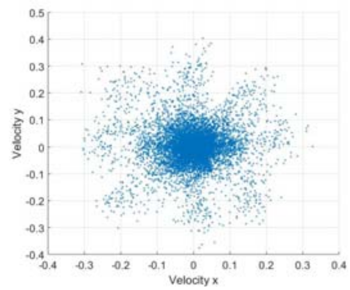


Lab: Neural ECoG Data

Record arm motion



Velocity y

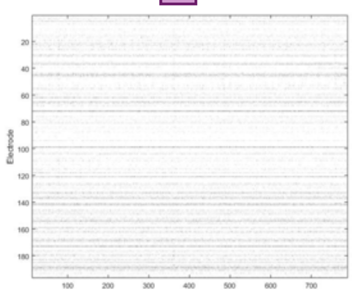


Velocity x

Arm velocity

?

Neuron



Time

Spikes

Record spikes (ECoG in M1)

- ❑Read a monkey's brain!
- ❑Predict hand motion from electrode measurements
- ❑Model order selection
  - Which signals predict arm motion?
- ❑Data from https://crcns.org/
  - Open source website on neural data
  - Great for projects