Understanding Experimental Data, cont.

Eric Grimson

MIT Department Of Electrical Engineering and Computer Science

Remember our goal

- Want to find a model that fits experimental data well
- Model will then allow us to explain phenomena, and to make predictions about behavior in new settings
- Know that data is unlikely to be perfect, so have to account for uncertainty in measurements or observations
- Sometimes have theoretical knowledge of structure of model, but not always
    - In latter case, want to try to find best model from class of options

Solving for Least Squares (Recap)

$$\sum_{i=0}^{len(observed)-1} (observed[i] - predicted[i])^2$$

- Given observed data, and model prediction of expected values, can measure goodness of fit of model to observation using sum-of-squared-differences (or mean-squared-error)
- Want to find best model for predicting values
- Predicted values often come from mathematical expression, with set of parameters that can vary – typically a polynomial expression
- Use linear regression to find best model that minimizes difference – for polynomial model, this include coefficients, and may include order of polynomial

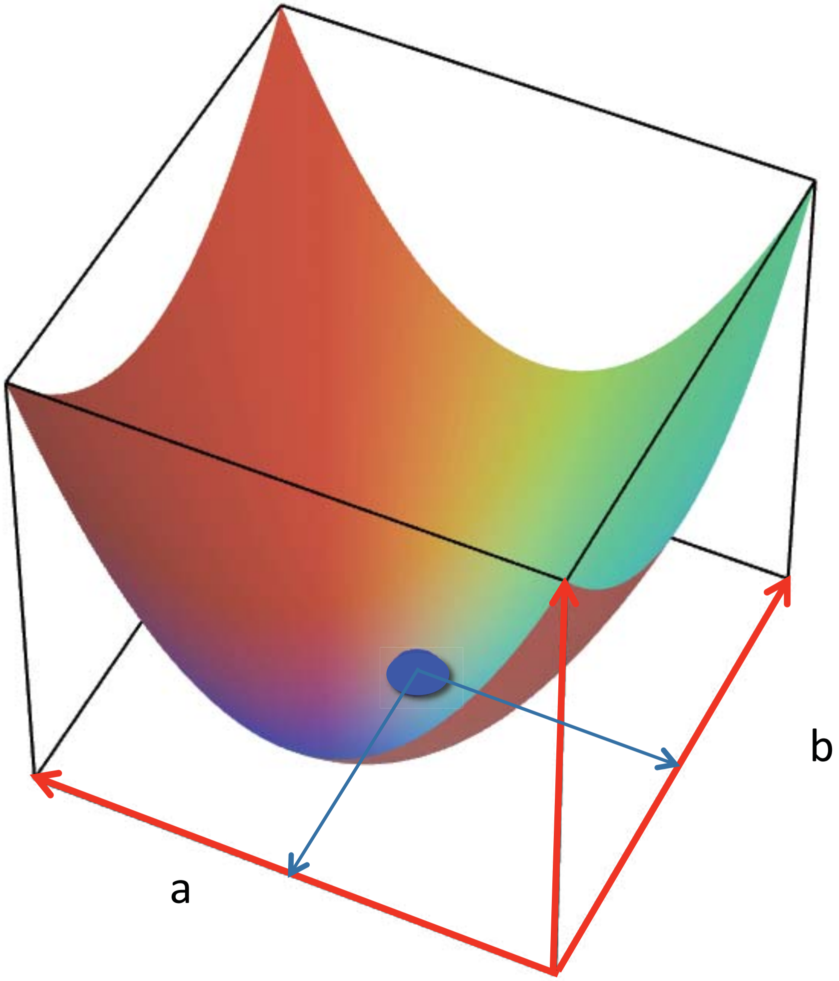
Solving for Least Squares (Recap)

$$\sum_{i=0}^{len(observed)-1} (observed[i] - predicted[i])^2$$

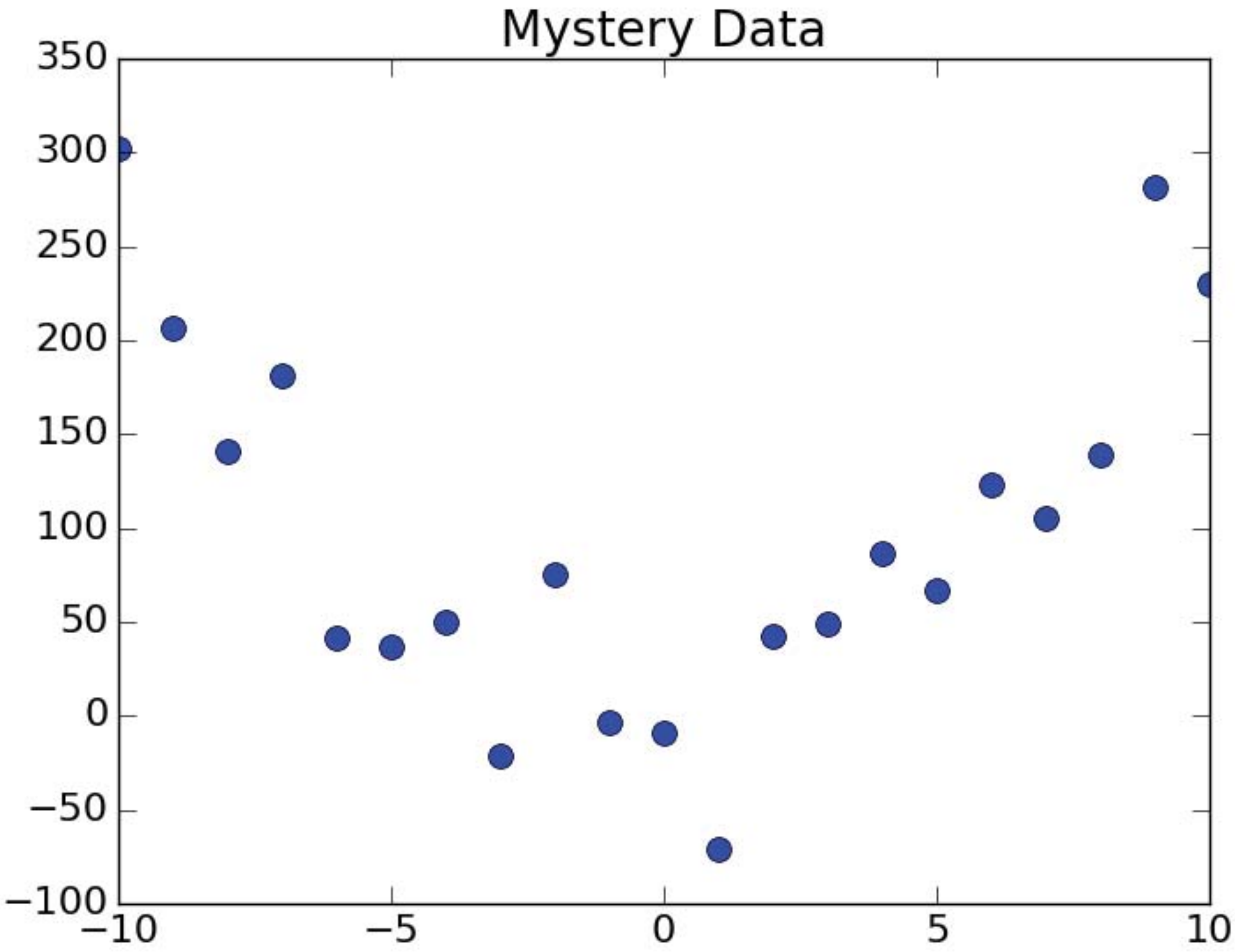
- Simple example:
  - Use a degree-one polynomial, *y = ax+b*, as model of our data (we want best fitting line)
- Find values of *a* and *b* such that when we use the polynomial to predict *y* values for all of the *x* values in our experiment, the squared difference of these values and the corresponding observed values is minimized
- A **linear regression** problem

Finding the best curve (simplest case)

- The set of all possible lines can be represented by a point in a-b space
- Imagine a surface in this space, where height of the surface is the value of the objective function
- Starting at any point on the surface, walk “downhill”, until you reach the “bottom”
- Corresponding point is best line to fit to data
- Can generalize to higher order models



Another Experiment (Recap)

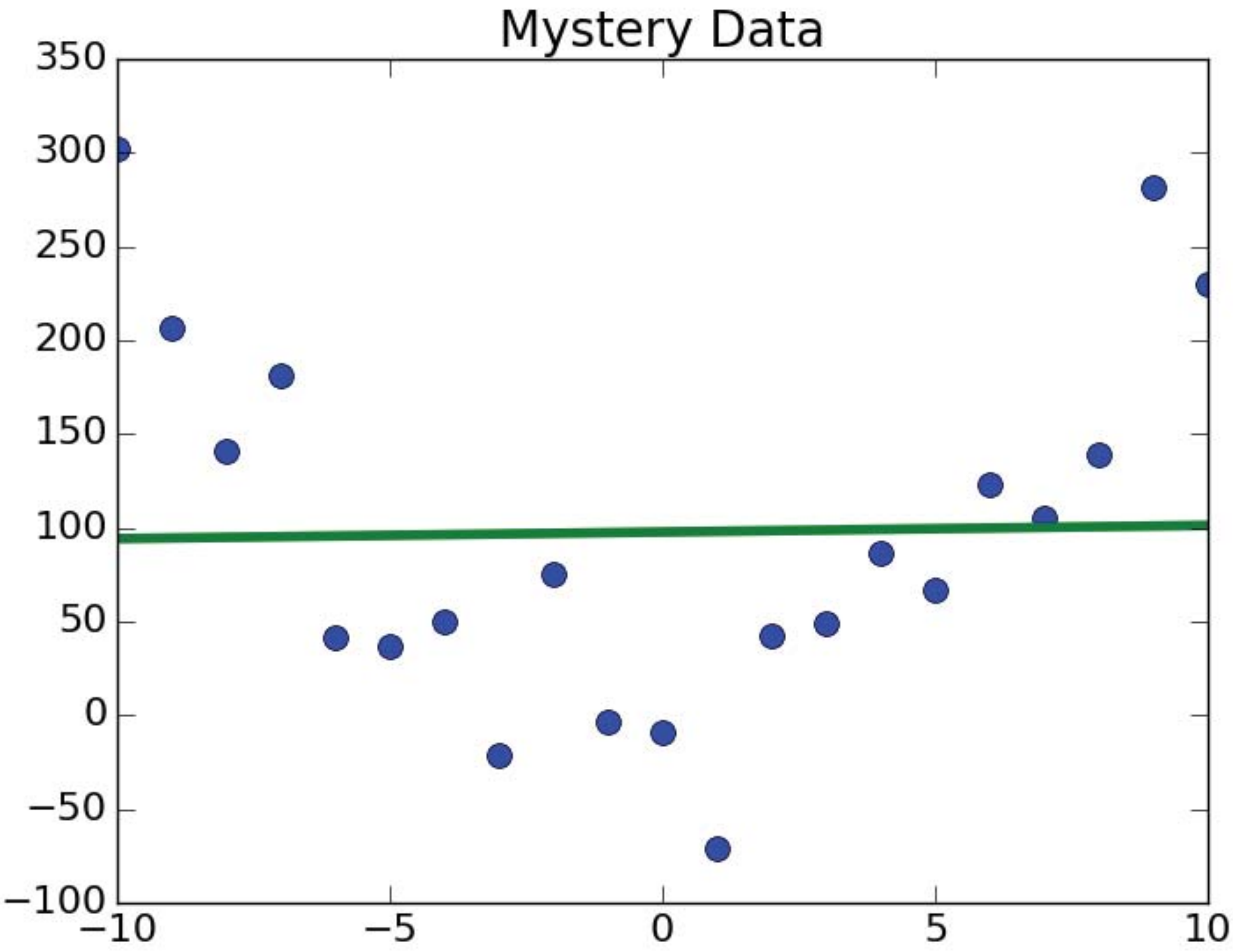


Fit a Line

```
model1 = pylab.polyfit(xVals, yVals, 1)
pylab.plot(xVals, pylab.polyval(model1, xVals),
           'r--', label = 'Linear Model')
```

- Remember that `pylab.polyfit` will find parameters of best fitting polynomial of described order
    - In this case (with argument n = 1), find the values of *a* and *b*, such that *y = ax + b* best matches the observed yVals
- Remember that `pylab.polyval` will generate predicted yVals given parameters of model

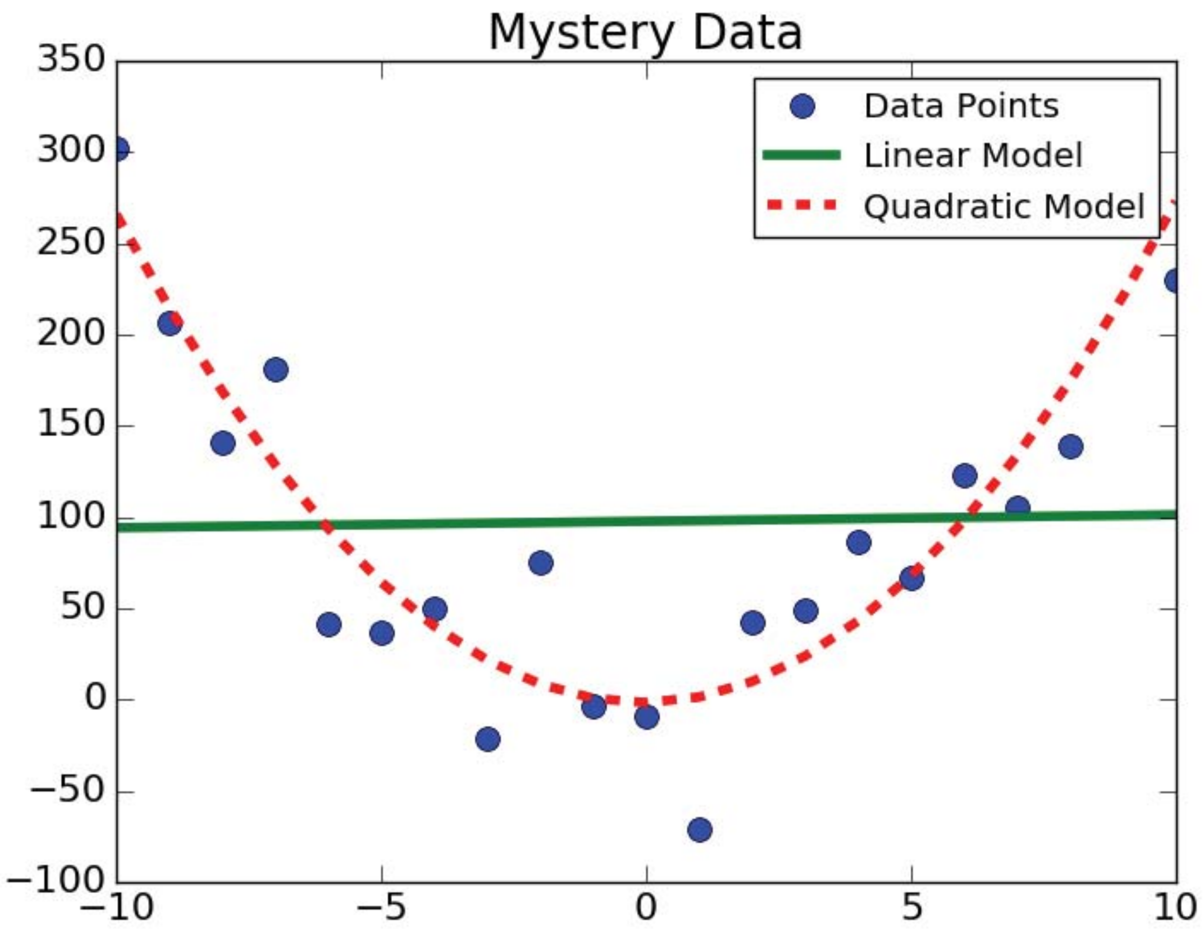
Fit a Line



Let’s Try a Higher-degree Model

```
model2 = pylab.polyfit(xVals, yVals, 2)
pylab.plot(xVals, pylab.polyval(model2, xVals),
           'r--', label = 'Quadratic Model')
```

Quadratic Appears to be a Better Fit



Can We Get a Tighter Fit?

- What if we try fitting higher order polynomials to the data?
  - Does this give us a better fit?

- How would we measure that?
  - In absence of other information (e.g., theoretical insights into order of model), R^2 (coefficient of determination) gives us decent measure of the tightness of the model fit
  - In principle, a model with a higher R^2 value is a “better” fit

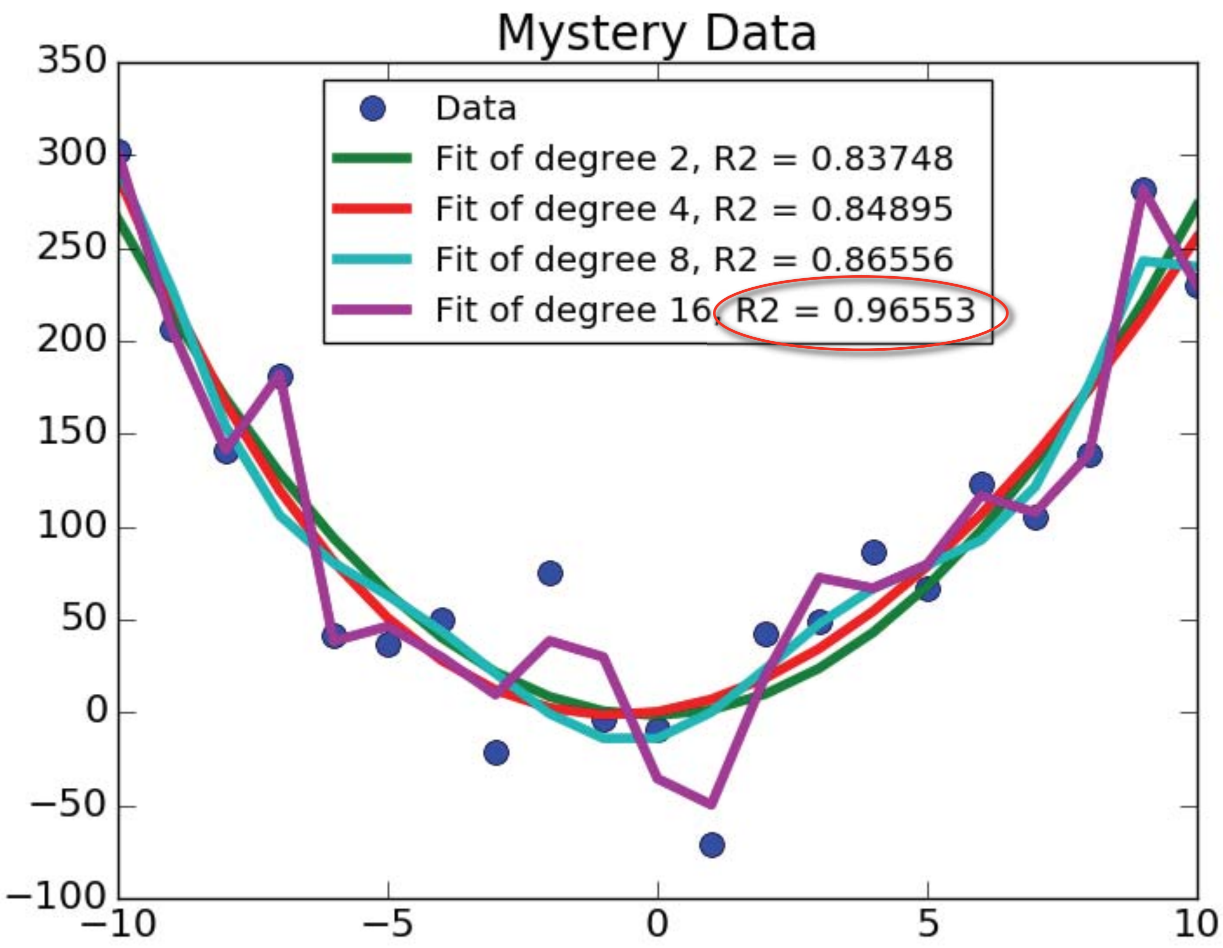
$$R^2 = 1 - \frac{\sum_i (y_i - p_i)^2}{\sum_i (y_i - \mu)^2}$$

← Error in estimates

← Variability in measured data

Y_i are measured values
P_i are predicted values
μ is mean of measured values

Can We Get a Tighter Fit?

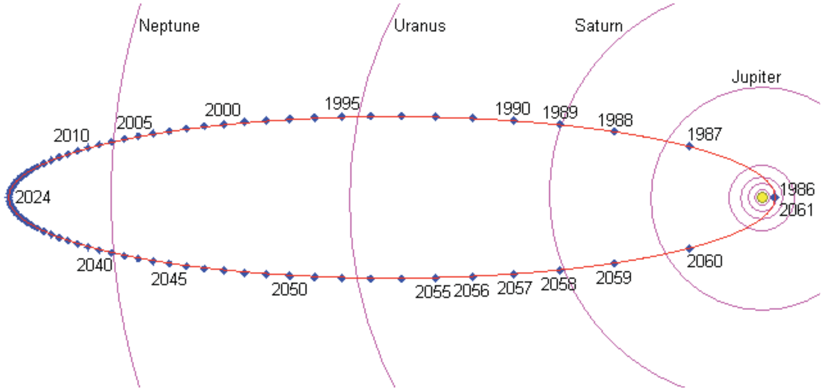


Why We Build Models

- Looks like an order 16 fit is really good – so should we just use this as our model?
  - To answer, need to ask – why build models in first place?
- Help us understand process that generated the data
  - E.g., the properties of a particular linear spring
- Help us make predictions about out-of-sample data
  - E.g., predict the displacement of a spring when a force is applied to it
  - E.g., predict the effect of treatment on a patient
  - E.g., predict the outcome of an election
- A good model helps us do both of these things

Motivation for Mystery Data – Parabola

- Trajectory of a particle under the influence of a uniform gravitational field (e.g. Halley's Comet)
- Position of center of mass of a football pass
- Design of a load-bearing arch



How Mystery Data Was Generated

```
def genNoisyParabolicData(a, b, c, xVals, fName):
    yVals = []
    for x in xVals:
        theoreticalVal = a*x**2 + b*x + c
        yVals.append(theoreticalVal + random.gauss(0, 35))
    f = open(fName,'w')
    f.write('x        y\n')
    for i in range(len(yVals)):
        f.write(str(yVals[i]) + ' ' + str(xVals[i]) + '\n')
    f.close()

#parameters for generating data
xVals = range(-10, 11, 1)
a, b, c = 3, 0, 0
genNoisyParabolicData(a, b, c, xVals,  ’Mystery Data.txt')
```

If data was generated by quadratic, why was 16th order polynomial the “best” fit?

Let’s Look at Two Data Sets

```
degrees = (2, 4, 8, 16)

random.seed(0)
xVals1, yVals1 = getData('Dataset 1.txt')
```

model 1

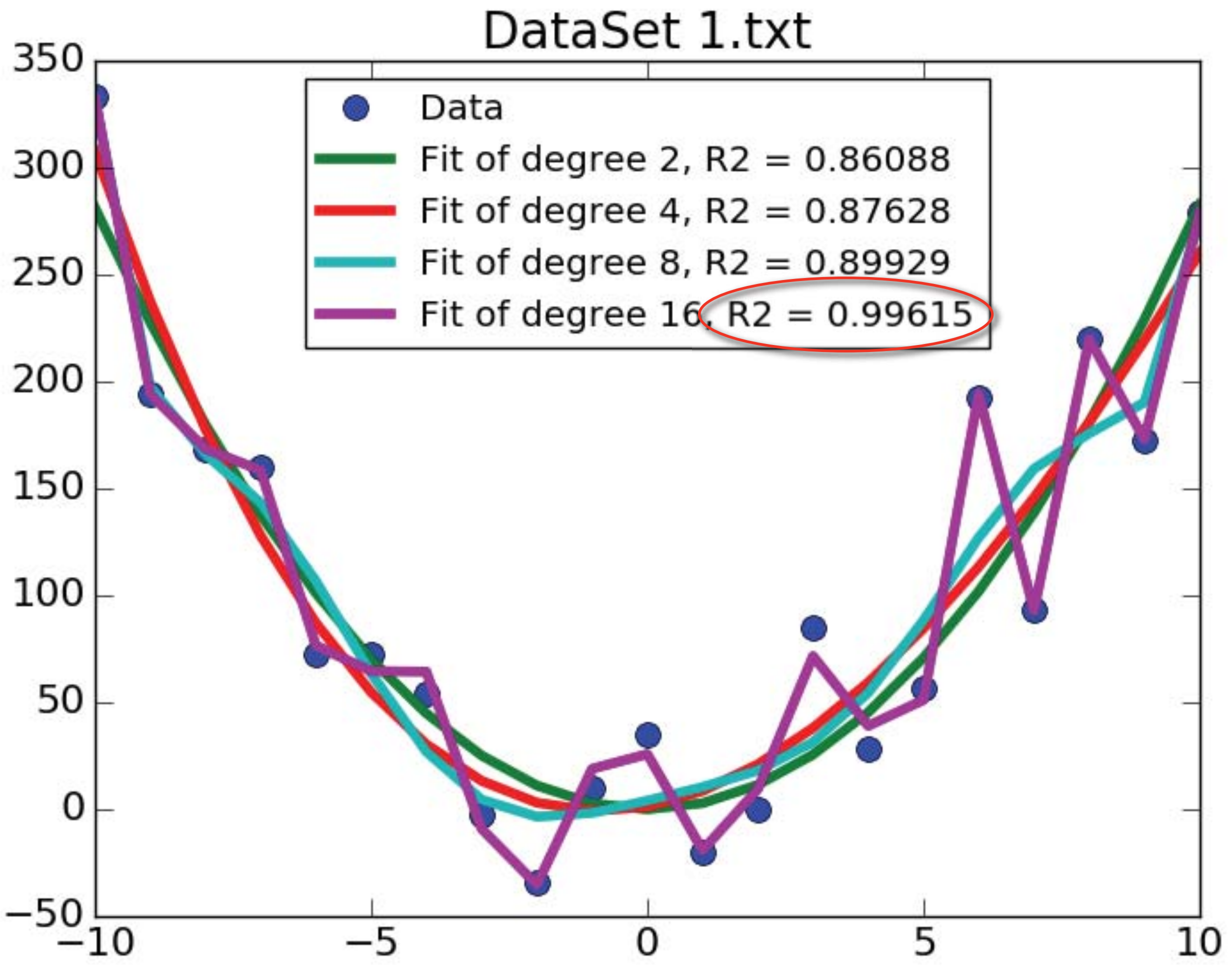
```
models1 = genFits(xVals1, yVals1, degrees)
testFits(models1, degrees, xVals1, yVals1,
        'DataSet 1.txt')

pylab.figure()
```

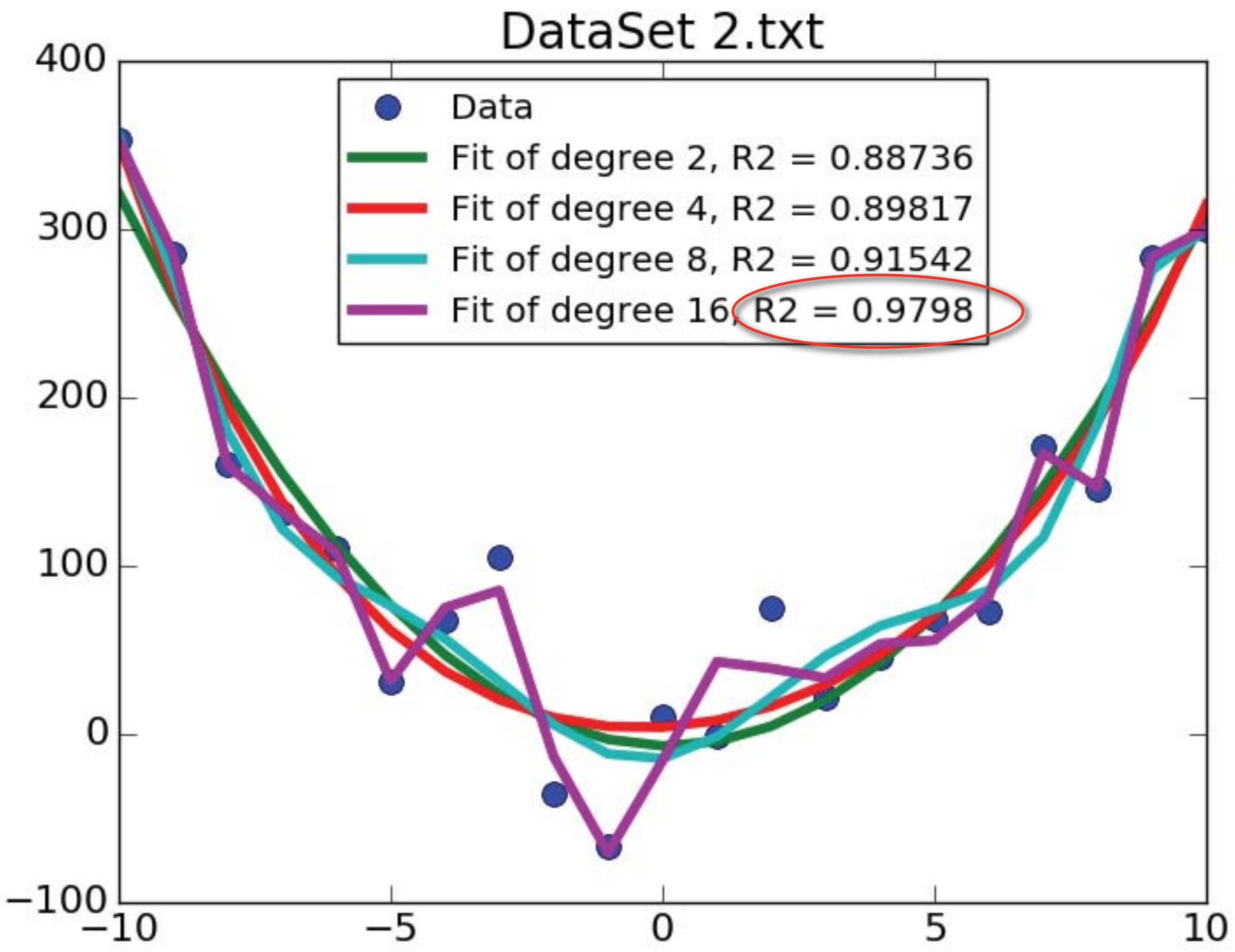
model2.

```
xVals2, yVals2 = getData('Dataset 2.txt')
models2 = genFits(xVals2, yVals2, degrees)
testFits(models2, degrees, xVals2, yVals2,
         'DataSet 2.txt')
```

Fits for Dataset 1



Fits for Dataset 2



Hence Degree 16 Is Tightest Fit

- "Best" fitting model is still order 16 polynomial for both data sets, **but** we know data was generated using an order 2 polynomial?
- What we are seeing comes from training error
  - How well the model performs on the data from which it was learned
  - Small training error a necessary condition for a great model, **but not a sufficient one**
- We want model to work well on other data generated by the same process
  - Measurements for other weights on the spring
  - Positions of comets under different forces
  - Voters other than those surveyed
- In other words, the model needs to generalize

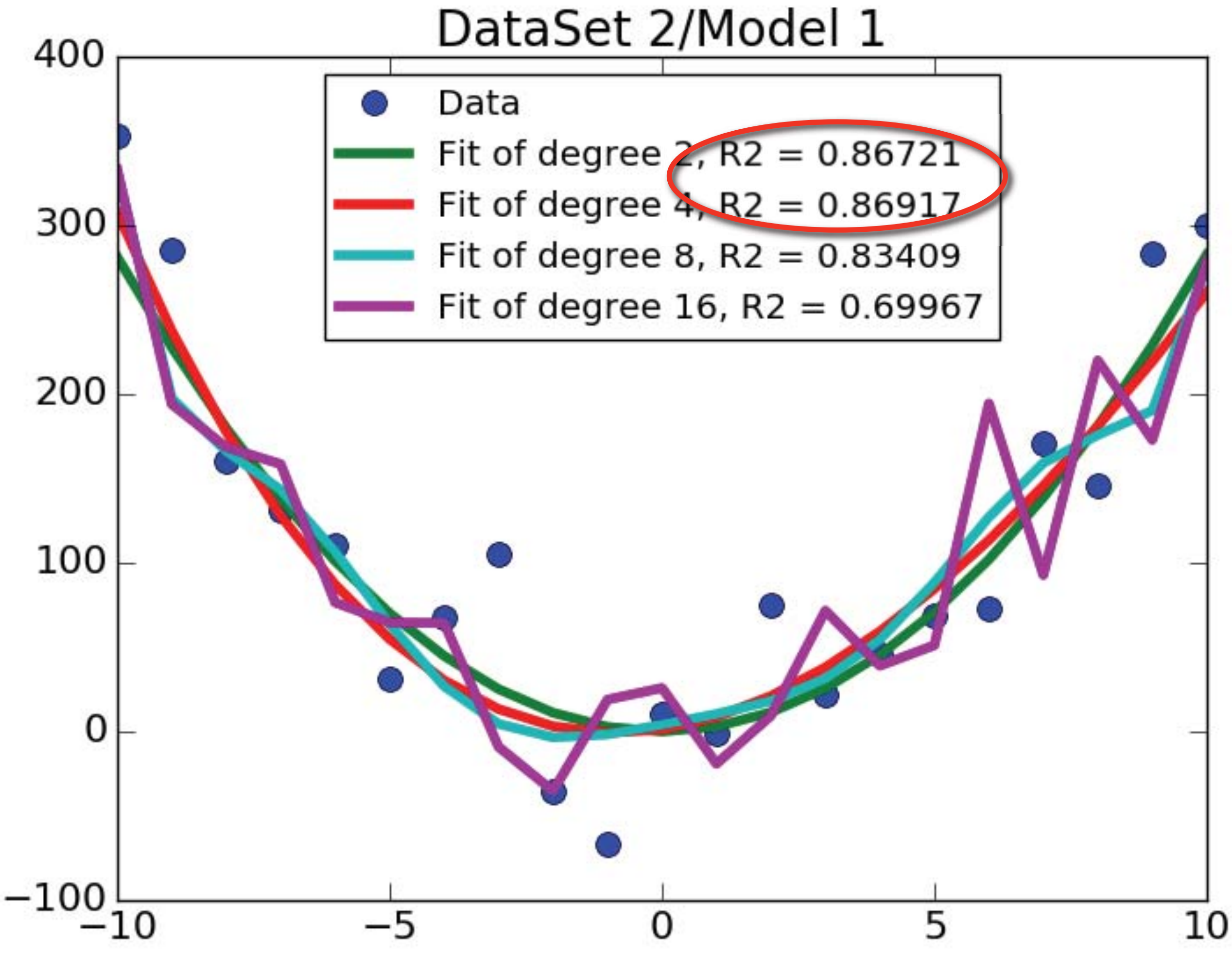
Cross Validate

- Generate models using one dataset, and then test them on another dataset
  - Use models for Dataset 1 to predict points for Dataset 2
  - Use models for Dataset 2 to predict points for Dataset 1
- Expect testing error to be larger than training error
- A better indication of generalizability than training error

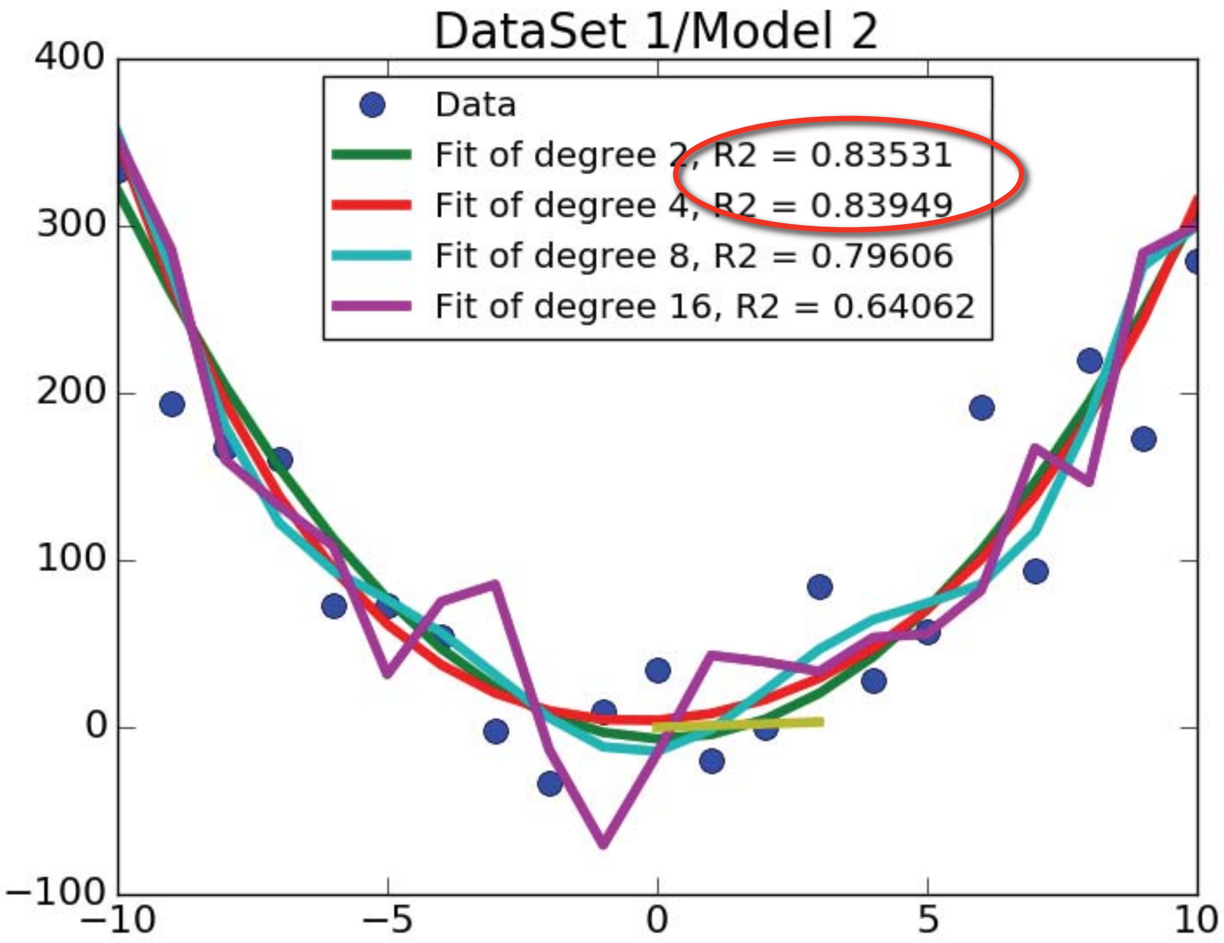
Test Code

```
pylab.figure()
testFits(models1, degrees, xVals2, yVals2,
         'DataSet 2/Model 1')
pylab.figure()
testFits(models2, degrees, xVals1, yVals1,
         'DataSet 1/Model 2')
```

Train on Dataset 1, Test on Dataset 2



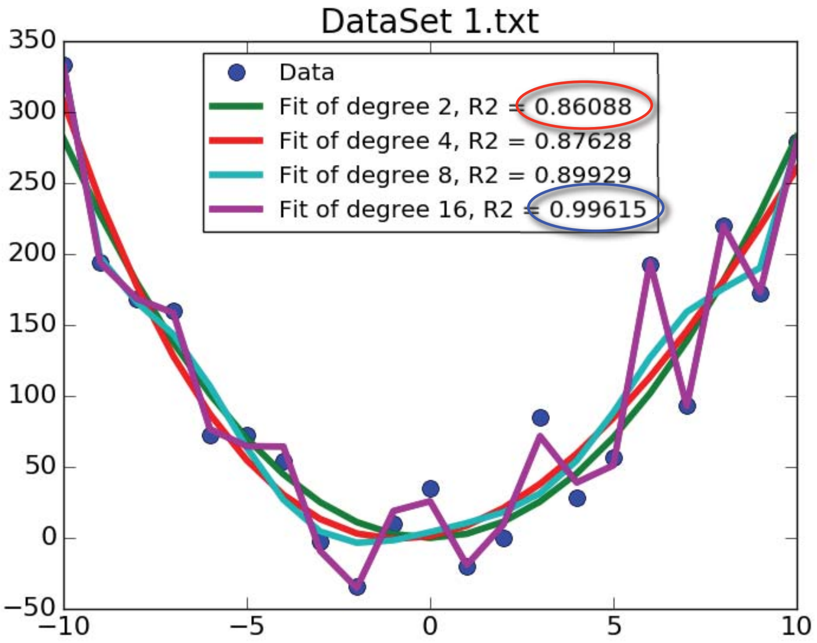
Train on Dataset 2, Test on Dataset 1

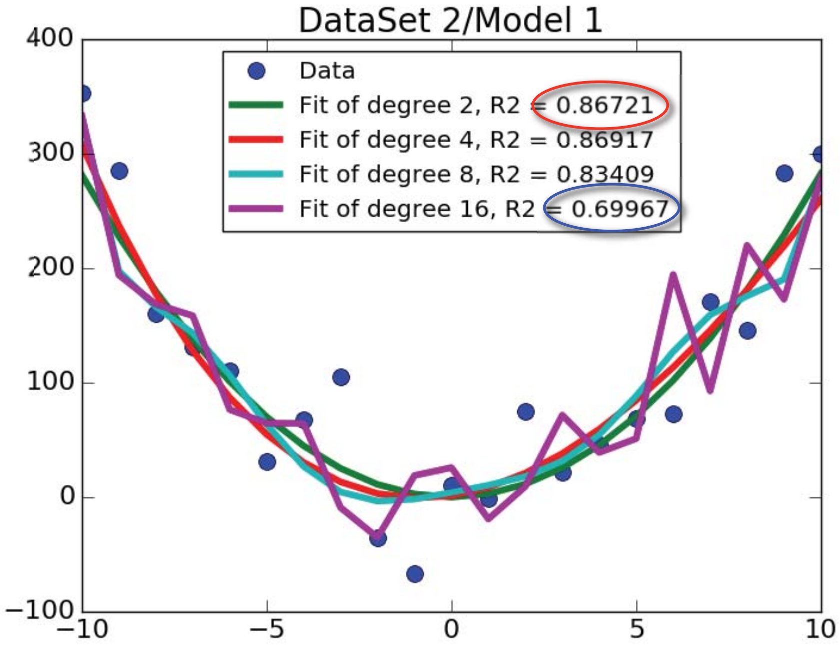


Cross Validation

- Now can see that based on R^2 numbers, best model is more likely to be 2nd order or 4th order polynomial (we know it is actually 2nd order, and difference in R^2 values is pretty small), but certainly not 16th order
- Example of over fitting to the data
- Can see that if we only fit model to training data, we may not detect that model is too complex; but training on one data set, then testing on a second helps expose this problem

Training and Testing Errors





Increasing the Complexity

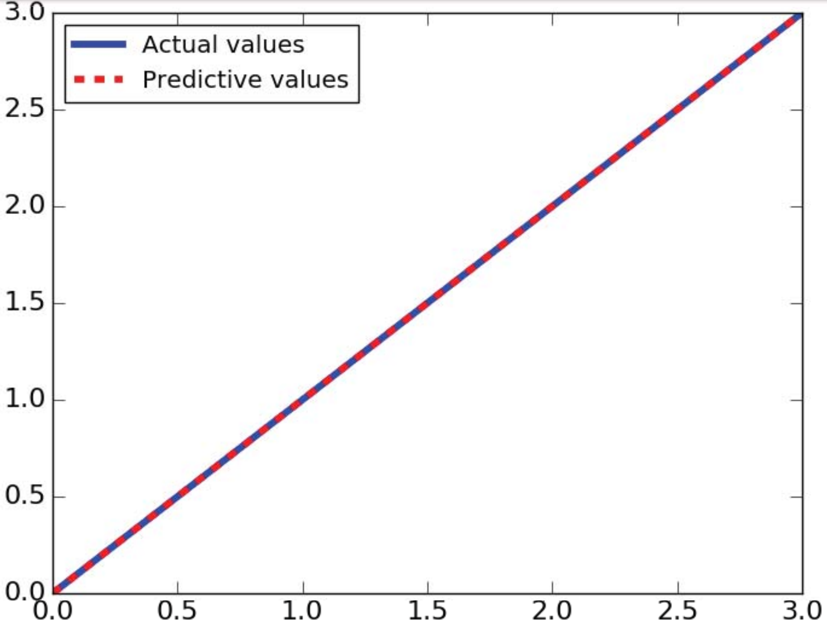
- Why do we get a "better" fit on training data with higher order model, but then do less well on handling new data?
- What happens when we increase order of polynomial during training?
  - Can we get a worse fit to training data?
- If extra term is useless, coefficient will merely be zero
- But if data is noisy, can fit the noise rather than the underlying pattern in the data
  - May lead to a "better" R^2 value, but not really a "better" fit

Fitting a Quadratic to a Perfect Line

```
xVals = (0,1,2,3)
yVals = xVals
pylab.plot(xVals, yVals, label = 'Actual values')
```

```
a,b,c = pylab.polyfit(xVals, yVals, 2)
print('a =', round(a, 4), 'b =', round(b, 4),
        'c =', round(c, 4))
```

```
estYVals = pylab.polyval((a,b,c), xVals)
pylab.plot(xVals, estYVals, 'r--', label ='Predictive values')
print('R-squared = ', rSquared(yVals, estYVals))
```



$$y = ax^2 + bx + c$$

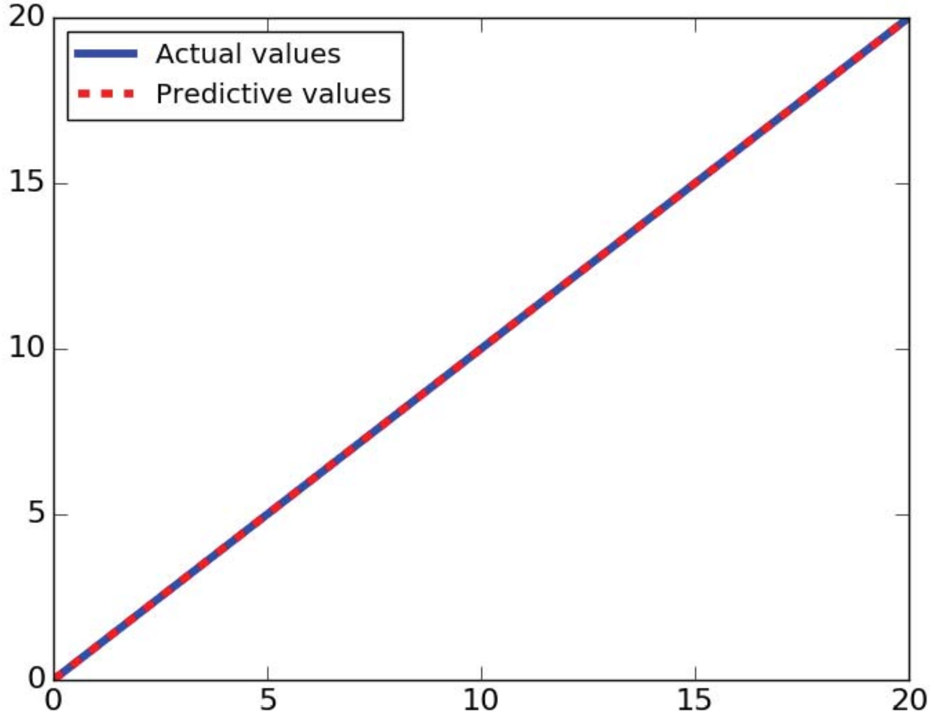
$$y = 0x^2 + 1x + 0$$

$$y = x$$

R-squared = 1.0

Predict Another Point Using Same Model

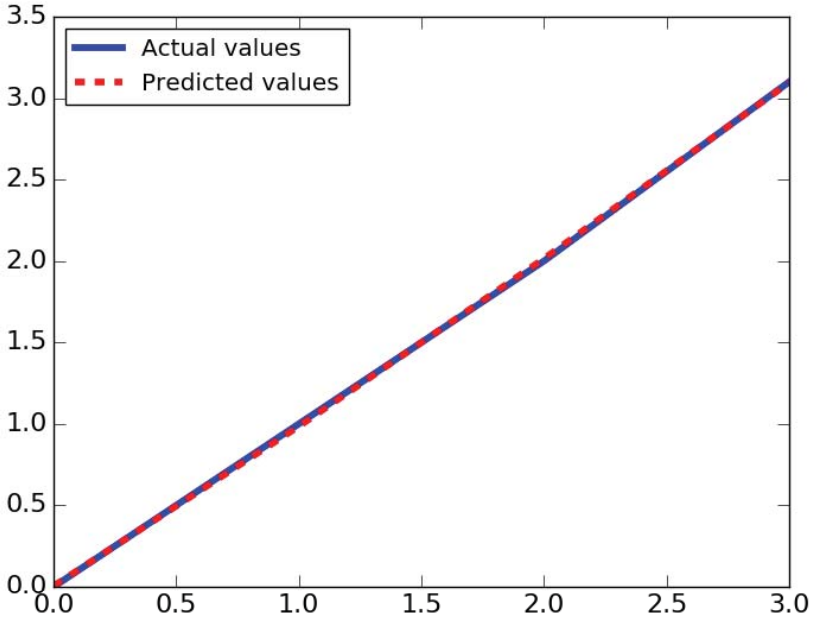
```
xVals = xVals + (20,)
yVals = xVals
pylab.plot(xVals, yVals, label = 'Actual values')
estYVals = pylab.polyval((a,b,c), xVals)
pylab.plot(xVals, estYVals, 'r--', label = 'Predictive values')
print('R-squared = ', rSquared(yVals, estYVals))
```



R-squared = 1.0

Simulate a Small Measurement Error

```
xVals = (0,1,2,3)
yVals = (0,1,2,3.1)
pylab.plot(xVals, yVals, label = 'Actual values')
model = pylab.polyfit(xVals, yVals, 2)
print(model)
estYVals = pylab.polyval(model, xVals)
pylab.plot(xVals, estYVals, 'r--', label = 'Predicted values')
print('R-squared = ', rSquared(yVals, estYVals))
```



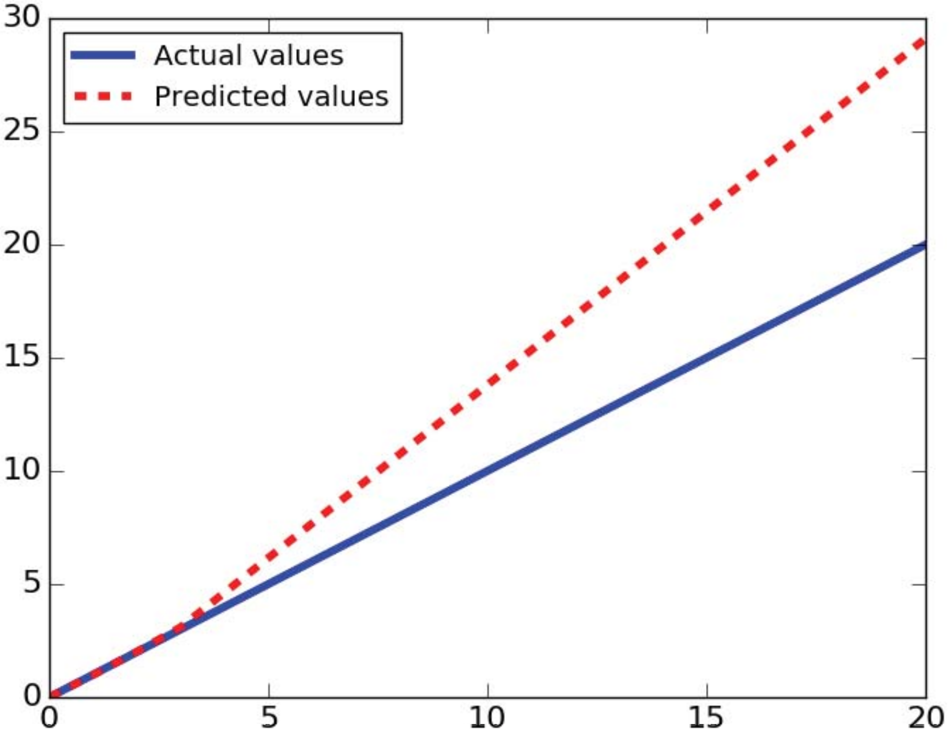
$$y = ax^2 + bx + c$$

$$y = .025x^2 + .955x + .005$$

R-squared = 0.9994

Predict Another Point Using Same Model

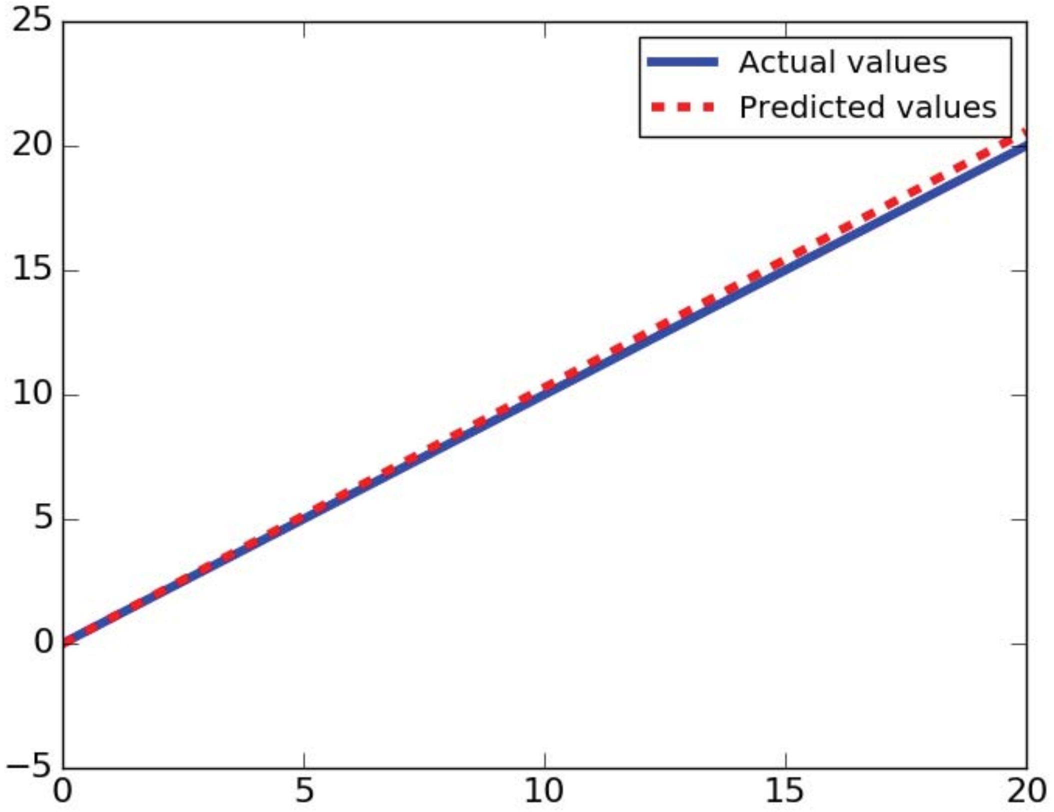
```
xVals = xVals + (20,)
yVals = xVals
estYVals = pylab.polyval(model, xVals)
print('R-squared = ', rSquared(yVals, estYVals))
pylab.figure()
pylab.plot(xVals, estYVals)
```



R-squared = 0.7026

Suppose We Had Used a First-degree Fit

- model = pylab.polyfit(xVals, yVals, 1)

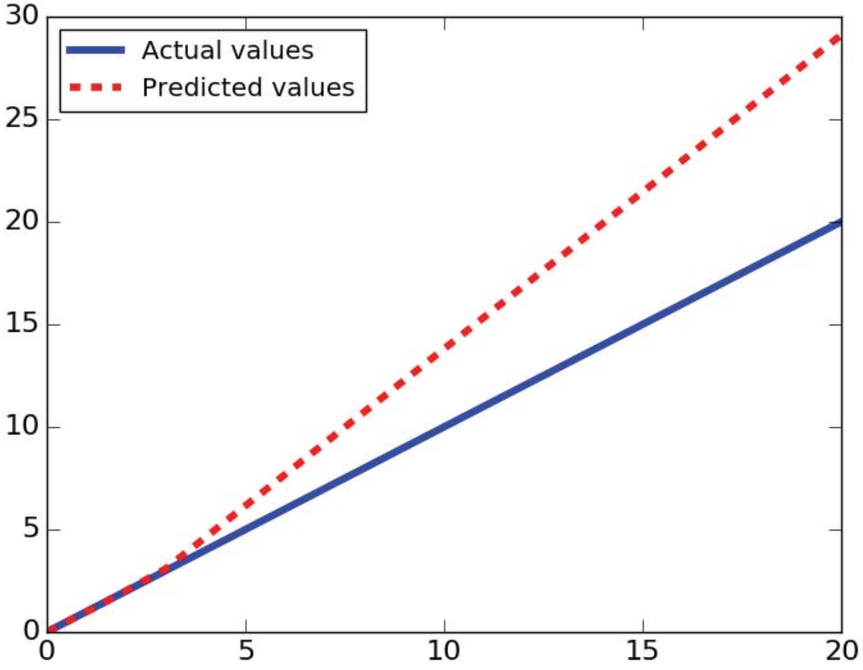


R-squared = 0.9988

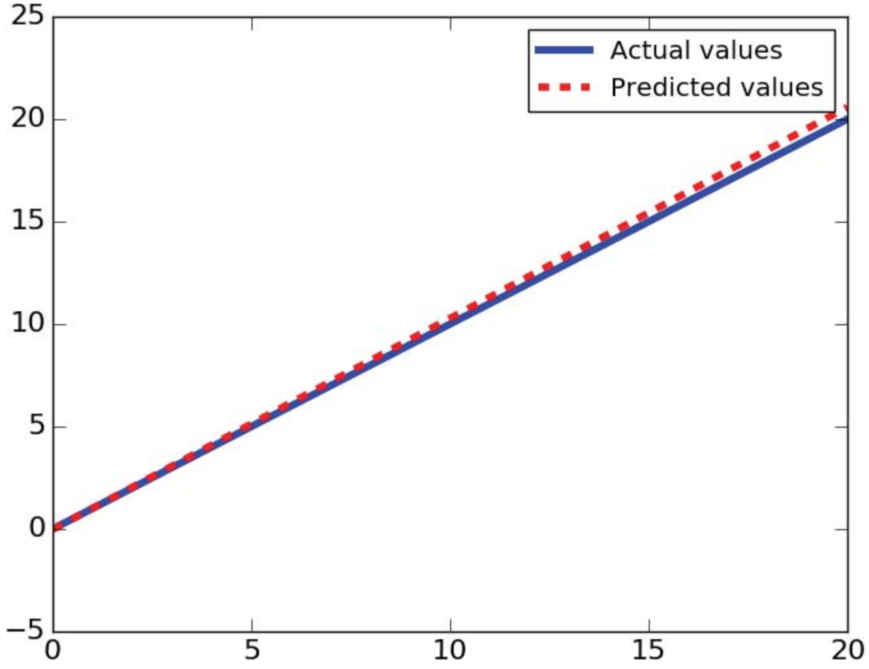
Comparing first and second degree fits

- Predictive ability of first order fit much better than second order fit

Degree 2 polynomial



Degree 1 polynomial

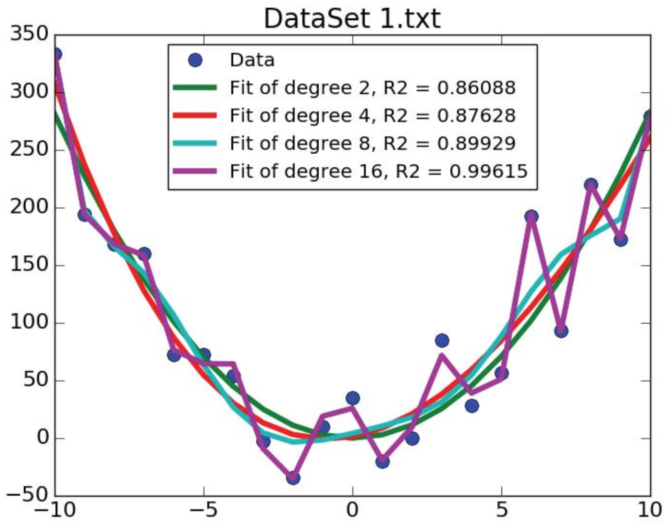


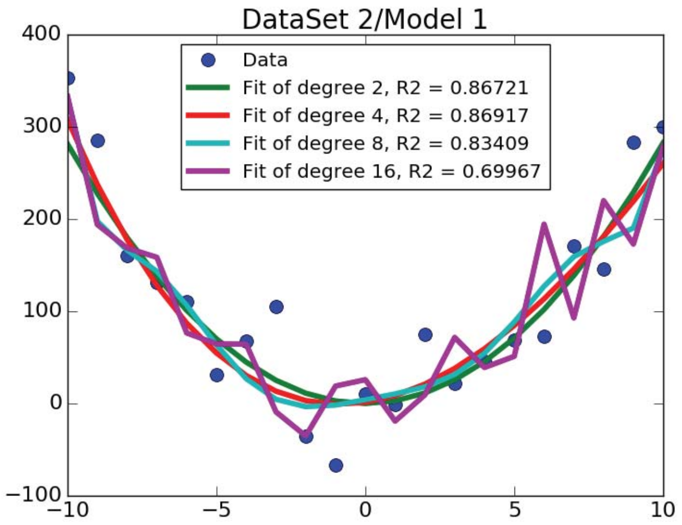
The Take Home Message

- Choosing an overly-complex model leads to overfitting to the training data
- Increases the risk of a model that works poorly on data not included in the training set
- On the other hand choosing an insufficiently complex model has other problems
    - As we saw when we fit a line to data that was basically parabolic
    - "Everything should be made as simple as possible, but not simpler" – Albert Einstein

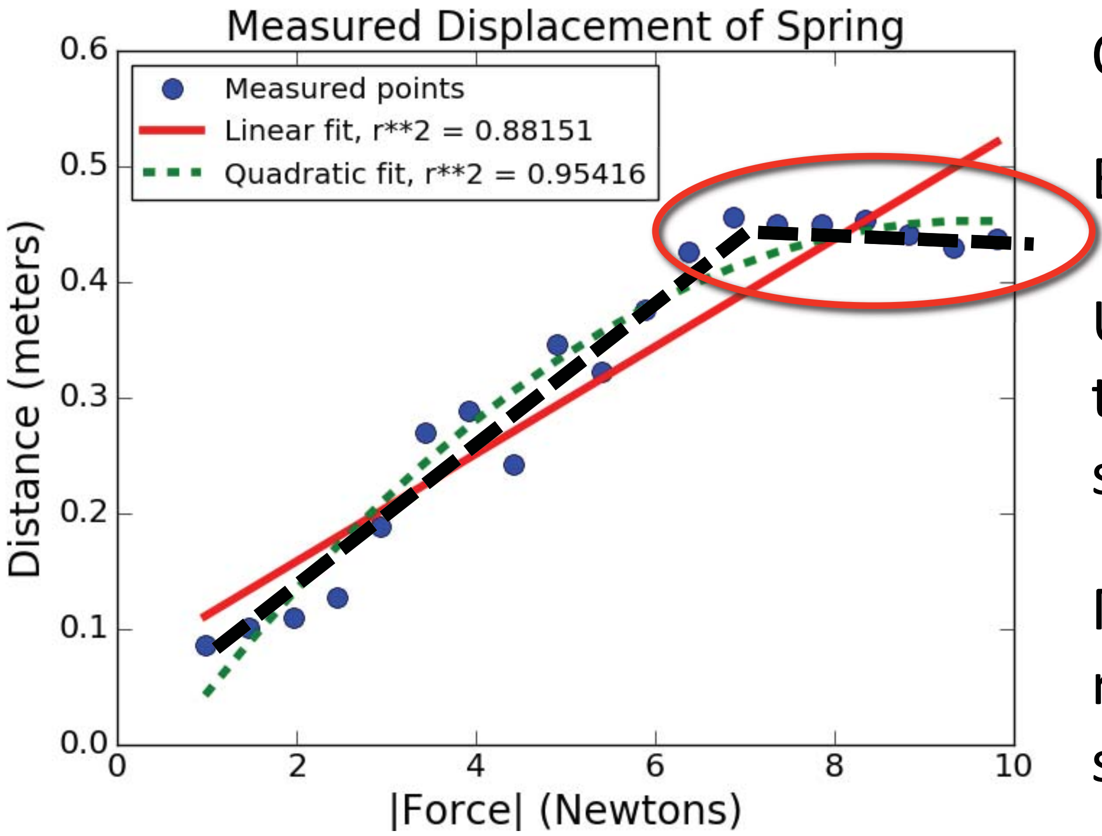
Balancing Fit with Complexity

- In absence of theory predicting order of model, can engage in a search process
  - Fit a low order model to training data
  - Test on new data and record R^2 value
  - Increase order of model and repeat
  - Continue until fit on test data begins to decline





Returning to Where We Started



Quadratic fit tighter

But remember Hooke

Unless we believe theory is wrong, that should guide us

Model holds until reach elastic limit of spring

Should probably fit different models to different segments of data

Can visualize as search process – find best place to break into two parts, such that both linear segments have high R^2 fits

Suppose We Don’t Have a Solid Theory

- Use cross-validation results to guide the choice of model complexity
- If dataset small, use leave-one-out cross validation
- If dataset large enough, use k-fold cross validation or repeated-random-sampling validation

Leave-one-out Cross Validation

```
Let D be the original data set

testResults = []
for i in range(len(D)):
    training = D[:].pop(i)
    model = buildModel(training)
    testResults.append(test(model, D[i]))

Average testResults
```

k-fold very similar

Applies when we have large amount of data

D partitioned into k equal size sets

Model trained on k-1 sets, and tested on remaining set

Repeated Random Sampling

```
Let D be the original data set
    n be the number of random samples
        usually n between 20% and 50%
    k be number of trials

testResults = []
for i in range(k)
    randomly select n elements for testSet,
       keep rest for training
    model = buildModel(training)
    testResults.append(test(model, testSet))

Average testResults
```

An Example, Temperature By Year

- Task: Model how the mean daily high temperature in the U.S. varied from 1961 through 2015
- Get means for each year and plot them
- Randomly divide data in half n times
  - For each dimensionality to be tried
    - Train on one half of data
    - Test on other half
    - Record r-squared on test data
- Report mean r-squared for each dimensionality

A Boring Class

```
class tempDatum(object):
    def __init__(self, s):
        info = s.split(',')
        self.high = float(info[1])
        self.year = int(info[2][0:4])
    def getHigh(self):
        return self.high
    def getYear(self):
        return self.year
```

Read Data

```
def getTempData():
    inFile = open('temperatures.csv')
    data = []
    for l in inFile:
        data.append(tempDatum(l))
    return data
```

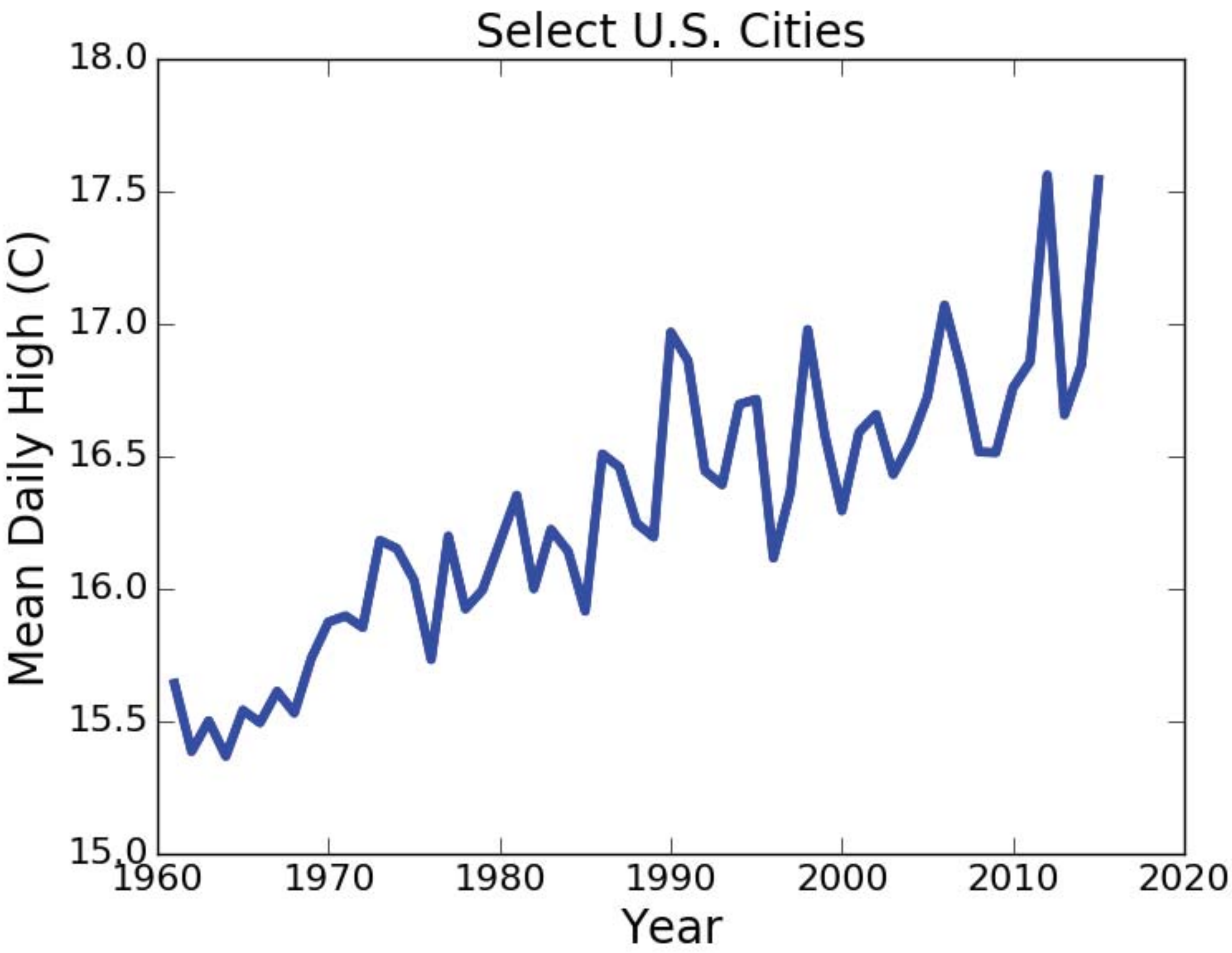
Get Means

```
def getYearlyMeans(data):
    years = {}
    for d in data:
        try:
            years[d.getYear()].append(d.getHigh())
        except:
            years[d.getYear()] = [d.getHigh()]
    for y in years:
        years[y] = sum(years[y])/len(years[y])
    return years
```

Get and Plot Data

```
data = getTempData()
years = getYearlyMeans(data)
xVals, yVals = [], []
for e in years:
    xVals.append(e)
    yVals.append(years[e])
pylab.plot(xVals, yVals)
pylab.xlabel('Year')
pylab.ylabel('Mean Daily High (C)')
pylab.title('Select U.S. Cities')
```

The Whole Data Set



Initialize Things

```
numSubsets = 10
dimensions = (1, 2, 3, 4)
rSquares = {}
for d in dimensions:
    rSquares[d] = []
```

Split Data

```python
def splitData(xVals, yVals):
    toTrain = random.sample(range(len(xVals)),
                            len(xVals)//2)
    trainX, trainY, testX, testY = [],[],[],[]
    for i in range(len(xVals)):
        if i in toTrain:
            trainX.append(xVals[i])
            trainY.append(yVals[i])
        else:
            testX.append(xVals[i])
            testY.append(yVals[i])
    return trainX, trainY, testX, testY
```

Train, Test, and Report

```
for f in range(numSubsets):
    trainX,trainY,testX,testY = splitData(xVals, yVals)
    for d in dimensions:
        model = pylab.polyfit(trainX, trainY, d)
        #estYVals = pylab.polyval(model, trainX)
        estYVals = pylab.polyval(model, testX)
        rSquares[d].append(rSquared(testY, estYVals))

print('Mean R-squares for test data')
for d in dimensions:
    mean = round(sum(rSquares[d])/len(rSquares[d]), 4)
    sd = round(numpy.std(rSquares[d]), 4)
    print('For dimensionality', d, 'mean =', mean,
          'Std =', sd)
```

Results

```
Mean R-squares for test data
For dimensionality 1 mean = 0.7535 Std = 0.0656
For dimensionality 2 mean = 0.7291 Std = 0.0744
For dimensionality 3 mean = 0.7039 Std = 0.0684
For dimensionality 4 mean = 0.7169 Std = 0.0777
```

- Line seems to be the winner
  - Highest average r-squared
  - Smallest deviation across trials
  - Simplest model

Why we should run multiple sets

- Note that deviations are a decimal order of magnitude smaller than means
  - Suggests that while there is good agreement, deviations are large enough there could be a noticeable range of variation across trials

- Suppose we had just run one trial
  - Here are the R^2 values for each trial of linear fit
    - [0.7828002156420516, 0.80637964025052067, 0.79637132757274265, 0.78433885743211906, 0.76001112024853124, 0.57088936507035748, 0.72115408562589023, 0.74358276762149023, 0.79031455375148507, 0.77920238586399471]
  - If we had only run one split, and happened to get **this** result, we might have reached a different conclusion about validity of linear model

Wrapping Up Curve Fitting

- We can use linear regression to fit a curve to data
  - Mapping from independent values to dependent values
- That curve is a model of the data that can be used to predict the value associated with independent values we haven't seen (out of sample data)
- R-squared used to evaluate model
  - Higher not always "better" because of risk of over fitting
- Choose complexity of model based on
  - Theory about structure of data
  - Cross validation
  - Simplicity