

# Understanding Experimental Data, cont.

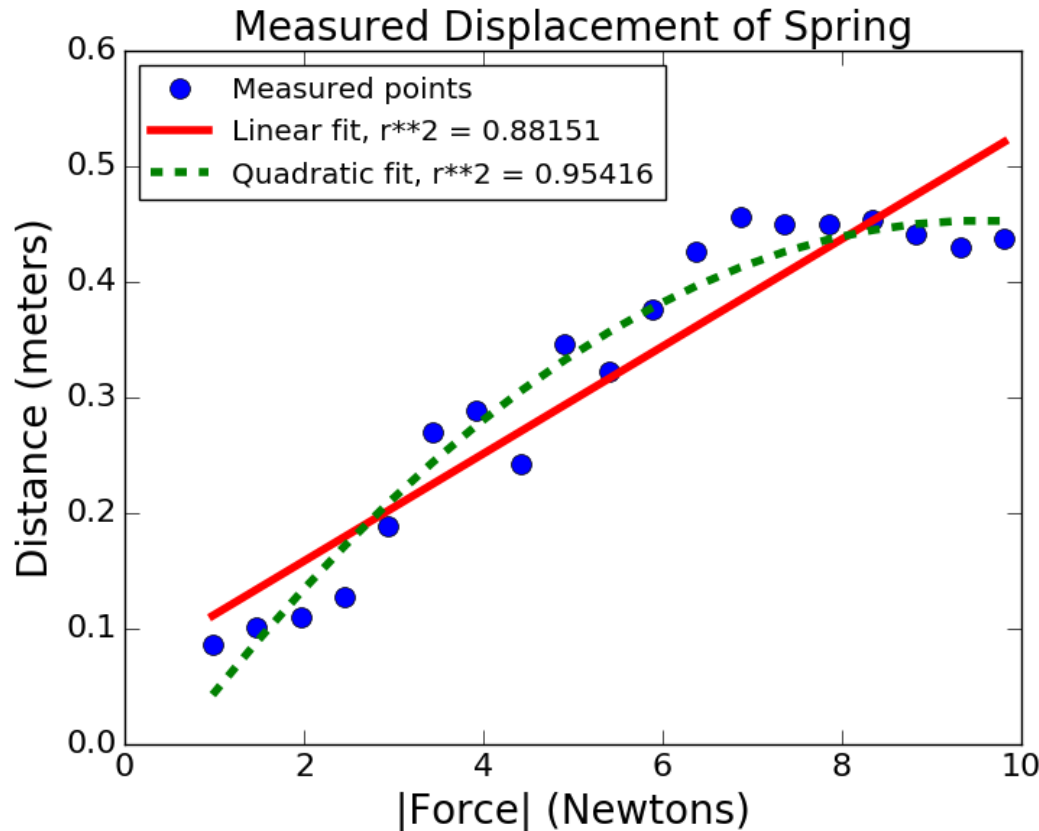
---

# 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

# Returning to Where We Started



Quartic fit tighter

But remember Hooke

Unless we believe  
theory is wrong, that  
should guide us

Holds up to elastic  
limit of spring

Should probably fit  
different models to  
different segments of  
data

# 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-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

$D$  partitioned into  $k$  equal size sets

Model trained on  $k-1$ , and tested on remaining

# Repeated Random Sampling

---

Let  $D$  be the original data set

$n$  be the number of random samples

```
testResults = []  
for i in range(n)  
    randomly partition  $D$  into two sets:  
        training and test  
    model = buildModel(training)  
    testResults.append(test(model, test))
```

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
```

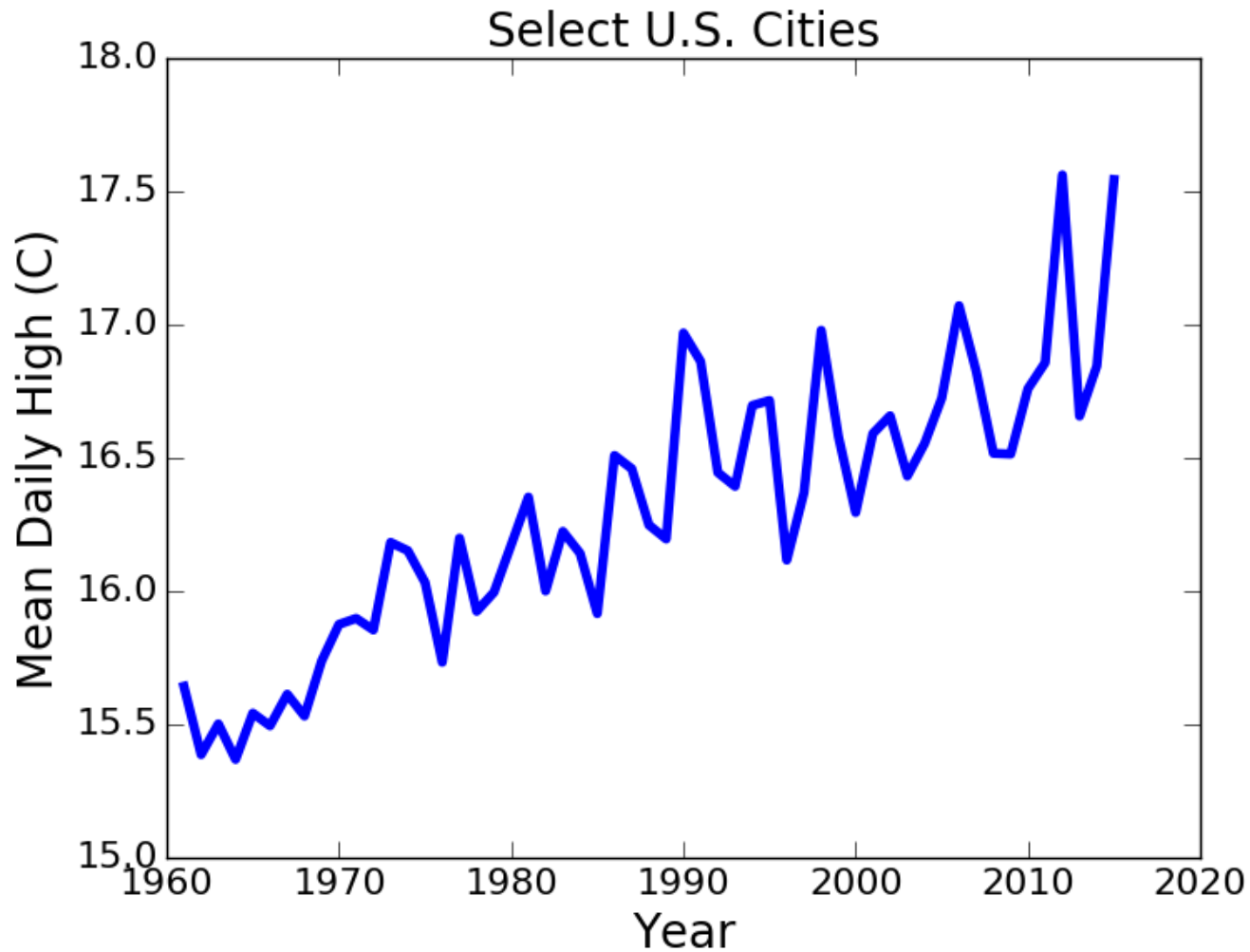
# Get and Plot Data

---

```
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)
rSquares = {}
for d in dimensions:
    rSquares[d] = []
```

# Split Data

---

```
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

- Line seems to be the winner
  - Highest average r-squared
  - Simplest 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

