Numeric Regression

葉建華

jhyeh@mail.au.edu.tw

http://jhyeh.csie.au.edu.tw/

Linear Regression

Linear regression

Pros: Easy to interpret results, computationally inexpensive

Cons: Poorly models nonlinear data

Works with: Numeric values, nominal values

Regression

- To predict a numeric target value
- A simple way: an equation for the target value with respect to the inputs (regression equation)
  - E.g. forecast the horsepower of your sister’s boyfriend’s automobile

```
HorsePower =

0.0015*annualSalary - 0.99*hoursListeningToPublicRadio
```

  - 0.0015 and 0.99 are called regression weights
- Regression: find regression weights!
- Nonlinear regression: not linear combination

General Approach

General approach to regression

1. Collect: Any method.
2. Prepare: We'll need numeric values for regression. Nominal values should be mapped to binary values.
3. Analyze: It's helpful to visualized 2D plots. Also, we can visualize the regression weights if we apply shrinkage methods.
4. Train: Find the regression weights.
5. Test: We can measure the R2, or correlation of the predicted value and data, to measure the success of our models.
6. Use: With regression, we can forecast a numeric value for a number of inputs. This is an improvement over classification because we're predicting a continuous value rather than a discrete category.

Explanation

- Input data matrix **X**, regression weights vector w

$$y_1 = X^T{}_1 w$$

- Use error minimization to find w

$$\sum_{i=1}^{m} (y_i - x_i^T w)^2 \quad \mathbf{X}^T(\mathbf{y}-\mathbf{X}\mathbf{w})$$

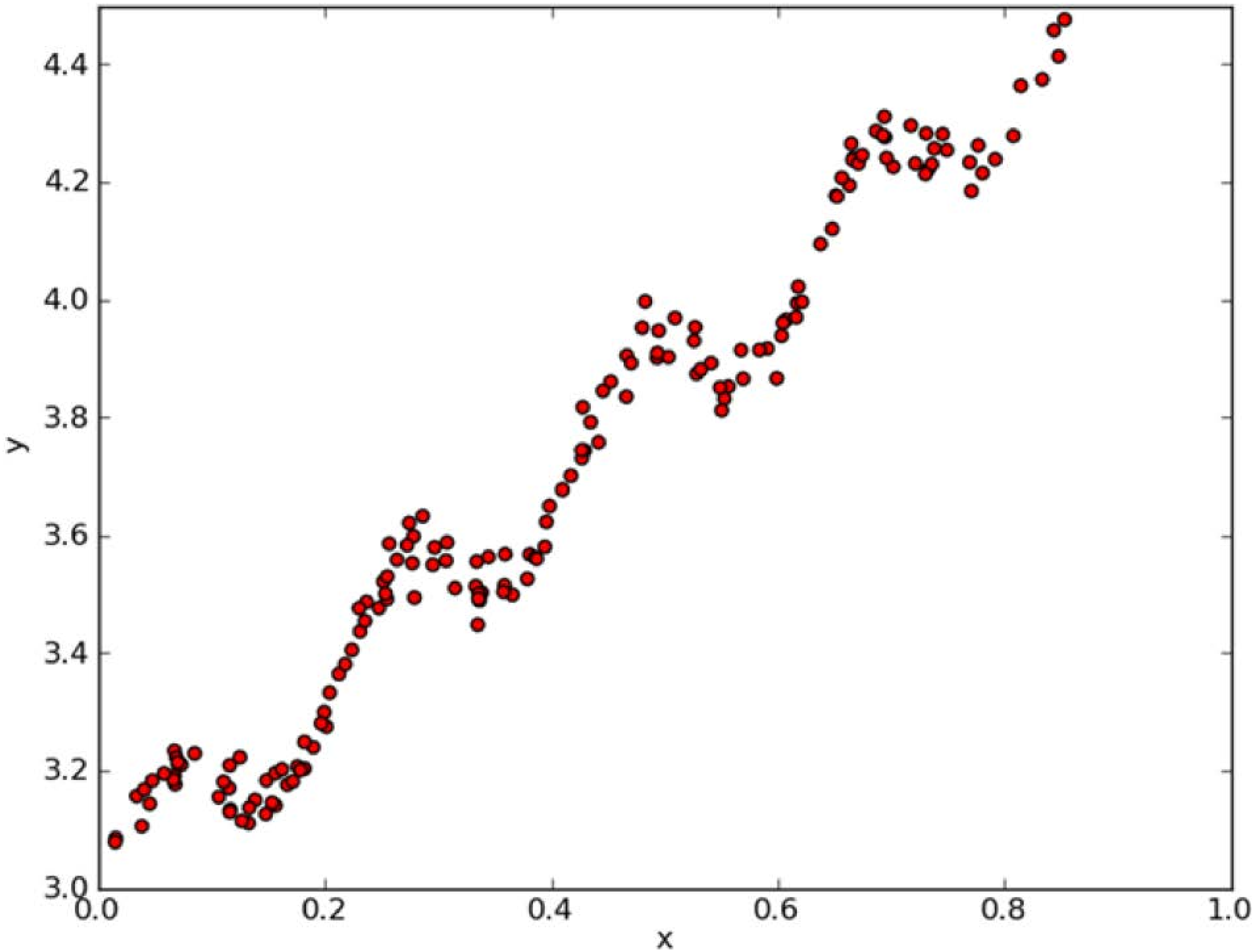
, in matrix notation $(\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w})$

- Solve by taking derivative: $\mathbf{X}^T(\mathbf{y}-\mathbf{X}\mathbf{w})$ and set to 0

  then $\hat{w} = (X^T X)^{-1} X^T y$

  (matrix inverse must exists!)

Example Data



Standard Regression Function

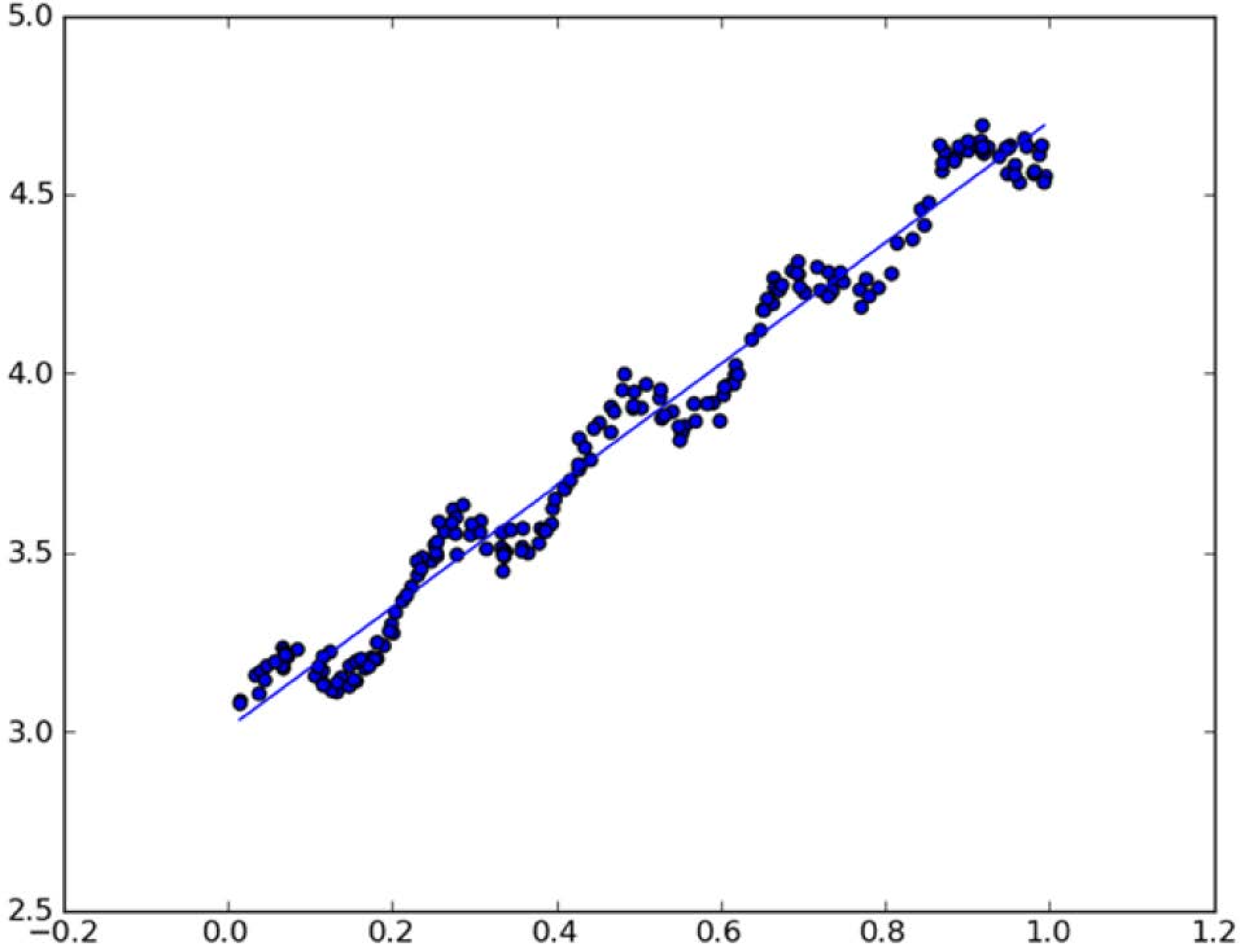
Listing 8.1 Standard regression function and data-importing functions

```
from numpy import *

def loadDataSet(fileName):
    numFeat = len(open(fileName).readline().split('\t')) - 1
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr =[]
        curLine = line.strip().split('\t')
        for i in range(numFeat):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))
    return dataMat,labelMat

def standRegres(xArr,yArr):
    xMat = mat(xArr); yMat = mat(yArr).T
    xTx = xMat.T*xMat
    if linalg.det(xTx) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    ws = xTx.I * (xMat.T*yMat)
    return ws
```

Best Fit Line of Regression



Locally Weighted Linear Regression, LWLR

- One problem with linear regression is that it tends to underfit the data
  - Lowest mean-squared error for unbiased estimators
- Locally weighted linear regression (LWLR)
  - Give a weight to data points near the data point of interest
  - Uses kernel like SVM to weight nearby points more heavily

$$\hat{w} = (X^T W X)^{-1} X^T W y$$ W is a matrix to weight data points

  - Gaussian kernel: $$w(i,i) = \exp\left(\frac{\left|x^{(i)} - x\right|}{-2k^2}\right)$$

Gaussian Kernel

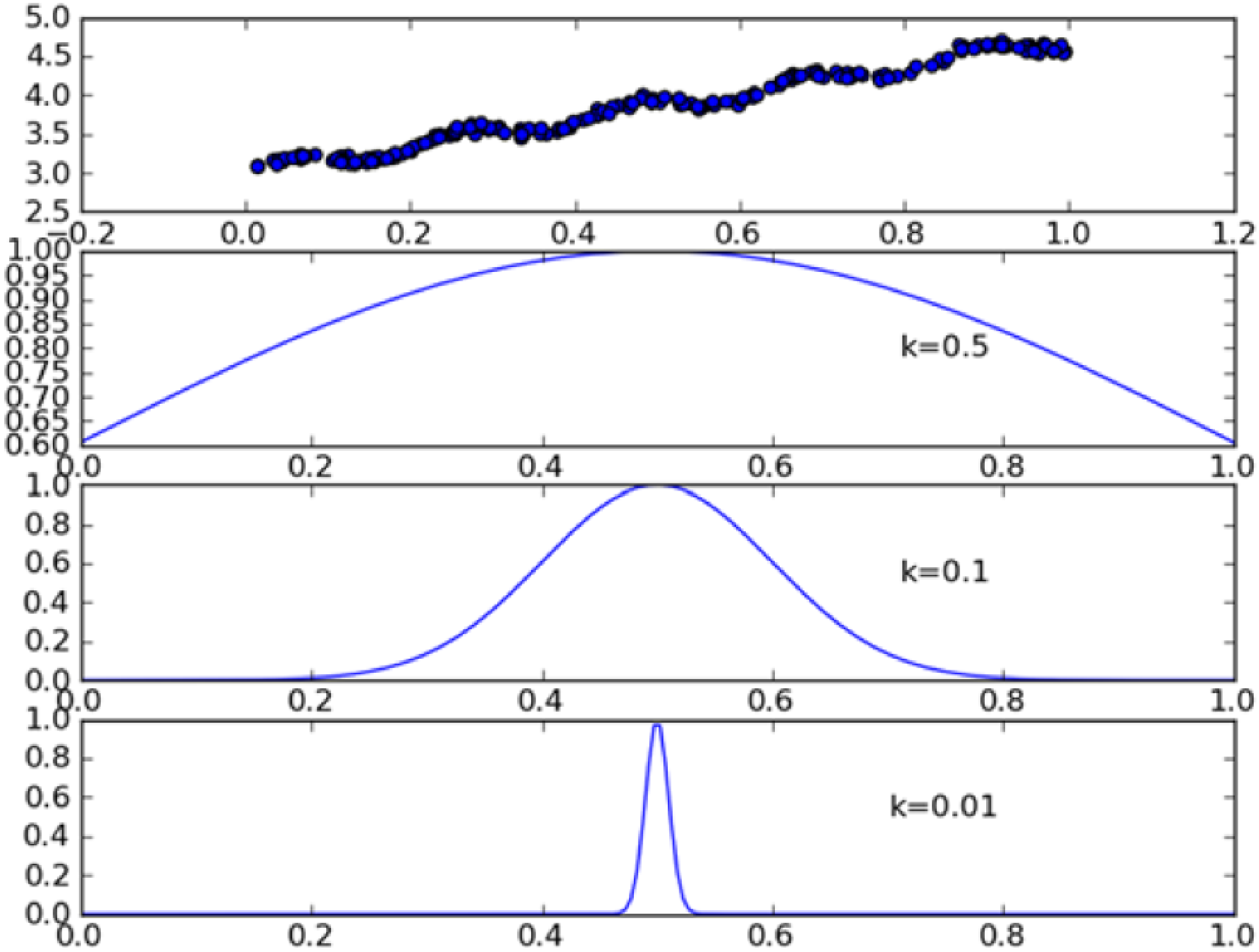


Figure 8.4 **Plot showing the original data in the top frame and the weights applied to each piece of data (if we were forecasting the value of `x=0.5`.) The second frame shows that with `k=0.5`, most of the data is included, whereas the bottom frame shows that if `k=0.01`, only a few local points will be included in the regression.**

LWLR Algorithm

Listing 8.2 Locally weighted linear regression function

```
def lwlr(testPoint,xArr,yArr,k=1.0):
    xMat = mat(xArr); yMat = mat(yArr).T
    m = shape(xMat)[0]
    weights = mat(eye((m)))
    for j in range(m):
        diffMat = testPoint - xMat[j,:]
        weights[j,j] = exp(diffMat*diffMat.T/(-2.0*k**2))
    xTx = xMat.T * (weights * xMat)
    if linalg.det(xTx) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    ws = xTx.I * (xMat.T * (weights * yMat))
    return testPoint * ws

def lwlrTest(testArr,xArr,yArr,k=1.0):
    m = shape(testArr)[0]
    yHat = zeros(m)
    for i in range(m):
        yHat[i] = lwlr(testArr[i],xArr,yArr,k)
    return yHat
```

1 Create diagonal matrix

2 Populate weights with exponentially decaying values

LWLR Smoothing

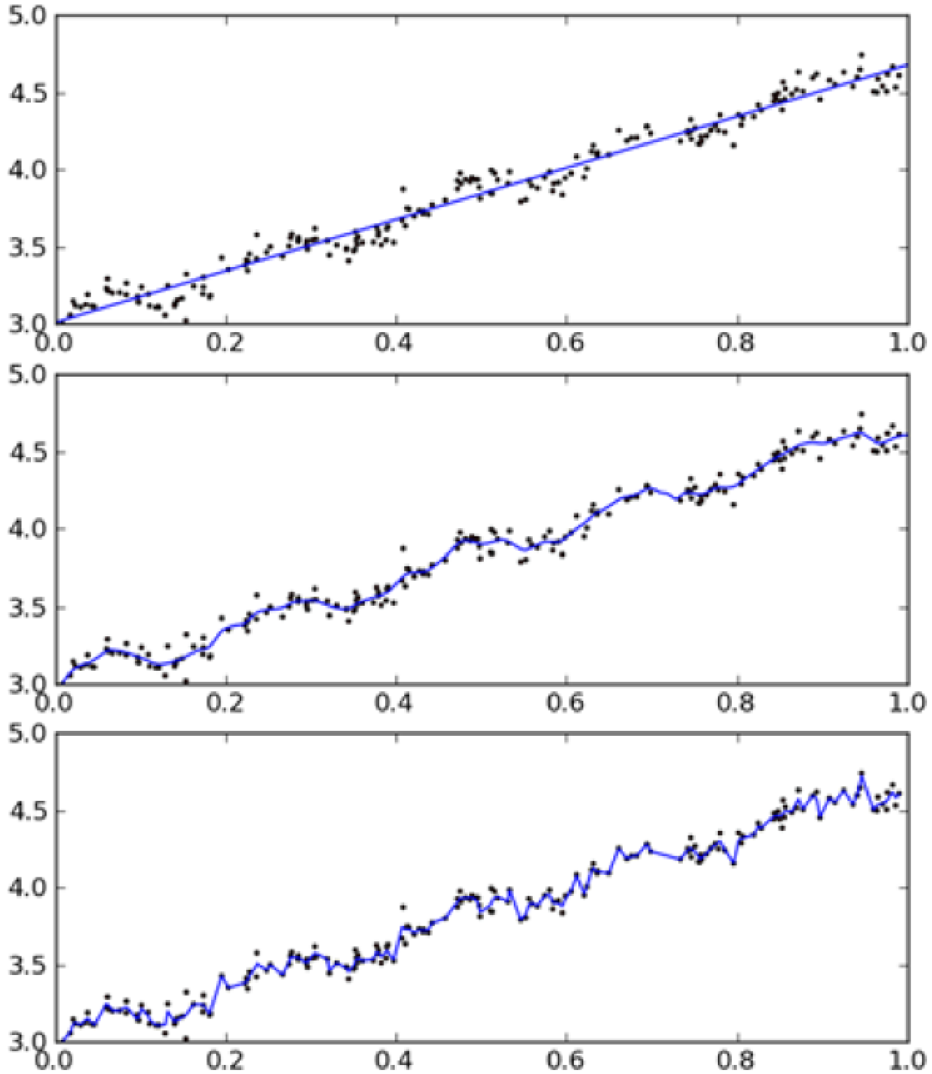


Figure 8.5 **Plot showing locally weighted linear regression with three smoothing values. The top frame has a smoothing value of `k=1.0`, the middle frame has `k=0.01`, and the bottom frame has `k=0.003`. The top value of k is no better than least squares. The middle value captures some of the underlying data pattern. The bottom frame fits the best-fit line to noise in the data and results in overfitting.**

When Linear Regression not Work

- More features than data points
  - m data points, n features, n>m
  - Not full rank, no inverse matrix
  - Solution: shrinkage methods
- Shrinkage method: ridge regression, lasso

Ridge Regression

- Add additional matrix λI to the matrix

$$\hat{w} = (X^T X + \lambda I)^{-1} X^T y$$

- Can also used to add bias into our estimations
- Constraints: $\sum_{k=1}^{n} w_k^2 \leq \lambda$

Ridge Regression Algorithms

Listing 8.3 Ridge regression

```
def ridgeRegres(xMat,yMat,lam=0.2):
    xTx = xMat.T*xMat
    denom = xTx + eye(shape(xMat)[1])*lam
    if linalg.det(denom) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    ws = denom.I * (xMat.T*yMat)
    return ws

def ridgeTest(xArr,yArr):
    xMat = mat(xArr); yMat=mat(yArr).T
    yMean = mean(yMat,0)
    yMat = yMat - yMean
    xMeans = mean(xMat,0)
    xVar = var(xMat,0)
    xMat = (xMat - xMeans)/xVar
    numTestPts = 30
    wMat = zeros((numTestPts,shape(xMat)[1]))
    for i in range(numTestPts):
        ws = ridgeRegres(xMat,yMat,exp(i-10))
        wMat[i,:]=ws.T
    return wMat
```

1 **Normalization code**

Regression Coefficient

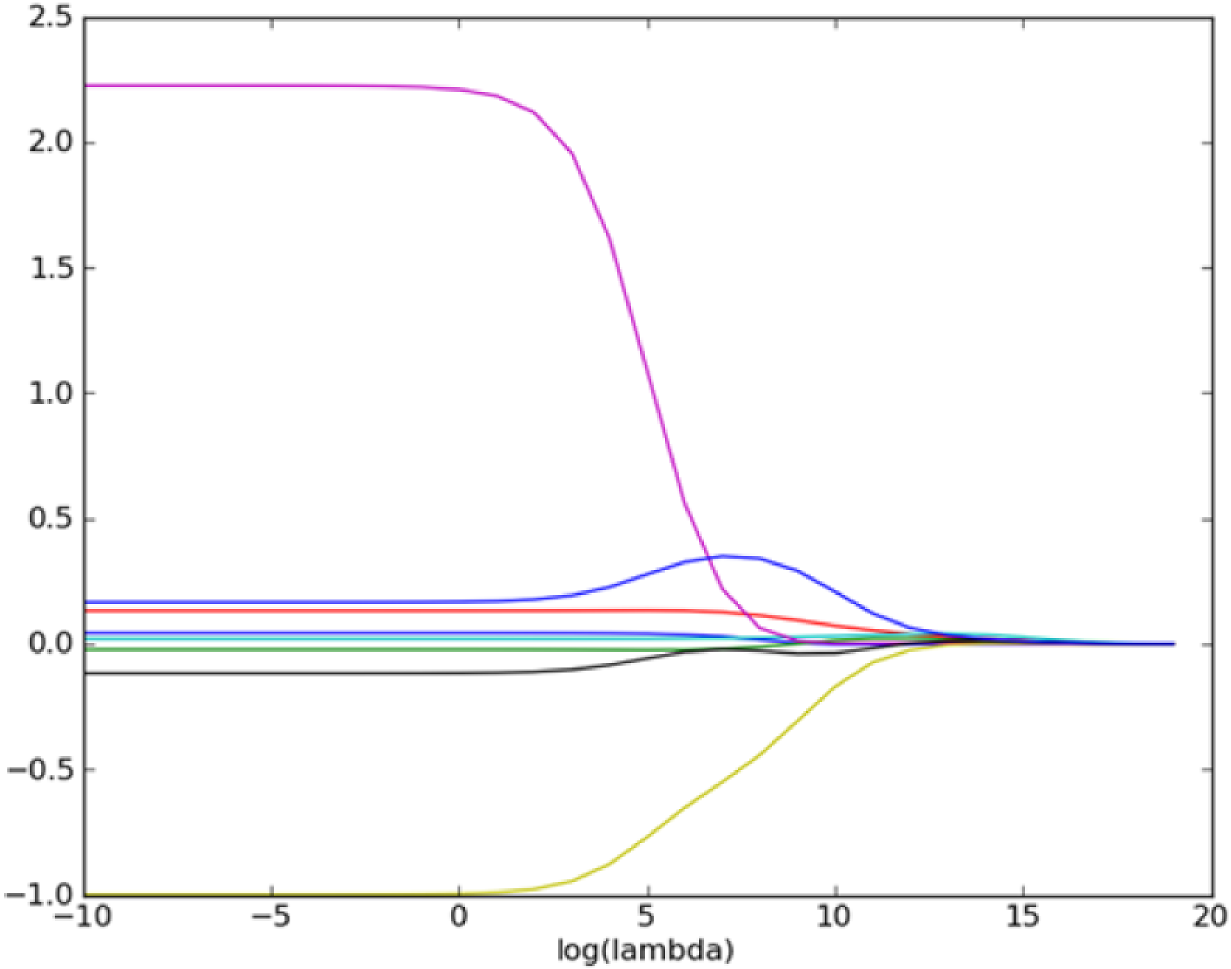


Figure 8.6 Regression coefficient values while using ridge regression. For very small values of λ the coefficients are the same as regular regression, whereas for very large values of λ the regression coefficients shrink to 0. Somewhere in between these two extremes, you can find values that allow you to make better predictions.

Lasso

- Similar to ridge regression except the constraints

$$\sum_{k=1}^{n} |w_k| \leq \lambda$$

Forward Stagewise Regression

- Easier algorithm than the lasso, gives close results
- A greedy algorithm
  - Each step it reduce the error the most at that step

Pseudo Algorithm

Regularize the data to have 0 mean and unit variance

For every iteration:

  Set lowestError to $+\infty$

  For every feature:

    For increasing and decreasing:

      Change one coefficient to get a new W

      Calculate the Error with new W

      If the Error is lower than lowestError: set Wbest to the current W

    Update set W to Wbest

Forward Stagewise Regression Algorithm

Listing 8.4 Forward stagewise linear regression

```
def stageWise(xArr,yArr,eps=0.01,numIt=100):
    xMat = mat(xArr); yMat=mat(yArr).T
    yMean = mean(yMat,0)
    yMat = yMat - yMean
    xMat = regularize(xMat)
    m,n=shape(xMat)
    ws = zeros((n,1)); wsTest = ws.copy(); wsMax = ws.copy()
    for i in range(numIt):
        print ws.T
        lowestError = inf;
        for j in range(n):
            for sign in [-1,1]:
                wsTest = ws.copy()
                wsTest[j] += eps*sign
                yTest = xMat*wsTest
                rssE = rssError(yMat.A,yTest.A)
                if rssE < lowestError:
                    lowestError = rssE
                    wsMax = wsTest
        ws = wsMax.copy()
        returnMat[i,:]=ws.T
    return returnMat
```

Regression Coefficient

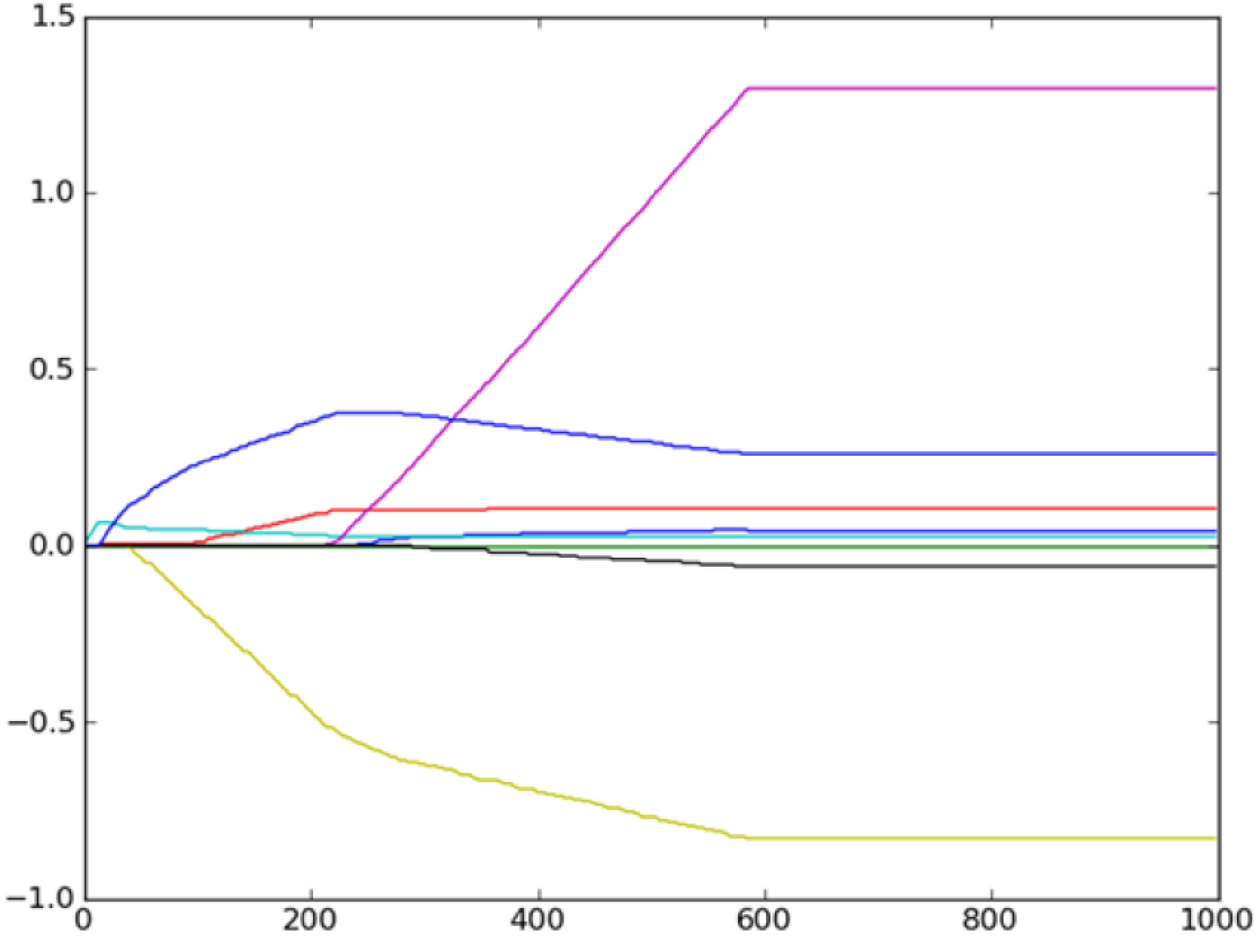


Figure 8.7 **Coefficient values from the abalone dataset versus iteration of the stagewise linear regression algorithm. Stagewise linear regression gives values close to the lasso values with a much simpler algorithm.**

Bias/Variance Tradeoff

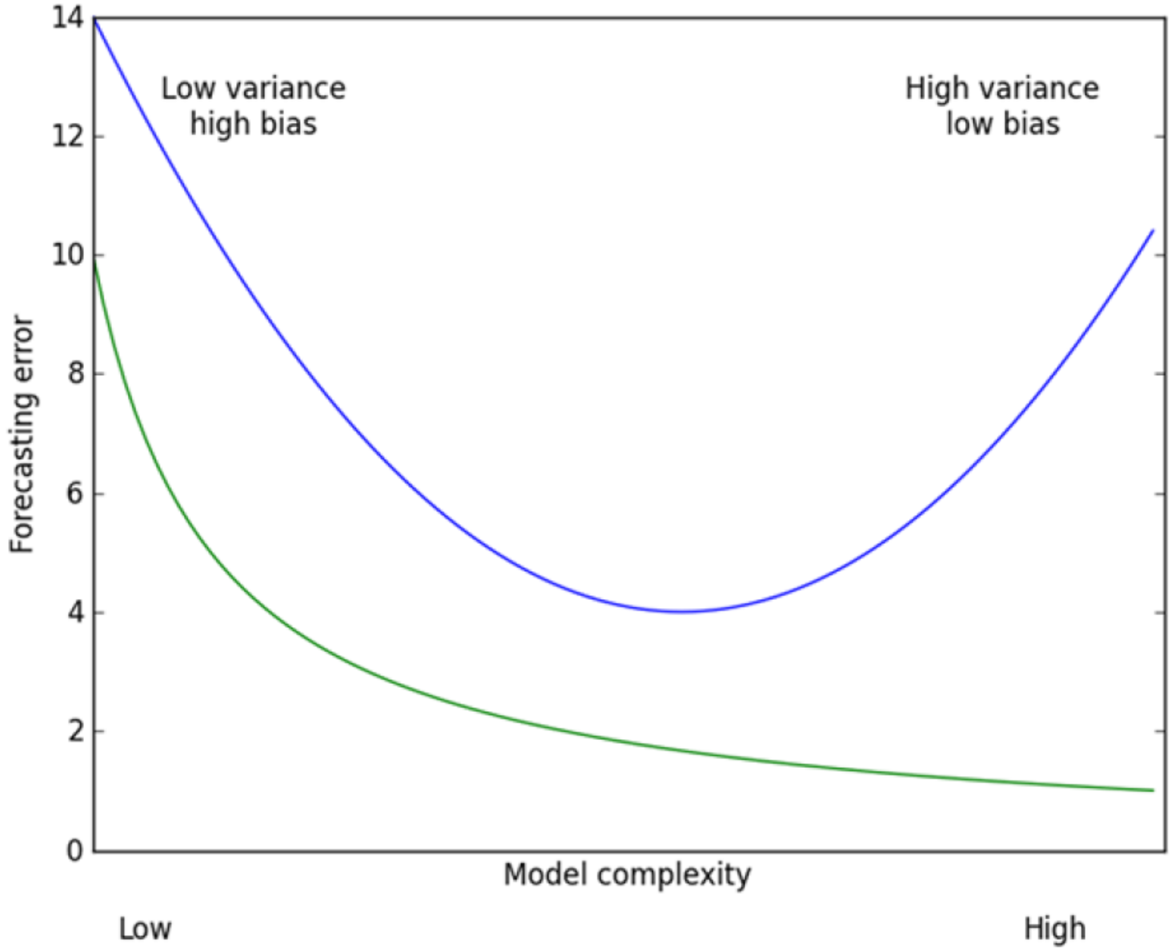


Figure 8.8 **The bias variance tradeoff illustrated with test error and training error. The training error is the top curve, which has a minimum in the middle of the plot. In order to create the best forecasts, we should adjust our model complexity where the test error is at a minimum.**

Cross-Validation Test

Listing 8.6 Cross-validation testing with ridge regression

```
def crossValidation(xArr,yArr,numVal=10):
    m = len(yArr)
    indexList = range(m)
    errorMat = zeros((numVal,30))
    for i in range(numVal):
        trainX=[]; trainY=[]
        testX = []; testY = []
        random.shuffle(indexList)
        for j in range(m):
            if j < m*0.9:
                trainX.append(xArr[indexList[j]])
                trainY.append(yArr[indexList[j]])
            else:
                testX.append(xArr[indexList[j]])
                testY.append(yArr[indexList[j]])
        wMat = ridgeTest(trainX,trainY)
        for k in range(30):
            matTestX = mat(testX); matTrainX=mat(trainX)
            meanTrain = mean(matTrainX,0)
            varTrain = var(matTrainX,0)
            matTestX = (matTestX-meanTrain)/varTrain
            yEst = matTestX * mat(wMat[k,:]).T + mean(trainY)
            errorMat[i,k]=rssError(yEst.T.A,array(testY))
    meanErrors = mean(errorMat,0)
    minMean = float(min(meanErrors))
    bestWeights = wMat[nonzero(meanErrors==minMean)]
    xMat = mat(xArr); yMat=mat(yArr).T
    meanX = mean(xMat,0); varX = var(xMat,0)
    unReg = bestWeights/varX
    print "the best model from Ridge Regression is:\n",unReg
    print "with constant term: ",\
          -1*sum(multiply(meanX,unReg)) + mean(yMat)
```

1. **Create training and test containers**
2. **Split data into test and training sets**
3. **Regularize test with training params**
4. **Undo regularization**

Summary

- Regression is the process of predicting a target value similar to classification
  - Ridge regression is an example of a shrinkage method
- Another shrinkage method that's powerful is the lasso
- The lasso is difficult to compute, but stagewise linear regression is easy to compute and gives results close to those of the lasso
- Shrinkage methods can also be viewed as adding bias to a model and reducing the variance