Part 5: Multivariate Linear Regression

Mark Bebbington
AgHortC 2.09A
m.bebbington@massey.ac.nz

Reading for this Part

There is very little in the text on this part (Section 2.4).

Copies of relevant material from Hastie, T., Tibshirani, R., Friedman, J. "The Elements of Statistical Learning" can be found on the webpage.

Recall `Learning'

- Supervised learning
  - Algorithms learn by example
    - Datasets have input(s) and associated answer(s) [outputs or targets]
- Notation
  - Inputs (vectors): $x_i = (x_{ji}, j = 1,\ldots,p)$, i=1,…,n
  - Outputs : y_i, i = 1,…,n
  - Targets : t_i, i = 1,…,n
- Variable Types
  - Quantitative – continuous
    - Regression
  - Qualitative – categorical (discrete)
    - Classification
  - Ordinal (ordered categorical)

Nearest Neighbour Methods

Output of an input should be similar to outputs from other inputs `close' in input space

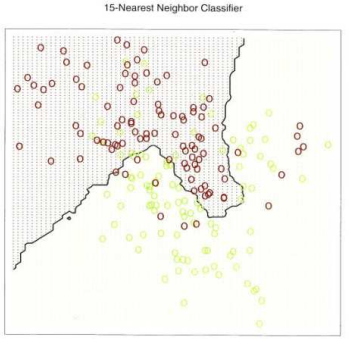


FIGURE 2.2. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (*GREEN = 0, RED = 1*) and then fit by 15-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the 15-nearest neighbors.*

Linear Methods

Fit straight `line'

(hyperplane) through data

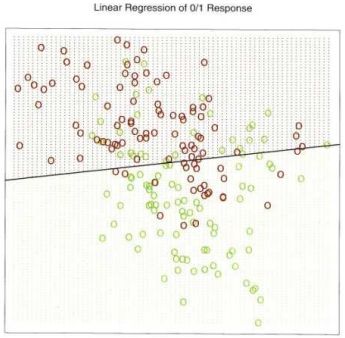


FIGURE 2.1. *A classification example in two dimensions. The classes are coded as a binary variable—GREEN = 0, RED = 1—and then fit by linear regression. The line is the decision boundary defined by $x^T\hat{\beta} = 0.5$. The red shaded region denotes that part of input space classified as RED, while the green region is classified as GREEN.*

Linear Model vs. Nearest Neighbour

Method	Structural Assumptions	Accuracy	Stability
Linear Model	Huge	Possibly inaccurate	Stable
Nearest Neighbour	Mild	Often accurate	Can be unstable

Linear Models and Least Squares

In univariate linear regression, a scalar output y was predicted by a scalar input x, and we had N pairs of input-output data.

Now suppose that a scalar output y is predicted by a vector (with p elements) input $x = (x_1\ x_2\ \ldots\ x_{p)}$.

Thus we predict the output y via

$$\hat{y} = \beta_0 + \sum_{j=1}^{p} x_j\beta_j$$

Linear Models and Least Squares

which can be written as $y = x^T\beta$, where $x^T = (1\ x_1\ x_2\ \ldots\ x_p)$, and $\beta = (\beta_0\ \beta_1\ \beta_2\ \ldots\ \beta_p)^T$.

We choose β to minimize the residual sum of squares

$$RSS(\beta) = \sum_{i=1}^{N}(y_i - x_i^T\beta)^2$$
$$= (y - X\beta)^T(y - X\beta)$$

where X is an N x (p+1) matrix with each row an input vector (augmented by a leading 1), β is a (p+1) x 1 vector, and y an N x 1 vector

Solve for β

Differentiating wrt β, we get the *normal equations*

$$X^T(y - X\beta) = 0,$$

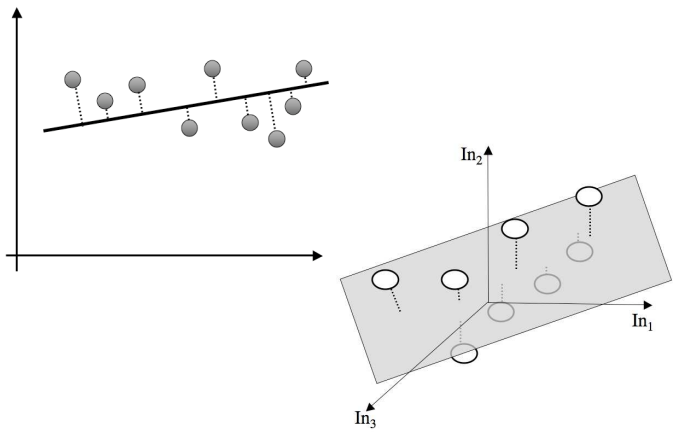
which, if X^TX is nonsingular, has the unique solution

$$\hat{\beta} = (X^TX)^{-1}X^Ty.$$

The fitted value at some point x_0 is

$$\hat{y}(x_0) = x_0^T\hat{\beta}$$

As a function in input space, $f(x) = x^T\beta$ is linear, and $f'(x) = \beta$ is a vector that points in the steepest uphill direction



Linear regression in 2 and 3 dimensions

Linear Regression in NumPy

```
def linregp(X,y):
    # takes N x p matrix X and a N x 1 vector y, and fits
    # linear regression y = X*beta. NOTE indenting
    X1 = concatenate((ones((shape(X)[0],1)),X),axis=1)
    betahat = linalg.inv(transpose(X1)*X1)*transpose(X1)*y
    yhat = X1*betahat
    RSS = transpose(y – yhat)*(y – yhat)
    return betahat,yhat,RSS

temp = linregp(X,y)
betahat = temp[0]
yhat = temp[1]
RSS = temp[2]
```

So 'linregp(X,y)' returns 3 arguments – a (p+1) x 1 vector betahat, an Nx1 vector yhat, and a scalar RSS

Sampling properties of $\hat{\beta}$

Assume that the y_i are uncorrelated, with variance σ^2, and the x_i are fixed (non-random).

The covariance matrix is then

$$\text{Var}(\hat{\beta}) = (X^TX)^{-1}\sigma^2$$

σ^2 is estimated by

$$\hat{\sigma}^2 = \frac{1}{n-p-1}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Inference

➢We now assume that the linear model is the correct one ☺, and that the errors are additive and Gaussian.

$$y = \beta_0 + \sum_{j=1}^{p} x_j \beta_j + \varepsilon$$

where $\varepsilon \sim N(0,\sigma^2)$. Then

$$\hat{\beta} \sim N(\beta,(X^TX)^{-1}\sigma^2)$$

which is a multivariate normal distribution with mean β (the true parameters). Also, $\hat{\beta}$ and $\hat{\sigma}^2$ are independent, and $(n - p - 1)\,\hat{\sigma}^2 \sim \sigma^2\chi^2_{n-p-1}$

Hypothesis Test: 1 Coefficient

➢To test H_0: $\beta_j = 0$, we have the Z-score (i.e., ~N(0,1))

$$z_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{v_j}}$$

where v_j is the jth diagonal element of $(X^TX)^{-1}$, and $z_j \sim t_{n-p-1}$

Thus $|z_j| > 2$ indicates a significant effect.

In NumPy

```
X1 = concatenate((ones((shape(X)[0],1)),X),axis=1)
yhat = linregp(X,y)[1]
sigmahat2 = (1.0/(len(y) - len(beta)))*sum((yhat - y)**2)
varbeta = linalg.inv(transpose(X1)*X1)*sigmahat2
z = beta/sqrt(diag(varbeta))
```

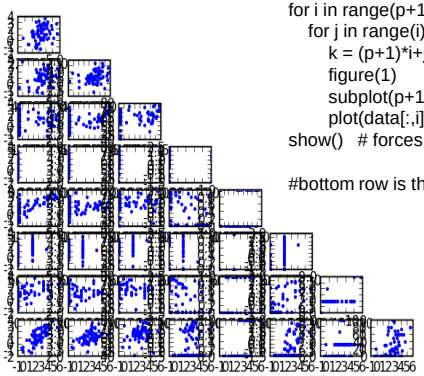

Example: Linear regression

- Using the prostate cancer data 'prostate_train.txt' on the website.
- Predictor variables are, in order,
  - log cancer volume, log prostate weight, age, lbph, svi, lcp, Gleason score, pgg45
  - (the full terms are very long, and incomprehensible unless you are a medico).
  - The last column is the response lpsa.

In order to get the data, we use

```
data = loadtxt('prostate_train.txt')   # this reads it in as
   a list of numbers (1-D array)
p = 8
data = data.reshape(-1,p+1)  # and this reshapes it into a
   matrix
X = data[:,0:p]   # the predictors are the first 8 columns
y = data[:,p]     # the response is the last column
```

Having a look at the data



```
from pylab import *
for i in range(p+1):
    for j in range(i):
        k = (p+1)*i+j+1
        figure(1)
        subplot(p+1,p+1,k)
        plot(data[:,i],data[:,j],'.')
show()   # forces pylab to print current file

#bottom row is the response variable
```


Correlation matrix of the predictor variables

```
covX = cov(transpose(X)) # need to transpose X to get a p x p covariance matrix
sdX = sqrt(diag(covX)) # Note that sdX is an array
print 'sdX = ',sdX

sdX =  [  1.24259052   0.47660045   7.50220833   1.46365294   0.41998902
   1.40073378   0.70886357  29.30176439]

corrX = covX/dot(transpose(mat(sdX)),mat(sdX))
print 'corrX = ',corrX

corrX =
matrix([[ 1.        ,  0.3002,  0.2863,  0.0631,  0.5929, 0.6920,  0.4264,  0.4831],
        [ 0.3002,  1.    ,  0.3167,  0.4370,  0.1810, 0.1568,  0.0235,  0.0741],
        [ 0.2863,  0.3167,  1.    ,  0.2873,  0.1289, 0.1729,  0.3659,  0.2758],
        [ 0.0631,  0.4370,  0.2873,  1.    , -0.1391, -0.0885, 0.0329, -0.0304],
        [ 0.5929,  0.1810,  0.1289, -0.1391,  1.    , 0.6712,  0.3068,  0.4813],
        [ 0.6920,  0.1568,  0.1729, -0.0885,  0.6712,  1.    ,  0.4764, 0.6625],
        [ 0.4264,  0.0235,  0.3659,  0.0329,  0.3068,  0.4764,  1.    ,  0.7570],
        [ 0.4831,  0.0741,  0.2758, -0.0304,  0.4813, 0.6625,  0.7570,  1.     ]])
```


Debugging – why doesn't my program work? (Ex. 5.1)

- A program to perform a linear regression on the prostate cancer data can be found in 'prostate_linreg_0.py'.
- Can you get it to work?
- Hint: In order to find out what is going wrong, you need to insert print statements (or occasionally plots) to show you what the variables going into the state look like.
  - Sometime you need `print X`, but much of the time `print X.shape` will suffice.

Output

```
sigmahat2 =  0.507351447557
beta, z =  [[ 0.42915914  0.27623733]
 [ 0.57654251  5.3662843 ]
 [ 0.61402307  2.75080324]
 [-0.0190009  -1.395901  ]
 [ 0.1448479   2.05584031]
 [ 0.73720852  2.46925469]
 [-0.20632451 -1.86691279]
 [-0.02950392 -0.14668635]
 [ 0.00946517  1.73784091]]
```

From which we see that the 1st, 2nd, 4th and 5th predictors appear to be significant (|z| > 2)

Debugged Program

The errors were

1) Inconsistent use of matrices and arrays. I choose to fix this by working with matrices. You may prefer arrays.
   a) X = matrix(X), y = transpose(matrix(y)), etc.
   b) This will make sigmahat2 a matrix – recast to float()
2) Use of beta rather than betahat lower in program
3) Note that z is still calculated even if the matrices are of different shapes, but the answer is non-sensical!!

- Program in 'prostate_linreg1.py'

How many predictors?

A good fit to the data is nice, or is it?

- We can get a perfect fit by fitting an N parameter model
  - What is wrong with this?
    - How well does the model work on data it is not fitted to?

Model Validation requires checking the model (fitted to the 'training' data) against independent (test) data to see how well it predicts.

Measuring Performance

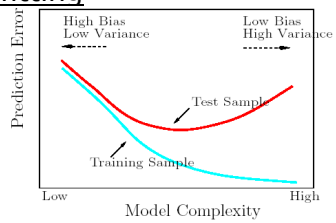
- Target variable, Y
- Vector of inputs, X
- Prediction model, $\hat{f}(X)$

- Typical choices of *Loss function:*

$$L\left(Y,\hat{f}(X)\right)=\begin{cases}\left(Y-\hat{f}(X)\right)^2 & \text{squared error}\\ \left|Y-\hat{f}(X)\right| & \text{absolute error}\end{cases}$$

Bias-Variance trade-off

- Bias = lack of accuracy
  - Decreases with more predictors
- Variance = lack of precision
  - Increases with more predictors
- Training error - Overfitting
  - consistently decreases with model complexity (no. of predictors)



161. Figure 7.1: *Behavior of test sample and training sample error as the model complexity is varied.*

How many predictors?

- Answer – remove all predictors with |z| < 2
  - What is wrong with this?
    - If predictor 7, say, is removed, how is the z-value of predictor 8 affected?
      - Need to test removal of single parameters / groups of parameters
    - What are the scales of the variables – are values for predictor 1 consistently in the hundreds, and predictor 2 less than 1. How does this affect things?
      - Standardization

Standardization

Standardize the predictors to have unit variance by dividing by the standard deviation.

```
for i in range(p):    #standardize data
  X[:,i] = X[:,i]/sdX[i]
```

We can then do the regression

```
sigmahat2 =  0.507351447557
beta, z =  [[ 0.42915914  0.27623733]
 [ 0.71640625  5.3662843 ]
 [ 0.29264367  2.75080324]
 [-0.14254874 -1.395901  ]
 [ 0.21200705  2.05584031]
 [ 0.30961948  2.46925469]
 [-0.28900571 -1.86691279]
 [-0.02091425 -0.14668635]
 [ 0.27734618  1.73784091]]
```

➢Program in 'prostate_linreg2.py'

Nested Models

A model M1 is nested in a model M2 if all of the predictors in M1 are included in M2

A			
A		C	
A		C	D
A	B	C	D

Nested Models

A	B		
	B	C	D
A		C	D

These models cannot be nested

Hypothesis test: Groups of Coefficients

➢Suppose we have two nested models with p_0 and $p_1 > p_0$ parameters. To test if the `last' $p_1 - p_0$ parameters can be set to 0, we use the F-statistic:

$$F = \frac{(RSS_0 - RSS_1)(p_1 - p_0)}{RSS_1/(n - p_1 - 1)}$$

where RSS_0 is for the smaller model etc. F measures the change in RSS per additional parameter in the bigger model. Under our assumptions, and if the null H_0: smaller model is correct,

$$F \sim F_{p_1 - p_0, n - p_1 - 2}$$

Regression without intercept

Recall that only predictors 1,2,4 and 5 (intercept is predictor '0') appear to be significant. We will test this.

As the intercept is not significant, we need a new function for linear regression without an intercept:

```
def linregp_ni(X,y):
   # takes N x p matrix X and a N x 1 vector y, and fits
   # linear regression y = X*beta with zero intercept.
   betahat = linalg.inv(transpose(X)*X)*transpose(X)*y
   yhat = X*betahat
   RSS = transpose(y – yhat)*(y – yhat)
   return betahat,yhat,RSS
```


Test reduced model

```
# specify 2 models
X1 = concatenate((ones((shape(X)[0],1)),X),axis=1)
                                        # augment with intercept term
X0 = concatenate((X[:,0:2],X[:,3:5]),axis=1)
                                # take only 1st, 2nd, 4th and 5th predictors

# do regression(s)
temp1 = linregp_ni(X1,y)
temp0 = linregp_ni(X0,y)
betahat1 = temp1[0]
RSS1 = temp1[2]
betahat0 = temp0[0]
yhat0 = temp0[1]
RSS0 = temp0[2]

# check z-scores for smaller model
sigmahat2 = float((1.0/(len(y) - len(betahat0)))*transpose(yhat0 -
y)*(yhat0 - y))
varbeta = linalg.inv(transpose(X0)*X0)*sigmahat2
z = multiply(betahat0,1/transpose(matrix(sqrt(diag(varbeta)))))
print 'beta, z = ',concatenate((betahat0,z),axis=1)

beta, z =  [[  0.63178909   5.54516572]
 [  0.21346882  12.22553765]
 [  0.22427706   2.45206077]
 [  0.28826277   2.5493217 ]]
```


Inference

```
# check z-scores for smaller model
sigmahat2 = float((1.0/(len(y) - len(betahat0)))*transpose(yhat0 -
y)*(yhat0 - y))
varbeta = linalg.inv(transpose(X0)*X0)*sigmahat2
z = multiply(betahat0,1/transpose(matrix(sqrt(diag(varbeta)))))
print 'beta, z = ',concatenate((betahat0,z),axis=1)

beta, z =  [[  0.63178909   5.54516572]
 [  0.21346882  12.22553765]
 [  0.22427706   2.45206077]
 [  0.28826277   2.5493217 ]]

F = float(((RSS0 - RSS1)/(len(betahat1) - len(betahat0)))/(RSS1/(len(y) -
len(betahat1) - 2)))
print 'F = ',F

F =  1.32491515642
```

Which has a P-value of 0.2664 (Prob($F_{5,58}$ > 1.325) = 0.2664), and is not significant. Thus the larger model is not significantly better (for the extra parameters)

➢Program in 'prostate_linreg3.py'

Confidence Intervals

A 1-2α CI for β_j is

$$\hat{\beta}_j \pm z_{1-\alpha}\sqrt{v_j}\hat{\sigma}$$

while an approximate 1-2α confidence set for the entire parameter β is

$$C_\beta = \left\{\beta : (\hat{\beta}-\beta)^T X^T X(\hat{\beta}-\beta) \le \hat{\sigma}^2 \chi^2_{p+1,1-\alpha}\right\}$$

which generates a confidence interval for the function $f(x) = x^T\beta$ of $\{x^T\beta : \beta \in C_\beta\}$

Exercise 5.2 (homework)

Write a program to calculate the confidence intervals for $\beta_1, \beta_2, \beta_4, \beta_5$ in the subset model, and output the result graphically.

Hint: An easy way to do this is simulate random points, retaining them only if they are in the confidence set.

Subset selection and coefficient shrinkage

Two problems with least squares estimates:

- Prediction accuracy
  - low bias, but large variance. Improve accuracy by shrinking or setting some coefficients to zero, sacrificing a little bias for reduced variance, improving overall accuracy
- Interpretation
  - a large number of predictors may contain a smaller subset with strongest effects, getting the `big picture' by ignoring some of the small details

Exercise 5.3

Apply the reduced regression model for the prostate cancer data to the test dataset (use the parameters calculated from the training data), and consider the result. Compare with the result when the model is fitted to the test data.

Subset selection

Retain only a subset of variables, eliminating the rest from the model. Two approaches:

- Best subset regression
  - find for each $k \in \{0,1,2,\ldots,p\}$ the subset of size k that gives the smallest RSS. Note that the subsets need not be nested.
  - As RSS is decreasing with k, it cannot be used to select k. Instead choice of k involves a tradeoff between bias and variance.
  - Chose the model that minimizes an estimate of the expected prediction error.

Forward Stepwise Selection

- Rather than search all possible subsets, seek a good `path'
  - Start with the intercept, sequentially adding the predictor that most improves the fit

If the current model has k inputs (estimates $\hat{\beta}$, say) and we want to add another (estimates $\tilde{\beta}$), then the improvement in fit is the F-statistic

$$F = \frac{RSS(\hat{\beta}) - RSS(\tilde{\beta})}{RSS(\tilde{\beta})/(n-k-2)}$$

- continue adding in parameters until no predictor achieves an F-ratio greater than the 90th or 95th percentile of the $F_{1,n-k-2}$ distribution.

Stepwise Regression in Python

```
def fstepreg(X,y,p):
   oldbeta = mean(y, axis=0)  #start with intercept
   oldX1 = ones(shape(y))  #intercept term
   oldRSS = transpose(y - oldX1*oldbeta)*(y - oldX1*oldbeta)
   inclpred = [-1]    # LIST of included predictors, start with -1, the intercept
   exclpred = list(arange(p)) # LIST of predictors not included
   continu = 1
   while continu:
      bestRSS = 100*oldRSS  # some large number
      for i in range(len(exclpred)): # try adding each predictor to current model
          j = exclpred[i]
          X1 = concatenate((oldX1,X[:,j:j+1]),axis=1)  #add jth col. of X to input matrix
          beta = linregp_ni(X1,y)[0]  #fit linear regression model
          RSS = transpose(y - X1*beta)*(y - X1*beta)
          if RSS < bestRSS:  # best additional predictor so far
               bestRSS = RSS   # --record the details
               bestpred = j
               bestbeta = beta
      F = (oldRSS-bestRSS)/(bestRSS/(len(y)-len(bestbeta)-1)) #F-ratio for best add. predictor
      if F > F90[len(y) - len(bestbeta) - 1,1]:  #significant? --update model
         oldX1 = concatenate((oldX1,X[:,bestpred:bestpred+1]),axis=1)
         oldbeta = bestbeta
         oldRSS = bestRSS
         inclpred.append(bestpred)
         exclpred.remove(bestpred)
      else:  # best add. predictor not significant - finished
         continu = 0
      if len(exclpred) == 0:  # stop if all predictors included
         continu = 0
   return oldbeta,oldRSS,inclpred,exclpred
```


Prostate data

```
F90 = loadtxt('F90.txt') # get alpha = 0.9 values
                         # of F distribution
F90 = F90.reshape(-1,10)

temp = fstepreg(X,y,p)
betahat = temp[0]
RSS = temp[1]
inclpred = temp[2]
exclpred = temp[3]

print 'included predictors = ',inclpred
print 'excluded predictors = ',exclpred
print 'betahat = ',betahat
print 'RSS = ',float(RSS)
```

```
included predictors =  [-1, 0, 1, 4, 3]
excluded predictors =  [2, 5, 6, 7]
betahat =  [[-0.32593568]
 [ 0.62815437]
 [ 0.25680825]
 [ 0.28216895]
 [ 0.20492731]]
RSS =  32.8149841999
```

- Program in 'prostate_fwdsubreg.py'

Selected Model

Intercept, and the 1st,2nd,5th and 4th predictors (lcavol, weight, svi and lbph)

The predictors are those used in the subset example, but we have the intercept as we started with it.

Exercise 5.4

Modify 'prostate_fwdsubreg.py' so that it doesn't necessarily start with the intercept.

Note: the intercept can't be treated the same way as the other predictors, as the X^TX matrix will be singular if it is the only predictor in X

F-ratio stopping rule

- Provides only local control of search
- Does not attempt to find the `best' model along the sequence examined
  - We can chose the model that minimises an estimate of expected prediction error.

Backward Stepwise Selection

- Starts with the full model, dropping the predictor that gives the smallest value of F at each step, stopping when every predictor gives an F-value above the threshold when dropped.
- Requires n > p, while forward stepwise can always be used
- Hybrid strategies consider both forward and backward steps, making the `best' move at each stage.
  - Needs a parameter to set the threshold for add/drop moves.

Exercise 5.5

Implement backward stepwise regression on the prostate cancer data, starting with the full model.

Shrinkage Methods

- Subset selection produces a more interpretable model, with possibly lower prediction error
- But it is discrete – variables are either retained or discarded – and thus often has high variance
- Shrinkage methods are more continuous:
  - Ridge regression
  - The Lasso

Ridge Regression

- Shrinks regression coefficients by imposing a penalty on their size
- Minimises a penalized residual sum of squares

$$\hat{\beta}^{ridge} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

where $\lambda \geq 0$ is a complexity parameter; the larger, the greater the amount of shrinkage.

- The parameters are shrunk towards zero, preventing huge offsetting positive and negative coefficients.

Ridge regression: solution

Writing the penalized residual sum of squares in matrix form:

$$RSS(\lambda) = (y-X\beta)^T(y-X\beta) + \lambda\beta^T\beta$$

which has solution

$$\hat{\beta}^{ridge} = (X^TX + \lambda I)^{-1}X^Ty$$

Where I is the pxp identity matrix.

- Note that with the quadratic penalty, the solution is again a linear function of y.
- Solution adds a +ve constant to the diagonal of X^TX, making it nonsingular, even if X^TX is not of full rank.
- In the case of orthogonal inputs, the ridge estimates are a scaled version of the least squares estimates

Ridge regression, NB

- Ridge solutions are not invariant under scaling of inputs (try this yourself)
  - Standardize inputs beforehand
- The intercept needs to be left out of the penalty term. Penalizing this would make the procedure dependent on the origin chosen for Y (ie, adding a constant would not simply shift the predictions by the same amount)

Reparameterization using centered inputs

- The solution to the penalized residual sum of squares can be separated into two parts, by replacing x_{ij} by $x_{ij} - \bar{x}_{j\bullet}$, where the last term is the average with j fixed.
  - We then estimate β_0 by $\bar{y} = (1/n)\sum_{i=1}^{n} y_i$
  - The remaining coefficients can then be estimated by ridge regression without intercept, using the centered x_{ij}
- We shall assume this centering has been done, so the input matrix X has p, not p+1, columns

Ridge regression in NumPy

```
def ridgereg(X,y,lam):
# takes N x p matrix X (no intercept column) and a N x 1 vector y, and
# fits ridge regression y = X*beta with lam*transpose(beta)*beta
# penalty term
    betahat = linalg.inv(transpose(X)*X +
   lam*eye(X.shape[1]))*transpose(X)*y
    # add intercept term
    betahat0 = mean(y, axis=0)
    betahat = concatenate((betahat0,betahat),axis=0)
    X1 = concatenate((ones((shape(X)[0],1)),X),axis=1)
    # calculate fitted yhat
     yhat = X1*betahat
    RSS = transpose(y-yhat)*(y-yhat) # RSS not penalized
    return betahat,yhat,RSS
```

- Center the inputs

```
meanX = mean(X, axis=0)
Xc = X - ones(shape(y))*meanX
```


Prostate Example

```
lam = 32   ## SEE LATER
temp = ridgereg(Xc,y,lam)
betahat = temp[0]
yhat = temp[1]
RSS = temp[2]
print 'betahat = ',betahat
print 'RSS = ',RSS

betahat =  [[ 2.45234522]
 [ 0.3893775 ]
 [ 0.23853869]
 [-0.02881583]
 [ 0.1568548 ]
 [ 0.22037858]
 [ 0.03402117]
 [ 0.04694587]
 [ 0.12384315]]
RSS =  35.0799374476
```

- Program in 'prostate_ridgereg.py'

Exercise 5.6

For ridge regression, using the prostate cancer data as necessary, show that

- the solution is not invariant under scaling of inputs.
- penalizing the intercept makes the procedure dependent on the origin chosen for Y
- examine how the fitted model changes with 'lam'

Singular Value Decomposition

- The SVD of the centered matrix X is

$$X = UDV^T$$

where U and V are nxp and pxp orthogonal matrices, spanning the column and row space of X, respectively, and D is a diagonal matrix, with entries

$$d_1 \geq d_2 \geq \ldots \geq d_p > 0$$

called the singular values of X

SVD in NumPy

```
(U,D,Vt) = linalg.svd(X, full_matrices=0)
        # try > help(linalg.svd)
print 'D = ',D
```

Using the prostate data, after centering

```
D =  [ 15.0383645   10.37974957   8.27167793
  6.38816904   5.48125767
   4.98617689   4.29863604   3.39037097]
```

```
V = transpose(Vt)
diagD = diag(D)
Xc_svd = U*diagD*transpose(V)
```


Principal Components

What does a small value of d_j mean?

- The SVD of the centered matrix X is an expression of the principal components of the variables in X.
  - The sample covariance matrix is $S = X^TX/n$, and from the SVD, $X^TX = VD^2V^T$, which is the eigen decomposition of X.

Principal Components

- The eigenvectors v_j are called the principal component directions of X, and $z_j = Xv_j = u_jd_j$ the principal components of X
  - The first PCD has the largest sample variance among normalized linear combinations of the columns of X, $Var(z_1) = d_1^2/n$
  - Subsequent PCDs have maximum variance $Var(z_j) = d_j^2/n$, subject to being orthogonal to earlier ones.
  - The last PCD has minimum variance
- Hence small values of d_j correspond to directions in the column space of X having small variance.

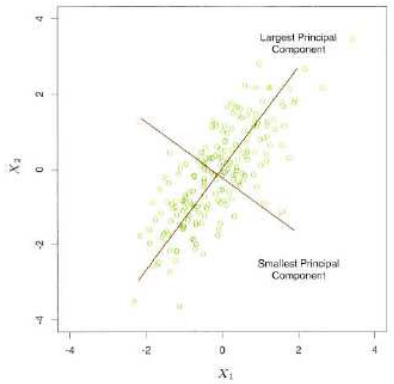


FIGURE 3.8. *Principal components of some input data points. The largest principal component is the direction that maximizes the variance of the projected data, and the smallest principal component minimizes that variance. Ridge regression projects* $\mathbf{y}$ *onto these components, and then shrinks the coefficients of the low-variance components more than the high-variance components.*

Principal components of ridge regression

Using the prostate data

```
V =  [[ 0.43599584  0.16758044  0.24092132  0.0303608   0.39573236
   0.45997711  0.39453064  0.44611892]
 [-0.03227514 -0.56469351 -0.41986465 -0.6502636   0.17548565  0.170018
   0.05512569  0.13494557]
 [ 0.28475054  0.38999626 -0.36634386 -0.06190952  0.4215279   0.20636854
  -0.5530985  -0.32029321]
 [-0.05461908  0.01188075 -0.7533558   0.53917791 -0.13979017  0.10108685
   0.19665613  0.2649233 ]
 [ 0.39776015 -0.69191378  0.15179509  0.46222069  0.07040449  0.12225473
  -0.17762098 -0.27367907]
 [-0.63588774 -0.13434217  0.12063284  0.26049982  0.67246729 -0.04927231
  -0.12736018  0.15596757]
 [ 0.28247126  0.01918153 -0.16074853 -0.01095237  0.3969871  -0.68455483
   0.45566909 -0.24501759]
 [-0.28712968  0.06032713 -0.0219931   0.00786043  0.02461322  0.46980754
   0.49123416 -0.67136858]]
```

Huh???

SVD of Least Squares

- The least squares fitted vector can be written as

$$X\hat{\beta}^{ls} = X(X^TX)^{-1}X^Ty = UU^Ty$$

- While the ridge solutions are

$$\begin{aligned} X\hat{\beta}^{ridge} &= X(X^TX + \lambda I)^{-1}X^Ty \\ &= UD(D^2 + \lambda I)^{-1}DU^Ty \\ &= \sum_{j=1}^{p} u_j \frac{d_j^2}{d_j^2 + \lambda} u_j^Ty \end{aligned}$$

where the u_j are the columns of U. Since $\lambda \geq 0$, the fraction is ≤ 1, and represents the `shrinkage'. Note that more shrinkage is applied to basis vectors with greater d_j.

Degrees of freedom

The effective degrees of freedom of the ridge regression fit is

$$df(\lambda) = tr\left[X(X^TX + \lambda I)^{-1}X^T\right] = \sum_{i=1}^{p} \frac{d_j^2}{d_j^2 + \lambda}$$

which is monotone in λ. Using prostate data

```
lam = 32
dflambda = 0
for i in range(p):
   dflambda = dflambda + (D[i:i+1]**2)/(D[i:i+1]**2 + lam)
print 'dflambda = ',float(dflambda)
```

```
dflambda =  4.44068952992
```

```
dflambda2 = trace(Xc*linalg.inv(transpose(Xc)*Xc+lam*eye(8))*transpose(Xc))
print 'dflambda2 = ',dflambda2
```

```
dflambda2 =  4.44068952992
```

- Program in `prostate_svd.py'

Exercise 5.7

Write a program to obtain least squares and ridge regression results using the SVD. Test on the prostate cancer training data.

Aside: The Lasso

- Another shrinkage method, replacing the quadratic (L_2) penalty $\Sigma_j \beta_j^2$ by the L_1 penalty $\Sigma_j |\beta_j|$
- The solutions are now nonlinear in y
- For large enough penalty, some of the coefficients will be exactly zero, and so the lasso does a kind of continuous subset selection.

Principal components regression

- Instead of regressing on the original predictors, regress on the principal components calculated from them
  - Thus step-wise regression is easy – just include the PCs in decreasing order of their singular values
- Form derived input columns $z_m = Xv_m$ and then regress y on $z_1, z_2, \ldots z_M$ for some $M \leq p$.
- Since the z's are orthogonal, this is just a sum of univariate regressions

$$\hat{y}^{pcr} = \bar{y} + \sum_{m=1}^{M} \hat{\theta}_m z_m = \bar{y} + \sum_{m=1}^{M} \left(\frac{z_m^T y}{z_m^T z_m} \right) z_m$$

and hence

$$\hat{\beta}^{pcr}(M) = \sum_{m=1}^{M} \hat{\theta}_m v_m$$

Principal components regression

- As usual, standardize inputs first
- If M=p, we get the usual least square estimates, as Z=UD spans the column space of X. For M<p we get a reduced regression.
- The result is similar to ridge regression, although ridge regression shrinks depending on the size of the eigenvalue, while PCR simply discards the p-M smallest eigenvalue components

PCR in NumPy

```
m = 4
Z = Xc*V      # use centered inputs
Z4 = Z[:,0:m] # first m PCs
Z41 = concatenate((ones((shape(Z4)[0],1)),Z4),axis=1)  # add intercept term
thetahat = linregp_ni(Z41,y)[0]
print 'thetahat = ',thetahat

thetahat =  [[ 2.45234522]
 [ 0.57214958]
 [-0.63809705]
 [-0.19749053]
 [ 0.33716487]]

V4 = V[:,0:m]
betahat_pcr = concatenate((thetahat[0],V4*thetahat[1:m+1,0]),axis=0)
print 'betahat_pcr = ',betahat_pcr

betahat_pcr =  [[ 2.45234522]
 [ 0.10517917]
 [ 0.2055363 ]
 [-0.03445983]
 [ 0.29174113]
 [ 0.79495294]
 [-0.21409201]
 [ 0.17742969]
 [-0.195782  ]]
```

- Program in 'prostate_pcr.py'

Selection versus shrinkage

- PCR and ridge regression behave similarly. The latter may be preferred as it shrinks smoothly, not in discrete steps.
- The lasso, ridge regression and best subset selection can be seen as Bayes estimates (see later) with different priors.
  - However they are modes of the posterior distribution, only ridge regression is also a posterior mean.