Model Selection and Assessment

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

Reading for this Part

- Sections 3.3.4 – 3.3.7 in the book
  - Some pages copied from Hastie et al are on the web site
- You can get the following PDF files from the library (in rough order of readability):
  - Ward EJ (2008) A review and comparison of four commonly used Bayesian and maximum likelihood model selection tools. Ecological Modelling 211, 1-10
  - Chatfield C (1995) Model uncertainty, data mining and statistical inference. J Roy Stat Soc A, 158, 419-466
  - Draper D (1995) Assessment and propagation of model uncertainty. J Roy Stat Soc B, 57, 45-97
  - Efron B (2004) The estimation of prediction error: covariance penalties and cross-validation. J Amer Stat Soc 99, 619-632
  - Shen X, Huang H-C (2006) Optimal model assessment, selection and combination. J Amer Stat Soc 101, 554-568
- Other electronic resources:
  - http://www.mdl-research.org/

Model Building

1. Determine a priori form of model relating predictors and response
   - Requires knowledge of subject matter
   - Try linear unless reason not to
2. For each predictor, specify allowed complexity or nonlinearity (d.o.f.)
3. Data reduction
   - Reduce number of predictors
     - Redundant variables?
     - Principal components?

Model Validation

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 against independent data to see how well it predicts.

Model Selection and Assessment

Two separate goals:

- *Model Selection:*
  - Estimate the performance of different models in order to choose the best (or at least a good) one
- *Model Assessment:*
  - Once the final model is chosen, estimate its prediction error (generalization error) on new data.

Splitting the data

Split the dataset (randomly) into three parts:

- *Training set:*
  - used to fit the models.
- *Validation set:*
  - used to estimate prediction error for model selection.
- *Test set:*
  - used to assess the prediction error of the final chosen model.

A typical split might be 50:25:25

Example: Prostate data

```
data = loadtxt('prostate_train.txt') # training data
p = 8
data = data.reshape(-1,p+1) ## reformat data as X and Y
y = data[:,p]
X = data[:,0:p]
y = transpose(matrix(y))
X = matrix(X)

# standardize data to unit variance
covX = cov(transpose(X))
sdX = sqrt(diag(covX))
for i in range(p):
  X[:,i] = X[:,i]/sdX[i]
Xred = concatenate((X[:,0:2],X[:,3:5]),axis=1) # reduced model

data2 = loadtxt('prostate_test.txt') ## test data
data2 = data2.reshape(-1,p+1)
y2 = data2[:,p] ## reformat data as X and Y
X2 = data2[:,0:p]
y2 = transpose(matrix(y2))
X2 = matrix(X2)
for i in range(p):
  X2[:,i] = X2[:,i]/sdX[i] # standardise according to TRAINING data variance
Xred2 = concatenate((X2[:,0:2],X2[:,3:5]),axis=1) # reduced model
```


Test on Independent Data

```
tempfull = linregp(X,y)  # full model, beta est. from
                         # training data
betafull = tempfull[0]
RSSfull = tempfull[2]
# fit full model, training, to test data
# USE beta estimated from training data !!!
X21 = concatenate((ones((shape(X2)[0],1)),X2),axis=1)
yhatfull2 = X21*betafull
RSSfull2 = transpose(y2 - yhatfull2)*(y2 - yhatfull2)

tempred = linregp_ni(Xred,y) # reduced model, beta est. from
                             # training data
betared = tempred[0]
RSSred = tempred[2]
# fit reduced model, training, to test data
# USE beta estimated from training data !!!
yhatred2 = Xred2*betared
RSSred2 = transpose(y2 - yhatred2)*(y2 - yhatred2)

print 'full model: RSS(train), RSS(test) =',RSSfull,RSSfull2
print 'reduced model: RSS(train), RSS(test) =',RSSred,RSSred2
```

```
full model: RSS(train), RSS(test) = [[ 29.42638396]] [[ 17.58986697]]
reduced model: RSS(train), RSS(test) = [[ 32.90740736]] [[ 14.9072031]]
```

- Program in `prostate_split.py`

Exercise 6.1

Using the prostate data, split the training data 45:22, so as to have training, validation and test data

Fit preferred model(s) from previous notes, exercises and assignments to the training data to estimate the parameters, then evaluate them on the validation data

Use the test data to estimate the prediction error of the best model, and compare with prediction error from the training and validation sets.

Data poor situation

- The rest of this part looks at methods for when there is insufficient data to split into separate training, validation and test sets.
  - *Approximate validation* either
    - analytically (AIC, BIC, MDL, SRM)
    - by efficient sample reuse (cross-validation, bootstrap)

Measuring Performance

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

- Typical choices of *Loss function:*

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

Generalization Error

- *Test error* (Generalization error)

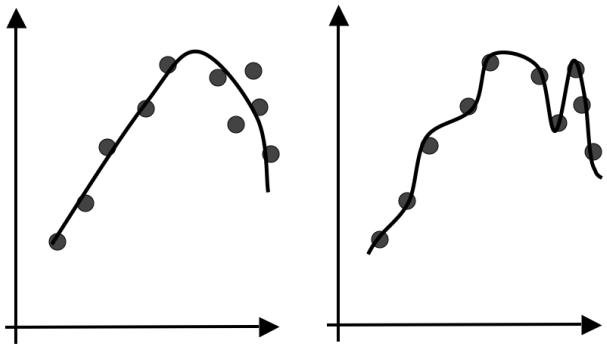
$$Err=E\left[L\left(Y,\hat{f}(X)\right)\right]$$

  - Expected prediction error over independent test sample
    - This expectation averages anything that is random, including the randomness in the training sample that produced the model
- *Training error*

$$\overline{err}=\frac{1}{N}\sum_{i=1}^{n}L\left(y_i,\hat{f}(x_i)\right)$$

  - average loss over training sample
    - not a good estimate of test error

Training Error - Overfitting



Bias-variance trade-off

- There is an optimal model complexity that gives minimum test error.
- Training error is not a good estimate of the test error.
- There is a bias -variance tradeoff in choosing the appropriate complexity of the model.

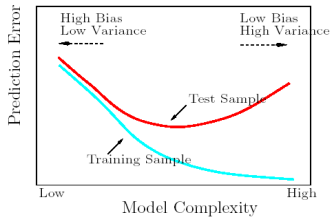


Figure 7.1: *Behavior of test sample and training sample error as the model complexity is varied.*

Optimism of the Training Error Rate

Typically, the training error rate will be less than the true test error. The same data is being used to fit the method and assess its error.

The *optimism* is the expected difference between the *in-sample error*

$$\mathrm{Err}_{in} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{Err}(x_i)$$

and the training error:

$$\mathrm{op} \equiv \mathrm{Err}_{in} - E_y(\overline{err})$$

The amount by which $\overline{err}$ underestimates the true error depends on how strongly y_i affects its own prediction.

For the linear fit with *d* inputs, $Y = f(X) + \varepsilon$, this simplifies to

$$\mathrm{Err}_{in} - E_y(\overline{err}) = 2 \cdot \frac{d}{N}\sigma_\varepsilon^2$$

How to estimate prediction error

- Estimate the *optimism* and then add it to the *training error rate*.
  - Methods such as AIC, BIC work in this way for a special class of estimates that are *linear* in their *parameters*.
- Estimating *in-sample error* is used for *model selection*.
- Methods like *cross-validation* and *bootstrap:*
  - direct estimates of the *extra-sample error* (the average prediction error when both features - position of evaluation points - and responses are new).
  - can be used with any loss function
  - used for *model assessment*.

In NumPy

```
yhatfull = linregp(X,y)[1] # full model, training
yhatred = linregp_ni(Xred,y)[1] # reduced model, training

## Optimism
opfull = 2.0*len(betafull)*sigmahat2/len(y)
opred = 2.0*len(betared)*sigmahat2/len(y)

## Training error
errfull = RSSfull/len(y)
errred = RSSred/len(y)

## In sample error
Errfull = errfull + opfull
Errred = errred + opred

print 'err,op,Err (full)=',errfull,opfull,Errfull
print 'err,op,Err (reduced)=',errred,opred,Errred
```

```
err,op,Err (full)= 0.439199760572 0.136303373971 0.575503134543
err,op,Err (reduced)= 0.49115533374 0.0605792773203 0.55173461106
```

- Program in 'prostate_split.py'

Example: Prostate Data

Recall full (9 predictor) and reduced (4 predictors, not including the intercept)

Optimism

err,op,Err (full)= 0.439199760572 0.136303373971 0.575503134543

err,op,Err (reduced)= 0.49115533374 0.0605792773203 0.55173461106

➢ Reduced is better

Compare with test using independent data:

full model, RSS(test) = [[17.58986697]]

reduced model, RSS(test) = [[14.9072031]]

➢ Same conclusion

Effective Number of Parameters

$y=\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$ Vector of Outcomes, similarly for predictions

$\hat{y} = Sy$ Linear fit (e.g. linear regression, quadratic shrinkage – ridge, splines)

where S is an $N \times N$ matrix, depending on input vectors x_i but not on y_i

effective number of parameters: $d(S) = \text{trace}(S)$

and d(S) is the correct *d* for C_p

$$C_p = \overline{\text{err}} + \frac{2d}{N}\hat{\sigma}_\varepsilon^2$$

Degrees of freedom

The effective degrees of freedom of the ridge regression fit is

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

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
```

Akaike Information Criterion (AIC) for model selection

- Choose the model giving smallest AIC over the set of models considered.
  - Given a set of models $f_\alpha(x)$ indexed by a tuning parameter α, define

$$\mathrm{AIC}(\alpha) = \overline{\mathrm{err}}(\alpha) + 2\frac{d(\alpha)}{N}\hat{\sigma}_\varepsilon^2$$

  where $\overline{\mathrm{err}}(\alpha) = \mathrm{RSS}/N$
- Find the tuning parameter $\hat{\alpha}$ that minimizes the function, and the final chosen model is $f_{\hat{\alpha}}(x)$

In NumPy

```
N = len(y)

# get sigma2 from full (low-bias) model
sigma2 = (std(yhatfull - y))**2
print 'sigma2=',sigma2

sigma2= 0.439199760572

# calculate AIC
# full model
aicfull = RSSfull/N + 2.0*len(betafull)*sigma2/N
# reduced model
aicred = RSSred/N + 2.0*len(betared)*sigma2/N
print 'AIC (full | reduced) =',aicfull,aicred

AIC (full | reduced) = [[ 0.55719373]] [[ 0.5435971]]
```

- Program in 'prostate_aic.py'

Example: Prostate Data

Recall full (9 predictor) and reduced (4 predictors, not including the intercept)

AIC (full | reduced) = 0.55719373 | 0.5435971

- Reduced is better

Compare with test using independent data:

full model, RSS(test) = 17.58986697

reduced model, RSS(test) = 14.9072031

- Same conclusion

Debugging Exercise

'prostate_aic_0.py' is a working version of 'prostate_aic.py' that produces different values.

What is the error?

NB: This is a more difficult type of debugging – the program runs, after all ☺

Exercise 6.2

By modifying the forward stepwise regression program for the prostate cancer data, obtain a set of 9 nested regression models, with 1,2,…,9 parameters, respectively.

Calculate the AICs for these 9 nested models, and hence identify the best.

Check these against the test data.

Bayesian Approach and BIC

Bayesian Information Criterion (BIC):

Assuming Gaussian model: σ_ε^2 known,

$$\mathrm{BIC} = \frac{N}{\sigma_\varepsilon^2}[\overline{\mathrm{err}} + (\log N)\cdot\frac{d}{N}\sigma_\varepsilon^2]$$

So BIC is proportional to AIC except for the log(N) rather than factor of 2. For $N>e^2$ (approx 7.4), BIC penalizes complex models more heavily.

AIC or BIC?

- BIC is asymptotically consistent as a selection criterion.
  - Given a family of models including the true model, the probability that BIC will select the correct one approaches one as the sample size becomes large.
- AIC does not have the above property.
  - It tends to choose more complex models as $N \to \infty$.
- For small or moderate samples, BIC often chooses models that are too simple, because of its heavy penalty on complexity.
- BIC is also formally identical to the minimum description length (MDL) approach, where the model is used as a code to transmit the parameters, inputs and outputs. The best model is the one with the smallest message length

Exercise 6.3

Repeat Exercise 6.2 using BIC, and compare with the AIC results

BIC: How much better is a model?

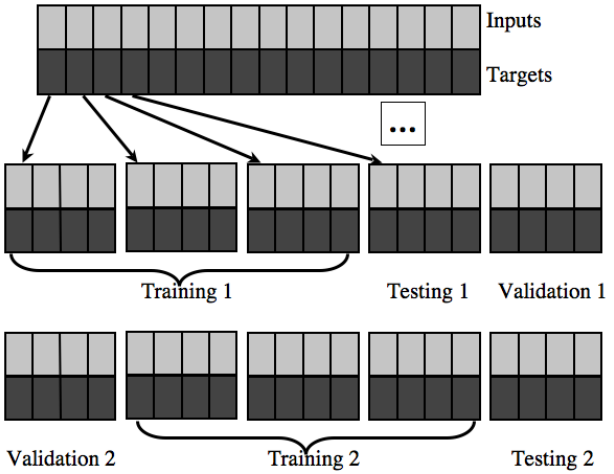
We may want to know more than just the relative *ranking* of various models

Once we have the BIC for a set of M models, the posterior probability of model m is

$$\frac{e^{-\frac{1}{2}\mathrm{BIC}_m}}{\sum_{k=1}^{M} e^{-\frac{1}{2}\mathrm{BIC}_k}}$$

Cross-Validation

- The simplest and most widely used method for estimating prediction error.
- The idea is to directly estimate the extra sample error, when the method $\hat{f}(X)$ is applied to an independent test sample from the joint distribution of X and Y
- In K-fold cross-validation, we split the data into roughly equal-size parts. For the k-th part, fit the model to the other K-1 parts and calculate the prediction error of the fitted model when predicting the k-th part of the data.



Cross-Validation (2)

- The cross-validation estimate of prediction error is

$$\mathrm{CV}(\alpha) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}^{-k(i)}(x_i, \alpha)).$$

- This $\mathrm{CV}(\alpha)$ provides an estimate of the test error, and we find the tuning parameter $\hat{\alpha}$ that minimizes it.
- Our final chosen model will be $f(x, \hat{\alpha})$, which we fit to all the data.

Cross-Validation in NumPy

```
K = 5 # divide training data into K sets
RSSCV = zeros((K,2)) # Storage for RSS
setno = zeros((N,))  # ARRAY indexing which set the row belongs to
for i in range(N):
    setno[i] = random.randint(0,K,1) # assign data among K sets
for k in range(K):
    yk = y[setno == k]                          ## data in kth set
    Xk = X[setno == k,:]
    Xkred = Xred[setno == k,:]
    ynotk = y[setno != k]                       ## data not in kth set
    Xnotk = X[setno != k,:]
    Xnotkred = Xred[setno != k,:]
    betakfull = linregp(Xnotk,ynotk)[0]         ## fit model to data
                                                ## not in kth set
    betakred = linregp_ni(Xnotkred,ynotk)[0]
    Xk = concatenate((ones((shape(Xk)[0],1)),Xk),axis=1)
    ykfull = Xk*betakfull                       ## test model on data
                                                ## in kth set
    RSSCV[k,0] = transpose(yk - ykfull)*(yk - ykfull)
    ykred = Xkred*betakred
    RSSCV[k,1] = transpose(yk - ykred)*(yk - ykred)
```


Output

```
MRSSCV = sum(RSSCV, axis = 0)/N      ## get average
                                     ## over all data

print 'Total Loss (CV) [train | test]=',RSSCV
print 'Mean Loss RSS [train | test]=',MRSSCV
```

```
Total Loss (CV) [train | test]= [[ 10.06978204
  10.7250108 ]
 [  6.9546141    6.68171365]
 [  6.88769352   7.07132628]
 [  6.62944691   6.90369979]
 [  9.84254578   5.88584293]]
Mean Loss RSS [train | test]= [ 0.6027475
  0.55623274]
```


Example: Prostate Data

Recall full (9 predictor) and reduced (4 predictors, not including the intercept)

Total Loss (CV) [train | test]= [[10.06978204 10.7250108]
 [6.9546141   6.68171365]
 [6.88769352   7.07132628]
 [6.62944691   6.90369979]
 [9.84254578   5.88584293]]
Mean Loss RSS [train | test]= [0.6027475 0.55623274]

- Reduced is better

Compare with test using independent data:
full model, RSS(test) = [[17.58986697]]
reduced model, RSS(test) = [[14.9072031]]

- Same conclusion

Debugging Exercise

'prostate_crossval_0.py' is a slightly different version of 'prostate_crossval.py' that doesn't run.

Try and work out why, without looking at the version that works.

Exercise 6.4

Modify 'prostate_crossval.py' to make the sets as equal as possible. Then test to see if there is any noticeable difference.

Value of K?

- If $K = N$, then CV is approximately unbiased, but has high variance. The computational burden is also high. This is sometimes called the Jackknife
- On the other hand, with $K = 5$, say, CV has low variance but more bias.
- If the learning curve has a considerable slope at the given training set size, CV will overestimate the true prediction error.
  - K = 5 or 10 is recommended as a good compromise

Exercise 6.5

Try different values of K and examine the results (mean and standard deviation of the mean loss RSS) using multiple runs

Bootstrap Method

- General tool for assessing statistical accuracy.
- Suppose we have a model to fit the training data

$$Z = \{(x_i, y_i), i = 1, \ldots, N\}.$$

  - The idea is to draw random samples with replacement of size N from the training data. This process is repeated B times to get B bootstrap datasets.
    - Refit the model to each of the bootstrap datasets and examine the behavior of the fits over B replications.

Bootstrap (2)

- Let S(Z) be any quantity computed from the data Z.
  - Bootstrap sampling can estimate any aspect of the distribution of S(Z)
  - For example, its variance is estimated by

$$\hat{\mathrm{Var}}(S(Z)) = \frac{1}{B-1}\sum_{b=1}^{B}(S(Z^{*b}) - \bar{S}^{*})^2,$$

where

$$\bar{S}^{*} = \sum_b S(Z^{*b})/B,$$

and Z^{*b} is the bth bootstrap sample

Bootstrap (3)

In particular, in regression, $\beta = \beta(Z)$ is a quantity computed from the data, as is RSS = RSS(Z).

Bootstrap used to estimate prediction error

Fit the model on a set of bootstrap samples keeping track of how well it predicts the original training set.

If $\hat{f}^{*b}(x_i)$ is the predicted value at x_i from the model fitted to the bth bootstrap dataset

$$\hat{Err}_{boot} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i)).$$

In NumPy

```
B = 100                 # number of bootstrap samples
X1 = concatenate((ones((shape(X)[0],1)),X),axis=1)
BRSS = zeros((B,2))     # bootstrap estimated RSS
                        # [full | reduced models]
for b in range(B):
    ytemp = matrix(zeros(shape(y)))  ## bootstrap samples
    Xtemp = matrix(zeros(shape(X)))
    Xredtemp = matrix(zeros(shape(Xred)))
    for n in range(N):
        temp = random.randint(0,N,1)
        ## add selected to bootstrap sample
        ytemp[n] = y[temp]
        Xtemp[n,:] = X[temp,:]
        Xredtemp[n,:] = Xred[temp,:]
    ## fit models to bootstrap samples to estimate beta
    bootfull = linregp(Xtemp,ytemp)[0]
    bootred = linregp_ni(Xredtemp,ytemp)[0]
    ## calculate RSS (bootstrap errors)
    BRSS[b,0] = transpose(y - X1*bootfull)*(y - X1*bootfull)/N
    BRSS[b,1] = transpose(y - Xred*bootred)*(y - Xred*bootred)/N

Errboot = mean(BRSS,axis = 0)
print 'Bootstrap Error [full | reduced]=',Errboot

Bootstrap Error [full | reduced]= [ 0.50888163  0.52247026]
```

The problem of bias

- The bootstrap datasets are acting as the training samples, while the original training set is acting as the test sample
  - Have observations in common
    - Unrealistic (too good)
  - Fix by
    - Bootstrap cross validation
    - The 0.632 estimator

Bootstrap CV

- Fit the model on a set of bootstrap samples keeping track of predictions from bootstrap samples not containing that observation.
- The leave-one-out bootstrap estimate of prediction error is

$$\widehat{Err}^{(1)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} L(y_i, \hat{f}^{*b}(x_i)).$$

where C^{-i} is the set of indices of the bootstrap samples b that do not contain i

In NumPy

```
BRSScv = zeros((N,3))  # error for yi [full | reduced | number of estimates]
for b in range(B):
    ...    ## ... INDICATES LINES FROM PREVIOUS PROGRAM ##
    unselected = list(arange(N))    ## the (un)selected elements we will test over (CV)
    for n in range(N):
        temp = random.randint(0,N,1)
        if sum(unselected == temp) > 0:
            unselected.remove(temp)
        ytemp[n] = y[temp]   ## add selected to bootstrap sample
        ...
    bootfull = linregp(Xtemp,ytemp)[0] ## fit to bootstrap samples
    ...
    lenus = len(unselected)
    for i in range(lenus):   ## bootstrap CV errors
        j = unselected[i]
        BRSScv[j,0] = BRSScv[j,0] + (y[j] - X1[j,:]*bootfull)**2
        BRSScv[j,1] = BRSScv[j,1] + (y[j] - Xred[j,:]*bootred)**2
        BRSScv[j,2] = BRSScv[j,2] + 1
for n in range(N):  ## normalize bootstrap CV errors by number of bootstrap sets
    BRSScv[n,0] = BRSScv[n,0]/BRSScv[n,2]
    BRSScv[n,1] = BRSScv[n,1]/BRSScv[n,2]
BRSScv = BRSScv[:,0:2]

Errbootcv = mean(BRSScv,axis = 0)
print 'Bootstrap CV Error [full | reduced]=',Errbootcv

Bootstrap CV Error [full | reduced]= [ 0.6664066   0.59417235]
```


The 0.632 Estimator

- Not out of the woods yet.
  - Average number of distinct observations in each bootstrap sample is approximately 0.632 N
  - Bias will roughly behave like that of two-fold cross-validation (biased upwards).
  - The "0.632 estimator" is designed to get rid of this bias.

$$\widehat{Err}^{(0.632)} = 0.368 \cdot \overline{err} + 0.632 \cdot \widehat{Err}^{(1)}.$$

In NumPy

```
## Training errors
errfull = float(linregp(X,y)[2]/N)
errred = float(linregp_ni(Xred,y)[2]/N)
errall = array([errfull,errred])
est632 = 0.632*Errbootcv + 0.368*errall
print 'Bootstrap 0.632 estimator [full
  | reduced]=',est632

Bootstrap 0.632 estimator [full |
  reduced]= [ 0.58279448  0.55626209]
```

- Program in 'prostate_boot.py'

Example: Prostate Data

Recall full (9 predictor) and reduced (4 predictors, not including the intercept)

Bootstrap
Bootstrap Error [full | reduced]= [0.50888163 0.52247026]
Bootstrap CV Error [full | reduced]= [0.6664066 0.59417235]
Bootstrap 0.632 estimator [full | reduced]= [0.58279448 0.55626209]

- Reduced is better

Compare with test using independent data:
full model, RSS(test) = [[17.58986697]]
reduced model, RSS(test) = [[14.9072031]]

- Same conclusion

A final step

Model Averaging:

- averaging the predictions from different models to achieve improved performance.
- not applicable to linear regression

Methods of Model Averaging

- Simple unweighted average of predictions (each model equally likely)
- Weighted (frequentist) average
- Bayesian averaging: use BIC to estimate posterior model probabilities and weight each model depending on fit and how many parameters it uses
- *Bagging:*
  - Acronym for 'Bootstrap aggregation'
    - averages the prediction over a collection of bootstrap samples, thus reducing the variance in prediction.

Bayesian Model Averaging

- Candidate models: $M_m, m = 1,\ldots,M.$
- Posterior distribution and mean:

$$\Pr(\zeta \mid Z) = \sum_{m=1}^{M} \Pr(\zeta \mid M_m, Z)\Pr(M_m \mid Z),$$

$$E(\zeta \mid Z) = \sum_{m=1}^{M} E(\zeta \mid M_m, Z)\Pr(M_m \mid Z).$$

- Bayesian prediction (posterior mean) is a weighted average of individual predictions, with weights proportional to posterior probability of each model.
- Posterior model probabilities $\Pr(M_m|Z)$ can be estimated by BIC.

Bagging

- Consider the regression problem with training data $Z = \{(x_1, y_1),\ldots,(x_N, y_N)\}$
- Fit a model and get a prediction $\hat{f}(x)$ at the input x.
- For each bootstrap sample $Z^{*b}, b = 1,\ldots,B,$
  - fit the model, with prediction $\hat{f}^{*b}(x)$.
  - Then the bagging (or, bagged) estimate is:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

Exercise 6.6

Modify the bootstrap program to construct a bagged linear regression estimate for the prostate cancer data.