Best-subset Linear Regression Analysis on Prostate Cancer Data (in R)

Lucy L.

2024-01-14

This exercise was derived from "The Elements of Statistical Learning," chapter 7 exercise 7.9: "For the prostate data of Chapter 3, carry out a best-subset linear regression analysis, as in Table 3.3 (third column from left). Compute the AIC, BIC, five- and tenfold cross-validation, and bootstrap .632 estimates of prediction error. Discuss the results." The functions in this write-up were written by John Weatherwax (email: `wax@alum.mit.edu`).

Here I include a summary of the prediction error estimates (mean squared error, MSE) by method. Extended discussion and the recommended models per method can be found from scrolling through this document.

```
##                                Method Test.MSE
## 1 Cross-validation (5- and 10-fold)    0.492
## 2                                 AIC    0.517
## 3                                 BIC    0.492
## 4                   Bootstrap 0.632    0.490
```

The models recommended by CV, BIC, and bootstrap 0.632 are the simplest and recommend the same two predictors. In fact, BIC and bootstrap 0.632 recommend the same model. Yet, bootstrap 0.632 results in a lower estimated prediction error than BIC, and the lowest out of all four.

```
estimate_632_Err = function(DF, x_vars, y_var, B=1000){
    # Create a formula for our model and fit it to this data:
    x_pt = paste(x_vars, collapse='+')
    y_pt = paste(c(y_var,'~'), collapse='')
    fmla = as.formula(paste(c(y_pt, x_pt), collapse=''))

    m = lm(fmla, data=DF)

    # Predict the insample error:
    err_bar = mean(residuals(m)^2)

    # Generate bootstrap samples:
    # the number of samples in the original dataset
    N = dim(DF)[1]
    SI = sample(N, N*B, replace=TRUE)
    # sample indices that are in each bootstrap replica
    SI = matrix(SI, nrow=B, ncol=N, byrow=TRUE)

    # Build linear models using each bootstrapped sample:
    all_lms = vector (mode='list', length=B)
```

```
    for( bi in 1:B ){
        m = lm(fmla, data=DF[SI[bi, ], ])
        all_lms[[bi]] =m
    }

    # Estimate Err^{(1)}
    err_parts = c()
    for( ii in 1:N ){

        # Find the set C^{(-i)} the set of bootstrap samples that don't
        # contain the observation x_i
        Cminusi = c()
        for( bi in 1:B ){
            if( !(ii %in% SI[bi,]) ){
                Cminusi = c(Cminusi, bi)
            }
        }
        if( length(Cminusi)==0 ){
            next
        }

        # Predict x_i using each of the models in C^{(-i)}
        residuals = c()
        for(ci in Cminusi){
            y_hat = predict(all_lms[[ci]], newdata=DF[ii,])
            residuals = c(residuals, DF[ii, y_var] - y_hat)
        }
        err_parts = c(err_parts, mean(residuals^2))
    }
    err_1 = mean(err_parts)

    # Combine:
    estimate = 0.368 * err_bar + 0.632 * err_1

    result = list(B=B, err_bar=err_bar, err_1=err_1, estimate=estimate)

    return(result)
}
```

```
# Sets up the predictor models to be used with the R command lm and all
# subsets cross validation

cv_all_subsets <- function(k,D,numberOfCV=10){
  # Input:
  #   k = the number of predictors to include in the search
  #   D = training data frames where the last column is the response variable :
  #   numberOfCV = number of cross validations to do
  #
  # Output:
  #   cvraw = ( y - y_hat )^2 i.e. the residual squared.
  #           Blocks of y_hat are predicted based on all the other data.
  #           Doing this repeatedly is the cross-validation part.
  #   cvm = mean of the residual squares vector.
```

```
#   cvsd = stderr of the mean residual square error.

p = dim( D )[2] - 1
# subtract the response which is the last column in this data frame
stopifnot( (0 <= k) && (k <= p) )
# check the input argument k
responseName = names(D)[p+1]
# the the name of the response

if( k==0 ){
  # do the k=0 (no features) subset as a special case:
  form                = paste( responseName, " ~ +1" )
  # this is the only possible formula in this case
  res                 = cvLMTrainNTest( form, D, numberOfCV )
  bestErrors          = res$cvraw
  bestFeaturesIndices = NULL
  bestFormula         = form

}else{
  # do all remaining k>0 subsets:
  allPosSubsetsSizeK = combn(p,k)
  # get all possible subsets of size k
  numOfSubsets = dim(allPosSubsetsSizeK)[2]

  for( si in 1:numOfSubsets ){
    print(sprintf("si=%10d; subsetSize=%10d; numOfSubsets=%10d; percentDone=%10.6f",
                  si,k,numOfSubsets,si/numOfSubsets))
    featIndices = allPosSubsetsSizeK[,si]
    featNames   = as.vector(names(D))[featIndices]

    # construct a formula needed for the linear regression:
    form = paste( responseName, " ~ " )
    for ( ki in 1:k ){
      if( ki==1 ){
        form = paste( form, featNames[ki], sep=" " )
      }else{
        form = paste( form, featNames[ki], sep="+" )
      }
    }

    res  = cvLMTrainNTest( form, D, numberOfCV )
    espe = res$cvraw

    if(si==1){
      # initial subset of size k becomes current best
      bestErrors          = espe
      bestFeaturesIndices = featIndices
      bestFormula         = form
    }else{
      # all subsequent subsets of size k must beat the current best
      if( mean(espe) < mean(bestErrors) ){
        bestErrors          = espe
        bestFeaturesIndices = featIndices
```

```
          bestFormula         = form
        }
      }
    }
    # end for subsets of size k loop we have an estimate of the best
    # formula predicted on the training data
  }
  # end if else k==0 special case

  # package a structure to send out:
  res = list(bestErrors,bestFeaturesIndices,bestFormula)

  return(res)
}
```

```
# Does Cross Validation of the Linear Model specified by the formula "form"

cvLMTrainNTest <- function( form, D, numberOfCV ){
  # Input:
  #   form = a string representation of the formula to use in the R call to "lm"
  #   D = training data frame where the last column is the response variable
  #   numberOfCV = number of cross validations to do
  #
  # Output:
  #  MSPE = vector of length numberOfCV with components mean square
  #  prediction errors extracted from each of the numberOfCV test data sets


  nSamples = dim( D )[1]
  p = dim( D )[2] - 1
  # subtract the response which is the last column in this data frame
  responseName = names(D)[p+1]
  # the the name of the response assumed to be the last column in the data frame

  # needed for the cross vaildation (CV) loops:
  nCVTest    = round( nSamples*(1/numberOfCV) )
  # each CV run will have this many test points
  nCVTrain   = nSamples-nCVTest
  # each CV run will have this many training points

  for (cvi in 1:numberOfCV) {

    testInds = (cvi-1)*nCVTest + 1:nCVTest
    # this may attempt to access sample indexes > nSamples
    testInds = intersect( testInds, 1:nSamples )
    # so we restrict this list here
    DCVTest  = D[testInds,1:(p+1)]
    # get the predictor + response testing data

    # select this cross validation's section of data from all the training data:
    trainInds = setdiff( 1:nSamples, testInds )
    DCV       = D[trainInds,1:(p+1)]
    # get the predictor + response training data
```

```
response            = DCV[,p+1]
DCV[[responseName]] = NULL

# Standardize the predictors and demean the response
#
# Note that one can get the centering value (the mean)
# with the command attr(DCV,'scaled:center')
# and the scaling value (the sample standard deviation)
# with the command attr(DCV,'scaled:scale')

DCV              = scale( DCV )
responseMean     = mean( response )
response         = response - responseMean
DCVb             = cbind( DCV, response )
# append back on the response
DCVf             = data.frame( DCVb )
# a data frame containing all scaled variables of interest
names(DCVf)[p+1] = responseName
# fix the name of the response

# extract the centering and scaling information:
means = attr(DCV,"scaled:center")
stds  = attr(DCV,"scaled:scale")

# apply the computed scaling based on the training data to the testing data:
responseTest            = DCVTest[,p+1]
# in physical units (not mean adjusted)
DCVTest[[responseName]] = NULL
DCVTest                 = t( apply( DCVTest, 1, '-', means ) )
DCVTest                 = t( apply( DCVTest, 1, '/', stds ) )
DCVTestb                = cbind( DCVTest, responseTest - responseMean )
# append back on the response
DCVTestf                = data.frame( DCVTestb )
# a data frame containing all scaled variables of interest
names(DCVTestf)[p+1]    = responseName
# fix the name of the response

# fit this linear model and compute the expected prediction error (EPE)
# using these features (this just estimates the mean in this case):
mk = lm( formula = form, data=DCVf )

pdt  = predict( mk, newdata=DCVTestf, interval="prediction" )[,1]
# get predictions on the test set
pdt  = pdt + responseMean
# add back in the mean.  Now in physical units (not mean adjusted)

if( cvi==1 ){
  predmat = pdt
  y       = responseTest
}else{
  predmat = c( predmat, pdt )
  y       = c( y, responseTest )
}
```

```
  }
  # end for cvi loop

  N = length(y)
  cvraw = ( y - predmat )^2
  cvm   = mean(cvraw)
  cvsd  = sqrt( var(cvraw)/N )
  l = list( cvraw=cvraw, cvm=cvm, cvsd=cvsd, name="Mean Squared Error" )

  return(l)
}
```

```
one_standard_error_rule <- function(complexityParam,cvResults){
  # Input:
  #   complexityParam = the sampled complexity parameter (duplicate cross
  #     validation results expected)
  #   cvResults = matrix where each column is a different value of the
  #     complexity parameter (with complexity increasing from left to right) ...
  #     each row is the square prediction error (SPE) of a different cross
  #     validation sample i.e. ( y_i - \hat{y}_i )^2
  #
  # Output:
  #   optCP = the optimal complexity parameter

  N = dim(cvResults)[1]

  # compute the complexity based mean and standard deviations:
  means = apply(cvResults,2,mean)
  stds  = sqrt( apply(cvResults,2,var)/N )

  # find the smallest espe:
  minIndex = which.min( means )

  # compute the confidence interval around this point:
  ciw = stds[minIndex]

  # add a width of one std to the min point:
  maxUncertInMin = means[minIndex] + 0.5*ciw

  # find the mean that is nearest to this value and that is a SIMPLER model
  # than the minimum mean model:
  complexityIndex = which.min( abs( means[1:minIndex] - maxUncertInMin ) )
  complexityValue = complexityParam[complexityIndex]

  # package everything to send out:
  res = list(complexityValue,complexityIndex,maxUncertInMin, means,stds)

  return(res)
}
```

```
load_prostate_data <- function(globalScale=FALSE, trainingScale=TRUE,
                               responseScale=FALSE){
```

```
X  = read.csv("prostate.csv")

if(globalScale){
  if(responseScale){
    lpsa = X$lpsa - mean(X$lpsa)
  }else{
    lpsa = X$lpsa
  }
  train = X$train
  X$lpsa = NULL
  X$train = NULL
  X = scale(X, TRUE, TRUE)
  Xf = data.frame(X)
  Xf$lpsa = lpsa
  Xf$train = train
  X = Xf
  rm(Xf)
  rm(lpsa)
}

# separate into training/testing sets
XTraining = subset(X, train)

# remove the training/testing column
XTraining$train = NULL
p = dim(XTraining)[2]-1
XTesting  = subset(X, train==FALSE)

# remove the training/testing column
XTesting$train = NULL

# Randomize the order of the rows in the Training data frame to make sure that
# each cross validation training/testing set is as random as possible

if(FALSE){
  nSamples = dim(XTraining)[1]
  inds = sample(1:nSamples, nSamples)
  XTraining = XTraining[inds,]
}

# Estimate the predictor statistics using the training set
# and then scale the testing set by the same statistics

if(trainingScale){
  X = XTraining
  if(responseScale){
    meanLpsa = mean(X$lpsa)
    lpsa = X$lpsa - meanLpsa
  }else{
    lpsa = X$lpsa
  }
  X$lpsa = NULL
  X = scale(X, TRUE, TRUE)
```

```
    means = attr(X,"scaled:center")
    stds = attr(X,"scaled:scale")
    Xf = data.frame(X)
    Xf$lpsa = lpsa
    XTraining = Xf

    # Scale the testing predictors by the same amounts:
    DCVTest  = XTesting
    if(responseScale){
      lpsaTest = DCVTest$lpsa - meanLpsa
    }else{
      # in physical units (not mean adjusted)
      lpsaTest = DCVTest$lpsa
    }
    DCVTest$lpsa = NULL
    DCVTest = t(apply(DCVTest, 1, '-', means))
    DCVTest = t(apply(DCVTest, 1, '/', stds))
    # append back on the response
    DCVTestb = cbind(DCVTest, lpsaTest)
    # a data frame containing all scaled variables of interest
    DCVTestf = data.frame(DCVTestb)
    # fix the name of the response
    names(DCVTestf)[p+1] = "lpsa"
    XTesting = DCVTestf
  }

  return(list(XTraining, XTesting))
}
```

```
# Group all cvResults by k (the subset size) and compute statistics:
OSE = one_standard_error_rule( complexityParam, cvResults )
complexVValue = OSE[[1]]
complexIndex  = OSE[[2]]
means = OSE[[4]]
stds = OSE[[5]]
oseHValue = OSE[[3]]

# Pick the best model, retrain over the entire data set, and predict on the
# testing data set:
bestModelFormula = bestPredictorFormula[ complexIndex ]

DCV  = XTraining

lpsa = DCV[,p+1] # get the response and delete it from the data frame
DCV$lpsa = NULL

# Standardize the predictors and demean the response
# Note that one can get the centering value (the mean) with the command
# attr(DCV,'scaled:center') and the scaling value (the sample standard
# deviation) with the command attr(DCV,'scaled:scale')

DCV      = scale( DCV )
lpsaMean = mean( lpsa )
```

```
lpsa     = lpsa - lpsaMean
DCVb     = cbind( DCV, lpsa )
# append back on the response
DCVf     = data.frame( DCVb )
# a data frame containing all scaled variables of interest


# extract the centering and scaling information:

means = attr(DCV,"scaled:center")
stds  = attr(DCV,"scaled:scale")

# apply the computed scaling based on the training data to the testing data:

DCVTest  = XTesting
# in physical units (not mean adjusted)
lpsaTest = DCVTest[,p+1]
NTest    = length(lpsaTest)
DCVTest$lpsa = NULL
DCVTest  = t( apply( DCVTest, 1, '-', means ) )
DCVTest  = t( apply( DCVTest, 1, '/', stds ) )
# append back on the response
DCVTestb = cbind( DCVTest, lpsaTest - lpsaMean )
# a data frame containing all scaled variables of interest
DCVTestf = data.frame( DCVTestb )
# fix the name of the response
names(DCVTestf)[p+1] = "lpsa"


# fit this linear model and compute the expected prediction error (EPE)
# using these features (this just estimates the mean in this case):
mk = lm( formula = bestModelFormula, data=DCVf )

# print the coefficients of this model:
# add back in the mean
# print( lpsaMean, digits=4 )
print( coef( mk ), digits=4 )
```

```
## (Intercept)      lcavol     lweight
##   1.583e-16   7.799e-01   3.519e-01
```

```
# get predictions on the test set
pdt  = predict( mk, newdata=DCVTestf, interval="prediction" )[,1]
# add back in the mean.  Now in physical units (not mean adjusted)
pdt  = pdt + lpsaMean
mErr = mean( (lpsaTest - pdt)^2 )
print( mErr )
```

```
## [1] 0.4924823
```

```
sErr = sqrt( var( (lpsaTest - pdt)^2 )/NTest )
#print( sErr )
```

Only the results of 10-fold cross validation are shown here, but the results from 5-fold cross validation are similar (`lcavol` and `lweight` were both chosen again with coefficients similar in magnitude.). I used the "one-standard-error" rule to select the optimal subset of size `k` ($k = 2$). The model chosen from 10-fold CV is thus the following (approximately):

$$\hat{y} = 0.779(lcavol) + 0.352(lweight)$$

(The intercept is 0.0000000000000001583, which is approximately zero.) The MSE was calculated to be about 0.492.

```
# the last column is the response
pp = dim(XTraining)[2]-1
nSamples = dim(XTraining)[1]

full_model = lm(lpsa ~ ., data = XTraining)

# AIC stepwise selection:
step(full_model, k=2)
```

```
## Start:  AIC=-37.13
## lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason +
##     pgg45
##
##           Df Sum of Sq    RSS     AIC
## - gleason  1    0.0109 29.437 -39.103
## <none>                 29.426 -37.128
## - age      1    0.9886 30.415 -36.914
## - pgg45    1    1.5322 30.959 -35.727
## - lcp      1    1.7683 31.195 -35.218
## - lbph     1    2.1443 31.571 -34.415
## - svi      1    3.0934 32.520 -32.430
## - lweight  1    3.8390 33.265 -30.912
## - lcavol   1   14.6102 44.037 -12.118
##
## Step:  AIC=-39.1
## lpsa ~ lcavol + lweight + age + lbph + svi + lcp + pgg45
##
##           Df Sum of Sq    RSS     AIC
## <none>                 29.437 -39.103
## - age      1    1.1025 30.540 -38.639
## - lcp      1    1.7583 31.196 -37.216
## - lbph     1    2.1354 31.573 -36.411
## - pgg45    1    2.3755 31.813 -35.903
## - svi      1    3.1665 32.604 -34.258
## - lweight  1    4.0048 33.442 -32.557
## - lcavol   1   14.8873 44.325 -13.681

##
## Call:
## lm(formula = lpsa ~ lcavol + lweight + age + lbph + svi + lcp +
##     pgg45, data = XTraining)
##
## Coefficients:
```

```
## (Intercept)       lcavol      lweight          age         lbph          svi
##    0.259062     0.573930     0.619209    -0.019480     0.144426     0.741781
##         lcp        pgg45
##   -0.205417     0.008945
```

```
subset_model = lm(lpsa ~ lcavol + lweight + age + lbph + svi + lcp + pgg45,
                  data = XTraining)
subset_model_test_preds =  predict(subset_model,
                           newdata = XTesting,
                           interval="prediction" )[,1]

mean((XTesting$lpsa - subset_model_test_preds)^2)
```

```
## [1] 0.5165135
```

The model recommended by the minimized AIC criterion is the following:

$$\hat{y} = 0.259 + 0.574(lcavol) + 0.619(lweight) - 0.019(age) + 0.144(lbph) + 0.742(svi) - 0.205(lcp) + 0.009(pgg45)$$

Only the predictor `gleason` was removed. The MSE for this model is about 0.517.

```
# BIC stepwise selction:
step(full_model, k=log(nSamples))
```

```
## Start:  AIC=-17.29
## lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason +
##     pgg45
##
##           Df Sum of Sq    RSS      AIC
## - gleason  1    0.0109 29.437 -21.4653
## - age      1    0.9886 30.415 -19.2762
## - pgg45    1    1.5322 30.959 -18.0892
## - lcp      1    1.7683 31.195 -17.5803
## <none>                 29.426 -17.2854
## - lbph     1    2.1443 31.571 -16.7775
## - svi      1    3.0934 32.520 -14.7929
## - lweight  1    3.8390 33.265 -13.2741
## - lcavol   1   14.6102 44.037   5.5196
##
## Step:  AIC=-21.47
## lpsa ~ lcavol + lweight + age + lbph + svi + lcp + pgg45
##
##           Df Sum of Sq    RSS      AIC
## - age      1    1.1025 30.540 -23.2065
## - lcp      1    1.7583 31.196 -21.7830
## <none>                 29.437 -21.4653
## - lbph     1    2.1354 31.573 -20.9779
## - pgg45    1    2.3755 31.813 -20.4703
## - svi      1    3.1665 32.604 -18.8249
## - lweight  1    4.0048 33.442 -17.1239
## - lcavol   1   14.8873 44.325   1.7517
##
```

```
## Step:  AIC=-23.21
## lpsa ~ lcavol + lweight + lbph + svi + lcp + pgg45
##
##           Df Sum of Sq    RSS      AIC
## - lcp      1    1.5297 32.069 -24.1367
## - lbph     1    1.6778 32.218 -23.8280
## - pgg45    1    1.7831 32.323 -23.6094
## <none>                 30.540 -23.2065
## - svi      1    3.2957 33.836 -20.5450
## - lweight  1    3.4138 33.954 -20.3117
## - lcavol   1   13.9954 44.535  -2.1355
##
## Step:  AIC=-24.14
## lpsa ~ lcavol + lweight + lbph + svi + pgg45
##
##           Df Sum of Sq    RSS      AIC
## - pgg45    1    0.7455 32.815 -26.8016
## - lbph     1    2.0047 34.074 -24.2788
## <none>                 32.069 -24.1367
## - svi      1    2.1873 34.257 -23.9207
## - lweight  1    3.4238 35.493 -21.5449
## - lcavol   1   12.6227 44.692  -6.1045
##
## Step:  AIC=-26.8
## lpsa ~ lcavol + lweight + lbph + svi
##
##           Df Sum of Sq    RSS     AIC
## - lbph     1    2.0928 34.908 -26.864
## <none>                 32.815 -26.802
## - lweight  1    3.1545 35.969 -24.857
## - svi      1    3.2002 36.015 -24.772
## - lcavol   1   15.7863 48.601  -4.691
##
## Step:  AIC=-26.86
## lpsa ~ lcavol + lweight + svi
##
##           Df Sum of Sq    RSS      AIC
## - svi      1    2.1841 37.092 -27.0027
## <none>                 34.908 -26.8641
## - lweight  1    7.4048 42.313 -18.1797
## - lcavol   1   16.8065 51.714  -4.7362
##
## Step:  AIC=-27
## lpsa ~ lcavol + lweight
##
##           Df Sum of Sq    RSS     AIC
## <none>                 37.092 -27.003
## - lweight  1     7.437 44.529 -18.964
## - lcavol   1    36.522 73.614  14.716

##
## Call:
## lm(formula = lpsa ~ lcavol + lweight, data = XTraining)
##
```

```
## Coefficients:
## (Intercept)       lcavol      lweight
##     -1.0494       0.6276       0.7384
```

```
subset_model2 = lm(lpsa ~ lcavol + lweight, data = XTraining)
subset_model2_test_preds =  predict(subset_model2,
                            newdata = XTesting,
                            interval="prediction" )[,1]

mean((XTesting$lpsa - subset_model2_test_preds)^2)
```

```
## [1] 0.4924823
```

The model recommended by the minimized BIC criterion is the following:

$$\hat{y} = -1.05 + 0.628(lcavol) + 0.738(lweight)$$

This is much smaller than the model recommended according to the AIC criterion since the BIC criterion penalizes the number of parameters relatively greatly. The MSE for this model is about 0.492.

```
# Estimate the prediction error using the "0.632 method"
err = estimate_632_Err(XTraining, c('lcavol', 'lweight'), 'lpsa', B=500)

mk = lm(lpsa ~ lcavol + lweight, data=XTraining)
y_hat = predict(mk, newdata=XTesting)
mse_test = mean((y_hat - XTesting[, 'lpsa'])^2)
print(sprintf('B= %d; err_bar= %.2f; err_1= %.2f; (0.632 estimate)= %.2f; mse_test= %.2f',
              err$B, err$err_bar, err$err_1, err$estimate, mse_test))
```

```
## [1] "B= 500; err_bar= 0.55; err_1= 0.63; (0.632 estimate)= 0.60; mse_test= 0.49"
```

Here I used the 0.632 estimator with bootstrap samples to calculate the prediction error (test MSE) of the given model $\hat{y} = -1.05 + 0.628(lcavol) + 0.738(lweight)$, which came out to be about 0.49.

The formula underlying this calculation weights the bootstrapped-based estimate $\widehat{Err}^{(1)}$ (0.63) and the training error $\overline{err}$ (0.55).