Department of Statistics

STATS 784: Data Mining

Assignment 1 2016 Model Answer

Question 1

You got full marks if you identified two applications, gave a reasonable precis of your source, and identified your source.

Question 2

For the Boston Housing data, construct a linear predictor using a subset (possibly all) of the features in the data set Boston (which you can find in the MASS package).

We will use the subset selection functions to find the subset with smallest PE.

First, we load the Boston data set from the MASS package and change the variables **chas** and **rad** to be factors (they are coded as integers in the data set):

```
> library(MASS)
> data(Boston)Boston$chas = factor(Boston$chas)
> Boston$rad = factor(Boston$rad)

> str(Boston)
'data.frame':   506 obs. of  14 variables:
 $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
 $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
 $ chas   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
 $ rm     : num  6.58 6.42 7.18 7 7.15 ...
 $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
 $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad    : Factor w/ 9 levels "1","2","3","4",..: 1 2 2 3 3 3 5 5 5 5 ...
 $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
 $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
 $ black  : num  397 397 393 395 397 ...
 $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
 $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

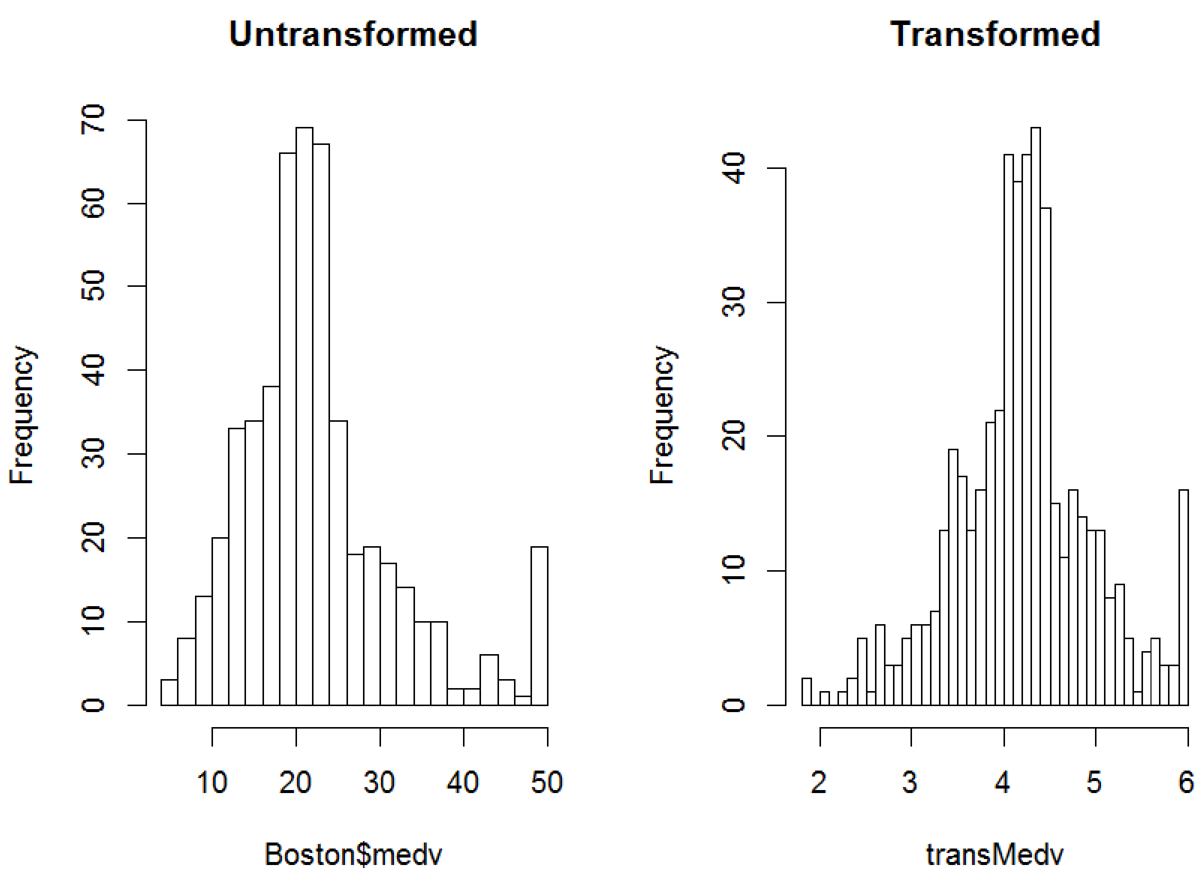
Next we transform the target to make it more symmetric, using the Box-Cox transformation (see Lecture 7 for more information)

```
> library(caret)
```

```
> myTrans = BoxCoxTrans(Boston$medv)
> myTrans
Box-Cox Transformation

506 data points used to estimate Lambda

Input data summary:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   5.00   17.02   21.20   22.53   25.00   50.00

Largest/Smallest: 10
Sample Skewness: 1.1

Estimated Lambda: 0.2

> transMedv = predict(myTrans, Boston$medv)
```

Let's plot the result and see if it has had an effect:

```
> par(mfrow = c(1,2))
> hist(Boston$medv, main = "Untransformed", nclass=30)
> hist(transMedv, main = "Transformed", nclass=30)
```



Not much in it but we will use the transformed data (a bit more symmetric).

Note the spike at 50 – data over 50 has been censored.

Let’s do some variable selection to see if we need all the variables:

```
library(R330)
nullModel = lm(transMedv~1, data=Boston)
fullModel = lm(transMedv ~ . - medv, data=Boston)
selectedModelBack = step(fullModel, formula(nullModel), direction="back")
selectedModelForward = step(nullModel, formula(fullModel), direction="forward")
selectedModelBoth = step(nullModel, formula(fullModel), direction="both")

> selectedModelBack$call
lm(formula = transMedv ~ crim + zn + chas + nox + rm + dis +
    rad + tax + ptratio + black + lstat, data = Boston)

> selectedModelForward$call
lm(formula = transMedv ~ lstat + ptratio + rm + crim + dis +
    nox + chas + black + rad + tax + zn, data = Boston)

> selectedModelBoth$call
lm(formula = transMedv ~ lstat + ptratio + rm + crim + dis +
    nox + chas + black + rad + tax + zn, data = Boston)
```

Seems like we can drop the variables indus and age.
Lets try all possible regressions:

```
> Allposregs(selectedModelBoth)
      rssp sigma2 adjRsq      Cp     AIC     BIC     CV crim zn indus chas1 nox rm
1  101.468  0.201  0.637 347.523 853.523 861.976 10.169    0  0     0     0   0  0
2   87.179  0.173  0.687 229.890 735.890 748.569  8.771    0  0     0     0   0  0
3   79.318  0.158  0.715 166.074 672.074 688.980  8.052    0  0     0     0   0  1
4   73.606  0.147  0.735 120.256 626.256 647.389  7.519    1  0     0     0   0  1
5   71.119  0.142  0.743 101.431 607.431 632.790  7.310    1  0     0     0   0  1
6   66.795  0.134  0.759  67.230 573.230 602.815  6.897    1  0     0     0   1  1
7   65.095  0.131  0.764  54.996 560.996 594.808  6.777    1  0     0     1   1  1
8   63.575  0.128  0.769  44.274 550.274 588.313  6.676    1  0     0     1   1  1
9   62.284  0.126  0.774  35.466 541.466 583.731  6.576    1  0     0     1   1  1
10  60.982  0.123  0.778  26.562 532.562 579.054  6.462    1  0     0     1   1  1
11  60.536  0.123  0.779  24.832 530.832 581.551  6.434    1  1     0     1   1  1
12  60.102  0.122  0.780  23.191 529.191 584.136  6.409    1  1     0     1   1  1
13  59.565  0.121  0.782  20.699 526.699 585.871  6.368    1  1     0     1   1  1
14  59.016  0.120  0.783  18.099 524.099 587.497  6.326    1  1     0     1   1  1
15  58.636  0.120  0.784  16.921 522.921 590.545  6.300    1  1     0     1   1  1
16  58.363  0.119  0.785  16.633 522.633 594.485  6.288    1  1     0     1   1  1
17  58.254  0.119  0.785  17.725 523.725 599.803  6.291    1  1     1     1   1  1
18  58.035  0.119  0.785  17.886 523.886 604.190  6.286    1  1     0     1   1  1
19  57.935  0.119  0.785  19.048 525.048 609.579  6.291    1  1     1     1   1  1
20  57.929  0.119  0.785  21.000 527.000 615.757  6.334    1  1     1     1   1  1
   age dis rad2 rad3 rad4 rad5 rad6 rad7 rad8 rad24 tax ptratio black lstat
1    0   0    0    0    0    0    0    0    0     0   0       0     0     1
2    0   0    0    0    0    0    0    0    0     0   0       1     0     1
3    0   0    0    0    0    0    0    0    0     0   0       1     0     1
4    0   0    0    0    0    0    0    0    0     0   0       1     0     1
5    0   1    0    0    0    0    0    0    0     0   0       1     0     1
6    0   1    0    0    0    0    0    0    0     0   0       1     0     1
7    0   1    0    0    0    0    0    0    0     0   0       1     0     1
8    0   1    0    0    0    0    0    0    0     0   0       1     1     1
9    0   1    0    0    0    0    0    0    0     1   0       1     1     1
```

```
10   0   1   0   0   0   0   0   0   0   1   1   1   1   1
11   0   1   0   0   0   0   0   0   0   1   1   1   1   1
12   0   1   0   0   0   0   0   1   0   1   1   1   1   1
13   0   1   0   1   0   0   0   1   0   1   1   1   1   1
14   0   1   0   1   0   0   0   1   1   1   1   1   1   1
15   0   1   0   1   0   1   0   1   1   1   1   1   1   1
16   0   1   0   1   1   1   0   1   1   1   1   1   1   1
17   0   1   0   1   1   1   0   1   1   1   1   1   1   1
18   0   1   1   1   1   1   1   1   1   1   1   1   1   1
19   0   1   1   1   1   1   1   1   1   1   1   1   1   1
20   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

Looks like model 18 has the smallest CV, which agrees with the stepwise regressions. Note that **allpossregs** treats the dummy variables for **rad** as separate variables, in practice we would include them all (model 18).

Calculate estimates of prediction error for your predictor using the R330 functions ***cross.val*** *and* ***err.boot****, as well as the functions* ***crossval*** *and* ***bootpred*** *in the bootstrap package.*

How accurate do you think your estimates are?

Let's calculate the CV and bootstrap estimates using the R330 functions. I have modified the **cross.val** function to return the CV estimates from all the random splits:

```
my.cross.val = function (f, nfold = 10, nrep = 10, ...){


# f:        an lm object returned by lm
# nfold:    the number of folds (e.g. 5 for 5-fold CV)
# nrep:     the number of random splits

# returns the CV estimates from each split


    X <- model.matrix(f$terms, model.frame(f))
    y = fitted.values(f) + residuals(f)
    n <- dim(X)[1]
    CV <- numeric(nrep)
    pred.error <- numeric(nfold)
    m <- n%/%nfold
    for (k in 1:nrep) {
        rand.order <- order(runif(n))
        yr <- y[rand.order]
        Xr <- X[rand.order, ]
        sample <- 1:m
        for (i in 1:nfold) {
              use.mat <- as.matrix(Xr[-sample,])
              test.mat <- as.matrix(Xr[sample,])
```

```
            y.use = yr[-sample]
            new.data <- data.frame(test.mat)
            fit <- lm(y.use ~ -1+use.mat)
            my.predict = test.mat%*%coefficients(fit)
            pred.error[i] <- sum((yr[sample] - my.predict)^2)/m
            sample <- if(i==nfold) (max(sample)+1):n else sample + m

          }
          CV[k] <- mean(pred.error)
      }
CV
}
```

We will do 100 random splits and check the consistency of the results:

```
cross.vals = my.cross.val(selectedModelBoth, nrep=100)
> mean(cross.vals)
[1] 0.1261296
> sd(cross.vals)
[1] 0.001914598
```

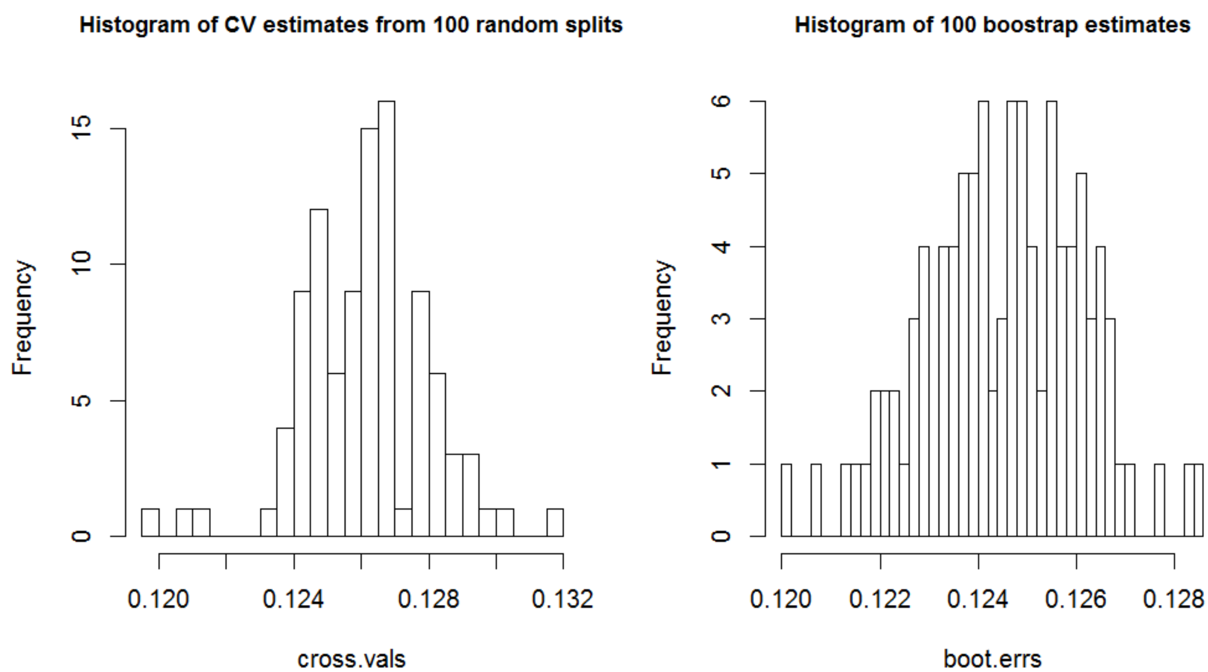
Estimate seems pretty accurate.

We can do the same for the bootstrap. Rather than rewrite the err.boot function we can put it in a loop:

```
boot.errs = numeric(100)
for(i in 1:100)boot.errs[i] = err.boot(selectedModelBoth, B=50)$Err

> mean(boot.errs)
[1] 0.1245297
> sd(boot.errs)
[1] 0.001635938
```

The bootstrap and cross-validation are giving similar results. Histograms of the 100 repeats are drawn by the code

```
par(mfrow=c(1,2))
hist(cross.vals, nclass=30, cex.main = 0.9,
   main = "Histogram of CV estimates from 100 random splits")
hist(boot.errs, nclass=30, cex.main = 0.9,
 main = "Histogram of 100 boostrap estimates")
```



Finally, we can also use the functions in the bootstrap package to get similar results.

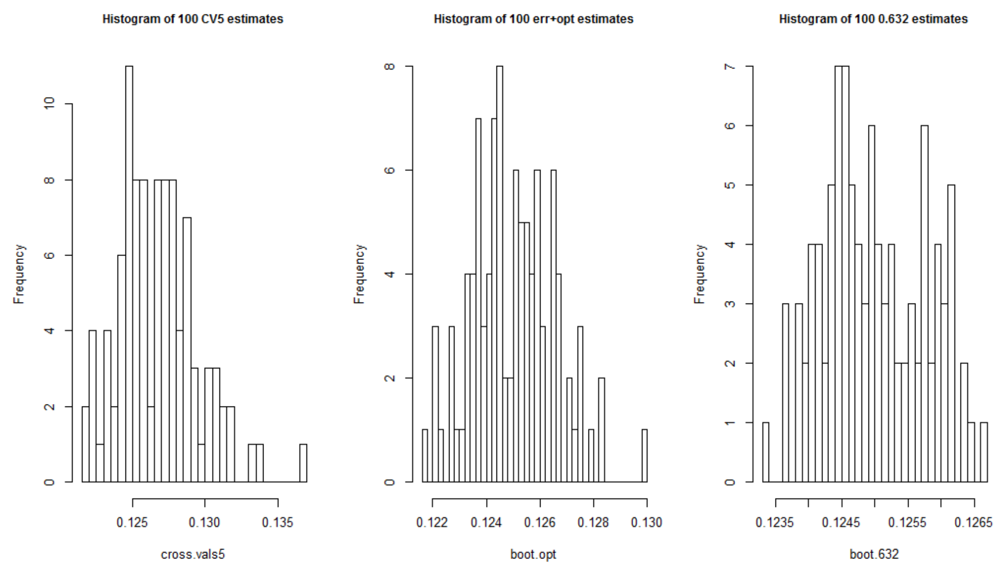
```
> library(bootstrap)

# define funcions theta.fit, theta predict and sq.err
# to fit the linear model,calculate the predictor
# and the error

> theta.fit <- function(x,y){
lsfit(x,y)
}
> theta.predict <- function(fit,x){
  cbind(1,x)%*%coef(fit)
}
> sq.err <- function(y,yhat) { (y-yhat)^2}

> x = model.matrix(selectedModelBoth)[,-1]
> y = transMedv

# note use of the function model.matrix to get the X-matrix
# convenient when  there are dummy variables)

> cross.vals5 = numeric(100)
> boot.opt = numeric(100)
> boot.632 = numeric(100)
```

```
for(i in 1:100){
  results.cv = crossval(x,y,theta.fit,theta.predict, ngroup=5)
  cross.vals5[i] = mean((y-results.cv$cv.fit)^2)
  results.boot = bootpred(x,y,nboot=50,theta.fit,theta.predict,
    err.meas=sq.err)
  boot.opt[i] = results.boot[[1]] + results.boot[[2]]
  boot.632[i] = results.boot[[3]]
}
```

```
> mean(cross.vals5)
[1] 0.1267213
> mean(boot.opt)
[1] 0.1249585
> mean(boot.632)
[1] 0.1249685
>
> sd(cross.vals5)
[1] 0.002756525
> sd(boot.opt)
[1] 0.001539274
> sd(boot.632)
[1] 0.0007646353
```

And histograms

```
par(mfrow=c(1,3))
hist(cross.vals5, nclass=30, cex.main = 0.9,
 main = "Histogram of 100 CV5 estimates")
hist(boot.opt, nclass=30, cex.main = 0.9,
 main = "Histogram of 100 err+opt estimates")
hist(boot.632, nclass=30, cex.main = 0.9,
 main = "Histogram of 100 0.632 estimates")
```



Summarizing, we get for the 100 separate estimates

Method	Mean	Std Dev
my.cross.val (10 fold)	0.126	1.9×10^{-3}
err.boot	0.125	1.6×10^{-3}
crossval (5 fold)	0.127	2.7×10^{-3}
err+opt	0.125	1.5×10^{-3}
0.632	0.125	0.7×10^{-3}

The methods are consistent, indicating a PE of about 0.125. The 0.632 estimate seems the most accurate (smallest sd). These results show the importance of using several random splits when using cross-validation.

Marking: I gave 20 points for each question.

For question 1: 5 points for identifying two applications, 5 points per application for a good description, 5 points for identifying the sources.

For question 2: 5 points for choosing a model, 5 points for usin g the R330 functions, 5 points for the bootstrap functions, 5 points for assessing the error.