High Diminsional Statistics-Sheet 4-Exercise 4

Hamed Vaheb

02/05/2022

Use the same prostate dataset as in the last sheet, which can be found at this clickable link

We want to perform the regularization method known as Adaptive Lasso, which seeks to minimize

$RSS(b) + \lambda \sum_{j=1}^{p} \hat{w}_j |b_j|$,

where $RSS(b)$ is the residual sum of square of b, λ is the tuning parameter (chosen through 10-fold cross validation), b_j are the p estimated coefficients and $\hat{w}_j$ are adaptive weights. We recall that

$$\hat{w}_j = \frac{1}{(|b_j^{in}|)^\gamma},$$

where b_j^{in} is an initial estimate of the coefficients and γ is a positive constant for adjustment of the adaptive weights.

Prepare

First of all we import the dataset:

```
url <- "https://hastie.su.domains/ElemStatLearn/datasets/prostate.data"
df <- read.table(url, sep = '\t', header = TRUE)
df %>% head(10)
```

```
##     X     lcavol  lweight age       lbph svi       lcp gleason pgg45       lpsa
## 1   1 -0.5798185 2.769459  50 -1.3862944   0 -1.386294       6     0 -0.4307829
## 2   2 -0.9942523 3.319626  58 -1.3862944   0 -1.386294       6     0 -0.1625189
## 3   3 -0.5108256 2.691243  74 -1.3862944   0 -1.386294       7    20 -0.1625189
## 4   4 -1.2039728 3.282789  58 -1.3862944   0 -1.386294       6     0 -0.1625189
## 5   5  0.7514161 3.432373  62 -1.3862944   0 -1.386294       6     0  0.3715636
## 6   6 -1.0498221 3.228826  50 -1.3862944   0 -1.386294       6     0  0.7654678
## 7   7  0.7371641 3.473518  64  0.6151856   0 -1.386294       6     0  0.7654678
## 8   8  0.6931472 3.539509  58  1.5368672   0 -1.386294       6     0  0.8544153
## 9   9 -0.7765288 3.539509  47 -1.3862944   0 -1.386294       6     0  1.0473190
## 10 10  0.2231436 3.244544  63 -1.3862944   0 -1.386294       6     0  1.0473190
##    train
## 1   TRUE
## 2   TRUE
## 3   TRUE
## 4   TRUE
## 5   TRUE
## 6   TRUE
## 7  FALSE
## 8   TRUE
## 9  FALSE
## 10 FALSE
```

Question 1

Build the regression model for the variable prostate antigen (lpsa. Use it to obtain the initial estimates of the coefficients b_j^{in}.

At first we build the regression model:

```
##
## Call:
## lm(formula = df$lpsa ~ ., data = features)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.76644 -0.35510 -0.00328  0.38087  1.55770
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.181561   1.320568   0.137  0.89096
## lcavol       0.564341   0.087833   6.425 6.55e-09 ***
## lweight      0.622020   0.200897   3.096  0.00263 **
## age         -0.021248   0.011084  -1.917  0.05848 .
## lbph         0.096713   0.057913   1.670  0.09848 .
## svi          0.761673   0.241176   3.158  0.00218 **
## lcp         -0.106051   0.089868  -1.180  0.24115
## gleason      0.049228   0.155341   0.317  0.75207
## pgg45        0.004458   0.004365   1.021  0.31000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6995 on 88 degrees of freedom
## Multiple R-squared:  0.6634, Adjusted R-squared:  0.6328
## F-statistic: 21.68 on 8 and 88 DF,  p-value: < 2.2e-16
```

The estimators of b_0 and $b_j \in \{1, ..., 8\}$ are respectively:

```r
coef(model_lm)
```

```
##  (Intercept)       lcavol      lweight          age         lbph          svi
##  0.181560845  0.564341280  0.622019788 -0.021248185  0.096712522  0.761673402
##          lcp      gleason        pgg45
## -0.106050939  0.049227934  0.004457512
```

Perform lasso regression with 10-fold cross validation to find the best λ, from which we will create the initial estimates the coefficients

```r
X <- data.matrix(features)
y <- df$lpsa

cv_lasso <- cv.glmnet(X, y,
                      ## type.measure: loss to use for cross-validation.
                      ## K = 10 is the default.
                      nfold = 10,
                      ## ‘alpha = 1’ is the lasso penalty, and ‘alpha = 0’ the ridge penalty.
                      alpha = 1)
## Penalty vs CV MSE plot
```

Store the coefficients of the lasso model in which we set λ to be `lambda.min` which is λ at which the smallest mean squared error (MSE) is achieved.

```
## s: Value(s) of the penalty parameter ‘lambda’ at which
##    predictions are required. Default is the entire sequence used
##    to create the model.
lasso_coef <- coef(cv_lasso, s = cv_lasso$lambda.min)
```

Question 2

Create the Adaptive Weights $\hat{w_j}$ for $\gamma = 0.5, 1$ and 2. Choose the best λ through 10-fold cross validation.

The procedure I pursue is as following:

1. Using initial estimates of b_j^{in} derived in Qeustion 1, I Construct w (adaptive weights) by adjusting the `lasso_coef` using 3 three different choices for γ. For this purpose, I define a function `weight_func` which constructs adaptive weights given γ and b, and I make sure it doesn't contain ∞ as a value.
2. We define the function `alasso_cv_gamma` which takes γ as its only argument and builds a 10-fold cross-validation adaptive lasso model using adaptive weights that are created based on γ.
3. We compare the three models defines in step 2 in order to find the γ that leads to the "best" model, i.e., the model that has the λ that leads to the least mse loss.
4. As the ultimate goal is to find the best λ, I create my final cross-validation model with the best γ, and output the best λ. Moreover, I report the coefficients' estimates.
5. For the purpose of illustration, I plot values of coefficients for two model variations, i.e., simple lasso and adaptive lasso. Finally, I plot the lambda choices of the cross-validation model.

Step 1: Construct Adaptive Weights

```
#best_lasso_coef <- as.numeric(coef(cv_lasso, s = #cv_lasso$lambda.min))[-1]

## constructing w which consists of adaptive weights
## The intercept estimate should be dropped.
p <- nrow(lasso_coef)-1
b <- lasso_coef[1:p]
weight_func <- function(x,gamma){
  1/(abs(x)**gamma)
}
gamma <- c(1/2,1,2)
adap_weights <- sapply(b, weight_func,gamma[2])
adap_weights
```

```
## [1]  5.449856  1.819027  1.642293 53.076800 11.041996  1.389725 13.067723
## [8] 24.777622
```

```
# Replacing values estimated as Infinite for 999999999
adap_weights[adap_weights == Inf] <- 999999999
```

Step 2: Define the function alasso_cv_gamma

```
alasso_cv_gamma <- function(gamma) {
    adap_weights <- sapply(b, weight_func, gamma)
    adap_weights[adap_weights == Inf] <- 999999999
    alasso_cv <- cv.glmnet(X,y,nfold = 10,alpha = 1,penalty.factor = adap_weights,keep = TRUE)
    y_pred <- predict(alasso_cv, newx = X, s = alasso_cv$lambda.min)
    return(y_pred)
}
```

Step 3: Find the best gamma

```
## create a dictionary with keys as gamma and values as MSE
dict = c()
keys = c()
for (g in gamma)
{
  keys <- append(keys, as.numeric(g))
  y_pred <- alasso_cv_gamma(g)
  mse <- mean((y_pred - y)^2)
  dict[sprintf("%f",as.numeric(g))] <- as.numeric(mse)

}

gamma_best <- keys[which.min(dict)]
dict
```

```
##  0.500000  1.000000  2.000000
## 0.4439530 0.4714817 0.4656138
```

Step 4: Report the best labmda and coefficients' estimates

```
adap_weights <- sapply(b, weight_func, gamma_best)
adap_weights[adap_weights == Inf] <- 999999999
alasso_cv_best <- cv.glmnet(X,y, nfold = 10, alpha = 1, penalty.factor = adap_weights, keep = TRUE)
y_pred <- predict(alasso_cv_best, newx = X, s = alasso_cv_best$lambda.min)



#obtain the minimum lambda
alasso_cv_best$lambda.min
```

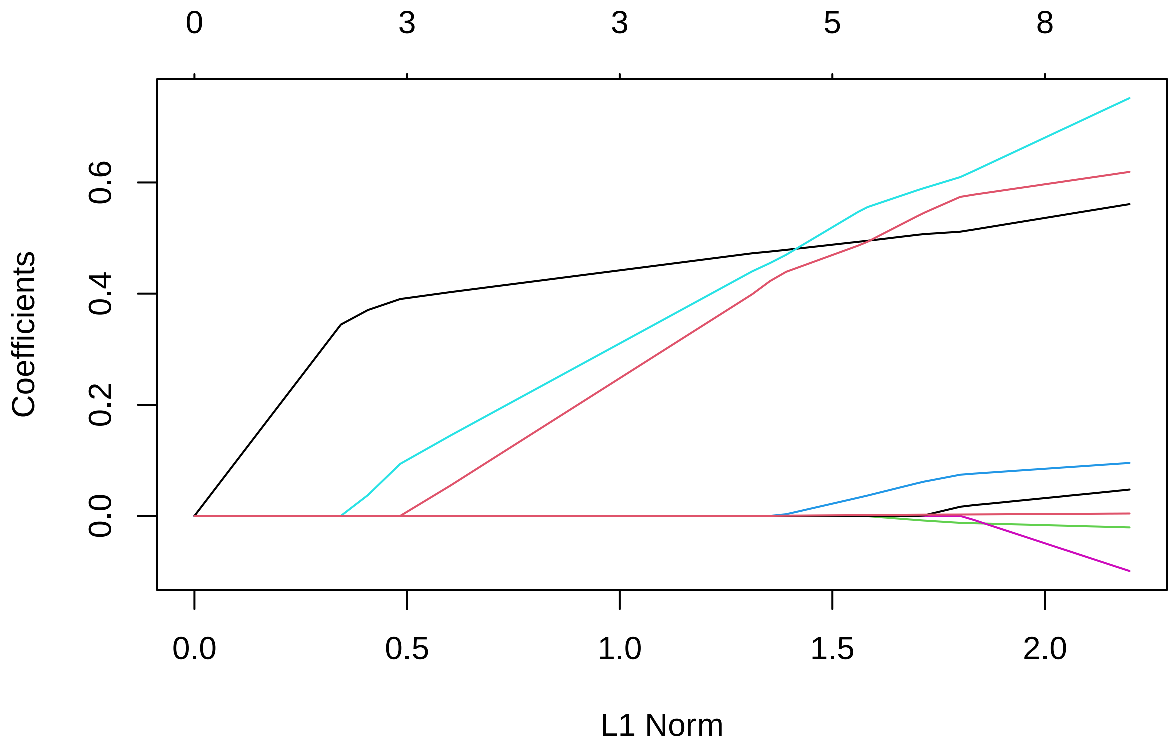
```
## [1] 0.004823694
```

```
#obtain the corresponding coefficients
coef(alasso_cv_best, s = alasso_cv_best$lambda.min)
```

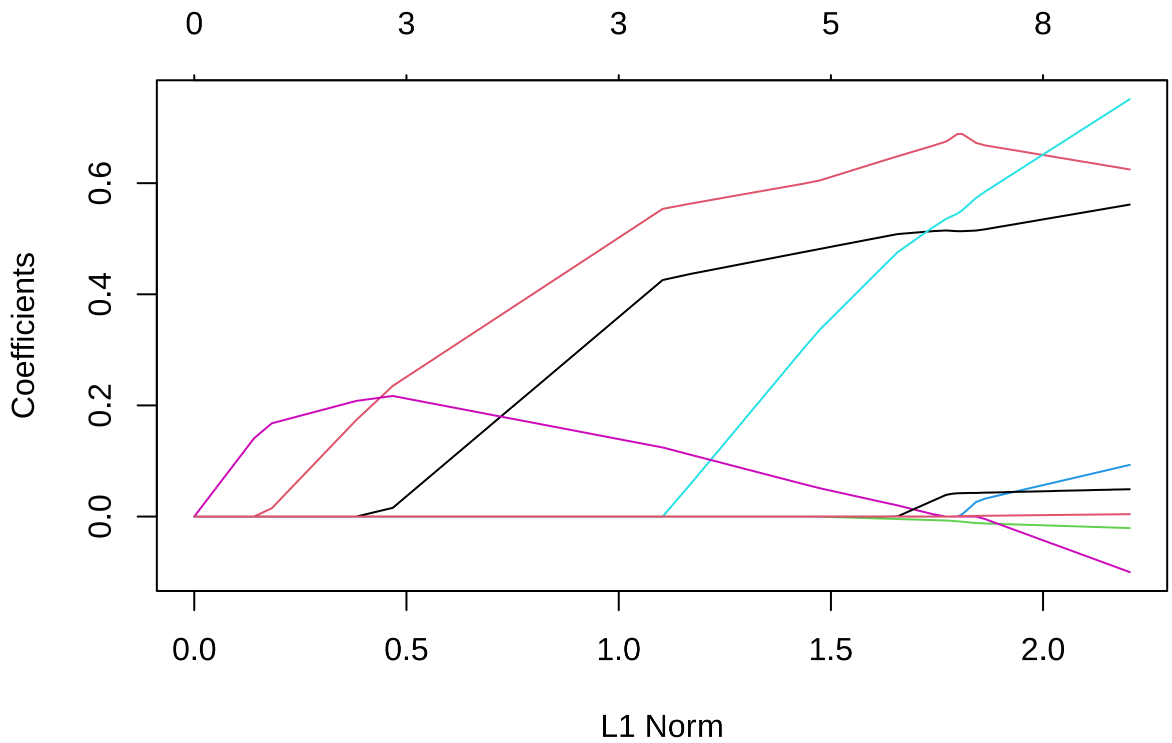
```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.096575102
## lcavol       0.556091641
## lweight      0.629922897
## age         -0.019741987
## lbph         0.085687021
## svi          0.731214190
## lcp         -0.088494073
## gleason      0.048489142
## pgg45        0.003927875
```

Step 5: Plotting the estimations of the coefficients for all models

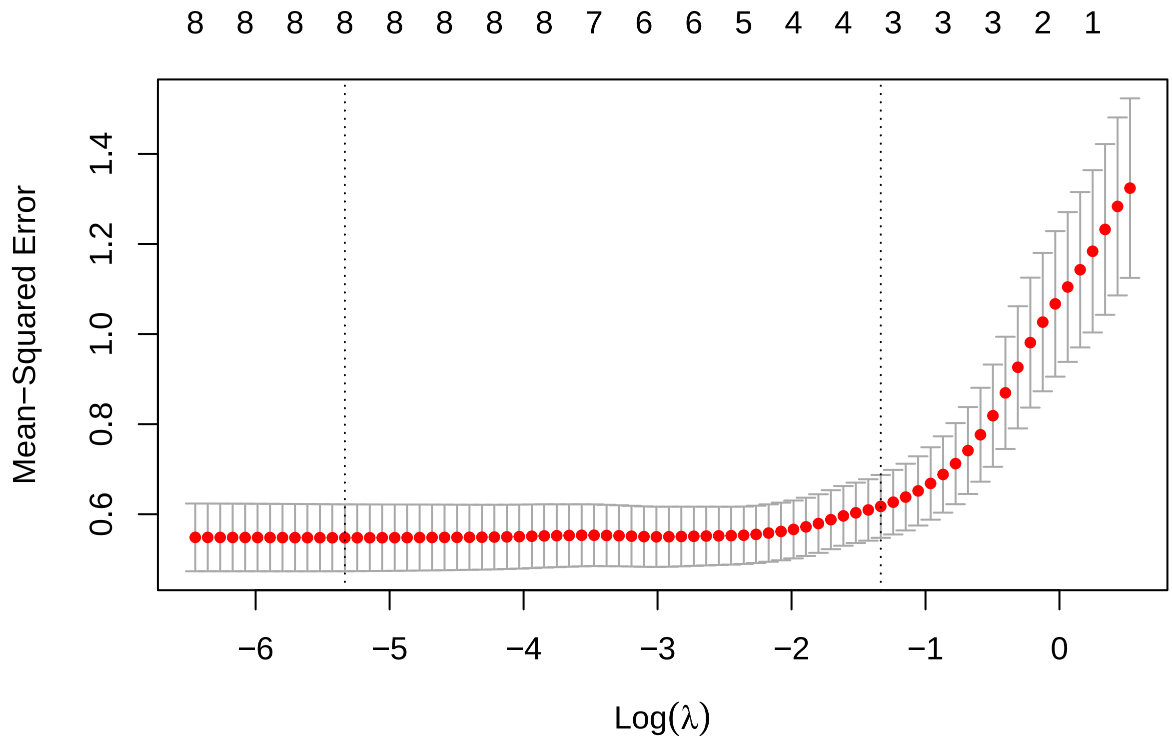
```
model_lasso <- glmnet(features, df$lpsa, alpha = 1)
alasso <- glmnet(features, df$lpsa, alpha = 1, penalty.factor = adap_weights)
plot(model_lasso)
```




```
plot(alasso)
```




```
## Penalty vs CV MSE plot
plot(alasso_cv_best)
```



Question 3:

Use the glmnet function to execute the adaptive Lasso. Plot the area under ROC (receiver operating characteristic) curve, also called AUC, and report the values of minimum λ (obtained for miminum AUC).

We discretize the target variable (ibsa), i.e., convert it from continous type to discrete one so as to use the metric AUC (since it is a classification metric). We do the same for the predictions. THen, we can compare them using AUC metric. Finally, we plot the ROC-AUC curve.

Step 1: Descretize the target variable (ibsa)

```
y_disc <- as.integer(discretize(y, breaks = 2, labels=c(0, 1)))
y_pred_disc <- as.integer(discretize(y_pred, breaks = 2, labels=c(0, 1)))
```

Step 2: Calculate AUC

```
## Extract predicted probabilities and observed outcomes.
## pROC for ROC construction
roc <- pROC::roc(y_disc ~ y_pred_disc)
```

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```
auc <- auc(y_disc, y_pred)
```

```
## Setting levels: control = 1, case = 2
```

```
## Warning in roc.default(response, predictor, auc = TRUE, ...): Deprecated use a
## matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.
```

```
## Setting direction: controls < cases
```

```r
auc
```

```
## Area under the curve: 0.8971
```

Step 3: Plot ROC-AUC Curve

```r
## Plot an ROC curve with AUC and threshold
plot(roc, print.auc = TRUE, print.thres = TRUE, print.thres.best.method = "youden")
```