STAT636 - Homework 5

Daniel Osorio - dcosorioh@tamu.edu
Department of Veterinary Integrative Biosciences
Texas A&M University

1. Consider the `Life_Expectancy` data. Let the response Y be `Life.expectancy`, and consider the numeric variables `Alcohol`, `percentage.expenditure`, `Total.expenditure`, `GDP`, `Income.composition.of.resources`, and `Schooling` as the predictor variables; call these x_1, x_2, ..., x_6, respectively. For the purpose of predicting the value of Y , we will use linear regression models of the form

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_6 x_{6i} + \epsilon_i$$

Where ϵ_i are IID $\mathcal{N}(0, \sigma^2)$, $i = 1, 2, \ldots, n$

```
> lifeExpectancy <- read.csv("Life_Expectancy.csv")
> lifeExpectancy <- lifeExpectancy[,c(
+   "Life.expectancy", "Alcohol", "percentage.expenditure",
+   "Total.expenditure", "GDP", "Income.composition.of.resources",
+   "Schooling")]
> colnames(lifeExpectancy) <- c("Y", paste0("X",1:6))
> lifeExpectancy <- lifeExpectancy[complete.cases(lifeExpectancy),]
> N <- nrow(lifeExpectancy)
```

   (a) Using a usual least squares linear regression model (the `lm` function in R) (Using `na.omit` function to remove the row that there are some missing data.):

      i. Use leave-one-out cross-validation to estimate the MSE of your model.

```
> SE <- sapply(seq_len(N), function(x){
+   model <- lm(formula = Y~.,
+               data = lifeExpectancy[-x,])
+   Yhat <- predict(model,lifeExpectancy[x,2:7, drop=FALSE])
+   return((lifeExpectancy[x,1] - Yhat)^2)
+ })
> mean(SE)
[1] 26.58507
```

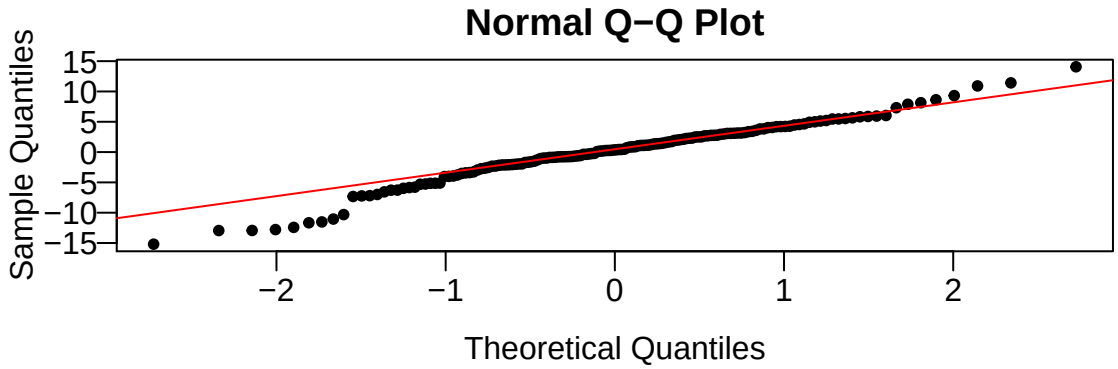
      ii. Use bootstrap (1000 times) to estimate the standard deviation of your MSE estimate. (`set.seed(2)` before sampling)

```
> set.seed(2)
> b <- sapply(seq_len(1000), function(x){
+   mean(sample(SE, replace = TRUE))
+ })
> sd(b)
[1] 3.736676
```

   (b) Make a diagnostics to check assumptions of the linear regression model in number 1. Specifically, please:

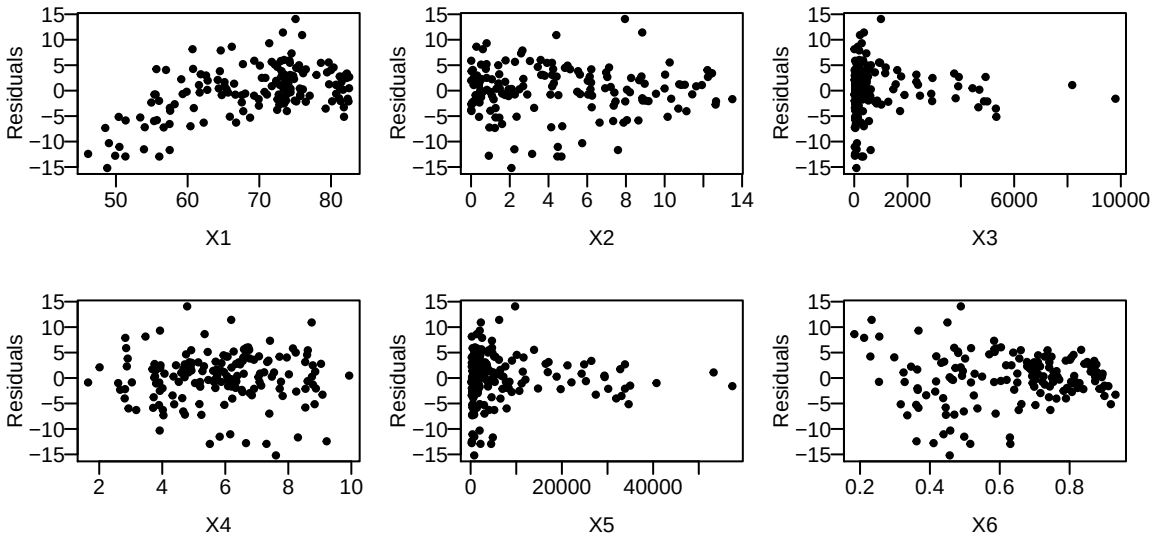
i. Make Normal QQ plot of residuals. Does the residuals appear normally distributed? *Yes, residuals appears to be normally distributed*

```
> model <- lm(formula = Y~., data = lifeExpectancy)
> par(mar=c(3,3,2,1), mgp=c(2,0.5,0))
> qqnorm(residuals(model), pch = 20, las=1)
> qqline(residuals(model), col = "red")
```



ii. Create scatterplots for each covariate, with the covariate on x-axis and residuals on y-axis. Do you see any problematic patterns? *Yes, residuals are positively correlated with x_1, and variability is not equal across all the range in x_3 and x_5*

```
> par(mfrow=c(2,3),mar=c(3,3,2,1), mgp=c(2,0.5,0))
> scatterPlots <- sapply(1:6, function(x){
+   plot(y = residuals(model),
+        x = lifeExpectancy[,x],
+        xlab = paste0("X",x),
+        ylab = "Residuals",
+        las =1,
+        pch=20)
+ })
```



(c) Using regularized (lasso-based) linear regression (the `glmnet` and `cv.glmnet` functions from `glmnet` package in `R`, with `family='gaussian'` and `alpha = 1`):

i. Based on cross-validation, using the `cv.glmnet` function, what is the optimal value of the tuning parameter `lambda`?

```
> X <- as.matrix(lifeExpectancy[,2:7])
> Y <- lifeExpectancy[,1]
> fittedModel <- glmnet::cv.glmnet(x = X, y = Y,
```

```
+                                        family = "gaussian",
+                                        alpha = 1)
> print(lambda <- fittedModel$lambda.min)
[1] 0.2007096
```

ii. Use leave-one-out cross-validation (code it up yourself) and glmnet to estimate the MSE of the lasso model using the optimal tuning parameter.

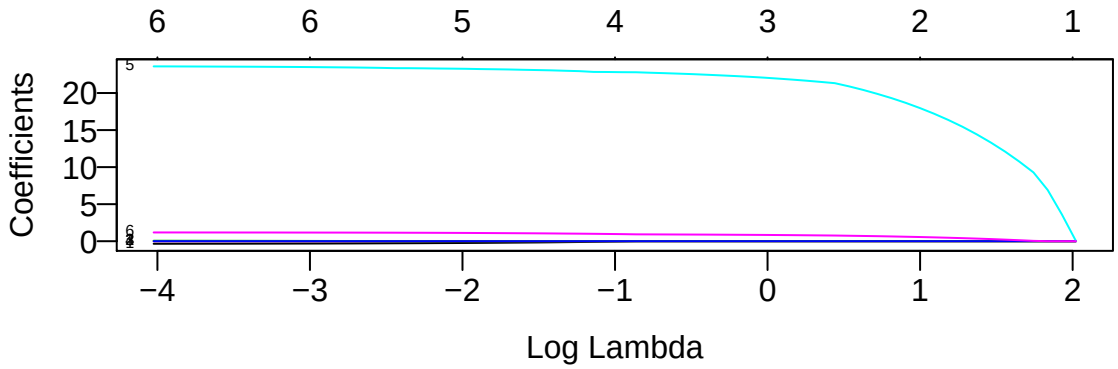
```
> SE <- sapply(seq_len(N), function(z){
+   fittedModel <- glmnet::glmnet(x = X[-z,], y = Y[-z],
+                                 family = "gaussian",
+                                 alpha = 1, lambda = lambda)
+   Yhat <- predict(fittedModel,newx = X[z,,drop=FALSE])
+   return((Y[z] - Yhat) ^ 2)
+ })
> mean(SE)
[1] 26.34441
```

iii. Use bootstrap (again, code yourself, 1000 times) to estimate the standard deviation of your MSE estimate.

```
> b <- sapply(seq_len(1000), function(x){
+   mean(sample(SE, replace = TRUE))
+ })
> sd(b)
[1] 3.536297
```

iv. Compare the estimated β coefficients from the lasso model (using λ you gotten from (i)) to the least-squares model, and confirm that the lasso model's coefficient estimates have been 'shrunken' toward 0.

```
> as.numeric(coef(lm(formula = Y~.,data = lifeExpectancy)))
[1]  4.057963e+01 -3.649839e-01  4.582845e-04  7.942828e-02  3.968866e-05
[6]  2.363990e+01  1.190499e+00
> as.numeric(coef(glmnet::glmnet(x = X,y = Y,family = "gaussian",
+                                alpha = 1, lambda = lambda)))
[1]  4.207565e+01 -1.869143e-01  2.382802e-04  0.000000e+00  4.923415e-05
[6]  2.312898e+01  1.074836e+00
> fittedModel <- glmnet::glmnet(x = X,y = Y,family = "gaussian", alpha = 1)
> par(mar=c(3,3,2,1), mgp=c(2,0.5,0))
> plot(fittedModel, xvar = "lambda", label = TRUE, las=1)
```



2. Consider the `HOF` data. Let the response Y be the indicator for whether players are in the Hall of Fame (1 for "yes", 0 for "no"), and consider the numerical variables `H`, `HR`, and `AVG` as the predictor

variables (using `na.omit` before your job); call these x_1,x_2,x_3, respectively. Randomly split the $n = 1780$ training data sample observations into 2/3 for training and 1/3 for testing (Select 2/3 from data whose HOF is 1 and also 2/3 from data whose HOF is 0 and combine these two. Use `set.seed(2)` before sampling).
Let p_i be the probability player i is in the Hall of Fame, conditional on that player's predictor variable values:

$$p_i = Pr(Y_i = 1|x_{1i}, x_{2i}, x_{3i})$$

and consider the logistic regression model

$$log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta 3 x_{3i}$$

for $i = 1, 2, \ldots, n$. After fitting the model, obtaining parameter estimates $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$, we will predict Hall of Fame status for an individual with predictor variable values of x_1^*, x_2^*, x_3^* based on his estimated probability $\hat{p}$, where

$$\hat{p} = \frac{exp\left(\hat{\beta}_0 + \hat{\beta}_1 x_1^* + \hat{\beta}_2 x_2^* + \hat{\beta}_3 x_3^*\right)}{1 + exp\left(\hat{\beta}_0 + \hat{\beta}_1 x_1^* + \hat{\beta}_2 x_2^* + \hat{\beta}_3 x_3^*\right)}$$

```
> set.seed(2)
> HOF <- read.csv("hof_data.csv")
> X <- HOF[,c("HOF","H", "HR", "AVG")]
> X[,1] <- as.numeric(X[,1])-1
> Y1 <- which(X[,1] == 1)
> Y0 <- which(X[,1] == 0)
> randomSelection <- c(
+   sample(x = Y1,size = round(length(Y1)*2/3)),
+   sample(x = Y0,size = round(length(Y0)*2/3)))
> Training <- X[randomSelection,]
> Testing <- X[-randomSelection,]
```

Specifically, we will predict that $Y = 1$ if $\hat{p} > k$, for some choice of $k \in [0, 1]$. Fit the model to the training data:

```
> fittedModel <- glm(HOF~.,data=Training, family = "binomial")
```

(a) Using the default choice of $k = 0.5$:

i. Report the misclassification rate, sensitivity, and specificity of your model when applied to the training data.

```
> hat <- predict(object = fittedModel, Training[,2:4], type = "response")
> hat <- ifelse(test = hat > 0.5,yes = 1, no = 0)
> print(ConfussionMatrix <- table(Observed=Training[,1],Predicted=hat))
```

```
        Predicted
Observed   0   1
       0 638   7
       1  14  19
```

```
> # Misclassification rate
> ConfussionMatrix[2,1]/sum(ConfussionMatrix[2,])
```

```
[1] 0.4242424
```

```
> # Sensitivity (True Positive Rate)
> ConfussionMatrix[2,2]/sum(ConfussionMatrix[2,])
```

```
[1] 0.5757576
> # Specificity (True Negative Rate)
> ConfussionMatrix[1,1]/sum(ConfussionMatrix[1,])
[1] 0.9891473
```

ii. Report the misclassification rate, sensitivity, and specificity of your model when applied to the test data. Comment of the relationship between the performance measures, testing compared to training. *Missclassification rate (1-TPR): measures the proportion of positives which yield negative test outcomes with the test; Sensitivity (TPR): measures the proportion of actual positives that are correctly identified as such; Specificity (TNR): measures the proportion of actual negatives that are correctly identified as such.*

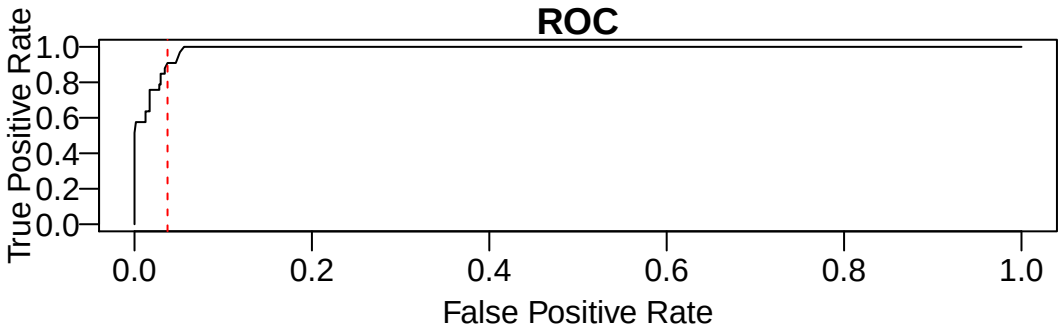
```
> hat <- predict(object = fittedModel, Testing[,2:4], type = "response")
> hat <- ifelse(test = hat > 0.5,yes = 1, no = 0)
> print(ConfussionMatrix <- table(Observed=Testing[,1],Predicted=hat))
          Predicted
Observed   0   1
       0 319   3
       1   8   8
> # Misclassification rate
> ConfussionMatrix[2,1]/sum(ConfussionMatrix[2,])
[1] 0.5
> # Sensitivity (True Positive Rate)
> ConfussionMatrix[2,2]/sum(ConfussionMatrix[2,])
[1] 0.5
> # Specificity (True Negative Rate)
> ConfussionMatrix[1,1]/sum(ConfussionMatrix[1,])
[1] 0.9906832
```

(b) Now use leave-one-out (LOO) cross validation (CV) to 'tune' the model with respect to k on the training data, using misclassification rate as the guiding performance measure.

```
> N <- nrow(Training)
> LOO <- sapply(seq_len(N), function(x){
+   fittedModel <- glm(HOF~.,data=Training[-x,], family = "binomial")
+   predict(object = fittedModel, Training[x,2:4], type = "response")
+ })
> K <- sapply(seq(0,1,0.01), function(x){
+   as.numeric(LOO > x)
+ })
> ROC <- t(apply(K,2,function(hat){
+   ConfussionMatrix <- table(Observed=Training[,1],
+                             Predicted=factor(hat, levels = c(0,1)))
+   c(ConfussionMatrix[1,2]/sum(ConfussionMatrix[1,]),
+     ConfussionMatrix[2,2]/sum(ConfussionMatrix[2,]))
+ }))
```

i. Report an ROC curve.

```
> par(mar=c(4,4,1,1), mgp=c(1.6,0.5,0))
> plot(ROC, type = "l", ylab = "True Positive Rate",
+      xlab = "False Positive Rate", main = "ROC", las = 1)
> optimalK <- which.max((order(ROC[,1],decreasing = FALSE) +
+                          order(ROC[,2], decreasing = FALSE))/2)
> abline(v=ROC[optimalK,1], col="red", lty = 2)
```



ii. What is the optimal choice of k, and what are the CV-based estimates of misclassification rate, sensitivity, and specificity that correspond to this choice of k?

```
> # Optimal K
> seq(0,1,0.01)[optimalK]
[1] 0.12
> fittedModel <- glm(HOF~.,data=Training, family = "binomial")
> hat <- predict(object = fittedModel, Testing[,2:4], type = "response")
> hat <- ifelse(hat > seq(0,1,0.01)[optimalK], 1,0)
> print(ConfussionMatrix <- table(Observed=Training[,1],
+                                 Predicted=K[,optimalK]))
        Predicted
Observed   0   1
       0 621  24
       1   3  30
> # Misclassification rate
> ConfussionMatrix[2,1]/sum(ConfussionMatrix[2,])
[1] 0.09090909
> # Sensitivity (True Positive Rate)
> ConfussionMatrix[2,2]/sum(ConfussionMatrix[2,])
[1] 0.9090909
> # Specificity (True Negative Rate)
> ConfussionMatrix[1,1]/sum(ConfussionMatrix[1,])
[1] 0.9627907
```

(c) Now we will perform the lasson on the training data.

i. Use cross-validation to choose the tuning parameter λ.

```
> X <- as.matrix(Training[,2:4])
> Y <- Training[,1]
> fittedModel <- glmnet::cv.glmnet(x = X, y = Y, family = "binomial")
> lambda <- fittedModel$lambda.min
```

ii. Fit a lasso regression model on the training set and computing the coefficients using λ above.

```
> fittedModel <- glmnet::glmnet(X,Y,family = "binomial", lambda = lambda)
> coef(fittedModel)
4 x 1 sparse Matrix of class "dgCMatrix"
                       s0
(Intercept) -13.144083522
H             0.004651503
HR            0.004065602
AVG           .
```

iii. Evaluate its misclassification rate, sensitivity and specificity on the training data set using λ which you get from above.

```
> hat <- round(predict(fittedModel, newx = X, type = "response"))
> print(ConfussionMatrix <- table(Observed=Training[,1], Predicted=hat))
        Predicted
Observed   0   1
       0 639   6
       1  14  19
> # Misclassification rate
> ConfussionMatrix[2,1]/sum(ConfussionMatrix[2,])
[1] 0.4242424
> # Sensitivity (True Positive Rate)
> ConfussionMatrix[2,2]/sum(ConfussionMatrix[2,])
[1] 0.5757576
> # Specificity (True Negative Rate)
> ConfussionMatrix[1,1]/sum(ConfussionMatrix[1,])
[1] 0.9906977
```