Lab 11

Model Selection and Prediction

In this lab, we will run a simulation to see if forward stepwise AIC finds the "true" model of a fake dataset that we generate. Then we will use two other methods for model selection, which focus on out-of-sample prediction, rather than likelihood or variance explained in Y. The measures we will use to compare the models with respect to prediction are the root mean square error (RMSE) from a 10-fold cross-validation and the predicted root mean square error (PRMSE) from a validation set that we will subset from our generated data before running potential models.

First, we will generate some fake data and store it in a matrix.

Initial setup

We are going to generate 500 observations of 8 X variables and one Y variable, so we'll start by creating an empty matrix with 500 rows and 9 columns.

```r
# Load required libraries: MASS for stepAIC; caret for train and trainControl
library(MASS)
library(caret)

# Set the seed because we are generating random data
set.seed(123)

# Set number of observations
n <- 500

# Create a 500x9 matrix filled with zeros
# The rows will be observations and the columns will be variables (8 X's and 1 Y)
data <- matrix(data = 0, nrow = n, ncol = 9)
```

Generating the X variables

Now that we have a place to store our data, we can generate each variable. All of our independent variables (X's) will come from a normal distribution with mean 1 and standard deviation 2.

```r
# Loop through the numbers 1 through 8
# In each iteration, draw 500 values from a normal distribution of mean 1 and standard deviation 2
# Store these values in the empty matrix in the column corresponding the iteration number
for (j in 1:8) {
  data[ , j] <- rnorm(n = n, mean = 1, sd = 2)
}

# Change the matrix into a data frame
data <- data.frame(data)

# Rename the X variables X1-X8, and the last column Y (still just zeros)
# The paste0() function puts together "X" and one of the numbers, with no space separating
# So "X" and "1" become "X1", etc.
names(data) <- c(paste0("X", 1:8), "Y")
```

Generate the outcome variable, Y

Now that we have 8 independent variables, we can generate the outcome variable Y. Since we are creating this data, we know the true relationship of the X variables to Y–we are the ones deciding that relationship!

There are two things we want to accomplish here:
(1) We want Y to have some kind of linear relationship with some (but maybe not all) of the X's.
(2) We want Y to look a little messy, like real data does. Most of the time data do not show a perfect linear relationship. There's "noise" in the data caused by random error.

To accomplish these goals we will first generate Y as a linear combination of the first 4 X variables (exclude X5, X6, X7, and X8). Then we add "noise" by adding random error to Y, drawn from a normal distribution of mean 0 and standard deviation 1.5.

```
# Generate the values for the outcome variable Y using some of the X variables
# This means we are setting a linear relationship between some of the X variables with the outcome Y
data$Y <- 3 + 2*data$X1 + 1.5*data$X2 + -1*data$X3 + -1.5*data$X4

# Add some "noise" to the Y variable because real data doesn't have a perfect relationship
data$Y <- data$Y + rnorm(n = n, mean = 0, sd = 1.5)
```

Run regression of "true" formula and stepwise AIC

Next, we run a regression using the true formula: Y regressed on the X variables 1-4. These were the variables used to generate Y, so we know they are the only ones that really belong in the regression line equation. We can compare this to a model with all of the covariates:

```
# Regression using "true" formula
true_model <- lm(Y ~ X1 + X2 + X3 + X4, data = data)
summary(true_model)
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2 + X3 + X4, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2142 -1.0867  0.0128  0.9408  5.1076
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.76951    0.09847   28.13   <2e-16 ***
## X1           2.03772    0.03454   59.00   <2e-16 ***
## X2           1.52495    0.03342   45.62   <2e-16 ***
## X3          -0.91580    0.03425  -26.74   <2e-16 ***
## X4          -1.49422    0.03328  -44.90   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.494 on 495 degrees of freedom
## Multiple R-squared:  0.9369, Adjusted R-squared:  0.9364
## F-statistic:  1838 on 4 and 495 DF,  p-value: < 2.2e-16
```

```
# Regression using the full model (all variables)
full_model <- lm(Y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8, data = data)
summary(full_model)
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1549 -1.0528  0.0460  0.9337  5.1024
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.695543   0.118631  22.722   <2e-16 ***
## X1           2.035220   0.034727  58.606   <2e-16 ***
## X2           1.527005   0.033585  45.467   <2e-16 ***
## X3          -0.912913   0.034509 -26.454   <2e-16 ***
## X4          -1.494173   0.033393 -44.746   <2e-16 ***
## X5           0.018943   0.035902   0.528    0.598
## X6          -0.009064   0.033398  -0.271    0.786
## X7           0.026573   0.033975   0.782    0.435
## X8           0.038783   0.033815   1.147    0.252
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.497 on 491 degrees of freedom
## Multiple R-squared:  0.9372, Adjusted R-squared:  0.9362
## F-statistic:   916 on 8 and 491 DF,  p-value: < 2.2e-16
```

Now, we will use forward stepwise AIC to fit a model on the data. Here we are pretending we do not know the true formula for modeling Y. When the direction is set to forward, stepAIC starts with an intercept only model, then it tries out all models of just one variable (8), and it keeps the one with the lowest AIC (provided it is lower than the intercept-only model). From there it tests adding a second variable to the model and picks the second variable that most lowers the AIC. This continues until adding variables no longer improves the model.

Backward stepwise AIC uses the same idea, but starts with the full model (all variables) and deletes them one at a time rather than adding them in. Using one direction or the other can produce different models as the "best" fit.

NOTE: For backward stepwise AIC, we would not need the scope argument; just enter the full model and set the direction to "backward."

```r
# Intercept-only model
null_model <- lm(Y ~ 1, data = data)

# Forward stepwise AIC
AIC_model <- stepAIC(null_model,
                     scope = list(upper = full_model, lower = null_model),
                     direction = "forward")
```

```
## Start:  AIC=1780.4
## Y ~ 1
##
##        Df Sum of Sq     RSS    AIC
## + X1    1    7560.7  9964.7 1500.1
## + X4    1    3597.7 13927.7 1667.5
## + X2    1    2965.7 14559.7 1689.7
## + X3    1     444.8 17080.6 1769.5
## + X6    1     158.9 17366.5 1777.8
## <none>              17525.4 1780.4
## + X7    1      28.4 17497.0 1781.6
## + X5    1      17.8 17507.6 1781.9
## + X8    1       5.7 17519.7 1782.2
##
## Step:  AIC=1500.1
## Y ~ X1
##
##        Df Sum of Sq    RSS    AIC
## + X2    1    3487.4 6477.3 1286.7
## + X4    1    2977.9 6986.8 1324.6
## + X3    1     688.4 9276.4 1466.3
## + X6    1      61.1 9903.6 1499.0
## <none>              9964.7 1500.1
## + X5    1       6.5 9958.3 1501.8
## + X7    1       3.6 9961.2 1501.9
## + X8    1       0.1 9964.7 1502.1
##
## Step:  AIC=1286.73
## Y ~ X1 + X2
##
##        Df Sum of Sq    RSS     AIC
## + X4    1    3775.2 2702.1  851.59
## + X3    1     868.9 5608.5 1216.71
## + X6    1      43.5 6433.8 1285.36
## <none>              6477.3 1286.73
## + X8    1       5.0 6472.4 1288.34
## + X7    1       2.2 6475.1 1288.56
## + X5    1       0.6 6476.8 1288.69
##
## Step:  AIC=851.59
## Y ~ X1 + X2 + X4
##
##        Df Sum of Sq    RSS    AIC
## + X3    1   1596.62 1105.5 406.72
## + X6    1     15.49 2686.6 850.72
## <none>              2702.1 851.59
## + X8    1     10.15 2692.0 851.71
## + X7    1      4.89 2697.2 852.68
## + X5    1      0.18 2701.9 853.56
##
## Step:  AIC=406.72
## Y ~ X1 + X2 + X4 + X3
##
##        Df Sum of Sq    RSS    AIC
## <none>              1105.5 406.72
## + X8    1   2.80429 1102.7 407.45
## + X7    1   1.43318 1104.1 408.07
## + X5    1   0.59994 1104.9 408.45
## + X6    1   0.07836 1105.4 408.68
```

The output above shows the process of the forward stepwise AIC. We can prevent this from printing by setting trace = FALSE as an argument in stepAIC(). We can check the final decision by printing AIC_model. As we can see below, stepAIC chose the correct model.

```
AIC_model
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2 + X4 + X3, data = data)
##
## Coefficients:
## (Intercept)           X1           X2           X4           X3
##      2.7695       2.0377       1.5249      -1.4942      -0.9158
```

Assessing the prediction performance of the two models

Next we will choose a model based on predictive performance for new data. We will start by using 10-fold cross-validation.

Cross-validation

10-fold CV involves taking the data and randomly splitting it into 10 groups. We fit 10 models, each using 90% of the data; then, we use these models to predict Y for the remaining 10% of the data. Each time, we calculate the RMSE (a measure of how much the predicted values differ from the true values), and average these 10 RMSEs to get an RMSE for the whole dataset. Smaller RMSE is better because it means that out of sample prediction is more accurate.

NOTE: Because CV splits the data into ten groups randomly, we need to set the seed to reproduce the results we get. Because random sampling occurs differently in the caret package compared to my code, we get slightly different numbers.

First, we do this by hand to gain understanding of what's going on:

```
set.seed(333)

# BY HAND METHOD:
grp_IDs <- sample(c(1:n), n)
RMSEs_true <- RMSEs_full <- rep(NA, 10)
tn <- seq(from=0, to=n, by=n/10)
for(i in 1:10){
  train <- data[grp_IDs[-c((tn[i]+1):(tn[i+1]))],]
  valid <- data[grp_IDs[c((tn[i]+1):(tn[i+1]))],]
  fullmod <- lm(Y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8, data = train)
  truemod <- lm(Y ~ X1 + X2 + X3 + X4, data = train)
  predfull <- predict(fullmod, newdata=valid[,1:8])
  predtrue <- predict(truemod, newdata=valid[,1:4])
  RMSEs_full[i] = sqrt(mean((predfull - valid$Y)^2))
  RMSEs_true[i] = sqrt(mean((predtrue - valid$Y)^2))
}

paste("The full model RMSE by CV is", mean(RMSEs_full))
```

```
## [1] "The full model RMSE by CV is 1.49507387753766"
```

```
paste("The true model RMSE by CV is", mean(RMSEs_true))
```

```
## [1] "The true model RMSE by CV is 1.48850271105531"
```

Now we do the same thing but with functions in the caret package

```r
set.seed(333)

# Save a setting of CV that specifies 10 groups
train.control <- trainControl(method = "cv", number = 10)

# Run CV on true model
true_cv <- train(formula(true_model), data = data, method = "lm",
                 trControl = train.control)

# Run CV on full model
full_cv <- train(formula(full_model), data = data, method = "lm",
                trControl = train.control)

# Pull RMSE from CV results of each model
paste("The full model RMSE by CV is", as.numeric(full_cv$results["RMSE"]))
```

```
## [1] "The full model RMSE by CV is 1.4984509977645"
```

```r
paste("The true model RMSE by CV is", as.numeric(true_cv$results["RMSE"]))
```

```
## [1] "The true model RMSE by CV is 1.49588484991131"
```

These RMSE values are really close, but generally the smaller value indicates better performance, the true model did better than the full model.

PRMSE from validation set

Split the data into training and validation sets

We split our data into a training/testing set (to build potential models) and a validation set (to be used for out-of-sample predictions to obtain a PRMSE).

```r
# Leave the last 150 observations out as the validation set
data_valid <- data[351:500, ]

# Use the first 350 observations as the training/testing set
data_train <- data[1:350, ]

# Re-save full and true models, but fit them only using the training data
full_model <- lm(Y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8, data = data_train)
true_model <- lm(Y ~ X1 + X2 + X3 + X4, data = data_train)
```

Now, we will predict the Y outcomes for the validation data using the model fit with the training data. We check how well each model predicted Y by comparing the predicted Y's to the actual Y's via predicted root mean squared error (PRMSE).

```r
# Predict validation set Y's for full model
predY_full <- predict(full_model, newdata = data_valid)

# Predict validation set Y's for true model
predY_true <- predict(true_model, newdata = data_valid)

# PRMSE for full_model
paste("The full model PRMSE on the validation set is", sqrt(mean((predY_full - data_valid$Y)^2)))
```

```
## [1] "The full model PRMSE on the validation set is 1.5642323741993"
```

```r
# PRMSE for true_model
paste("The true model PRMSE on the validation set is", sqrt(mean((predY_true - data_valid$Y)^2)))
```

```
## [1] "The true model PRMSE on the validation set is 1.55802179847819"
```

Again, the results are close, but again the true model appears to perform a little better than the full model.

Question to think about

1. Under what conditions do these methods choose the true model? Can you change the code above so that at least one method chooses the full model over the true model?