Lab 3

Lewis White

2023-01-25

Lab 3: Predicting the age of abalone

Abalones are marine snails. Their flesh is widely considered to be a desirable food, and is consumed raw or cooked by a variety of cultures. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope – a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the age.

The data set provided includes variables related to the sex, physical dimensions of the shell, and various weight measurements, along with the number of rings in the shell. Number of rings is the stand-in here for age.

Data Exploration

Pull the abalone data from Github and take a look at it.

```
#load in the data from github
abdat <- read_csv(file = "https://raw.githubusercontent.com/MaRo406/eds-232-machine-lear
ning/main/data/abalone-data.csv")
```

```
## New names:
## Rows: 4177 Columns: 10
## ── Column specification
## ──────────────────────────────────────────────────────── Delimiter: "," chr
## (1): Sex dbl (9): ...1, Length, Diameter, Height, Whole_weight, Shucked_weight,
## Visce...
## ℹ Use `spec()` to retrieve the full column specification for this data. ℹ
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## • `` -> `...1`
```

```
glimpse(abdat)
```

```
## Rows: 4,177
## Columns: 10
## $ ...1           <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, …
## $ Sex            <chr> "M", "M", "F", "M", "I", "I", "F", "F", "M", "F", "F", …
## $ Length         <dbl> 0.455, 0.350, 0.530, 0.440, 0.330, 0.425, 0.530, 0.545,…
## $ Diameter       <dbl> 0.365, 0.265, 0.420, 0.365, 0.255, 0.300, 0.415, 0.425,…
## $ Height         <dbl> 0.095, 0.090, 0.135, 0.125, 0.080, 0.095, 0.150, 0.125,…
## $ Whole_weight   <dbl> 0.5140, 0.2255, 0.6770, 0.5160, 0.2050, 0.3515, 0.7775,…
## $ Shucked_weight <dbl> 0.2245, 0.0995, 0.2565, 0.2155, 0.0895, 0.1410, 0.2370,…
## $ Viscera_weight <dbl> 0.1010, 0.0485, 0.1415, 0.1140, 0.0395, 0.0775, 0.1415,…
## $ Shell_weight   <dbl> 0.150, 0.070, 0.210, 0.155, 0.055, 0.120, 0.330, 0.260,…
## $ Rings          <dbl> 15, 7, 9, 10, 7, 8, 20, 16, 9, 19, 14, 10, 11, 10, 10, …
```

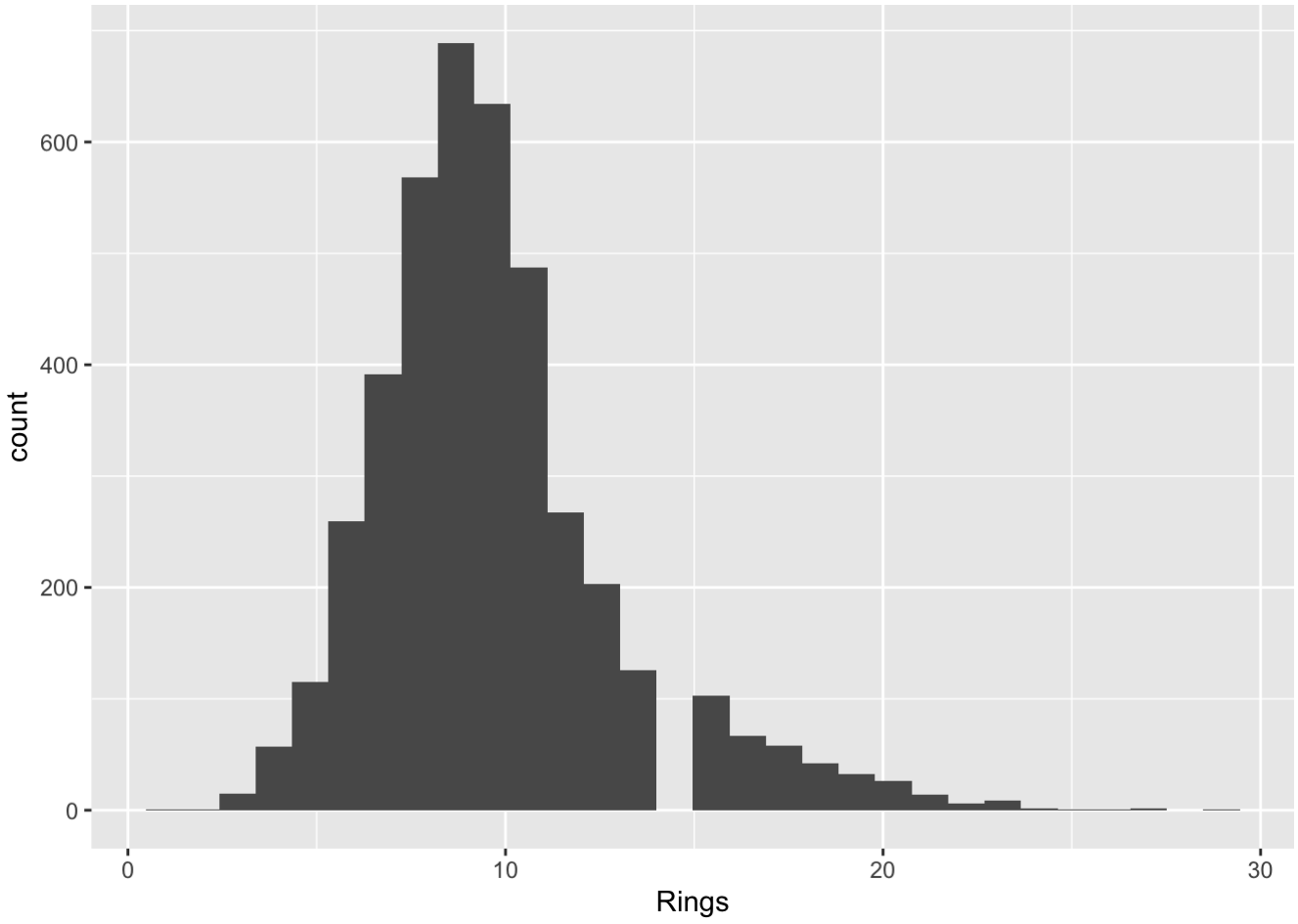
```
abdat_no_index <- abdat[,-1] #removing the index column from the data set
```

Data Splitting

- ***Question 1***. Split the data into training and test sets. Use a 70/30 training/test split.

```
set.seed(123) #set a seed for reproducibility
split <- initial_split(data = abdat_no_index,
                       prop = 0.7,
                       strata = "Rings")

abalone_train <- training(split) #creating the training data

abalone_test <- testing(split) #creating the testing data


#checking the distribution of the Rings variable to see whether a log transformation sho
uld be applied.
ggplot(data = abdat_no_index, aes(x = Rings)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
#The distribution seems fairly normal (although there is a slight skew to the right), so keeping it as is should be okay.
```

We'll follow our text book's lead and use the caret package in our approach to this task. We will use the glmnet package in order to perform ridge regression and the lasso. The main function in this package is glmnet(), which can be used to fit ridge regression models, lasso models, and more. In particular, we must pass in an x matrix of predictors as well as a y outcome vector, and we do not use the y~x syntax.

Fit a ridge regression model

- ***Question 2***. Use the model.matrix() function to create a predictor matrix, x, and assign the Rings variable to an outcome vector, y.

```
#Create training feature matrices using model.matrix() (auto encoding of categorical variables)

X <- model.matrix(Rings ~ ., data = abalone_train)[,-1] #compare Rings to all other predictors in the data. [,-1] removes the intercept variable.

head(X)
```

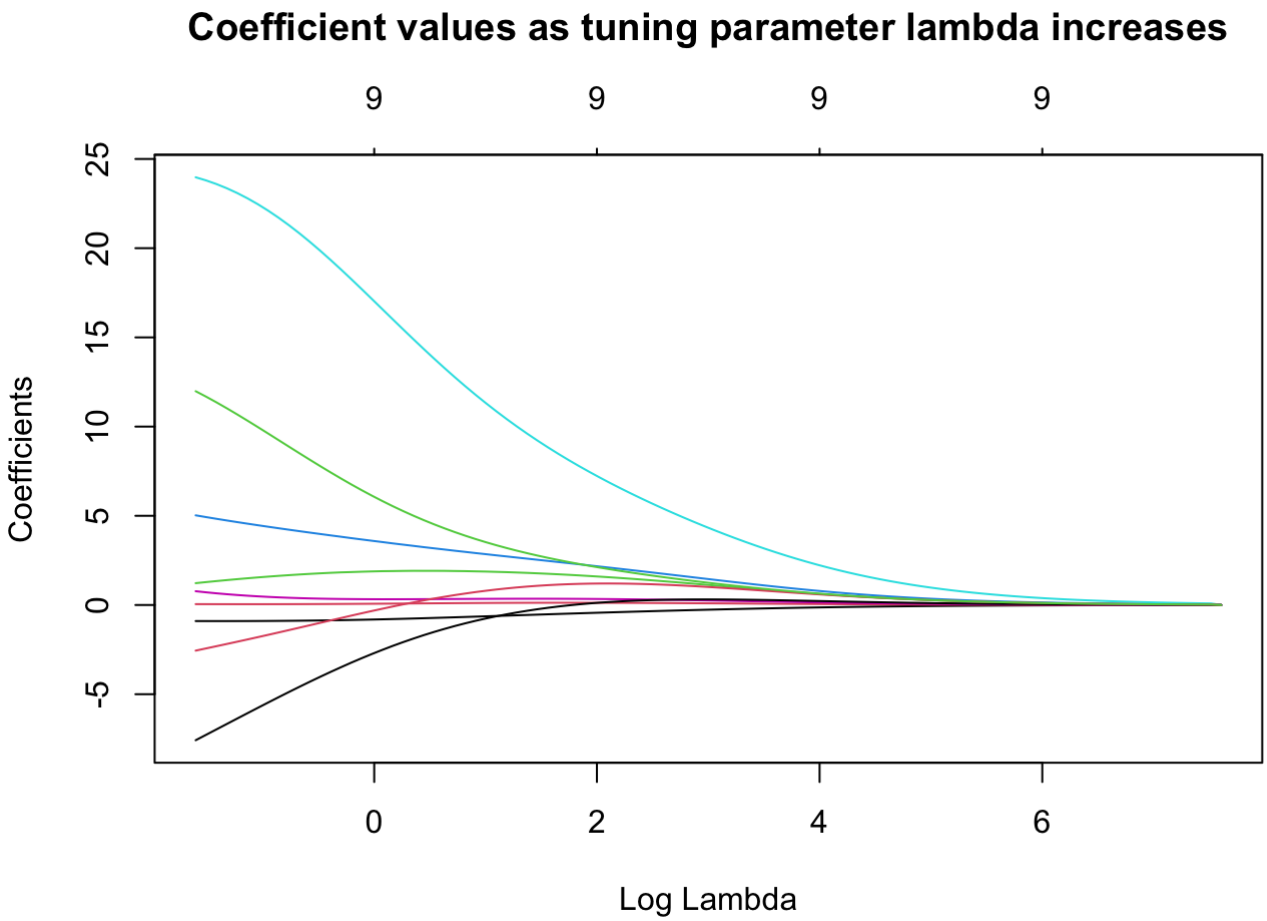
```
##   SexI SexM Length Diameter Height Whole_weight Shucked_weight Viscera_weight
## 1    1    0  0.330    0.255  0.080       0.2050         0.0895         0.0395
## 2    0    1  0.365    0.295  0.080       0.2555         0.0970         0.0430
## 3    0    1  0.465    0.355  0.105       0.4795         0.2270         0.1240
## 4    1    0  0.240    0.175  0.045       0.0700         0.0315         0.0235
## 5    1    0  0.205    0.150  0.055       0.0420         0.0255         0.0150
## 6    1    0  0.210    0.150  0.050       0.0420         0.0175         0.0125
##   Shell_weight
## 1        0.055
## 2        0.100
## 3        0.125
## 4        0.020
## 5        0.012
## 6        0.015
```

```
Y <- abalone_train$Rings #assign Rings value to outcome vector Y

head(Y)
```

```
## [1] 7 7 8 5 5 4
```

- ***Question 3***. Fit a ridge model (controlled by the alpha parameter) using the glmnet() function. Make a plot showing how the estimated coefficients change with lambda. (Hint: You can call plot() directly on the glmnet() objects).

```
#fit a ridge model, passing X,Y,alpha to glmnet()
abalone_ridge <- glmnet(x = X,
                        y = Y,
                        alpha = 0) #alpha = 0 indicates ridge regression

plot(abalone_ridge, xvar = "lambda")
title("Coefficient values as tuning parameter lambda increases", line = 3)
```



Using *k*-fold cross validation resampling and tuning our models

In lecture we learned about two methods of estimating our model's generalization error by resampling, cross validation and bootstrapping. We'll use the *k*-fold cross validation method in this lab. Recall that lambda is a tuning parameter that helps keep our model from over-fitting to the training data. Tuning is the process of finding the optima value of lamba.

- ***Question 4***. This time fit a ridge regression model and a lasso model, both with using cross validation. The glmnet package kindly provides a cv.glmnet() function to do this (similar to the glmnet() function that we just used). Use the alpha argument to control which type of model you are running. Plot the results.

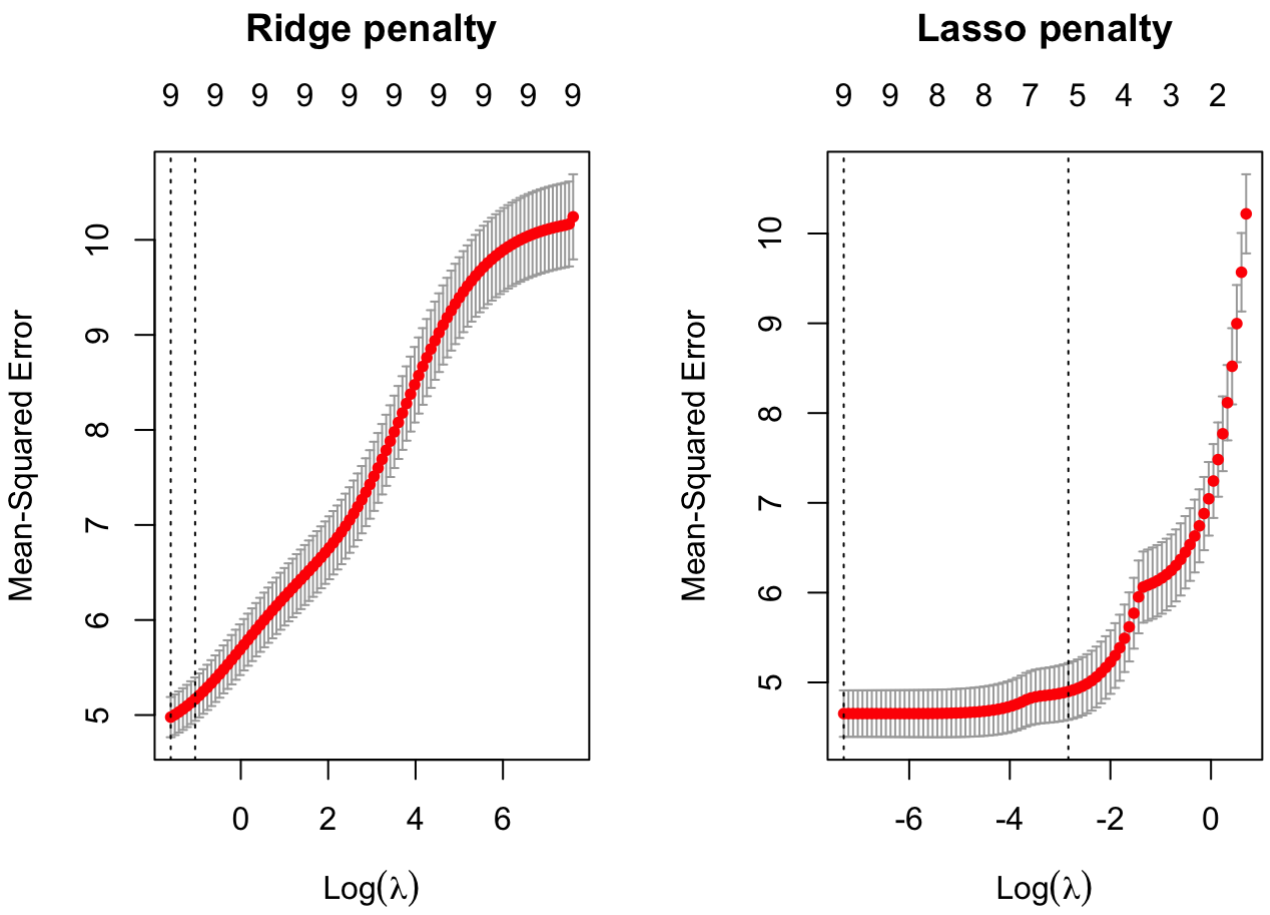
```
#Apply RIDGE regression with cross validation to the abalone data ----
cv_abalone_ridge <- cv.glmnet(x = X,
                              y = Y,
                              alpha = 0)

cv_abalone_ridge
```

```
##
## Call:  cv.glmnet(x = X, y = Y, alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure     SE Nonzero
## min 0.2012   100   4.978 0.2116       9
## 1se 0.3516    94   5.168 0.2279       9
```

```
#       Lambda   Index   Measure     SE     Nonzero
# min   0.2012    100    4.978     0.2116       9
# 1se   0.3516    94     5.168     0.2279       9


#Apply LASSO regression with cross validation to the abalone data ----
cv_abalone_lasso <- cv.glmnet(x = X,
                              y = Y,
                              alpha = 1)

cv_abalone_lasso
```

```
##
## Call:  cv.glmnet(x = X, y = Y, alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure     SE Nonzero
## min 0.00067    87   4.653 0.2587       9
## 1se 0.05866    39   4.899 0.3162       5
```

```
#        Lambda   Index   Measure     SE      Nonzero
# min   0.00275    72     4.810     0.2561       8
# 1se   0.05925    39     5.058    0.2782        5


#plotting the results ----
par(mfrow = c(1, 2))
plot(cv_abalone_ridge, main = "Ridge penalty\n\n")
plot(cv_abalone_lasso, main = "Lasso penalty\n\n")
```



- ***Question 5***. Interpret the graphs. What is being show on the axes here? How does the performance of the models change with the value of lambda?

On the lower x axis we have the logged value of the tuning parameter lambda, on the y axis we have the mean-squared error value, and on the upper x axis we have the number of non-zero coefficients included in the model. The chart shows how the mean squared error changes as lambda increases.

For the ridge model, as lambda increases, the mean-squared error increases pretty quickly. As a result the minimum MSE occurs at a lambda very close to 0, as does the 1 standard error above the minimum MSE value. No feature selection is applied, so all 9 predictors remain in the model

For the Lasso model, it initially looks like the lambda value doesn't have as strong of an effect on the MSE, but the axis values differ from the Ridge model. For the lasso model, minor changes in lambda greatly increase the MSE and the minimum MSE occurs extremely close to when lambda is 0 (the actual value was 0.002). The 1se lambda value (0.05) is close to 0 as well. In this model, feature selection occurs ~ the minimum MSE has 8 predictors and the 1-SE MSE has 5 predictors.

- ***Question 6***. Inspect the ridge model object you created with cv.glmnet(). The $cvm column shows the MSEs for each cv fold. What is the minimum MSE? What is the value of lambda associated with this MSE minimum?

```
print(paste("The minimum MSE for the ridge model was", min(cv_abalone_ridge$cvm), "and i
t occured at a lambda value of", min(cv_abalone_ridge$lambda)))
```

```
## [1] "The minimum MSE for the ridge model was 4.978248482699 and it occured at a lambd
a value of 0.201214210150836"
```

- ***Question 7***. Do the same for the lasso model. What is the minimum MSE? What is the value of lambda associated with this MSE minimum?

```
print(paste("The minimum MSE for the lasso model was", min(cv_abalone_lasso$cvm), "and i
t occured at a lambda value of", min(cv_abalone_lasso$lambda)))
```

```
## [1] "The minimum MSE for the lasso model was 4.65276174691365 and it occured at a lam
bda value of 0.000674390080148105"
```

Data scientists often use the “one-standard-error” rule when tuning lambda to select the best model. This rule tells us to pick the most parsimonious model (fewest number of predictors) while still remaining within one standard error of the overall minimum cross validation error. The cv.glmnet() model object has a column that automatically finds the value of lambda associated with the model that produces an MSE that is one standard error from the MSE minimum ($lambda.1se).

- ***Question 8.*** Find the number of predictors associated with this model (hint: the $nzero is the # of predictors column).

```
#ridge model doesn't include feature selection, so the number of predictors at the 1-SE
value is still 10.


#LASSO MODEL ----

#find the MSE value when the lambda value is equal to the 1-SE MSE value.
cv_abalone_lasso$cvm[cv_abalone_lasso$lambda == cv_abalone_lasso$lambda.1se] # 5.041068
```

```
## [1] 4.899354
```

```
#the lambda value at the 1-SE MSE
cv_abalone_lasso$lambda.1se  # lambda for this MSE
```

```
## [1] 0.05865501
```

```
# 0.04891337

#the number of features in the model when the lambda value is equal to the 1-SE MSE lamb
da value
cv_abalone_lasso$nzero[cv_abalone_lasso$lambda == cv_abalone_lasso$lambda.1se]
```

```
## s38
##   5
```

```
# s40
#  5
```

There are 9 coefficients in the lasso model when the MSE is at it's minimum and 5 when the MSE is 1-SE away from the minimum value.

- **Question 9.** Which regularized regression worked better for this task, ridge or lasso? Explain your answer.

I think the lasso regularized regression worked better for this task. The minimum (and 1-SE) MSE values were both lower than the minimum MSE for the ridge regression. Additionally, if we're aiming to have the simplest model possible, the 1-SE version of the lasso model eliminated 4 predictors, leaving us with just 5 in the model. For these reasons, it seems like the Lasso model is a better fit for this task.