Lab4

Mateo Robbins

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.

```
abdat<- dat <- read_csv(file = "https://raw.githubusercontent.com/MaRo406/eds-232-machine-learning/main/
```

```
## 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, ~
```

Data Splitting

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

```
# Stratified sampling with the rsample package
set.seed(123) #set a seed for reproducibility
split <- initial_split(data =abdat,
                       prop= .70)
split
```

```
## <Training/Testing/Total>
## <2923/1254/4177>
```

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 yx syntax.

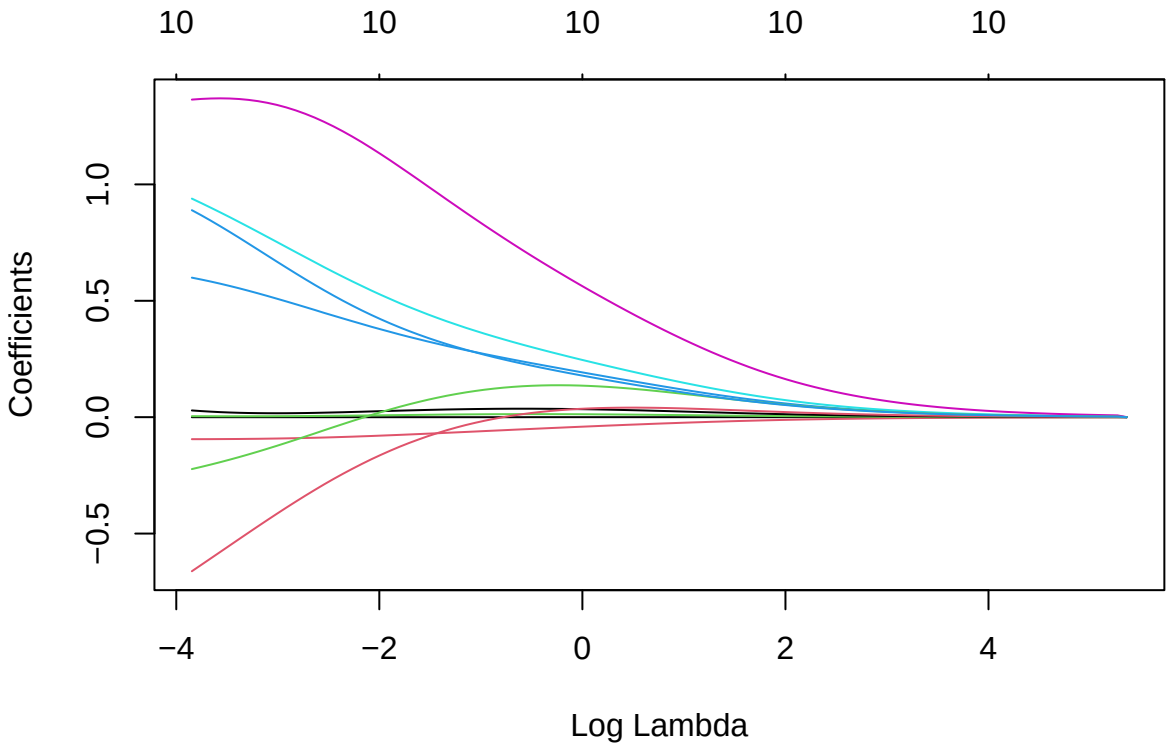
Fit a ridge regression model

- ***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 ~ ., abdat )[,-1]# remove intercept column


Y<- log(abdat$Rings)

#ex <- (abdat$Rings) # check to see if I should Log, slighly better logged

# hist(Y)
# hist(ex)
```

- ***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()
ridge <- glmnet(x = X,
                y = Y,
                alpha = 0)

#plot() the glmnet model object
plot(ridge, xvar = 'lambda')
```



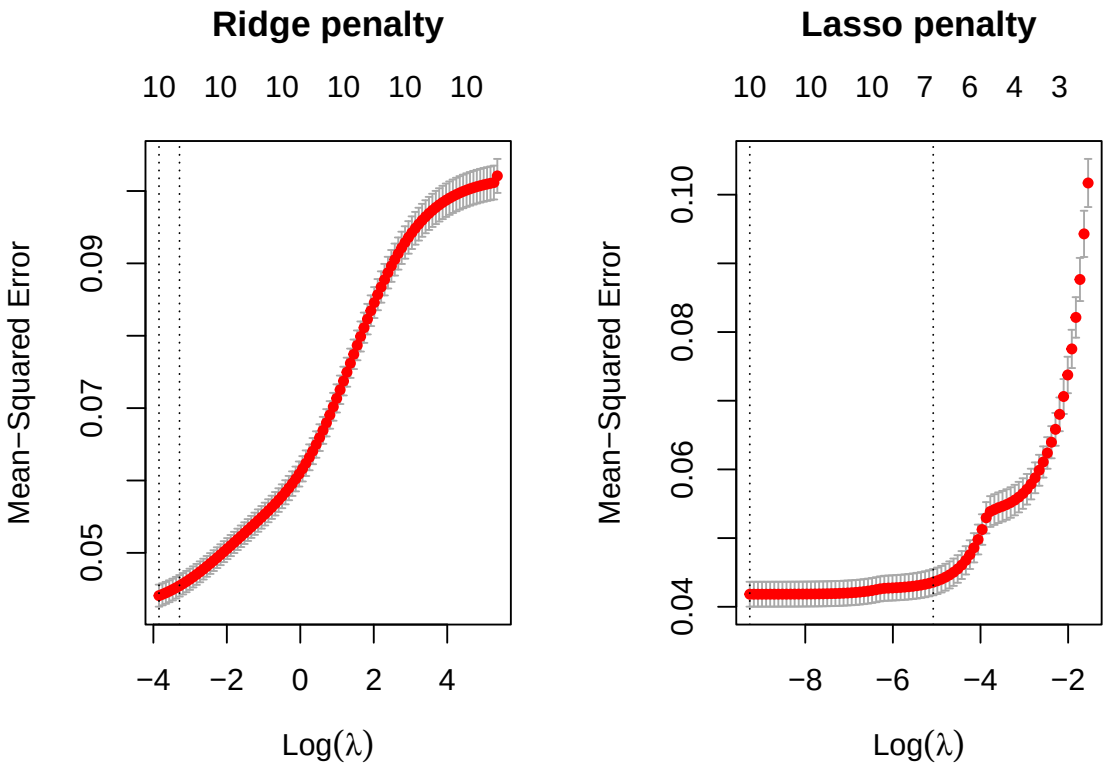
Using *k*-fold cross validation resampling and tuning our models

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 cross-validation (CV) ridge regression to abdat data.  Same arguments as before to glmnet()

#default n-folds = 10 most optimal
ridge <- cv.glmnet(
  x = X,
  y = Y,
  alpha = 0
)

# Apply CV lasso regression to Ames data
lasso <-  cv.glmnet(
  x = X,
  y = Y,
```

```
  alpha = 1
)

# plot results
par(mfrow = c(1, 2))


plot(ridge, main = "Ridge penalty\n\n")
plot(lasso, main = "Lasso penalty\n\n")
```



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

  The x-axis is log of lambda which controls the size of the penalty, the y-axis is the objective function, and the top axis is the number of features. As lambda increases the penalty increases and the performance of the model decreases with a higher objective function. The dashed lines on the x-axis tell us the location of the optimal lambda for the model. The ridge penalty indicates the optimal lambda is between -3 and -4 with 10 features, and the lasso penalty indicates the optimal lambda is between -5 and -9 between 7 and 10 features respectively. The lasso penalty has a wider range of features and loses performance has it loses features lower than 7. Both perform well at ~0.04 MSE with 10 features.

- ***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?

```
min(ridge$cvm)
```

```
## [1] 0.044089
```

```
max(ridge$cvm)
```

```
## [1] 0.1020853
```

The minimum MSE is 0.044089 and the maximum MSE is 0.1020853

- ***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?

```
min(lasso$cvm)
```

```
## [1] 0.04182889
```

```
max(lasso$cvm)
```

```
## [1] 0.1017027
```

  - **The minimum MSE is 0.04182889 and the maximum MSE is 0.107027**

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$cvm[ridge$lambda == ridge$lambda.1se]  # 1-SE rule
```

```
## [1] 0.04544854
```

```
lasso$cvm[lasso$lambda == lasso$lambda.1se]  # 1-SE rule
```

```
## [1] 0.04358488
```

```
lasso$nzero[lasso$lambda == lasso$lambda.1se] # No. of coef | 1-SE MSE
```

```
## s38
##   7
```

The ridge penalty has 10 predictors has it does not perform feature engineering and the lasso penalty has 7 predictors associated with the model.

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

  The lasso penalty worked better for this task as it the most parsimonious will 7 features as opposed to ten for the ridge penalty while also having a slighlty lower MSE within one standard error.