Herbst_Lab3

Amanda Herbst

2024-01-25

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.

```
# read in abalone data
abdat<- read_csv(file = "https://raw.githubusercontent.com/MaRo406/eds-232-machine-learning/main/data/al
                 show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```
# glimpse raw data
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, ~
```

```
# remove ...1 column which is just an index
abdat <- abdat %>%
  select(-...1)
```

Data Splitting

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

```r
# Data splitting with {rsample}
set.seed(123) #set a seed for reproducibility
split <- initial_split(abdat, prop = 0.7)

# training data
ab_train <-  training(split)

# testing data
ab_test  <-  testing(split)
```

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

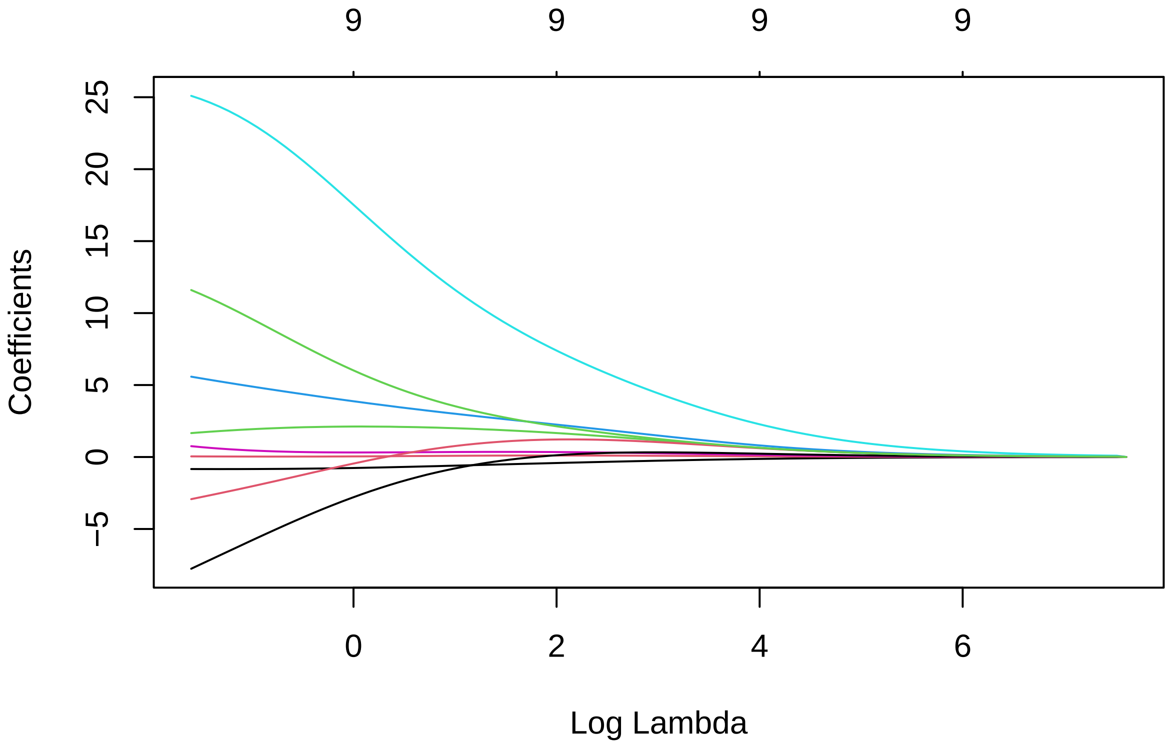
- ***Question 2***. Use the model.matrix() function to create a predictor matrix, x, and assign the Rings variable to an outcome vector, y.

```r
# Create predictor matrix for outcome variable Rings (auto encoding of categorical variables)
x <- model.matrix(Rings ~ ., data = ab_train)[,-1]
```

- ***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).

```r
# Fit ridge model to abalone training data
ab_ridge <- glmnet(
  x = x,
  y = ab_train$Rings, # Rings are distributed approximately normally
  alpha = 0
)

# plot ridge model coefficients changing with lambda
plot(ab_ridge, xvar = "lambda")
```



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

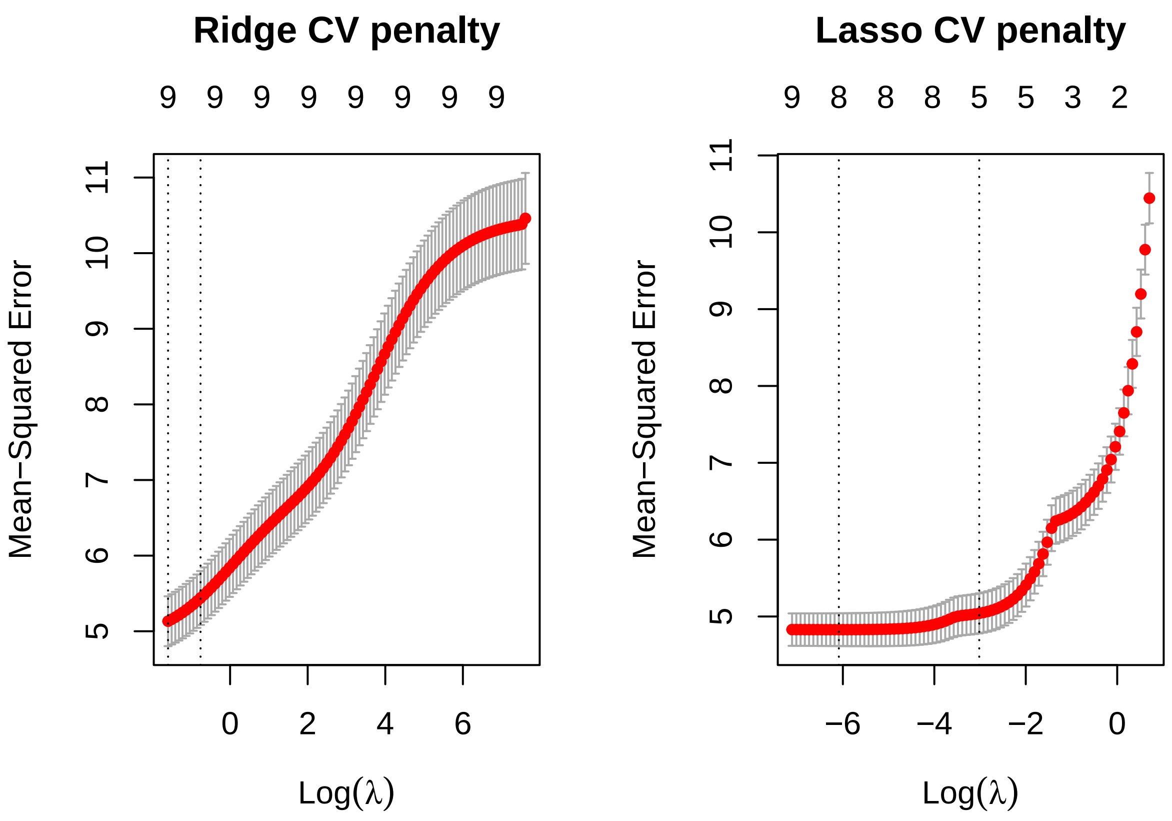
In lecture we learned about two methods of estimating our model's generalization error by resampling, cross validation and bootstrapping. We'll use the *k*-fold cross validation method in this lab. Recall that lambda is a tuning parameter that helps keep our model from over-fitting to the training data.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

```r
# Apply CV ridge regression to abalone training data.
ab_cv_ridge <- cv.glmnet(
  x = x,
  y = ab_train$Rings,
  alpha = 0
)

# Apply CV lasso regression to abalone training data
ab_cv_lasso <- cv.glmnet(
  x = x,
  y = ab_train$Rings,
  alpha = 1
)

# plot ridge and lasso CV models
par(mfrow = c(1, 2))
plot(ab_cv_ridge, main = "Ridge CV penalty\n\n")
plot(ab_cv_lasso, main = "Lasso CV penalty\n\n")
```



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

On the y-axes is mean-squared error of each model, and on the x-axes is the log of lambda, a tuning parameter that is used to minimize SSE and prevent overfitting. The vertical dotted line on the left indicates the optimal lambda based on where the minimum MSE occurs. The second dotted line (on the right) indicates the lambda at 1 standard error above the minimum MSE. This second line indicates a lambda with a low MSE while also increasing the parsimony of the model. The ridge regularization model appears to have a slighly higher MSE than lasso and, due to a higher sloped curve at the start, a smaller range in log(lambda) from minimum MSE to 1 standard error above. For the ridge model, the MSE increases at close to a constant rate as log(lambda) increases. In contrast, the MSE of the lasso model increase exponentially with increases in log(lambda).

- ***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?

```r
# Ridge model
# minimum MSE
min(ab_cv_ridge$cvm) # 5.1
```

```
## [1] 5.13292
```

```r
# lambda for this min MSE
ab_cv_ridge$lambda.min # 0.20
```

```
## [1] 0.2021107
```

The minimum MSE for the ridge model is about 5.1 and the lambda at that minimum is about 0.2.

- ***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?

```
# minimum MSE
min(ab_cv_lasso$cvm) # 4.8
```

```
## [1] 4.828691
```

```
# lambda for this min MSE
ab_cv_lasso$lambda.min # 0.002
```

```
## [1] 0.002270358
```

The minimum MSE for the lasso model is about 4.8 and teh lambda at that minimum is about 0.002

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).

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

```
## s40
##   5
```

There are 5 predictors associated with the lasso model at one standard error above the minimum MSE

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

The lasso regularized regression works better for this task because it minimizes the mean squared error better (4.8 compared to 5.1 for ridge). Additionally, it pares down the number of predictor variables to 5 which is better because it helps to avoid overfitting and any correlation between variables.