Unit 3 Lecture 3: Ridge regression

October 12, 2021

In this R demo, we will learn about the `glmnet` package and how to run a cross-validated ridge regression using the `cv.glmnet()` function.

First, let's install and load the `glmnet` package.

```
# install.packages("glmnet")
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-2
```

Let's also source a function called `plot_glmnet` to help us plot our results:

```
source("../../functions/plot_glmnet.R")
```

We will be applying ridge regression to study the effect of 97 socioeconomic factors on violent crimes per capita based on data from 90 communities in Florida:

```
crime_data = read_csv("../../data/CrimeData_FL.csv")
```

```
## Rows: 90 Columns: 98

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (98): population, household.size, race.pctblack, race.pctwhite, race.pct...

##
## 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.
```

```
crime_data
```

```
## # A tibble: 90 x 98
##    population household.size race.pctblack race.pctwhite race.pctasian
##         <dbl>          <dbl>         <dbl>         <dbl>         <dbl>
##  1      16023           2.63         13.8           83.9          1.42
##  2      29721           2.34          3.52          95.1          1.03
##  3      10205           2.46          1.06          97.4          1.04
##  4     124773           2.47         29.1           68.2          1.75
##  5      13024           2.25         31.3           67.2          0.5
##  6     280015           2.44         25.0           70.9          1.35
##  7      79443           2.94          3.48          93.1          2.12
##  8      16444           2.57          5.38          91.2          1.96
```

```
##  9      46194            2.28         20.1            77.7           0.63
## 10      14044            2.17          0.48           98.3           0.58
## # ... with 80 more rows, and 93 more variables: race.pcthisp <dbl>,
## #   age.pct12to21 <dbl>, age.pct12to29 <dbl>, age.pct16to24 <dbl>,
## #   age.pct65up <dbl>, pct.urban <dbl>, med.income <dbl>, pct.wage.inc <dbl>,
## #   pct.farmself.inc <dbl>, pct.inv.inc <dbl>, pct.socsec.inc <dbl>,
## #   pct.pubasst.inc <dbl>, pct.retire <dbl>, med.family.inc <dbl>,
## #   percap.inc <dbl>, white.percap <dbl>, black.percap <dbl>,
## #   indian.percap <dbl>, asian.percap <dbl>, hisp.percap <dbl>, ...
```

Standardizing the features and train/test split

The syntax of `glmnet()` is slightly different from that of `lm()` and `glm()`. Instead of specifying a formula, it takes arguments `x` and `y` representing the matrix of features and the vector of responses. To create these, we use the following syntax:

```
X = model.matrix(violentcrimes.perpop ~ ., data = crime_data)[,-1]
Y = crime_data %>% pull(violentcrimes.perpop)
```

We remove the intercept term via `[,-1]` because it will be added in automatically by `glmnet`.

Then we standardize the matrix `X`, as discussed in the slides:

```
X_ctr = apply(X, 2, function(col)(col-mean(col)))
X_std = apply(X_ctr, 2, function(col)(col/sqrt(sum(col^2)/nrow(X_ctr))))
```

Finally, let's split the (standardized) data into training and testing, as usual:

```
set.seed(471)
train_samples = sample(1:nrow(crime_data), 0.8*nrow(crime_data))
X_std_train = X_std[train_samples,]
X_std_test = X_std[-train_samples,]
Y_train = Y[train_samples]
Y_test = Y[-train_samples]
```

Running a cross-validated ridge regression

Finally, we call `cv.glmnet` on `X_std_train` and `Y_train`.

```
ridge_fit = cv.glmnet(x = X_std_train, y = Y_train, alpha = 0, nfolds = 10)
```

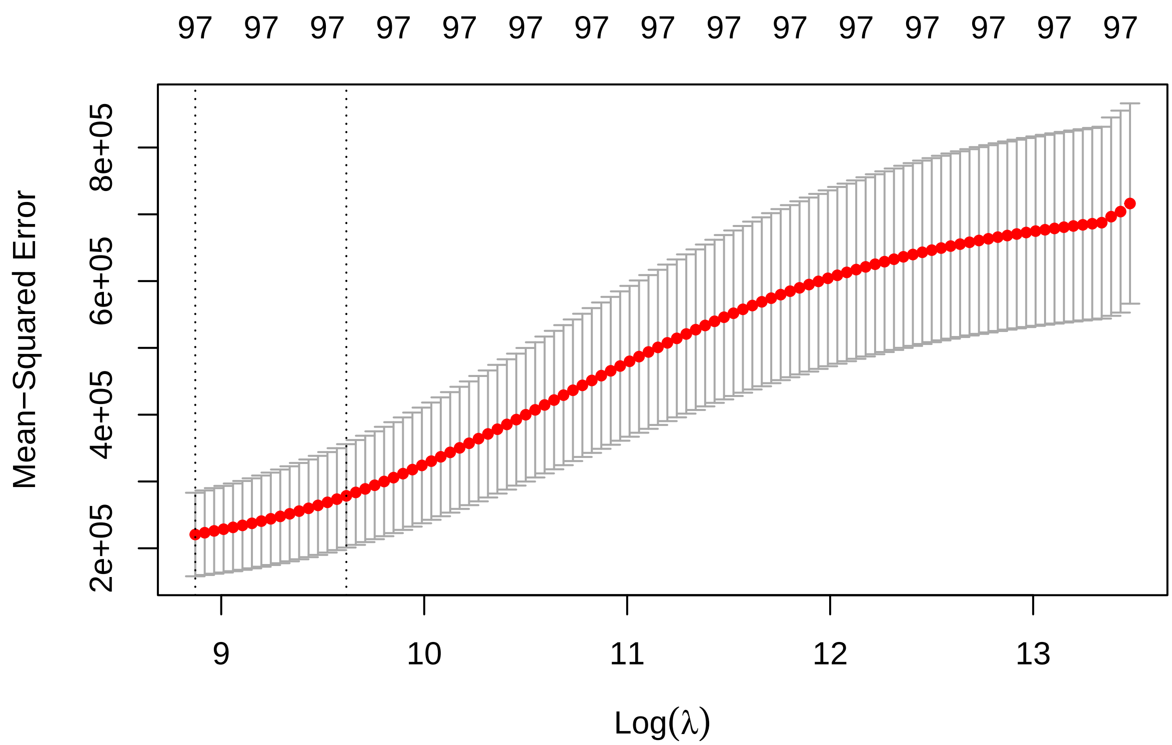
A few things to note:

- the sequence of penalty parameters is automatically chosen for you
- `alpha = 0` means "ridge regression" (we'll discuss other values of alpha next lecture)
- `nfolds` specifies the number of folds for cross-validation

Inspecting the results

The `glmnet` package has a very nice `plot` function to produce the CV plot:

```
plot(ridge_fit)
```



The `ridge_fit` object has several fields with information about the fit:

```
# lambda sequence
head(ridge_fit$lambda)
```

```
## [1] 713062.9 680653.1 649716.3 620185.7 591997.3 565090.1
```

```
# CV estimates
head(ridge_fit$cvm)
```

```
## [1] 716135.0 704041.8 696485.3 687461.6 685869.8 684210.5
```

```
# CV standard errors
head(ridge_fit$cvsd)
```

```
## [1] 149960.0 151175.0 148506.5 143594.0 143369.5 143135.3
```

```
# lambda achieving minimum CV error
ridge_fit$lambda.min
```

```
## [1] 7130.629
```

```
# lambda based on one-standard-error rule
ridge_fit$lambda.1se
```

```
## [1] 15009.29
```

To get the fitted coefficients at the selected value of lambda:

```
coef(ridge_fit, s = "lambda.1se") %>% head()
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                          s1
## (Intercept)    1197.8370420
## population        7.7016017
## household.size    0.8577325
```

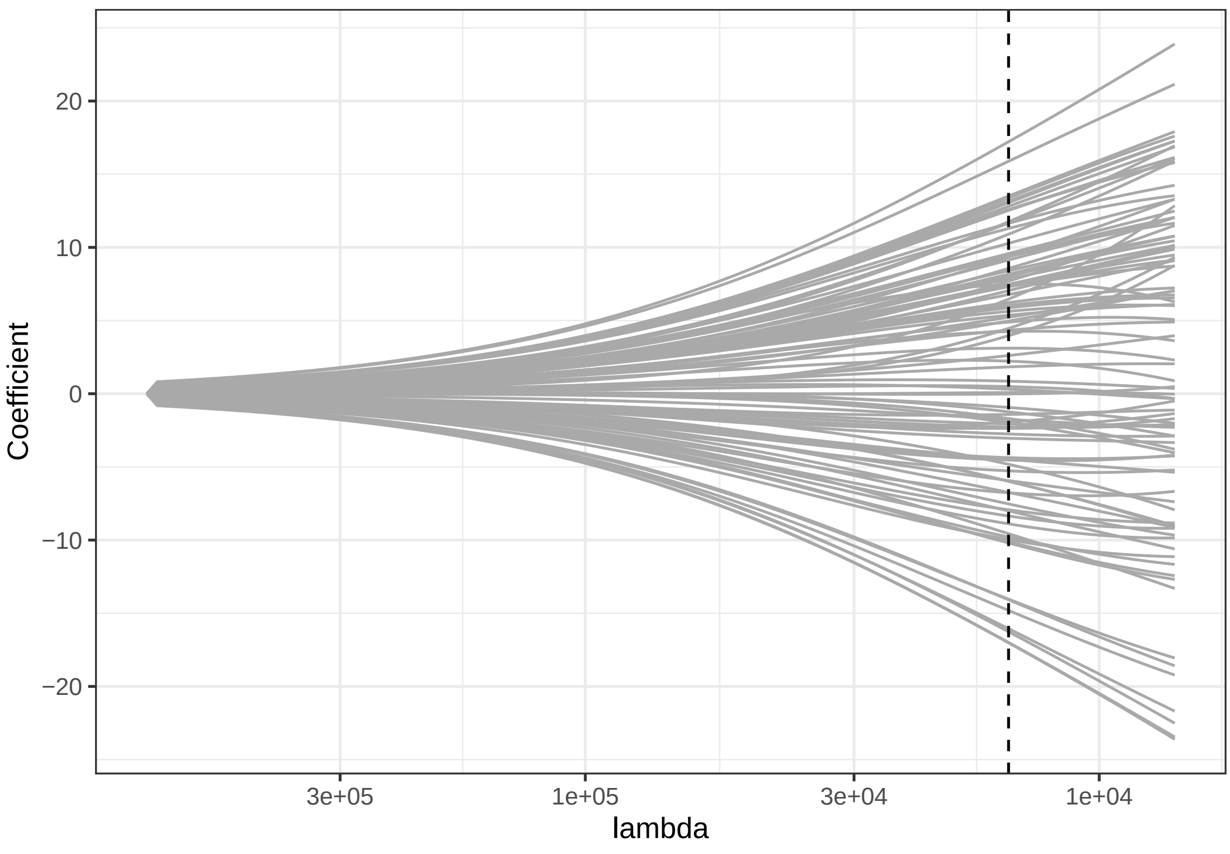
```
## race.pctblack    13.3820714
## race.pctwhite   -14.0796980
## race.pctasian    -4.8229375
```

```
coef(ridge_fit, s = "lambda.min") %>% head()
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                          s1
## (Intercept)    1183.8282446
## population       11.5013105
## household.size    0.3592821
## race.pctblack    17.6027889
## race.pctwhite   -18.5888142
## race.pctasian    -7.9364023
```

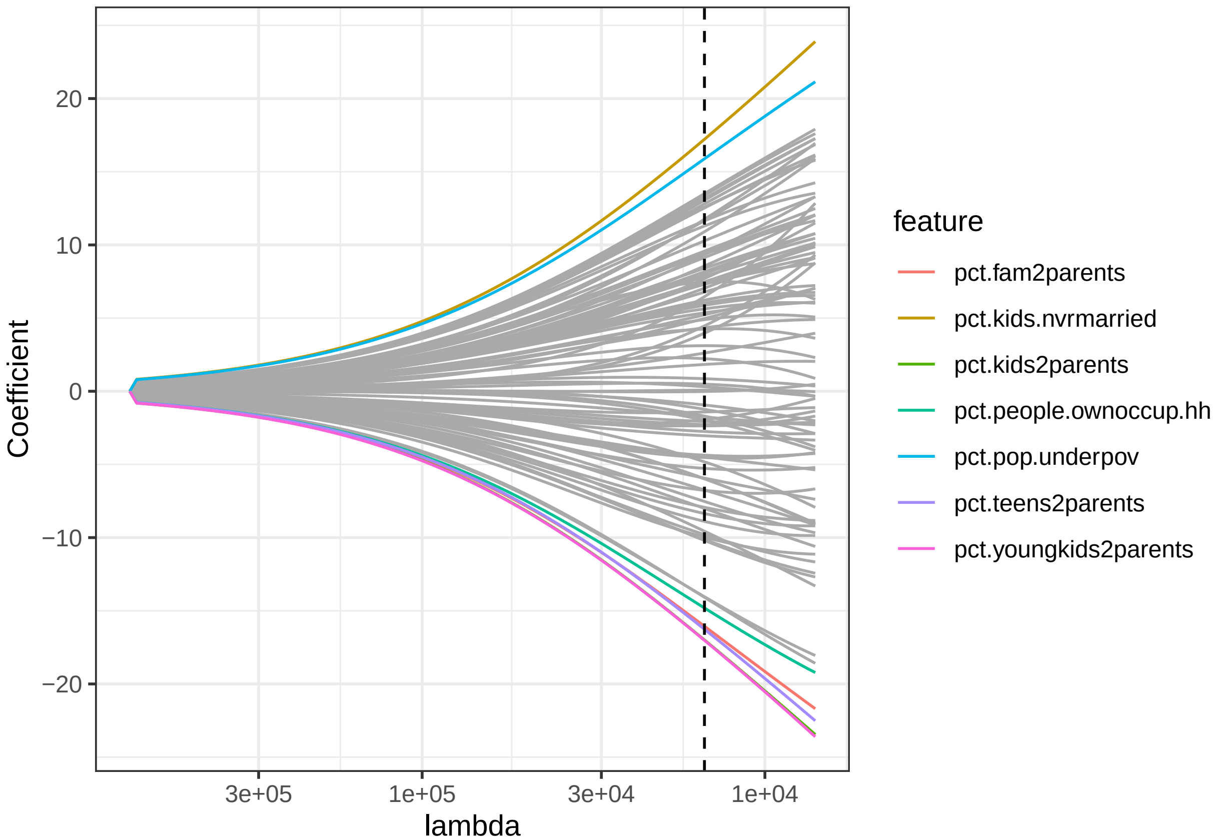
To visualize the fitted coefficients as a function of lambda, we can make a plot of the coefficients like we saw in class. To do this, we can use the `plot_glmnet` function, which by default shows a dashed line at the lambda value chosen using the one-standard-error rule:

```
plot_glmnet(ridge_fit)
```



If we want to annotate the features with the top few coefficients, we can use the `features_to_plot` argument:

```
plot_glmnet(ridge_fit, features_to_plot = 7)
```



To interpret these coefficient estimates, recall that they are for the *standardized* features.

Making predictions

To make predictions on the test data, we can use the `predict` function (which we've seen before):

```
ridge_predictions = predict(ridge_fit,
                            newx = X_std_test,
                            s = "lambda.1se") %>% as.numeric()
ridge_predictions
```

```
##  [1] 1692.6549 1336.5710 1059.0094  810.7225  721.6361  740.8567 1004.8753
##  [8]  844.6853  871.6307  772.2077  598.1545 1372.3992 1255.7212 1257.5605
## [15] 1430.3861  742.5142  744.3514  750.0670
```

We can evaluate the root-mean-squared-error as before:

```
RMSE = sqrt(mean(ridge_predictions - Y_test)^2)
RMSE
```

```
## [1] 193.9508
```

Ridge logistic regression

We can also run a ridge-penalized logistic regression. Let's try it out on `default_data`.

```
default_data = ISLR2::Default %>%
  as_tibble() %>%
```

```
  mutate(default = as.numeric(default == "Yes"))
default_data
```

```
## # A tibble: 10,000 x 4
##    default student balance income
##      <dbl> <fct>     <dbl>  <dbl>
##  1       0 No         730. 44362.
##  2       0 Yes        817. 12106.
##  3       0 No        1074. 31767.
##  4       0 No         529. 35704.
##  5       0 No         786. 38463.
##  6       0 Yes        920.  7492.
##  7       0 No         826. 24905.
##  8       0 Yes        809. 17600.
##  9       0 No        1161. 37469.
## 10       0 No           0  29275.
## # ... with 9,990 more rows
```

We generate the model matrix, then standardize, then split:

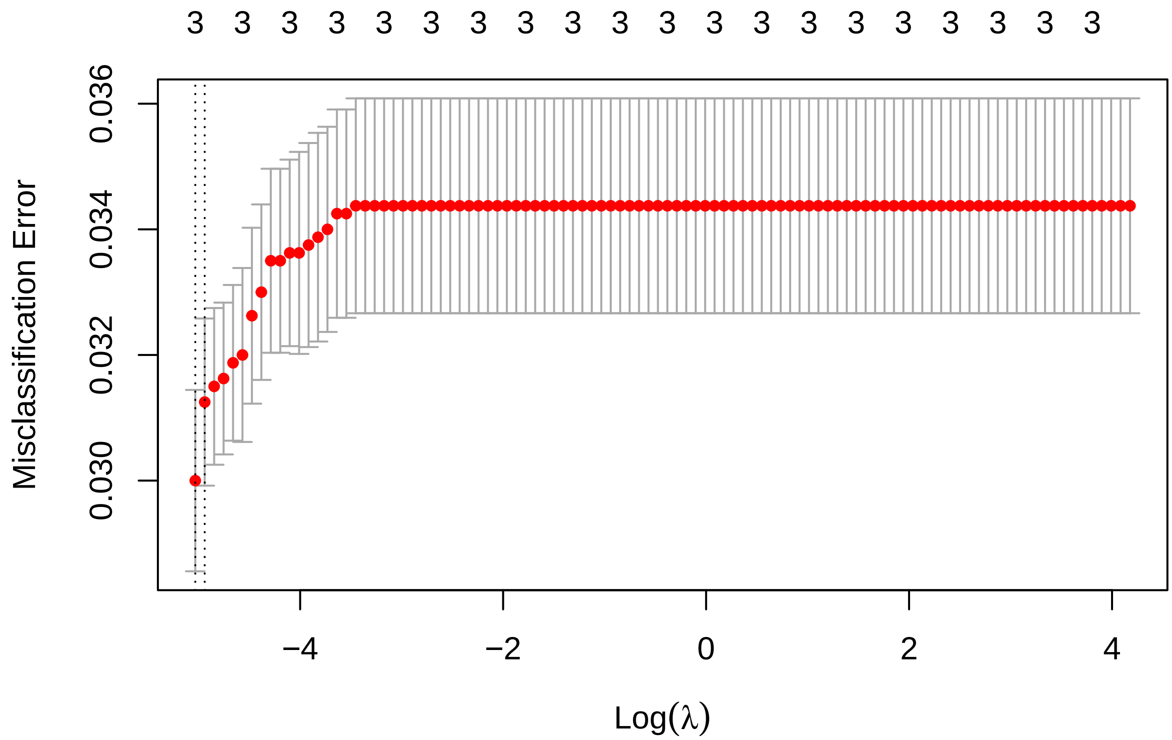
```
# generate model matrix
X = model.matrix(default ~ ., data = default_data)[,-1]
Y = default_data %>% pull(default)

# standardize
X_ctr = apply(X, 2, function(col)(col-mean(col)))
X_std = apply(X_ctr, 2, function(col)(col/sqrt(sum(col^2)/nrow(X_ctr))))

# split
set.seed(471)
train_samples = sample(1:nrow(default_data), 0.8*nrow(default_data))
X_std_train = X_std[train_samples,]
X_std_test = X_std[-train_samples,]
Y_train = Y[train_samples]
Y_test = Y[-train_samples]
```

To run the logistic ridge regression, we call `cv.glmnet` as before, adding the argument `family = binomial` to specify that we want to do a logistic regression and the argument `type.measure = "class`: to specify that we want to use the misclassification error during cross-validation.

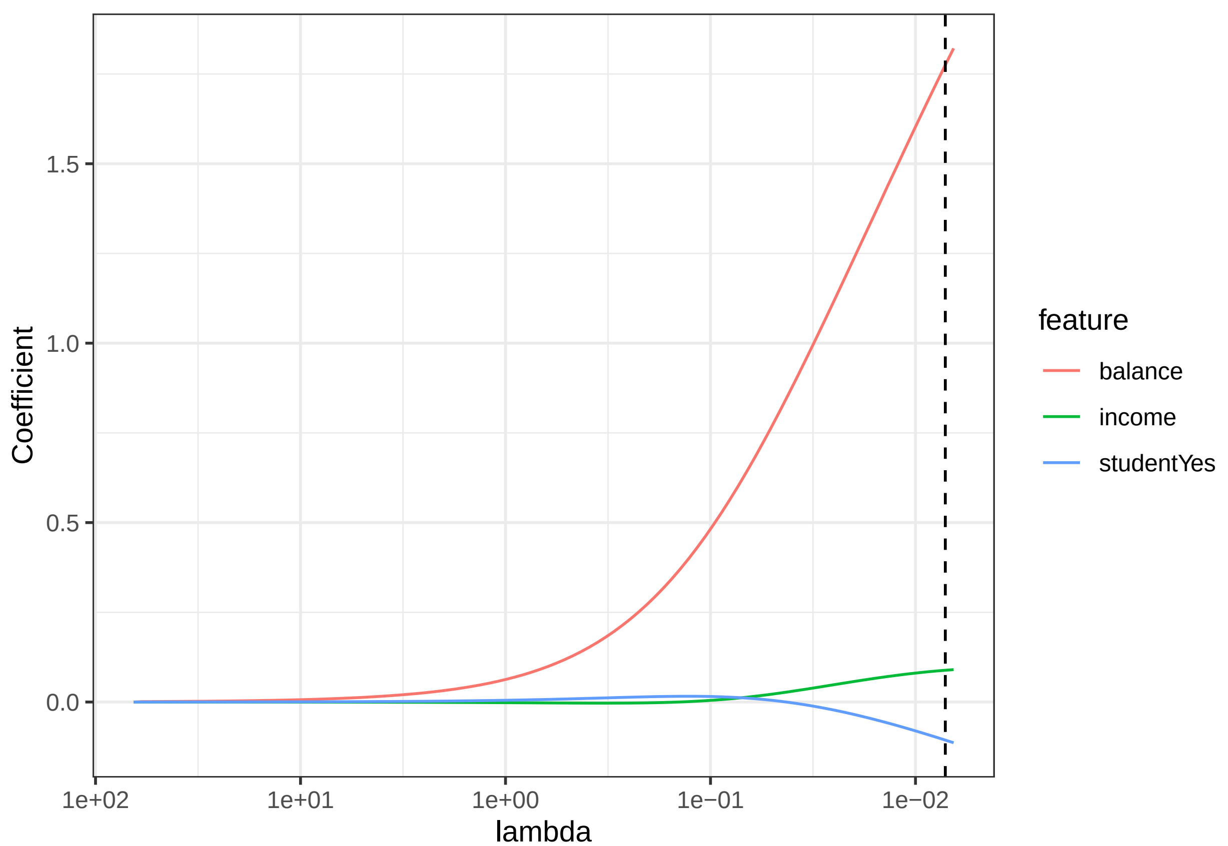
```
ridge_fit = cv.glmnet(x = X_std_train, y = Y_train, alpha = 0, nfolds = 10,
                      family = "binomial", type.measure = "class")
```

We can then take a look at the CV plot and the trace plot as before:

```
plot(ridge_fit)
```




```r
plot_glmnet(ridge_fit, features_to_plot = 3)
```



To predict using the fitted model, we can use the `predict` function again, this time specifying `type = "response"` to get the predictions on the probability scale (as opposed to the log-odds scale).

```r
probabilities = predict(ridge_fit, newx = X_std_test,
                        s = "lambda.1se", type = "response") %>% as.numeric()
head(probabilities)
```

```
## [1] 0.0003402425 0.0252037023 0.0014303995 0.0022884882 0.0012022554
## [6] 0.0047751518
```

We can threshold the probabilities to get binary predictions as we did with regular logistic regression.