Unit 3 Lecture 3: Ridge regression

October 10, 2023

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

```
library(tidyverse)
library(glmnetUtils) # for cv.glmnet()
library(stat471)     # for plot_glmnet(), coef_tidy()
```

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("CrimeData_FL.csv")
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
## # i 80 more rows
## # i 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>, ...
```

Let's split the data into training and testing, as usual:

```
set.seed(471)
train_samples <- sample(1:nrow(crime_data), 0.8 * nrow(crime_data))
crime_train <- crime_data |> filter(row_number() %in% train_samples)
crime_test <- crime_data |> filter(!(row_number() %in% train_samples))
```

Running a cross-validated ridge regression

We call `cv.glmnet` on `crime_train`:

```
set.seed(471)
ridge_fit <- cv.glmnet(
```

```
  violentcrimes.perpop ~ ., # formula notation, as usual
  alpha = 0,                # alpha = 0 for ridge (stay tuned for more)
  nfolds = 10,              # number of folds
  data = crime_train        # data to run ridge on
)
```

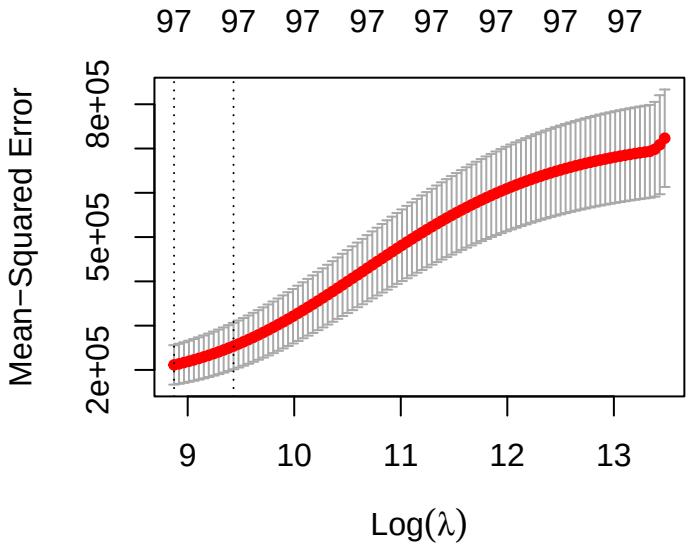
A few things to note:

- 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
- the columns of the matrix `X` are being standardized for you behind the scenes; there is no need to standardize yourself

Inspecting the results

The `glmnetUtils` package has a `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] 723198.4 708573.4 698343.2 693646.9 692021.4 690326.8
```

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

```
## [1] 110115.5 111834.4 107008.7 104828.4 104651.7 104467.4
```

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

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

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

```
## [1] 12460.98
```

To get the fitted coefficients at the selected value of lambda, we can use the `coef_tidy()` function from the `stat471` package:

```
coef_tidy(ridge_fit, s = "lambda.1se")
```

```
## # A tibble: 98 x 2
##    feature        coefficient
##    <chr>                <dbl>
##  1 (Intercept)    4336.
##  2 population        0.000102
##  3 household.size    2.87
##  4 race.pctblack     1.17
##  5 race.pctwhite    -1.20
##  6 race.pctasian    -7.83
##  7 race.pcthisp      0.774
##  8 age.pct12to21     1.67
##  9 age.pct12to29     1.31
## 10 age.pct16to24     2.02
## # i 88 more rows
```

```
coef_tidy(ridge_fit, s = "lambda.min")
```

```
## # A tibble: 98 x 2
##    feature        coefficient
##    <chr>                <dbl>
##  1 (Intercept)    5226.
##  2 population        0.000137
##  3 household.size    1.34
##  4 race.pctblack     1.42
##  5 race.pctwhite    -1.46
##  6 race.pctasian   -11.3
##  7 race.pcthisp      0.959
##  8 age.pct12to21     2.07
##  9 age.pct12to29     1.62
## 10 age.pct16to24     2.64
## # i 88 more rows
```

If `s` is not specified then `s = lambda.1se` will be chosen by default:

```
coef_tidy(ridge_fit)
```

```
## # A tibble: 98 x 2
##    feature        coefficient
##    <chr>                <dbl>
##  1 (Intercept)    4336.
##  2 population        0.000102
##  3 household.size    2.87
##  4 race.pctblack     1.17
##  5 race.pctwhite    -1.20
##  6 race.pctasian    -7.83
##  7 race.pcthisp      0.774
```

```
##  8 age.pct12to21     1.67
##  9 age.pct12to29     1.31
## 10 age.pct16to24     2.02
## # i 88 more rows
```

NOTE: `cv.glmnet()` standardized the features behind the scenes, but returns coefficients that should be interpreted in the context of the original scale of the features. For instance, a coefficient of 2.9 for `household.size` means that `violentcrimes.perpop` increases by 2.9 on average when the household size goes up by one household member.

Side note: `coef_tidy()` works for usual linear model fits as well:

```
lm_fit <- lm(violentcrimes.perpop ~ ., data = crime_train)
coef_tidy(lm_fit)
```

```
## # A tibble: 98 x 2
##    feature         coefficient
##    <chr>                 <dbl>
##  1 (Intercept)    -461686.
##  2 population          -0.0242
##  3 household.size  -26507.
##  4 race.pctblack     2670.
##  5 race.pctwhite     2152.
##  6 race.pctasian      -12.0
##  7 race.pcthisp       459.
##  8 age.pct12to21     2740.
##  9 age.pct12to29     2325.
## 10 age.pct16to24    -4122.
## # i 88 more rows
```

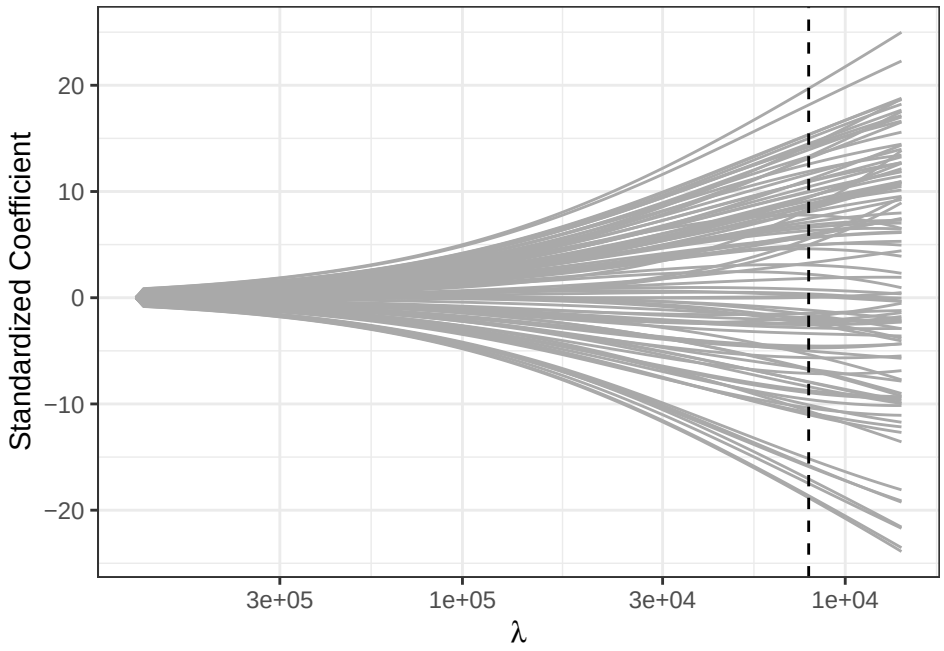
Note that some of the coefficients are undefined because $p > n$:

```
lm_fit |>
  coef_tidy() |>
  summarise(num_NA_coeffs = sum(is.na(coefficient)))
```

```
## # A tibble: 1 x 1
##   num_NA_coeffs
##           <int>
## 1            26
```

To visualize the fitted ridge coefficients as a function of lambda, we can make a trace plot like we saw in class. To do this, we can use the `plot_glmnet()` function from the `stat471` package, which by default shows a dashed line at the lambda value chosen using the one-standard-error rule:

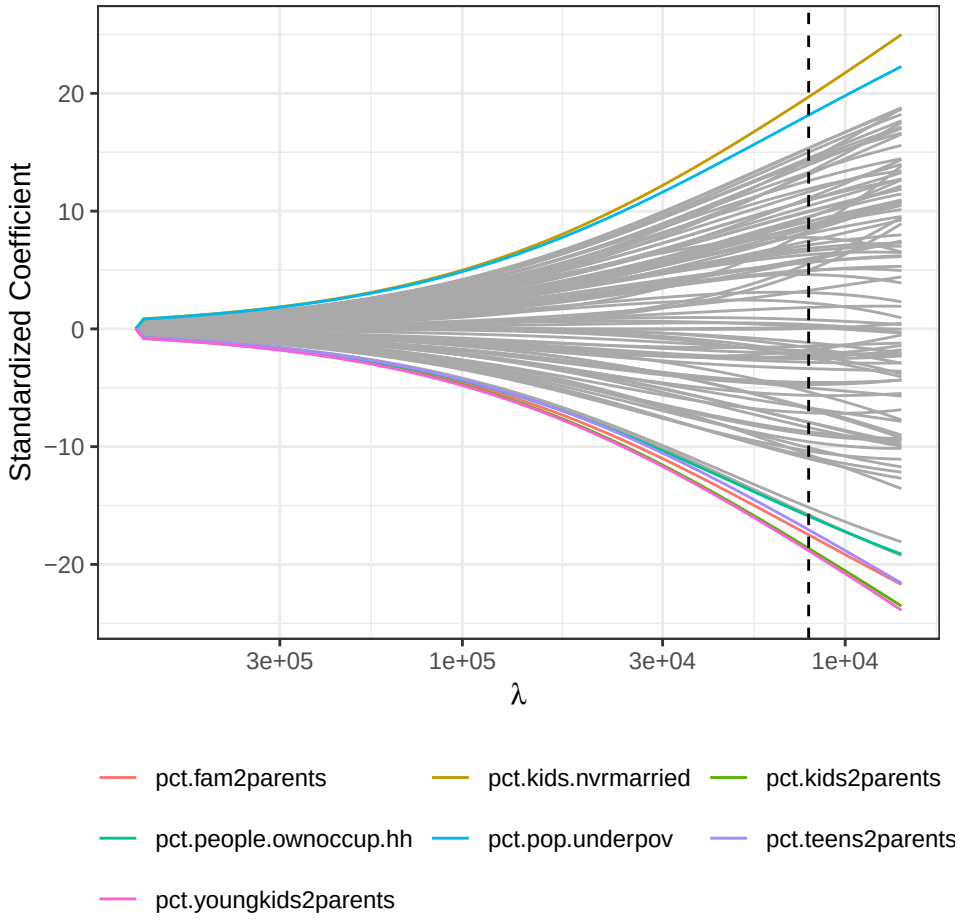
```
plot_glmnet(glmnet_fit = ridge_fit, data = crime_train)
```



NOTE: We must pass in the data as well as the fit object into `plot_glmnet()`.

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

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



NOTE: Unlike the output of `coef_tidy()`, the coefficients plotted by `plot_glmnet()` are for the *standardized* features. For example, let's see what is the standard deviation of `household.size`:

```
crime_train |>
  summarize(sd(household.size))
```

```
## # A tibble: 1 x 1
##   `sd(household.size)`
##                  <dbl>
## 1                0.273
```

So if we have a coefficient of 0.78 for the standardized version of this variable, it means that when the household size increases by one standard deviation (i.e. by 0.27 household members), the response increases by 0.8. We can translate this back to the unnormalized scale by noting that increasing the household size by one household member will increase the response by $0.8/0.27 = 2.9$, which is the unnormalized coefficient we found before.

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, newdata = crime_test, s = "lambda.1se")
ridge_predictions
```

```
##       lambda.1se
##  [1,]  1728.4273
##  [2,]  1342.5847
##  [3,]  1040.6852
##  [4,]   771.8934
##  [5,]   681.1764
##  [6,]   700.3370
##  [7,]   981.5614
##  [8,]   814.8565
##  [9,]   841.2080
## [10,]   749.3758
## [11,]   555.5066
## [12,]  1381.7639
## [13,]  1251.6341
## [14,]  1258.0162
## [15,]  1442.3838
## [16,]   710.4379
## [17,]   703.6399
## [18,]   711.3818
```

We can cast this to a vector using `as.numeric()`:

```
ridge_predictions <- as.numeric(ridge_predictions)
ridge_predictions
```

```
##  [1] 1728.4273 1342.5847 1040.6852  771.8934  681.1764  700.3370  981.5614
##  [8]  814.8565  841.2080  749.3758  555.5066 1381.7639 1251.6341 1258.0162
## [15] 1442.3838  710.4379  703.6399  711.3818
```

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

```
RMSE <- sqrt(mean((ridge_predictions - crime_test$violentcrimes.perpop)^2))
RMSE
```

```
## [1] 397.7994
```

Ridge logistic regression

We can also use `cv.glmnet()` to run a ridge-penalized logistic regression. Let's try it out on a binarized version of `crime_data`:

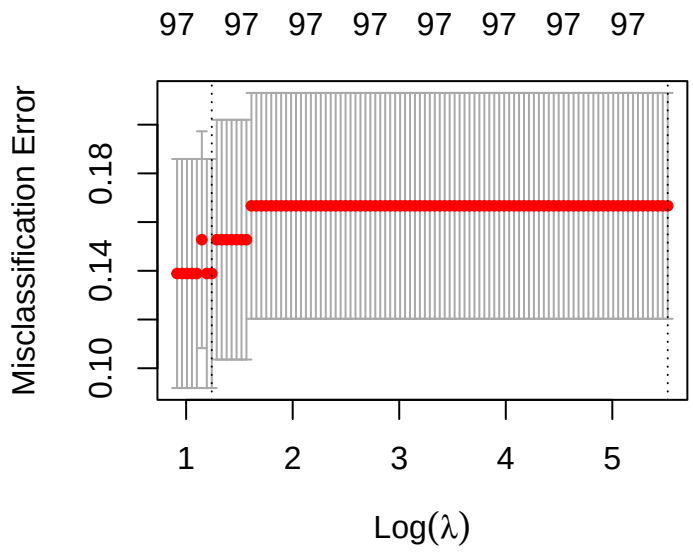
```
crime_train_binary <- crime_train |>
  mutate(high_crime = as.numeric(violentcrimes.perpop > 2000)) |>
  select(-violentcrimes.perpop)
crime_test_binary <- crime_test |>
  mutate(high_crime = as.numeric(violentcrimes.perpop > 2000)) |>
  select(-violentcrimes.perpop)
```

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.

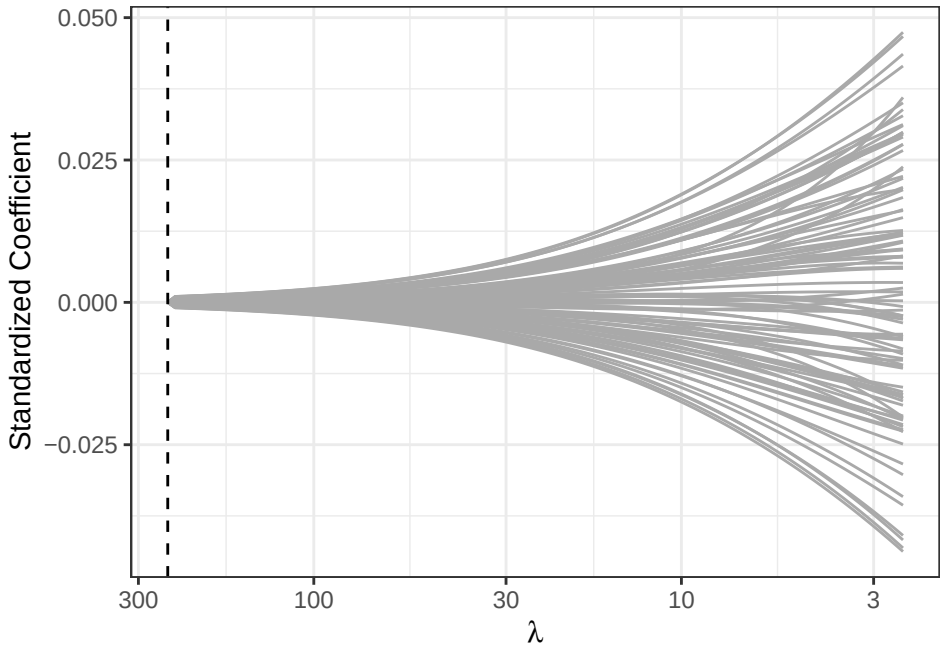
```
set.seed(471)
ridge_fit <- cv.glmnet(
  high_crime ~ .,           # formula notation, as usual
  alpha = 0,                # alpha = 0 means ridge
  nfolds = 10,              # number of CV folds
  family = "binomial",      # to specify logistic regression
  type.measure = "class",   # use misclassification error in CV
  data = crime_train_binary # train on crime_train_binary data
)
```

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

```
plot(ridge_fit)
```




```
plot_glmnet(ridge_fit, crime_train_binary)
```



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).

```
probabilities <- predict(
  ridge_fit,                    # fit object
  newdata = crime_test_binary,  # new data to test on
  s = "lambda.1se",             # which value of lambda to use
  type = "response"             # to output probabilities
) |>
  as.numeric()                  # convert to vector
head(probabilities)
```

```
## [1] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
```

We can threshold the probabilities at 0.5 to get binary predictions as we did with regular logistic regression.

Exercise: Weighted ridge logistic regression

1. What fraction of the observations in `crime_train_binary` are high crime?
2. Use `coef_tidy()` to find the ridge coefficients for the ridge logistic regression above. Why did cross-validation choose the value of lambda that gave these coefficients?
3. Use the results of the first two problems to explain why the predicted probabilities of `high_crime` obtained above are all equal to 0.1666667.
4. Let's weight the positive training observations five-fold:

   ```
   weights <- 1*(crime_train_binary$high_crime == 0) + 5*(crime_train_binary$high_crime == 1)
   ```

   Rerun a logistic ridge regression, additionally passing the above weight vector to the `weights` argument of `cv.glmnet()`. Store the result in an object called `ridge_fit_weighted`.
5. Produce the CV plot, and comment on how and why it differs from that of the unweighted logistic ridge regression.