

## Unit 3 Lecture 4: Lasso regression

October 17, 2023

In this R demo, we will learn about the `glmnetUtils` package and how to run cross-validated lasso and elastic net regressions using the `cv.glmnet()` and `cva.glmnet()` functions, respectively.

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

We will be applying lasso and elastic net regressions to study the crime data from the previous lecture

```
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.pcthispan <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:

```
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 lasso regression

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

```
lasso_fit <- cv.glmnet(
  violentcrimes.perpop ~ ., # formula notation, as usual
  alpha = 1,               # alpha = 1 for lasso
```

```

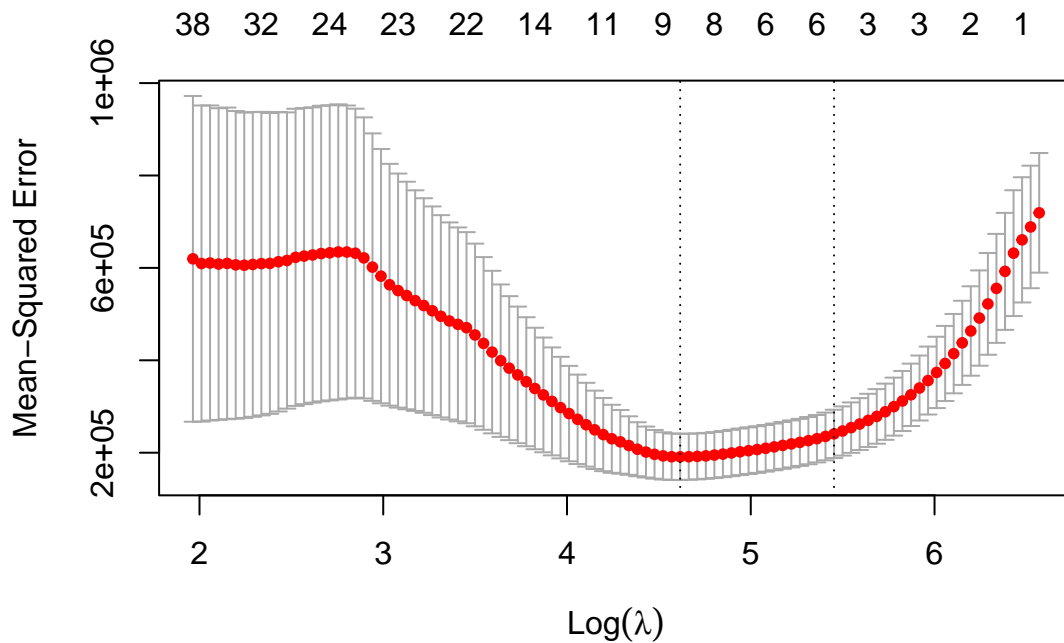
nfold = 10,          # number of folds
data = crime_train   # data to run lasso on
)

```

## Inspecting the results

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

```
plot(lasso_fit)
```



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

```
# lambda sequence
```

```
head(lasso_fit$lambda)
```

```
## [1] 713.0629 680.6531 649.7163 620.1857 591.9973 565.0901
```

```
# number of nonzero coefficients
```

```
head(lasso_fit$nzcoef)
```

```
## s0 s1 s2 s3 s4 s5
```

```
## 0 1 1 2 2 2
```

```
# CV estimates
```

```
head(lasso_fit$cvm)
```

```
## [1] 719195.4 688634.8 660698.6 631908.0 592507.2 555672.5
```

```
# CV standard errors
```

```
head(lasso_fit$cvstd)
```

```
## [1] 129486.1 132612.6 135495.0 136165.2 126567.6 117928.5
```

```
# lambda achieving minimum CV error
```

```
lasso_fit$lambda.min
```

```
## [1] 101.0748
```

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

```
## [1] 233.4959
```

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

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

```
## # A tibble: 98 x 2
##   feature      coefficient
##   <chr>         <dbl>
## 1 (Intercept)    1139.
## 2 population         0
## 3 household.size    0
## 4 race.pctblack     0
## 5 race.pctwhite     0
## 6 race.pctasian     0
## 7 race.pcthispanic 0
## 8 age.pct12to21     0
## 9 age.pct12to29     0
## 10 age.pct16to24    0
## # i 88 more rows
```

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

```
## # A tibble: 98 x 2
##   feature      coefficient
##   <chr>         <dbl>
## 1 (Intercept)    7696.
## 2 population         0
## 3 household.size    0
## 4 race.pctblack     0
## 5 race.pctwhite     0
## 6 race.pctasian     0
## 7 race.pcthispanic 0
## 8 age.pct12to21     0
## 9 age.pct12to29     0
## 10 age.pct16to24    0
## # i 88 more rows
```

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

```
coef_tidy(lasso_fit)
```

```
## # A tibble: 98 x 2
##   feature      coefficient
##   <chr>         <dbl>
## 1 (Intercept)    1139.
## 2 population         0
## 3 household.size    0
## 4 race.pctblack     0
## 5 race.pctwhite     0
## 6 race.pctasian     0
## 7 race.pcthispanic 0
## 8 age.pct12to21     0
```

```
## 9 age.pct12to29      0
## 10 age.pct16to24     0
## # i 88 more rows
```

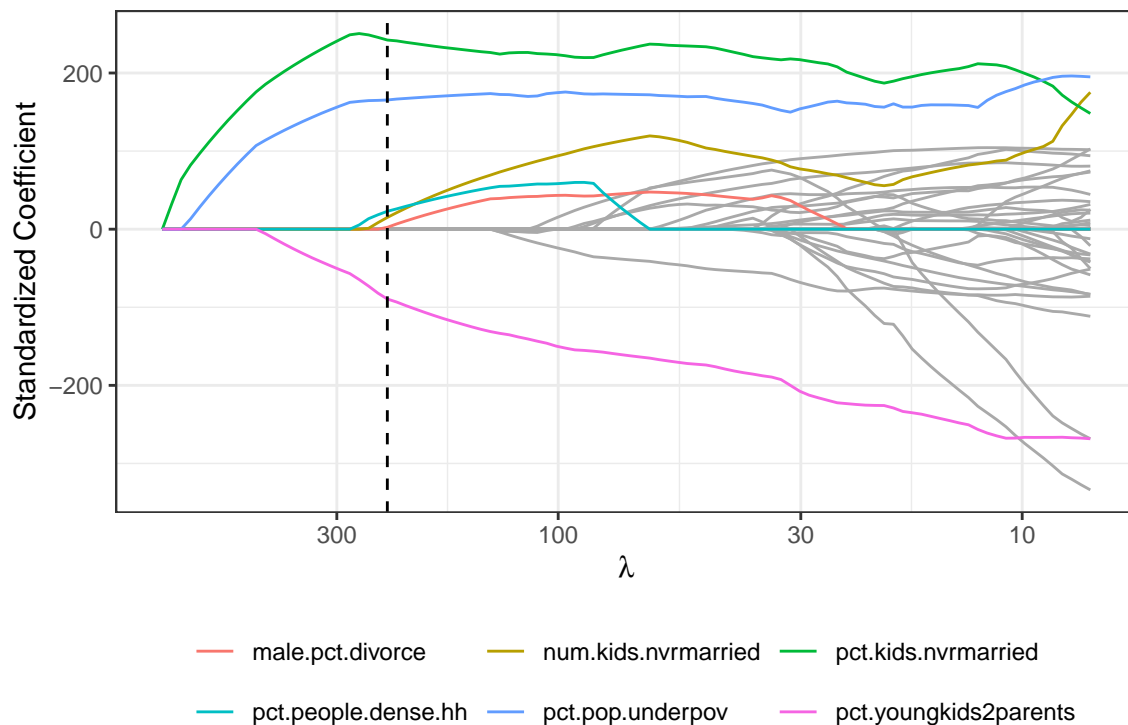
Note that these coefficient vectors are sparse. We can get a list of the nonzero standardized coefficients as follows:

```
coef_tidy(lasso_fit) |>
  filter(coefficient != 0)
```

```
## # A tibble: 7 x 2
##   feature      coefficient
##   <chr>      <dbl>
## 1 (Intercept) 1139.
## 2 pct.pop.underpov 22.7
## 3 male.pct.divorce 0.912
## 4 pct.youngkids2parents -7.27
## 5 num.kids.nvrmarried 0.00301
## 6 pct.kids.nvrmarried 72.7
## 7 pct.people.dense.hh 5.38
```

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(lasso_fit, crime_train)
```



By default, `plot_glmnet` annotates the features with nonzero coefficients. 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):

```
lasso_predictions <- predict(lasso_fit,
  newdata = crime_test,
  s = "lambda.1se"
) |> as.numeric()
lasso_predictions

## [1] 1901.3904 1331.1849 950.1509 809.1830 757.2346 756.9001 864.4810
## [8] 786.1132 820.8253 1063.3772 647.6317 1343.7530 1163.0144 1078.2869
## [15] 1366.4746 741.4528 883.7059 820.3576
```

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

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

## [1] 400.7576
```

## Elastic net regression

Next, let's run an elastic net regression. We can do this via the `cva.glmnet()` function:

```
elnet_fit <- cva.glmnet(
  violentcrimes.perpop ~ ., # formula notation, as usual
  nfolds = 10,             # number of folds
  data = crime_train       # data to run elastic net on
)
```

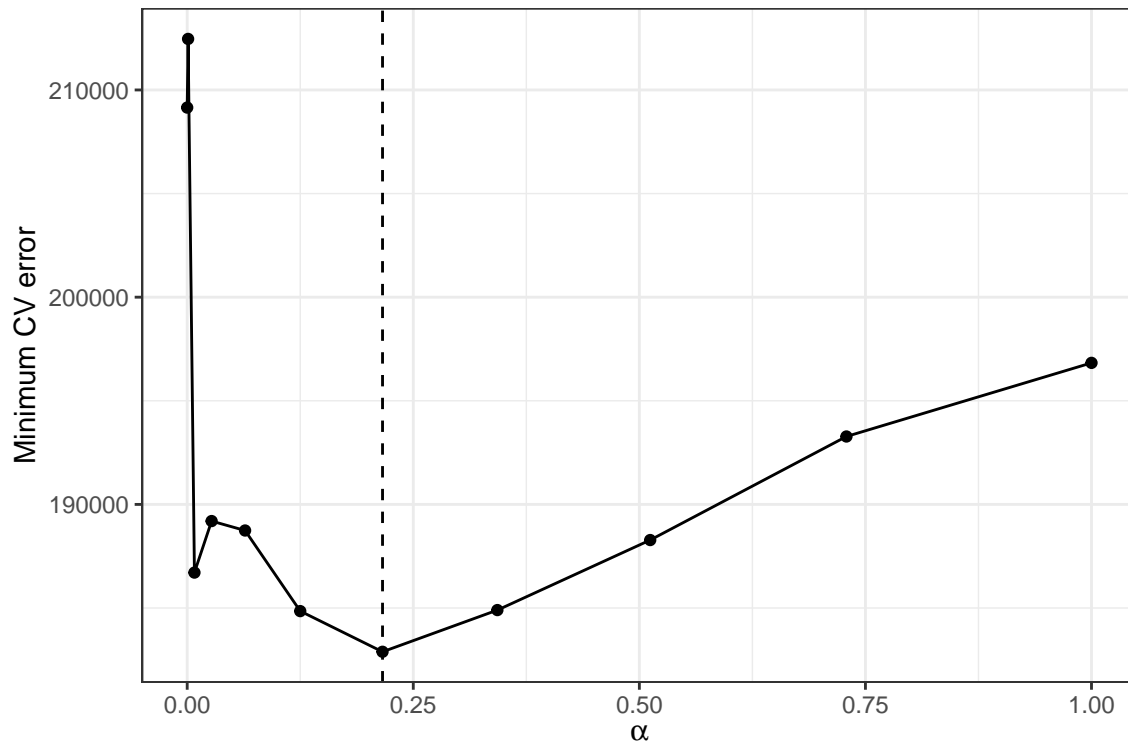
The following are the values of `alpha` that were used:

```
elnet_fit$alpha

## [1] 0.000 0.001 0.008 0.027 0.064 0.125 0.216 0.343 0.512 0.729 1.000
```

We can plot the minimum CV error for each value of `alpha` using the helper function `plot_cva_glmnet()` from `plot_glmnet.R`:

```
plot_cva_glmnet(elnet_fit)
```



We can then extract the `cv.glmnet` fit object based on the optimal `alpha` using `extract_best_elnet` from `plot_glmnet.R`:

```
elnet_fit_best <- extract_best_elnet(elnet_fit)
```

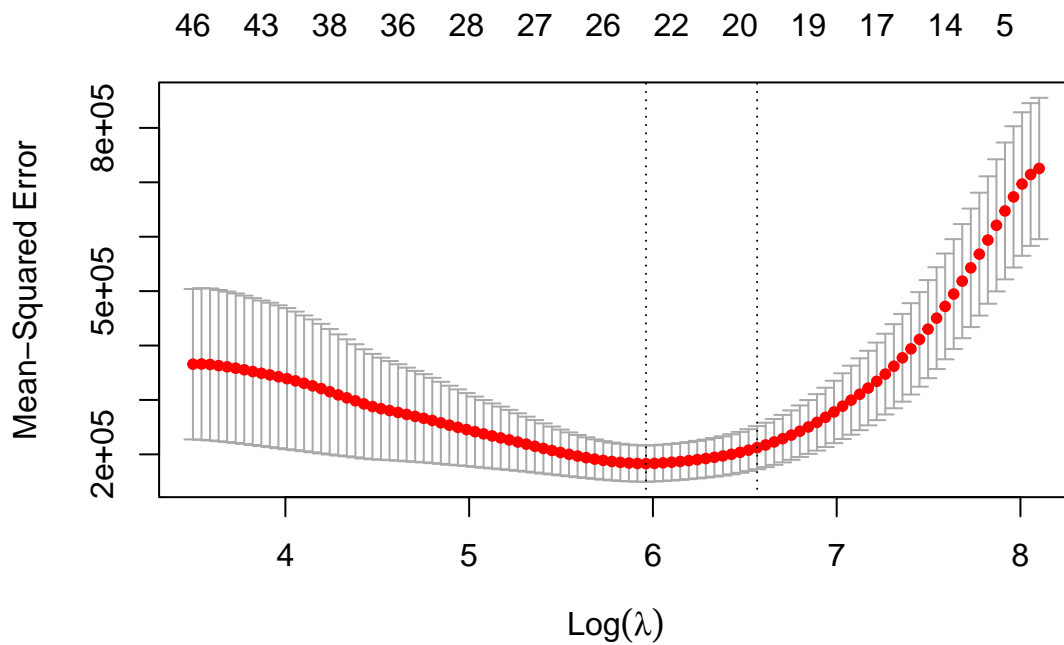
The `elnet_fit_best` object is a usual `glmnet` fit object, with an additional field called `alpha` specifying which value of `alpha` was used:

```
elnet_fit_best$alpha
```

```
## [1] 0.216
```

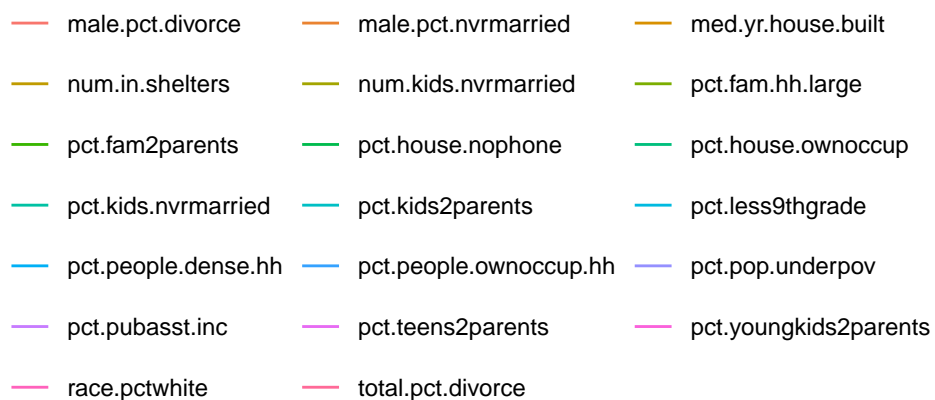
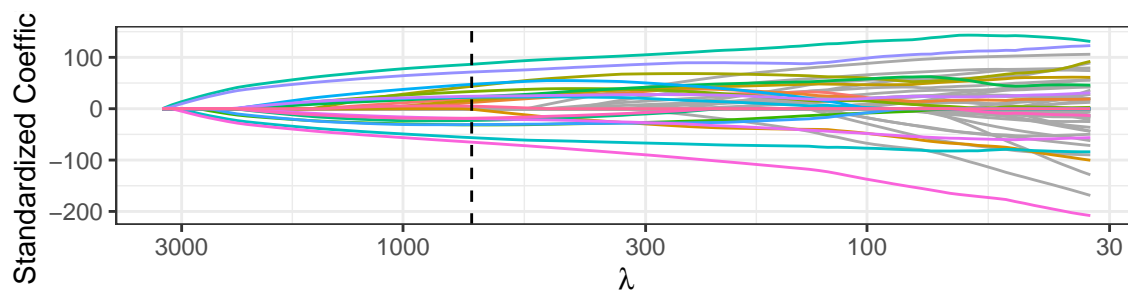
We can make a CV plot to select `lambda` as usual:

```
plot(elnet_fit_best)
```



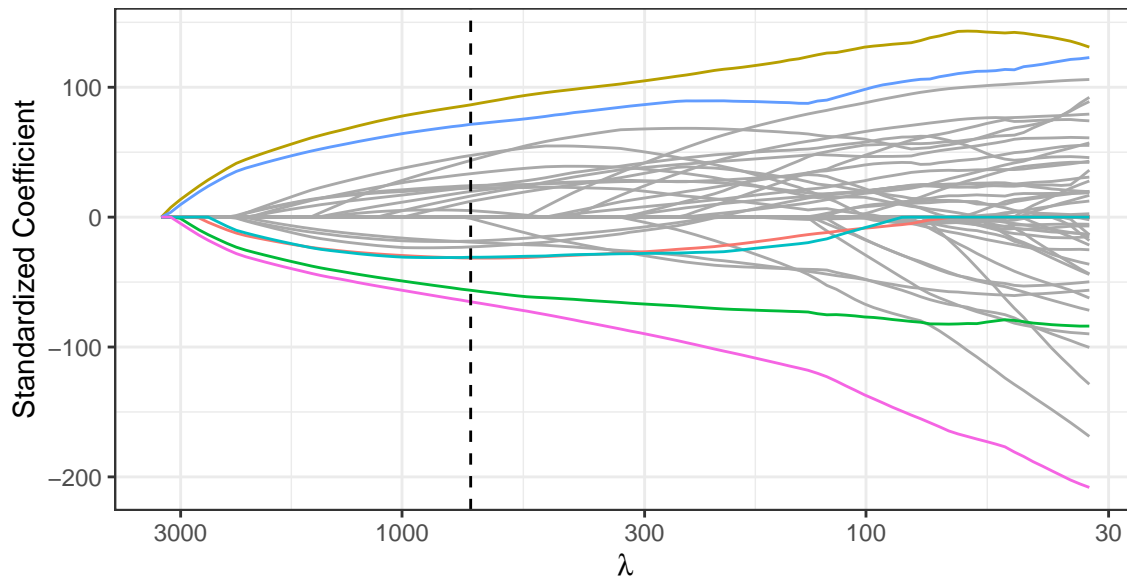
And we can make a trace plot for this optimal value of  $\alpha$ :

```
plot_glmnet(elnnet_fit_best, crime_train)
```



This is too many features to highlight, so let's choose a smaller number:

```
plot_glmnet(elnnet_fit_best, crime_train, features_to_plot = 6)
```



— pct.fam2parents      — pct.kids.nvrmarried      — pct.kids2parents  
— pct.people.ownoccup.hh      — pct.pop.underpov      — pct.youngkids2parents

We can make predictions and evaluate test error using the `elnet_fit_best` object:

```

elnet_predictions <- predict(elnet_fit,
  alpha = elnet_fit$alpha,
  newdata = crime_test,
  s = "lambda.1se"
) |> as.numeric()
elnet_predictions

```

```

## [1] 1737.2357 1344.0327 1036.0706 762.1868 671.2751 690.3775 975.6859
## [8] 807.5988 833.6900 744.3694 545.3223 1384.0474 1250.4805 1258.0862
## [15] 1445.2206 702.8391 693.6531 701.8896

```

```

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

```

```
## [1] 392.1493
```

## Lasso logistic regression

We can also run a lasso-penalized logistic regression. Let's try it out on a binarized version of the crime data:

```

# redefine response based on whether violentcrimes.perpop is above the median
crime_binary_train <- crime_train |>
  mutate(
    violentcrimes.perpop =
      as.numeric(violentcrimes.perpop > median(violentcrimes.perpop))
  )
crime_binary_test <- crime_test |>
  mutate(
    violentcrimes.perpop =
      as.numeric(violentcrimes.perpop > median(violentcrimes.perpop))
  )

```



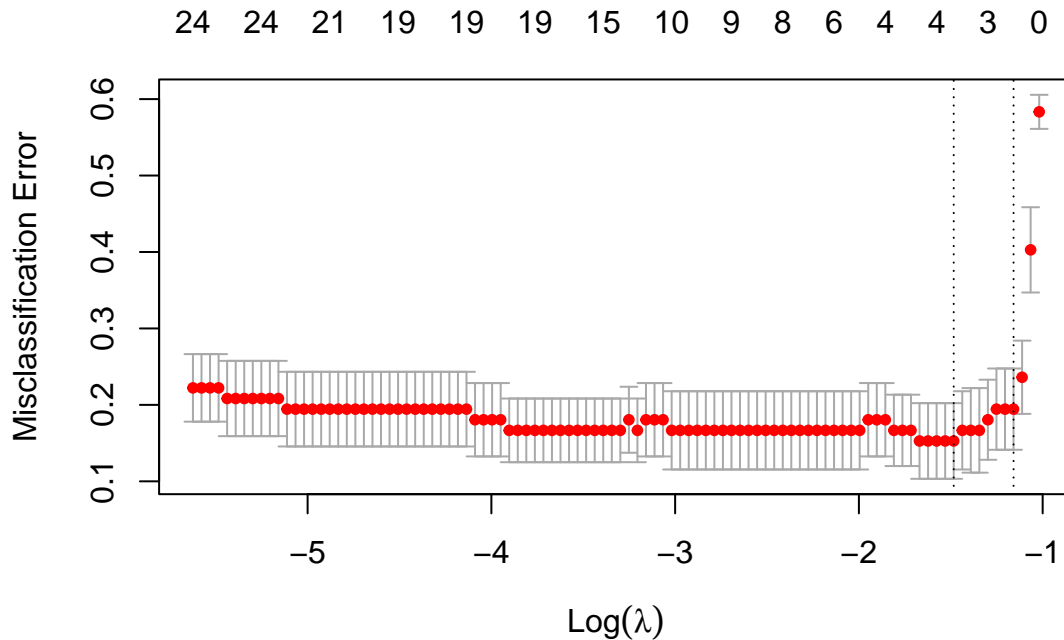
```
)
```

To run the logistic lasso 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.

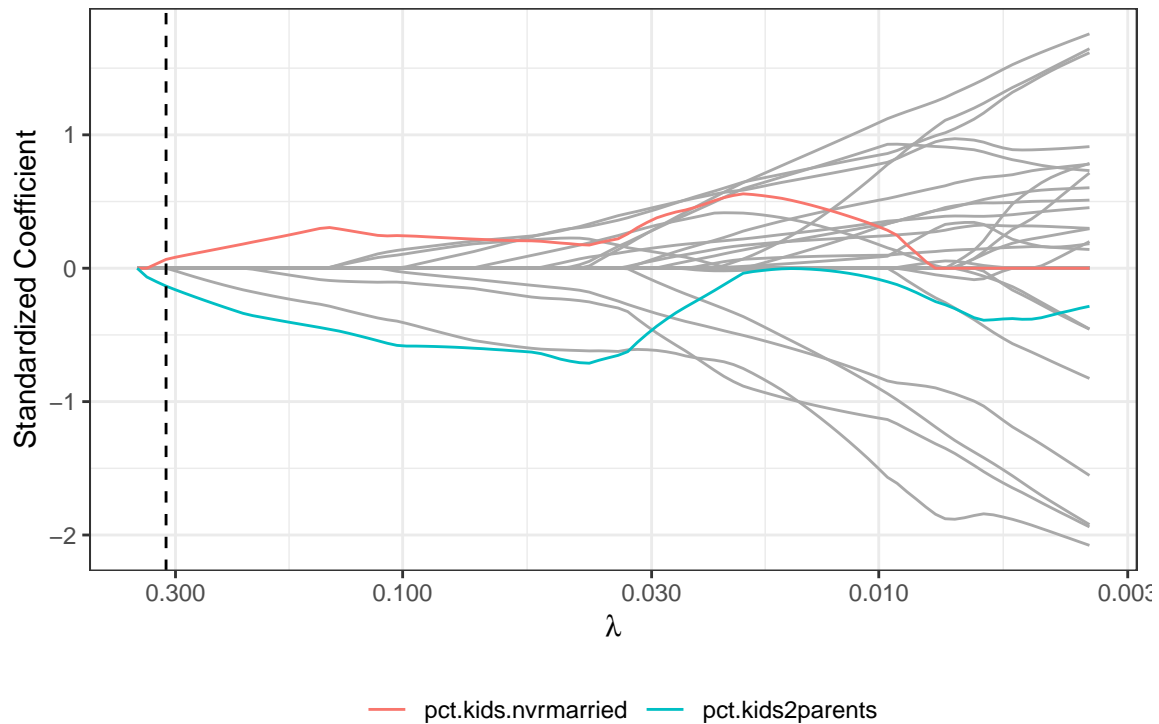
```
lasso_fit <- cv.glmnet(
  violentcrimes.perpop ~ .,      # formula notation, as usual
  alpha = 1,                    # alpha = 0 means lasso
  nfolds = 10,                  # number of CV folds
  family = "binomial",          # logistic regression
  type.measure = "class",       # use misclassification error
  data = crime_binary_train     # train on crime_binary_train data
)
```

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

```
plot(lasso_fit)
```



```
plot_glmnet(lasso_fit, crime_binary_train)
```



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

```
## [1] 0.5750745 0.5179763 0.4617836 0.4392386 0.4409267 0.4460029
```

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