Supervised Learning

Regularization

June 26th, 2023

Previously...

We talked about variable selection in the linear model context:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p$$

Why would we attempt to select a **subset** of the p variables?

- *To improve model interpretability*
  - **Occam's razor:** simplest model wins!
  - Eliminating uninformative predictors is obviously a good thing when your goal is to tell the story of how your predictors are associated with your response.
- *To improve prediction accuracy*
  - Eliminating uninformative predictors can lead to lower variance in the test-set MSE, at the expense of a slight increase in bias

Best subset selection

- Start with the **null model** $\mathcal{M}_0$ (intercept-only) that has no predictors
    - just predicts the sample mean for each observation
- For $k = 1, 2, \ldots, p$ (each possible number of predictors)
    - Fit **all** $\binom{p}{k} = \frac{p!}{k!(p-k)!}$ with exactly k predictors
    - Pick the best (some criteria) among these $\binom{p}{k}$ models, call it $\mathcal{M}_k$
    - Best can be up to the user: cross-validation error, highest adjusted R^2, etc.
- Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$

This is not typically used in research!

- only practical for a smaller number of variables

Remember the bias-variance tradeoff

$$\text{MSE} = (\text{Bias})^2 + \text{Variance}$$

- Introduce bias but decrease variance to improve predictions
- *Some questions*
    - How do we know that there is a *trade-off* between Bias and Variance?
    - How can we check that it is $(\text{Bias})^2$ and not (Bias) without doing *any* calcs?

Shrinkage methods: Ridge regression

Ridge regression introduces a **shrinkage penalty** $\lambda \geq 0$ by minimizing:

$$\sum_{i}^{n} \left(Y_i - \beta_0 - \sum_{j}^{p} \beta_j x_{ij}\right)^2 + \lambda \sum_{j}^{p} \beta_j^2 = \text{RSS} + \lambda \sum_{j}^{p} \beta_j^2$$

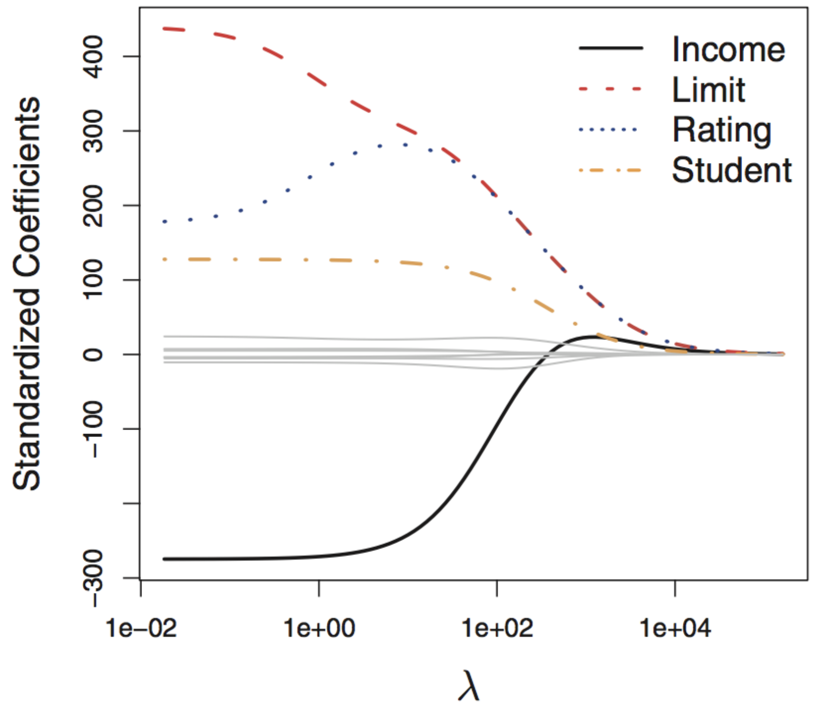
or more succinctly we want to minimize:

$$\|\mathbf{Y} - \mathbf{X}\beta\|_2^2 + \lambda \|\beta\|_2^2$$

- as λ increases $\Rightarrow$ flexibility of models decreases
  - **increases bias, but decreases variance**
- for fixed value of λ, ridge regression fits only a single model
  - need to use cross-validation to **tune** λ

Shrinkage methods: Ridge regression

For example: note how the magnitude of the coefficient for `Income` trends as $\lambda \to \infty$



The coefficient **shrinks towards zero**, but never actually reaches it

- `Income` is always a variable in the learned model, regardless of the value of λ

Shrinkage methods: Lasso regression

Ridge regression **keeps all variables**

But we may believe there is a **sparse** solution

Lasso enables variable selection with λ by minimizing:

$$\sum_{i}^{n}\left(Y_i-\beta_0-\sum_{j}^{p}\beta_j X_{ij}\right)^2+\lambda\sum_{j}^{p}|\beta_j| = \mathrm{RSS}+\lambda\sum_{j}^{p}|\beta_j|$$

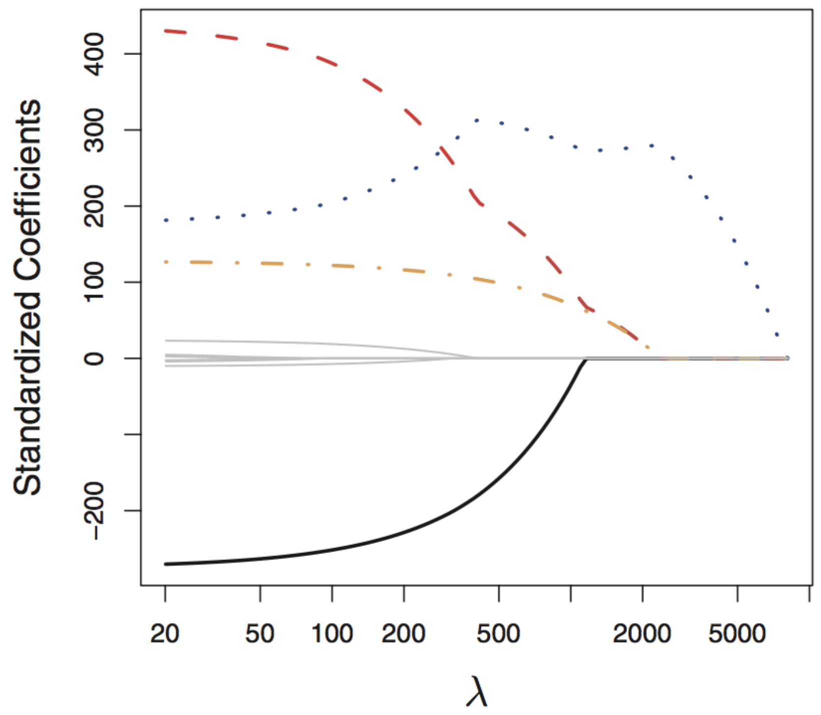
or more succinctly we want to minimize:

$$\|\mathbf{Y}-\mathbf{X}\beta\|_2^2+\lambda\|\beta\|_1$$

- Lasso uses an ℓ_1 ("ell 1") penalty
- as λ increases $\Rightarrow$ flexibility of models decreases
    - **increases bias, but decreases variance**
- Can handle the $p > n$ case, i.e. more variables than observations!

Shrinkage methods: Lasso regression

Lasso regression **performs variable selection** yielding **sparse** models



The coefficient shrinks towards and **eventually equals zero** at $\lambda \approx 1000$

- if the optimum value of λ is larger, then `Income` would NOT be included in the learned model

Which do we use?

Best of both worlds? Elastic net

$$\sum_{i}^{n}\left(Y_i - \beta_0 - \sum_{j}^{p} \beta_j X_{ij}\right)^2 + \lambda\left[(1-\alpha)\|\beta\|_2^2/2 + \alpha\|\beta\|_1\right]$$

- $||\beta||_1$ is the ℓ_1 norm: $||\beta||_1 = \sum_j^p |\beta_j|$
- $||\beta||_2$ is the ℓ_2, Euclidean, norm: $||\beta||_2 = \sqrt{\sum_j^p \beta_j^2}$
- Ridge penalty: $\lambda \cdot (1-\alpha)/2$
- Lasso penalty: $\lambda \cdot \alpha$
- α controls the **mixing** between the two types, ranges from 0 to 1
  - $\alpha = 1$ returns lasso
  - $\alpha = 0$ return ridge

Caveats to consider...

- For either ridge, lasso, or elastic net: **you should standardize your data**
- Common convention: within each column, compute then subtract off the sample mean, and compute the divide off the sample standard deviation:

$$\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{s_{x,j}}$$

- `glmnet` package does this by default and reports coefficients on the original scale
- λ and α are **tuning parameters**
- Have to select appropriate values based on test data / cross-validation
- When using `glmnet`, the `cv.glmnet()` function will perform the cross-validation for you

Example data: NFL teams summary

Created dataset using `nflfastR` summarizing NFL team performances from 1999 to 2020

```r
library(tidyverse)
nfl_teams_data <- read_csv("https://shorturl.at/uwAV2")
nfl_model_data <- nfl_teams_data %>%
  mutate(score_diff = points_scored - points_allowed) %>%
  # Only use rows with air yards
  filter(season >= 2006) %>%
  dplyr::select(-wins, -losses, -ties, -points_scored, -points_allowed, -season, -team)
```

Introduction to `glmnet`

We will use the `glmnet` package for ridge, lasso, and elastic net

```r
library(glmnet)
```

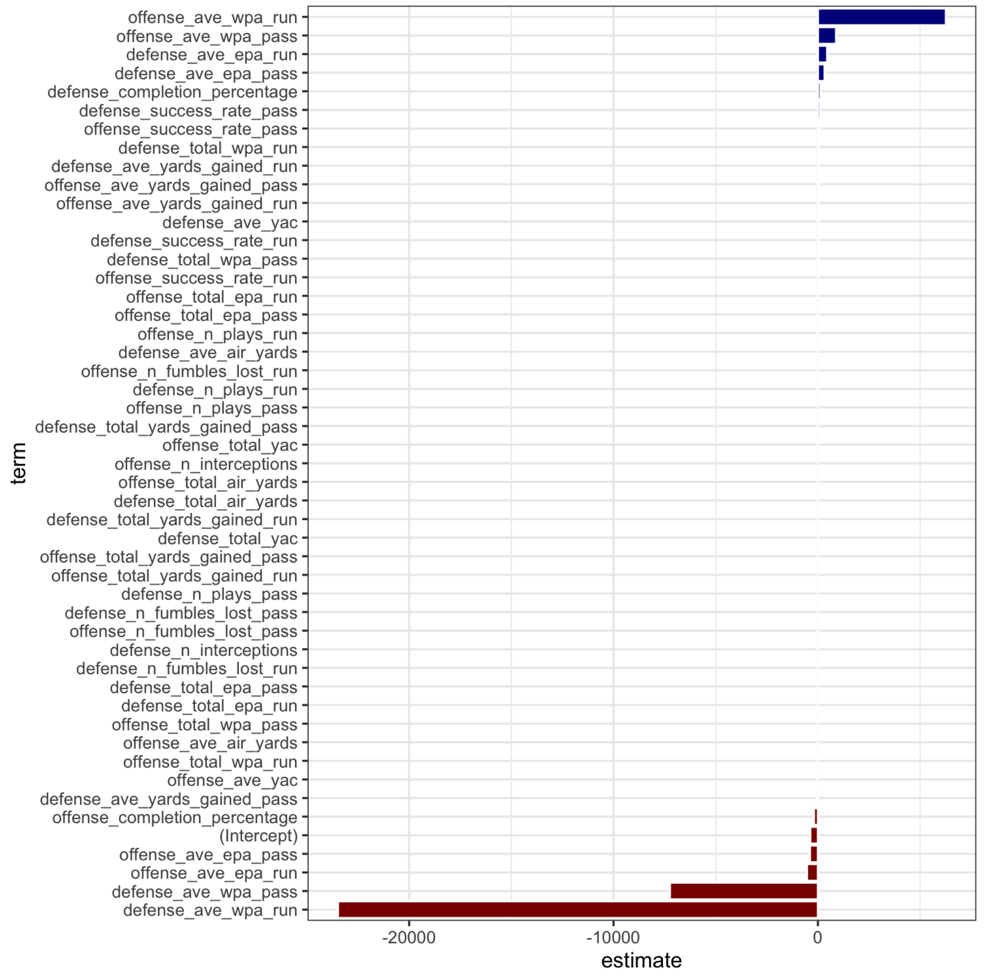
- could use the `model.matrix()` function (which converts factors to 0-1 dummy variables!)

```r
model_x <- nfl_model_data %>%
  dplyr::select(-score_diff) %>%
  as.matrix()
model_y <- nfl_model_data$score_diff
# model_x <- model.matrix(score_diff ~ ., nfl_model_data)[, -1]
```

Initial model with `lm()`

- What do the initial regression coefficients look like?
- Use `broom` to tidy model output for plotting

```
init_reg_fit <- lm(score_diff ~ ., nfl_model_
library(broom)
tidy(init_reg_fit) %>%
  mutate(coef_sign = as.factor(sign(estimate)
         term = fct_reorder(term, estimate))
  ggplot(aes(x = term, y = estimate, fill = c
  geom_bar(stat = "identity", color = "white"
  scale_fill_manual(values = c("darkred", "da
                    guide = FALSE) +
  coord_flip() + theme_bw()
```

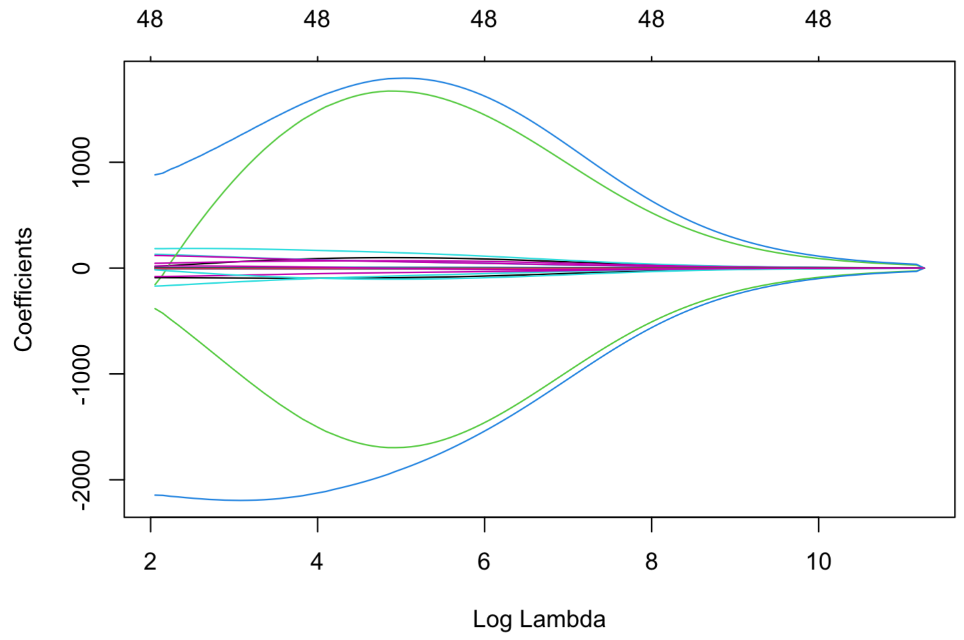


Ridge regression example

Perform ridge regression using `glmnet` with `alpha = 0` (more on that later)

By default it standardizes your predictors and fits model across a range of λ values (can plot these!)

```
init_ridge_fit <- glmnet(model_x, model_y, alpha = 0)
plot(init_ridge_fit, xvar = "lambda")
```

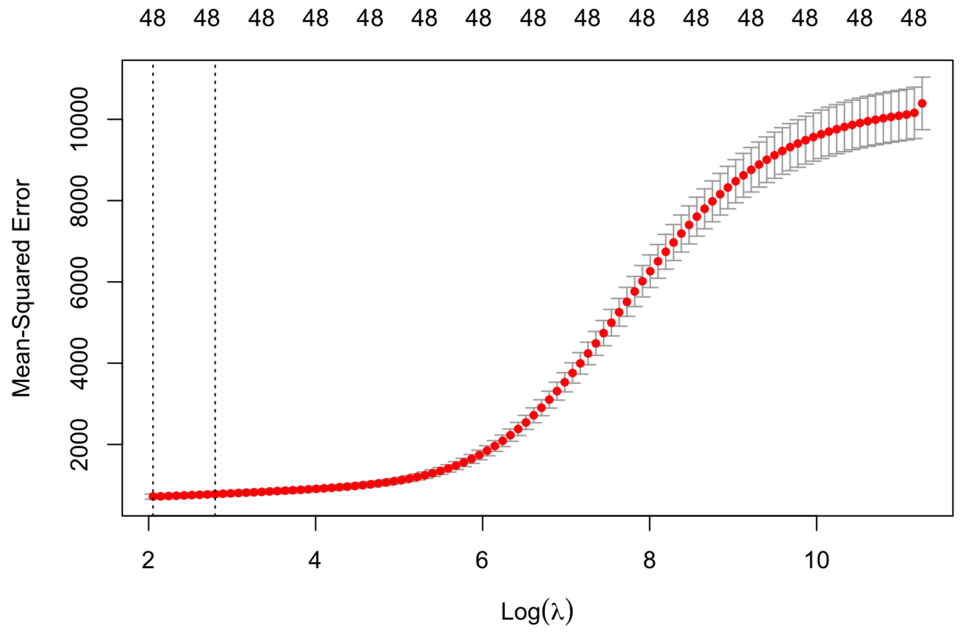


Ridge regression example

We use cross-validation to select λ with `cv.glmnet()` which uses 10-folds by default

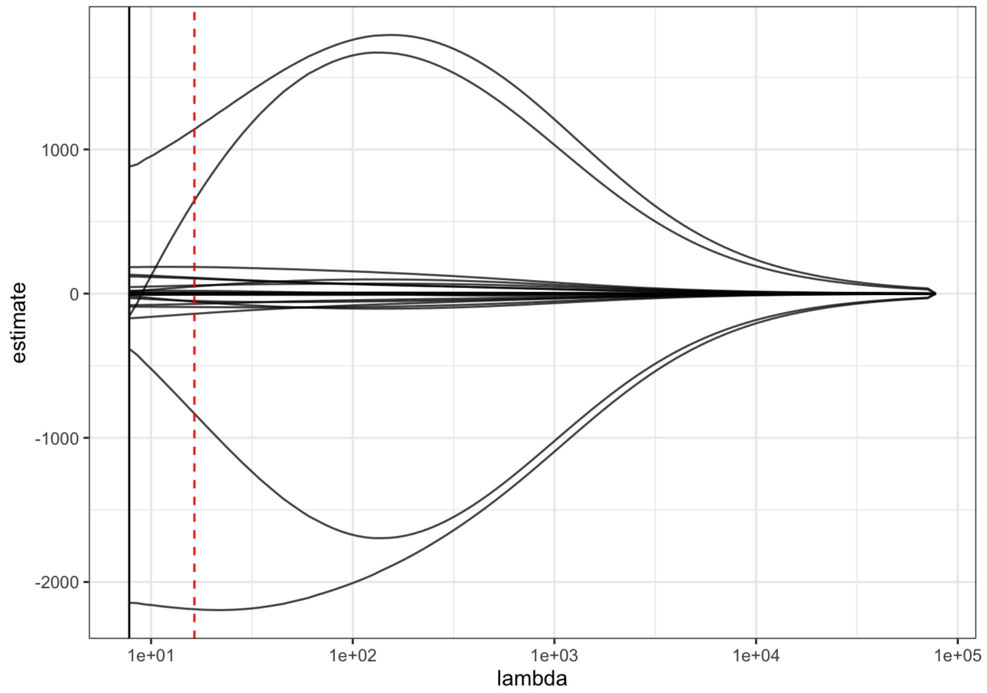
- specify ridge regression with `alpha = 0`

```
fit_ridge_cv <- cv.glmnet(model_x, model_y, alpha = 0)
plot(fit_ridge_cv)
```



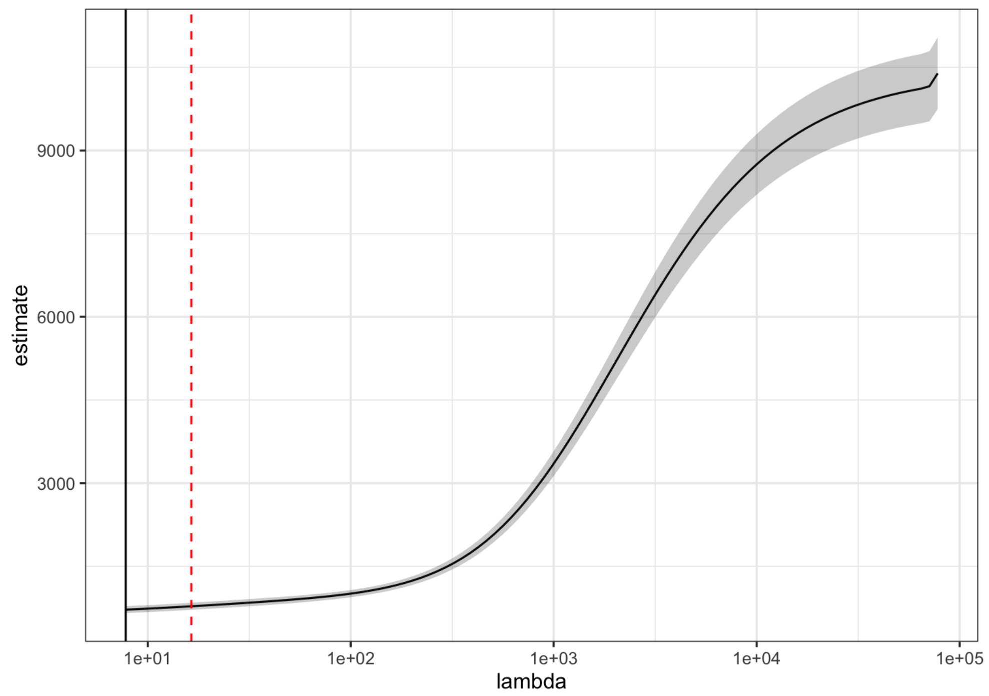
Tidy ridge regression

```
tidy_ridge_coef <- tidy(fit_ridge_cv$glmnet.f
tidy_ridge_coef %>%
  ggplot(aes(x = lambda, y = estimate,
             group = term)) +
  scale_x_log10() +
  geom_line(alpha = 0.75) +
  geom_vline(xintercept =
               fit_ridge_cv$lambda.min) +
  geom_vline(xintercept =
               fit_ridge_cv$lambda.1se,
             linetype = "dashed", color = "re
  theme_bw()
```



Tidy ridge regression

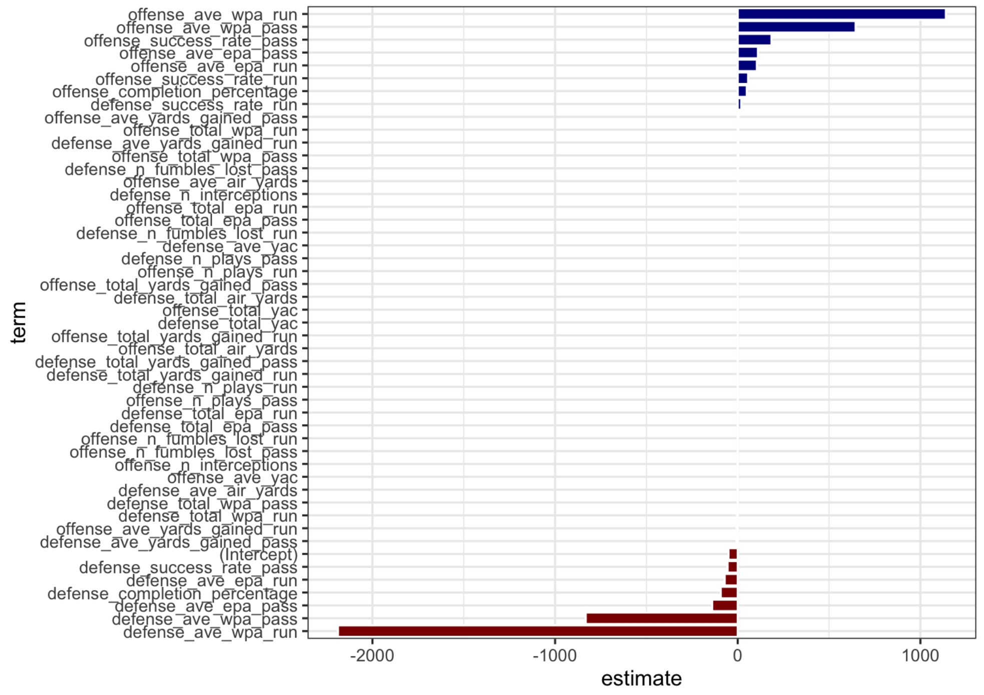
```
tidy_ridge_cv <- tidy(fit_ridge_cv)
tidy_ridge_cv %>%
  ggplot(aes(x = lambda, y = estimate)) +
  geom_line() + scale_x_log10() +
  geom_ribbon(aes(ymin = conf.low,
                  ymax = conf.high), alpha =
  geom_vline(xintercept =
               fit_ridge_cv$lambda.min) +
  geom_vline(xintercept =
               fit_ridge_cv$lambda.1se,
             linetype = "dashed", color = "re
  theme_bw()
```



Ridge regression coefficients

Coefficients using the **1 standard error rule** λ

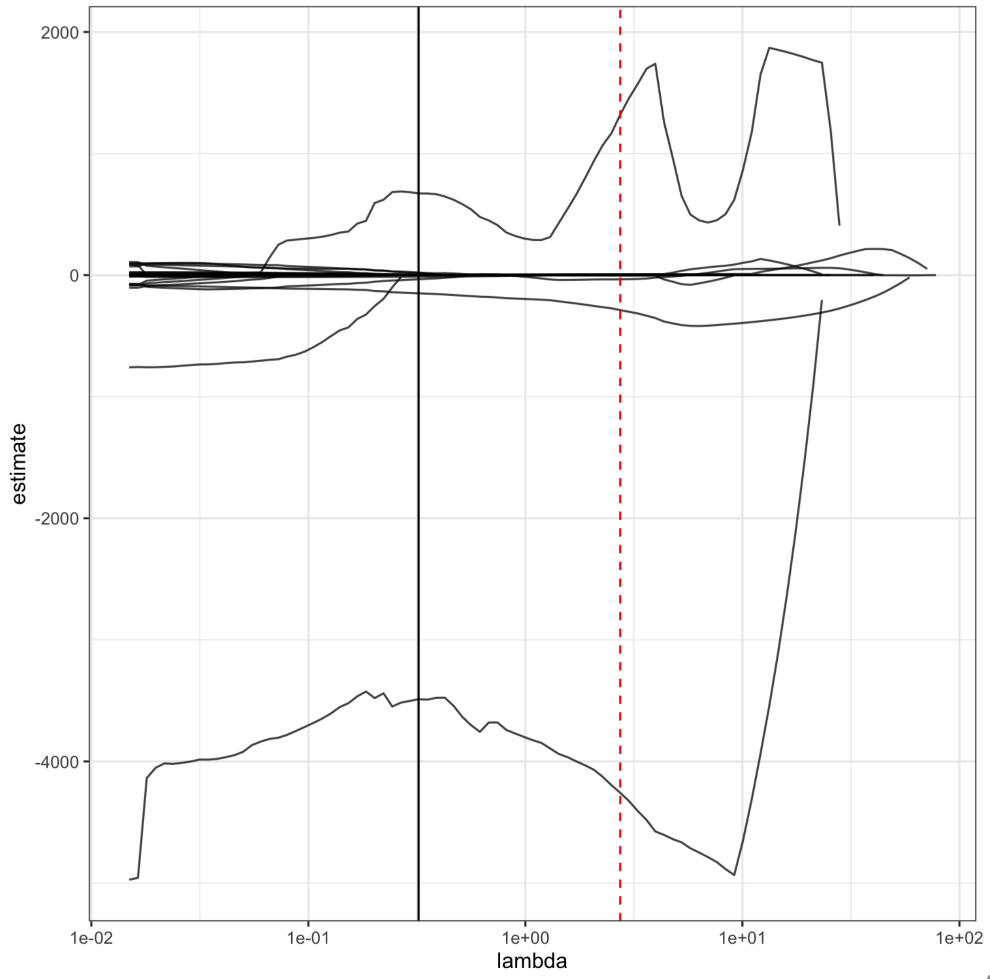
```
tidy_ridge_coef %>%
  filter(lambda == fit_ridge_cv$lambda.1se) %
  mutate(coef_sign = as.factor(sign(estimate)
         term = fct_reorder(term, estimate))
  ggplot(aes(x = term, y = estimate, fill = c
  geom_bar(stat = "identity", color = "white"
  scale_fill_manual(values = c("darkred", "da
  coord_flip() + theme_bw()
```



Lasso regression example

Similar syntax to ridge but specify `alpha = 1`:

```
fit_lasso_cv <- cv.glmnet(model_x, model_y,
                          alpha = 1)
tidy_lasso_coef <- tidy(fit_lasso_cv$glmnet.f
tidy_lasso_coef %>%
  ggplot(aes(x = lambda, y = estimate,
             group = term)) +
  scale_x_log10() +
  geom_line(alpha = 0.75) +
  geom_vline(xintercept =
               fit_lasso_cv$lambda.min) +
  geom_vline(xintercept =
               fit_lasso_cv$lambda.1se,
             linetype = "dashed", color = "re
  theme_bw()
```

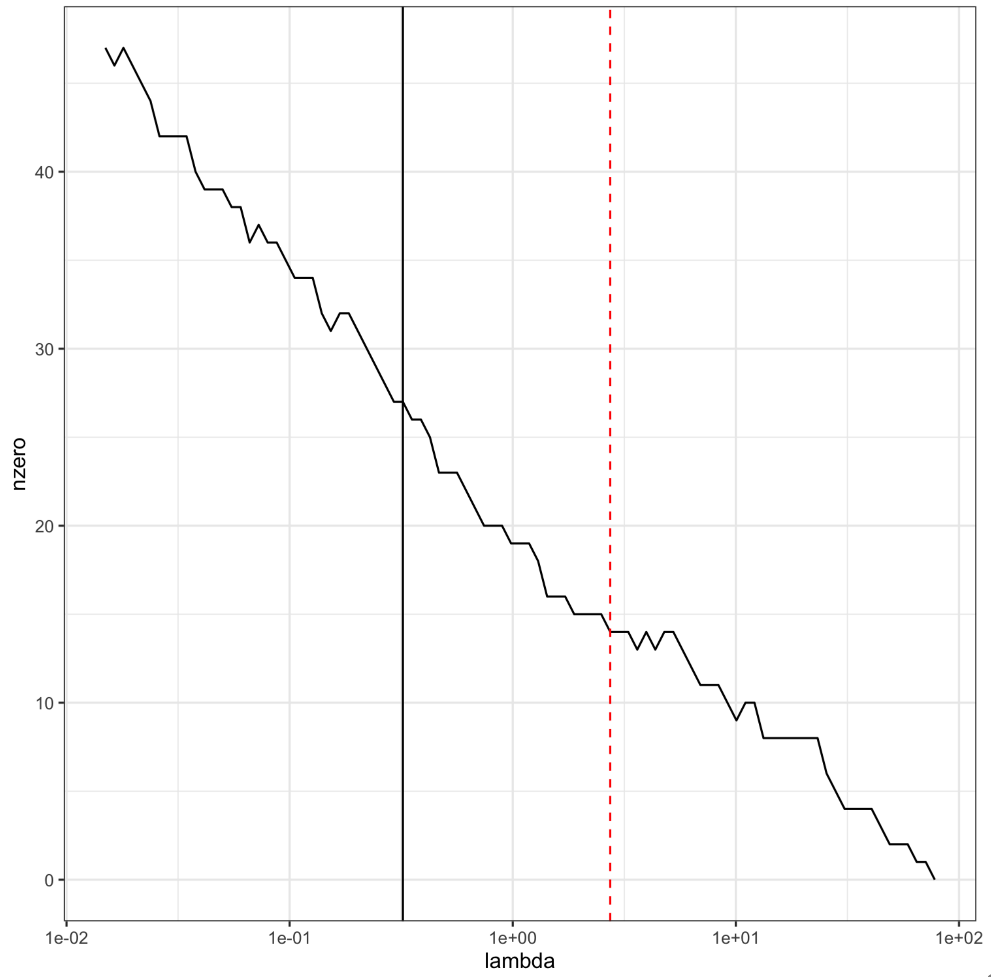


Lasso regression example

Number of non-zero predictors by λ

```
tidy_lasso_cv <- tidy(fit_lasso_cv)
tidy_lasso_cv %>%
  ggplot(aes(x = lambda, y = nzero)) +
  geom_line() +
  geom_vline(xintercept = fit_lasso_cv$lambda
  geom_vline(xintercept = fit_lasso_cv$lambda
             linetype = "dashed", color = "re
  scale_x_log10() + theme_bw()
```

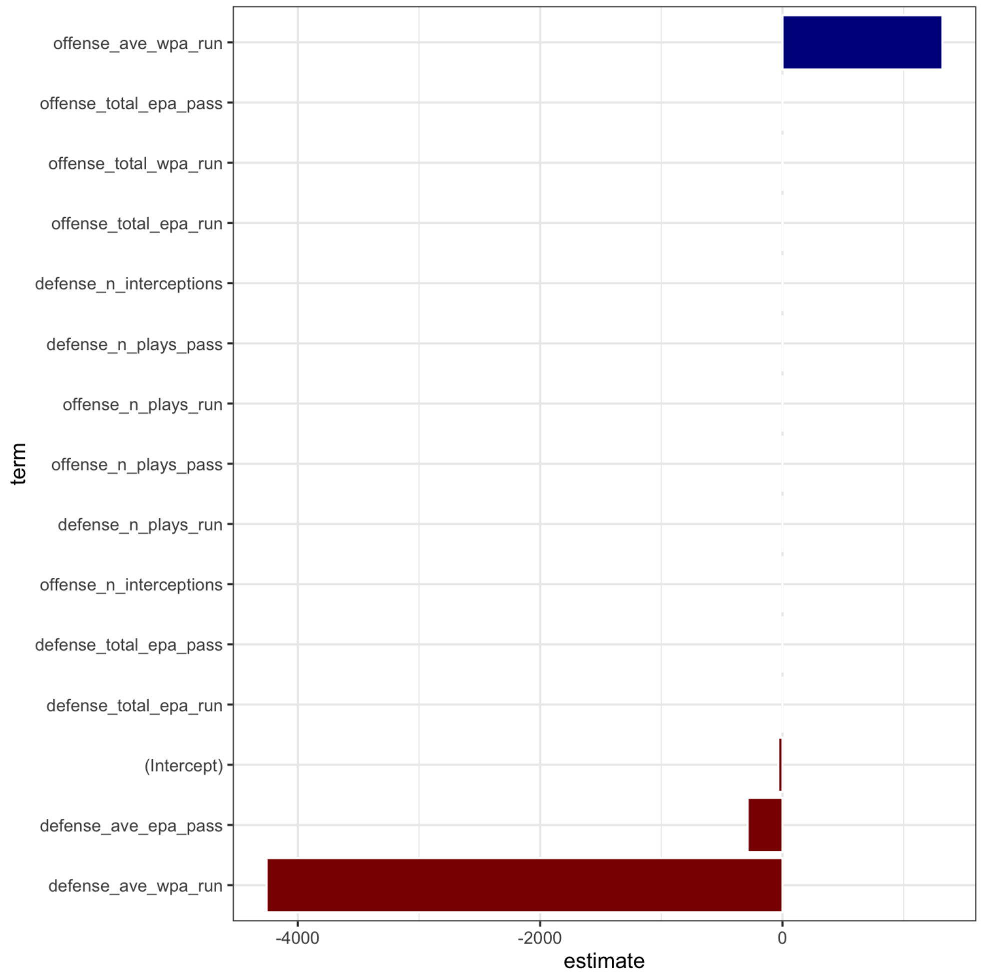
Reduction in variables using **1 standard error rule** λ



Lasso regression example

Coefficients using the **1 standard error rule** λ

```
tidy_lasso_coef %>%
  filter(lambda == fit_lasso_cv$lambda.1se) %
  mutate(coef_sign = as.factor(sign(estimate)
         term = fct_reorder(term, estimate))
  ggplot(aes(x = term, y = estimate,
             fill = coef_sign)) +
  geom_bar(stat = "identity", color = "white"
  scale_fill_manual(values = c("darkred", "da
                    guide = FALSE) +
  coord_flip() +
  theme_bw()
```



Elastic net example

Need to tune both λ and α - can do so manually with our own folds

```r
set.seed(2020)
fold_id <- sample(rep(1:10, length.out = nrow(model_x)))
```

Then use cross-validation with these folds for different candidate `alpha` values:

```r
cv_en_25 <- cv.glmnet(model_x, model_y, foldid = fold_id, alpha = .25)
cv_en_50 <- cv.glmnet(model_x, model_y, foldid = fold_id, alpha = .5)
cv_ridge <- cv.glmnet(model_x, model_y, foldid = fold_id, alpha = 0)
cv_lasso <- cv.glmnet(model_x, model_y, foldid = fold_id, alpha = 1)
```

Can see which one had the lowest CV error among its candidate λ values:

```r
which.min(c(min(cv_en_25$cvm), min(cv_en_50$cvm), min(cv_ridge$cvm), min(cv_lasso$cvm)))
```

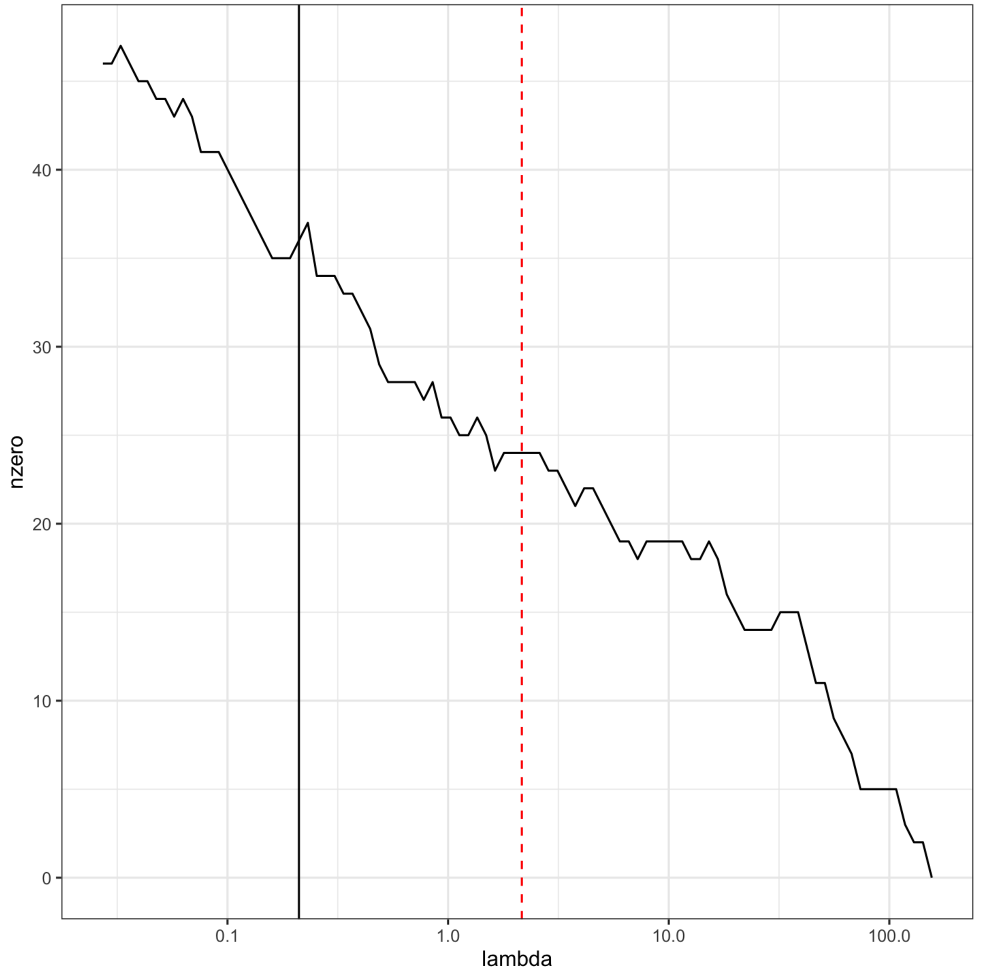
```
## [1] 2
```

Elastic net example

Can view same type of summary

```r
tidy(cv_en_50) %>%
  ggplot(aes(x = lambda, y = nzero)) +
  geom_line() +
  geom_vline(xintercept = cv_en_50$lambda.min
  geom_vline(xintercept = cv_en_50$lambda.1se
             linetype = "dashed",
             color = "red") +
  scale_x_log10() +
  theme_bw()
```

- More relaxed than lasso for variable entry



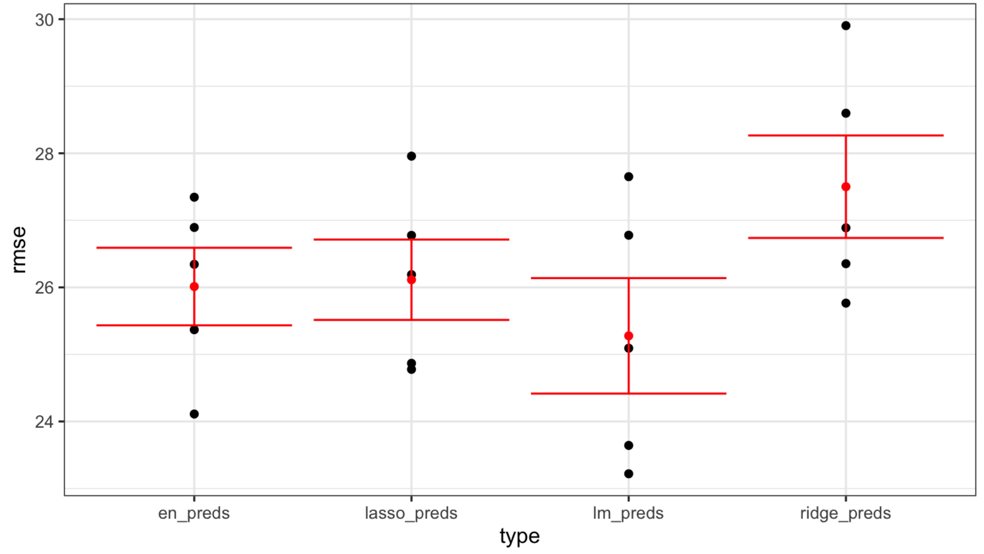
Comparison of models based on holdout performance

```r
set.seed(2020)
nfl_model_data <- nfl_model_data %>% mutate(test_fold = sample(rep(1:5, length.out = n())))
holdout_predictions <-
  map_dfr(unique(nfl_model_data$test_fold),
          function(holdout) {
            # Separate test and training data:
            test_data <- nfl_model_data %>% filter(test_fold == holdout)
            train_data <- nfl_model_data %>% filter(test_fold != holdout)

            # Repeat for matrices
            test_x <- as.matrix(dplyr::select(test_data, -score_diff))
            train_x <- as.matrix(dplyr::select(train_data, -score_diff))

            # Train models:
            lm_model <- lm(score_diff ~ ., data = train_data)
            ridge_model <- cv.glmnet(train_x, train_data$score_diff, alpha = 0)
            lasso_model <- cv.glmnet(train_x, train_data$score_diff, alpha = 1)
            en_model <- cv.glmnet(train_x, train_data$score_diff, alpha = .5)

            # Return tibble of holdout results:
            tibble(lm_preds = predict(lm_model, newdata = test_data),
                   ridge_preds = as.numeric(predict(ridge_model, newx = test_x)),
                   lasso_preds = as.numeric(predict(lasso_model, newx = test_x)),
                   en_preds = as.numeric(predict(en_model, newx = test_x)),
                   test_actual = test_data$score_diff, test_fold = holdout)
          })
```

Predictions compared to `lm`?

Compute RMSE across folds with std error intervals

```
holdout_predictions %>%
  pivot_longer(lm_preds:en_preds,
               names_to = "type", values_to =
  group_by(type, test_fold) %>%
  summarize(rmse =
              sqrt(mean((test_actual - test_p
  ggplot(aes(x = type, y = rmse)) +
  geom_point() + theme_bw() +
  stat_summary(fun = mean, geom = "point",
               color = "red") +
  stat_summary(fun.data = mean_se, geom = "er
               color = "red")
```



In this case `lm` actually "beat" regularization, but within intervals