Lecture 005

Shrinkage methods

Edward Rubin

Admin

Admin

Material

Last time

- Linear regression
- Model selection
  - Best subset selection
  - Stepwise selection (forward/backward)

Today

- `tidymodels`
- Shrinkage methods

Admin

Upcoming

Readings

- *Today* *ISL* Ch. 6
- *Next* *ISL* 4

Problem sets Soon!

Shrinkage methods

Intro

Recap: **Subset-selection methods** (last time)

1. algorithmically search for the "best" subset of our p predictors
2. estimate the linear models via least squares

Shrinkage methods

Intro

Recap: **Subset-selection methods** (last time)

1. algorithmically search for the "best" subset of our p predictors
2. estimate the linear models via least squares

These methods assume we need to choose a model before we fit it...

Intro

Recap: **Subset-selection methods** (last time)

1. algorithmically search for the "best" subset of our p predictors
2. estimate the linear models via least squares

These methods assume we need to choose a model before we fit it...

Alternative approach: **Shrinkage methods**

- fit a model that contains all p predictors
- simultaneously: shrink† coefficients toward zero

† Synonyms for *shrink*: constrain or regularize

Shrinkage methods

Intro

Recap: **Subset-selection methods** (last time)

1. algorithmically search for the "best" subset of our p predictors
2. estimate the linear models via least squares

These methods assume we need to choose a model before we fit it...

Alternative approach: **Shrinkage methods**

- fit a model that contains all p predictors
- simultaneously: shrink† coefficients toward zero

Idea: Penalize the model for coefficients as they move away from zero.

† Synonyms for *shrink*: constrain or regularize

Shrinkage methods

Why?

Q How could shrinking coefficients toward zero help our predictions?

Shrinkage methods

Why?

Q How could shrinking coefficients toward zero help our predictions?

A Remember we're generally facing a tradeoff between bias and variance.

Shrinkage methods

Why?

Q How could shrinking coefficients toward zero help our predictions?

A Remember we're generally facing a tradeoff between bias and variance.

- Shrinking our coefficients toward zero **reduces the model's variance**.[†]
- **Penalizing** our model for **larger coefficients** shrinks them toward zero.
- The **optimal penalty** will balance reduced variance with increased bias.

† Imagine the extreme case: a model whose coefficients are all zeros has no variance.

Shrinkage methods

Why?

Q How could shrinking coefficients toward zero help our predictions?

A Remember we're generally facing a tradeoff between bias and variance.

- Shrinking our coefficients toward zero **reduces the model's variance**.[†]
- **Penalizing** our model for **larger coefficients** shrinks them toward zero.
- The **optimal penalty** will balance reduced variance with increased bias.

Now you understand shrinkage methods.

- Ridge regression
- Lasso
- Elasticnet

† Imagine the extreme case: a model whose coefficients are all zeros has no variance.

Ridge regression

Back to least squares (again)

Recall Least-squares regression gets $\hat{\beta}_j$'s by minimizing RSS, *i.e.*,

$$\min_{\hat{\beta}} \text{RSS} = \min_{\hat{\beta}} \sum_{i=1}^{n} e_i^2 = \min_{\hat{\beta}} \sum_{i=1}^{n} \left(y_i - \underbrace{\left[\hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + \cdots + \hat{\beta}_p x_{i,p} \right]}_{=\hat{y}_i} \right)^2$$

Ridge regression

Back to least squares (again)

Recall Least-squares regression gets $\hat{\beta}_j$'s by minimizing RSS, *i.e.*,

$$\min_{\hat{\beta}} \text{RSS} = \min_{\hat{\beta}} \sum_{i=1}^{n} e_i^2 = \min_{\hat{\beta}} \sum_{i=1}^{n} \left(y_i - \underbrace{\left[\hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + \cdots + \hat{\beta}_p x_{i,p} \right]}_{=\hat{y}_i} \right)^2$$

Ridge regression makes a small change

- adds a shrinkage penalty = the sum of squared coefficents $\left(\lambda \sum_j \beta_j^2\right)$
- minimizes the (weighted) sum of RSS and the shrinkage penalty

Ridge regression

Back to least squares (again)

Recall Least-squares regression gets $\hat{\beta}_j$'s by minimizing RSS, *i.e.*,

$$\min_{\hat{\beta}} \text{RSS} = \min_{\hat{\beta}} \sum_{i=1}^{n} e_i^2 = \min_{\hat{\beta}} \sum_{i=1}^{n} \left(y_i - \underbrace{\left[\hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + \cdots + \hat{\beta}_p x_{i,p} \right]}_{=\hat{y}_i} \right)^2$$

Ridge regression makes a small change

- adds a shrinkage penalty = the sum of squared coefficents $\left(\lambda \sum_j \beta_j^2 \right)$
- minimizes the (weighted) sum of RSS and the shrinkage penalty

$$\min_{\hat{\beta}^R} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Ridge regression

Ridge regression

$$\min_{\hat{\beta}^R} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Least squares

$$\min_{\hat{\beta}} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2$$

λ (≥ 0) is a tuning parameter for the harshness of the penalty.

$\lambda = 0$ implies no penalty: we are back to least squares.

Ridge regression

Ridge regression

$$\min_{\hat{\beta}^R} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Least squares

$$\min_{\hat{\beta}} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2$$

λ (≥ 0) is a tuning parameter for the harshness of the penalty.

$\lambda = 0$ implies no penalty: we are back to least squares.

Each value of λ produces a new set of coefficents.

Ridge regression

Ridge regression

$$\min_{\hat{\beta}^R} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Least squares

$$\min_{\hat{\beta}} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2$$

λ (≥ 0) is a tuning parameter for the harshness of the penalty.

$\lambda = 0$ implies no penalty: we are back to least squares.

Each value of λ produces a new set of coefficents.

Ridge's approach to the bias-variance tradeoff: Balance

- reducing **RSS**, *i.e.*, $\sum_i \left(y_i - \hat{y}_i \right)^2$
- reducing **coefficients** (ignoring the intercept)

λ determines how much ridge "cares about" these two quantities.†

† With $\lambda = 0$, least-squares regression only "cares about" RSS.

λ and penalization

Choosing a *good* value for λ is key.

- If λ is too small, then our model is essentially back to OLS.
- If λ is too large, then we shrink all of our coefficients too close to zero.

Ridge regression

λ and penalization

Choosing a *good* value for λ is key.

- If λ is too small, then our model is essentially back to OLS.
- If λ is too large, then we shrink all of our coefficients too close to zero.

Q So what do we do?

Ridge regression

λ and penalization

Choosing a *good* value for λ is key.

- If λ is too small, then our model is essentially back to OLS.
- If λ is too large, then we shrink all of our coefficients too close to zero.

Q So what do we do?

A Cross validate!

(You saw that coming, right?)

Ridge regression

Penalization

Note Because we sum the **squared** coefficients, we penalize increasing *big* coefficients much more than increasing *small* coefficients.

Example For a value of β, we pay a penalty of $2\lambda\beta$ for a small increase.[†]

- At $\beta = 0$, the penalty for a small increase is 0.
- At $\beta = 1$, the penalty for a small increase is 2λ.
- At $\beta = 2$, the penalty for a small increase is 4λ.
- At $\beta = 3$, the penalty for a small increase is 6λ.
- At $\beta = 10$, the penalty for a small increase is 20λ.

Now you see why we call it *shrinkage*: it encourages small coefficients.

† This quantity comes from taking the derivative of $\lambda\beta^2$ with respect to β.

Ridge regression

Penalization and standardization

Important Predictors' **units** can drastically **affect ridge regression results**.

Why?

Penalization and standardization

Important Predictors' **units** can drastically **affect ridge regression results**.

Why? Because $\mathbf{x}_j$'s units affect β_j, and ridge is very sensitive to β_j.

Ridge regression

Penalization and standardization

Important Predictors' **units** can drastically **affect ridge regression results**.

Why? Because $\mathbf{x}_j$'s units affect β_j, and ridge is very sensitive to β_j.

Example Let x_1 denote distance.

Least-squares regression

If x_1 is *meters* and $\beta_1 = \mathbf{3}$, then when x_1 is *km*, $\beta_1 = \mathbf{3,000}$.

The scale/units of predictors do not affect least squares' estimates.

Penalization and standardization

Important Predictors' **units** can drastically **affect ridge regression results**.

Why? Because $\mathbf{x}_j$'s units affect β_j, and ridge is very sensitive to β_j.

Example Let x_1 denote distance.

Least-squares regression

If x_1 is *meters* and $\beta_1 = \mathbf{3}$, then when x_1 is *km*, $\beta_1 = \mathbf{3,000}$.
The scale/units of predictors do not affect least squares' estimates.

Ridge regression pays a much larger penalty for $\beta_1 = \mathbf{3,000}$ than $\beta_1 = \mathbf{3}$.
You will not get the same (scaled) estimates when you change units.

Ridge regression

Penalization and standardization

Important Predictors' **units** can drastically **affect ridge regression results**.

Why? Because $\mathbf{x}_j$'s units affect β_j, and ridge is very sensitive to β_j.

Example Let x_1 denote distance.

Least-squares regression

If x_1 is *meters* and $\beta_1 = \mathbf{3}$, then when x_1 is *km*, $\beta_1 = \mathbf{3,000}$.

The scale/units of predictors do not affect least squares' estimates.

Ridge regression pays a much larger penalty for $\beta_1 = \mathbf{3,000}$ than $\beta_1 = \mathbf{3}$.

You will not get the same (scaled) estimates when you change units.

Solution Standardize your variables, *i.e.*, `x_stnd = (x - mean(x))/sd(x)`.

Penalization and standardization

Important Predictors' **units** can drastically **affect ridge regression results**.

Why? Because $\mathbf{x}_j$'s units affect β_j, and ridge is very sensitive to β_j.

Example Let x_1 denote distance.

Least-squares regression

If x_1 is *meters* and $\beta_1 = 3$, then when x_1 is *km*, $\beta_1 = 3,000$.
The scale/units of predictors do not affect least squares' estimates.

Ridge regression pays a much larger penalty for $\beta_1 = 3,000$ than $\beta_1 = 3$.
You will not get the same (scaled) estimates when you change units.

Solution Standardize your variables, *i.e.*, `recipes::step_normalize()`.

Ridge regression

Example

Let's return to the credit dataset—and pre-processing with `tidymodels`.

Recall We have 11 predictors and a numeric outcome `balance`.

We can standardize our **predictors** using `step_normalize()` from `recipes`:

```
# Load the credit dataset
credit_df = ISLR::Credit %>% clean_names()
# Processing recipe: Define ID, standardize, create dummies, rename (lowercase)
credit_recipe = credit_df %>% recipe(balance ~ .) %>%
  update_role(id, new_role = "id variable") %>%
  step_normalize(all_predictors() & all_numeric()) %>%
  step_dummy(all_predictors() & all_nominal()) %>%
  step_rename_at(everything(), fn = str_to_lower)
# Time to juice
credit_clean = credit_recipe %>% prep() %>% juice()
```

Ridge regression

Example

For ridge regression[†] in R, we will use `glmnet()` from the `glmnet` package.

The **key arguments** for `glmnet()` are

- `x` a **matrix** of predictors
- `y` outcome variable as a vector
- `standardize` (`T` or `F`)
- `alpha` elasticnet parameter
  - `alpha=0` gives ridge
  - `alpha=1` gives lasso
- `lambda` tuning parameter (sequence of numbers)
- `nlambda` alternatively, R picks a sequence of values for λ

† And lasso!

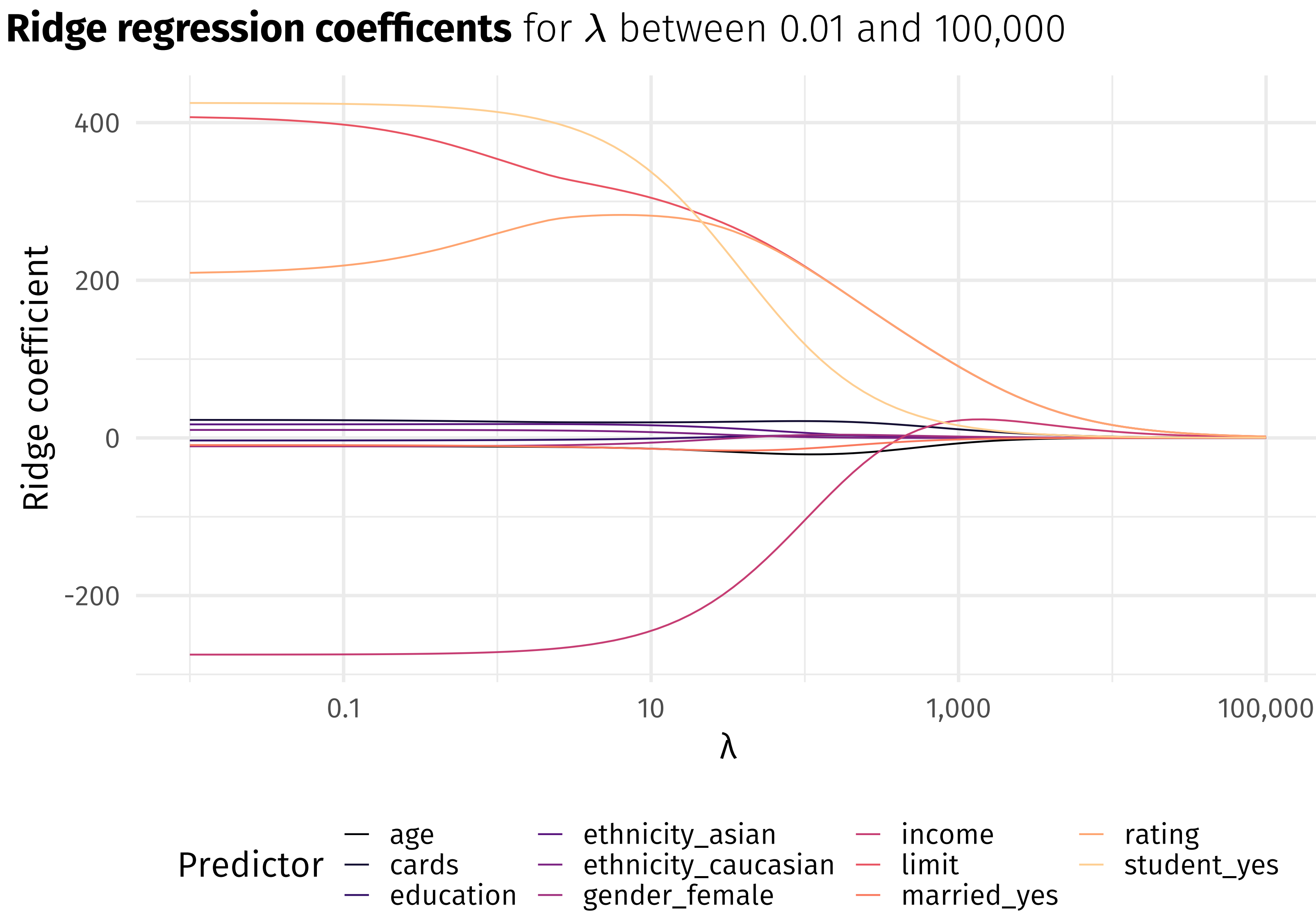
Ridge regression

Example

We just need to define a decreasing sequence for λ, and then we're set.

```
# Define our range of lambdas (glmnet wants decreasing range)
lambdas = 10^seq(from = 5, to = -2, length = 100)
# Fit ridge regression
est_ridge = glmnet(
  x = credit_clean %>% dplyr::select(-balance, -id) %>% as.matrix(),
  y = credit_clean$balance,
  standardize = F,
  alpha = 0,
  lambda = lambdas
)
```

The `glmnet` output (`est_ridge` here) contains estimated coefficients for λ. You can use `predict()` to get coefficients for additional values of λ.



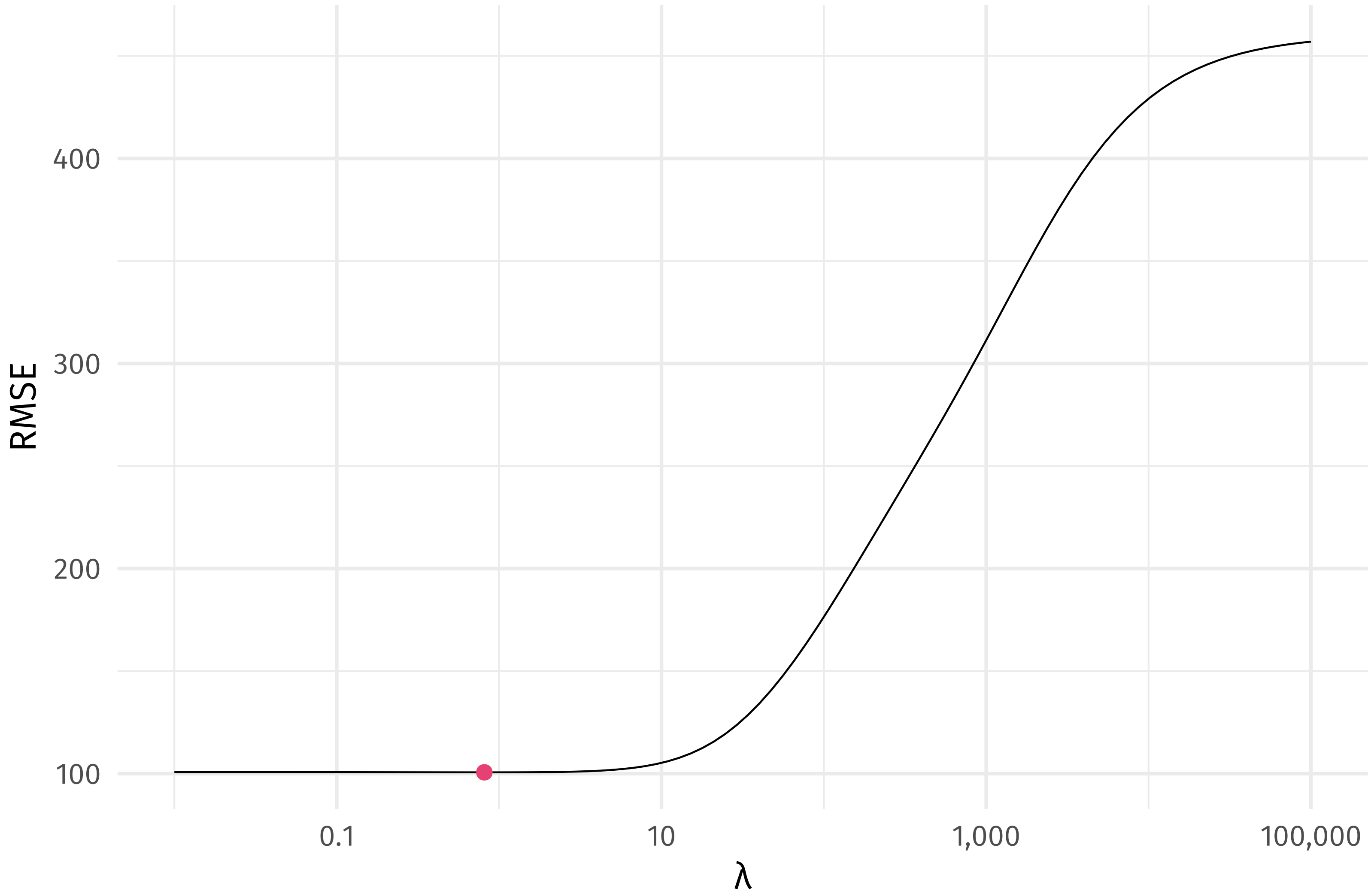
Ridge regression coefficents for λ between 0.01 and 100,000
Ridge coefficient
400
200
0
-200
0.1
10
1,000
100,000
λ
Predictor
age
cards
education
ethnicity_asian
ethnicity_caucasian
gender_female
income
limit
married_yes
rating
student_yes

Example

`glmnet` also provides convenient cross-validation function: `cv.glmnet()`.

```
# Define our lambdas
lambdas = 10^seq(from = 5, to = -2, length = 100)
# Cross validation
ridge_cv = cv.glmnet(
  x = credit_clean %>% dplyr::select(-balance, -id) %>% as.matrix(),
  y = credit_clean$balance,
  alpha = 0,
  standardize = F,
  lambda = lambdas,
  # New: How we make decisions and number of folds
  type.measure = "mse",
  nfolds = 5
)
```

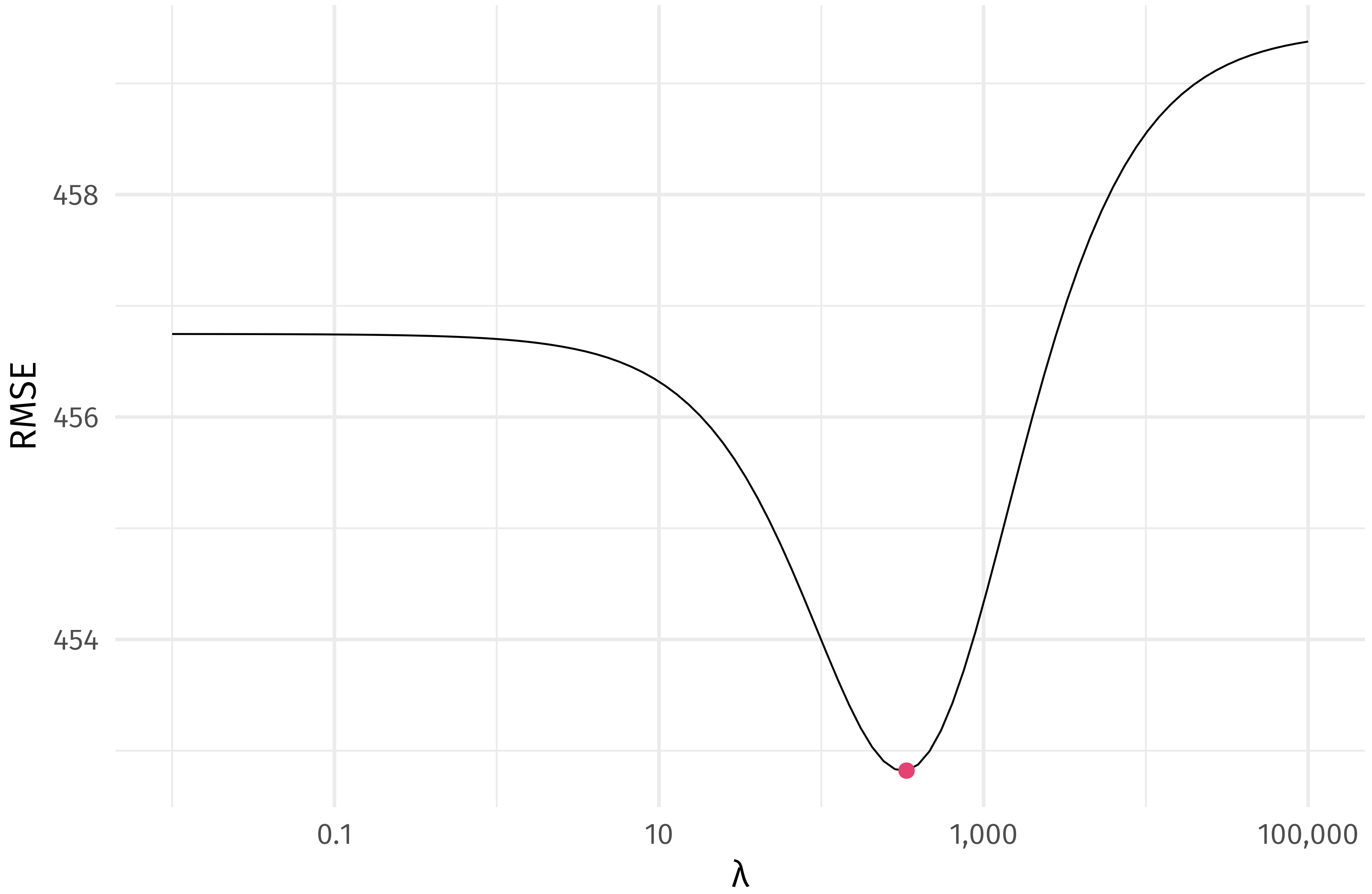
Cross-validated RMSE and λ: Which λ minimizes CV RMSE?



Often, you will have a minimum more obviously far from the extremes.

Recall: Variance-bias tradeoff.

Cross-validated RMSE and λ: Which λ minimizes CV RMSE?



Ridge regression

In `tidymodels`

`tidymodels` can also cross validate (and fit) ridge regression.

- Back to our the `linear_reg()` model 'specification'.
- The penalty λ (what we want to tune) is `penalty` instead of `lambda`.
- Set `mixture = 0` inside `linear_reg()` (same as `alpha = 0`, above).
- Use the `glmnet` engine.

```
# Define the model
model_ridge = linear_reg(penalty = tune(), mixture = 0) %>% set_engine("glmnet")
```

Example of ridge regression with `tidymodels`

```r
# Our range of lambdas
lambdas = 10^seq(from = 5, to = -2, length = 1e3)
# Define the 5-fold split
set.seed(12345)
credit_cv = credit_df %>% vfold_cv(v = 5)
# Define the model
model_ridge = linear_reg(penalty = tune(), mixture = 0) %>% set_engine("glmnet")
# Define our ridge workflow
workflow_ridge = workflow() %>%
  add_model(model_ridge) %>% add_recipe(credit_recipe)
# CV with our range of lambdas
cv_ridge =
  workflow_ridge %>%
  tune_grid(
    credit_cv,
    grid = data.frame(penalty = lambdas),
    metrics = metric_set(rmse)
  )
# Show the best models
cv_ridge %>% show_best()
```

With `tidymodels`...

Next steps: Finalize your workflow and fit your last model.

Recall: `finalize_workflow()`, `last_fit()`, and `collect_predictions()`

Prediction in R

Otherwise: Once you find your λ via cross validation,

1. Fit your model on the full dataset using the optimal λ

```
# Fit final model
final_ridge =  glmnet(
  x = credit_clean %>% dplyr::select(-balance, -id) %>% as.matrix(),
  y = credit_clean$balance,
  standardize = T,
  alpha = 0,
  lambda = ridge_cv$lambda.min
)
```

Prediction in R

Once you find your λ via cross validation

1. Fit your model on the full dataset using the optimal λ

2. Make predictions

```
predict(
  final_ridge,
  type = "response",
  # Our chosen lambda
  s = ridge_cv$lambda.min,
  # Our data
  newx = credit_clean %>% dplyr::select(-balance, -id) %>% as.matrix()
)
```

Ridge regression

Shrinking

While ridge regression *shrinks* coefficients close to zero, it never forces them to be equal to zero.

Drawbacks

1. We cannot use ridge regression for subset/feature selection.
2. We often end up with a bunch of tiny coefficients.

Ridge regression

Shrinking

While ridge regression *shrinks* coefficients close to zero, it never forces them to be equal to zero.

Drawbacks

1. We cannot use ridge regression for subset/feature selection.
2. We often end up with a bunch of tiny coefficients.

Q Can't we just drive the coefficients to zero?

Shrinking

While ridge regression *shrinks* coefficients close to zero, it never forces them to be equal to zero.

Drawbacks

1. We cannot use ridge regression for subset/feature selection.
2. We often end up with a bunch of tiny coefficients.

Q Can't we just drive the coefficients to zero?

A Yes. Just not with ridge (due to $\sum_j \hat{\beta}_j^2$).

Lasso

Lasso

Intro

Lasso simply replaces ridge's *squared* coefficients with absolute values.

Lasso

Intro

Lasso simply replaces ridge's *squared* coefficients with absolute values.

Ridge regression

$$\min_{\hat{\beta}^R} \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Lasso

$$\min_{\hat{\beta}^L} \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 + \lambda \sum_{j=1}^{p} \left|\beta_j\right|$$

Everything else will be the same—except one aspect...

Lasso

Shrinkage

Unlike ridge, lasso's penalty does not increase with the size of β_j.

You always pay λ to increase $|\beta_j|$ by one unit.

Lasso

Shrinkage

Unlike ridge, lasso's penalty does not increase with the size of β_j.

You always pay λ to increase $|\beta_j|$ by one unit.

The only way to avoid lasso's penalty is to **set coefficents to zero**.

Lasso

Shrinkage

Unlike ridge, lasso's penalty does not increase with the size of β_j.

You always pay λ to increase $|\beta_j|$ by one unit.

The only way to avoid lasso's penalty is to **set coefficents to zero**.

This feature has two **benefits**

1. Some coefficients will be **set to zero**—we get "sparse" models.
2. Lasso can be used for subset/feature **selection**.

Lasso

Shrinkage

Unlike ridge, lasso's penalty does not increase with the size of β_j.

You always pay λ to increase $|\beta_j|$ by one unit.

The only way to avoid lasso's penalty is to **set coefficents to zero**.

This feature has two **benefits**

1. Some coefficients will be **set to zero**—we get "sparse" models.
2. Lasso can be used for subset/feature **selection**.

We will still need to carefully select λ.

Lasso

Example

We can also use `glmnet()` for lasso.

Recall The **key arguments** for `glmnet()` are

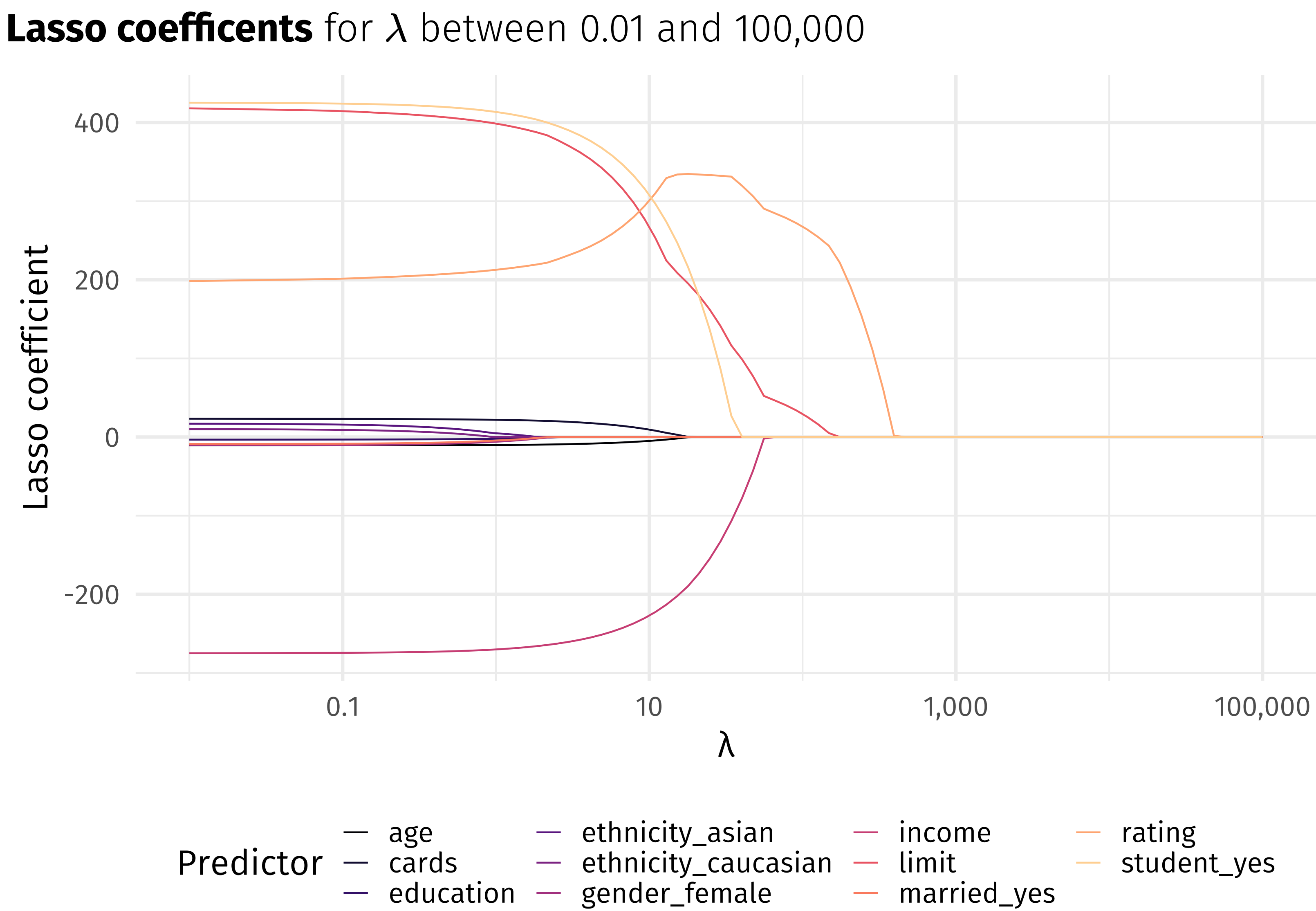
- `x` a **matrix** of predictors
- `y` outcome variable as a vector
- `standardize` (`T` or `F`)
- `alpha` elasticnet parameter
  - `alpha=0` gives ridge
  - **`alpha=1` gives lasso**
- `lambda` tuning parameter (sequence of numbers)
- `nlambda` alternatively, R picks a sequence of values for λ

Example

Again, we define a decreasing sequence for λ, and we're set.

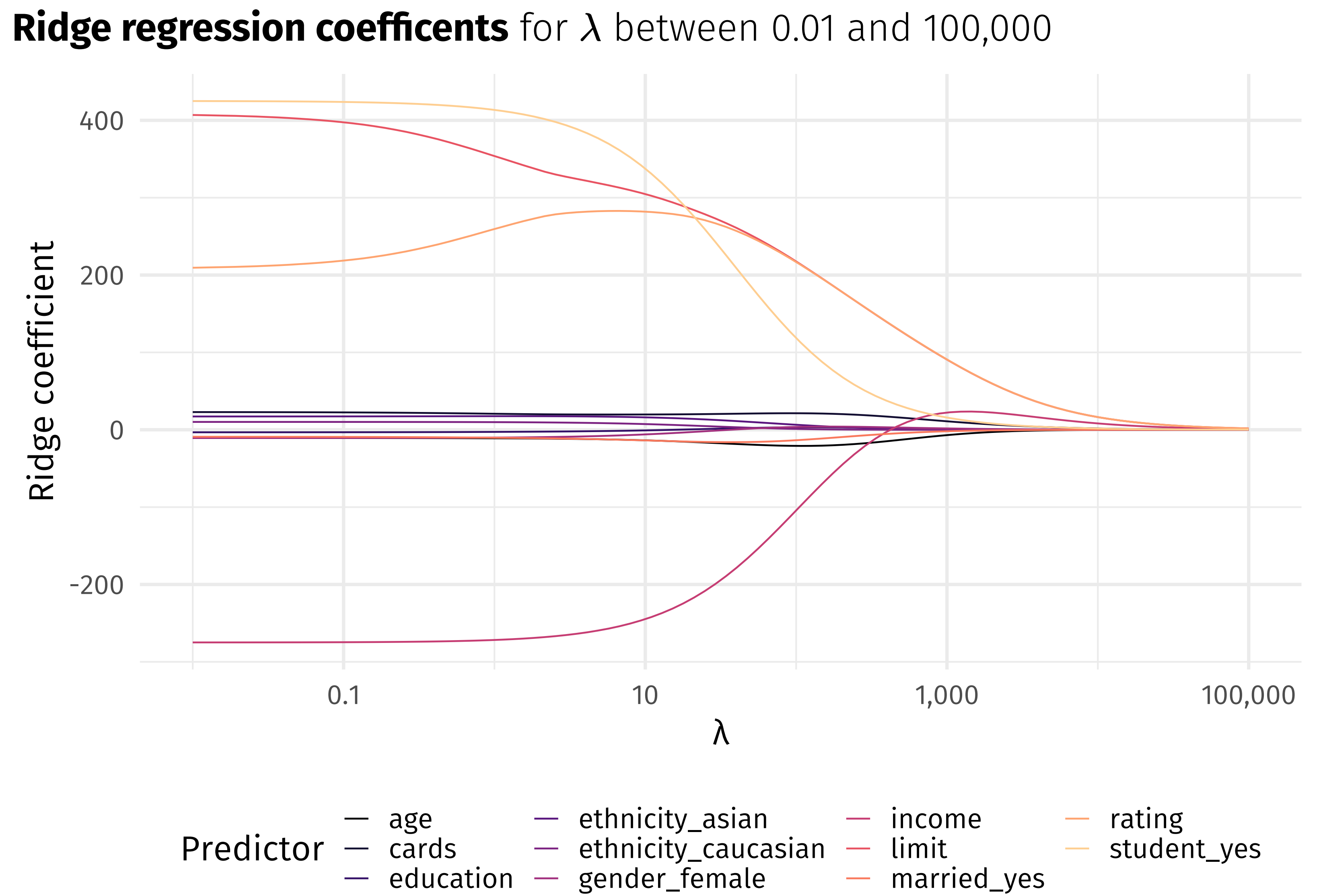
```r
# Define our range of lambdas (glmnet wants decreasing range)
lambdas = 10^seq(from = 5, to = -2, length = 100)
# Fit lasso regression
est_lasso = glmnet(
  x = credit_clean %>% dplyr::select(-balance, -id) %>% as.matrix(),
  y = credit_clean$balance,
  standardize = F,
  alpha = 1,
  lambda = lambdas
)
```

The `glmnet` output (`est_lasso` here) contains estimated coefficients for λ. You can use `predict()` to get coefficients for additional values of λ.



Lasso coefficents for λ between 0.01 and 100,000
Lasso coefficient
400
200
0
-200
0.1
10
1,000
100,000
λ
Predictor
age
cards
education
ethnicity_asian
ethnicity_caucasian
gender_female
income
limit
married_yes
rating
student_yes

Compare lasso's tendency to force coefficients to zero with our previous ridge-regression results.

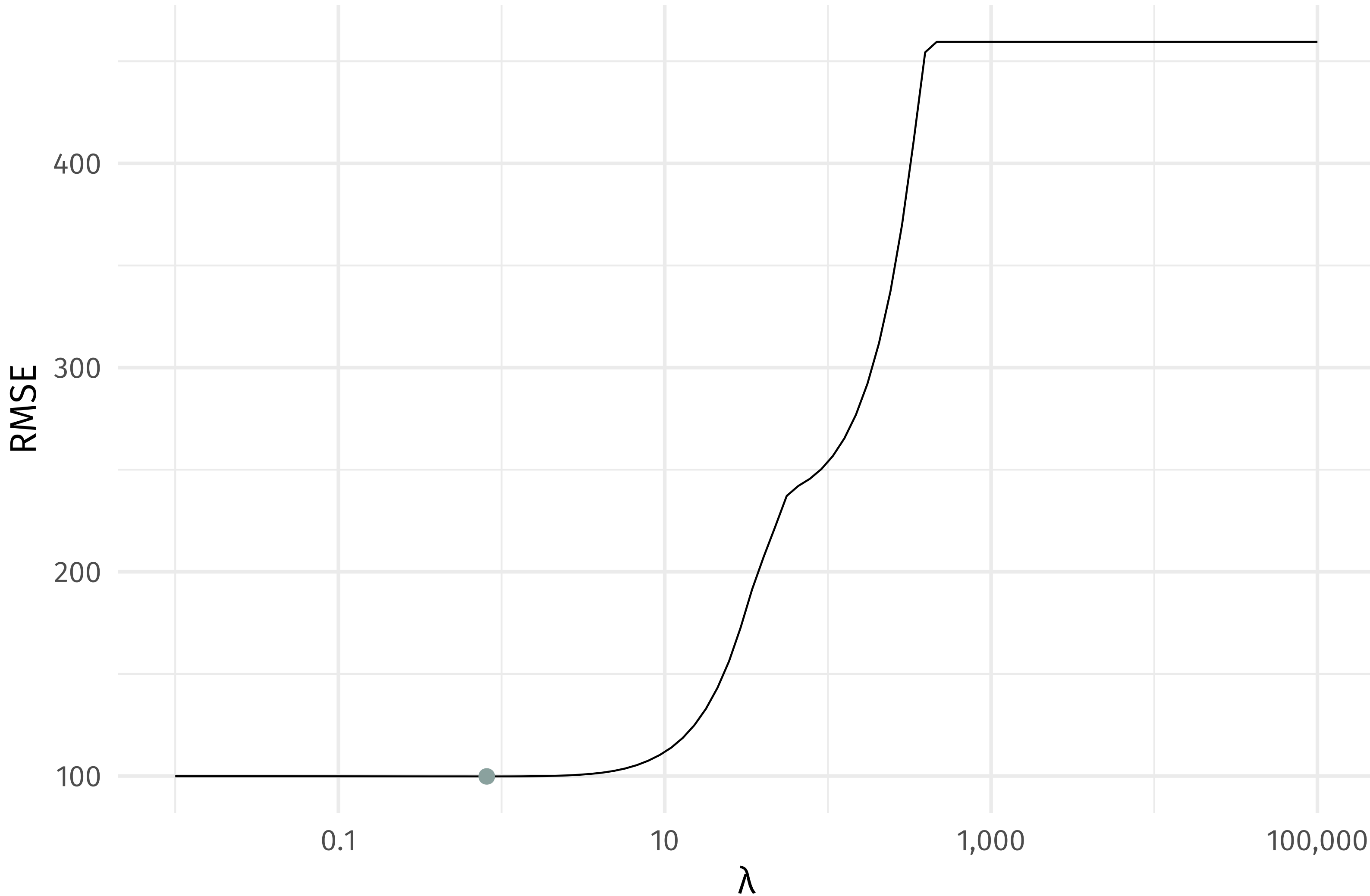

Ridge regression coefficents for λ between 0.01 and 100,000
Ridge coefficient
400
200
0
-200
0.1
10
1,000
100,000
λ
Predictor
age
cards
education
ethnicity_asian
ethnicity_caucasian
gender_female
income
limit
married_yes
rating
student_yes

Example

We can also cross validate λ with `cv.glmnet()`.

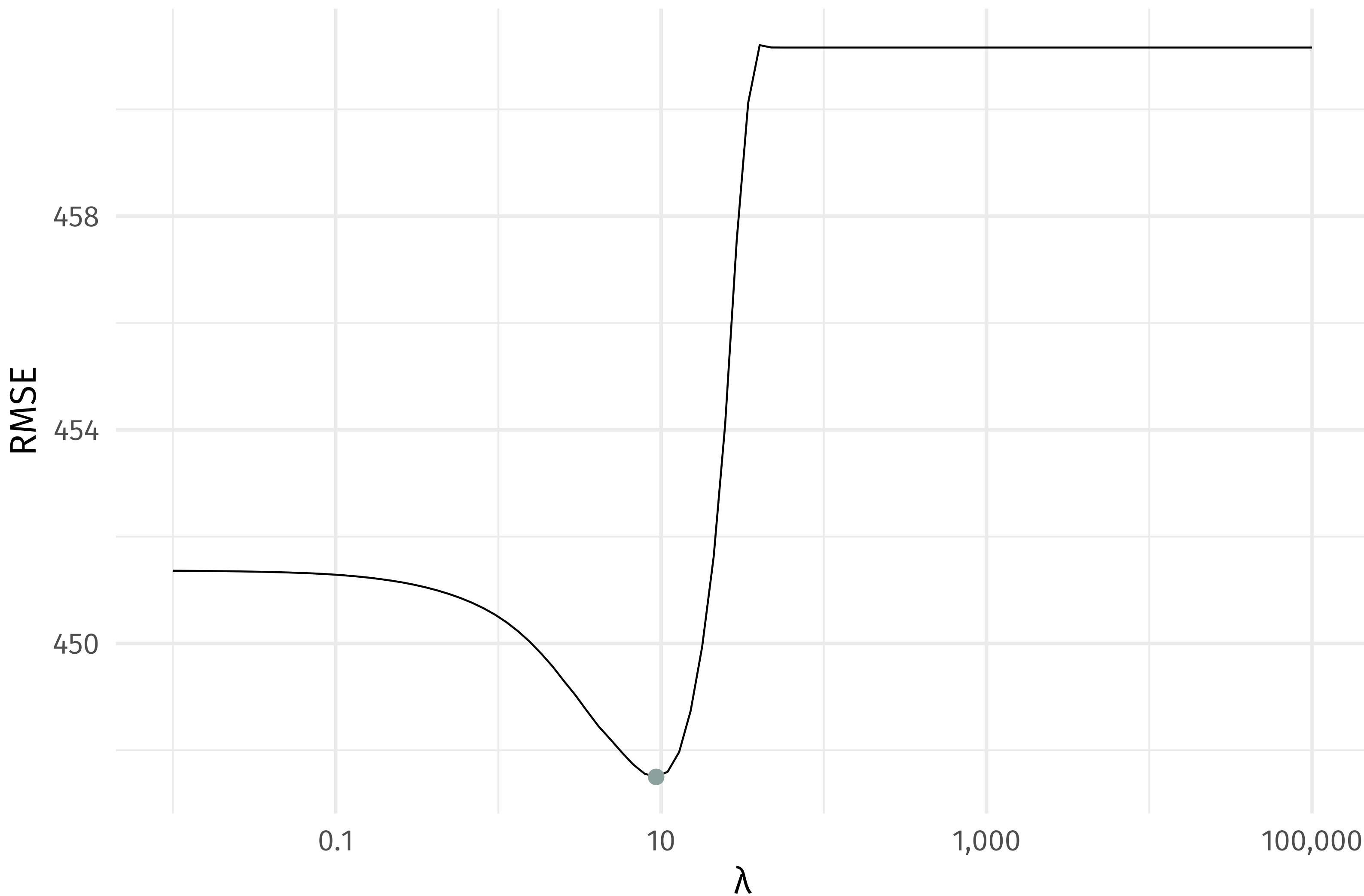
```
# Define our lambdas
lambdas = 10^seq(from = 5, to = -2, length = 100)
# Cross validation
lasso_cv = cv.glmnet(
  x = credit_clean %>% dplyr::select(-balance, -id) %>% as.matrix(),
  y = credit_clean$balance,
  alpha = 1,
  standardize = F,
  lambda = lambdas,
  # New: How we make decisions and number of folds
  type.measure = "mse",
  nfolds = 5
)
```

Cross-validated RMSE and λ: Which λ minimizes CV RMSE?



Again, you will have a minimum farther away from your extremes…

Cross-validated RMSE and λ: Which λ minimizes CV RMSE?



So which shrinkage method should you choose?

Ridge or lasso?

Ridge regression

+ shrinks $\hat{\beta}_j$ *near* 0
- many small $\hat{\beta}_j$
- doesn't work for selection
- difficult to interpret output
+ better when all $\beta_j \neq 0$

Best: p is large & $\beta_j \approx \beta_k$

Lasso

+ shrinks $\hat{\beta}_j$ to 0
+ many $\hat{\beta}_j = 0$
+ great for selection
+ sparse models easier to interpret
- implicitly assumes some $\beta = 0$

Best: p is large & many $\beta_j \approx 0$

Ridge or lasso?

Ridge regression

+ shrinks $\hat{\beta}_j$ *near* 0
- many small $\hat{\beta}_j$
- doesn't work for selection
- difficult to interpret output
+ better when all $\beta_j \neq 0$

Best: p is large & $\beta_j \approx \beta_k$

Lasso

+ shrinks $\hat{\beta}_j$ to 0
+ many $\hat{\beta}_j = 0$
+ great for selection
+ sparse models easier to interpret
- implicitly assumes some $\beta = 0$

Best: p is large & many $\beta_j \approx 0$

> [N]either ridge… nor the lasso will universally dominate the other.

ISL, p. 224

Ridge *and* lasso

Why not both?

Elasticnet combines ridge regression and lasso.

Ridge *and* lasso

Why not both?

Elasticnet combines ridge regression and lasso.

$$\min_{\beta^E} \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 + (1-\alpha)\lambda \sum_{j=1}^{p} \beta_j^2 + \alpha\lambda \sum_{j=1}^{p} \left|\beta_j\right|$$

We now have two tuning parameters: λ (penalty) and α (mixture).

Ridge *and* lasso

Why not both?

Elasticnet combines ridge regression and lasso.

$$\min_{\beta^E} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2 + (1-\alpha)\lambda \sum_{j=1}^{p} \beta_j^2 + \alpha\lambda \sum_{j=1}^{p} |\beta_j|$$

We now have two tuning parameters: λ (penalty) and α (mixture).

Remember the `alpha` argument in `glmnet()`?

- $\alpha = 0$ specifies ridge
- $\alpha = 1$ specifies lasso

Why not both?

We can use `tune()` from `tidymodels` to cross validate both α and λ.

Note You need to consider all combinations of the two parameters.
This combination can create *a lot* of models to estimate.

For example,

- 1,000 values of λ
- 1,000 values of α

leaves you with 1,000,000 models to estimate.[†]

† 5,000,000 if you are doing 5-fold CV!

Cross validating elasticnet in `tidymodels`

```
# Our range of λ and α
lambdas = 10^seq(from = 5, to = -2, length = 1e2)
alphas = seq(from = 0, to = 1, by = 0.1)
# Define the 5-fold split
set.seed(12345)
credit_cv = credit_df %>% vfold_cv(v = 5)
# Define the elasticnet model
model_net = linear_reg(
  penalty = tune(), mixture = tune()
) %>% set_engine("glmnet")
# Define our workflow
workflow_net = workflow() %>%
  add_model(model_net) %>% add_recipe(credit_recipe)
# CV elasticnet with our range of lambdas
cv_net =
  workflow_net %>%
  tune_grid(
    credit_cv,
    grid = expand_grid(mixture = alphas, penalty = lambdas),
    metrics = metric_set(rmse)
  )
```

Cross validating elasticnet in `tidymodels` with `grid_regular()`

```
# Our range of λ and α
lambdas = 10^seq(from = 5, to = -2, length = 1e2)
alphas = seq(from = 0, to = 1, by = 0.1)
# Define the 5-fold split
set.seed(12345)
credit_cv = credit_df %>% vfold_cv(v = 5)
# Define the elasticnet model
model_net = linear_reg(
  penalty = tune(), mixture = tune()
) %>% set_engine("glmnet")
# Define our workflow
workflow_net = workflow() %>%
  add_model(model_net) %>% add_recipe(credit_recipe)
# CV elasticnet with our range of lambdas
cv_net =
  workflow_net %>%
  tune_grid(
    credit_cv,
    grid = grid_regular(mixture(), penalty(), levels = 100:100),
    metrics = metric_set(rmse)
  )
```

In case you are curious: The *best* model had $\lambda \approx 0.628$ and $\alpha \approx 0.737$.

CV estimates elasticnet actually reduced RMSE from ridge's 118 to 101.

Sources

These notes draw upon

- An Introduction to Statistical Learning (*ISL*)
  James, Witten, Hastie, and Tibshirani

Table of contents