Shrinkage Methods and Hyperparameter tunning

AU STAT627

Emil Hvitfeldt

2021-06-09

Shrinkage Methods

We saw last Monday how PCA can be used as a dimensionality reduction method

This can help us deal with data sets with many variables, high dimensional data

Another way to deal with high dimensional data is to use **feature selection**

We will talk about shrinkage methods

Shrinkage Methods

Fit a model with all the predictors + constraints to the coefficient estimates

These constraints will typically try to drag the coefficients towards zero (hence the name)

Shrinkage methods can significantly reduce the variance of the coefficient estimates

Shrinkage Methods

Two best-known techniques

- Ridge regression
- Lasso regression

Ridge Regression

We recall that the least-squares estimator is found by minimizing RSS

$$\text{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

ridge regression builds on this method

Ridge Regression

Introducing a shrinkage penalty

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{p}\beta_j^2$$

where $\lambda \geq 0$ is a **tuning parameter**

Choice of λ

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

The value of λ have a big effect on the parameter estimates

- If $\lambda = 0$ we don't have any penalization and we get the standard OLS estimates
- If $\lambda \to \infty$ then all the parameters estimates will be shrunk towards zero

Somewhere in the middle is where interesting things start to happen. The different coefficients will be shrunk at different rates

Choice of λ

There is no way to select the best value of λ from the data directly. We will have to try different values out and pick which one performs best

Luckily for this method, The algorithm can fit the model for multiple values of λ at once, leaving us with only 1 model fit

We will see later how this is done

Importance of variable scales

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

In general, for regression models we are not worried about the scale of the variables as the fit will be the same regardless

However, the penalty term here is not scale-invariant

You must scale your variables to have them influence the model evenly

Bias - Variance trade-off

The beauty of this method is that different values of λ slides across the bias/variance spectrum

Lasso Regression

One of the main downsides to ridge regression is that the coefficients will not be shrunk to 0 exactly (unless $\lambda = \infty$)

We technically have to use all the parameters each time we predict and try to explain the model

Lasso regression tries to deal with this issue

Ridge regression

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

Lasso Regression

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

Lasso Regression

Lasso regression allows us to shrink some of the coefficients directly down to 0 depending on the value of λ

This allows Lasso to ask as a variable selection method

Fitting a Ridge Regression Model

First, let us select some data. We will be using the `concrete` data set from {modeldata}.

```r
library(tidymodels)

data("biomass")

biomass_scaled <- biomass %>%
  select(where(is.numeric)) %>%
  scale() %>%
  as_tibble()
```

```r
biomass_scaled
```

```
## # A tibble: 536 x 6
##     carbon hydrogen  oxygen nitrogen  sulfur
##      <dbl>    <dbl>   <dbl>    <dbl>   <dbl>   <
##  1  0.149    0.152   0.404   -0.561  -0.482   0.
##  2  0.118    0.202   0.252   -0.737  -0.482   0.
##  3 -0.0466   0.285   0.711   -0.813  -0.441  -0.
##  4 -0.314   -0.408  -0.275    1.87   -0.152  -0.
##  5 -0.151   -0.0489  0.199   -0.0649 -0.441  -0.
##  6 -0.284    0.243   0.151    0.809  -0.276  -0.
##  7 -0.110    0.444  -0.0354   1.35   -0.0694 -0.
##  8 -0.255    0.202   0.104    0.524  -0.0694 -0.
##  9  0.0497   0.0346  0.215   -0.233  -0.482  -0.
## 10 -0.117    0.369   0.132    0.103  -0.276  -0.
## # … with 526 more rows
```

Fitting a Ridge Regression Model

A Ridge regression model specification can be specified using `linear_reg()` with `mixture = 0`

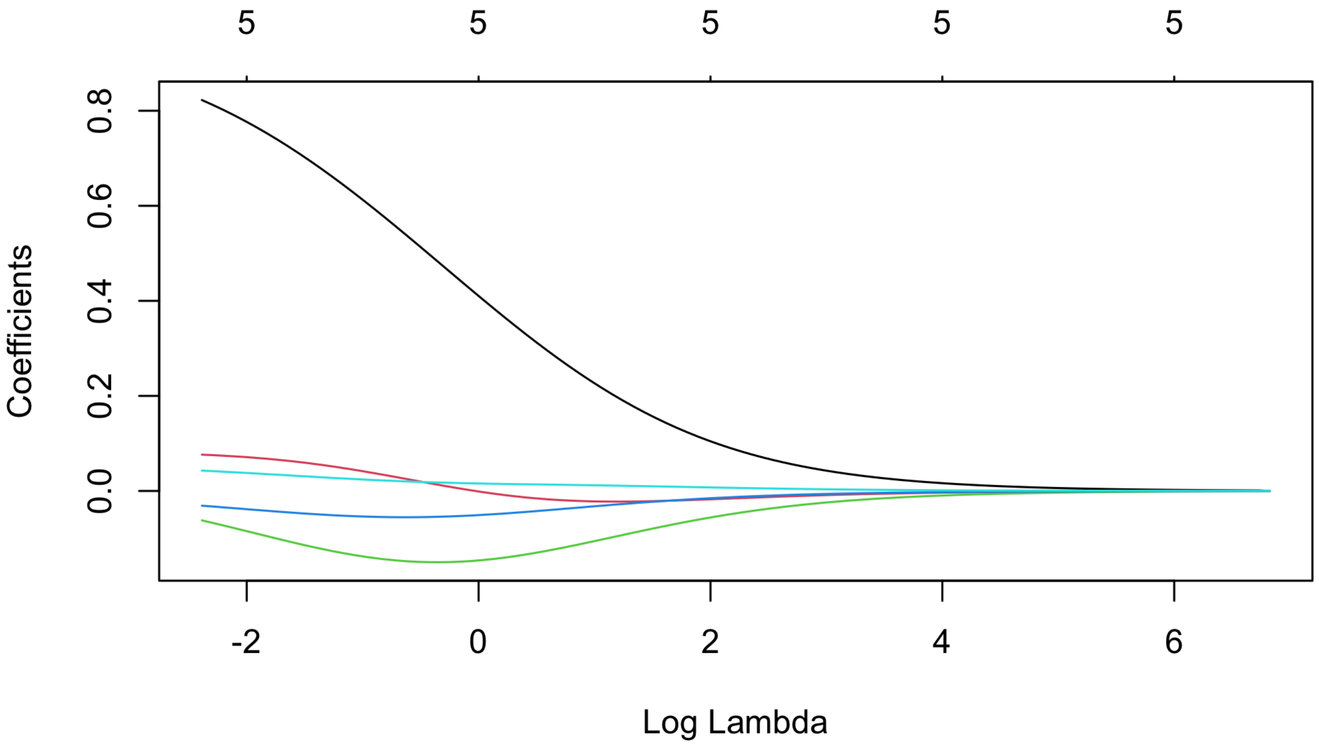
```r
ridge_spec <- linear_reg(mixture = 0, penalty = 0) %>%
  set_engine("glmnet")
```

Then we can fit it just like normal

```r
ridge_fit <- ridge_spec %>%
  fit(HHV ~ ., data = biomass_scaled)
```

Plotting coefficients for Ridge Model

```
plot(ridge_fit$fit, xvar = "lambda")
```



Predicting for a Ridge Regression model

```
predict(ridge_fit, new_data = biomass_scaled)
```

```
## # A tibble: 536 x 1
##       .pred
##       <dbl>
##  1  0.106
##  2  0.0993
##  3 -0.0543
##  4 -0.337
##  5 -0.157
##  6 -0.262
##  7 -0.0995
##  8 -0.220
##  9  0.0167
## 10 -0.0916
## # … with 526 more rows
```

Predicting for a Ridge Regression model

Specifying a value lets up predict

```
predict(ridge_fit, new_data = biomass_scaled, penalty = 100)
```

```
## # A tibble: 536 x 1
##         .pred
##         <dbl>
##  1 -0.000760
##  2 -0.000103
##  3 -0.00406
##  4 -0.00313
##  5 -0.00260
##  6 -0.00515
##  7 -0.00353
##  8 -0.00401
##  9 -0.000851
## 10 -0.00287
## # … with 526 more rows
```

Predicting for a Ridge Regression model

We could also specify the penalty directly when we fit the model, but there is not as often a usecase for this

```r
ridge_spec <- linear_reg(mixture = 0, penalty = 100) %>%
  set_engine("glmnet")

ridge_fit <- ridge_spec %>%
  fit(HHV ~ ., data = biomass_scaled)

predict(ridge_fit, new_data = biomass_scaled)
```

```
## # A tibble: 536 x 1
##         .pred
##         <dbl>
##  1 -0.000760
##  2 -0.000103
##  3 -0.00406
##  4 -0.00313
##  5 -0.00260
##  6 -0.00515
```

Fitting a Lasso Regression model

Fitting a lasso model is done the same way a ridge model is fit, instead, we have to set `mixture = 1`

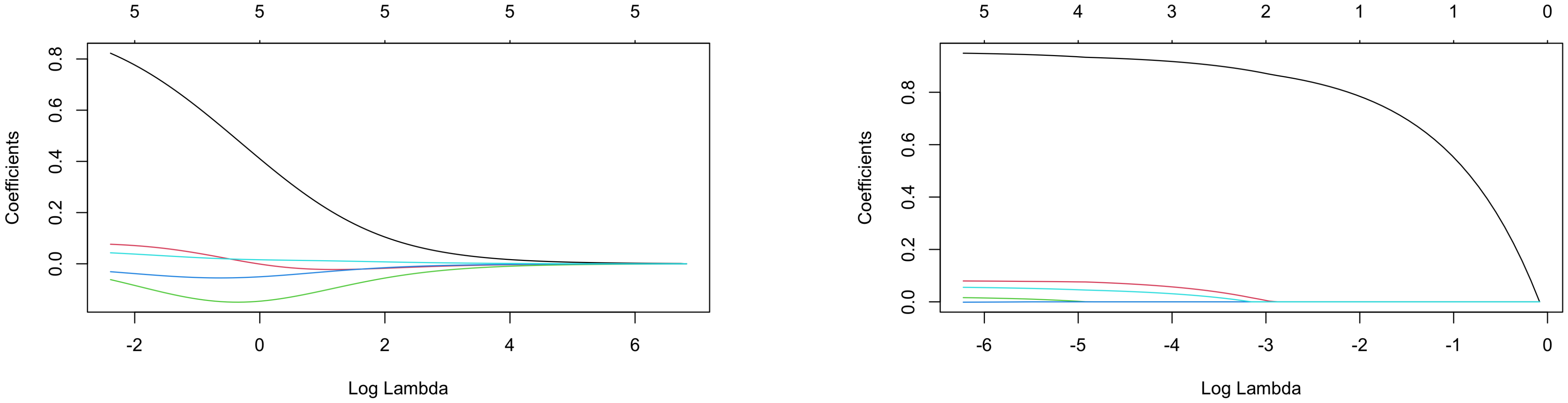
```
lasso_spec <- linear_reg(mixture = 1, penalty = 0) %>%
  set_engine("glmnet")

lasso_fit <- lasso_spec %>%
  fit(HHV ~ ., data = biomass_scaled)
```

Plotting coefficients for Lasso Model

```
lasso_fit$fit %>%
  plot(xvar = "lambda")
```

Ridge VS Lasso



Hyperparameter Tuning

Some models have parameters that can not directly be estimated from the data alone. These parameters are found by other means

The "simple" way is to try a lot of different values and pick the one that performs the best

Hyperparameter Tuning

We will use a cross-validation approach to find the best value for the penalty.

we start by splitting up our data and creating some folds.

```
biomass_split <- initial_split(biomass)

biomass_train <- training(biomass_split)

biomass_folds <- vfold_cv(biomass_train, v = 5)
```

Hyperparameter Tuning

Now we create a new lasso specification, but this time we use `tune()` to denote that we want to tune the `penalty` parameter.

```
lasso_spec <- linear_reg(mixture = 1, penalty = tune()) %>%
  set_engine("glmnet")
lasso_spec
```

```
## Linear Regression Model Specification (regression)
##
## Main Arguments:
##   penalty = tune()
##   mixture = 1
##
## Computational engine: glmnet
```

Hyperparameter Tuning

We also create a recipe to normalize (scale + center) all the predictors

```
rec_spec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
                   data = biomass) %>%
  step_normalize(all_predictors())
```

And we combine these two into a `workflow`

```
lasso_wf <- workflow() %>%
  add_model(lasso_spec) %>%
  add_recipe(rec_spec)
```

Hyperparameter Tuning

We also need to specify what values of the hyperparameters we are trying to tune we want to calculate. Since the lasso model can calculate all paths at once let us get back 50 evenly spaced values of λ

```
lambda_grid <- grid_regular(penalty(), levels = 50)
lambda_grid
```

```
## # A tibble: 50 x 1
##     penalty
##       <dbl>
##  1 1   e-10
##  2 1.60e-10
##  3 2.56e-10
##  4 4.09e-10
##  5 6.55e-10
##  6 1.05e- 9
##  7 1.68e- 9
##  8 2.68e- 9
##  9 4.29e- 9
```

Hyperparameter Tuning

We combine these things in `tune_grid()` which works much like `fit_resamples()` but takes a `grid` argument as well

```r
tune_rs <- tune_grid(
  object = lasso_wf,
  resamples = biomass_folds,
  grid = lambda_grid
)
```

Hyperparameter Tuning

We can see how each of the values of λ is doing with `collect_metrics()`

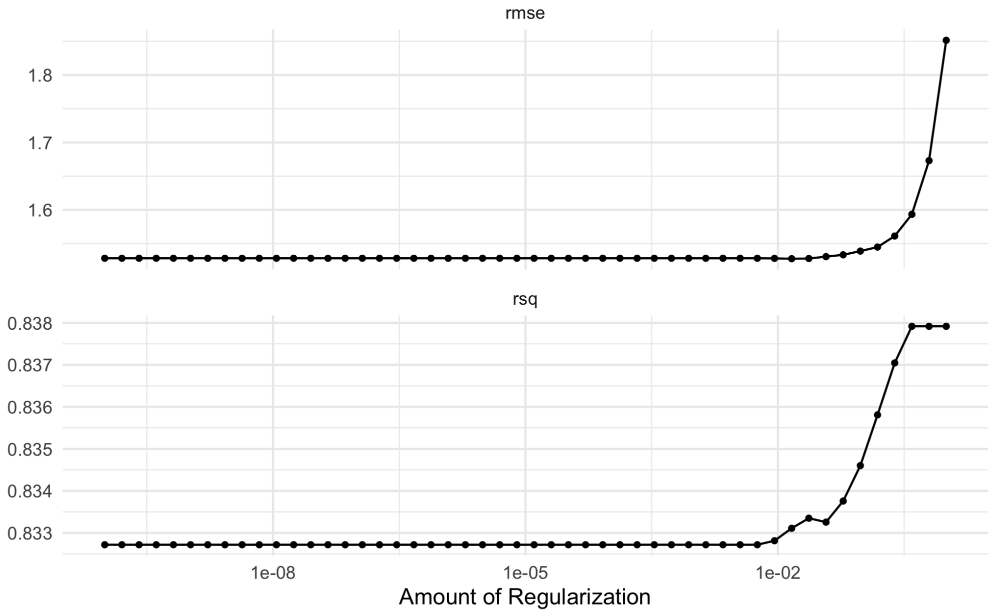
```
collect_metrics(tune_rs)
```

```
## # A tibble: 100 x 7
##     penalty .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 1   e-10 rmse    standard   1.53      5  0.258  Preprocessor1_Model01
##  2 1   e-10 rsq     standard   0.833     5  0.0455 Preprocessor1_Model01
##  3 1.60e-10 rmse    standard   1.53      5  0.258  Preprocessor1_Model02
##  4 1.60e-10 rsq     standard   0.833     5  0.0455 Preprocessor1_Model02
##  5 2.56e-10 rmse    standard   1.53      5  0.258  Preprocessor1_Model03
##  6 2.56e-10 rsq     standard   0.833     5  0.0455 Preprocessor1_Model03
##  7 4.09e-10 rmse    standard   1.53      5  0.258  Preprocessor1_Model04
##  8 4.09e-10 rsq     standard   0.833     5  0.0455 Preprocessor1_Model04
##  9 6.55e-10 rmse    standard   1.53      5  0.258  Preprocessor1_Model05
## 10 6.55e-10 rsq     standard   0.833     5  0.0455 Preprocessor1_Model05
## # … with 90 more rows
```

Hyperparameter Tuning

And there is even a plotting method that can show us how the different values of the hyperparameter are doing

```
autoplot(tune_rs)
```



Hyperparameter Tuning

Look at the best performing one with `show_best()` and select the best with `select_best()`

```
tune_rs %>%
  show_best("rmse")
```

```
## # A tibble: 5 x 7
##    penalty .metric .estimator  mean     n std_err .config
##      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 1.46e- 2 rmse    standard    1.53     5   0.257 Preprocessor1_Model41
## 2 2.33e- 2 rmse    standard    1.53     5   0.257 Preprocessor1_Model42
## 3 9.10e- 3 rmse    standard    1.53     5   0.258 Preprocessor1_Model40
## 4 1   e-10 rmse    standard    1.53     5   0.258 Preprocessor1_Model01
## 5 1.60e-10 rmse    standard    1.53     5   0.258 Preprocessor1_Model02
```

```
best_rmse <- tune_rs %>%
  select_best("rmse")
```

Hyperparameter Tuning

Remember how the specification has `penalty = tune()`?

```
lasso_wf
```

```
## ══ Workflow ═══════════════════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: linear_reg()
##
## ── Preprocessor ───────────────────────────────────────────────────────────────
## 1 Recipe Step
##
## • step_normalize()
##
## ── Model ──────────────────────────────────────────────────────────────────────
## Linear Regression Model Specification (regression)
##
## Main Arguments:
##   penalty = tune()
##   mixture = 1
##
```

Hyperparameter Tuning

We can update it with `finalize_workflow()`

```
final_lasso <- finalize_workflow(lasso_wf, best_rmse)
final_lasso
```

```
## ══ Workflow ════════════════════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: linear_reg()
##
## ── Preprocessor ────────────────────────────────────────────────────────────────
## 1 Recipe Step
##
## • step_normalize()
##
## ── Model ───────────────────────────────────────────────────────────────────────
## Linear Regression Model Specification (regression)
##
## Main Arguments:
##   penalty = 0.0145634847750124
##   mixture = 1
```

Hyperparameter Tuning

And this finalized specification can now we can fit using the whole training data set.

```
fitted_lasso <- fit(final_lasso, biomass_train)
```

Hyperparameter Tuning

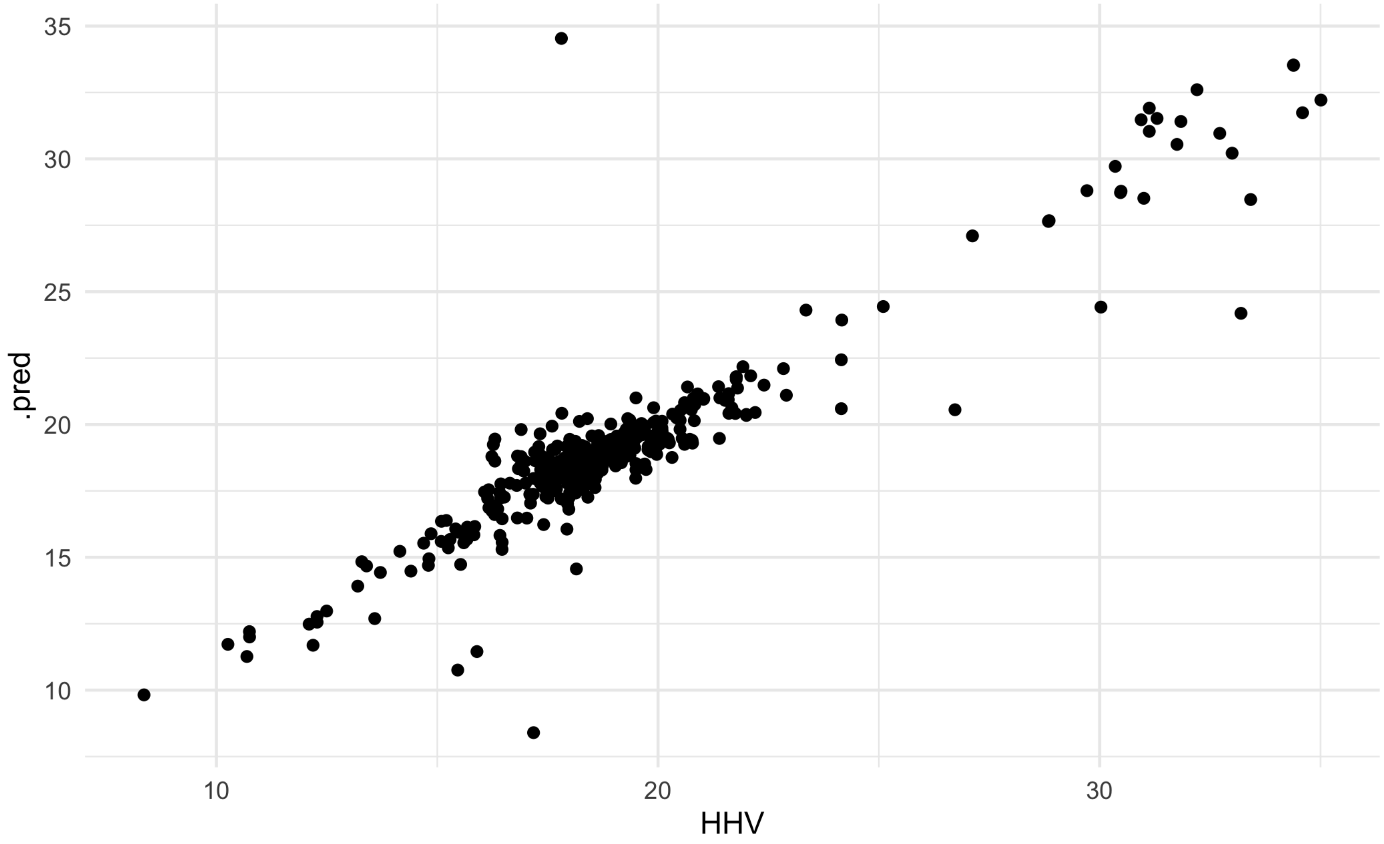
this fitted model can now be used to predict on the testing data set

```
biomass_test <- training(biomass_split)

predict(fitted_lasso, biomass_test)
```

```
## # A tibble: 402 x 1
##    .pred
##    <dbl>
##  1  19.1
##  2  28.5
##  3  22.1
##  4  19.1
##  5  19.2
##  6  17.3
##  7  18.3
##  8  19.1
##  9  19.4
## 10  15.9
## # … with 392 more rows
```

Hyperparameter Tuning


.pred
HHV