Penalized Regression

Ridge, Lasso, ElasticNet

DS 6030 | Fall 2022

penalized.pdf

Contents

1 Shrinkage and Penalized Regression Intro

1.1 Required R Packages

We will be using the R packages of:

- `glmnet` for ridge, lasso, and elasticnet regression
- `broom` for obtaining tidy model outputs
- `tidyverse` for data manipulation and visualization

```
library(glmnet)
library(broom)
library(tidyverse)
```

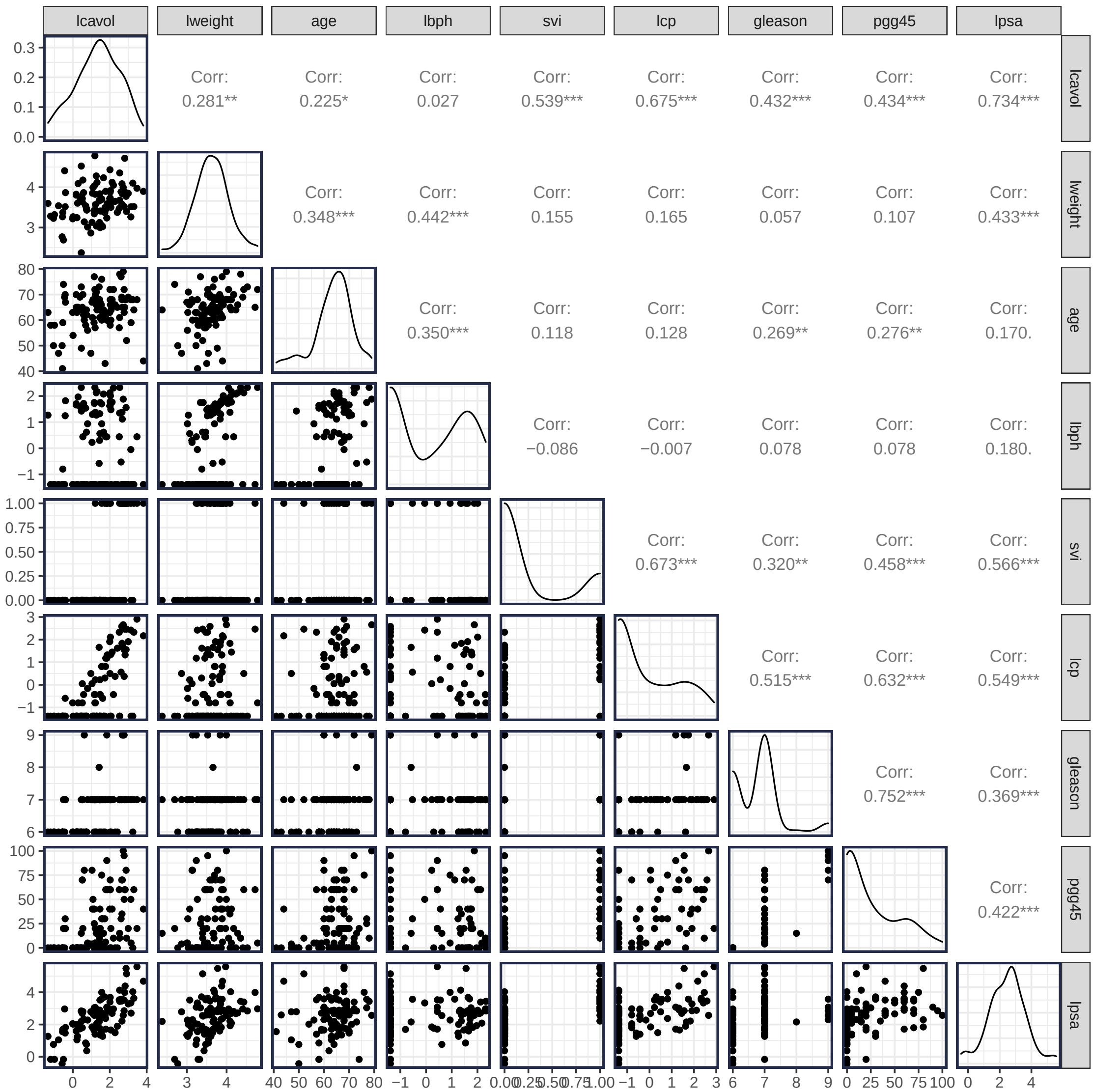
> Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

1.2 Prostate Cancer Data

The Elements of Statistical Learning (ESL) text has a description of a prostate cancer dataset used in a study by Stamey et al. (1989). They examined the correlation between the level of prostate-specific antigen and a number of clinical measures in men who were about to receive a radical prostatectomy.

The variables are:

- log cancer volume (`lcavol`)
- log prostate weight (`lweight`)
- `age`
- log of the amount of benign prostatic hyperplasia (`lbph`)
- seminal vesicle invasion (`svi`)
- log of capsular penetration (`lcp`)
- Gleason score (`gleason`)
- percent of Gleason scores 4 or 5 (`pgg45`)
- *outcome variable* is the log of prostate-specific antigen, (`lpsa`)

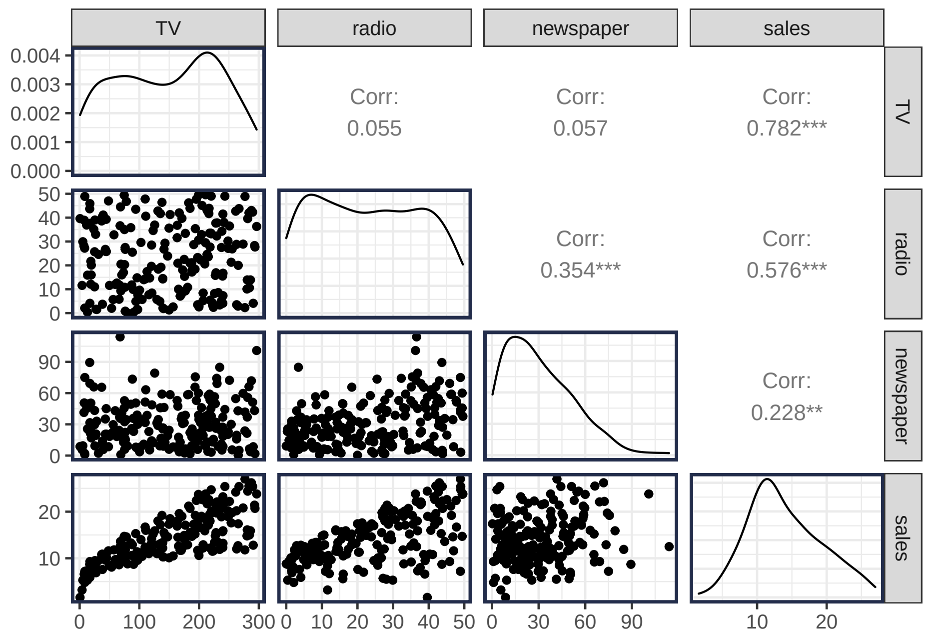


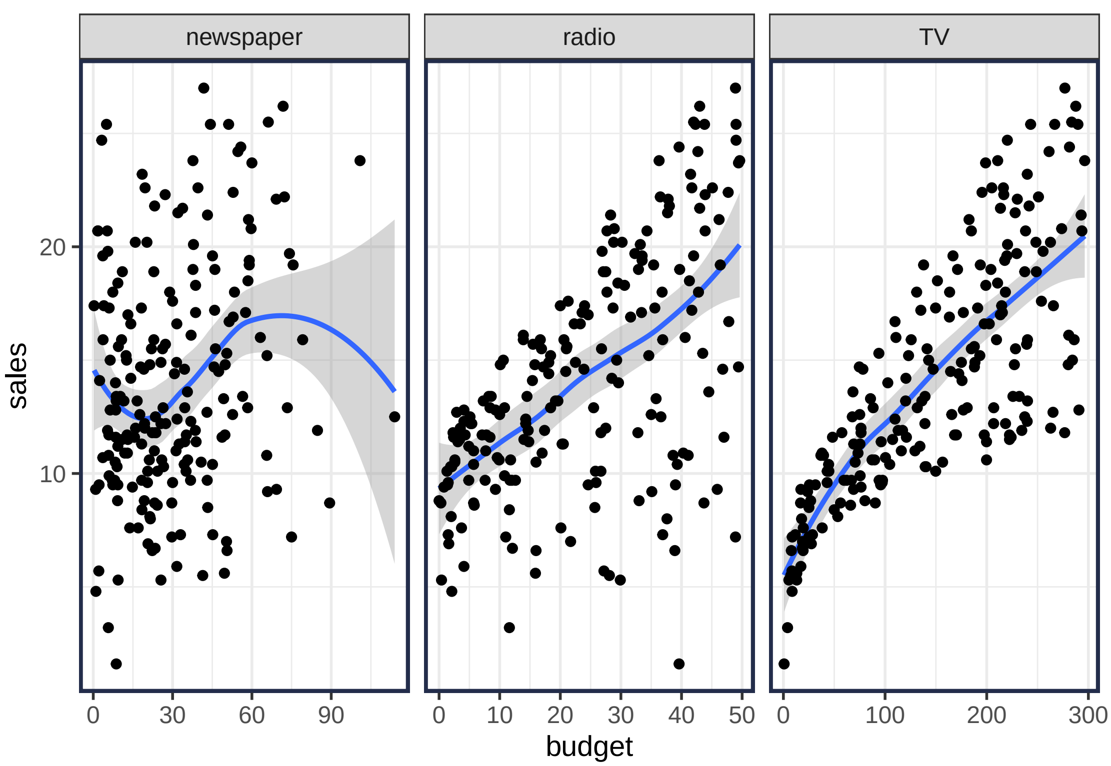
1.3 Advertising Data

The Introduction to Statistical Learning (ISL) text has some data on advertising.

These data give the sales of a product (in thousands of units) under advertising budgets (in thousands of dollars) of TV, radio, and newspaper.

The goal is to predict `sales` for a given `TV`, `radio`, and `newspaper` budget.





1.4 Linear Regression (OLS)

The standard generic form for a linear regression model is

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots, +\beta_p X_p + \epsilon$$

- Y is the response or dependent variable

- $X_1, X_2, \ldots, X_p$ are called the p explanatory, independent, or predictor variables
- the greek letter ϵ (epsilon) is the random error variable
- For example:

$$\text{sales} = \beta_0 + \beta_1 \times (\text{TV}) + \beta_2 \times (\text{radio}) + \beta_3 \times (\text{newspaper}) + \text{error}$$

Training data is used to estimate the *model parameters* or *coefficients*.

$$\left[\begin{array}{cccccc|c} x_{11} & x_{12} & \cdot & \cdot & \cdot & x_{1p} & y_1 \\ x_{21} & x_{22} & \cdot & \cdot & \cdot & x_{2p} & y_2 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ x_{n1} & x_{n2} & \cdot & \cdot & \cdot & x_{np} & y_n \end{array}\right]$$

Producing the predictive model:

$$\hat{y}(x_1, x_2, \ldots, x_p) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \ldots, + \hat{\beta}_p x_p$$

- where $\hat{\beta}_j$ are the weights assigned to each variable
- these weights are the values that minimize the residual sum of squares (RSS) for predicting the training data
- For example:

$$\widehat{\text{sales}} = 2.939 + 0.046 \times (\text{TV}) + 0.189 \times (\text{radio}) \times -0.001 \times (\text{newspaper})$$

- The *complexity* of an OLS regression model is the *number of estimated parameters*
  - it is $p + 1$ (using the notation above), where the $+1$ is added for the intercept.

1.5 Estimation

- The weights/coefficients (β) are the *model parameters*
- OLS uses the weights/coefficients that minimize the RSS loss function over the training data

$$\begin{aligned} \hat{\beta} &= \arg\min_{\beta} \ \text{RSS}(\beta) \qquad \text{Note: } \beta \text{ is a } \textit{vector} \\ &= \arg\min_{\beta} \ \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \beta))^2 \\ &= \arg\min_{\beta} \ \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} + \ldots + \beta_p x_{ip})^2 \end{aligned}$$

OLS equivalently minimizes the MSE since $\text{MSE} = \text{RSS}/n$.

1.5.1 Matrix notation

$$f(\mathbf{x};\beta) = \mathbf{x}^\mathsf{T}\beta$$

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \qquad X = \begin{bmatrix} 1 & X_{11} & X_{12} & X_{13} & \dots & X_{1p} \\ 1 & X_{21} & X_{22} & X_{23} & \dots & X_{2p} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & X_{n2} & X_{n3} & \dots & X_{np} \end{bmatrix} \qquad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}$$

$$\mathrm{RSS}(\beta) = (\mathrm{Y} - \mathrm{X}\beta)^\mathsf{T}(\mathrm{Y} - \mathrm{X}\beta)$$

$$\begin{aligned} \frac{\partial \mathrm{RSS}(\beta)}{\partial \beta} &= 2X^\mathsf{T}(Y - X\beta) \\ &\implies X^\mathsf{T}Y = X^\mathsf{T}X\beta \\ &\implies \boxed{\hat{\beta} = (X^\mathsf{T}X)^{-1}X^\mathsf{T}Y} \end{aligned}$$

1.5.2 OLS in R with `lm()`

```
#-- Fit OLS
prostate.lm = lm(lpsa~., data=prostate.train)
prostate.lm %>% broom::tidy()
#> # A tibble: 9 x 5
#>   term        estimate std.error statistic    p.value
#>   <chr>          <dbl>     <dbl>     <dbl>      <dbl>
#> 1 (Intercept)   0.429     1.55       0.276 0.783
#> 2 lcavol        0.577     0.107      5.37  0.00000147
#> 3 lweight       0.614     0.223      2.75  0.00792
#> 4 age          -0.0190    0.0136    -1.40  0.168
#> 5 lbph          0.145     0.0705     2.06  0.0443
#> 6 svi           0.737     0.299      2.47  0.0165
#> # ... with 3 more rows
```

1.6 Some Problems with least squares estimates

There are a few problems with using least squares estimation (OLS) to estimate the regression parameters (coefficients)

- *Prediction Accuracy*
  - the least squares estimates in high dimensional data may have low bias but can suffer from large variance.
  - Prediction accuracy can sometimes be improved by shrinking or setting some coefficients to zero.
  - By doing so we sacrifice a little bit of bias to reduce the variance of the predicted values, and hence may improve the overall prediction accuracy.
  - Some predictors may not have any predictive value and only increase noise
- *Interpretation*: With a large number of predictors, we often would like to determine a smaller subset that exhibit the strongest effects. In order to get the "big picture", we are willing to sacrifice some of the small details
  - When $p > n$ least squares won't work at all

1.7 Improving Least squares

We will examine 3 standard approaches to improve on least squares estimates

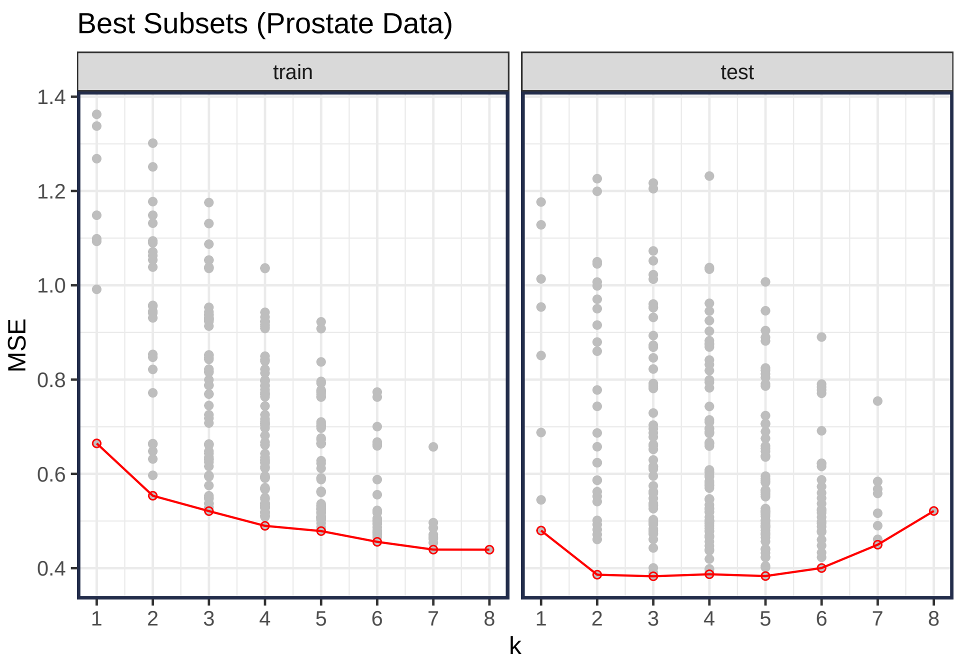
1. Subset Selection
    - Only use a subset of predictors, but estimate with OLS
    - Examples: *best subsets*, *forward step-wise*
2. Shrinkage/Penalized/Regularized Regression
    - Instead of an "all or nothing" approach, shrinkage methods force the coefficients closer toward 0.
    - Examples: *ridge*, *lasso*, *elastic net*
3. Dimension Reduction with Derived Inputs
    - Use a subset of linearly transformed predictors
    - Examples: *PCA*, *PLS*

All three methods introduce some additional bias in order to reduce variance and *hopefully* improve prediction.

2 Subset Selection

Subset selection methods attempt to find the best subset of predictors to use in the model

- Best Subsets finds the best (usually in terms of minimum RSS/MSE) combination of k predictors
  - k is the *tuning parameter*
- Stepwise Selectors takes a greedy approach by sequentially adding (forward) or deleting (backward) the predictor that most improves the fit
  - This is a computational necessity for high dimensional data
  - Tuning parameter options:
    a. Number of predictors (k)
    b. Inclusion/Exclusion criteria: AIC/BIC, adjusted R^2, p-values, etc.



Note

Subset selection methods remove predictors by setting their coefficients to 0 (e.g., $\hat{\beta} = 0$)

- These "all or nothing" approaches can be very unstable. A small change in the data can completely change the model

predictor	lm	best_subset	bootstrap
(Intercept)	0.43	-1.05	-0.33
lcavol	0.58	0.63	0.51
lweight	0.61	0.74	0.54
age	-0.02	0.00	0.00
lbph	0.14	0.00	0.14
svi	0.74	0.00	0.67
lcp	-0.21	0.00	0.00
gleason	-0.03	0.00	0.00
pgg45	0.01	0.00	0.00

3 Shrinkage Methods

Instead of an "all or nothing" approach, shrinkage methods force the coefficients closer toward 0.

- Usually this is accomplished through penalized regression where a penalty is imposed on the size of the coefficients
- Equivalently, the size of the coefficients are *constrained* not to exceed a threshold

3.1 Two Representations

The penalized optimization (Lagrangian form)

$$\hat{\beta} = \arg\min_{\beta} \left\{ l(\beta) + \lambda P(\beta) \right\}$$

An equivalent representation is (constrained optimization)

$$\begin{aligned} \hat{\beta} &= \arg\min_{\beta}\ l(\beta) \qquad \text{subject to } \ P(\beta) \leq t \\ &= \arg\min_{\beta:\ P(\beta)\leq t}\ l(\beta) \end{aligned}$$

where

- $l(\beta)$ is the loss function (e.g. mean squared error, negative log-likelihood)
- $P(\beta)$ is the penalty term (as a function of the model parameters)
- $\lambda \geq 0$ is the strength of the penalty
- t is the penalty budget

3.2 Penalties

Examples penalties:

- Ridge Penalty

$$P(\beta) = \sum_{j=1}^{p} |\beta_j|^2 = \beta^{\mathsf{T}}\beta$$

- Lasso Penalty

$$P(\beta) = \sum_{j=1}^{p} |\beta_j|$$

- Best Subsets

$$P(\beta) = \sum_{j=1}^{p} |\beta_j|^0 = \sum_{j=1}^{p} 1_{(\beta_j \neq 0)}$$

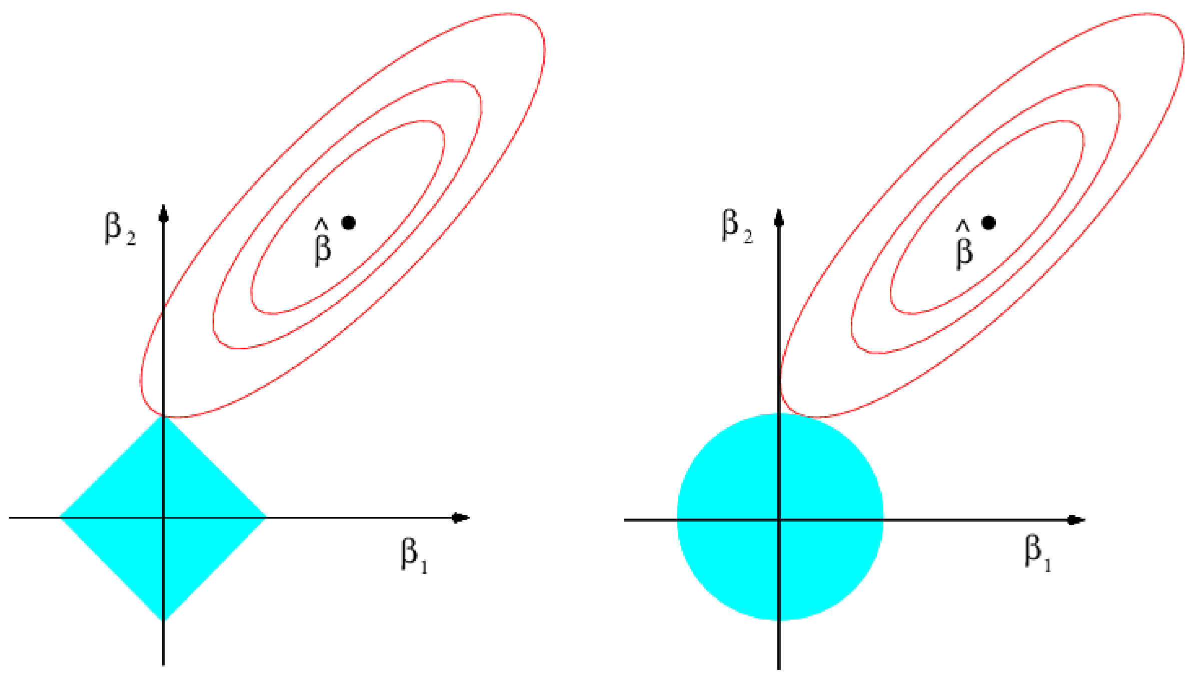
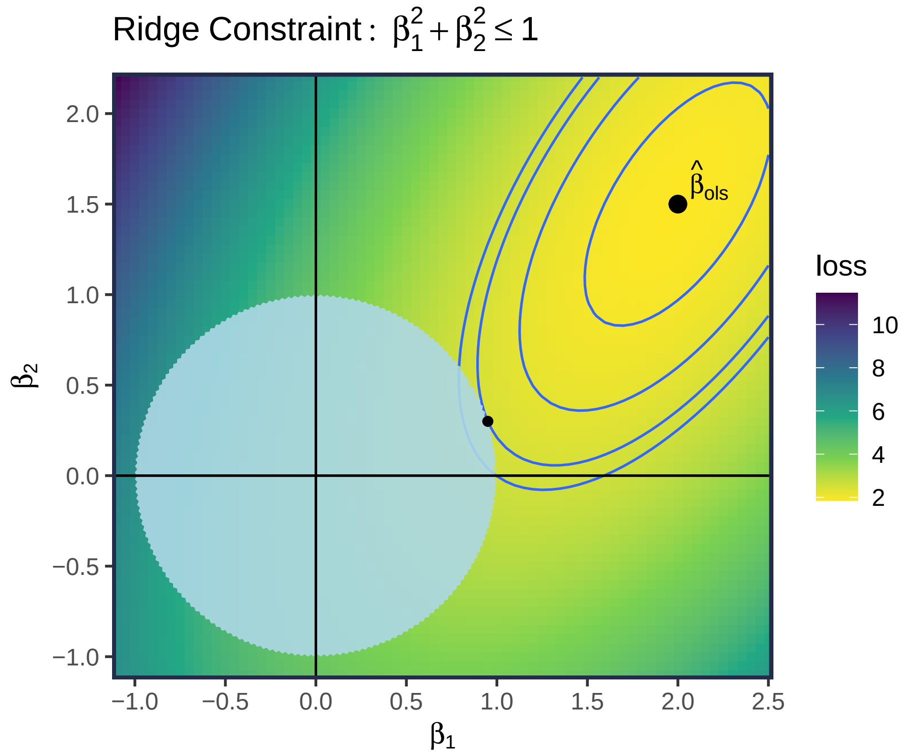
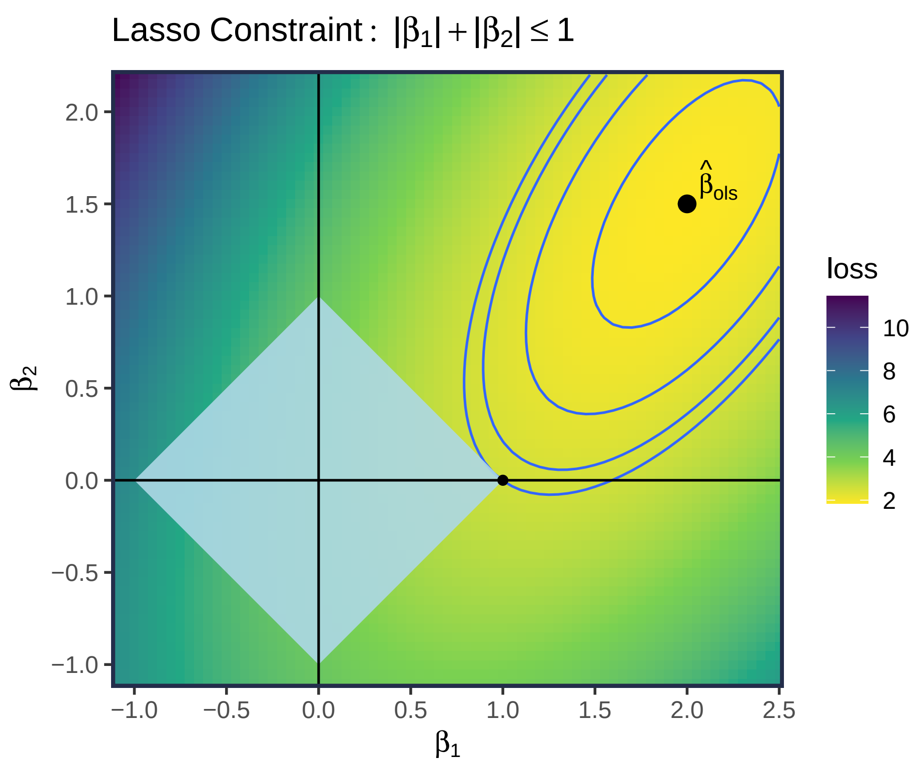


Figure 1: Contours of the error and constraint functions for the lasso (left) and ridge (right). The solid blue areas are the constraint regions, $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, while the red ellipses are the contours of the RSS (residual sum of squares).


Ridge Constraint : $\beta_1^2 + \beta_2^2 \leq 1$
$\hat{\beta}_{ols}$
loss
10
8
6
4
2
β_2
β_1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Lasso Constraint : $|\beta_1| + |\beta_2| \leq 1$
$\hat{\beta}_{ols}$
loss
10
8
6
4
2
β_2
β_1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

4 Ridge Regression

For ridge regression

$$l(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 = \text{MSE}$$
$$P(\beta) = \sum_{j=1}^{p} |\beta_j|^2 \qquad \text{(Notice that the intercept, } \beta_0\text{, is not penalized)}$$

Note

Watch for how software defines the loss. Some use MSE, others use SSE = n*MSE.
E.g., if loss is RSS = SSE = n*MSE, then $P(\beta)$ is a function of the sample size n!

So the ridge solution becomes:

$$\begin{aligned}
\hat{\beta}_\lambda^{\text{ridge}} &= \arg\min_\beta \ \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|^2 \\
&= \arg\min_\beta \ \text{MSE}(\beta) + \lambda\sum_{\text{j}=1}^{\text{p}}|\beta_{\text{j}}|^2 \\
&= \arg\min_\beta \ \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + n\lambda\sum_{j=1}^{p}|\beta_j|^2 \\
&= \arg\min_\beta \ \text{RSS}(\beta) + \text{n}\lambda\sum_{\text{j}=1}^{\text{p}}|\beta_{\text{j}}|^2
\end{aligned}$$

Your Turn #1 : Ridge Regression

1. What happens when $\lambda = 0$?
2. What happens when $\lambda \uparrow \infty$?
3. Why is it important to scale the predictor variables?

4.1 Scaling

Because the penalty is based on the magnitude of the coefficients, it is important to *scale* the predictors so they are all treated equally.

- This is important because predictors on different scales will be penalized with different strengths.
- The type of scaling that allows equal treatment is to **divide each predictor by its standard deviation**.
  - The resulting predictors should have the property: $\mathbf{x}_j^\mathsf{T}\mathbf{x}_j = c, \forall j$.

Consider the *advertising data*:

```
lm(sales ~ TV + radio + newspaper, data=advert) %>%
  broom::tidy()
```

term	estimate	std.error	statistic	p.value
(Intercept)	2.939	0.312	9.422	0.00
TV	0.046	0.001	32.809	0.00
radio	0.189	0.009	21.893	0.00
newspaper	-0.001	0.006	-0.177	0.86

- While the raw magnitude of `radio` > `TV`, the t-statistic (estimate/std.error) shows that `TV` has a stronger effect. This is because `radio` has a larger standard error.
- Look at what happens if we transform `newspaper` spending into thousands of dollars

```
lm(sales ~ TV + radio + I(newspaper/1000), data=advert) %>%
  broom::tidy()
```

term	estimate	std.error	statistic	p.value
(Intercept)	2.939	0.312	9.422	0.00
TV	0.046	0.001	32.809	0.00
radio	0.189	0.009	21.893	0.00
I(newspaper/1000)	-1.037	5.871	-0.177	0.86

```
- the coefficient of `newspaper` now has the largest magnitude!
- However we see that the t-statistic (and p-values) haven't changed
```

- Let's divide each predictor by its standard deviation

```
advert %>% summarize_all(sd)  # standard deviation
#> # A tibble: 1 x 4
#>      TV radio newspaper sales
#>   <dbl> <dbl>     <dbl> <dbl>
#> 1  85.9  14.8      21.8  5.22
```

```
advert %>%
  mutate_at(vars(-sales), ~.x/sd(.x))
#> # A tibble: 200 x 4
#>      TV radio newspaper sales
#>   <dbl> <dbl>     <dbl> <dbl>
#> 1 2.68  2.55       3.18  22.1
#> 2 0.518 2.65       2.07  10.4
#> 3 0.200 3.09       3.18   9.3
#> 4 1.76  2.78       2.69  18.5
#> 5 2.11  0.727      2.68  12.9
#> 6 0.101 3.29       3.44   7.2
#> # ... with 194 more rows
```

```
advert %>%
  mutate_at(vars(-sales), ~.x/sd(.x)) %>%
  lm(sales ~ TV + radio + newspaper, data=.) %>%
  broom::tidy()
```

term	estimate	std.error	statistic	p.value
(Intercept)	2.939	0.312	9.422	0.00
TV	3.929	0.120	32.809	0.00
radio	2.799	0.128	21.893	0.00
newspaper	-0.023	0.128	-0.177	0.86

- Notice that the `scaled` coefficients are `original x std.dev`.

term	original	std.dev	scaled
(Intercept)	2.939	NA	2.939
TV	0.046	85.85	3.929
radio	0.189	14.85	2.799
newspaper	-0.001	21.78	-0.023

- While this type of transformation won't impact any

Software

- Check the description of the implementation for penalized regression models. Most will properly scale the data for you behind the scenes. But if not, then you should do so first.
- In R, the `scale()` function can both center and scale the predictors. Centering is perfectly fine as it only impact the intercept term which isn't penalized.

Other Transformations

There are other types of transformations that re-scale the predictors.

1. Power transformations (e.g., Box-Cox)
2. Rank/Quantile scaling. Convert each value to its associated quantile or rank. E.g., the smallest value gets scored 1 and largest value gets scored n. Implicitly used by predictions trees.
3. Range scaling. E.g., $x' = \frac{\max - x}{\max - \min}$ to force between $[0, 1]$.

These could all be used, but the statistical scaling (standard deviation) is the most common. But use whichever approach gives the best predictions!

NB: Ensure all transformations are done on the training data and applied to the test data, else data leakage will occur.

4.2 Estimation

- Ridge regression has two types of parameters that need to be estimated
  1. Model parameters: β
  2. Tuning parameter: λ
- The tuning parameter λ controls the model complexity and effective degrees of freedom (edf).
- Given a specific value of λ, the model parameters β are easy to estimate as we show below.

The optimization function can be written:

$$\begin{aligned}
\hat{\beta}_\lambda^{\text{ridge}} &= \arg\min_\beta \; \ell(\beta) + \lambda P(\beta) \\
&= \arg\min_\beta \; J(\beta; \lambda)
\end{aligned}$$

where

$$J(\beta) = (\mathbf{y} - \mathbf{X}\beta)^\mathsf{T}(\mathbf{y} - \mathbf{X}\beta) + \lambda \beta^\mathsf{T}\beta$$

And the solution must satisfy

$$\frac{\partial J(\beta)}{\partial \beta} = \frac{\partial l(\beta)}{\partial \beta} + \lambda \frac{\partial P(\beta)}{\partial \beta} = \mathbf{0}$$

For ridge regression, this becomes

Ridge Regression Solution

Resources

The Matrix Cookbook has some common matrix and vector derivative expressions.

4.3 Ridge Regression Properties

$$\hat{\beta}_\lambda^{\text{ridge}} = \left(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda \mathbf{I_p}\right)^{-1} \mathbf{X}^\mathsf{T}\mathbf{Y}$$

- Ridge regression always works, even when $\mathbf{X}$ is not full rank because $\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda \mathbf{I_p}$ is always invertible for $\lambda > 0$
- Ridge tends to shrink correlated predictors together.

- For $0 < \lambda < 2\sigma^2/\sum_j |\beta_j|^2$, ridge regression has a lower mean square prediction error than least squares (Theobald 1974)!
- (Quasi) Bayesian Interpretation: If $\beta \sim N(0, \tau^2\mathbf{I_p})$ is the prior distribution, and τ and the standard deviation σ are assumed known, then the posterior mode (and hence mean since Gaussian) of β, given the data, is

$$\mathrm{E}[\beta|\mathcal{D}] = \left(\mathbf{X}^\mathsf{T}\mathbf{X} + \frac{\sigma^2}{\tau^2}\mathbf{I_p}\right)^{-1}\mathbf{X}^\mathsf{T}\mathbf{Y}$$

  which is equivalent to using $\lambda = \sigma^2/\tau^2$.

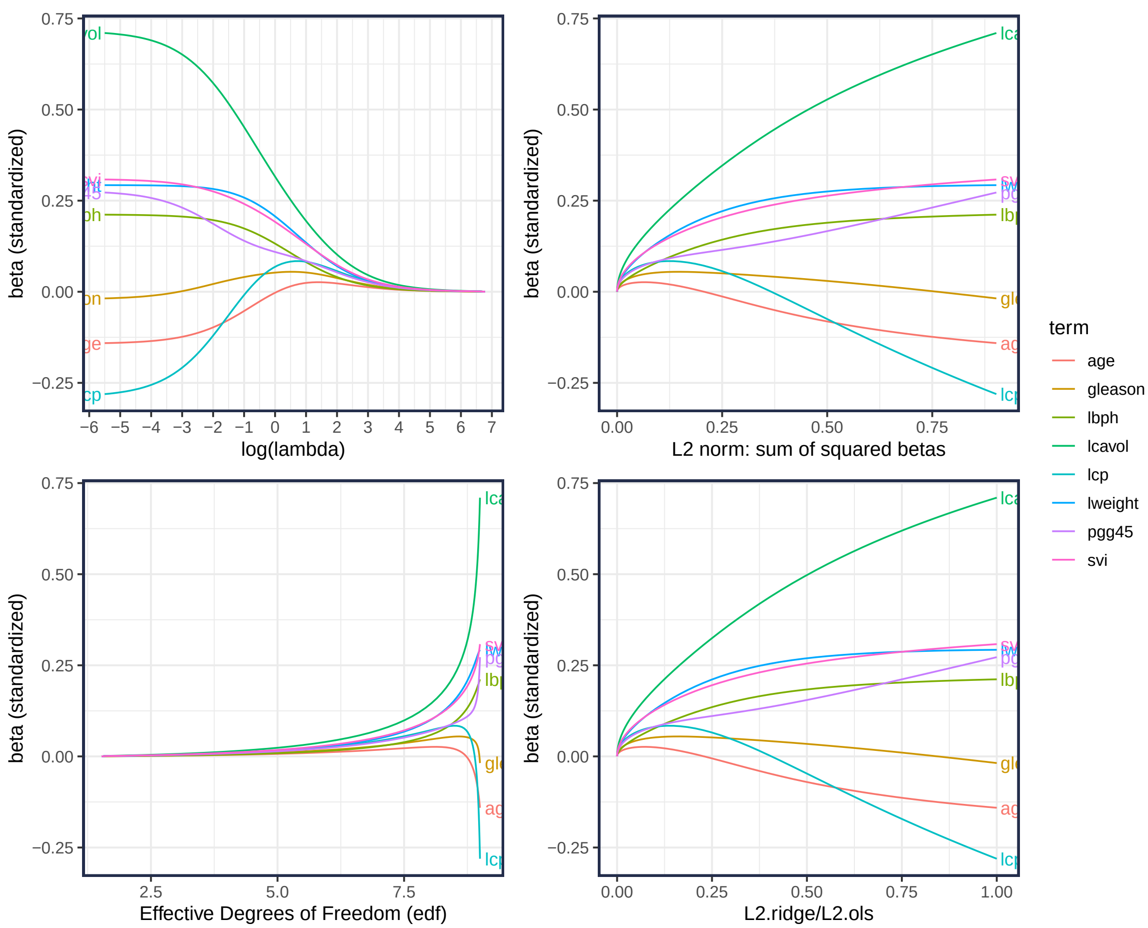
4.4 Solution Paths

Ridge Regression introduces a set of models indexed by λ

- $\lambda = 0$ gives β^{LS}
- $\lambda = \infty$ gives $\beta_j = 0 \quad j = 1, \ldots, p$
- As λ goes up, variance decreases and bias increases.

It can be illustrative to plot the *coefficient path* against:

a. λ or $\log(\lambda)$
b. $P(\hat{\beta}_j(\lambda)) = \sum_{j=1}^{p} |\hat{\beta}_j(\lambda)|^2$
c. $df(\lambda)$ (effective degrees of freedom)
d. $P(\hat{\beta}_j(\lambda))/P(\hat{\beta}_j(\lambda = 0))$ (ratio of ridge penalty to penalty at OLS solution)



4.5 Effective Degrees of Freedom (edf)

The *tuning parameter* for a ridge regression model is the λ that controls the strength of penalty

$$\hat{\beta}_{\lambda}^{\text{ridge}} = \arg\min_{\beta} \ \text{MSE}(\beta) + \lambda \sum_{\text{j}=1}^{\text{p}} |\beta_{\text{j}}|^2$$

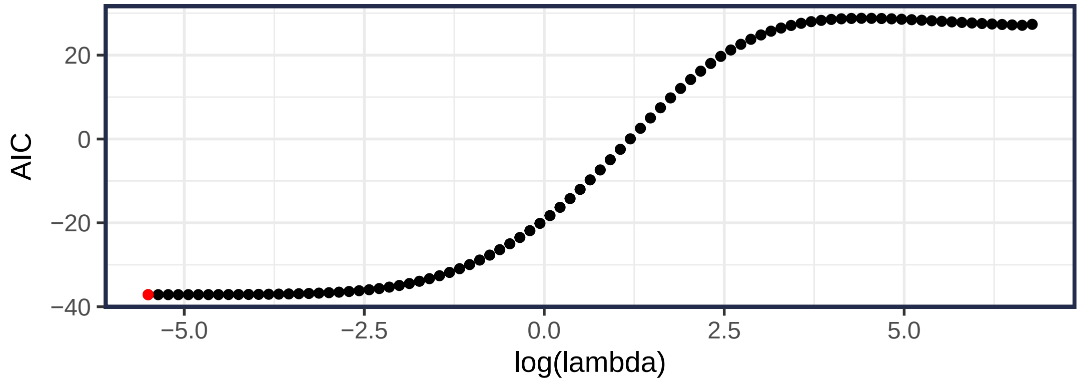
- $\lambda = 0$ gives β^{LS}
- $\lambda = \infty$ gives $\beta_j = 0 \quad j = 1, \ldots, p$
- As λ goes up, variance decreases and bias increases.

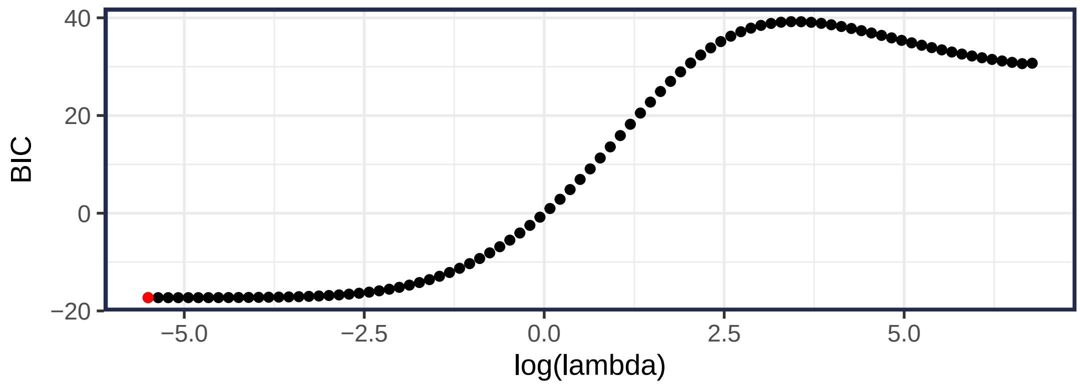
The *effective degrees of freedom*, $\mathrm{df}(\lambda)$ is the trace of the hat matrix, H_λ

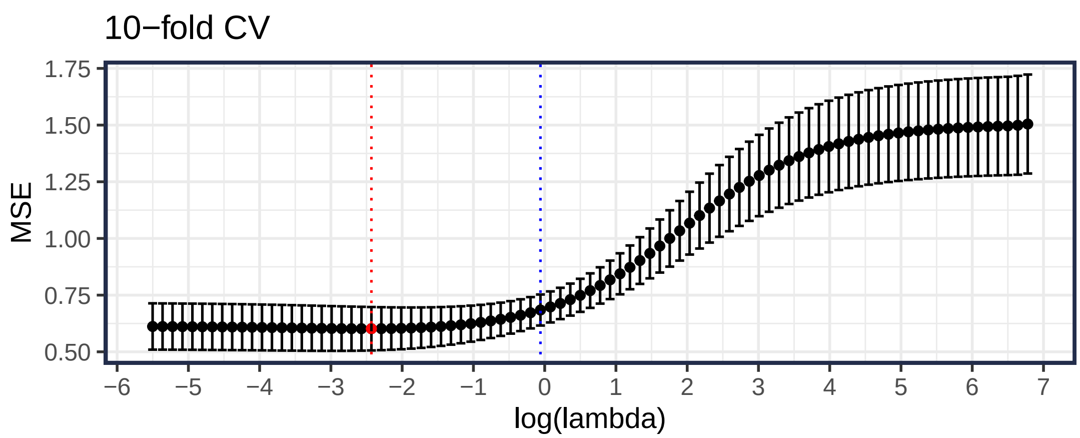
EDOF Calculation

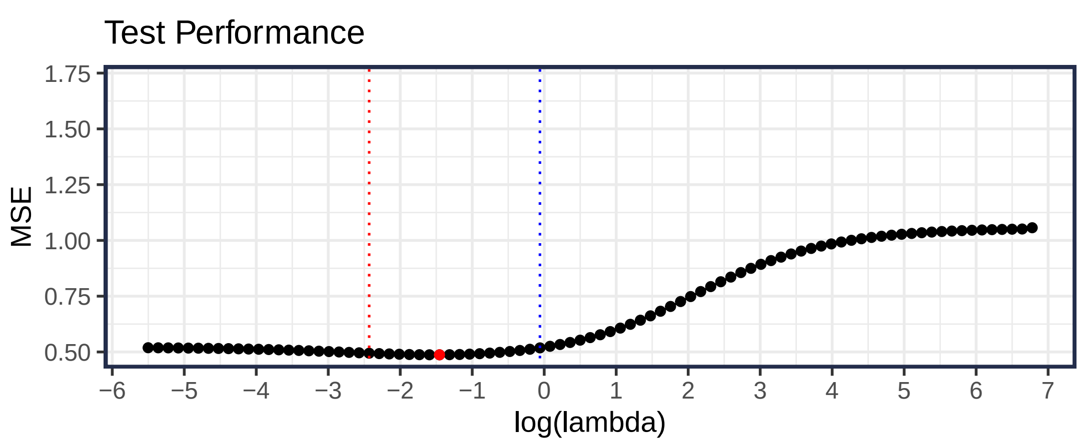
4.6 Tuning Parameter Selection

How about AIC/BIC or resampling?


AIC
20
0
−20
−40
−5.0
−2.5
0.0
2.5
5.0
log(lambda)


BIC
40
20
0
−20
−5.0
−2.5
0.0
2.5
5.0
log(lambda)


10−fold CV
MSE
1.75
1.50
1.25
1.00
0.75
0.50
−6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7
log(lambda)


Test Performance
MSE
1.75
1.50
1.25
1.00
0.75
0.50
−6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7
log(lambda)

4.7 Ridge Regression functions in R

There are several R packages that have functions for ridge regression.

- The `MASS` package has the function `lm.ridge()`
- The `glmnet` package has the functions `glmnet()` and `cv.glmnet()`
  - Use `alpha=0` for ridge regression
  - We will use this function for the *lasso* and *elasticnet* models
  - Note: input model matrix instead of data frame; does not handle formulas!
  - Check out An Introduction to `glmnet` for more information on using `glmnet` package for ridge, lasso, and elasticnet penalized regression.
- All of these functions will center, scale, and transform the output back to the original units, so you do not need to do any of the scaling yourself

4.7.1 `glmnet()`

We use demonstrate how to use `glmnet()` and `cv.glmnet()` using the prostate data. First load the data:

```
#-- Load raw data
data.url = 'https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data'
prostate = readr::read_tsv(data.url, col_select=-1) # remove row numbers
```

Then get the model matrices using `glmnet::makeX()`

```
#-- Get model matrices (returns a list of `x` and `xtest`)
X = glmnet::makeX(
  train = prostate %>% filter(train) %>% select(-lpsa, -train),
  test = prostate %>% filter(!train) %>% select(-lpsa, -train)
  )

X.train = X$x
Y.train = prostate %>% filter(train) %>% pull(lpsa)

X.test = X$xtest
Y.test = prostate %>% filter(!train) %>% pull(lpsa)
```

Note that we don't need to add a column of ones for the intercept term - `glmnet()` will add that for you with the default `intercept=TRUE` argument.

Now we can fit the ridge regression (`glmnet()` setting `alpha = 0`)

```
ridge = glmnet(X.train, Y.train, alpha = 0)
broom::tidy(ridge)   # predictions
#> # A tibble: 900 x 5
#>   term         step estimate lambda dev.ratio
#>   <chr>       <dbl>    <dbl>  <dbl>     <dbl>
#> 1 (Intercept)     1     2.45   879.  3.56e-36
#> 2 (Intercept)     2     2.43   801.  5.24e- 3
#> 3 (Intercept)     3     2.43   730.  5.74e- 3
#> 4 (Intercept)     4     2.43   665.  6.30e- 3
#> 5 (Intercept)     5     2.43   606.  6.91e- 3
#> 6 (Intercept)     6     2.43   552.  7.57e- 3
#> # ... with 894 more rows
```

Notice that `glmnet()` creates its own λ sequence. Internally, `glmnet()` uses a clever algorithm that sequentially decreases λ to quickly estimate the coefficients in about the same time it takes to run OLS! This means that you should not usually set lambda during the fitting. Instead, set the `predict()` argument `s` if you want to use a specific λ value.

glmnet lambda sequence

Notice the suggestion from the `help(glmnet)` concerning the `lambda` parameter:

> A user supplied lambda sequence. Typical usage is to have the program compute its own lambda sequence based on nlambda and lambda.min.ratio. Supplying a value of lambda overrides this. WARNING: use with care. Avoid supplying a single value for lambda (for predictions after CV use `predict()` instead). Supply instead a decreasing sequence of lambda values. `glmnet` relies on its warms starts for speed, and its often faster to fit a whole path than compute a single fit.

More often, we will want to use cross-validation to estimate λ. Helpfully, `glmnet` package provides a `cv.glmnet()` to implement k-fold cross-validation. Set `nfolds` to control the number of folds.

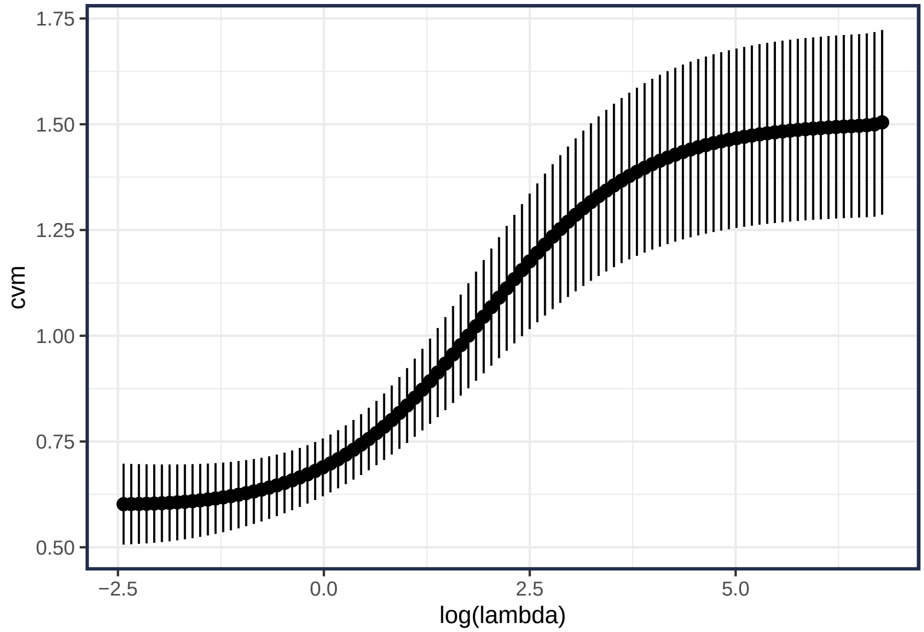
```
set.seed(2021)             # don't forget to set seed for the folds
ridge_cv = cv.glmnet(X.train, Y.train, alpha=0,
                     nfolds = 10)    # 10-fold cross-validation
```

We can get the performance over lambda using the `broom::tidy()` function

```
broom::tidy(ridge_cv)
#> # A tibble: 100 x 6
#>   lambda estimate std.error conf.low conf.high nzero
#>    <dbl>    <dbl>     <dbl>    <dbl>     <dbl> <int>
#> 1   879.     1.50     0.218     1.29      1.72     8
#> 2   801.     1.50     0.218     1.28      1.72     8
#> 3   730.     1.50     0.217     1.28      1.71     8
#> 4   665.     1.50     0.217     1.28      1.71     8
#> 5   606.     1.50     0.217     1.28      1.71     8
#> 6   552.     1.49     0.217     1.28      1.71     8
#> # ... with 94 more rows
```

Plot CV performance (function of $\log \lambda$)

```
with(ridge_cv, tibble(lambda, cvm, cvsd, cvup, cvlo)) %>%
  ggplot(aes(log(lambda), cvm)) +
  geom_pointrange(aes(ymin=cvlo, ymax=cvup))
```



And we have the optimal values of lambda (average mse)

```
ridge_cv$lambda.min # best lambda
#> [1] 0.08789
ridge_cv$lambda.1se # best lambda within one se of optimal
#> [1] 0.9873
```

Notice from the help that `cv.glmnet()` fits `nfolds+1` models; the extra fit using the entire training data. The `nfolds` fits are used to estimate the tuning parameter λ and then, for convenience, a final model is produced using all the available data. This allows you to get coefficients or make predictions without having to call (non-cv) `glmnet()`.

We can get a matrix of coefficients using the `coef()` function:

```
# coef(ridge_cv, s=ridge_cv$lambda) %>% head # estimates for all lambda values
coef(ridge_cv, s = "lambda.min")            # using lambda.min
#> 9 x 1 sparse Matrix of class "dgCMatrix"
#>                     s1
#> (Intercept)  0.095550
#> lcavol       0.492656
#> lweight      0.601228
#> age         -0.014818
#> lbph         0.137966
#> svi          0.679288
#> lcp         -0.116653
#> gleason      0.017256
#> pgg45        0.007078
```

And make predictions on the test data:

```
# yhat = predict(ridge_cv, X.test, s = ridge_cv$lambda) # matrix; all lambda
yhat = predict(ridge_cv, X.test, s = "lambda.min")
mean( (Y.test - yhat)^2 )  # test MSE
#> [1] 0.4944
```

5 Lasso

5.1 The Lasso

For lasso regression

$$l(\beta) = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2$$
$$P(\beta) = \sum_{j=1}^{p} |\beta_j| \qquad \text{(Notice that } \beta_0 \text{ is not penalized)}$$

The lasso solution becomes:

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

Why is it important to scale the predictor variables?

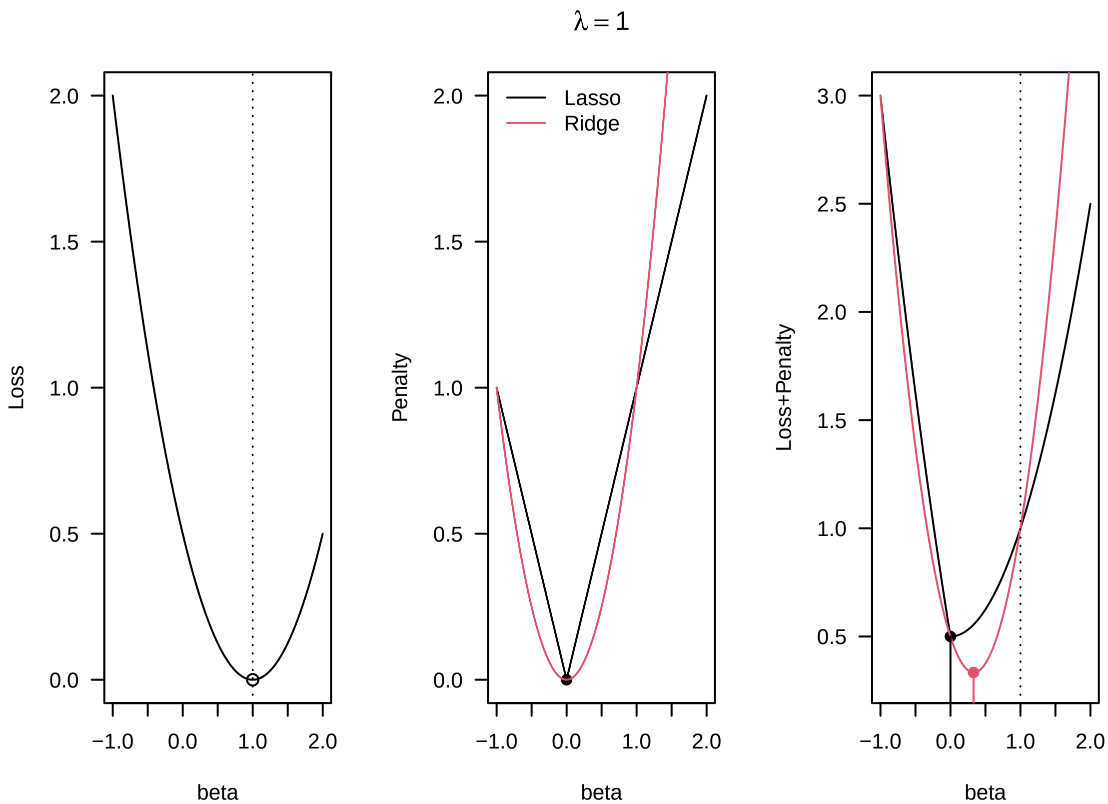
5.2 Lasso Penalty

- By using a L_1 penalty, lasso penalty can shrink some coefficients all the way to 0 (unlike the ridge penalty)
- This effectively removes predictors from the model (like the stepwise procedures), but in a type of continuous fashion
- Lasso stands for "Least Absolute Shrinkage and Selection Operator"

5.3 Example of 1D Lasso Selection

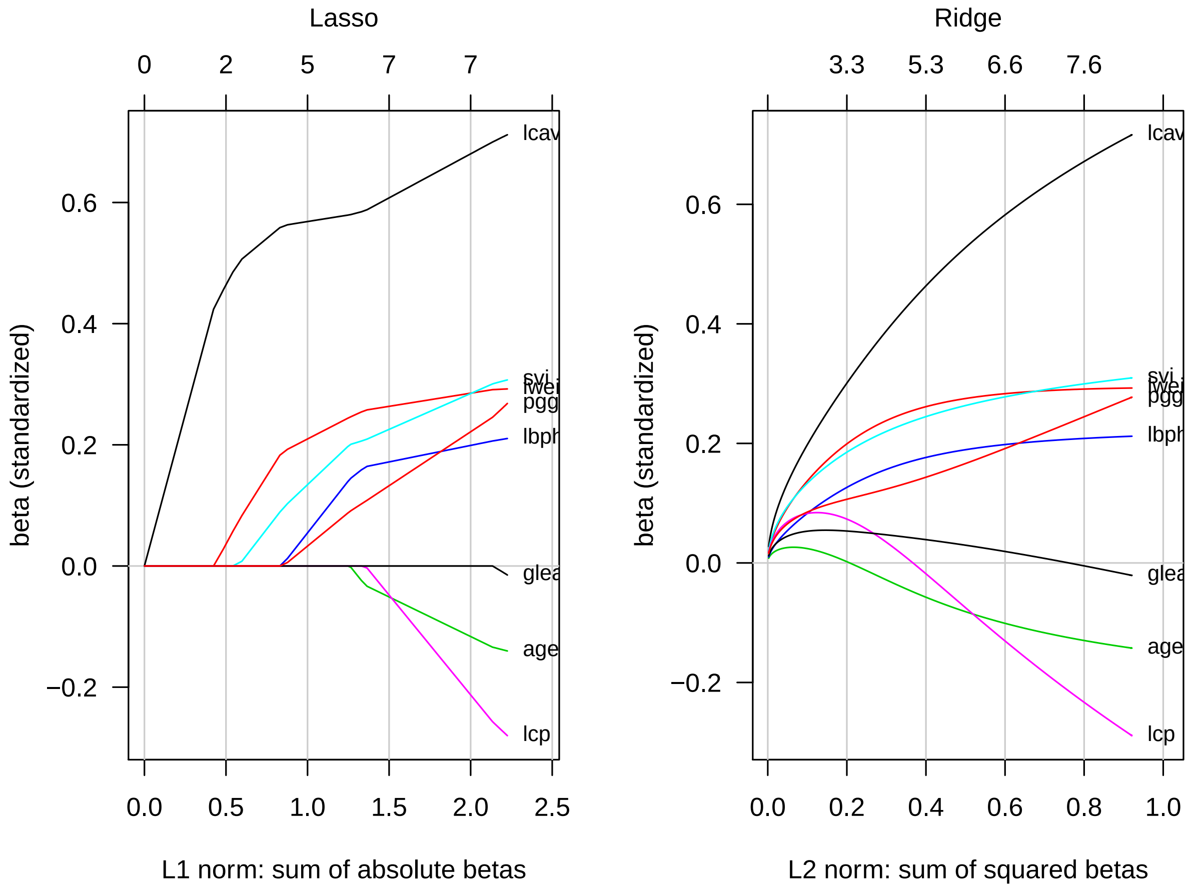
Suppose the simplified setting of fitting a loss function of $l(\beta) = \frac{1}{2}(1 - \beta)^2$.

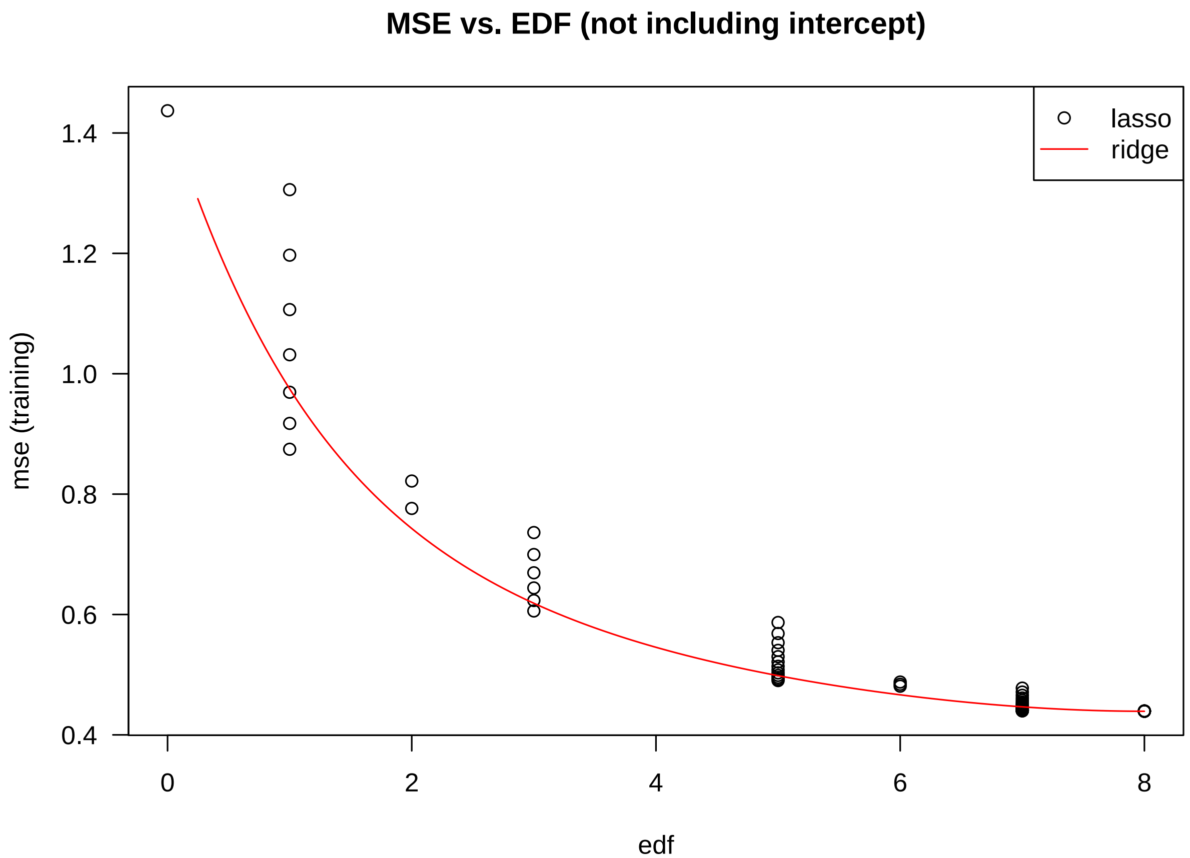
- This loss is the squared deviation from 1.
- The lasso penalty is $|\beta|$.
- The objective function is $l(\beta) + \lambda|\beta|$



5.4 Comparing Lasso and Ridge Regression

Prostate Cancer Data from ESL book: Figs 3.8, 3.10 and Table 3.3

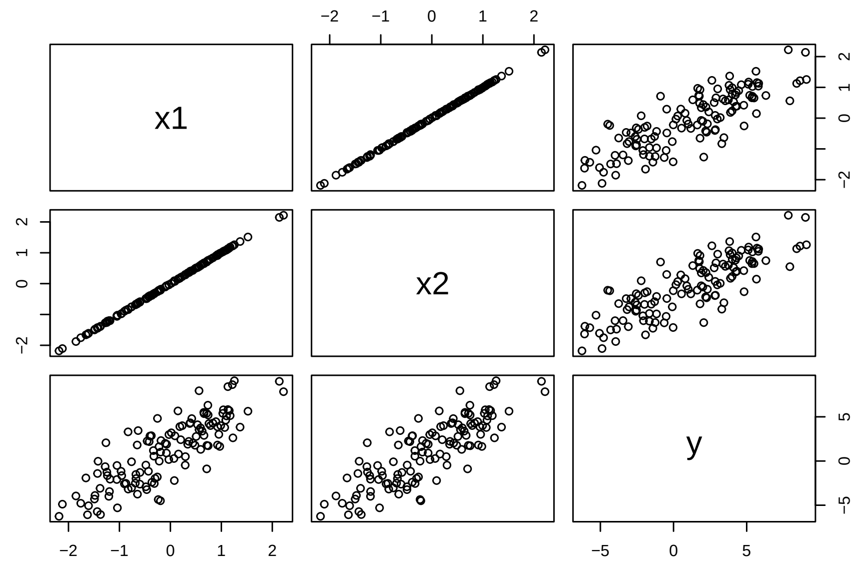

Lasso
0 2 5 7 7
beta (standardized)
lcav
svi
lwei
pgg
lbph
glea
age
lcp
0.0 0.5 1.0 1.5 2.0 2.5
L1 norm: sum of absolute betas
Ridge
3.3 5.3 6.6 7.6
L2 norm: sum of squared betas
0.6 0.4 0.2 0.0 −0.2
0.0 0.2 0.4 0.6 0.8 1.0

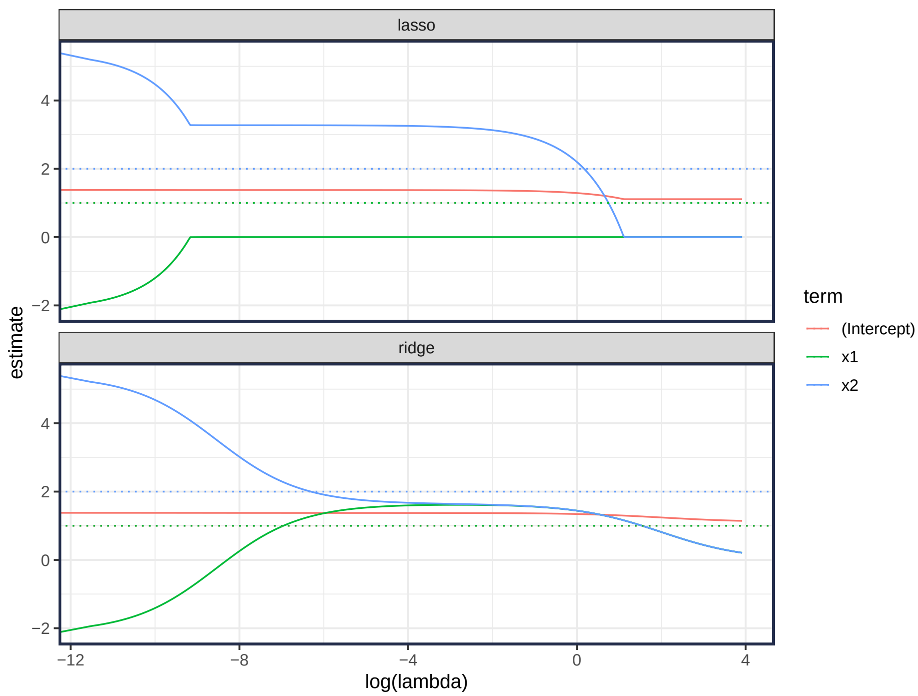


5.4.1 Example with Strong Correlation

Consider a problem with strong multicollinearity:

```
#-- Generate Data
set.seed(10)
n = 125
x1 = rnorm(n)
x2 = rnorm(n, mean=x1, sd=.01)
cor(x1,x2)                              # strong correlation
#> [1] 0.9999
y = rnorm(n, mean=1+1*x1+2*x2, sd=2)   # f(x) = 1 + 1x_1 +2x_2

#-- Pairs Plot
pairs(cbind(x1, x2, y))
```





predictor	true	ols	ridge	lasso
(Intercept)	1	1.38	1.36	1.37
x1	1	-3.30	1.58	3.15
x2	2	6.57	1.57	0.11

Ridge and Lasso using $\lambda_{\min}$ from cross-validation.

- Notice that the OLS coefficients have negative signs and large magnitude but a small constraint (penalty) produces a much closer result.

- That is, a small ridge or lasso penalty controlled the high variance.
- Ridge tends to shrink correlated predictors together
- Lasso tends to choose one and set other(s) to zero.

5.5 Effective Number of Parameters for Lasso

- Unlike ridge regression, the lasso is *not* a linear smoother. There is no way to write $\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$.
- Thus, estimating the effective degrees of freedom is not based on trace of hat matrix.
- It turns out that the number of non-zero coefficients is a decent approximation of the effective number of parameters
- We can use this value ($df = \sum_j \mathbb{1}(|\beta_j| > 0)$) in AIC/BIC/GCV for selecting λ
  - Note: the df is not continuous in λ, so the min SSE model would have smallest λ within the set with $df = k$

5.6 Lasso Regression functions in R

The `glmnet()` function with `alpha = 1` implements lasso regression.

See the above section on Ridge Regression in R for more details.

```
#-- Load raw data
data.url = 'https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data'
prostate = readr::read_tsv(data.url, col_select=-1) # remove row numbers
```

```
#-- Get model matrices (returns a list of `x` and `xtest`)
X = glmnet::makeX(
  train = prostate %>% filter(train) %>% select(-lpsa, -train),
  test = prostate %>% filter(!train) %>% select(-lpsa, -train)
  )

X.train = X$x
Y.train = prostate %>% filter(train) %>% pull(lpsa)

X.test = X$xtest
Y.test = prostate %>% filter(!train) %>% pull(lpsa)
```

I'm going to fit several different models using cross-validation. To ensure equality, I'm going to set the fold structure manually and apply to all models using the `foldid` argument in `cv.glmnet()`

```
#- Get K-fold partition (so consistent to all models)
set.seed(721)                    # set seed for replicability
n.folds = 10                     # number of folds for cross-validation
fold = sample(rep(1:n.folds, length=nrow(X.train)))
# vector of fold labels
# notice how this is different than:  sample(1:K,n,replace=TRUE),
#  which won't give equal group sizes
```

Now we can fit several models

```
#-- OLS
fit.ls = lm(Y.train~X.train)
beta.ls = coef(fit.ls)
yhat.ls = cbind(1, X.test) %*% coef(fit.ls)

#-- Ridge
```

```r
fit.ridge = cv.glmnet(X.train, Y.train, alpha=0, foldid=fold)
beta.ridge = coef(fit.ridge, s="lambda.min")
yhat.ridge = predict(fit.ridge, newx = X.test, s="lambda.min")

#-- Lasso
fit.lasso = cv.glmnet(X.train, Y.train, alpha=1, foldid=fold)
beta.lasso = coef(fit.lasso, s="lambda.min")
yhat.lasso = predict(fit.lasso, newx = X.test, s="lambda.min")

#-- Elastic Net
a = .8  # set alpha for elastic net
fit.enet = cv.glmnet(X.train, Y.train, alpha=a, foldid=fold)
beta.enet = coef(fit.enet, s="lambda.min")
yhat.enet = predict(fit.enet, newx = X.test, s="lambda.min")
```

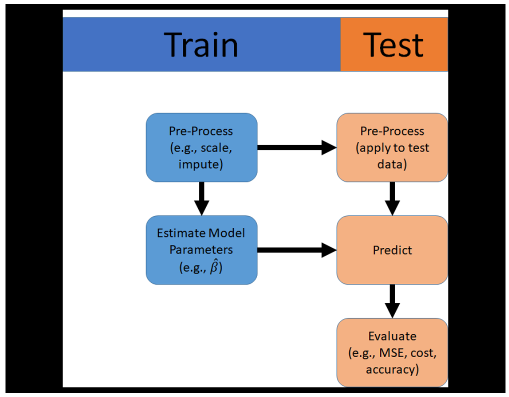
And evaluate their performance on the test data

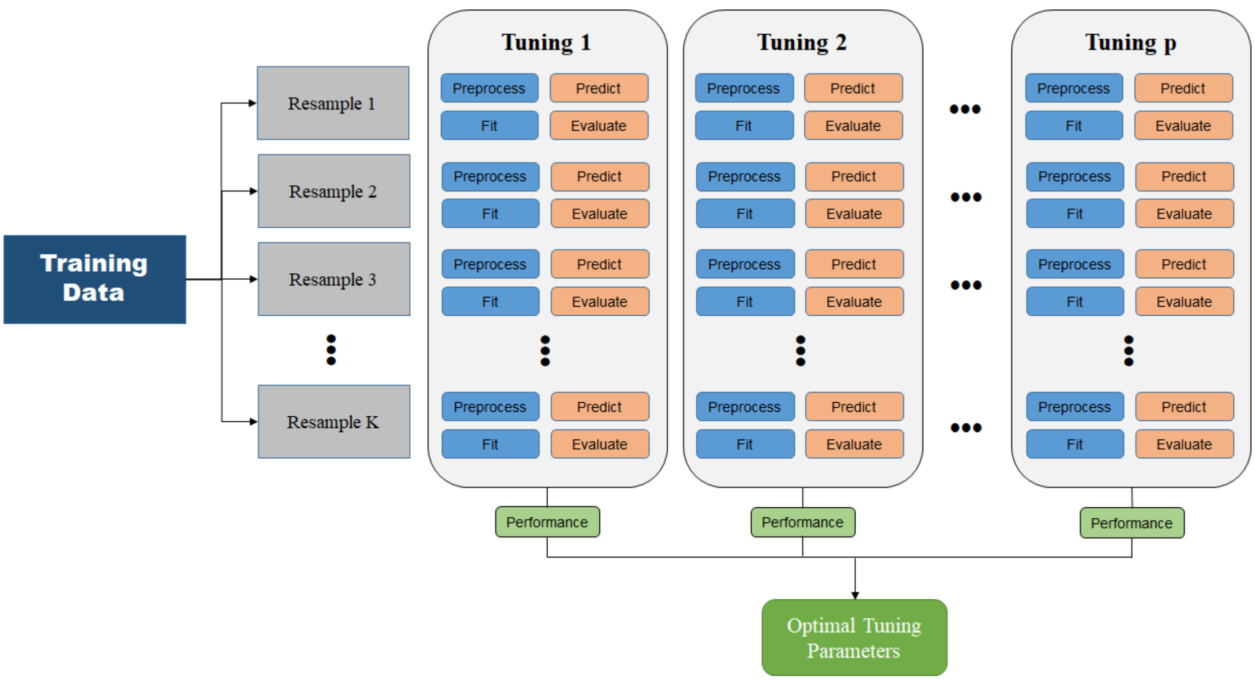
```r
#-- MSE
YHAT = list(ols = yhat.ls, ridge = yhat.ridge,
              lasso=yhat.lasso, enet=yhat.enet)

purrr::map_dbl(YHAT, ~mean( (Y.test-.x)^2 )) %>% sort
#>   enet  lasso  ridge    ols
#> 0.4904 0.4912 0.4944 0.5213

# sapply(YHAT, function(yhat) mean((Y.test - yhat)^2))
```

```r
#-- Coefficients
tibble(variable=c("(Intercept)", colnames(X.train)),
       ols = beta.ls,
       ridge=beta.ridge[,1],
       lasso=beta.lasso[,1],
       enet = beta.enet[,1])
```

variable	ols	ridge	lasso	enet
(Intercept)	0.429	0.096	0.155	0.152
lcavol	0.577	0.493	0.541	0.537
lweight	0.614	0.601	0.593	0.593
age	-0.019	-0.015	-0.015	-0.014
lbph	0.145	0.138	0.134	0.134
svi	0.737	0.679	0.662	0.660
lcp	-0.206	-0.117	-0.139	-0.135
gleason	-0.030	0.017	0.000	0.000
pgg45	0.009	0.007	0.007	0.007

5.7 Cross-Validation and Penalized Regression





5.8 Elastic Net

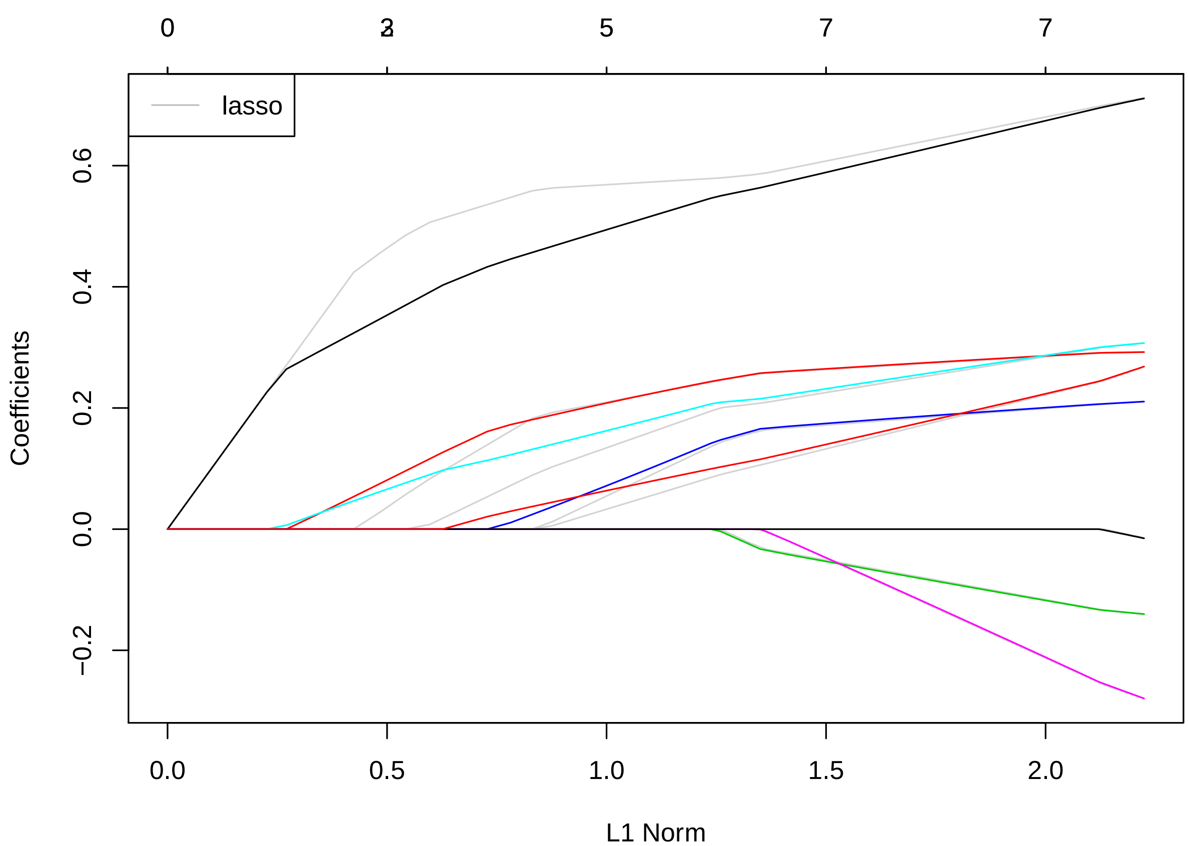
The Elastic Net Penalty can help with selection (like lasso) and shrinks together correlated predictors (like ridge).

$$P(\beta, \alpha) = \sum_{j=1}^{p} \alpha\beta_j^2 + (1-\alpha)|\beta_j| \qquad \text{Eq 3.54 on pg 73 of ESL}$$

$$P(\beta, \alpha) = \sum_{j=1}^{p} \frac{(1-\alpha)}{2}\beta_j^2 + \alpha|\beta_j| \qquad \text{glmnet R package}$$

5.8.1 Comparing Elastic Net to Lasso and Ridge

Elastic Net with $\alpha = 0.5$



5.8.2 Elastic Net Tuning

Notice that elastic net models have two *tuning parameters*: $\alpha \in [0, 1]$ controls the type of penalty ($\alpha = 0$ for ridge, $\alpha = 1$ for lasso) and $\lambda \geq 0$ controls the strength of penalty.

While `cv.glmnet()` provides a fast implementation to estimate λ, there is not a shortcut that I know of for evaluating α; a basic loop over a grid of α values will work.

Note

There are few ways to jointly optimize α and λ.

1. Create a grid of α and λ values and fit models for every value. The problem is that the algorithm in `glmnet()` is optimized to search over a sequence of λ values in one pass. Also, due to the different penalties a reasonable λ sequence for a lasso penalty, it may not be reasonable for ridge penalty.
2. Use one resampling pass to estimate α, then another to estimate λ given $\hat{\alpha}$.
3. The choice of $\alpha > 0$ may not be very influential on performance, so may not have much practical need to optimize.

GRAPHIC of Cross-validation scaling within each fold

5.9 Relaxed Lasso

The relaxed lasso is basically an approach to use lasso for variable selection. The approach is as follows:

1. Use cross-validation to select λ.
2. Find the non-zero coefficients and select only those predictors to be in the model.
3. Fit an unpenalized (e.g., OLS) model using the selected features.

This attempts to use the lasso variable selection to reduce variance, but unpenalized estimation to reduce bias in the model parameters.

The `relax` argument of `glmnet()` and `cv.glmnet()` provide a convenient implementation. Note that in prediction, the γ argument allows a balance between the penalized and unpenalized fits.

5.10 Categorical Predictors in Penalized Regression

1. How does lasso/ridge treat categorical predictors?
2. How does lasso/ridge treat interaction terms?
3. How does lasso/ridge treat basis expansions of a single variable, e.g. polynomial?

5.11 Group Lasso

- L groups of predictors
  - categorical variable with 3 levels will be in a group of 3 predictors
- Let X_l be $n \times p_l$ matrix of group l predictors
- β_l is $p_l \times 1$ group coefficients

$$J(\beta) = \ell(\beta) + P(\beta, \lambda)$$

$$\ell(\beta) = \left\| Y - \beta_0 1 - \sum_{l=1}^{L} X_l \beta_l \right\|_2^2$$

$$P(\beta, \lambda) = \sum_{l=1}^{L} \sqrt{p_l} \, \|\beta_l\|_2$$

6 More Resources

- glmnet tutorial
- broom tutorial
  - Using broom with glmnet
- ISLR Lab 6.5.2