

DS 740

Data Mining

Ridge Regression, LASSO, and Elastic Net with Assessment
Regression with Penalties

Important note: Transcripts are **not** substitutes for textbook assignments.

Learning Objectives One

By the end of this lesson, you will be able to:

- State the constrained formulations for the Ridge Regression (RR), LASSO, and Elastic Net model fit goal functions.
- Understand basic behavior of penalized regression as shrinking coefficients toward origin.
- Visualize constraints as restrictions in the coefficient space.
- Understand pathway model fit of penalized regression.
- Explain Elastic Net as a “compromise” between RR and LASSO.
- Compare and contrast penalized regression methods.



Learning Objectives Two

By the end of this lesson, you will be able to:

- Fit penalized regression models using `glmnet` function in R.
- Understand need for validation methods.
- Apply cross-validation to penalized regression models using `cv.glmnet` function in R, for the purpose of model selection.
- Apply bootstrapping to estimate error of coefficients.



Standardization

Constrained optimization treats all coefficients on same scale:

$$g(\beta_0, \dots, \beta_p) \leq s$$

Predictors can have very different means and standard deviations.

Typically, standardize predictors: Subtract mean and divide by standard deviation.

Result: all have mean 0 and st. dev. 1.

Default option in `glmnet` function.

As we saw with k-nearest neighbors, when we want to treat predictors approximately equally with some sort of distance or size measure, we typically opt to standardize the values. That is, we subtract the mean and divide by the standard deviation for each predictor so that all wind up with a mean of 0 and a standard deviation of 1. As coefficients for regression depend on the size or the magnitude of predictor values and we are measuring size of the associated coefficients within the penalty, we will standardize predictors by default in our penalized regression model fitting.

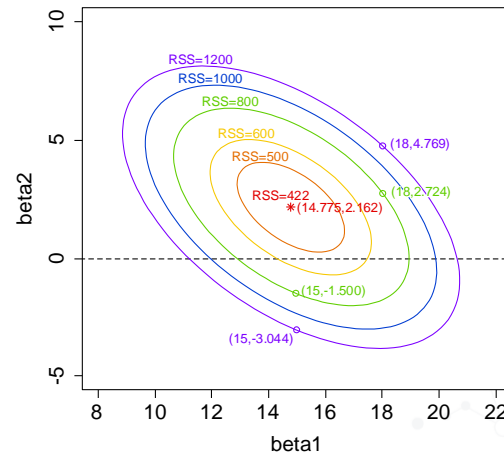
Visual Optimization for Linear Regression

Multiple linear regression
with two predictors:

$$\min_{\beta_0, \beta_1, \beta_2} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2})^2$$
$$= \min_{\beta_0, \beta_1, \beta_2} RSS$$

Ovals: (β_0, β_1) pairs that produce
predictions with various RSS values

Red star (*): least-squares (β_0, β_1)



To visualize how constraints work, we will begin by looking at the minimization of the RSS, or residual sum of squares, in our usual multiple regression model using two predictors for simplicity. The graph at the right shows possible choices for the β_1 and β_2 regression coefficients. And the ovals track pairs of coefficients that predict response values, which will have the listed RSS values.

For example, using either of the purple pair of coefficients produces predicted values, which have a residual sum of squares of 1,200. Moving in smaller, inner ovals finds coefficients with lower residual sum of squares until we get to the least squares solution, shown as the red star at the center. This set of coefficients would produce predicted values with the minimum possible residual sum of squares of 422.

Ridge Regression (RR)

Ridge Regression (RR) Optimization

Penalized:

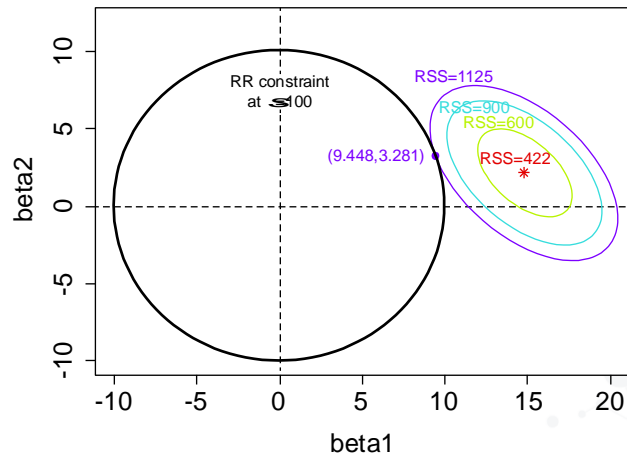
$$\min_{\beta_0, \dots, \beta_p} \left\{ RSS + \lambda \cdot \sum_{j=1}^p \beta_j^2 \right\}$$

Constrained:

$$\begin{aligned} \min_{\beta_0, \dots, \beta_p} \quad & RSS \\ \text{subject to} \quad & \sum_{j=1}^p \beta_j^2 \leq s \end{aligned}$$

Example constraint used:

$$\text{is } \beta_1^2 + \beta_2^2 \leq 100$$



We now consider our first penalized regression method. We look at both the penalized and constrained formulations for the method known as ridge regression, abbreviated as RR. The name is derived from the original concept and computation of possible solutions.

The function of the coefficients that is used for the penalty and constraint is the sum of the squared beta values. Note that this restricts beta values to be more similar and to not have particular terms take over. Also note that we do not constrain the intercept. We'll take a look at a visual example to understand why we don't have particular terms take over or be unselected from the model.

Looking at the same example from the previous page in which we have concentric circles moving outwards from the least squares solution at the red star, we find where the circles intersect with our constraint. The constraint used in this example is that the sum of the squared coefficients is less than or equal to 100. Because that constraint region is circular, the concentric circles for residual sum of squares will generally almost never meet that at a point where one of the coefficients will be zero. So that means we find a solution that has both non-zero beta 1 and beta 2.

RR Properties

- Originally introduced by Hoerl & Kennard in 1970.
- As λ increases, reduces variance of coefficients with trade-off in increased bias.
- "Shrinks" coefficients (overall) towards the origin but does not set any to zero.
- Solution has closed form: $\hat{\beta}^{RR} = (X^T X + \lambda I)^{-1} X^T y$

The ridge regression method was the first penalized regression model introduced in 1970 to reduce variance and coefficients with a tradeoff in increased bias. The variance bias tradeoff occurs as coefficients are constrained and shrunk towards the origin. And in that way, their variance must be reduced. This method was applicable before intensive computing methods were available, since its solution has a closed matrix form - very similar to that of multiple linear regression. However, a result of the nice answer to the minimization is that this method cannot do predictor selection. In other words, it does not set any of the coefficients equal to 0.

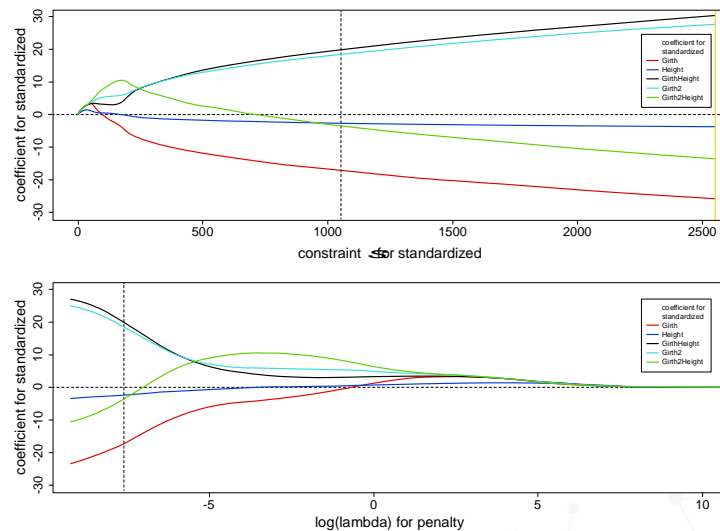
Trees Example: RR Coefficient Pathways

Constraint* at multiple regression solution:

$$\sum_{j=1}^5 \hat{\beta}_j^2 = 2551.4$$

Dotted line shows all 5 predictors retained at $s = 1052.6$ and $\log(\lambda) = \log(0.0005)$

*on standardized predictors



In the top plot, we observe the behavior of the coefficients as the constraint tuning parameter s changes. Note that the coefficient shrink-- overall they get smaller as the value of s decreases. So we see a widening of the pathways from left to right.

Correspondingly, the bottom plot shows the behavior of the coefficients across different values of λ . And here because of the inverse relationship between λ and s , the shrinkage in coefficients occurs as λ increases. Note that this is actually plotted against the log of λ for purposes of saving space.

The 5 coefficient values for a single fit are shown as the values along the dotted line drawn on each plot. The values of the coefficients are the points at which the colored lines-- the pathways of the coefficients-- intersect with that dotted line. Note that the values of each coefficient are the same in the two plots. In other words, they are at the same solution.

So the s of 1,052.6 corresponds to the λ of 0.0005 in our constrained and penalized formulations. That is, the coefficients are merely tracked differently between the two plots, either by constraint or by penalty tuning parameters. And we use whichever one is more convenient for a particular purpose.

Least Absolute Selection and Shrinkage Operator (LASSO)

Least Absolute Selection and Shrinkage Operator (LASSO) Optimization

Penalized:

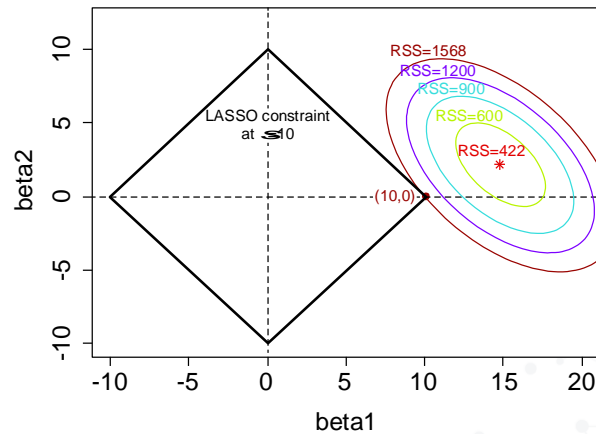
$$\min_{\beta_0, \dots, \beta_p} \left\{ RSS + \lambda \cdot \sum_{j=1}^p |\beta_j| \right\}$$

Constrained:

$$\begin{aligned} \min_{\beta_0, \dots, \beta_p} \quad & RSS \\ \text{subject to} \quad & \sum_{j=1}^p |\beta_j| \leq s \end{aligned}$$

Example constraint used is:

$$|\beta_1| + |\beta_2| \leq 10$$



We move on to our second penalized regression model. The penalized and constrained formulations for the least absolute selection and shrinkage operator-- also known as LASSO-- look very similar to those for a ridge regression, except we're now using a different function of the coefficients in the penalty and constraint. The function of the coefficients used is the sum of the absolute values. This keeps the sign of the coefficients the same, but does not constrain larger values differently than smaller values.

The visual illustration of LASSO parameter selection in our previous example is shown again here by concentric circles representing the different values of residual sum of squares, moving outward from the least squares solution at the red star. We look for the point at which the circles intersect with our constraint, which in this case for the LASSO, is absolute value of beta 1 plus absolute value of beta 2 equals 10.

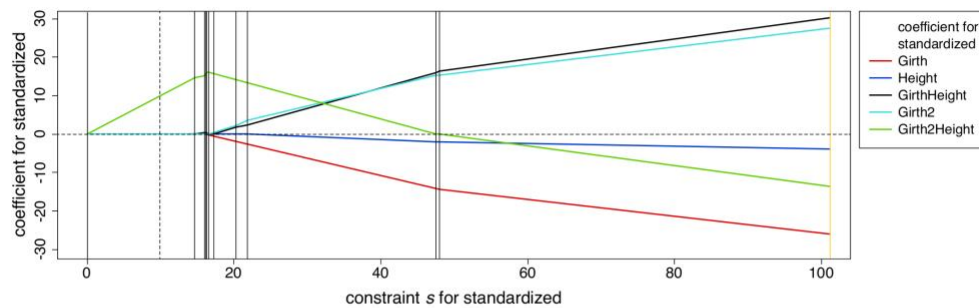
Visually, note that this constraint has corners in the beta 1 beta 2 space. Because of this, the intersection easily occurs at corners, meaning that one of the coefficients-- in this case, beta 2-- is selected to be zero. Equivalently, this means variable 2 is dropped from the model. While this is only a two variable case, we use it to see why corners-- so to speak-- in the constraint space as defined by the LASSO constraint allow us to drop predictors.

LASSO properties

- Originally introduced by Tibshirani in 1996.
- As λ (in penalty) increases, reduces variance of coefficients with trade-off in increased bias.
- “Shrinks” coefficients (overall) and coefficients can be set to 0.
- Solution does *not* have closed form: must use quadratic programming.

Computing methods had progressed enough to introduce the more computationally intensive LASSO in 1996. Quadratic programming methods were used in the original solution. As for the coefficients that result, bias variance tradeoff occurs with a penalty on the sum of the absolute value of the coefficients. But now, variable selection can also occur, because the constraint has the so-called corners in the predictor space.

Trees Example: LASSO Coefficients Versus s, λ



Constraint* at multiple regression solution:

$$\sum_{j=1}^p |\beta_j| = s = 101.3$$

*on standardized predictors

Dotted line shows only 1 predictor retained at $s = 10$

We again, observe shrinkage of coefficients as the constraint tuning parameter s decreases. And correspondingly, we would see coefficient shrinkage as the penalty tuning parameter λ increased. We only show the visual for the constraint because there is a special characteristic of the coefficient pathways when viewed across the constraint. And that characteristic is that the pathways are piecewise linear between step points across values of the constraint tuning parameter s . This linearity however, does not hold across λ values. Note also that coefficient can be shrunk to zero and this linearity is part of that.

Elastic Net (E-Net)

Elastic Net (E-Net) Optimization

Penalized: $\min_{\beta_0, \dots, \beta_p} \left\{ RSS + \lambda \cdot \left[(1 - \alpha) \cdot \sum_{j=1}^p \beta_j^2 + \alpha \cdot \sum_{j=1}^p |\beta_j| \right] \right\}$

Constrained:

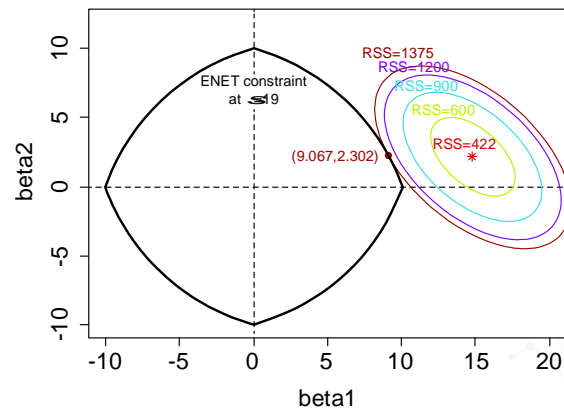
$$\min_{\beta_0, \dots, \beta_p} RSS$$

subject to

$$(1 - \alpha) \cdot \sum_{j=1}^p \beta_j^2 + \alpha \cdot \sum_{j=1}^p |\beta_j| \leq s$$

Example constraint used is:

$$0.1 \cdot (\beta_1^2 + \beta_2^2) + 0.9 \cdot (|\beta_1| + |\beta_2|) \leq 19$$



The final penalized regression method we look at is elastic net. The penalty is now a combination between those of ridge regression and LASSO. It is an average weighted by a new tuning parameter alpha of the sum of squared coefficients and the sum of absolute values of the coefficients. This results in a compromise, both mathematically and purpose wise, between the two previous methods. Note that alpha equaling 0 results in simply ridge regression, and alpha equaling 1 results in the LASSO.

Similar to the pictures we looked at previously, we are once again looking at the point where the circles representing the different values of residual sum of squares intersect with our constraint. The constraint here is again, a weighted average-- I picked an alpha of 0.9 to show the corners a little bit more clearly-- a weighted average between the sum of the squared coefficients and the sum of the absolute values of the coefficients. When we take a look at the constraint area, it is not straight edged. But it does have corners.

And because it has corners, the intersect at the intersection with the residual sum of squares space can occur at the corners. In this particular example, it does not occur that way. But such a thing is possible, again, because the corners in the constraint space are able to meet that residual sum of squares circle.

Elastic-Net Properties

- Originally introduced by Zou and Hastie in 2005.
- As λ (in penalty) increases, reduces variance of coefficients with trade-off in increased bias.
- “Shrinks” coefficients (overall) and coefficients can be set to 0.
 - $\sum_{j=1}^p \beta_j^2 \rightarrow$ balance highly correlated predictors
 - $\sum_{j=1}^p |\beta_j| \rightarrow$ parameter selection
- Solution does not have closed form.

Very similar properties hold for elastic net as those discussed previously. The combination of the two penalty functions allows for a compromise between the characteristics of the ridge regression and LASSO methods. It allows for both a balance among the highly correlated predictors, as well as selection of the predictors as needed. Note that pathways of coefficients can be visualized similarly, and they are linear as with the LASSO across values of the constraint tuning parameter s . And also like that LASSO, variable selection is allowed because the constraint-- the sum of the absolute values of beta-- plus the sum of the squares of beta weighted has corners in predictor space, as we saw in the previous slide.

Computation for Fitting Penalized Regression Models

Fitting Models

Computation based on original papers

- RR: `lm.ridge` from *MASS* package (no automatic standardization)
- LASSO: `lars` from *lars* package → Full “pathways” of coefficients
- Elastic Net: `enet` from *elasticnet* package → Full “pathways” of coefficients

Comprehensive computation

- All penalized: `glmnet` from *glmnet* package → full “pathway”
- **Pro**: one overall computational algorithm, generally works well
- **Con**: specific coefficient accuracy depends slightly on choices for /

Original papers for each model fitting method have unique packages in R. These packages do align better with the author's original model fitting procedures. However, there is a more convenient model fitting package available called `glmnet`. And this fits generalized linear models, including penalized models. Since it is so comprehensive, the help file is rather dense. But as long as the response is continuous, the model fit defaults to penalized and works very well, particularly if lambda values are specified.

Notes:

Package Manual: [glmnet](#) (pdf)

Fitting Models – using glmnet

1. Specify list of potential lambda values:

```
lambdalist = c(close_to_zero...large_values)
lambdalist = exp(c(neg_values...pos_values))
```

2. Apply function glmnet to fit each model type:

- RR ($\alpha=0$): `penfit = glmnet(x, y, alpha=0, lambda = lambdalist)`
- LASSO ($\alpha=1$): `penfit = glmnet(x, y, alpha=1, lambda = lambdalist)`
- ENET ($0 < \alpha < 1$): `penfit = glmnet(x, y, alpha= α , lambda = lambdalist)`

3. Obtain coefficients and predicted values from model fit:

```
coef(penfit, s=lambdavalue)
predict(penfit, newx=newxmatrix, s=lambdavalue)
```

When fitting models using Glmnet, we fit across a pathway of lambda values. And so the first thing we need to do is specify a list of lambda values, which can either be a list starting at 0 and going up to large values or an exponent of a list ranging from negative to positive values. The first will generally have equally-- will be set up for equally sequenced spaced values. The second will generally be set up to have denser values in the small lambdas, which tends to be preferred in terms of fitting to a variety of models.

The next step is to fit the appropriate model type. For ridge regression, this means using the Glmnet function on X matrix and Y response with alpha equal to zero and lambda specified to be lambda list. For lasso, the only thing that changes is alpha set equal to 1. For elastic net, we set alpha equal to some value between 0 and 1.

After having stored the model fit, we can take off coefficients using the COF function at a particular lambda value and/or use the Predict function on the fit of the model to predict at some new x, which has to be an X matrix with the same number of columns as was used in the original model fit, and at a particular lambda value.

Trees Example: Application

trees data, $n = 31$, $p = 5$

- Response: Volume
- Predictors: original two Girth, Height, with three transformations *GirthHeight*, *Girth2*, *Girth2Height*

Lambda values: `lambdalist = exp((-100:100)/10)`

Find (back on original scale):

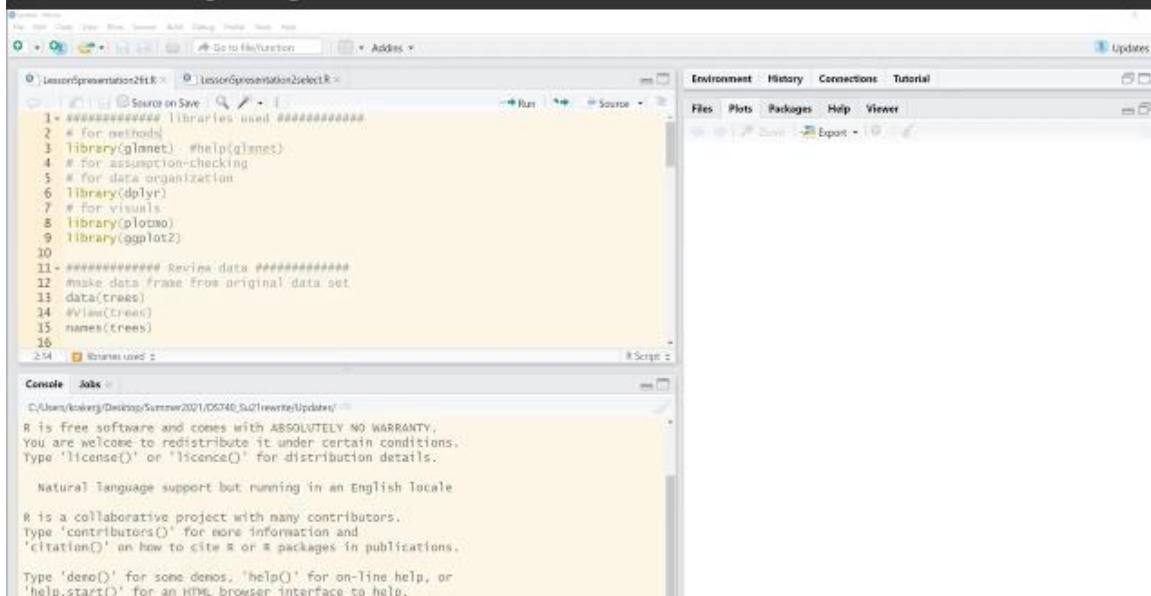
- RR ($\alpha = 0$) across lambdalist \rightarrow coef at $\lambda = 1$
- LASSO ($\alpha = 1$) across lambdalist \rightarrow coef at $\lambda = 0.2$
- ENET ($\alpha = 0.75$) across lambdalist \rightarrow coef at $\lambda = 0.4$



We revisit the trees data for fitting penalized regression models. While we only have five predictors, they are very highly correlated, because three of them are functions of the other two. And we would thus like to make an application of penalized regression methods to reduce the variability of the estimators and potentially do predictor selection as well at least for lasso and elastic net.

We'll be starting with a lambda list that is denser in the lower values-- that is, in the values close to zero-- and will fit ridge regression lasso and elastic net with an alpha of 0.75 just a possible value between 0 and 1. And then we'll pick out a particular lambda to take a look at for each of those fitted models, and we'll talk more later on in this presentation about how those lambdas should be optimally selected.

DS740 – Ridge Regression, LASSO, and Elastic Net with Assessment



```
1 ##### libraries used #####
2 # for methods
3 library(glmnet) #help(glmnet)
4 # for assumption-checking
5 # for data organization
6 library(dplyr)
7 # for visuals
8 library(plotmo)
9 library(ggplot2)
10
11 ##### Survive data #####
12 #make data frame from original data set
13 data(trees)
14 #view(trees)
15 names(trees)
16
```

Console

```
C:\Users\bokeg\Desktop\Summer2021\DS740_Su21write\Updates\
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
```

This slide represents a video/screencast in the lecture. The transcript does not substitute video content.

In today's presentation, we're going to be walking through fitting penalized regression models using our trees data set with some additional predictors as we've defined before. If you have not already defined the new data set with the additional predictors, let's run through that. In addition, we'll need some libraries to start with.

We'll be using the glmnet library for our model-fitting methods. We do not have any specific assumption checking, but we will be using dplyr for data organization and ggplot2 as well as a new library called plotmo. If you'd like to follow along with the visuals, those would be good libraries to have installed.

Starting again with our trees data set, we reorganize so that the response is at the beginning just for visualizations. So we have volume, girth, and height. We use the mutate function to add on three additional predictors again as we did in the prior presentation. And now we use the model matrix function on volume fit on the remaining five predictors, the remaining five variables, on this transform data set. And we exclude the first column, which is just a column of ones.

So I could for clarity take a look at the first few rows. And I see that this is just a matrix of those five predictor variables. We need that for use in the glmnet function. That is one of the inputs that is required. We also need to define our response. And I'm just going to call that y for ease of input. We'll also define the dimensions of our data. n is the number of observations. p is the number of predictors that we're using.

If I take a look at the same multiple linear regression that I fit in the previous presentation, we see the same basic output. Only now, we have a little x in front of our variable names because these are the variable names from the x matrix. And so that's how R labels those variables.

Here I chose a breadth of lambda values. And it might be useful to-- you can plot these, or you can take a look at, say, the min value in `lambdalist` or the max value in `lambdalist` to get an idea of the range. But the intention of this exponential over a pretty cut-up range of values is to get a breadth of options for potential lambdas.

As we've spoken about, our penalized regression will fit across all the lambdas. So we'll actually be coming up with a pathway of coefficients. And we're going to use the `plot.glmnet` function from the `plotmo` package to visualize that.

So fittingly, penalized regression is actually pretty straightforward. Once we've defined our x and our y, we use the alpha to specify which type of model. If it's ridge regression as it is here, we use alpha equals zero, and then we've already identified a range of potential lambda values. And again, we need to make sure that our lambda values themselves are all positive. But exponent of this range from negative to positive numbers makes all positive numbers.

And we get this nice little visual over here that tracks the coefficients, so for example, such as for girth, across either log lambda or lambda. So you can actually see the actual lambda penalties on an exponential scale, or take a look at the log lambda, which is displayed linearly.

The downside of this, the way that the default algorithm is explained, though, is you can see that log lambda goes from positive down to negative. So an option is to use `x var equals lambda`. And what that option allows us to do is to display the x scale for lambda in increasing order which is how we're used to seeing an x scale as we see it there.

All right, at this point, I'm just going to pick out a lambda value. So I'm going to pick out a lambda value of 1, which means that my log lambda-- my log of `RRlambdaused`, for ridge regression, `lambdaused` equals zero. And so what that looks like is I'm effectively taking a cross section of this pathway of coefficients at this point. So for example, the coefficient for girth squared should be, oh, maybe about 0.05. The coefficient for girth looks like it's going to be about 0.7.

So we can apply the `coef` function to the output from our `glmnet` fit at a particular chosen value of lambda. Now, oddly, the input name for this is `s equals`. And that's due to how this is tracked in the algorithm, but that's how you name that argument for the lambda used.

So if we run this, we see indeed-- we said at this cut point, it looks like girth squared-- yup, looks like about 0.05, a girth about 0.7. We could, if we wanted to, display this a little bit. Or nicely, we could just take r around $RRcoef$ to six decimal places. And that gives us nonscientific notation. There are other options for displaying in nonscientific notation, but that's an easy one to apply.

So we're going to go through the same process with `LASSOfit`. Effectively, the only thing we're changing in `glmnet` is the α argument. And so we're going to set α equals 1 for LASSO. And when we plot this, we see some of the pathways hit zero and just stay there. And that was visualized in our lecture discussion how some of the variables are effectively trimmed to zero.

So let's see this. As an example, I'm going to pick a λ of 0.2. So I'm plotting the log of 0.2, which lands about right here. And it looks like I should only have two, maybe three positive or I should say nonzero coefficients looks like for girth to height and girth times height or girth height.

So let's take a look at the coefficients. And yup, along with the intercept, of course, which we're always going to fit a nonpenalized intercept, we've got a coefficient for girth height and a coefficient for girth to height. And the other coefficients are dropped to zero, which means that those variables are not included as part of our fitted equation.

`ENETfit` can be applied with a variety of α values, anywhere between 0 and 1. I picked 0.75 for no particular reason. That's just one of a potential infinite number of values I could pick. If I were fitting elastic net, I would fit this for a variety of α s just to get a deeper variety of model options.

And same process here. Now I'm using a λ of 0.4. And we see a set of coefficients, nonzero coefficients, for looks like girth height, girth squared, and girth squared height. And we see those at this log λ value are the nonzero pathways.

So how well did each of these different models do at least when applied to the same data used to fit the model? Well, one way of assessing that is to compare our observed y values to the predicted y values or \hat{y} hats.

So I'm going to use the `predict` function on each of my three fits with `newx` just set to be-- to feed in the original x matrix at the particular selected λ value. And by the way, if you're wondering where those λ values come from, why did I pick that particular λ value? Well, we're going to talk about model selection in the next part of this presentation.

So I now have stored all of those \hat{y} values. I've organized them in a common vector called `y_hat` along with three repeats of my y vector. And if you want, you can take a

look at what's inside the `obs.pred` data frame to see how this was organized, but effectively we're stacking both the observed and predicted values for each of ridge regression, LASSO, and elastic net, and then we're going to plot these.

And so what we can observe here is, well, there's not a lot of difference between these model fits. That is, the y observed corresponds pretty much to the \hat{y} in the same way for all three methods. And that has to do with the particular λ values that I picked out for this situation.

You will notice perhaps a couple unusual things. These symbols appear to be darker. That's because there are actually two points plotted over each other here. And the biggest differences tend to be at the end points. And that has to do with ridge regression not being able to trim predictors while the other two methods are able to trim predictors, or at least that's the primary reason for that.

If we wanted to do a little bit of cross comparison, we could take a look at the coefficients, rounded coefficients side by side in a data frame. And as I said, with ridge regression, none of the predictors can be trimmed. That is, they all have nonzero coefficients whereas both LASSO and elastic net tend to trim out-- well, they both trim out girth and height. And LASSO also terms out girth squared. We will talk in the next part of this presentation about how to get or how to select those λ s.

Question 1

DS740 - Ridge Regression, LASSO, and Elastic Net with Assessment

Self Assessment

When fitting the LASSO model with $\lambda = 0.5$, how many predictors were retained?

☐ 5

☐ 4

☐ 3

☐ 2

☐ 1

Submit

Answer is at the end of the transcript

Assessment of Fitting Penalized Regression Models

Applying Cross-Validation and Bootstrap

It is important to select λ (also referenced as value s) carefully to best predict response; note that the coefficients can be very different.

How?

Using **cross-validation** for model selection

- Can write by hand as done previously
- Easier application: `cv.glmnet` from *glmnet* package

Reduces variance of coefficients (trade-off is increased bias).

How
to
Verify?

Using **bootstrap**

- Review Lesson 2, Presentation 2, as needed
- Use the function `glmnet` to estimate coefficients
- Need to write wrap-around function to input data and apply `glmnet`

Fitting the penalized regression models is relatively straightforward with *Glmnet*. But which model should we ultimately select? This depends on the selection of the tuning parameters α and λ , which is also referenced as s when we come up with the coefficients for the model. These in turn define the model to be fit according to the particular α and λ values.

To properly select the model we will be using cross-validation methods for model selection. As we've seen previously, we could write loops to conduct cross-validation computations. However, the *Glmnet* package has a ready made function `cv.glmnet` that does all the work for us.

We will implement that through an example noting that penalized regression methods are aimed to reduce the variability of coefficients. Because there are fewer assumptions about the model itself, there is less theoretical basis for computing standard error of the estimated coefficients.

So we move into a more computationally heavy process here for estimating standard errors by using the bootstrap. And you may find it helpful to revisit the presentation too from lesson 2 for review.

Computing CV measure – using `cv.glmnet`

1. Specify list of potential lambda and alpha values:

```
lambdalist = c(close_to_zero...large_values)
allalpha = c(...) # select a breadth of values
```

2. Apply function `cv.glmnet` to compute CV measure for models at each considered α [RR ($\alpha=0$), LASSO ($\alpha=1$), ENET ($0 < \alpha < 1$)]:

```
CVpenfit = cv.glmnet(x, y, alpha= $\alpha$ , lambda = lambdalist,
                    nfolds=nfolds, foldid=cvgroups)
```

3. Visualize CV measure across values of $\log(\lambda)$:

```
plot(CVpenfit)
```

4. Obtain lambda for “best” model, at a considered alpha:

```
CVpenfit$lambda.min # best lambda & log
CVpenfit$lambda.1se # within 1SE
min(CVpenfit$cvm)    # minimum of CV values, from $cvm
CVpenfit$cvm         # in order of lambda from LARGEST to smallest
```

We start, as before, by defining a Lambda list-- a set of possible Lambda values-- as well as a list of all alphas that we want to consider, which generally are equally spaced across the interval from 0 to 1. We'll apply function `cv.glmnet` for computing the cv measure for models at a particular alpha understanding that we can fit that across the entire pathways of potential lambda. So very similar initial syntax to how we use `Glmnet`.

We will need two additional inputs to the `cv.glmnet` function. Specifically, the number of folds and, if we like-- and we generally will specify this-- the fold IDs. That is the specific groupings that we want to use. An additional visualization of finding the minimum is found by simply plotting the stored output from the `cv.glmnet` fit, and this will plot the cv measure across values of $\log \lambda$. This is another reason that we often consider defining `r` lambda list as an exponent along with getting denser lambda values close to 0.

Finally, from the stored cross-validated results we can pick off the minimum lambda-- the lambda within one standard error-- and we can take a look at the actual cv measures themselves where we have to recognize that the cv measures are ordered consistent with the Lambda from largest to small. So effectively they're in backwards order from how we generally define Lambda. So just be careful if you're taking a look at the `cvm` output from `cv.glmnet`.

Body Fat Example Application

Using `bodyfat.csv` data set, with the author's description and adjustments (errors in *density* and *height* have been corrected)

- **Response:** *BodyFatSiri* (using Siri's equation)
- **Predictors:** consider all 14 predictors
- **Lambda values:** `lambda1ist = exp((-1200:100)/10)`
- **Cross-validation:** compute $CV_{(10)}$ to select between λ -values for each of RR ($\alpha = 0$), LASSO ($\alpha = 1$), Elastic-Net ($\alpha = 0.95$), and Elastic-Net ($\alpha = 0.50$).
- **Bootstrap:** to estimate the variability of the coefficient estimates for the model selected via cross-validation.

To demonstrate these cross-validation methods we will go back to the body fat example. It's a somewhat larger data set having more predictors, and many of those predictors are highly correlated. This fits in well with one of our purposes for using penalized regression.

We'll start with a wide variety of Lambda values again denser towards closer to 0 for the potential models that we'll consider. We're going to consider four alphas just for a start to get a little bit more variety. Ridge regression with alpha equals zero, Lasso with alpha equals 1, and elastic net with alphas of 0.95 and of 0.50.

We'll then use cross-validation to compute our cv measure for the purposes of model selection using the function `cv.glmnet`. We could consider different alphas as well as continuing to use cross-validation to select between the alphas, but for the purposes of this exercise we'll just work with the four possible alphas listed on the screen. At the last part of this, we will use bootstrap to estimate the variability of the coefficient estimates for the final model that we select.

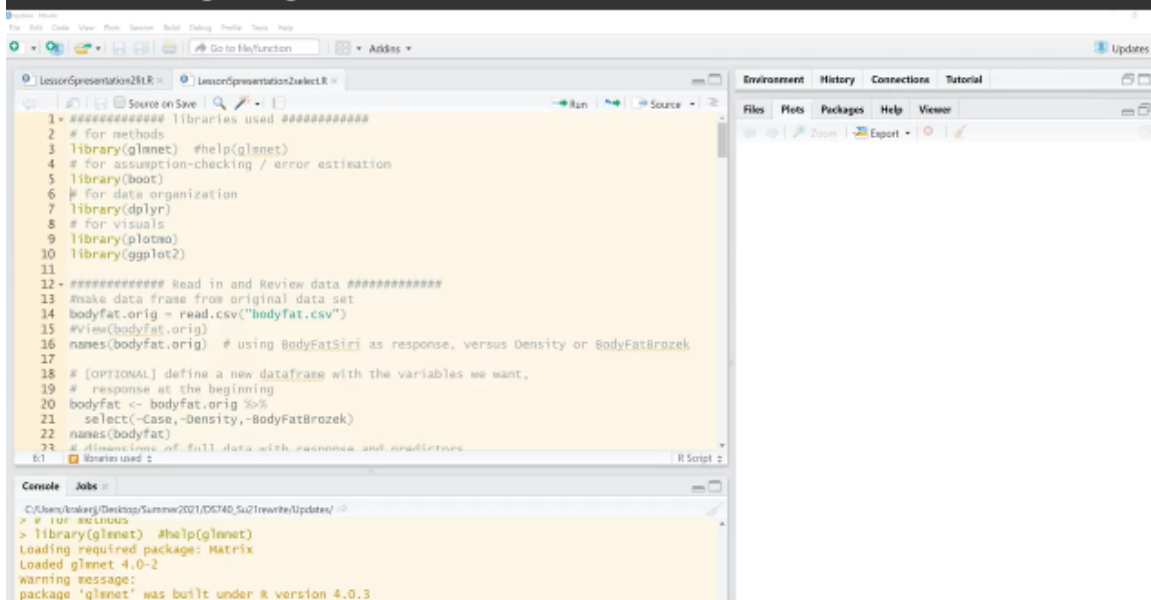
Notes:

Find the `bodyfat.csv` file in the online course.

Descriptions and adjustments of the data set:

[Fitting Percentage of Body Fat to Simple Body Measurements](#) - Roger W. Johnson

DS740 - Ridge Regression, LASSO, and Elastic Net with Assessment



```
1 ##### libraries used #####
2 # for methods
3 library(glmnet) #help(glmnet)
4 # for assumption-checking / error estimation
5 library(boot)
6 # for data organization
7 library(dplyr)
8 # for visuals
9 library(plotmo)
10 library(ggplot2)
11
12 ##### Read in and Review data #####
13 #make data frame from original data set
14 bodyFat.orig = read.csv("bodyfat.csv")
15 #View(bodyFat.orig)
16 names(bodyFat.orig) # using BodyFatSiri as response, versus Density or BodyFatBrozek
17
18 # [OPTIONAL] define a new dataframe with the variables we want,
19 # response at the beginning
20 bodyFat <- bodyFat.orig %>%
21   select(-Case, -Density, -BodyFatBrozek)
22 names(bodyFat)
23 # dimensions of full data with response and predictors
24 # libraries used :
```

Console Jobs

```
C:\Users\bnkary\Desktop\Summer2021\DS740_Su21\newrte\Updates/ >
> # for methods
> library(glmnet) #help(glmnet)
Loading required package: Matrix
Loaded glmnet 4.0-2
warning message:
package 'glmnet' was built under R version 4.0.3
```

This slide represents a video/screencast in the lecture. The transcript does not substitute video content.

We now take a look at a selection of penalized regression models. And for this presentation, we will be using the body fat data set as it is a little bit more complex in terms of numbers of predictors. We'll begin by opening up the required libraries, including GLMNet and Boot, which are needed for fitting the models and for bootstrapping, respectively. We'll use DPLYr player for organization and a couple libraries for visuals.

We have the data set stored. And we'll read that in as we've done before. We can take a look at the names. Of course, you can actually view the data set itself. One of the things that I'll be doing is a bit of organization of the data frame. I'm going to take the original data frame and pipe it to get rid of the predictor variables that are not needed. So when I run that line and take a look at the names, my response is the first listed variable. And then I had the 14 remaining variables as predictors. Size of this data frame, 252 observations. And as I said, it's a bigger data frame with 14 predictors.

As we did before, in terms of setting up the use of our glmnet, we'll need to make a model matrix of the predictor variables. So fitting the response and the remaining variables and pulling the matrix, the model matrix, the predictor matrix from that, that removing the first column of ones gives us our x. Always good to do a quick dimension check here, 252 by 14. I can also take a look at the first row of x and make sure it contains what I think it's supposed to contain.

Response variable n and p . And again, just typing them in to make sure they are the values I think they should be. We'll take a quick look at the regression output. When we fit the regression output, we get a quite reasonable r squared, not super high.

And there are some significant predictors in this case. But a lot of them are not significant compared to the standard errors. And as we've looked at previously we know that there is colinearity among the predictors. And so, at least, we want to trim out some of the correlated predictors. We might also be able to get rid of some predictors that aren't useful.

For example, it appears that knee, at least after the other predictors are included, is not at all useful. One way of doing that is, of course, to fit through `glmnet` to fit our penalized regression models. But there's a lot of them. We have a lot of different options, including options both for λ -- and up here I give a very wide range of potential λ values if I can get that easily on the screen here. So I have a very wide range of λ values. And I'm also going to consider several α values for ridge regression, Lasso, and then a couple for elastic net.

So how do we choose among all these different models that we could fit? Well, of course, the answer that we're going to be applying is by using cross-validation on assessment and selecting the model that minimizes the appropriate CV measure. So we'll set up CV groups as we've done previously. And we're going to use CV groups within the `cv.glmnet` function.

So rather than having to write our own loop, we can apply this auto function, automatically included function, which applies cross-validation to these specified groups. I'm going to consider four different α . I'm going to store them in a list as well as make storage space for the best λ and the min CV with four spots corresponding to each of those α s. And we'll see as we progress through the application of `cv.glmnet` how we'll make use of that storage.

There are a couple different options that I'd like to talk about with λ list. The first option is a bit of a denser array. That is it ranges from-- it would range from negative 12 to 12 but with a lot more values in it because we're taking 100-- we're putting a 100 values per unit value on the number-- unit on the number line. So one unit distance is subdivided into 100 different λ values.

We can make that less dense, but I'm going to wind up using the denser. After having worked with both these a little bit more, I chose a denser but also with a lower max value simply because, after some point after a high enough penalty value, we're not getting applicable fits for some of the methods.

It is possible, and perhaps even encouraged, to think about different λ lists depending on whether you're using lasso or whether you're using elastic net. But you

want to make sure that they cover the range of models that could possibly be the best. And so a way of checking that out could be to start with a denser array covering a large range of positive values and then trimming it down as per our discussion.

I'll also note that one of the reasons that I chose the denser array-- and we'll see this when we run our cross-validation for ridge regression-- was that the selected lambda value was quite small and so I wanted to get a little bit more accuracy on that. And so I wanted to make a denser array.

All right. So after that long explanation of selection for lambda list, I will mention that the selection for the alphas was ridge regression, lasso, and then generally picked values that were a little bit closer to the lasso value or one that was closer to the lasso value because I noticed that lasso seemed to be doing better. But I might also, in a bigger picture, want to do a wider array of alphas as well.

And so I'm trying to set this up to make that as convenient as possible, because if I'm just directly applying the command `cv.glmnet` to run the cross-validation for the glmnet model fitting process where I need my x and y as inputs, my lambda, which inputs my lambda list, I have to specify my alpha value, my nfolds, and my cross-validation groups for my fold IDs. When I run that, I can take a look at some of the output. And what you'll see on here-- and I'm going to just double check that I set up the right groups. And double check my plot here.

All right. So what we see here is both the lambda at which the minimization occurs. And you can see perhaps why I wanted to get a little bit of detail out here is that there are a lot of lambda values out here. And it is a very-- there are relatively small differences in terms of the mean square or cv value.

It's worthwhile to also mention that we have a Lambda one standard error away. So we can visualize these standard errors are actually gray bars around the CV value. And we're looking at the value that is one standard error away, or that produces a CV that is one standard error away. And that Lambda value is, well, it's quite a bit of a larger Lambda value, many, many times larger in fact.

The Lambda value is for the specifically best model is almost zero. But we could consider a much relatively larger value of up to 0.67, if we thought that might make a more interpretable model. With retrogression, it doesn't really take out any predictors. So that might not be particularly useful. We're going to stick, for the purposes of this presentation, we're going to stick with the best Lambda being the Lambda at which the minimum CV value occurs.

And so that is the ridge regression model that we'll select. That is we'll use this Lambda when Alpha is equal to zero. And since I said up here we want to keep track of our best Lambda and the CV for that best Lambda, we're going to peel off the minimum Lambda

value from our output, as well as the minimum CV value achieved. And that is visualized on this plot over here. That seems reasonable that this minimum occurs at a CV of about 20.4.

You could also take a look, if you felt like looking at a whole bunch of numbers, at the CV values themselves. As you can see, out to five decimal places we're getting a lot of values that are approximately the same. And that, again, might be another reason not just to pick the one Lambda at which the minimum occurs.

But there are distinctions. And the function stores the Lambda at which the CV value is minimized. And so we're going to in addition too, if we wanted to get a little bit of a closer look at this afterwards, we're going to store our best Lambda. We're going to store the minimum CV in the first location on these lists that we set up. And I'm going to take a closer look at this. Because these CV values are so close, we don't really see any distinction, any real change, in the curve here. We could narrow in even further in terms of the Y scale. But we're going to continue to see a pretty markedly flat line.

All right, so that's what happens with ridge regression. I could do the same thing with Lasso. I could do the same thing with Elastic Net. And all that code is provided there. And I could do the same thing with the Elastic Net with an Alpha of 0.5.

The reason of not going through each of those individually is that the CV.glm net function is applied very similarly. The only change that really is made is changing the Alpha value. We make plots. We store the best Lambda. We store the best CVM. That is the best CV measure.

So I'm going to run those for our Lasso and two Elastic Net models. You see our plots up here for each of those. And if we take a look at the best Lambda, we now have four values. And min CV for each of those Lambdas it looks like the minimum occurs at the Elastic Net with an Alpha of 0.95.

So we would out of these many, many potential models, and I say many potential models, because for each of the four Alphas, we considered hundreds of possible Lambdas for each of these. For each Alpha we picked out the best Lambda for which we got the minimum CV measure out of all Lambdas.

And so we're really comparing thousands of models here across the Lambda and Alpha combinations. Out of all of them, it looks like the CV measure minimum is 20.34283, which is for our Elastic Net with Alpha equals 0.95. All right, so we could compare this across all models. And that is in fact what I just did verbally. Pick out which of those is best, the one best Alpha and the one best Lambda. So let's take a look, one best Alpha and one best Lambda.

Then using that one best Alpha, we're still going to fit across our string of all possible Lambdas and then obtain the coefficients at the one best Lambda. So we'll use our GLM net function, as we did on the previous slide, to fit our pathway of best fits, and then pull off the coefficients at the one best Lambda. And unsurprisingly, knee is removed. BMI is also removed.

And if you took a look back at our correlations, BMI is, of course, highly correlated to weight and height. Because it is computed directly from those. Apparently, also chest is unneeded after other predictors are included.

And so we get our coefficient vector for this best model with one best Alpha and one best Lambda for our penalized regression. One less comment here. We could, again, plot this pathway of our best fit across different Lambdas and vertically identify where those coefficients are pulled from. That is the log Lambda value at which those coefficients are pulled.

The last item that I have here that we're going to be going through together is bootstrapping. I do want to briefly comment that this whole setup here could be summarized a little bit more easily by using a loop. And you can either practice that with this body fat data. Or additionally at the end I have a bonus practice with the trees data set that goes through a looping application through a variety of Alphas.

So you might wish to take a look at that, both for the application back to the trees data for selecting the best model, as well as for how to loop through a wider variety of Alpha values, rather than trying to rewrite the code for each different Alpha.

So once we, for our body fat data, got the best fitting model, we might also wish to estimate standard errors. I'm going to take a look at that here.

Lambda list. I am going to stick with my original Lambda list here. I'm going to define a function that takes in input data where I peel off the first column and the rows that are an index. And the x-variable should be everything but the first column, removing the first column and rows in index. I'll fit GLM net to x-boot and y-boot with one best Alpha and Lambda list and then return out my coefficients.

And just for a refresher, this is very similar to the Beta function for regression that we defined with input data and index, where we took our y-variable, our x predictor matrix, fit a linear model, and then returned the coefficients. So it's very similar, except that we are now using GLM net with the associated Alphas and Betas rather than the LM function.

So we'll go ahead and define that. We'll set a seed for consistency. Then we'll use the boot function from the boot package with y and x in a column together, see bind together, applied with this function in which we're returning the coefficients for 1,000

iterations. So this is going to take a little bit to run. And we'll see warnings similar to what we did before. Because we're including some larger penalties that we would not necessarily need to to reach an optimum.

When it gets to the end, we'll print the output. And again, a bit a reminder, the T_i -th column is the coefficient estimate for the i -th term. So the i -th column of this T is all the coefficient estimates. So if I was so inclined, in addition to printing this all out, I could see that my outputs here are corresponding to the different coefficients.

What could be a little bit confusing is the numbering. Because T_1 is actually for the intercept. T_2 is actually for the first predictor. So just keep that in mind. What we're looking for are the standard errors. And another way of getting that would be to take the standard deviation of the columns of this output T . Now, if we apply that same thing with our regression, and we take a look at the standard errors side by side, I did want to note that the standard error for, in particular, the BMI estimates is far reduced. The estimated standard error for the BMI coefficients is way down when using Elastic Net.

And so this points to the variance reduction feature of penalized regression. Because BMI was highly correlated with a couple of the other predictors. And you can see that their standard errors have also been reduced from multiple linear regression. That covers both model selection and reapplication of bootstrapping for error estimation.

Summary One

Ridge regression uses a penalty/constraint on the sum of the squared coefficients, while LASSO works with the sum of the absolute values of the coefficients. The penalty/constraint for Elastic Net is a weighted average of these two. The penalized form is better for mathematical optimization while the constrained form is more intuitive for understanding.

We can visualize these constraints as restrictions in the coefficient space, and coefficient estimates as the minimal RSS achieved at a particular constraint.



Summary Two

The fitted coefficients values can be plotted across values of the tuning parameter (either s or λ); that is, we visualize pathways of the coefficients as they are “shrunk” toward 0 (as s decreases and λ increases).

While all three penalized regression methods can shrink the coefficients (thus potentially strongly reducing their variability), only LASSO and Elastic Net can perform predictor selection by setting coefficients equal to 0 (due the “shape” of the constraint in the parameter space).



Summary Three


We use a common tool, `glmnet`, to perform the optimization for fitting the coefficient pathways for all three penalized regression models.

We must use more computationally-heavy methods to complete model selection (via cross-validation) as well as estimation of standard errors of the coefficients (via bootstrapping).



Question 1 answer

DS740 - Ridge Regression, LASSO, and Elastic Net with Assessment

 Feedback for Self Assessment

✓ Correct!

When fitting the LASSO model with $\lambda = 0.5$, how many predictors were retained?

Your answer:

2

Correct answer:

2