

REGRESSION III: COMPUTING A GOOD ESTIMATOR WITH REGULARIZATION

-APPLIED MULTIVARIATE ANALYSIS-

Lecturer: Darren Homrighausen, PhD

ANOTHER WAY TO CHOOSE THE MODEL

Let (X_0, Y_0) be a new observation that has the **same properties** as our sample D_n , but is **independent** of it.

Remember (prediction) risk: $\text{pred}(\hat{\beta}) = \mathbb{E}(Y_0 - X_0^\top \hat{\beta})^2$

The idea is to find a $\hat{\beta}$ from the sample D_n and then predict at a new observation to see how good of a job we've done.

We don't have any such new observation

So, we use some approximations to pred instead (AIC, BIC, etc.).

AN INTUITIVE IDEA

What if instead we tried a different idea?

Let's set aside one observation and predict it

Example: Set aside (X_1, Y_1) and fit $\hat{\beta}^{(1)}$ on $(X_2, Y_2), \dots, (X_n, Y_n)$

(The notation $\hat{\beta}^{(1)}$ just symbolizes leaving out the first observation before fitting $\hat{\beta}$)

Now, let's look at the (test) MSE of $\hat{\beta}^{(1)}$

$$\text{MSE}_1 = (Y_1 - X_1^\top \hat{\beta}^{(1)})^2$$

AN INTUITIVE IDEA

As the left off data point is **independent** of the data points used for estimation,

$$\mathbb{E}MSE_1 = \text{pred}(\hat{\beta}_{D_{\text{train}}}) \approx \text{pred}(\hat{\beta}_{D_n})$$

Where

- $D_{\text{train}} = \{(X_2, Y_2), \dots, (X_n, Y_n)\}$
- $D_{\text{test}} = \{(X_1, Y_1)\}$
- $\hat{\beta}_{D_{\text{train}}}$ is $\hat{\beta}$ **trained** only with observations in D_{train}
- $\hat{\beta}_{D_n} = \hat{\beta}$ is the estimator **trained** on all the data

AN INTUITIVE IDEA

Why stop there? We can do the same thing with the second observation as well:

$$\text{MSE}_2 = (Y_2 - X_2^\top \hat{\beta}^{(2)})^2$$

Repeating the notation as for MSE_1

AN INTUITIVE IDEA

$$\mathbb{E}MSE_2 = \text{pred}(\hat{\beta}_{D_{\text{train}}}) \approx \text{pred}(\hat{\beta}_{D_n})$$

Where

- $D_{\text{train}} = \{(X_1, Y_1), (X_3, Y_3) \dots, (X_n, Y_n)\}$
- $D_{\text{test}} = \{(X_2, Y_2)\}$
- $\hat{\beta}_{D_{\text{train}}}$ is $\hat{\beta}$ **trained** only with observations in D_{train}
- $\hat{\beta}_{D_n} = \hat{\beta}$ is the estimator **trained** on all the data

CROSS-VALIDATION

We can use this idea to form an estimate of pred

It is known as (leave-one-out) **cross-validation**¹

$$\text{CV}(\hat{\beta}) = \frac{1}{n} \sum_{i=1}^n \text{MSE}_i = \frac{1}{n} \sum_{i=1}^n (Y_i - \mathbf{x}_i^\top \hat{\beta}^{(i)})^2.$$

Now, we have another risk estimate **minimize CV**

¹We'll get to the leave-one-out part in a moment.



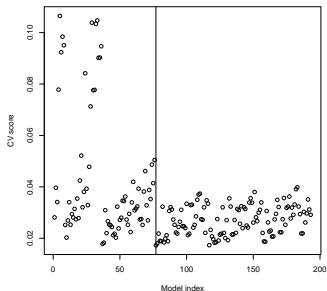
LET'S LOOK AT CV IN ACTION



I wrote two functions, which are in the file **1_CVfunc.r** (check the website) for doing model selection with cross-validation.

Here is an example on the Prostate data:

```
source('1_CVfunc.r')  
validationSets = 1:n  
cv = CVfunc(X,Y,n,validationSets,models)
```





LET'S LOOK AT CV IN ACTION



Results:

```
> models[which.min(cv),]  
      1      2      3      4      5      6      7      8  
TRUE TRUE FALSE TRUE TRUE FALSE FALSE FALSE  
> names(X)[models[which.min(cv),]]  
[1] "lcavol" "lweight" "lbph"    "svi"
```

In this case, we have come up with a model in between all subsets regression and its greedy approximations.

MORE GENERAL CROSS-VALIDATION PROCEDURES

There are two strong disadvantages to cross-validation as we defined it:

$$\text{CV}(\hat{\beta}) = \frac{1}{n} \sum_{i=1}^n (Y_i - \mathbf{x}_i^\top \hat{\beta}^{(i)})^2$$

These are:

- It is computationally demanding (we need to fit n different times).
- It is an **unbiased estimator** of $\text{pred}(\hat{\beta}_{n-1})$
(which means it can be **very high variance**)

K-FOLD CROSS-VALIDATION

A commonly used compromise is to randomly divide your data into K groups.

Let v_1, \dots, v_K correspond to these groups.

For Example: If we have data Z_1, Z_2, Z_3, Z_4, Z_5 , then we can have $K = 2$, and $v_1 = \{2, 5\}$ and $v_2 = \{1, 3, 4\}$

Then, we can form

$$CV_K(\hat{\beta}) = \frac{1}{K} \sum_{k=1}^K \frac{1}{|v_k|} \sum_{i \in v_k} (y_i - X_i^\top \hat{\beta}^{(v_k)})^2.$$

(For CV, $K = ?$ and $v_i = ?$)

K-FOLD CROSS-VALIDATION

A commonly used compromise is to randomly divide your data into K groups.

Let v_1, \dots, v_K correspond to these groups.

For Example: If we have data Z_1, Z_2, Z_3, Z_4, Z_5 , then we can have $K = 2$, and $v_1 = \{2, 5\}$ and $v_2 = \{1, 3, 4\}$

Then, we can form

$$CV_K(\hat{\beta}) = \frac{1}{K} \sum_{k=1}^K \frac{1}{|v_k|} \sum_{i \in v_k} (y_i - X_i^\top \hat{\beta}^{(v_k)})^2.$$

(For CV, $K = n$ and $v_i = \{i\}$)

RETROSPECTIVE SUMMARY

In the previous slides, we took a set of p predictors and reduced them to smaller set of variables using AIC, AICc, BIC, CV,....

There are three main reasons for this:

- **Interpretability:** It is much easier (and convincing) to decide that a few variables are the 'main' contributors to a response variable of interest.
- **Prediction accuracy:** Including larger number of variables reduces bias but increases variance. If we include too many, we can't predict well.
- **Parsimony:** Some variables are unrelated to the response in any meaningful way and we would like to remove them.

Regularization

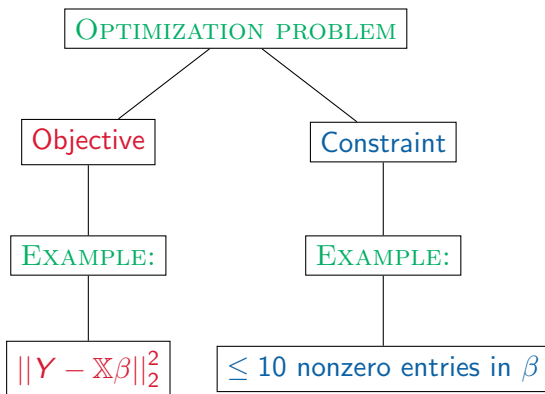
REGULARIZATION

Another way to control bias and variance is through **regularization** or **shrinkage**.

The idea is to make your estimates of β 'smaller', rather than set them to zero

(which is what all subsets does)

SOME OPTIMIZATION TERMS



A (constrained) optimization problem is phrased as

$$\min \ell(\beta) \text{ subject to } C(\beta)$$

where

- $\ell(\beta)$ is the **objective function**
- $C(\beta)$ is the **constraint**

REGULARIZATION

One way to do this for regression is via constraining squared error

$$\hat{\beta}_{\text{ridge},t} = \underset{\|\tilde{\beta}\|_2^2 \leq t}{\operatorname{argmin}} \|Y - \mathbb{X}\tilde{\beta}\|_2^2$$

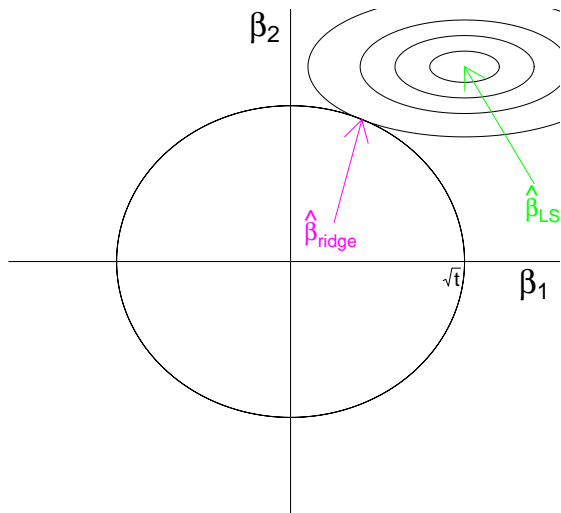
for any $t \geq 0$.

This procedure is called **ridge regression**

Compare this to **least squares**

$$\hat{\beta}_{LS} = \underset{\tilde{\beta} \in \mathbb{R}^p}{\operatorname{argmin}} \|Y - \mathbb{X}\tilde{\beta}\|_2^2$$

GEOMETRY OF RIDGE REGRESSION IN \mathbb{R}^2



RIDGE REGRESSION

An equivalent way to write

$$\hat{\beta}_{\text{ridge},t} = \underset{\|\beta\|_2^2 \leq t}{\operatorname{argmin}} \|Y - \mathbb{X}\beta\|_2^2 \quad (1)$$

is in the **Lagrangian** form

$$\hat{\beta}_{\text{ridge},\lambda} = \underset{\beta}{\operatorname{argmin}} \|Y - \mathbb{X}\beta\|_2^2 + \lambda \|\beta\|_2^2. \quad (2)$$

For every λ' there is a unique t' (and vice versa) that makes

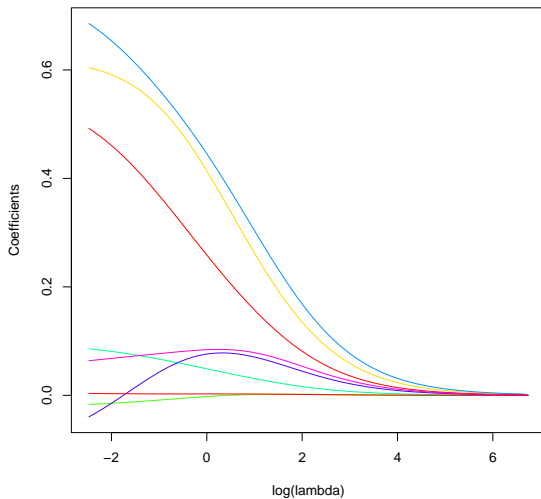
$$\hat{\beta}_{\text{ridge},\lambda'} = \hat{\beta}_{\text{ridge},t'}$$

Observe:

- $\lambda = 0$ (or $t = \infty$) makes $\hat{\beta}_{\text{ridge},\lambda=0} = \hat{\beta}_{LS}$
- Any $\lambda > 0$ (or $t < \infty$) penalizes larger values of β , effectively shrinking them.

Note: λ and t are known as **tuning parameters**

RIDGE REGRESSION PATH



Regularization and Rescaling

LEAST SQUARES IS INVARIANT TO RESCALING

Example: Let's multiply our design matrix by a factor of 10 to get $\tilde{\mathbf{X}} = 10\mathbf{X}$. Then:

$$\tilde{\beta}_{\text{OLS}} = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{Y} = \frac{1}{10} (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} = \frac{\hat{\beta}_{\text{OLS}}}{10}$$

So, multiplying our data by ten just results in our estimates being reduced by one tenth.

Hence, any prediction is left unchanged:

$$\tilde{\mathbf{X}} \tilde{\beta}_{\text{OLS}} = \mathbf{X} \hat{\beta}_{\text{OLS}}$$

This means, for instance, if we have a covariate measured in **miles**, then we will get the “same” answer if we change it to **kilometers**

LEAST SQUARES IS INVARIANT TO RESCALING: EXAMPLE

```
n = 20
set.seed(1)
X = runif(n,0,1)
Y = X*1.5 + rnorm(n,0,.25)
Xtilde  = 2*X
Ytilde  = Y - mean(Y)
```

LEAST SQUARES IS INVARIANT TO RESCALING: EXAMPLE

```
>summary(lm(formula = Y ~ X))
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.1065	0.1265	0.842	0.411
X	1.3341	0.2036	6.554	3.7e-06 ***

```
> summary(lm(formula = Y ~ Xtilde))
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.1065	0.1265	0.842	0.411
Xtilde	0.6671	0.1018	6.554	3.7e-06 ***

```
> summary(lm(formula = Ytilde ~ Xtilde))
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.7407	0.1265	-5.857	1.51e-05 ***
Xtilde	0.6671	0.1018	6.554	3.70e-06 ***

RIDGE REGRESSION IS NOT INVARIANT TO RESCALING

(See next few slides for how to compute the ridge solution via SVD as below)

```
u = X/sqrt(sum(X**2))
d = sqrt(sum(X**2))
lam = 1
betaHat = (d/(d+lam))*u**%Y
> print(betaHat)
      [,1]
[1,] 3.03864
```

```
u = Xtilde/sqrt(sum(X**2))
d = sqrt(sum(Xtilde**2))
lam = 1
betaTilde = (d/(d+lam))*u**%Y
> print(betaTilde)
      [,1]
[1,] 7.004142
> print(betaHat*2)
      [,1]
[1,] 6.07728
```

RIDGE REGRESSION IS NOT INVARIANT TO RESCALING

So, we need to choose a scale before we fit.

The agreed upon scale is to make each column of \mathbb{X} have

- Zero (sample) mean and
- (sample) standard deviation 1.

This can be easily done in R via the 'scale' function:

```
X = scale(X,center=T,scale=T)
```



THE SCALE FUNCTION



NOTE: A nice part of about the **scale** function is that it **keeps** the scalings:

```
X = runif(20,0,1)
X = scale(X,center=T,scale=T)
> attributes(X)
$dim
[1] 20  1

$'scaled:center'
[1] 0.5551671

$'scaled:scale'
[1] 0.2861179

> attributes(X)$'scaled:center'
[1] 0.5551671
```

Ridge regression

RIDGE REGRESSION

RECALL: The least squares solution can be written

$$\hat{\beta}_{\text{LS}} = (\mathbb{X}^{\top} \mathbb{X})^{-1} \mathbb{X}^{\top} \mathbf{Y}$$

It turns out through differential calculus, we can write out the ridge regression solution as well:

$$\hat{\beta}_{\text{ridge}, \lambda} = (\mathbb{X}^{\top} \mathbb{X} + \lambda I)^{-1} \mathbb{X}^{\top} \mathbf{Y}$$

Quite similar!

However, the λ can make all the difference..

REGULARIZATION - RIDGE REGRESSION

Using the **SVD**² ($\mathbb{X} = UDV^\top$), we can look even deeper.

$$\hat{\beta}_{\text{LS}} = (\mathbb{X}^\top \mathbb{X})^{-1} \mathbb{X}^\top Y = VD^{-1}U^\top Y = \sum_{j=1}^p \mathbf{v}_j \left(\frac{1}{d_j} \right) \mathbf{u}_j^\top Y$$

$$\hat{\beta}_{\text{ridge}, \lambda} = (\mathbb{X}^\top \mathbb{X} + \lambda I)^{-1} \mathbb{X}^\top Y = V(D^2 + \lambda I)^{-1} D U^\top Y = \sum_{j=1}^p \mathbf{v}_j \left(\frac{d_j}{d_j^2 + \lambda} \right) \mathbf{u}_j^\top Y$$

RIDGE SHRINKS THE DATA BY AN ADDITIONAL FACTOR OF λ

To see this, note:

$$\begin{aligned} (\mathbb{X}^\top \mathbb{X})^{-1} \mathbb{X}^\top &= (VD \underbrace{U^\top U}_{=I} DV^\top)^{-1} VDU^\top \\ &= (VD^2 V^\top)^{-1} VDU^\top \\ &= VD^{-2} \underbrace{V^\top V}_{=I} DU^\top = VD^{-1} U^\top \end{aligned}$$

²This is **after** centering/scaling

RIDGE REGRESSION: COMPUTATION

There are several ways to compute ridge regression

We can follow any conventional least squares solving technique (i.e.: QR factorization, Cholesky Decomposition, SVD,...):

$$(\mathbb{X}^T \mathbb{X} + \lambda I) \beta = \mathbb{X}^T Y$$

This can be computed via many techniques, for instance the **solve** function in **R**

$$Ax = b \Rightarrow \text{solve}(A, b)$$

RIDGE REGRESSION: COMPUTATION

Alternatively, we can actually solve it using `lm` in `R` if we make the following augmentation

$$\tilde{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{n+p} \text{ and } \tilde{X} = \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix}$$

```
lm(tildeY ~ ., data = Xtilde)
```

(To see this, multiply out $(\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top \tilde{Y}$ and note that it equals $(X^\top X + \lambda I)^{-1} X^\top Y$)

RIDGE REGRESSION: RECAP

IMPORTANT:

- As the constraint set is a sphere, each direction is treated equally. You should standardize your coefficient before fitting.
- Likewise, if included, **don't penalize the intercept**. If the sample means of the covariates are zero, then make the response have mean zero as well (and don't include intercept)

This means either

- ▶ solve

$$\min_{\beta} ||Y - (\beta_0 \mathbf{1} + \mathbb{X}\beta)||_2^2 + \lambda ||\beta||_2^2$$

- ▶ or standardize \mathbb{X} and center Y by \bar{Y}

RIDGE REGRESSION WITH GLMNET

Another way in **R** to do ridge regression is through **glmnet**

```
install.packages('glmnet')
library(glmnet)

Y = prostate$lpsa
X = as.matrix(prostate[,names(prostate)!=c('lpsa','train')])
X.df = prostate[,names(prostate)!=c('lpsa','train')]

ridge.out = glmnet(x=X,y=Y,alpha=0)
```

NOTE: Turning \mathbb{X} into a **matrix** data structure is crucial!

(Note that for **lm**, the opposite is true, \mathbb{X} must be a **data.frame**)

```
> ridge.out = glmnet(x=X.df,y=Y,alpha=0)
Error in elnet(x, is.sparse, ix, jx, y, weights, offset, ...
  (list) object cannot be coerced to type 'double'
```

RIDGE REGRESSION WITH GLMNET

```
> names(ridge.out)
[1] "a0"          "beta"        "df"          "dim"         "lambda"
[6] "dev.ratio"   "nulldev"     "npasses"     "jerr"        "offset"
[11] "call"        "nobs"

> length(ridge.out$lambda)
[1] 100

> ridge.out$lambda[c(1,2,3,4,99,100)]
[1] 843.42743826 768.49966923 700.22827669 638.02192650
[5] 0.09256606 0.08434274
```

RIDGE REGRESSION WITH GLMNET

To get the ridge solution at a particular λ value, say 10, do not do:

```
ridge.out = glmnet(x=X,y=Y,alpha=0,lambda=10)
```

The numerical properties of `glmnet` require running over entire grid.

Instead, use the `coef` or `predict` functions

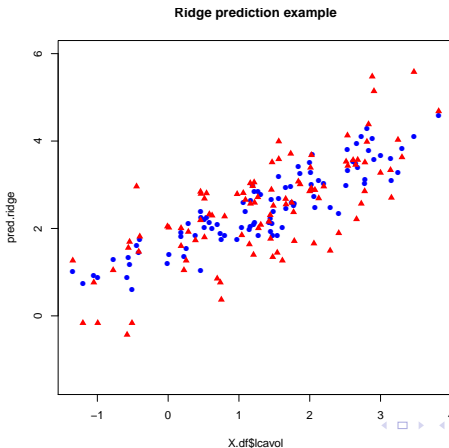
```
> coef(ridge.out,s=10)
9 x 1 sparse Matrix of class "dgCMatrix"

      1
(Intercept) 1.535925340
lcavol      0.064472545
lweight     0.106714247
age         0.001718621
lbph        0.012833216
svi         0.135482215
lcp         0.036522691
gleason     0.044727284
pgg45       0.001325149
```

RIDGE REGRESSION WITH GLMNET

```
pred.ridge = predict(ridge.out,X,s=.01)
```

```
plot(X.df$lcvavol,pred.ridge,ylim=c(-5,5),pch=16,col='blue')  
points(X.df$lcvavol,Y,col='red',pch=17)
```



Choosing λ

RIDGE REGRESSION: PICKING THE TUNING PARAMETER

The crucial choice with regularization is how large to set the tuning parameter λ .

While we can use AIC or BIC for this, conventionally people use cross-validation instead.

Think of CV_K as a function of λ , and pick its **minimum**:

$$\hat{\lambda} = \operatorname{argmin}_{\lambda \geq 0} CV_K(\lambda)$$

and we will use the estimator $\hat{\beta}_{\text{ridge}, \hat{\lambda}}$

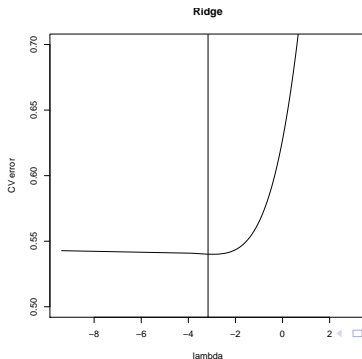
RIDGE REGRESSION: CV

We can do this easily with `glmnet` via the function `cv.glmnet`

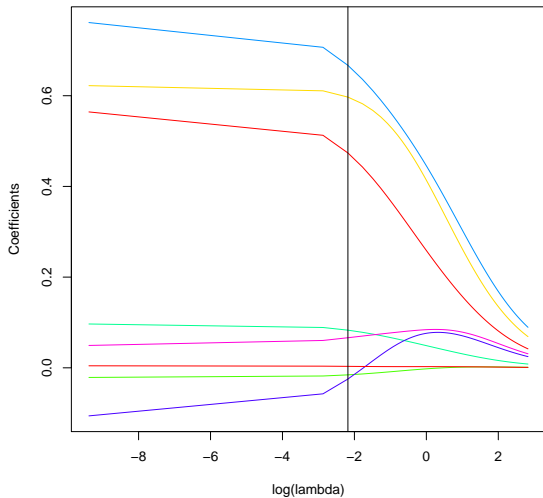
```
X = as.matrix(X)
> ridge.cv = cv.glmnet(x=X,y=Y,alpha=0)
> names(ridge.cv)
[1] "lambda"      "cvm"          "cvstd"        "cvup"
[5] "cvlo"        "nzero"        "name"         "glmnet.fit"
[9] "lambda.min"  "lambda.1se"
> ridge.out = ridge.cv$glmnet.fit
```


RIDGE REGRESSION: CV

```
lambda.hat = ridge.cv$lambda[which.min(ridge.cv$cvm)]  
plot(ridge.cv$lambda,ridge.cv$cvm,  
     xlab='lambda',ylab='CV error',main='Ridge',type='l')  
abline(v=lambda.hat)  
  
> print(log(lambda.hat))  
[1] -2.243919
```



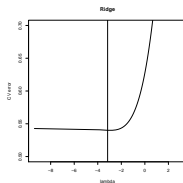
RIDGE REGRESSION: CV



SOME COMMENTS ABOUT GLMNET

Some further details

- Note that in this figure:



many solutions have **almost** the same CV error

In fact, since CV is a pred estimate, it is **random**

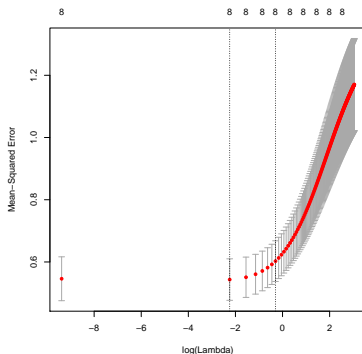
- The lower end point of the grid is somewhat arbitrary chosen

SOME COMMENTS ABOUT GLMNET

The function `cv.glmnet` comes with a `plotting` function

```
ridge.cv = cv.glmnet(x=X,y=Y,alpha=0)
plot(ridge.cv)
```

SOME COMMENTS ABOUT GLMNET



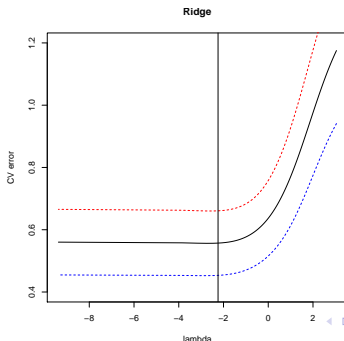
- The left-most dashed, vertical line occurs at the *CV* minimum
- The right-most dashed, vertical line is the
 - ▶ largest value of λ ...
 - ▶ such that the error is within one standard-error of the minimum(the so called one-standard-error rule. We'll see this is more important with a related method soon)

SOME COMMENTS ABOUT GLMNET

Alternatively, you can demand a 'significant' amount of CV reduction to discard full model

(By full model, I mean the model with all covariates)

```
plot(log(ridge.cv$lambda),ridge.cv$cvm,  
      xlab='lambda',ylab='CV error',main='Ridge',  
      type='l',ylim=c(.4,1.2))  
lines(log(ridge.cv$lambda),ridge.cv$cvup,col="red",lty=2)  
lines(log(ridge.cv$lambda),ridge.cv$cvlo,col="blue",lty=2)
```



SOME COMMENTS ABOUT GLMNET

The way that **glmnet** works is to

1. form a **grid** of λ values,
2. find the cross-validation error for each ridge solution on that grid
3. compute the minimum cross-validated λ : $\hat{\lambda}$
4. report $\hat{\beta}_{\text{ridge}, \hat{\lambda}}$ as the final solution

The important piece is that the final solution **depends** on which grid we choose

SOME COMMENTS ABOUT GLMNET

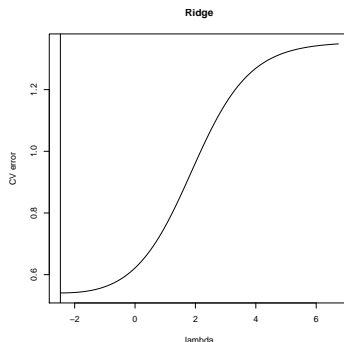
Unfortunately, there is not a good way to define both end points of this grid

(We will see later that for the **lasso** the grid is easier to define, but can have some of the same problems)

IMPORTANT: The minimum value of the grid is chosen arbitrarily

SOME COMMENTS ABOUT GLMNET

Example of a **bad** minimum

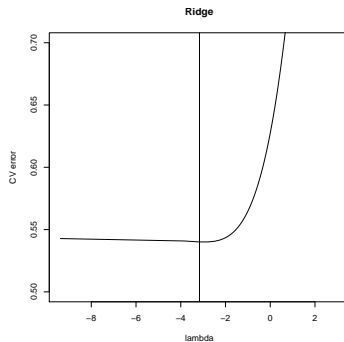


How to fix it:

```
ridge.cv    = cv.glmnet(x=X,y=Y,alpha=0)
min.lambda = min(ridge.cv$lambda)
lambda.new = seq(min.lambda*250,min.lambda*.001,length=1000)
ridge.cv    = cv.glmnet(x=X,y=Y,alpha=0,lambda=lambda.new)
lambda.hat = ridge.cv$lambda[which.min(ridge.cv$cvm)]
```

SOME COMMENTS ABOUT GLMNET

New minimum, after moving λ grid **smaller**:



Multicollinearity

RIDGE REGRESSION AND MULTICOLLINEARITY

Multicollinearity is a phenomenon in which a combination of predictor variables is extremely similar to another predictor variable. Some comments:

- A better term that is sometimes used is \mathbb{X} is ill-conditioned
- It means that one of its columns is nearly (or exactly) a linear combination of other columns. This is sometimes known as '(numerically) rank-deficient'.
- If $\mathbb{X} = UDV^T$ is ill-conditioned, then some elements of D are nearly zero
(remember, D is a diagonal matrix with decreasing entries)
- If we form $\hat{\beta}_{LS} = (\mathbb{X}^T \mathbb{X})^{-1} \mathbb{X}^T Y = VD^{-1}U^T Y$, then we see that the small entries of D are now huge (due to the inverse). This in turn creates a huge variance
($V\hat{\beta}_{LS} = (\mathbb{X}^T \mathbb{X})^{-1} = VD^{-2}V^T$)

RIDGE REGRESSION AND MULTICOLLINEARITY

Ridge Regression fixes this problem by preventing the division by a **near zero number**

Example: If a is a really small number, and $\lambda > 0$ is another number, then

$$\frac{1}{a} \approx \infty \quad \text{while} \quad \frac{1}{a + \lambda} \text{ is much smaller.}$$

To wit, $1/0.0001 = 10,000$, while $1/(0.0001 + 0.1) \approx 10$

Conclusion: $(\mathbf{X}^\top \mathbf{X})^{-1}$ can be really unstable, while $(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1}$ is not.

RIDGE REGRESSION AND MULTICOLLINEARITY: EXAMPLE

Consider the example of predicting blood pressure from a person's weight and body surface area.

```
blood = read.table('../data/bloodpress.txt',header=T)
```

```
Y = blood$BP
```

```
weight = blood$Weight #persons weight
```

```
bsa = blood$BSA
```

```
outBoth = lm(Y~bsa+weight)
```

```
summary(outBoth)
```

```
outBSA = lm(Y~bsa)
```

```
summary(outBSA)
```

```
outWeight = lm(Y~weight)
```

```
summary(outWeight)
```

RIDGE REGRESSION AND MULTICOLLINEARITY: EXAMPLE

```
lm(formula = Y ~ bsa + weight)
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	5.653	9.392	0.602	0.555
bsa	11.663	12.125	0.962	0.350
weight	-4.793	6.232	-0.769	0.452

```
lm(formula = Y ~ bsa)
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	2.7971	8.5284	0.328	0.747
bsa	2.3389	0.1792	13.052	1.29e-10 ***

```
lm(formula = Y ~ weight)
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	2.20531	8.66333	0.255	0.802
weight	1.20093	0.09297	12.917	1.53e-10 ***

RIDGE REGRESSION AND MULTICOLLINEARITY:

EXAMPLE

This code shows that we are estimating $(\hat{\alpha}, \hat{\beta})$ with

```
out = cv.glmnet(x=cbind(weight,bsa),y=Y,alpha=0,nfolds=4,  
               lambda=(1:100)/100)  
out$lambda[which.min(out$cvm)]  
[1] 0.35  
> out$glmnet.fit$beta[,which.min(out$cvm)]  
weight  bsa  
0.574 1.1462
```

$$\hat{\beta}_{\text{ridge},\hat{\lambda}} = (0.574, 1.1462)^{\top}$$

and

```
> out$glmnet.fit$a0[which.min(out$cvm)]  
s65  
6.059
```

$$\hat{\alpha}_{\text{ridge},\hat{\lambda}} = 6.059$$

A comparison

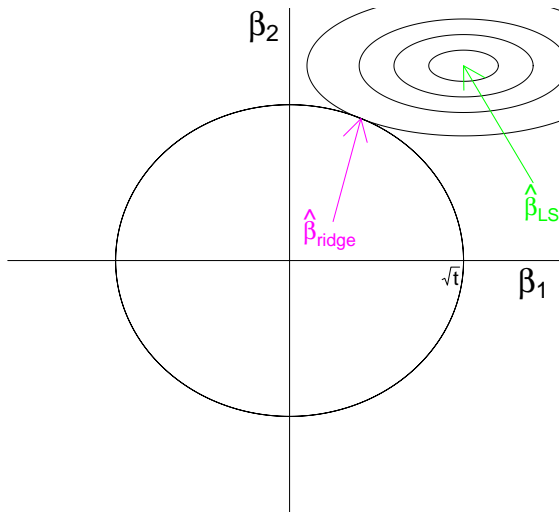
RIDGE REGRESSION: USING THE λ

Here is the chosen fit (along with some previous methods):

```
#for predictor coefficient estimates  
ridge.out$glmnet.fit$beta[,which.min(ridge.out$cvm)]  
#for intercept  
ridge.out$glmnet.fit$a0[which.min(ridge.out$cvm)]
```

Variable	Ridge	Full Linear Model	Forward and Backward
intercept	-0.017	0.181561	0.4947
lcavol	0.474	0.564341	0.543
lweight	0.597	0.622020	0.588
age	-0.015	-0.021248	-0.016
lbph	0.083	0.096713	0.101
svi	0.667	0.761673	0.715
lcp	-0.025	-0.106051	0
gleason	0.066	0.049228	0
pgg45	0.003	0.004458	0

GEOMETRY OF RIDGE REGRESSION IN \mathbb{R}^2



CAN WE GET THE BEST OF BOTH WORLDS?

To recap:

- Forward, backward, and all subsets regression offer good tools for model selection.
(but the optimization problem is nonconvex³)
- Ridge regression provides regularization, which trades off bias and variance and also stabilizes multicollinearity.
(problem is convex, but doesn't do model selection)

RIDGE REGRESSION $\min ||\mathbf{Y} - \mathbf{X}\beta||_2^2$ subject to $||\beta||_2^2 \leq t$

BEST LINEAR REGRESSION MODEL $\min ||\mathbf{Y} - \mathbf{X}\beta||_2^2$ subject to $||\beta||_0 \leq t$
($||\beta||_0$ = the number of nonzero elements in β)

³In fact, it is NP-hard

AN INTUITIVE IDEA

RIDGE REGRESSION $\min ||\mathbf{Y} - \mathbf{X}\beta||_2^2$ subject to $||\beta||_2^2 \leq t$

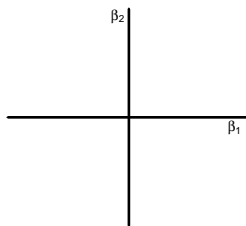
**BEST LINEAR
REGRESSION MODEL** $\min ||\mathbf{Y} - \mathbf{X}\beta||_2^2$ subject to $||\beta||_0 \leq t$

($||\beta||_0$ = the number of nonzero elements in β)

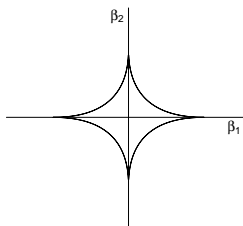
	BEST LINEAR REGRESSION MODEL	RIDGE REGRESSION
Computationally Feasible?	No	Yes
Does Model Selection?	Yes	No

Can we ‘interpolate’ $||\beta||_2$ and $||\beta||_0$ to find a method that does both?

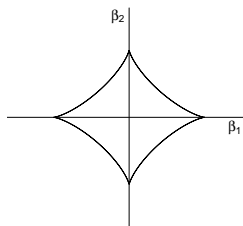
GEOMETRY OF REGULARIZATION IN \mathbb{R}^2 : CONVEXITY



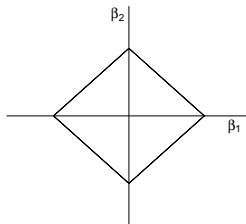
$$\|\beta\|_0 \leq t$$



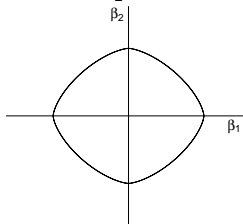
$$\|\beta\|_{\frac{1}{2}} \leq t$$



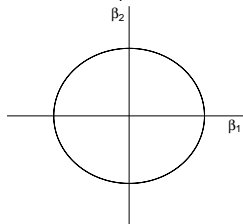
$$\|\beta\|_{\frac{3}{4}} \leq t$$



$$\|\beta\|_1 \leq t$$

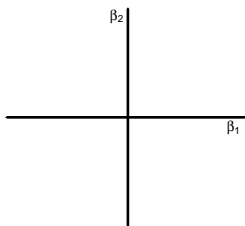


$$\|\beta\|_{\frac{3}{2}} \leq t$$

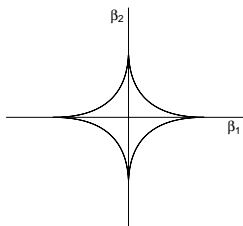


$$\|\beta\|_2 \leq t$$

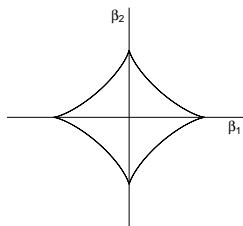
GEOMETRY OF REGULARIZATION IN \mathbb{R}^2 : CONVEXITY



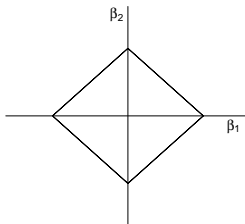
$$\|\beta\|_0 \leq t$$



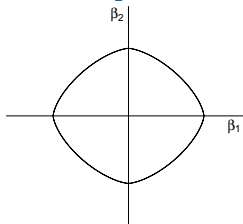
$$\|\beta\|_{\frac{1}{2}} \leq t$$



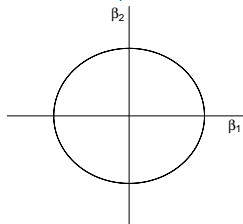
$$\|\beta\|_{\frac{3}{4}} \leq t$$



$$\|\beta\|_1 \leq t$$

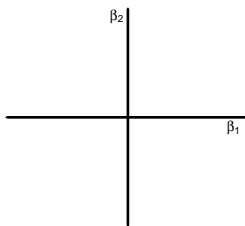


$$\|\beta\|_{\frac{3}{2}} \leq t$$

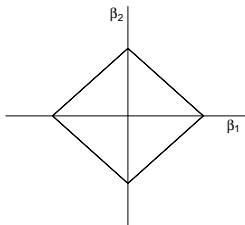


$$\|\beta\|_2 \leq t$$

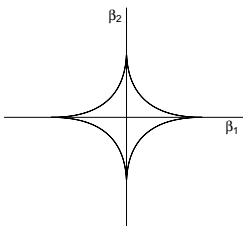
GEOMETRY OF REGULARIZATION IN \mathbb{R}^2 : CONVEXITY



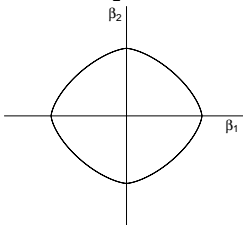
$$\|\beta\|_0 \leq t$$



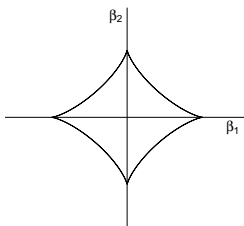
$$\|\beta\|_1 \leq t$$



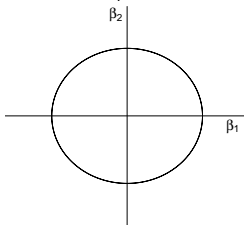
$$\|\beta\|_{\frac{1}{2}} \leq t$$



$$\|\beta\|_{\frac{3}{2}} \leq t$$

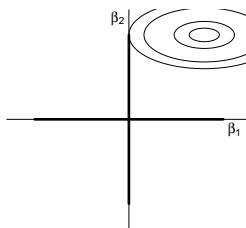


$$\|\beta\|_{\frac{3}{4}} \leq t$$

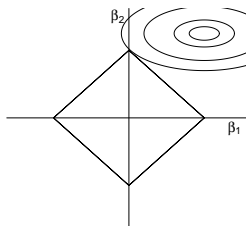


$$\|\beta\|_2 \leq t$$

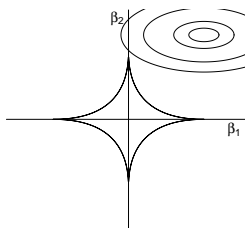
GEOMETRY OF REGULARIZATION IN \mathbb{R}^2 : MODEL SELECTION



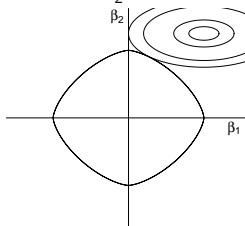
$$\|\beta\|_0 \leq t$$



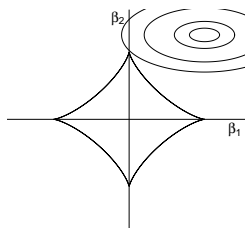
$$\|\beta\|_1 \leq t$$



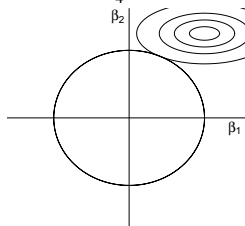
$$\|\beta\|_{\frac{1}{2}} \leq t$$



$$\|\beta\|_{\frac{3}{2}} \leq t$$

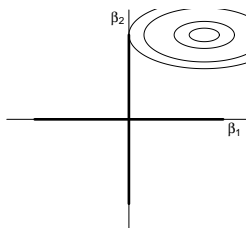


$$\|\beta\|_{\frac{3}{4}} \leq t$$

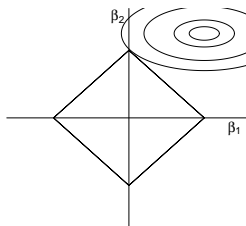


$$\|\beta\|_2 \leq t$$

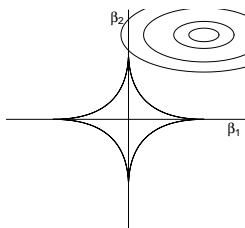
GEOMETRY OF REGULARIZATION IN \mathbb{R}^2 : MODEL SELECTION



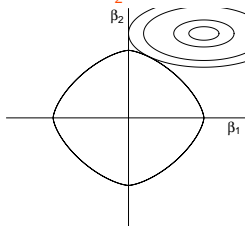
$$\|\beta\|_0 \leq t$$



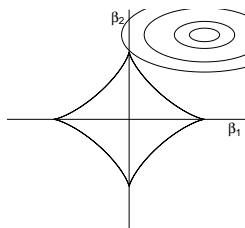
$$\|\beta\|_1 \leq t$$



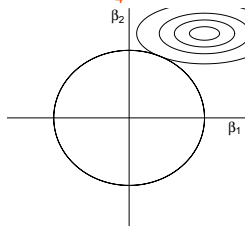
$$\|\beta\|_{\frac{1}{2}} \leq t$$



$$\|\beta\|_{\frac{3}{4}} \leq t$$

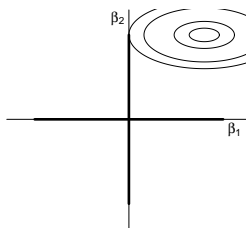


$$\|\beta\|_{\frac{3}{4}} \leq t$$

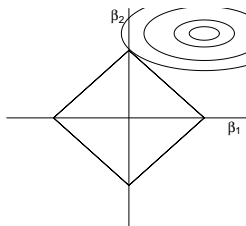


$$\|\beta\|_2 \leq t$$

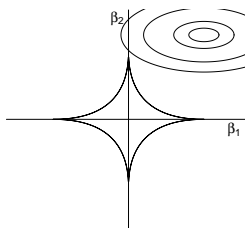
GEOMETRY OF REGULARIZATION IN \mathbb{R}^2 : MODEL SELECTION



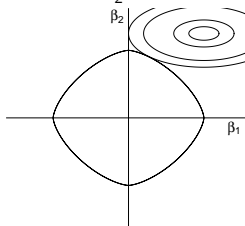
$$\|\beta\|_0 \leq t$$



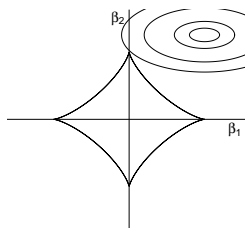
$$\|\beta\|_1 \leq t$$



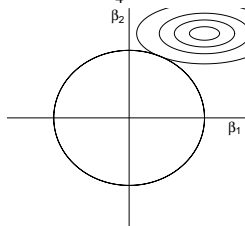
$$\|\beta\|_{\frac{1}{2}} \leq t$$



$$\|\beta\|_{\frac{3}{2}} \leq t$$

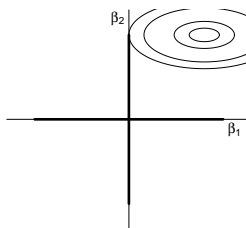


$$\|\beta\|_{\frac{3}{4}} \leq t$$

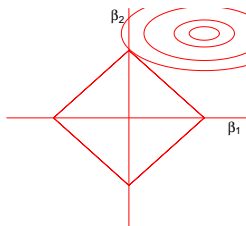


$$\|\beta\|_2 \leq t$$

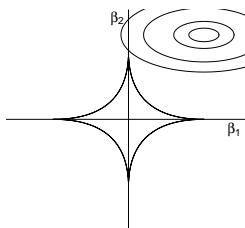
GEOMETRY OF REGULARIZATION IN \mathbb{R}^2 : BOTH



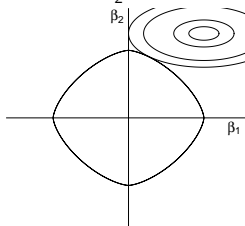
$$\|\beta\|_0 \leq t$$



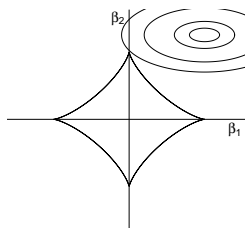
$$\|\beta\|_1 \leq t$$



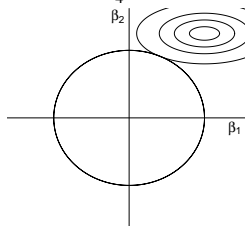
$$\|\beta\|_{\frac{1}{2}} \leq t$$



$$\|\beta\|_{\frac{3}{2}} \leq t$$



$$\|\beta\|_{\frac{3}{4}} \leq t$$

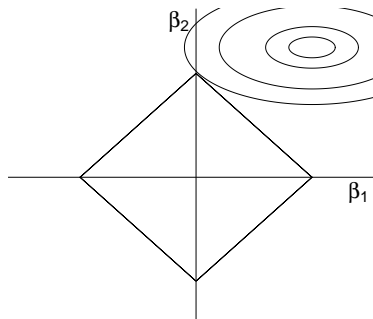


$$\|\beta\|_2 \leq t$$

SUMMARY

	CONVEX?	CORNERS?	
$ \beta _0$	No	Yes	
$ \beta _{\frac{1}{2}}$	No	Yes	
$ \beta _{\frac{3}{4}}$	No	Yes	
$ \beta _1$	Yes	Yes	✓
$ \beta _{\frac{3}{2}}$	Yes	No	
$ \beta _2$	Yes	No	

THE BEST OF BOTH WORLDS: $||\beta||_1$



This regularization set...

- ... is convex (computationally efficient)
- ... has corners (performs model selection)

The lasso

ℓ_1 -REGULARIZED REGRESSION

Known as

- 'lasso'
- 'basis pursuit'

The estimator satisfies

$$\hat{\beta}_{lasso}(t) = \underset{\|\beta\|_1 \leq t}{\operatorname{argmin}} \|\mathbb{Y} - \mathbb{X}\beta\|_2^2$$

In its corresponding Lagrangian dual form:

$$\hat{\beta}_{lasso}(\lambda) = \underset{\beta}{\operatorname{argmin}} \|\mathbb{Y} - \mathbb{X}\beta\|_2^2 + \lambda \|\beta\|_1$$

ℓ_1 -REGULARIZED REGRESSION

While the **ridge** solution can be easily computed

$$\hat{\beta}_{ridge,\lambda} = \underset{\beta}{\operatorname{argmin}} ||Y - X\beta||_2^2 + \lambda ||\beta||_2^2 = (X^T X + \lambda I)^{-1} X^T Y$$

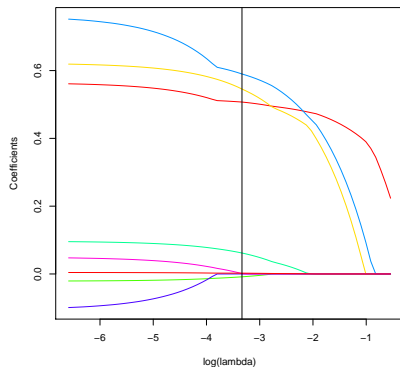
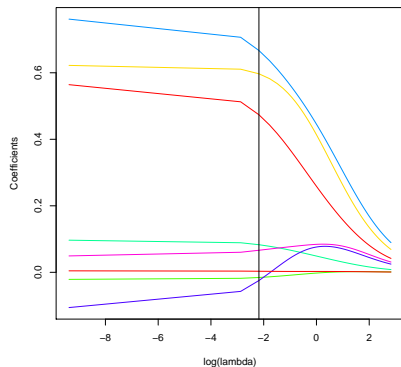
the **lasso** solution

$$\hat{\beta}_{lasso,\lambda} = \underset{\beta}{\operatorname{argmin}} ||Y - X\beta||_2^2 + \lambda ||\beta||_1 = ??$$

doesn't have a **closed form** solution.

However, as the optimization problem is **convex**, there exist efficient algorithms for computing it

COEFFICIENT PATH: RIDGE VS. LASSO



CHOOSING THE TUNING PARAMETER FOR LASSO

There are two main **R** implementations for finding $\hat{\beta}_{lasso,\lambda}$

- Using **glmnet**. Just change the '**alpha =0**' to '**alpha =1**', and you're lassoing:

```
lasso.out = glmnet(x=X,y=Y,alpha=1)
```
- Alternatively, there is the **lars** package

(Technically, we will talk about a third way, called **scaled sparse regression** (SSR). It differs in how the tuning parameter λ is chosen, but still uses the **lars** algorithm for minimizing)

THE LASSO IN R: GLMNET

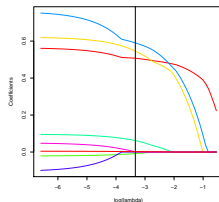
`glmnet` uses `gradient descent` to quickly fit the lasso solution

It can...

- handle other likelihoods than Gaussian
(This will become more important when we talk about `classification` and `generalized` linear models (GLM))
- supports/exploits sparse matrices
(We will cover this soon)
- uses warm restarts for the grid of λ to produce more stable fits/faster computations
(You'll just have to believe me on this one)

THE LASSO IN R: LARS

The **lars** approach is a **path** algorithm



This means we can exploit the fact that the **coefficient profiles** are piecewise linear to:

- not need a grid of λ 's
- make **exact** computations

To do this in **R**, we can do:

```
lasso.lars = lars(X,Y,type='lasso')
```

COMPARISON OF LARS AND GLMNET

There are two main problems with **glmnet**

- The same as with ridge, the λ grid can be poorly chosen



(In practice, the λ interval looks like $[\epsilon ||\mathbb{X}^T Y||_\infty, ||\mathbb{X}^T Y||_\infty]$ for a small ϵ .

Sometimes, this results in finding a boundary solution)

- The **gradient descent** approach is approximate. Sometimes the **thresh** parameter needs to be adjusted to be smaller:

```
fit = glmnet(X,Y,alpha=1,thresh=1e-16)
```

There are three main problems with **lars**

- It is slow(er)
- It doesn't directly support other likelihoods (such as for doing classification)
- It doesn't automatically produce the lasso fit when doing cross-validation

CHOOSING THE TUNING PARAMETER FOR LASSO

Of course, just like in Ridge, we need a way of choosing this tuning parameter.

We can just use **cross-validation** again, though this is still an area of active research:

Homrighausen, D. and McDonald, D.J. *Leave-one-out cross-validation is risk consistent for lasso*, Machine Learning

Homrighausen, D. and McDonald, D.J. *Risk consistency of cross-validation for lasso-type procedures*, Journal of Machine Learning Research

Homrighausen, D. and McDonald, D.J. *The lasso, persistence, and cross-validation*, (2013) International Conference on Machine Learning, JMLR 28(3), 1031–1039.

CHOOSING THE TUNING PARAMETER FOR LASSO

For cross-validation, the heavy lifting has been done for us

```
lasso.cv.glmnet = cv.glmnet(x=as.matrix(X),y=Y,alpha=1)
```

```
lasso.cv.lars   = cv.lars(x=as.matrix(X),y=Y,type='lasso')
```

We can also get the **predictions**:

$$\hat{Y}_0 = X_0^\top \hat{\beta}_{\text{lasso}, \hat{\lambda}}$$

and **coefficient** estimates

$$\hat{\beta}_{\text{lasso}, \hat{\lambda}}$$

with the following code..

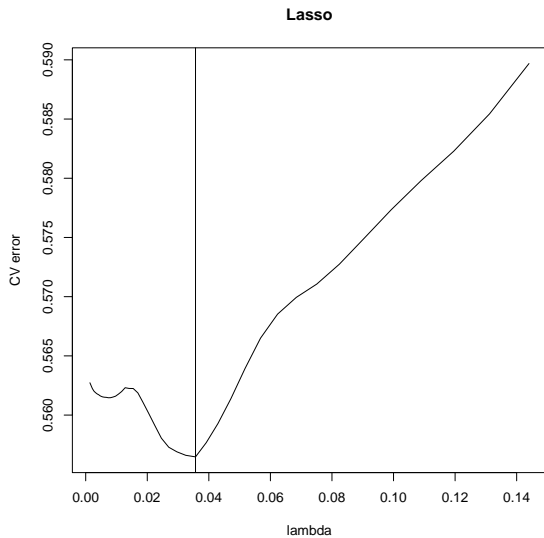
PREDICTIONS AND COEFFICIENTS: GLMNET

```
###  
# glmnet  
###  
lasso.cv.glmnet = cv.glmnet(X,Y,alpha=1,standardize=F)  
lasso.glmnet    = lasso.cv.glmnet$glmnet.fit  
  
Yhat.glmnet     = predict(lasso.cv.glmnet,X_0,s='lambda.min')  
betaHat.glmnet  = coef(lasso.cv.glmnet,s='lambda.min')
```

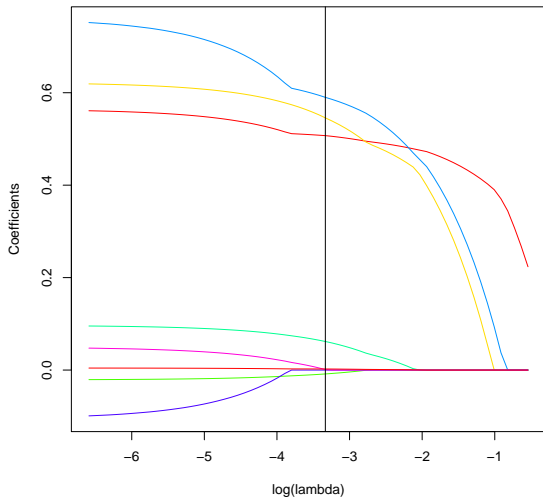
PREDICTIONS AND COEFFICIENTS: LARS

```
###  
# lars  
###  
lasso.lars      = lars(X,Y,type='lasso')  
lasso.cv.lars = cv.lars(X,Y,type='lasso',mode='fraction')  
  
frac.hat  = lasso.cv.lars$index[which.min(lasso.cv.lars$cv)]  
Yhat.lars = predict(lasso.lars,X_0,type='fit',  
                    mode='fraction',s=frac.hat)$fit  
betaHat   = coef(lasso.lars,mode='fraction',s=frac.hat)
```

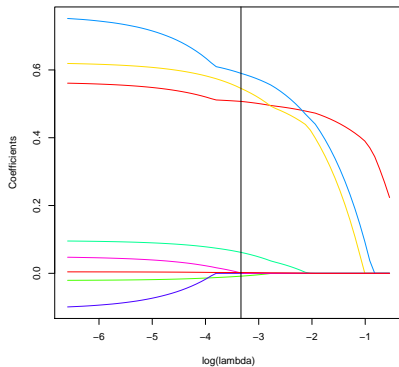
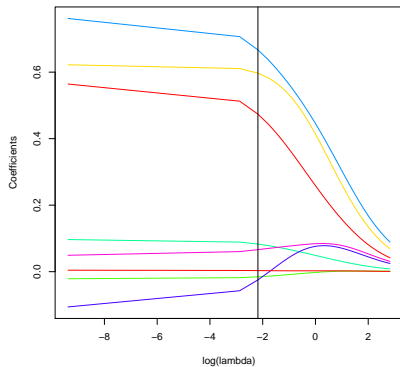
THE LASSO IN R



LASSO REGRESSION PATH



COMPARISON: REGRESSION PATH



Vertical line at minimum CV tuning parameter