

# REGULARIZATION

## -INTRODUCTION TO DATA SCIENCE-

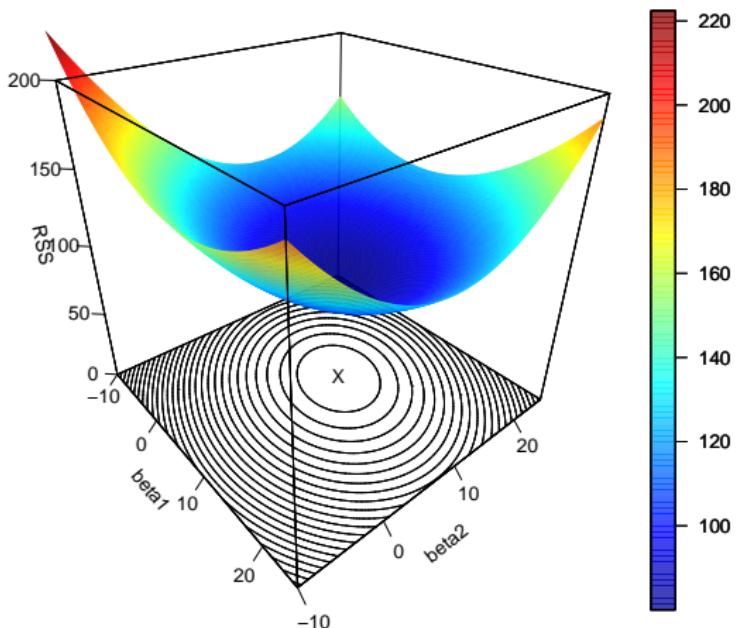
ISL 6.2

Lecturer: Darren Homrighausen, PhD

# Preamble:

- All subsets and its stepwise approximations involve trading off between global & slow and local & fast
- Instead, we approach this problem via **convex relaxation**, which means solving a nearby problem in a global & fast way.
- Different relaxations result in different procedures

# LEAST SQUARES



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

# REGULARIZATION

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

The idea is to make your estimates of  $\beta$  ‘smaller’, rather than set them to zero  
(which is what all subsets does)

One way to do this is called ridge regression<sup>1</sup>:

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

for any  $t \geq 0$ .

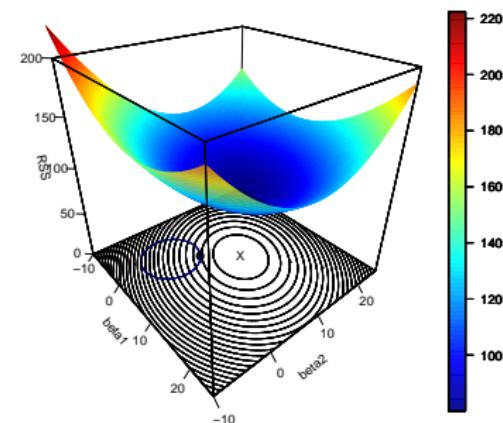
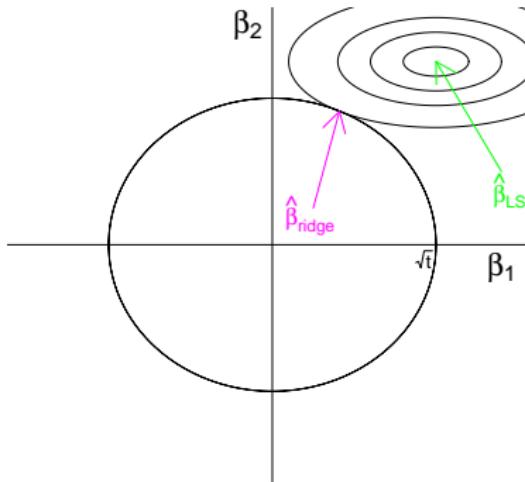
Compare this to least squares

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

---

<sup>1</sup>Hoerl, Kennard (1970)

# 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  $\lambda'$  there is a unique  $t'$  (and vice versa) that makes

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

# REGULARIZATION AND STANDARDIZATION

The coefficient vector isn't invariant to rescaling

If an intercept is included, do not penalize it:

$$\min_{\beta_0, \beta} \sum_{i=1}^n (Y_i - \beta_0 + \beta^\top X_i)^2 + \lambda \|\beta\|_2^2$$

The usual way of addressing this in regression is:

- Standardize all features for which scale is meaningful:

$$x_j \leftarrow \frac{(x_j - \text{mean}(x_j))}{\text{sd}(x_j)}$$

(So, don't standardize indicator variables, for instance)

- Standardize the supervisor  $Y \leftarrow Y - \text{mean}(Y)$
- Don't include an intercept

# RIDGE REGRESSION

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  $\beta$ , effectively shrinking them.

Note:  $\lambda$  and  $t$  are known as **tuning parameters**

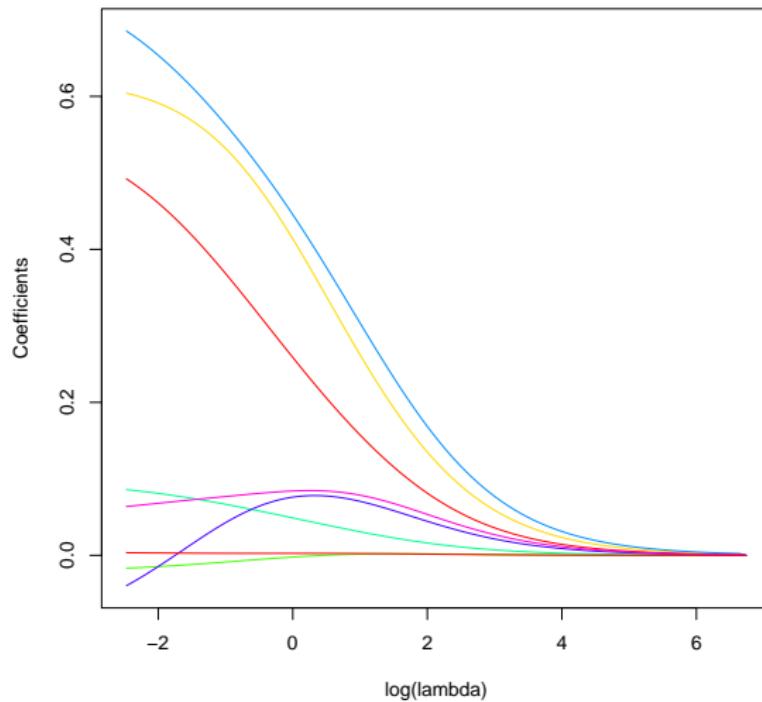
(Alternatively, hyper-parameters)

However we think about it, we have produced a **suite** of solutions

$$\{\hat{\beta}_{\text{ridge}}(\lambda) : \lambda \in [0, \infty)\}$$

What do these solutions look like?

# RIDGE REGRESSION PATH



## RIDGE REGRESSION

**REMINDER:** The least squares solution can be written:

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

However, if  $\text{rank}(\mathbb{X}) < p$ , then  $\hat{\beta}_{\text{LS}}$  is not unique. In fact,

for any  $b$  such that  $\mathbb{X}b = 0$

$\hat{\beta}_{\text{LS}} + b$  is a valid least squares solution.

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

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

Quite similar. However, the  $\lambda$  can make all the difference..

## REGULARIZATION - RIDGE REGRESSION

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

$$\hat{\beta}_{\text{LS}} = VD^{-1}U^\top Y = \sum_{j=1}^p v_j \left( \frac{1}{d_j} \right) u_j^\top Y$$

$$\hat{\beta}_{\text{ridge}}(\lambda) = V(D^2 + \lambda I)^{-1}DU^\top Y = \sum_{j=1}^p v_j \left( \frac{d_j}{d_j^2 + \lambda} \right) u_j^\top Y.$$

Similarly for predictions:

$$\mathbb{X}\hat{\beta}_{\text{LS}} = UU^\top Y = \sum_{j=1}^p u_j u_j^\top Y$$

$$\mathbb{X}\hat{\beta}_{\text{ridge}}(\lambda) = UD(D^2 + \lambda I)^{-1}DU^\top Y = \sum_{j=1}^p u_j \left( \frac{d_j^2}{d_j^2 + \lambda} \right) u_j^\top Y.$$

⇒ Ridge shrinks the data by an additional factor of  $\lambda$ .

## REGULARIZATION - RIDGE REGRESSION

Computationally, we need the ridge solution at many values of  $\lambda$

Naively, that would mean solving the **normal equations** many times

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

However, with the SVD form, we can compute  $\mathbb{X} = UDV^\top$  once and “solve” with only matrix multiplications

$$\hat{\beta}_{\text{ridge}}(\lambda) = \sum_{j=1}^p v_j \left( \frac{d_j}{d_j^2 + \lambda} \right) u_j^\top Y$$

# RIDGE IN A NEW SPACE

Note the matrix identity

$$(A - BC^{-1}E)^{-1}BC^{-1} = A^{-1}B(C - EA^{-1}B)^{-1}$$

Then,

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

This seemingly innocuous change can have large implications

## RIDGE IN A NEW SPACE: COMPUTATIONS

The ridge solution solves either the **normal equations**

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

or the alternative:

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

The ‘heavy lifting’ in each case is done with the inversion

- $\mathbb{X}^\top \mathbb{X} \in \mathbb{R}^{p \times p} \implies$  takes  $p^3$  computations,  $p^2$  space
- $\mathbb{X} \mathbb{X}^\top \in \mathbb{R}^{n \times n} \implies$  takes  $n^3$  computations,  $n^2$  space

**CONCLUSION:** Depending on the relative size of  $n$  and  $p$ , this could be a substantial savings

However, a much deeper realization is possible..

## (KERNEL) RIDGE REGRESSION

Suppose we want to predict at  $X$ , then

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

Also,

$$\mathbb{X}\mathbb{X}^\top = \begin{bmatrix} \langle X_1, X_1 \rangle & \langle X_1, X_2 \rangle & \cdots & \langle X_1, X_n \rangle \\ & \vdots & & \\ \langle X_n, X_1 \rangle & \langle X_n, X_2 \rangle & \cdots & \langle X_n, X_n \rangle \end{bmatrix}$$

and

$$X^\top \mathbb{X}^\top = [\langle X, X_1 \rangle, \langle X, X_2 \rangle, \dots, \langle X, X_n \rangle]$$

where  $\langle X, X' \rangle = X^\top X'$  is the Euclidean inner product.

If we transform  $X_i \mapsto \Phi(X_i)$  then we use  $\langle \Phi(X_i), \Phi(X_{i'}) \rangle$

Inserting  $\Phi$  is known as **kernelization** or a **kernel trick**

## (KERNEL) RIDGE REGRESSION

EXAMPLE: Suppose  $X = (\text{income}, \text{height})^\top$

Then we could specify the map

$$\begin{aligned}\Phi(X)^\top &= (\phi_1(X), \phi_2(X), \phi_3(X), \phi_4(X), \phi_5(X)) \\ &= (\text{income}, \text{height}, \text{income} * \text{height}, \text{income}^2, \text{height}^2)\end{aligned}$$

where, for instance,  $\phi_2(X) = x_2 = \text{height}$

The induced **feature matrix** is then

$$\mathbb{X} = \begin{bmatrix} \Phi(X_1) \\ \vdots \\ \Phi(X_n) \end{bmatrix} \in \mathbb{R}^{n \times 5}$$

## (KERNEL) RIDGE REGRESSION

Ordinarily, this would mean we need to solve the **normal equation** inversion for  $p = 5$

- $\mathbb{X}^\top \mathbb{X} \in \mathbb{R}^{p \times p} \implies$  takes  $p^3$  computations,  $p^2$  space

However, using the **kernel trick** we can solve instead

- $\mathbb{X}\mathbb{X}^\top \in \mathbb{R}^{n \times n} \implies$  takes  $n^3$  computations,  $n^2$  space

which is fixed in  $p$

**IMPLICATION:** We can add essentially arbitrary nonlinearity without paying higher computational, storage cost!

We will return to this again with **support vector machines (SVM)**

# Ridge in practice

## RIDGE REGRESSION: THE TUNING PARAMETER

We can use a degrees of freedom based risk estimator to choose  $\lambda$

The degrees of freedom of  $\hat{\beta}_{\text{ridge}}(\lambda)$  can be seen to be

$$\text{df} = \text{trace} \left[ \mathbb{X} (\mathbb{X}^T \mathbb{X} + \lambda I)^{-1} \mathbb{X}^T \right] = \sum_{j=1}^p \frac{d_j^2}{d_j^2 + \lambda}$$

(As  $\lambda \rightarrow 0$ , we get the number linearly independent columns of  $\mathbb{X}$ )

We can plug this into any of the GIC measures or..

Note that a common choice is **generalized cross-validation (GCV)**, which has the form:

$$\text{GCV}(\hat{\beta}) = \frac{\left\| \mathbb{Y} - \mathbb{X}\hat{\beta} \right\|_2^2}{(1 - \text{df}(\hat{\beta})/n)^2}$$

## RIDGE REGRESSION: THE TUNING PARAMETER

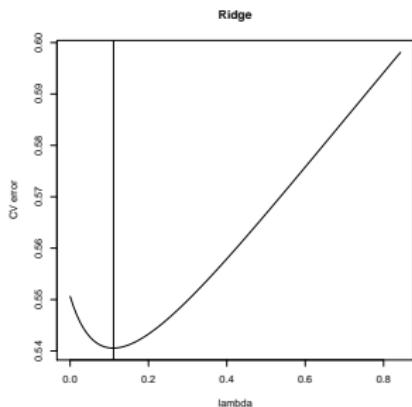
Nowadays, using  $K$ -fold cross-validation is common

Think of  $CV_K$  as a function of  $\lambda$ , and pick its **minimum**:

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

Now, we report  $\hat{\beta}_{\text{ridge}}(\hat{\lambda})$  as our estimator

# RIDGE REGRESSION: CROSS-VALIDATION PLOT



## 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,...):

$$(\mathbb{X}^\top \mathbb{X} + \lambda I) \beta = \mathbb{X}^\top Y \quad (\text{normal equations})$$

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

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

# RIDGE REGRESSION IN R

We will concentrate on a slightly more obtuse way, as it will make things easier later.

```
if(!require(glmnet)){install.packages('glmnet');require(glmnet)}  
ridge.out = cv.glmnet(x=X,y=Y,alpha=0)
```

glmnet automatically scales  $X$ , unless you set

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

## NOTE:

- The function `cv.glmnet` computes  $CV_K$  for a grid of  $\lambda$  values as well as the ridge solutions
- The function `glmnet` just computes the ridge solutions over a grid of  $\lambda$  values
- The feature matrix  $X$  must be a “matrix” data structure

```
> class(X)  
[1] "matrix"
```

# Ridge computations

## RIDGE REGRESSION: COMPARE

```
lam = 1
svd.out = svd(X)
ridge.svd = svd.out$v %*%
            diag(svd.out$d/(svd.out$d**2 + lam)) %*%
            t(svd.out$u) %*% Y

Ytilde = c(Y,rep(0,p))
Xtilde = rbind(X,diag(sqrt(lam),p))
ridge.lm = lm(Ytilde~Xtilde-1)

library(glmnet)
ridge.glmnet = glmnet(x=X,y=Y,alpha=0,
                      standardize=FALSE,intercept=FALSE,
                      lambda = seq(1.1/n,.9/n,length.out = 1000))
```

## RIDGE REGRESSION: COMPARE

```
> print(ridge.svd[1:3])
[1] 0.55869858 0.60999902 -0.01933688

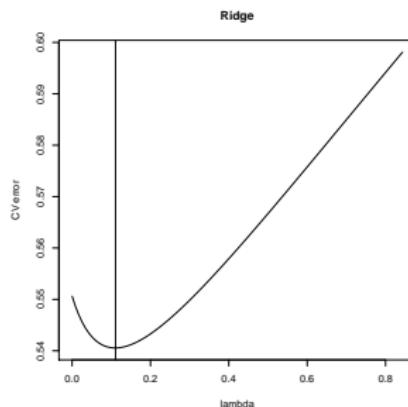
> print(coef(ridge.lm)[1:3])
[1] 0.55869858     0.60999902    -0.01933688

> print(coef(ridge.glmnet,s=lam/n)[1:4])
[1] 0.00000000 0.55911894 0.61313693 -0.01950889
```

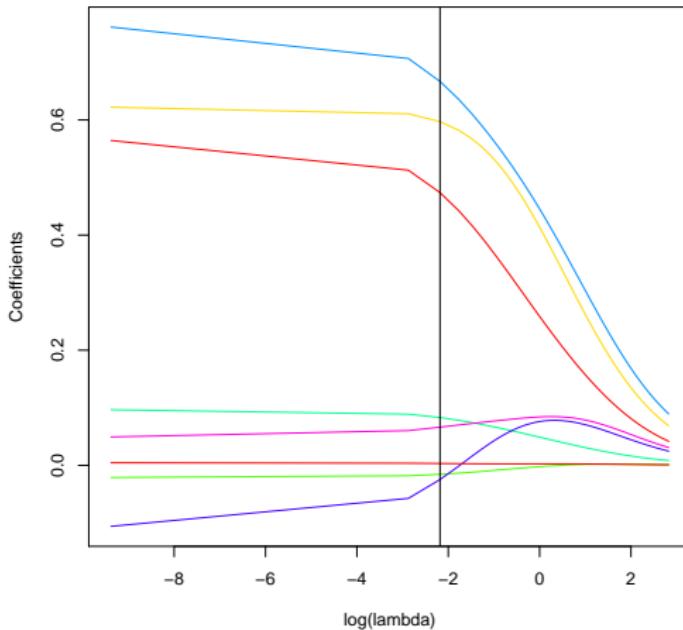
# RIDGE REGRESSION: CV PLOT

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

plot(ridge.out$lambda,ridge.out$cvm,
     xlab='lambda',ylab='CV error',main='Ridge',type='l')
lambdaHat = ridge.out$lambda[which.min(ridge.out$cvm)]
abline(v=lambdaHat)
```



# RIDGE REGRESSION PATH



IMPORTANT: Ridge doesn't set any coefficients to zero for any  $\lambda$



# RIDGE REGRESSION: USING GLMNET

Suppose we have training data  $X$  and  $Y$  loaded into R

Suppose we also have test data  $X_{\text{test}}$  loaded into R

```
out.glmnet = cv.glmnet(x=X,y=Y,alpha=0)
betaHat    = coef(out.glmnet,s='lambda.min')
Yhat       = predict(out.glmnet,type='link',
                     s='lambda.min',newx=Xtest)

#Alternatively
Yhat       = Xtest %*% betaHat
```

DARREN: THIS ONLY WORKS FOR 1 QUAL FEATURE. Do  $X = \text{MODEL.MATRIX}(., XDF)$  INSTEAD

```
> Xdf = data.frame('x1'=rnorm(50),
+                     'x2'=sample(c('yes','no'),replace=TRUE,size=50)
+                     'x3'=sample(c('male','female'),replace=TRUE,si
> head(Xdf)
      x1   x2     x3
1 -1.4209189 yes male
2  0.4667604 no  female
3 -1.7834921 yes male
4 -0.2805537 no  female
5  1.2221916 yes male
6  1.0255659 no  female
> X = model.matrix(~.-1,Xdf)
> head(X)
      x1 x2no x2yes x3male
1 -1.4209189    0     1     1
2  0.4667604    1     0     0
3 -1.7834921    0     1     1
4 -0.2805537    1     0     0
5  1.2221916    0     1     1
6  1.0255659    1     0     0
```

# 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 nonconvex)
- 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       $\min \|\mathbb{Y} - \mathbb{X}\beta\|_2^2 \text{ subject to } \|\beta\|_2^2 \leq t$

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

$(\|\beta\|_0 = \text{the number of nonzero elements in } \beta)$

# AN INTUITIVE IDEA

RIDGE REGRESSION

$$\min \|\mathbb{Y} - \mathbb{X}\beta\|_2^2 \text{ subject to } \|\beta\|_2^2 \leq t$$

BEST LINEAR  
REGRESSION MODEL

$$\min \|\mathbb{Y} - \mathbb{X}\beta\|_2^2 \text{ subject to } \|\beta\|_0 \leq t$$

( $\|\beta\|_0 = \text{the number of nonzero elements in } \beta$ )

Computationally Feasible?  
Does Model Selection?

BEST LINEAR  
REGRESSION MODEL

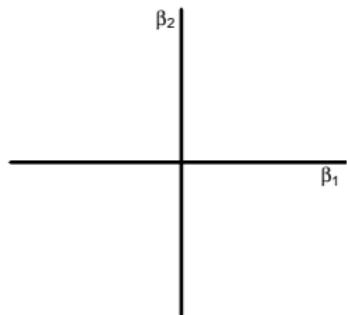
No  
Yes

RIDGE  
REGRESSION

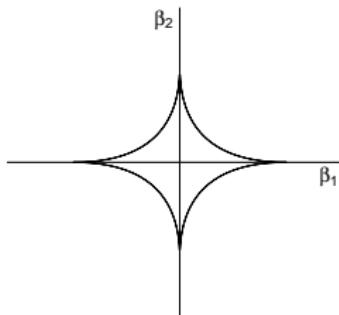
Yes  
No

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

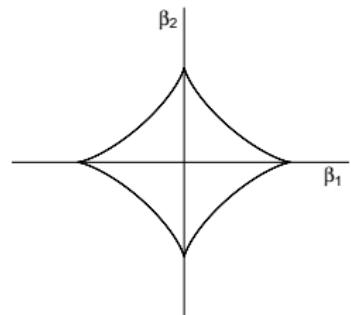
# GEOMETRY OF REGULARIZATION IN $\mathbb{R}^2$ : CONVEXITY



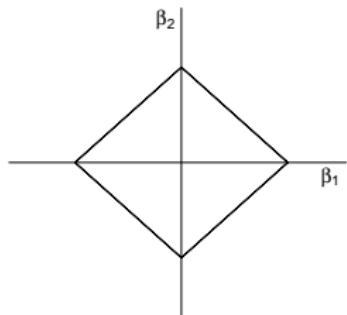
$$||\beta||_0 \leq t$$



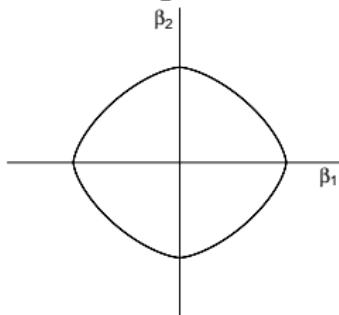
$$||\beta||_1 \leq t$$



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

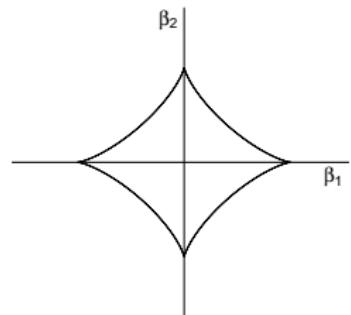
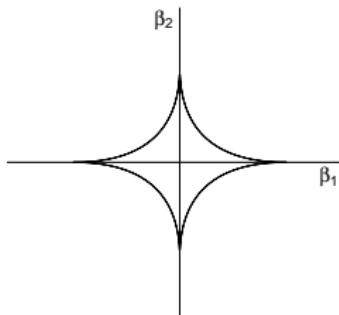
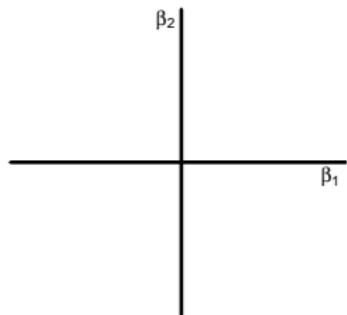


$$||\beta||_1 \leq t$$

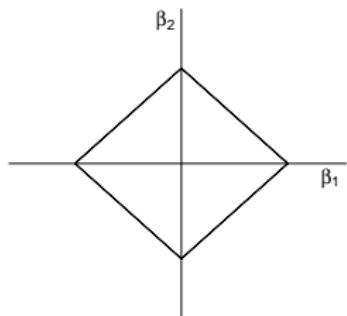


$$||\beta||_3 \leq t$$

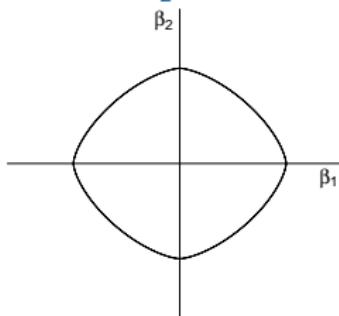
# GEOMETRY OF REGULARIZATION IN $\mathbb{R}^2$ : CONVEXITY



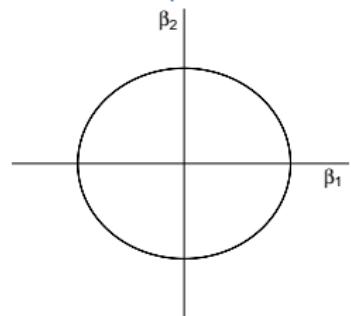
$$\|\beta\|_0 \leq t$$



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



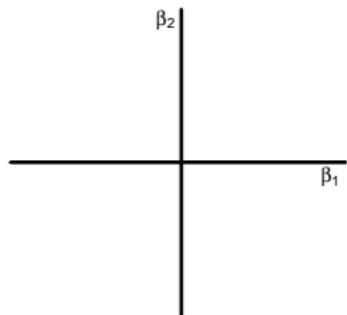
$$\|\beta\|_{\frac{3}{4}} \leq t$$



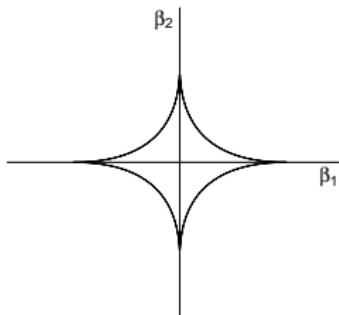
$$\|\beta\|_1 \leq t$$

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

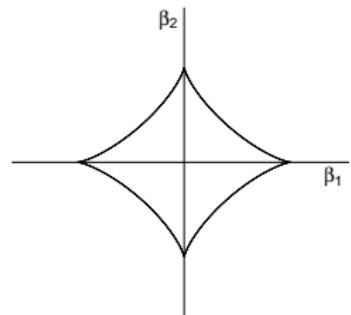
# GEOMETRY OF REGULARIZATION IN $\mathbb{R}^2$ : CONVEXITY



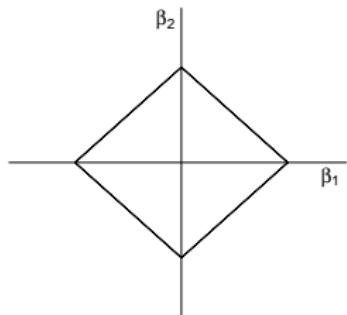
$$\|\beta\|_0 \leq t$$



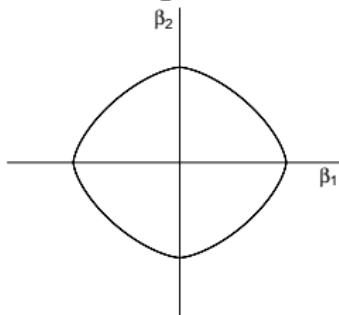
$$\|\beta\|_{\frac{1}{2}} \leq t$$



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

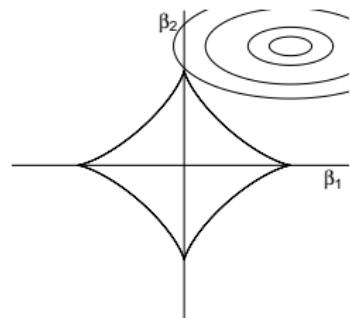
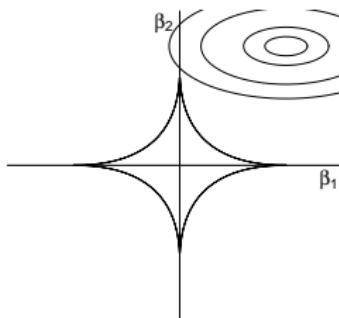
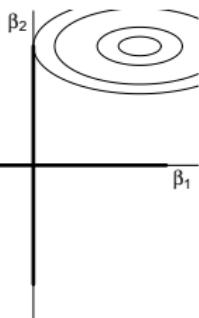


$$\|\beta\|_1 \leq t$$

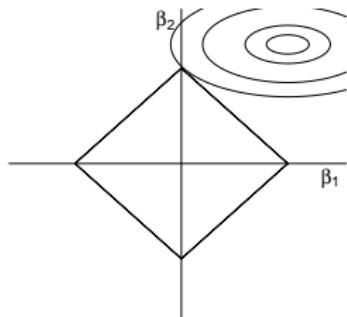


$$\|\beta\|_{\frac{3}{2}} \leq t$$

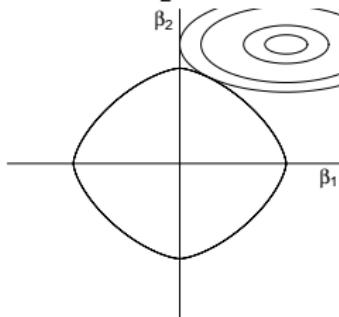
# GEOMETRY OF REGULARIZATION IN $\mathbb{R}^2$ : MODEL SELECTION



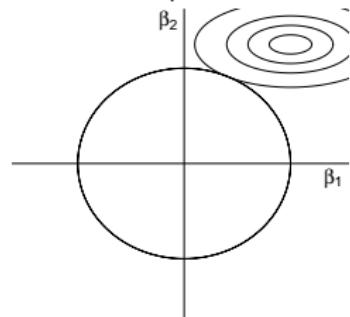
$$\|\beta\|_0 \leq t$$



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



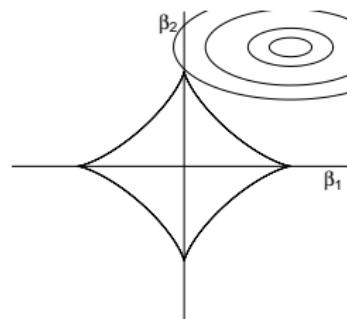
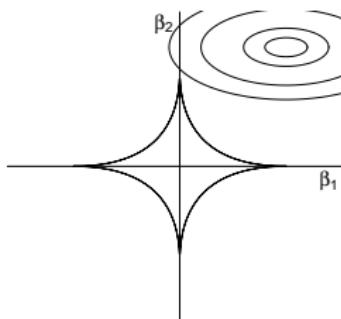
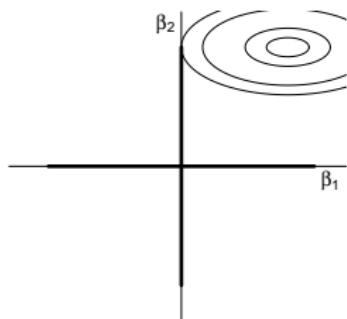
$$\|\beta\|_{\frac{3}{4}} \leq t$$



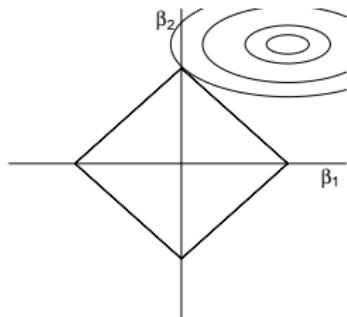
$$\|\beta\|_1 \leq t$$

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

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

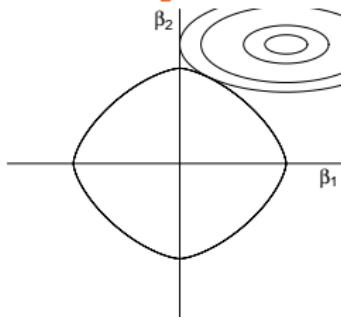


$$\|\beta\|_0 \leq t$$



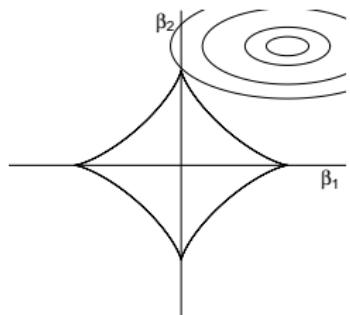
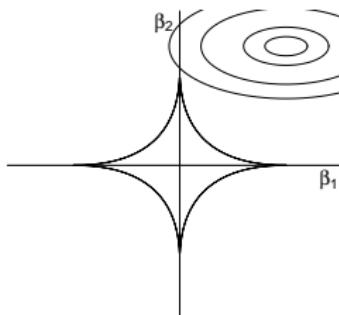
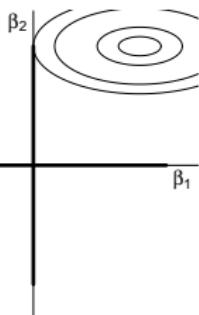
$$\|\beta\|_1 \leq t$$

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

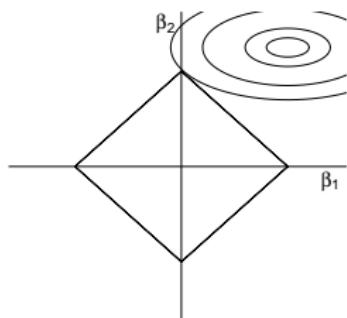


$$\|\beta\|_{\frac{3}{2}} \leq t$$

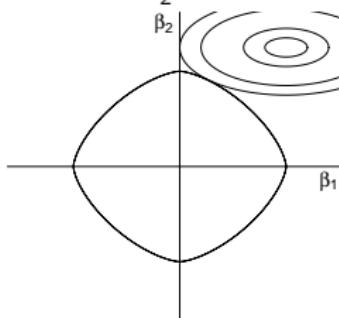
# GEOMETRY OF REGULARIZATION IN $\mathbb{R}^2$ : MODEL SELECTION



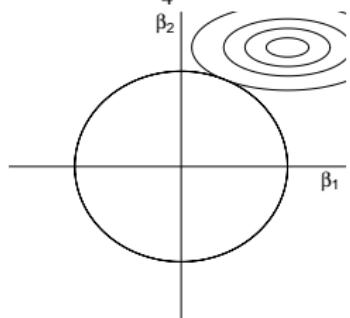
$$\|\beta\|_0 \leq t$$



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



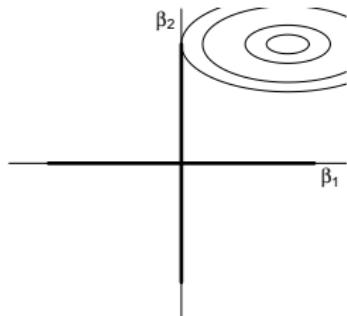
$$\|\beta\|_{\frac{3}{4}} \leq t$$



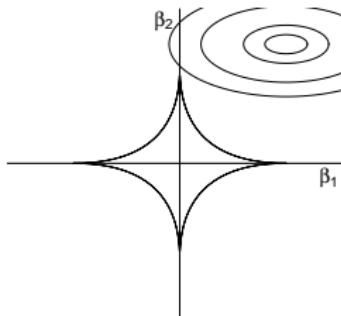
$$\|\beta\|_1 \leq t$$

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

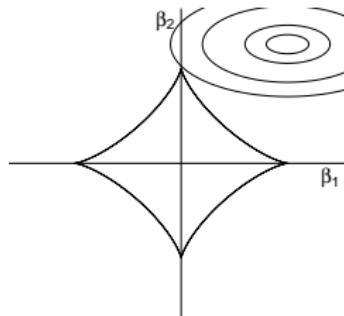
# GEOMETRY OF REGULARIZATION IN $\mathbb{R}^2$ : BOTH



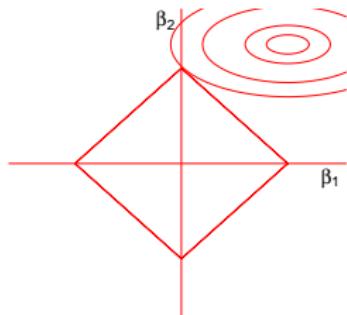
$$\|\beta\|_0 \leq t$$



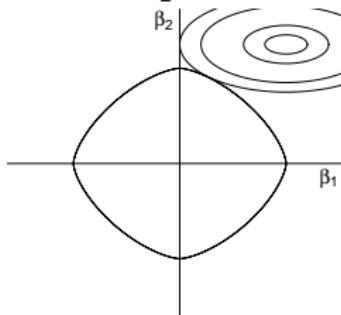
$$\|\beta\|_{\frac{1}{2}} \leq t$$



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



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



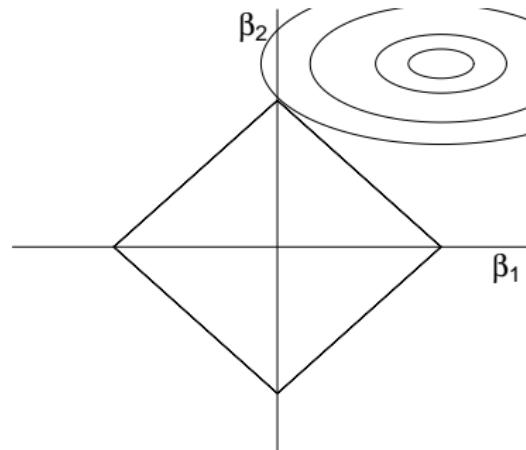
$$\|\beta\|_{\frac{3}{2}} \leq t$$

# 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

# LASSO

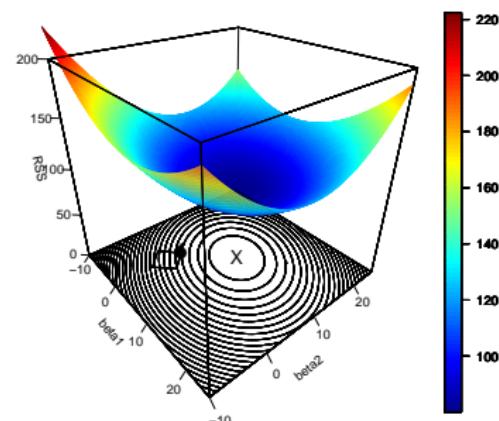
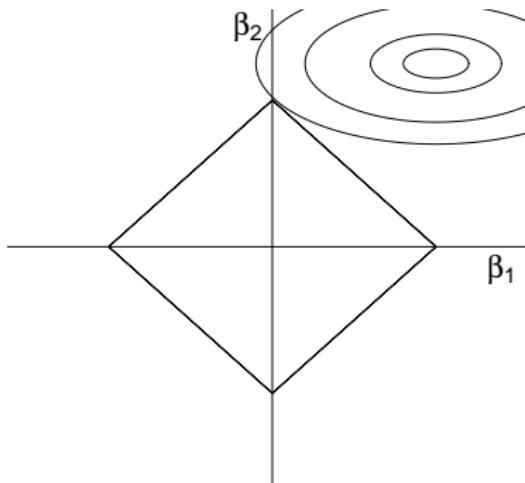
The estimator satisfies

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

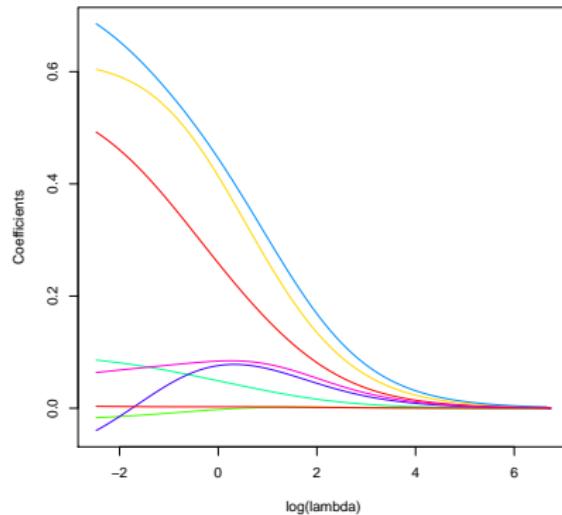
In its corresponding Lagrangian dual form:

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

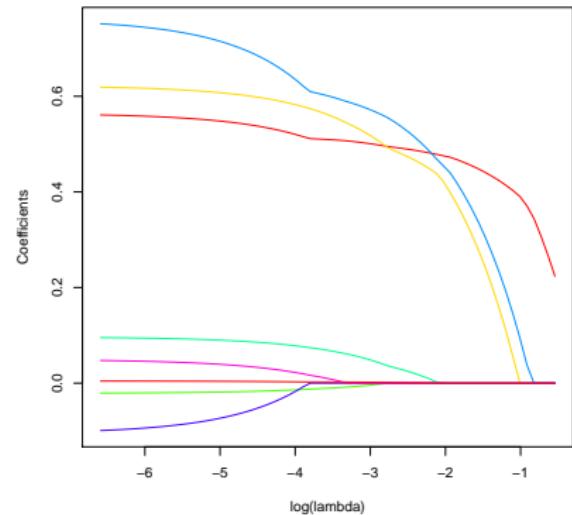
# GEOMETRY OF LASSO REGRESSION IN $\mathbb{R}^2$



# LASSO REGRESSION PATH



Ridge



Lasso

# THE LASSO IN R: GLMNET

Luckily, we already know how to lasso

Just change the '`alpha =0`' to '`alpha =1`', and you're lassoing

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

`glmnet` uses an iterative scheme for finding the lasso solution over a grid of  $\lambda$  values

(Technically, it uses `coordinate descent`, which is related to `gradient descent`)

It can...

- handle other likelihoods than Gaussian
- supports/exploits sparse matrices (e.g. for text processing)
- use warm restarts for the grid of  $\lambda$  to produce more stable fits/faster computations

# OPTIMALITY CONDITIONS: REVIEW

$$\text{minimize } F(x) \tag{3}$$

$$\text{subject to } x \in \mathbb{R}^P \tag{4}$$

Search for  $x_*$  such that  $\nabla F|_{x_*} = 0$

- Turns a geometric problem into an algebraic problem: solve for the point where the gradient vanishes
- Is necessary for optimality of  $x_*$ . Is sufficient if  $F$  is convex and smooth.

# GRADIENT DESCENT: INTUITION

A summary:

1. Start with some initial  $x^0$
2. Propose  $x$  to reduce  $F(x)$
3. Alternate between 1. and 2. until the objective function doesn't change (much).

Algorithmically, the implementations tend to look like

$$x[k + 1] \leftarrow x[k] + \alpha[k]v[k],$$

where

- $x[k]$  is the current value of the minimizing parameter
- $v[k]$  is a direction that (hopefully) reduces  $F$
- $\alpha[k]$  is a relaxation term.

## GRADIENT DESCENT

Define  $\nabla F|_x \in \mathbb{R}^p$  to be the vector of partial derivatives

Recall the general form  $x[k + 1] \leftarrow x[k] + \alpha[k]v[k]$

Gradient descent follows the path of steepest descent

$$x[k + 1] \leftarrow x[k] - \alpha[k]\nabla F|_{x[k]}$$

If  $F$  is a nice enough function, gradient descent is guaranteed to converge to a minimizer

## GRADIENT DESCENT EXAMPLE

If we look at multiple regression via least squares we get:

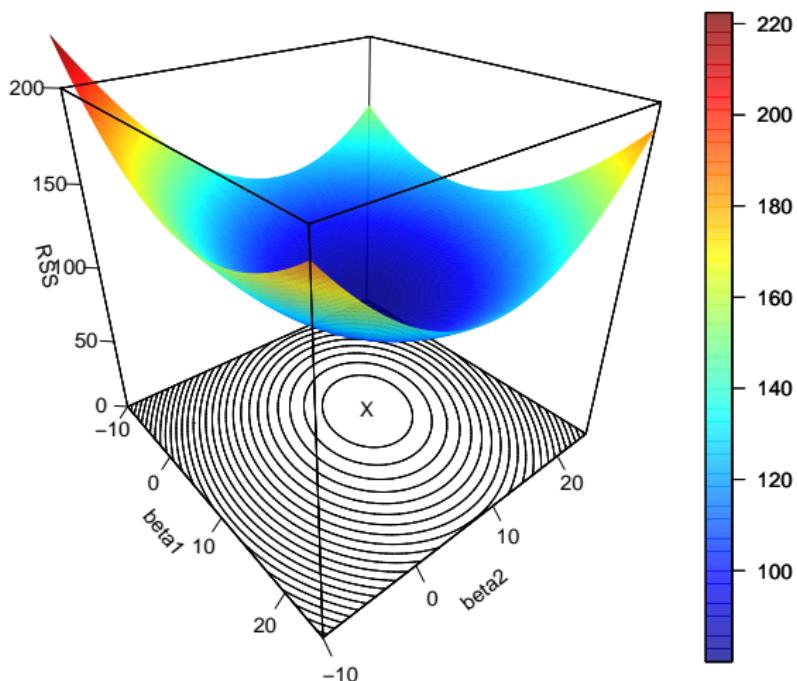
$$x \leftrightarrow \beta \quad F(x) \leftrightarrow \|Y - \mathbb{X}\beta\|_2^2$$

$$\begin{aligned} \min_{\beta} \|Y - \mathbb{X}\beta\|_2^2 &\Rightarrow \frac{\partial}{\partial \beta_j} \|Y - \mathbb{X}\beta\|_2^2 \\ &= \frac{\partial}{\partial \beta_j} \sum_{i=1}^n (Y_i - X_i^\top \beta)^2 \\ &= 2 \sum_{i=1}^n (Y_i - X_i^\top \beta) X_{ij} \end{aligned}$$

For each  $k = 1, 2, 3, \dots$ , we cycle over  $j$  and make the update:

$$\hat{\beta}_j[k+1] = \hat{\beta}_j[k] - \alpha[k] \sum_{i=1}^n (Y_i - X_i^\top \hat{\beta}[k]) X_{ij}$$

# GRADIENT DESCENT EXAMPLE



With  $RSS = \|Y - \mathbb{X}\beta\|_2^2$  for  $p = 2$

# STOCHASTIC GRADIENT DESCENT

The gradient descent update again:

$$\hat{\beta}_j[k+1] = \hat{\beta}_j[k] - \alpha[k] \sum_{i=1}^n (Y_i - X_i^\top \hat{\beta}[k]) X_{ij}$$

(We call this **batch gradient descent**)

Notice that it depends on **all** of the data → expensive to compute

Instead, we can randomly sample  $i$  from  $\{1, 2, \dots, n\}$  and use:

$$\hat{\beta}_j[k+1] = \hat{\beta}_j[k] - \alpha[k] (Y_i - X_i^\top \hat{\beta}[k]) X_{ij}$$

(We call this **stochastic gradient descent**)

## GRADIENT DESCENT: RELAXATION TERM

The gradient descent update again:

$$\hat{\beta}_j[k+1] = \hat{\beta}_j[k] - \alpha[k] \sum_{i=1}^n (Y_i - X_i^\top \hat{\beta}[k]) X_{ij}$$

For either batch or stochastic gradient descent  $\rightarrow$  choose  $\alpha[k]$

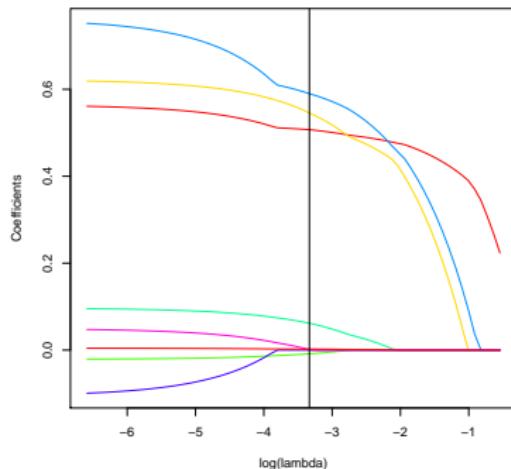
Two common choices:

- Set  $\alpha[k] = \alpha$  to some constant  
(Choose this constant very small)
- Set  $\alpha[k] = \alpha/k$  for a large(ish)  $\alpha$

In both cases, check how fast  $\|\nabla F\|_2$  is decreasing. If too slow, increase  $\alpha$

# THE LASSO IN R: LARS

Alternatively, the `lars` package exploits the fact that the coefficient profiles are piecewise linear, which leads to an algorithm with the same computational cost as the full least-squares fit on the data  
(Efron et al. (2004) for the LARS algorithm)



## CHOOSING THE TUNING PARAMETER FOR LASSO

Note that for the grid  $\lambda$ , we need only look over the interval

$$\left[0, \|\mathbb{X}^\top Y\|_\infty\right) = \left[0, \max_{1 \leq j \leq p} |x_j^\top Y|\right)$$

A grid of  $t$  has a similar restriction  $[0, t_0)$ , where

$$t_0 = \min_{\{b: \mathbb{X}b=0\}} \|\hat{\beta}_{\text{LS}} + b\|_1$$

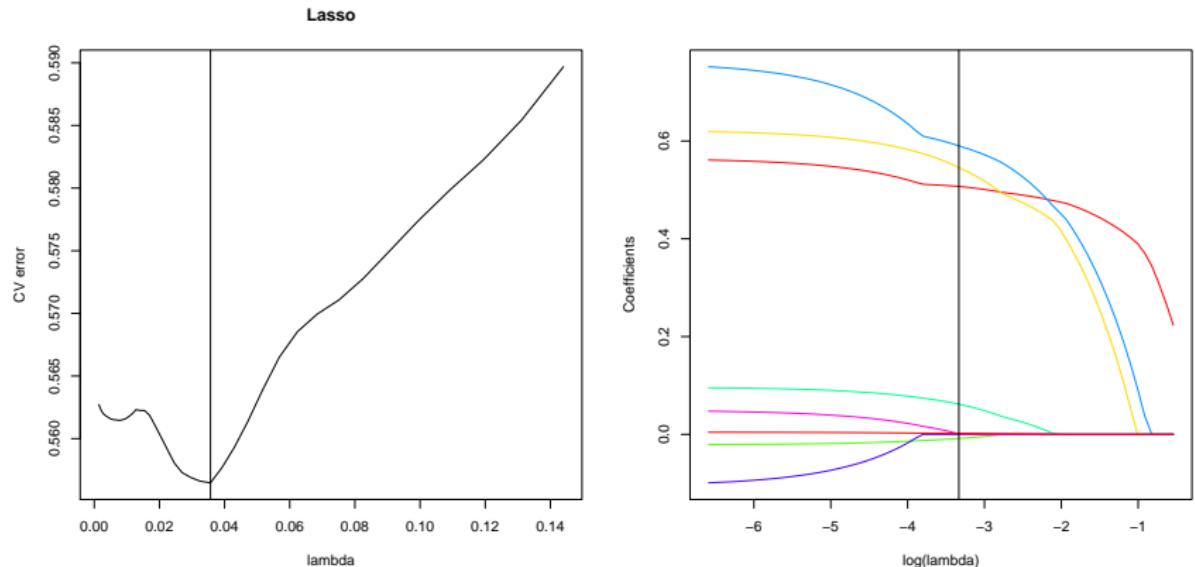
For cross-validation, the heavy lifting has been done for us

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

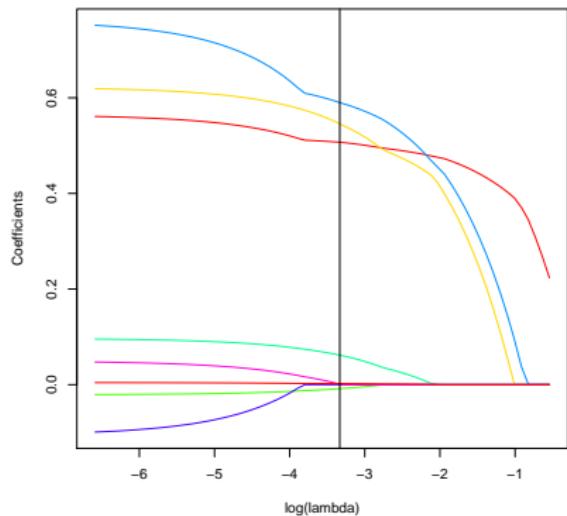
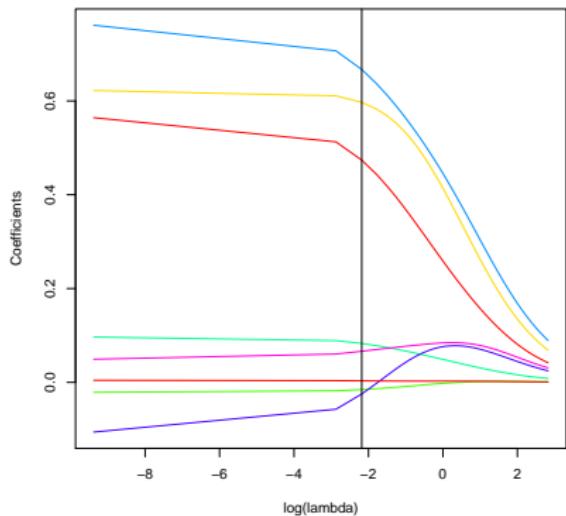
Alternatively, we can use GIC with:

$$\hat{df}(\lambda) = \# \text{ of nonzero entries in } \hat{\beta}_{\text{lasso}}(\lambda)$$

# THE LASSO COEFFICIENT PATH



# COMPARISON: REGRESSION PATH



Vertical line at minimum CV tuning parameter

# COMPARISON OF LARS AND GLMNET

There are two main problems with `glmnet`

- In practice, the  $\lambda$  interval looks like  $[\epsilon \|\mathbb{X}^\top Y\|_\infty, \|\mathbb{X}^\top Y\|_\infty)$  for a small  $\epsilon$ . Sometimes, this results in finding a boundary solution.
- The iterative nature sometimes results in bad coefficient vectors  
(such as having more than  $\min\{n, p\}$  nonzero coefficients, which shouldn't happen)

There are two main problems with `lars`

- It is slow(er)
- It doesn't support other tasks like classification

# LASSO: USING GLMNET

```
> head(X)
      x1 x2no x2yes
1 -0.6264538     1     0
2  0.1836433     0     1
3 -0.8356286     0     1
4  1.5952808     1     0

Y = rnorm(50) + X %*% c(2,1,0)
XtestDf = data.frame('x1'=rnorm(50),
                      'x2'=sample(c('yes','no'),replace=T,size=50))
Xtest = model.matrix(~.-1,XtestDf)
Ytest = rnorm(50) + Xtest %*% c(2,1,0)
```

How we proceed depends on the **scaling** used

## LASSO: USING GLMNET

Have `glmnet` scale the matrix

```
out.glmnet = cv.glmnet(x=X,y=Y,alpha=1,intercept=FALSE)
> out.glmnet$glmnet.fit$beta[,1:5]
3 x 5 sparse Matrix of class "dgCMatrix"
      s0          s1          s2          s3          s4
x1     . 0.17933060 0.3315882 0.4703213 0.5967298
x2no   . 0.04758533 0.1432868 0.2304860 0.3099386
x2yes  . . . . .
> coef(out.glmnet,s='lambda.min')
4 x 1 sparse Matrix of class "dgCMatrix"
           1
(Intercept) .
x1          1.85359875
x2no        1.10466792
x2yes       -0.07347329
Yhat        = predict(out.glmnet,type='link',
                     s='lambda.min',newx=Xtest)
> print(mean((Yhat-Ytest)**2))
[1] 1.303287
```

## LASSO: USING GLMNET

If we don't want `glmnet` to scale the entire matrix

```
qualFeature = c(FALSE,TRUE,TRUE)
scalings     = scale(X)
Xbar         = attributes(scalings)$'scaled:center'
> Xbar
      x1      x2no      x2yes
0.1004483 0.5200000 0.4800000
Xsd   = attributes(scalings)$'scaled:scale'
Xbar[qualFeature] = 0
Xsd[qualFeature] = 1

XScale = scale(X,center=Xbar,scale=Xsd)
> head(XScale)
      x1 x2no x2yes
1 -0.8743173    1    0
2  0.1000669    0    1
3 -1.1259126    0    1
4  1.7979834    1    0
```

## LASSO: USING GLMNET

If we don't want `glmnet` to scale the entire matrix

```
out.glmnet = cv.glmnet(x=XScale,y=Y,alpha=1,
                        intercept=FALSE,standardize=FALSE)
> coef(out.glmnet,s='lambda.min')
4 x 1 sparse Matrix of class "dgCMatrix"
   1
(Intercept) .
x1          1.494099
x2no        1.182825
x2yes       .
```

```
XtestScale = scale(Xtest,center=Xbar,scale=Xsd)
Yhat       = predict(out.glmnet,type='link',
                     s='lambda.min',newx=XtestScale)
> print(mean((Yhat-Ytest)**2))
[1] 1.330784
```

## LASSO: USING LARS

```
> out.lars = lars(X,Y,type='lasso',intercept=FALSE)
> summary(out.lars)#NOTE: It is bad practice to print output!!!!
LARS/LASSO
Call: lars(x = X, y = Y, type = "lasso", intercept = FALSE)
    Df      Rss      Cp
0   0 225.359 247.2196
1   1  55.694  25.4540
2   2  39.777  6.4612
3   3  35.636  3.0000
> out.lars$Cp
      0          1          2          3
247.219562  25.454022  6.461221  3.000000
attr(,"sigma2")
      3
0.7582225
attr(,"n")
[1] 50
```

# Postamble:

- All subsets and its stepwise approximations involve trading off between global & slow and local & fast
- Instead, we approach this problem via **convex relaxation**, which means solving a nearby problem in a global & fast way.  
(This can be seen by relaxing the non-convex constraint set to a convex set)
- Different relaxations result in different procedures  
(Choosing a 'circle' constraint set leads to ridge regression while choosing a 'diamond' constraint set leads to the lasso)