Additional topics on the regularization

-Introduction to Data Science-

(Extra notes)

Lecturer: Darren Homrighausen, PhD

Preamble:

- The automatically generated grid from glmnet needs to be checked
- Sparse matrices provide an efficient way to store data with a lot of zeros
- We can generalize the lasso and ridge regression into the relaxed lasso

Regularization

Reminder Regularization is the process of penalizing training error to reduce model complexity

We have seen a few such approaches so far:

- Model selection: This is

$$\text{training error} + \Gamma||\beta||_0$$

- Ridge: This is

$$\text{training error} + \Psi||\beta||_2^2$$

- Lasso: This is

$$\text{training error} + \Omega||\beta||_1$$

Some additional topics

1. Grids and cross-validation
2. Sparse matrices: In some cases, most of entries in $\mathbb{X}$ are zero and hence we can store/manipulate $\mathbb{X}$ much cheaper using sparse matrices
3. Elastic net: For use when features are highly related to each other
4. Refitted/Relaxed lasso: A proposal for reducing the lasso bias

Grids and cross-validation

Some comments about glmnet and CV

The way that glmnet works is to

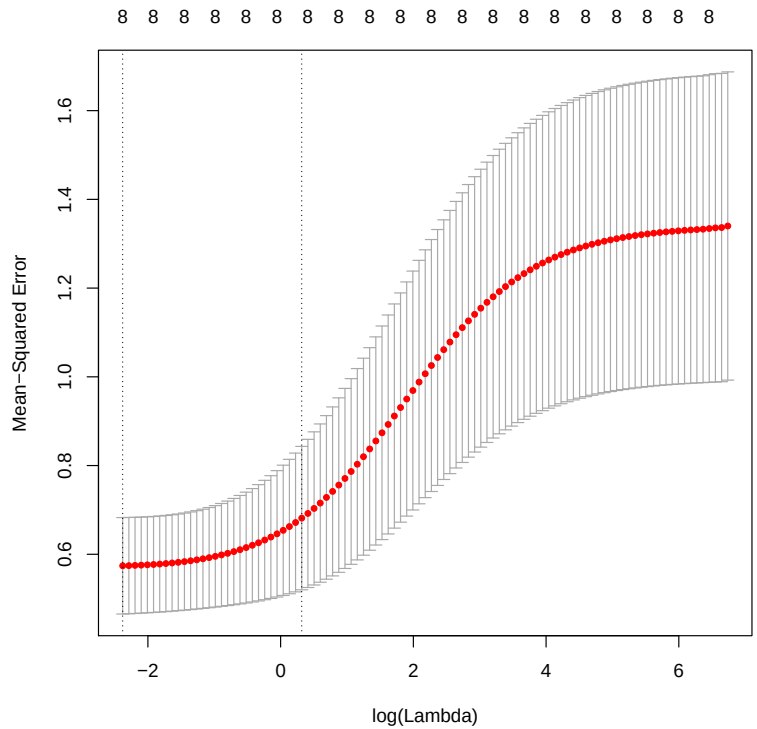
1. form a grid of λ values
2. find the cross-validation error for each solution $\hat{\beta}(\lambda)$ that grid
3. compute the CV minimizing λ: $\hat{\lambda}$
   (This is the 'lambda.min' option in glmnet)
4. report $\hat{\beta}(\hat{\lambda})$ as the solution

Important: the final solution depends on the grid

Some comments about glmnet and CV

The function cv.glmnet comes with a plotting function

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



- The left-most dotted, vertical line occurs at the CV minimum
- The right-most dotted, vertical line is the
  - largest value of λ ...
  - such that $CV(\lambda)$ is within one standard-error of $CV(\hat{\lambda})$

  (the so called one-standard-error rule)
- Notice the '8's (hazards of open source software..)

The one-standard-error rule

The CV estimator of the risk is designed to find procedures that have good prediction performance

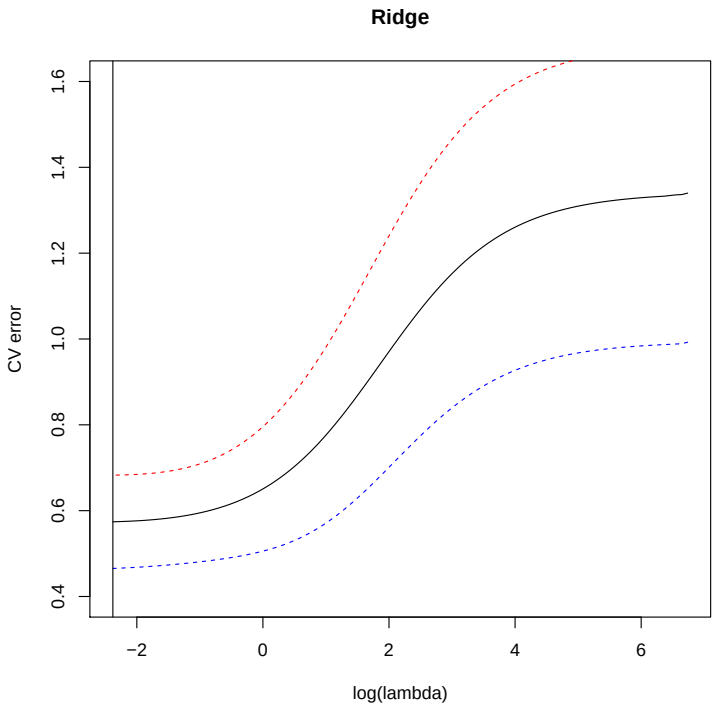
Sometimes, this isn't the main goal

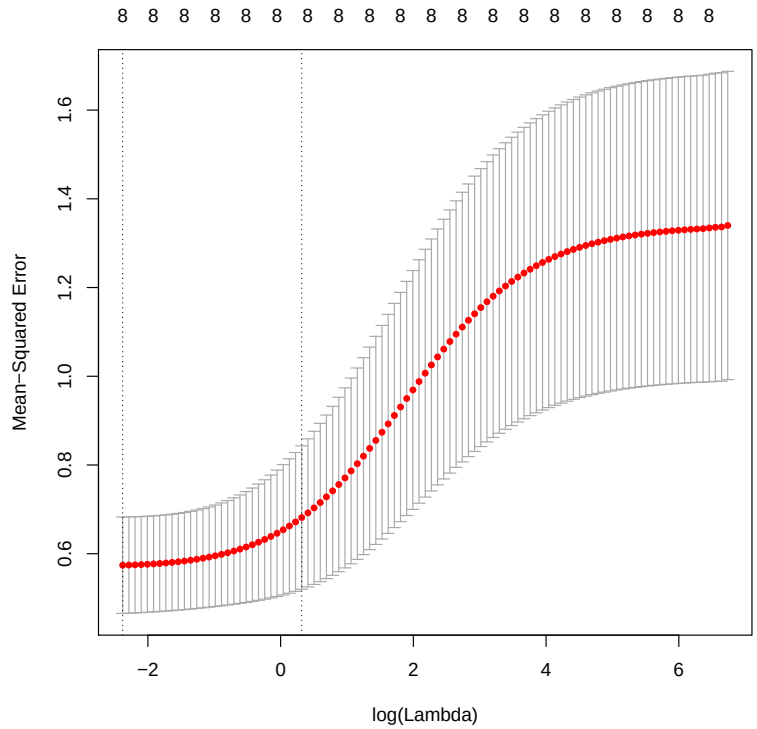
If we want to find the active set $\mathcal{S}$ of non-zero coefficient, using $\hat{\lambda}$ (that is, 'lambda.min') tends to under-regularize

Choosing the one-standard-error rule (that is, 'lambda.1se') adds additional regularization, which partially corrects this under-regularization

Some comments about glmnet and CV

We have made our own version of the same plot:






```
plot(log(ridge.cv$lambda),ridge.cv$cvm,
     xlab='log(lambda)',ylab='CV error',main='Ridge',
     type='l',,ylim=c(.4,1.6))
lines(log(ridge.cv$lambda),ridge.cv$cvup,col="red",lty=2)
lines(log(ridge.cv$lambda),ridge.cv$cvlo,col="blue",lty=2)
```

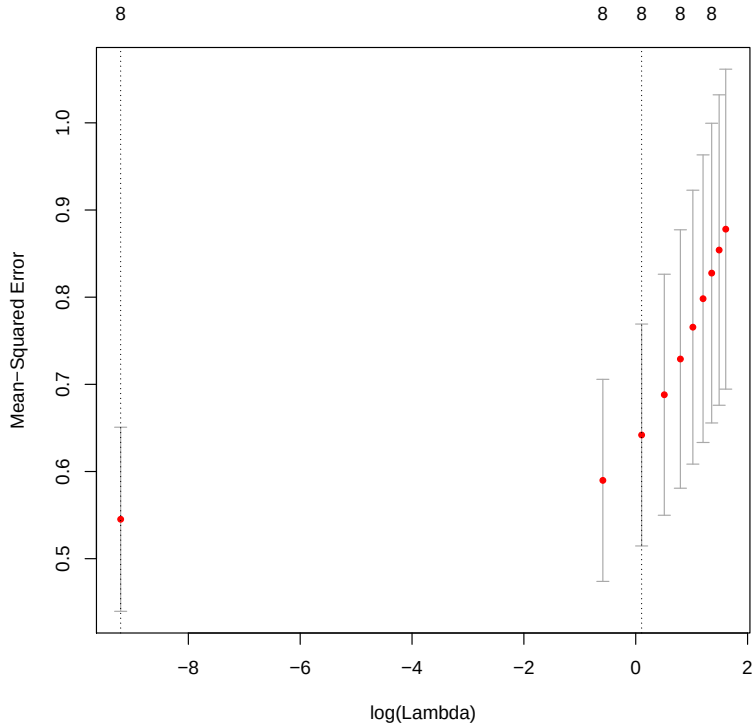
Some comments about glmnet and CV

Though glmnet automatically allocates a grid, it isn't necessarily any good

Sometimes...

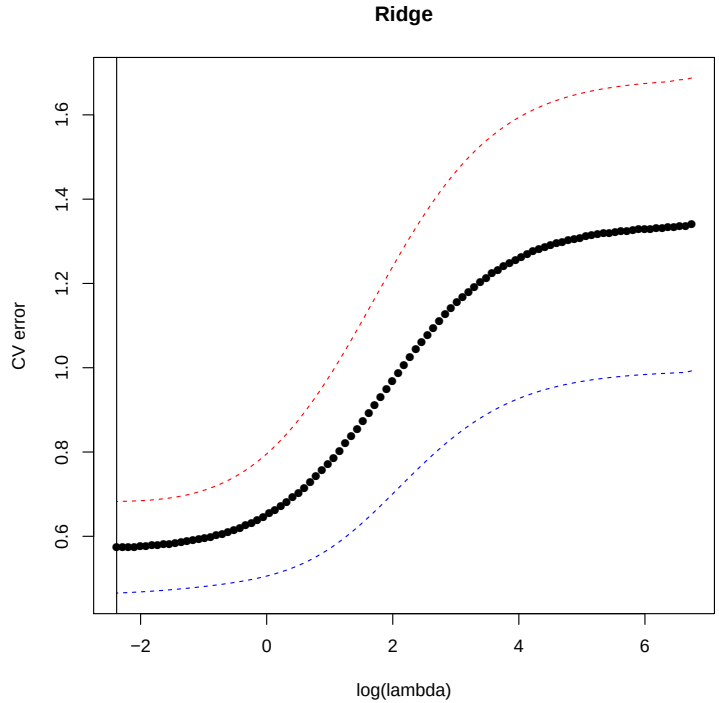
- the grid values are too far apart near the minimum
- the grid doesn't allow small/large enough λ values

Some comments about glmnet and CV



Example of a bad minimum: Grid values too far apart

Some comments about glmnet and CV

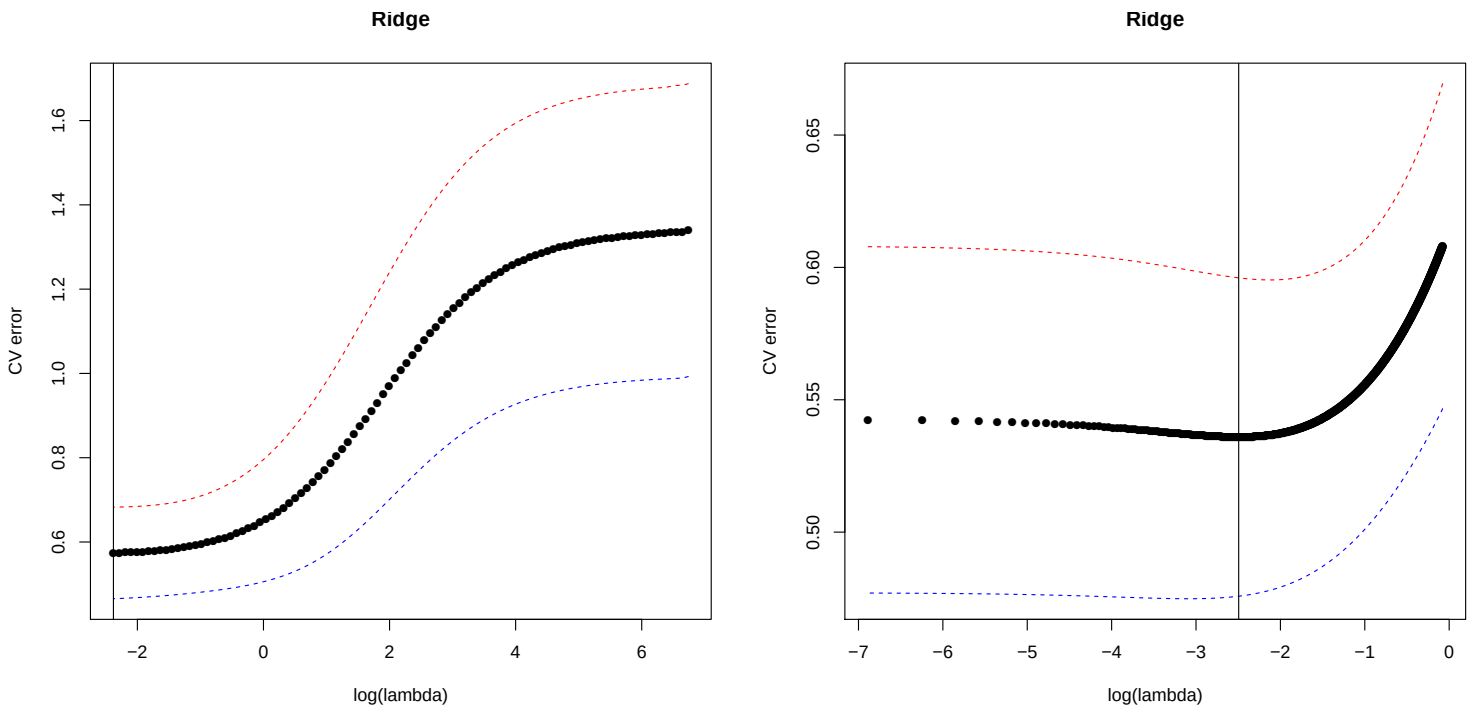


Example of a bad minimum: Grid values too large

Some comments about glmnet and CV

How to fix it:

```
ridge.cv    = cv.glmnet(x=X,y=Y,alpha=0)
min.lambda = min(ridge.cv$lambda)
lambda.new = seq(min.lambda*10,min.lambda*.001,length=1000)
ridge.cv    = cv.glmnet(x=X,y=Y,alpha=0,lambda=lambda.new)
lambda.hat = ridge.cv$lambda[which.min(ridge.cv$cvm)]
```



New minimum, after moving λ grid smaller

Sparse matrices

Sparse matrices

```
load("../data/hiv.rda")
X = hiv.train$x
> X[5:12,1:10]
     p1 p2 p3 p4 p5 p6 p7 p8 p9 p10
[1,]  0  0  0  0  0  0  0  0  0   0
[2,]  0  0  0  0  0  0  0  0  0   0
[3,]  0  0  0  0  0  0  0  0  0   0
[4,]  0  0  0  0  0  0  0  0  0   0
[5,]  0  0  0  0  0  0  0  1  0   0
[6,]  0  0  0  0  0  0  0  0  0   0
[7,]  1  0  0  0  0  0  0  0  0   0
[8,]  0  0  0  0  0  0  0  0  0   0
```

Many zero entries!

Sparse matrices

All numbers in R take up the same space

(Space in this context means RAM aka memory)

```
> print(object.size(0),units='auto')
48 bytes
> print(object.size(pi),units='auto')
48 bytes
```

Idea: If we can tell R in advance which entries are zero ...

- it doesn't need to save those numbers
- nor multiply/add with them

Sparse matrices

This can be accomplished in several ways in R

I usually use the Matrix package

```
library('Matrix')

Xspar = Matrix(X,sparse=T)
```

Sparse matrices

Let's take a look at the space difference

```
> print(object.size(X),units='auto')
1.1 Mb
> print(object.size(Xspar),units='auto')
140.7 Kb
```

Pretty substantial! Only 12.1% as large

Sparse matrices

Lastly, we can create sparse matrices without having the original matrix $\mathbb{X}$ ever in memory

This is usually done with three vectors of the same length:

- A vector with row numbers
- A vector with column numbers
- A vector with the entry value

```
i = c(1,2,2)
j = c(2,2,3)
val = c(pi,1.01,100)

sparseMat = sparseMatrix(i = i, j = j, x = val,dims=c(4,4))

regularMat = as(sparseMat,'dgeMatrix')
```

Sparse matrices

```
> print(sparseMat)
4 x 4 sparse Matrix of class "dgCMatrix"

[1,] . 3.141593   . .
[2,] . 1.010000 100 .
[3,] . .          . .
[4,] . .          . .
> print(regularMat)
4 x 4 Matrix of class "dgeMatrix"
     [,1]     [,2] [,3] [,4]
[1,]    0 3.141593    0    0
[2,]    0 1.010000  100    0
[3,]    0 0.000000    0    0
[4,]    0 0.000000    0    0
```

Sparse matrices

Sparse matrices 'act' like regular (dense) matrices

They just only keep track of which entries are non zero and perform the operation on these entries

For our purposes, glmnet (and other methods) automatically check to see if $\mathbb{X}$ is a sparse matrix object

This can be a substantial speed/storage savings for large, sparse matrices

Warning: be on the look out for your sparse matrix becoming non-sparse!

Elastic net

Elastic net

The ridge solution is always unique and does well when the features are highly related to each other:

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

The lasso solution

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

isn't necessarily unique, but it can do model selection

However, it can do poorly at model selection if the features are highly related to each other

Elastic net

The elastic net was introduced to combine both of these behaviors

It solves

$$\hat{\beta}_{\text{elastic}}(\alpha, \lambda) = \underset{\beta}{\text{argmin}} \left[||\mathbb{Y} - \mathbb{X}\beta||_2^2 + \lambda \left((1-\alpha)||\beta||_2^2 + \alpha||\beta||_1 \right) \right]$$

We can do the elastic net in R with glmnet

```
alpha = 0.5 #This value is just an example
out.elasticNet = glmnet(x = X, y = Y, alpha=alpha)
```

The parameter alpha needs to be set

There does not exist any convention for this, but CV can be used

(You have to write this up yourself, though. Usually, people just play around with different values)

Refitted/Relaxed lasso

Refitted lasso

Since lasso does both...

- regularization
- model selection

... it can produce a solution that has too much bias

A common approach is to do the following two steps:

1. choose the λ via the 'one-standard-error rule'
2. refit the (unregularized) least squares solution on the selected features

Refitted lasso

We can do this in R via

```
intercept = 2
set.seed(1000)
X = matrix(rnorm(100),nrow=20,ncol=5)
Y = X %*% c(2,1,0,0,0) + rnorm(20) + intercept

Xtest = matrix(rnorm(100),nrow=20,ncol=5)
Ytest = Xtest %*% c(2,1,0,0,0) + rnorm(20) + intercept
```

Refitted lasso, part 2

Getting coefficients... a discussion on data structures

```
#Get CV curve (setting nfold = 3 due to n = 20)
require(glmnet);lasso.cv.glmnet = cv.glmnet(X,Y,alpha=1,nfold=3)
#Get beta hat with one-standard-error rule
> coef(lasso.cv.glmnet,s='lambda.1se')
6 x 1 sparse Matrix of class "dgCMatrix"
                            1
(Intercept)  2.250439416
V1           1.534049609
V2           0.685827900
V3          -0.001978132
V4           .
V5           .
> coef(lasso.cv.glmnet,s='lambda.1se')[-1]
[1] 1.534049609 0.685827900 -0.001978132 0.000000000 0.000000000
```

Refitted lasso: part 3

Getting coefficients... actually saving the objects

```
betaHat.temp = coef(lasso.cv.glmnet,s='lambda.1se')[-1]

# Identify which features are nonzero
selectedFeatures = which(abs(betaHat.temp) > 1e-16)

# Run regular least squares using those features
refitted.lm  = lm(Y~X[,selectedFeatures])

# Getting the coefficients
> refitted.lm$coefficients
     (Intercept) X[, selectedFeatures]1 X[, selectedFeatures]2
       1.8338306              1.2823930              0.7056985
> betaHat.refit = as.numeric(refitted.lm$coefficients)
[1] 1.8338306 1.2823930 0.7056985
```

Refitted lasso: part 4

Getting predictions and getting test error estimate of the risk

```
Yhat.refit    = drop(Xtest[,selectedFeatures] %*%
                     betaHat.refit[-1] + betaHat.refit[1])

> print( mean((Ytest - Yhat.refit)**2) )
[1] 1.627199
```

Refitted lasso: part 5

Getting the CV-minimum lasso...

```
betaHat.lasso = as.numeric(coef(lasso.cv.glmnet,s='lambda.min'))
Yhat.lasso    = Xtest %*% betaHat.lasso[-1] + betaHat.lasso[1]
#Or, we can get predictions via
Yhat.lasso_pred = predict(lasso.cv.glmnet,s='lambda.min',
                          newx = Xtest)
> dim(Yhat.lasso)
[1] 20  1
> dim(Yhat.lasso_pred)
[1] 20  1
> dim(drop(Yhat.lasso_pred))
NULL
> all.equal(drop(Yhat.lasso),drop(Yhat.lasso_pred))
[1] TRUE
```

Refitted lasso: part 6

How does this compare to usual CV-minimum lasso?

```
> cat('Refitted lasso: ',betaHat.refit,
+     ' with indices: ',selectedFeatures,'\n')
Refitted lasso: 1.833831 1.282393 0.7056985  with indices:  1 2

> print(betaHat.lasso)
[1] 1.72147359 0.99918769 0.54281411 -0.09434769 0.00000000 0.00
> print( mean((Ytest - Yhat.refit)**2) )
[1] 1.627199
> print( mean((Ytest - Yhat.lasso)**2) )
[1] 1.888338
```

Refitted lasso

Important: Do not attempt to do inference with the reported p-values. These are absolutely not valid!

However, the parameter values are estimates of the effect of that feature

(And the importance, if the features are standardized)

```
> cat('Refitted lasso: ',betaHat.refit,
+     ' with indices: ',selectedFeatures,'\n')
Refitted lasso:  2.248048 1.810153 0.98263  with indices:  1 2
```

Relaxed lasso

We can define a slightly more general procedure

Let's introduce an additional tuning parameter $\gamma \in [0, 1]$

Then the relaxed lasso is

$$\hat{\beta}_{\text{relax}}(\gamma, \alpha, \lambda) = \gamma \hat{\beta}_{\text{refit}}(\alpha, \lambda) + (1 - \gamma)\hat{\beta}_{\text{elastic}}(\alpha, \lambda)$$

Very coarse grids for the tuning parameters γ, α are used

Postamble:

- The automatically generated grid from glmnet needs to be checked

  (The grid needs to be checked for grid points that are too large or too far apart)
- Sparse matrices provide an efficient way to store data with a lot of zeros

  (Coercion to sparse matrices saves time and space. Be wary of destroying sparsity, though)
- We can generalize the lasso and ridge regression into the relaxed lasso

  (The relaxed lasso reports a weighted combination of the elastic net and the refitted elastic net)