Additional topics on the penalization

-Applied Analytics-

(No book references)

Lecturer: Darren Homrighausen, PhD

Preamble:

- Sparse matrices provide an efficient way to store data with a lot of zeros
- We can generalize the lasso and ridge regression into the refitted lasso

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!

(Note that the homework has a sparse matrix. You can play around with using the sparse matrix data structure if you like, but for simplicity I didn't include it in the assignment)

Refitted lasso

Refitted lasso

Since elastic net (for $\alpha > 0$) does both...

- regularization
- model selection

... it tends to produce a solution that

(Here, I'm referring to the elastic net with α and λ chosen by CV)

- has too much bias
- selects too many features

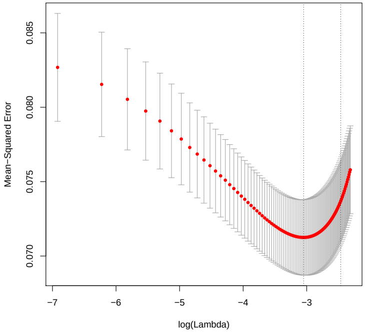
A common approach is to do the following two steps:

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

Some comments about glmnet and CV

The function cv.glmnet comes with a plotting function

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



- 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 $CV(\hat{\lambda})$

    (Note that the vertical bars are at 2*SE)

This is the so called one-standard-error rule: $\hat{\lambda}_{1se}$

The active set

Think of the p features, x_j, as being indexed from $1, 2, \ldots, p$

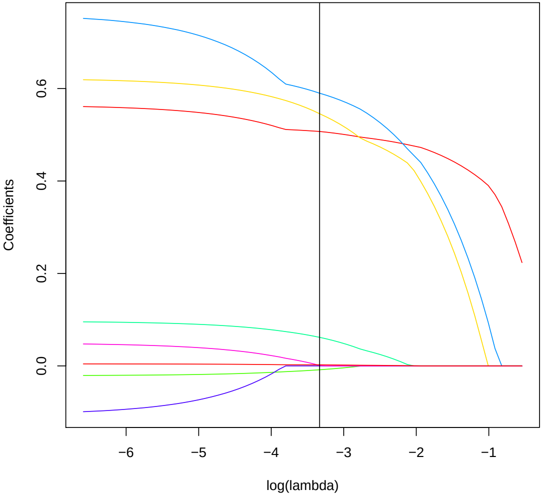
Suppose we estimate $\hat{\beta}_j = 0$, this is like saying x_j isn't related to the supervisor

(Hence, x_j is inactive or not selected)

So, let's collect all the j such that $\hat{\beta}_j \neq 0$, this is often notated $\hat{\mathcal{S}}$

This is known as the active or selected set

The active set



Different values of λ could result in a different active set: $\hat{\mathcal{S}}(\lambda)$

Refitted lasso

Define the refitted lasso:

1. Get the active set via the '1 standard error rule': $\hat{\mathcal{S}}(\hat{\lambda}_{1se})$
2. Refit with multiple linear regression using only the features in the active set:

$$\text{minimize over } \sum_{i=1}^{n} \left(Y_i - \left(\beta_0 + \sum_{j \text{ in } \hat{\mathcal{S}}(\hat{\lambda}_{1se})} \beta_j X_{ij} \right) \right)^2$$

Report this least squares solution as $\hat{\beta}_{refitted}$

Refitted logistic lasso

Note that we can apply this same concept to other methods

In particular, the output from the general elastic net regression
(assuming $\alpha > 0$, of course)

Or to logistic elastic net
(Instead of using least squares, we would use logistic regression)

Computing the refitted lasso

Refitted lasso

We can train lasso (over a grid of λ) with glmnet

```
require(glmnet)
lassoOut      = cv.glmnet(X ,Y, alpha=1)
```

Now, we need to get $\hat{\mathcal{S}}(\hat{\lambda}_{1se})$

```
betaHatTemp      = coef(lassoOut,s='lambda.1se')[-1]
Srefitted        = which(abs(betaHatTemp) > 1e-16)
```

Lastly, we can get the refitted lasso:

```
Xdf              = as.data.frame(X[,Srefitted])
refittedOut      = lm(Y ~ ., data = Xdf)
betaHatRefitted = coef(refittedOut)
```

If we have a test data set, we can get predictions via

```
XtestDF           = as.data.frame(Xtest[,Srefitted])
YhatTestRefitted = predict(refittedOut, XtestDF)
```

Postamble:

- 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 refitted lasso

  (The refitted lasso gets the selected features and computes then computes the least squares solution on these selected features)