

REGRESSION V: ADDITIONAL TOPICS ON THE LASSO

-APPLIED MULTIVARIATE ANALYSIS-

Lecturer: Darren Homrighausen, PhD

ℓ_1 -REGULARIZED REGRESSION

REMINDER 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:

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

SOME ADDITIONAL TOPICS

1. **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**
2. **ELASTIC NET:** For use when covariates are **related related** to each other
3. **REFITTED LASSO:** A proposal for **reducing** the lasso bias
4. **SCALED SPARSE REGRESSION:** A different take on choosing a particular λ than with CV

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 that number**

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(Matrix(sparseMat,sparse=F),'dgeMatrix')
```

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** automatically check to see if \mathbb{X} is a sparse matrix object

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

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
```

Elastic net

ELASTIC NET

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

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

The **lasso** solution

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

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

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

ELASTIC NET

The **elastic net** was introduced to combine both of these behaviors

It solves

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

We can do the elastic net in **R** with **glmnet**

```
alpha = 0.5
```

```
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 lasso

REFITTED LASSO

Since lasso does both

- regularization
- model selection

it can produce a solution that produces **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 covariates

REFITTED LASSO

We can do this in R via

```
#Get CV curve
lasso.cv.glmnet = cv.glmnet(X,Y,alpha=1)

#Get beta hat with one-standard-error rule
#      (remove intercept index -> [-1])
betaHat.temp = coef(lasso.cv.glmnet,s='lambda.1se')[-1]
# Identify which covariates are nonzero
selectedCovs = which(abs(betaHat.temp) > 1e-16)

# Run regular least squares using those covariates
refitted.lm = lm(Y~.,data=data.frame(X[,selectedCovs]))

##
# Output: either predictions or coefficients
##
Yhat.refitted = predict(refitted.lm,X_0[,selectedCovs])
betaHat.refitted = refitted.lm$coefficients
```

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 covariate

Scaled-sparse regression

SCALED-SPARSE REGRESSION

Recently, a new technique for finding the **lasso** solution has been proposed

Essentially, it reduces to the following idea:

- The optimally chosen λ looks like

$$\lambda_* \approx \sigma \sqrt{\frac{n}{\log p}}$$

where

- ▶ $\log p$ is the natural log of the # of covariates
- ▶ σ is the true standard deviation
- Hence, if we knew σ , we could form the optimal λ
- If we knew the optimal $\lambda = \lambda_*$, then we could estimate the variance via

$$\left\| Y - \mathbb{X} \hat{\beta}_{\text{lasso}, \lambda_*} \right\|_2^2$$

SCALED-SPARSE REGRESSION

This speaks to using an **iterative** approach

Scaled sparse regression (SSR) jointly estimates the regression coefficients and noise level in a linear model

It alternates between

1. estimating σ via

$$\hat{\sigma} = \sqrt{\frac{1}{n} \left\| Y - \mathbb{X} \hat{\beta}_{\text{lasso}, \lambda} \right\|_2^2}$$



2. setting $\lambda = \hat{\sigma} \sqrt{n / \log(p)}$

SCALED-SPARSE REGRESSION

We can do this in R via

```
library(scalreg)
lasso.ssr = scalreg(X = X,y = Y,LSE=F)
> names(lasso.ssr)
[1] "hsigma"          "coefficients"   "residuals"
[4] "fitted.values"  "type"           "call"
print(which(abs(lasso.ssr$coefficients)>1e-16))
```

Note that `scalreg` vexingly doesn't fit an intercept

( I have a work around for this, but it's a bit beyond the scope of this class )

SCALED-SPARSE REGRESSION

Also, the **LSE** parameter indicates if we want to do refitted lasso

Running

```
lasso.ssr = scalreg(X = X,y = Y,LSE=T)
```

Creates an object **lse**

```
> names(lasso.ssr)
[1] "hsigma"          "coefficients"   "residuals"
[4] "fitted.values"  "type"           "lse"
```

This object has all the relevant information. For instance predictions

```
Yhat.ssr.refitted = X_0 %*% lasso.ssr$lse$coefficients
```