Regression IV: Data Analysis Example

-Applied Multivariate Analysis-

Lecturer: Darren Homrighausen, PhD

An Example: Problem

The Portuguese Forest Service wants to find a model for predicting the severity of forest fires at a specific national park (Montesinho).

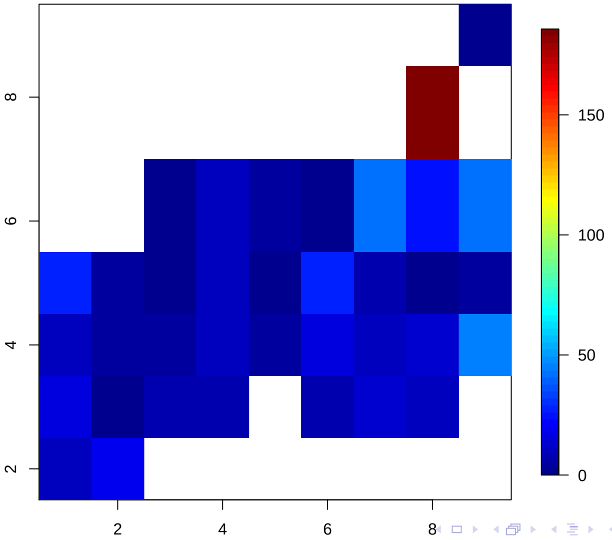
We've been hired to look at their data.

We wish to find important variables that are associated with the severity of fires.

An Example: Data Description

```
fire = read.table('../data/forestfires.csv',sep=',',
         header=T)
# Variables are:
#1. X - x-axis spatial coordinate within the
Montesinho park map: 1 to 9
#2. Y - y-axis spatial coordinate within the
Montesinho park map: 2 to 9
#3. month - month of the year: 'jan' to 'dec'
#4. day - day of the week: 'mon' to 'sun'
#5. FFMC - FFMC index from the FWI system: 18.7 to 96.20
#6. DMC - DMC index from the FWI system: 1.1 to 291.3
#7. DC - DC index from the FWI system: 7.9 to 860.6
#8. ISI - ISI index from the FWI system: 0.0 to 56.10
#9. temp - temperature in Celsius degrees: 2.2 to 33.30
#10. RH - relative humidity in %: 15.0 to 100
#11. wind - wind speed in km/h: 0.40 to 9.40
#12. rain - outside rain in mm/m2 : 0.0 to 6.4
```

Data visualization: Average fire area



```
x = X$x
y = X$y
x.un = sort(unique(x))
y.un = sort(unique(y))
plot.resp = rep(0,length(x.un)*length(y.un))
sweep = 0
for(i in x.un){
  for(j in y.un){
    sweep = sweep + 1
    plot.resp[sweep]  = mean(fire$area[x == i & y == j])
  }
}
plot.resp.mat = matrix(plot.resp,nrow=length(x.un),
                       ncol=length(y.un),byrow=T)
grid.list = list(x = x.un,y=y.un,z = plot.resp.mat)

require(fields)
image.plot(grid.list)
```

Training vs. testing

Suppose we set aside a subset of our data to evaluate our predictive capabilities

This set aside data is known as the test data

The remaining data that is used for estimation is the training data

Note: This is not quite the same as CV. While using CV we are only using the training data

Here is an example of how we might split this forest fire data

```
n       = nrow(fire)
nTrain  = round(n*0.98)
nTest   = n - nTrain
permute = sample(1:n,n,replace=FALSE)
train   = permute[1:nTrain]
test    = permute[(nTrain+1):n]
```

Transformations and object creation

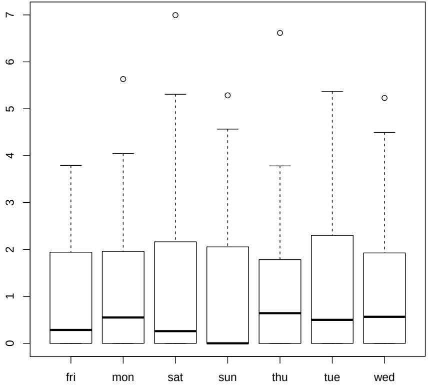
Due to the skewness in ‘area,’ we will log transform the response.

The ‘plus 1’ is due to many of the fires not burning any ground.

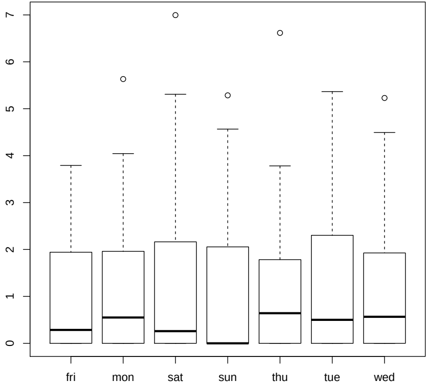
```
logArea = log(1+fire$area)
Ytrain  = logArea[train]
Ytest   = logArea[test]

X       = fire[,names(fire)!='area']
```

Plot for Day

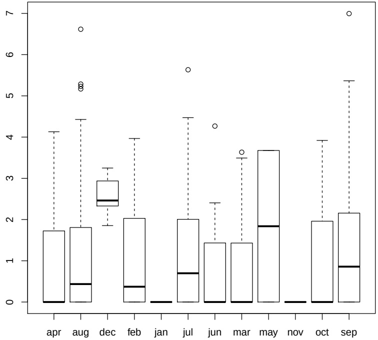


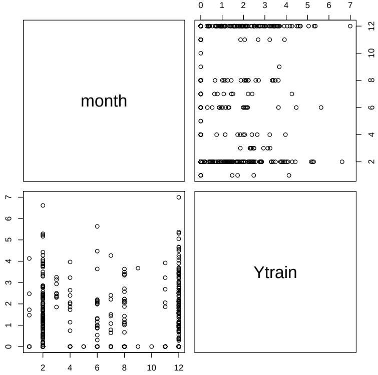
Plot for Day



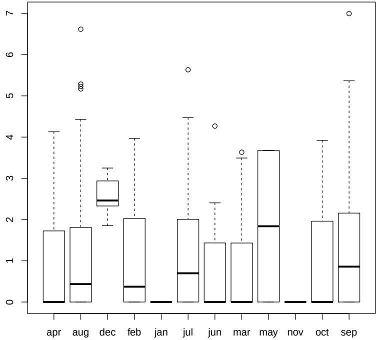

```
plot(fire$day,log(1+fire$area),ylab="log(area)")
X = X[,names(X)!='day']
```

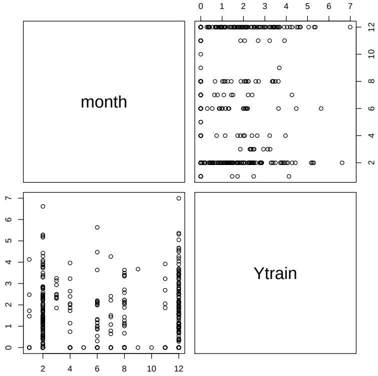
WHAT ABOUT MONTH?





What About Month?

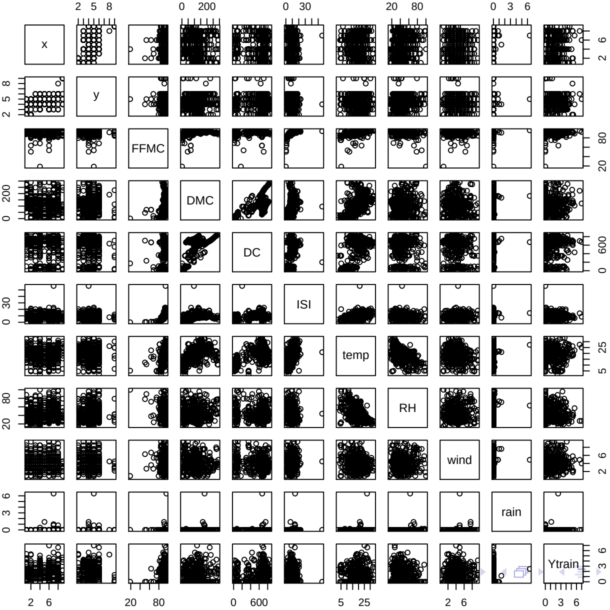




Some months might be important. We'll keep them. This leaves:

```
Xtrain  = X[train,]
Xtest   = X[test,]
```

Pairs plot (or Scatterplot Matrix)



Further data processing

Based on these plots, it appears that turning 'rain' into a dichotomous variable (equal to either 0 or 1) is appropriate.

We define a new variable 'rain'

```
rain        = X$rain
rain        = rain > 0
Xtrain$rain = rain[train]
Xtest$rain  = rain[test]
```

Also, we rescale the quantitative entries in $\mathbb{X}_{\text{train}}$

```
quant          = names(X)!=c('month')
Xtrain[,quant] = scale(Xtrain[,quant])
trainCenter = attributes(scale(Xtrain[,quant]))$'scaled:center'
trainScale  = attributes(scale(Xtrain[,quant]))$'scaled:scale'
```

And we rescale the quantitative entries in $\mathbb{X}_{\text{test}}$

```
Xtest[,quant]  = t(t(Xtest[,quant]) - trainCenter)
Xtest[,quant]  = t(t(Xtest[,quant])/trainScale)
```

Linear Model

```
x            0.058989   0.032494   1.815  0.07009 .
y           -0.016884   0.061438  -0.275  0.78358
monthaug     0.148795   0.841515   0.177  0.85973
monthdec     2.185436   0.816518   2.677  0.00769 **
monthfeb     0.210333   0.557555   0.377  0.70616
monthjan    -0.178002   1.215263  -0.146  0.88361
monthmay     0.582589   1.094097   0.532  0.59463
**some month omitted due to space
monthsep     0.726774   0.946730   0.768  0.44306
FFMC         0.010516   0.016614   0.633  0.52707
DMC          0.003543   0.001898   1.867  0.06249 .
DC          -0.001652   0.001315  -1.256  0.20959
ISI         -0.014988   0.017802  -0.842  0.40025
temp         0.050464   0.022334   2.260  0.02429 *
RH           0.004950   0.006333   0.782  0.43488
wind         0.066771   0.038192   1.748  0.08104 .
rainTRUE    -0.918718   0.532060  -1.727  0.08486 .
```

Forward

For doing Forward/Backward, we can treat the month variable as a group or individually. As a group, we do:

```
null = lm(Ytrain~1,data=as.data.frame(Xtrain))
full = lm(Ytrain~.,data=as.data.frame(Xtrain))
out  = step(null,scope=list(lower=null,upper=full),
    direction='forward')
Step:  AIC=344.62
Ytrain ~ DMC + wind + rain + x
        Df Sum of Sq    RSS    AIC
<none>                987.60 344.62
+ RH     1     2.980 984.62 345.06
+ DC     1     1.948 985.65 345.60
+ temp   1     1.778 985.82 345.69
+ ISI    1     1.708 985.89 345.73
+ FFMC   1     0.637 986.96 346.29
+ y      1     0.056 987.54 346.59
+ month 11    35.450 952.15 347.72
```

Forward: Individual Month Terms

We force R to consider them individually by creating:

```
XtrainInd = model.matrix(~., data=Xtrain,
             month=contrasts(Xtrain$month, contrasts=F))
```

```
### Forward Selection, ungrouped month
null = lm(Ytrain~1,data=as.data.frame(XtrainInd))
full = lm(Ytrain~.,data=as.data.frame(XtrainInd))
out  = step(null,scope=list(lower=null,upper=full),
        direction='forward')
```

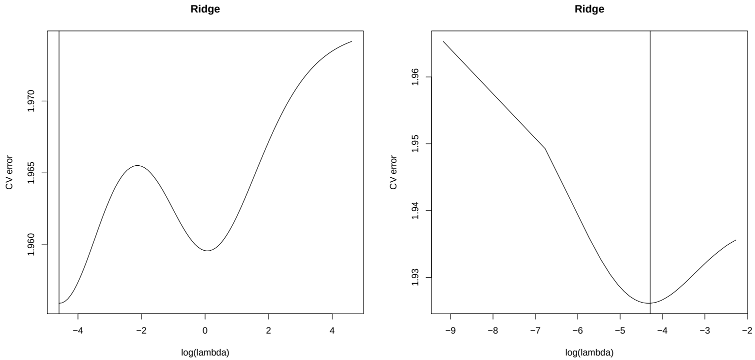
Forward

```
Step:  AIC=331.84
Ytrain ~ monthdec + temp + monthsep + x + wind + rainTRUE
<none>                        956.05 331.84
+ monthfeb   1     3.1688 952.88 332.12
+ DMC        1     2.8668 953.18 332.28
+ monthmay   1     1.7456 954.31 332.89
+ monthjun   1     1.4687 954.58 333.04
+ RH         1     1.3001 954.75 333.13
+ ISI        1     0.8796 955.17 333.36
+ monthnov   1     0.7754 955.28 333.42
+ monthmar   1     0.6342 955.42 333.49
+ FFMC       1     0.4080 955.64 333.61
+ monthjan   1     0.3139 955.74 333.67
+ DC         1     0.1116 955.94 333.78
+ monthoct   1     0.0377 956.01 333.82
+ monthaug   1     0.0109 956.04 333.83
+ monthjul   1     0.0046 956.05 333.83
+ y          1     0.0010 956.05 333.84
```

All Subsets

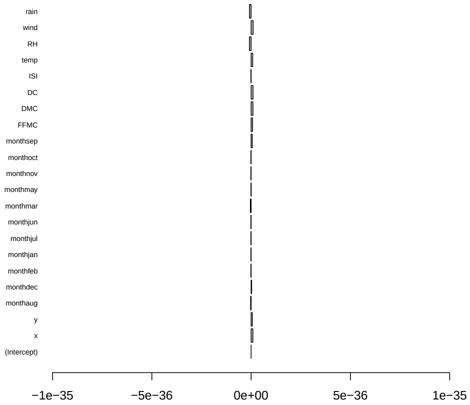
```
library(leaps)
leaps.plot =regsubsets(Ytrain~.,data=Xtrain, nbest=10,
    method='exhaustive')
out = summary(leaps.plot)
out$which[which.min(out$cp),]
(Intercept)           x           y   monthaug    monthdec
  TRUE            TRUE       FALSE       FALSE        TRUE
 monthfeb       monthjan    monthjul    monthjun    monthmar
  FALSE          FALSE       FALSE       FALSE       FALSE
 monthmay       monthnov    monthoct    monthsep        FFMC
  FALSE          FALSE       FALSE        TRUE       FALSE
   DMC             DC         ISI        temp          RH
  FALSE          FALSE       FALSE        TRUE       FALSE
   wind      rainTRUE
   TRUE          TRUE
```

Ridge Regression: Get minimum



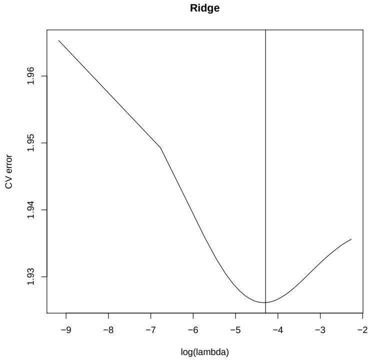

```
out.ridge = cv.glmnet(x=XtrainInd,y=Ytrain,alpha=0,
                      standardize=FALSE)
min.lambda = min(out.ridge$lambda)
lambda.new = seq(min.lambda*10,min.lambda*.01,length=100)
out.ridge = cv.glmnet(x=XtrainInd,y=Ytrain,alpha=0,
                      standardize=FALSE,lambda=lambda.new)
```

Ridge Regression: max(λ)



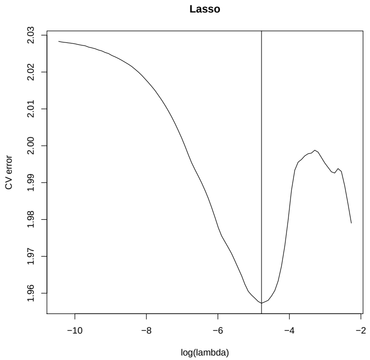

```
out.ridge = cv.glmnet(x=XtrainInd,y=Ytrain,alpha=0,
                      standardize=FALSE)
par(mar=c(5.1 ,6.1 ,4.1 ,2.1))
barplot(out.ridge$beta[,1],horiz=T,cex.names=.6,las=1)
```

Ridge Regression



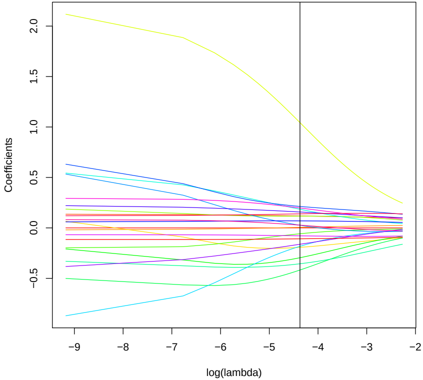

```
plot(log(ridge.out$lambda),ridge.out$cvm,
     xlab='log(lambda)', ylab='CV error',main='Ridge')
abline(v=log(ridge.out$lambda[which.min(ridge.out$cvm)]))
```

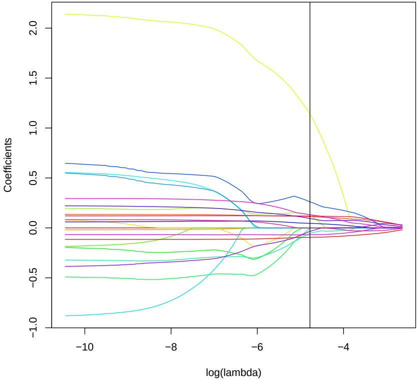
Lasso



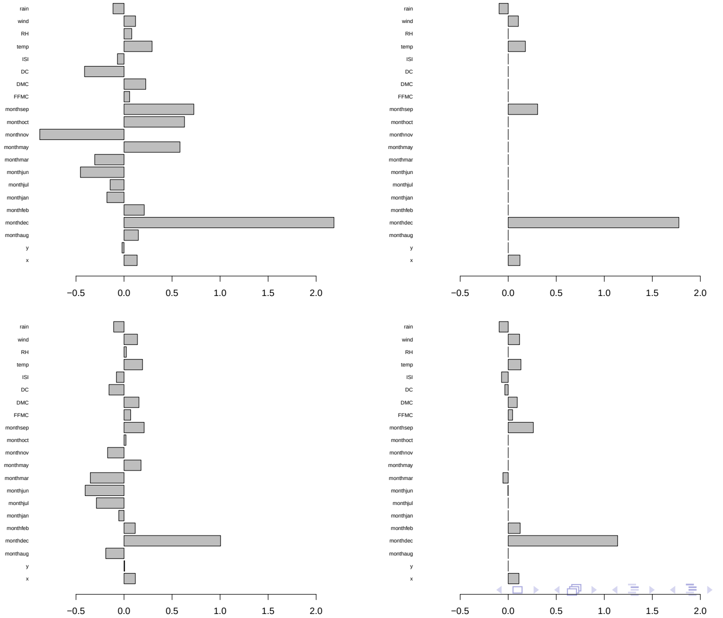

```
lasso.cv.glmnet = cv.glmnet(x=XtrainInd,y=Ytrain,alpha=1,
                            standardize=FALSE)
lamHat = lasso.cv.glmnet$lambda[which.min(lasso.cv.glmnet$cvm)]
plot(log(lasso.cv.glmnet $lambda),lasso.cv.glmnet $cvm,
        xlab='log(lambda)',ylab='CV error',main='Lasso')
abline(v=log(lamHat))
```

Ridge and lasso paths





Comparison: coefficients



Comparison: prediction

Each method has its own prediction function.

R will detect what type of prediction function is required

```
pred.lm    = predict(out.lm,as.data.frame(Xtest))
pred.lasso = predict(lasso.cv.glmnet,XtestInd,s='lambda.min')

p.val      = summary(out.lm)$coef[,4]
thresh     = .1
out.lmSig  = lm(Ytrain~.,
               data=as.data.frame(XtrainInd[,p.val<thresh]))
pred.lmSig = predict(out.lmSig,
                    as.data.frame(XtestInd[,p.val<thresh]))

predError = function(pred,test.data){
  return(sqrt(mean((pred - test.data)^2)))
}
predError(pred.lasso,Ytest) #For example
```

Comparison: prediction

Let's see how well these methods did at prediction:

Method	Prediction Error
LS	224.63
LS (significant)	16.42
Forward (grouped)	93.17
Forward (ungrouped)	4.21
Ridge	9.10
Lasso	1.10

Here:

- LS (significant): Keep covariates with p-value ≤ 0.1
- Forward (grouped): Treat month as a group
- Forward (ungrouped): Treat month individually

Some comments on this analysis

- If I did this analysis over, I would manually screen more of the months out before starting (or group them).
- There is a newer technique known as grouped lasso that can remove the variables as a group.
- Ridge did much worse than lasso and forward at prediction. This is not always the case.