Stat 8053: Variable Selection and Regularization, Fall 2013

In a regression problem with p predictors, we can reduce the dimension of the regression problem in two general ways:

- Replace $x_1, \ldots, x_p$, by $z_1, \ldots, z_q$, where $q < p$ and $z_j = \beta_j' x_j$ is a linear combination of the original p predictors. Nonlinear combinations are also possible, but not commonly used in the context we are studying. The most common methods that reduce dimension in this was are *principal components regression*, and newer methods like *SIR*. These will be discussed later in the semester. In addition, the usual linear model, and generalized linear model, are also special cases of this type of dimension reduction, provided that interactions and basis functions like polynomials and splines are not used.

- Devise some method for removing some of the predictors. If we concentrate on expectations, we could seek to find a partition $x = (x_G, x_D)$ such that
$$\mathrm{E}(y|x_G, x_D) = \mathrm{E}(y|x_G)$$
so once we know the "good" variables x_G the deleted variables x_D provide no further information.

We expect the model without x_D to be "better" in the sense estimates should be more precise and predictions should be better.

Why?

I can think of three general goals for model selection:

Learn about an effect of interest For example, about the effect of group membership, such as treatment or control, participate in a program or not, and so on. Other predictors may be relevant to the response, and the effect of the group membership variable can depend on other predictors. Methodology for this is to my knowledge not well developed

Variable discovery Identify the "active" predictors. For example, to find 10 or so active genes from a set of 100,000 potential genes based on a very limited data: See Ioannidis (2005), "Why most published research findings are false", `http://dx.doi.org/10.1371%2Fjournal.pmed.0020124`. This problem has an important multiple testing component.

Prediction Includes credit scoring, disease diagnosis, buyer behavior, and so on. This corresponds to much of machine learning.

Forward stepwise regression

We start with an initial class of models, e.g., $\mathrm{E}(Y|X = x) = \alpha + \beta' x$, $\mathrm{Var}(Y|X = x) = \sigma^2$, which is the linear regression model. All that is uncertain is which elements of β should be zero, and so dimension reduction is equivalent to finding the zeroes in β. We allow elements of X to be functionally related to allow for interactions and basis functions.

The usual forward selection algorithm is as follows:

1. Start with all elements of $\beta = 0$. Set $k = 0$.

2. Increment $k \to k + 1$.

3. At step k, add the variable that optimizes a criterion, usually reduction in deviance, or an information criterion like AIC or BIC.

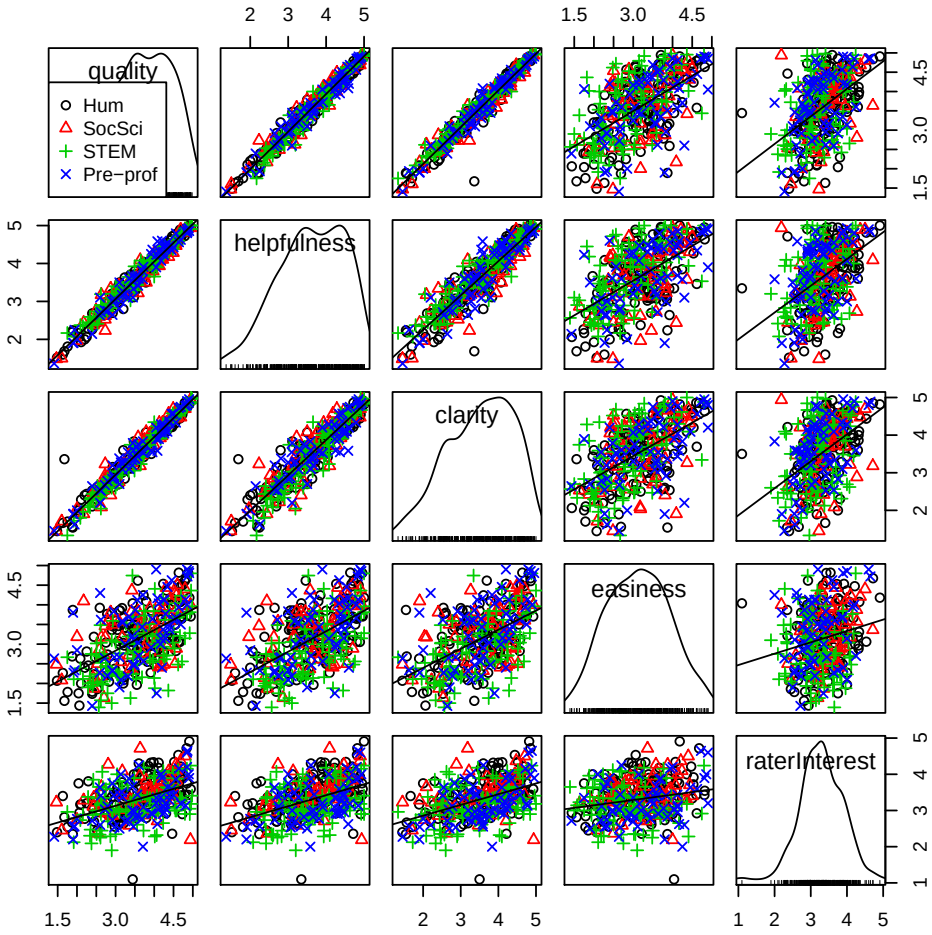
4. Stop if a stopping criterion is met or if all variables have been added. Older programs like SPSS use a "t to enter" criterion, while newer programs like `step` in R continue until AIC increases by the next deletion. See also Benjamini and Gavrilov (2009, *Annals*) for an FDA based method for stepwise regresion. If the criterion is not met, go to step 2.

Here is an example of the forward algorithm. The data in file `Rateprof` are aggregate professor ratings for 366 professors at a midwestern university (not the U of MN) from `RateMyProfessor.com`

```
> library(alr4)
> set.seed(1)
> str(Rateprof[, c(1, 5, 6, 8:12)])

'data.frame':        366 obs. of  8 variables:
 $ gender       : Factor w/ 2 levels "female","male": 2 2 2 2 2 2 2 2 2 2 ...
 $ pepper       : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ discipline   : Factor w/ 4 levels "Hum","SocSci",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ quality      : num  4.64 4.32 4.79 4.25 4.68 ...
 $ helpfulness  : num  4.64 4.55 4.72 4.46 4.68 ...
 $ clarity      : num  4.64 4.09 4.86 4.04 4.68 ...
 $ easiness     : num  4.82 4.36 4.6 2.79 4.47 ...
 $ raterInterest: num  3.55 4 3.43 3.18 4.21 ...

> scatterplotMatrix(~ quality + helpfulness + clarity + easiness + raterInterest|discipline, Rateprof, smoother=FALSE)
```




```
> Rateprof$score <- scale(with(Rateprof, (quality + helpfulness + clarity)/3))
> test <- sample(1:366, 200, replace=FALSE)
> m0 <- lm(score ~ 1, Rateprof, subset=test)
> m1 <- update(m0, ~ discipline + gender + pepper + easiness + raterInterest)
> (f <- as.formula(paste("~",paste(all.vars(formula(m1))[-1],collapse="+"))))

~discipline + gender + pepper + easiness + raterInterest
```

```
> library(MASS)
> FScoefs <- function(m0, m1, data, trace=FALSE) {
+    keepCoef <- function(m, aic) {
+     all <- names(coef(m1))
+     new <- names(coef(m))
+     ans <- rep(0,length(all))
+     ans[match(new, all)] <- coef(m)
+     ans
+    }
+    out <- with(data,stepAIC(m0, scope=list(lower=~1, upper=f), k=0,
+       trace=trace, keep=keepCoef, direction="forward"))
+    rownames(out$keep) <- names(coef(m1))
+    out$keep}
> coefs <- FScoefs(m0, m1, Rateprof, trace=TRUE)

Start:  AIC=-6
score ~ 1

                Df Sum of Sq    RSS     AIC
+ easiness       1    58.200 135.89 -77.290
+ raterInterest  1    36.504 157.59 -47.665
+ pepper         1    27.972 166.12 -37.120
+ discipline     3     4.290 189.80 -10.466
+ gender         1     0.085 194.01  -6.083
<none>                       194.09  -5.996

Step:  AIC=-77.29
score ~ easiness

                Df Sum of Sq    RSS      AIC
+ raterInterest  1   25.1780 110.72 -118.271
+ pepper         1   11.1474 124.75  -94.408
+ gender         1    2.7777 133.12  -81.420
+ discipline     3    1.8128 134.08  -79.976
<none>                       135.89  -77.290

Step:  AIC=-118.27
score ~ easiness + raterInterest
```

```
             Df Sum of Sq    RSS     AIC
+ pepper      1    8.1161 102.60 -133.50
+ discipline  3    3.0677 107.65 -123.89
+ gender      1    0.7572 109.96 -119.64
<none>                    110.72 -118.27

Step:  AIC=-133.5
score ~ easiness + raterInterest + pepper

             Df Sum of Sq     RSS     AIC
+ discipline  3    3.6378  98.961 -140.72
+ gender      1    0.8767 101.722 -135.21
<none>                    102.599 -133.50

Step:  AIC=-140.72
score ~ easiness + raterInterest + pepper + discipline

         Df Sum of Sq    RSS     AIC
+ gender  1   0.18585 98.775 -141.09
<none>                98.961 -140.72

Step:  AIC=-141.09
score ~ easiness + raterInterest + pepper + discipline + gender
```

The algorithm adds variables, one at a time, until adding the next variable increases, rather than decreases, the value of AIC. Factors are added all at once.

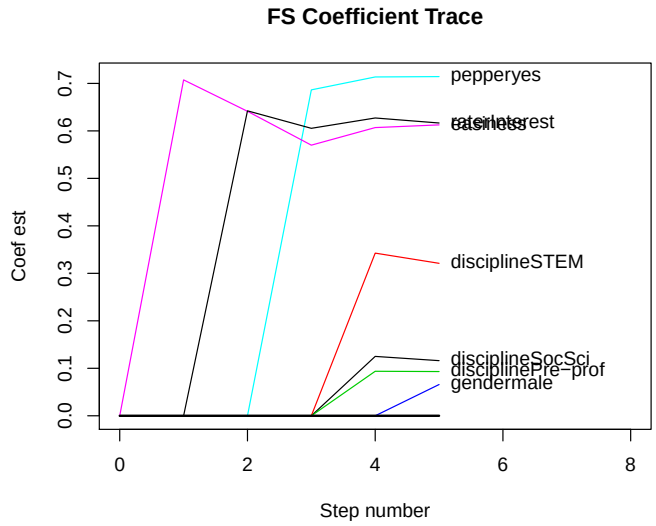
`stepAIC` is used rather than `step` because it allows saving information at each step via the `keep` argument, here saving the coefficients.

```
> print(coefs, digits=2)
```

```
                    [,1]  [,2]  [,3]  [,4]   [,5]   [,6]
(Intercept)       -0.061 -2.29 -4.19 -3.92 -4.252 -4.263
disciplineSocSci   0.000  0.00  0.00  0.00  0.125  0.116
disciplineSTEM     0.000  0.00  0.00  0.00  0.343  0.321
disciplinePre-prof 0.000  0.00  0.00  0.00  0.094  0.093
gendermale         0.000  0.00  0.00  0.00  0.000  0.066
pepperyes          0.000  0.00  0.00  0.69  0.714  0.714
easiness           0.000  0.71  0.64  0.57  0.607  0.613
raterInterest      0.000  0.00  0.64  0.61  0.627  0.617
```

Here is a *coefficient trace plot*:

```
> n <- length(coef(m1))-1
> steps <- 0:(dim(coefs)[2]-1)
> matplot(steps, t(coefs[-1,]),lty=1,type="l",xlim=c(0,n+1),
+   xlab="Step number",ylab="Coef est", main="FS Coefficient Trace")
> lines(c(0, 5), c(0, 0), lwd=2)
> xpos = rep(rev(steps)[1], n)
> ypos = coefs[-1, xpos[1]+1]
> text(xpos, ypos, rownames(coefs)[-1], cex=1, pos=4)
```



The figure shows the estimates for each variable at each step in the process. We have no stopping rule here, so the process is continued until all variables are included. At step zero, all coefficient estimates are zero. Coefficients become non-zero one at a time, and eventually reach their ols estimates from the "full" model.

The advantages of stepwise (FS, backward elimination or a hybrid) regression are:

1. Familiar, easily explained, widely used and implemented
2. Easily extended to other regression problems
3. Works pretty well, particularly for large n
4. Can be improved by fancy stopping rules

There are also several disadvantages:

1. Computational compromise to avoid "all possible" model computation. The machine learning people call "all possible" a *greedy search*.
2. Based on a model; if model is wrong, selection may be wrong.
3. Based on correlations only.

Lasso and regularization

Regularization has been intensely studied on the interface between statistics and computer science. We describe the basic idea through the *lasso*, Tibshirani (1996), as applied in the context of linear regression. The method starts by assuming a model like $\mathrm{E}(y|X = x) = \alpha + \beta' x$ and $\mathrm{Var}(Y|X) = \sigma^2$. The variable selection problem is formulated as finding the elements of β that equal zero. Estimates are chosen to minimize

$$\arg\min \sum (y_i - \alpha - \beta' x_i)^2 \text{ subject to } \sum |\beta_j| < t$$

This is equivalent to minimizing

$$\frac{1}{2n} \sum (y_i - \alpha - \beta' x_i)^2 + \lambda \sum |\beta_j|$$

which is just the usual least squares criterion with a penalty determined by λ for large coefficient estimates. If $\lambda = 0$ the lasso is the same as OLS; as λ increases, shorter vectors are preferred. There are many variations on this procedure, including application of it to other-than the linear model. Very high quality software in the package `glmnet` by Jerome Friedman, Trevor Hastie, Rob Tibshirani, is available in R. It computes estimates for a large number of values for λ at once. The "optimal" λ is selected by cross validation of some sort, although the validation procedure does not seem to be part of the `glmnet` package and you need to write your own.

Figure 1 illustrates how the lasso works, in the special case of exactly $p = 2$ predictors. The ellipses are contours of constant residual sum of squares, which is minimized at the point marked $\hat{\beta}$. As you move away from $\hat{\beta}$ the residual sum of squares increases, but all points on the same elliptical contour have the same value of *RSS*.

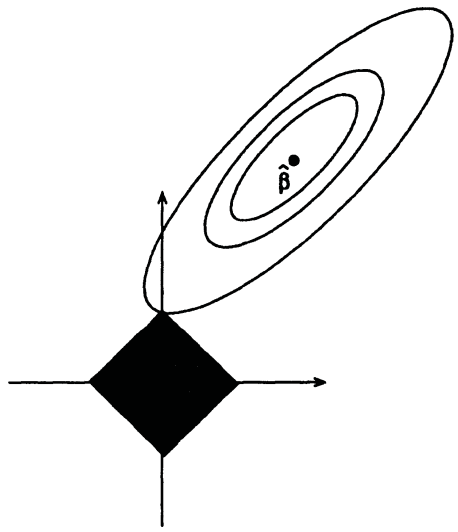


Figure 1: How lasso, works, from Tibshirani, 1996. The elliptical contours are centered at the ols estimator $\hat{\beta}$ are of the residual sum of squares function $\{\beta|(\beta-\hat{\beta})'X'X(\beta-\hat{\beta})=k\}$.

The black square is the set of vectors β that satisfy the constraint $\sum|\beta_j| \leq \lambda$, and for given λ the estimator will be the point with smallest residual sum of squares, and so the lasso estimator will be the point in the black square, or clearly on the boundary of the black square unless $\hat{\beta}=0$, that lies on the contour closest to $\hat{\beta}$. From the figure we can see that this will happen at one of the vertices of the black figure, at which one of the β_j is estimated to be exactly zero and the other is estimated to be non-zero. If the black figure is expanded, eventually the estimates of all the β_j will become non-zero, until for large enough λ the OLS estimate will be obtained. Thus for $\lambda=0$ we get all estimates equal zero, and for very large λ we get the OLS estimate.

Computing with glmnet

An important property of the lasso procedure is that it is *not invariant to linear transformations of the predictors*, because of the penalty based on absolute value of β. As a result it is "usual" to scale the data before beginning the analysis, usually to correlation scale. The `glmnet` package uses a very fast computing algorithm to compute estimates for many values of the penalty λ simultaneously. The "answer" requires selecting a value of λ, and this is done using cross-validation.

The most important function to use in the `glmnet` package is called `cv.glmnet`. The function `glmnet` is first run to get a sequence of λ-values that corresponding to getting one additional non-zero coefficient. Given this sequence of potential λs, the program does n-fold cross-validation, with $n=10$ by default, to estimate error, so `glmnet` is run n times, each with a fraction $(n-1)/n$ of the data, and prediction errors are accumulated on the remaining fold.

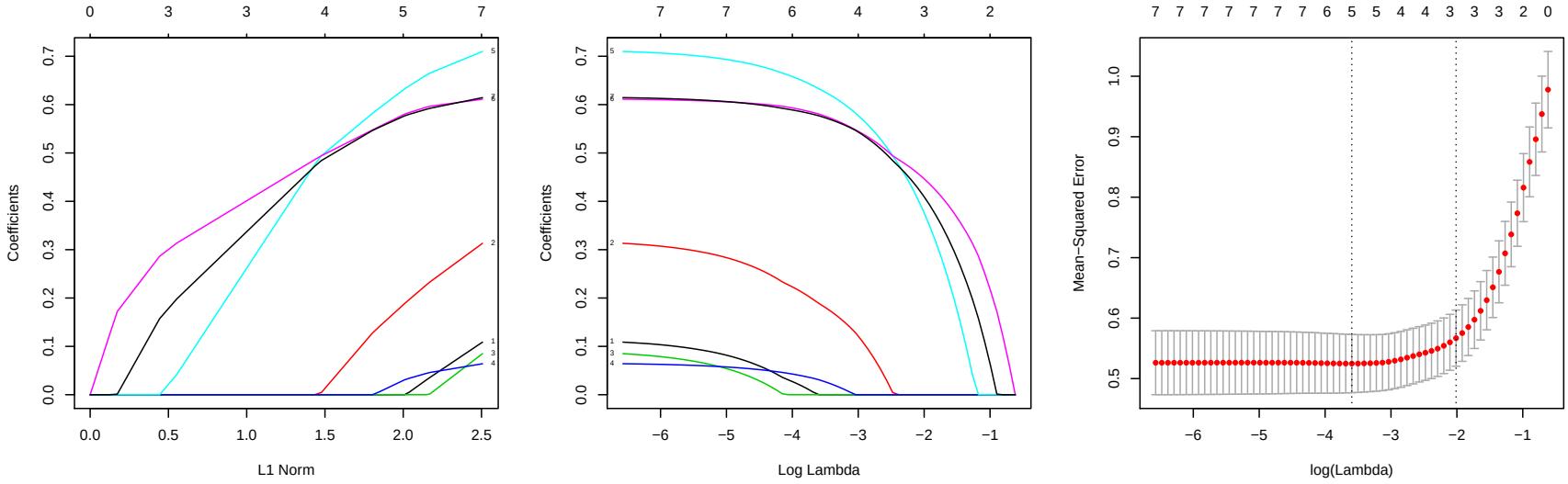
Here is the call to `cv.glmnet` for the example:

```
> library(glmnet)
> X <- model.matrix(update(m1, subset=1:dim(Rateprof)[1]))[, -1]
> Y <- Rateprof$score
> ans <- cv.glmnet(X[test, ], Y[test], standardize=TRUE)
```

Unlike most other functions we have seen, `glmnet` does not have a formula interface and you must specify the response and predictors explicitly. The argument `standardize=FALSE` is used because the data are already standardized.

The `glmnet` fit to all the (test) data is stored as `ans$glmnet.fit`. We can get the variable trace using

```
> par(mfrow=c(1, 3))
> plot(ans$glmnet.fit, "norm", label=TRUE)
> plot(ans$glmnet.fit, "lambda", label=TRUE)
> plot(ans)
```



This is a variable trace, just as with stepwise regression, the trace of the penalty parameter, and a record of the cross-validation. The standard error bars on the plot are estimated using the cross-validation. The minimum value of λ is

```
> log(ans$lambda.min)

[1] -3.594294
```

and a larger value of λ whose mean-square error is 1 SE larger is

```
> log(ans$lambda.1se)
```

```
[1] -2.01272

> coef(ans, s=ans$lambda.1se)

8 x 1 sparse Matrix of class "dgCMatrix"
                            1
(Intercept)        -2.8669772
disciplineSocSci    .
disciplineSTEM      .
disciplinePre-prof  .
gendermale          .
pepperyes           0.3801487
easiness            0.4481515
raterInterest       0.4119277
```

Apparently, the default is to use the `lambda.1se` as the solution.

Finally we can compute how well this does by applying the answer to the validation set.

```
> rss <- function(pred){
+    error <- (Y - pred)^2
+    c(Construction = sqrt(sum(error[test])/length(test)),
+      Validation=    sqrt(sum(error[-test])/(length(Y)-length(test))))
+  }
> rss.lassomin <- rss(predict(ans$glmnet.fit, s=ans$lambda.min, newx=X))
> rss.lasso1se <- rss(predict(ans$glmnet.fit, s=ans$lambda.1se, newx=X))
> befit <- stepAIC(m0,scope=list(lower=~1,upper=f),trace=F,direction="forward")
> rss.be <- rss(predict(befit, Rateprof))
> rss.ols <- rss(predict(m1, Rateprof))
> rbind(rss.lassomin, rss.lasso1se, rss.be, rss.ols)

             Construction Validation
rss.lassomin    0.7067690  0.7048039
rss.lasso1se    0.7432208  0.7463511
rss.be          0.7034244  0.7053879
rss.ols         0.7027636  0.7039740
```

Here are some advantages of the lasso method.

1. Large n, large p the linear model will be approximately OK.

2. $p >> n$ can be handled with this method and the automatic procedure produces an answer.

3. The argument `family=c("gaussian", "binomial", "poisson", "multinomial", "cox")` extends the method to continuous, binomial, and other problems.

4. Enormous literature of extensions and modifications

Here is a list of disadvantages.

1. The initial "full" model specification must include the "true" model as a special case or else information is lost. That occurs in this example in which the possibility that effects might differ by Sex are not included and are therefore not discovered.

2. The method is not invariant under rescaling columns of X or multiplying by a nonsingular matrix. The default is to scale all columns to have variance one, essentially working with a correlation matrix, but multiplication by a nonsingular matrix may completely change the results.

3. The method depends only on correlation coefficients, not on the original data, so it is likely to be sensitive to any modeling problems.

GlaucomaM

```
> show <- function(tt){
+   print(tt)
+   cat(paste("Misclassification rate =", round(1-sum(diag(tt))/sum(tt), 2),"\n"))
+   invisible()}
> data(GlaucomaM, package="TH.data")
> set.seed(123456)
> const <- sample(1:198, 150)
> library(randomForest)
> b2 <- randomForest(Class ~., GlaucomaM, subset=const)
> show(with(GlaucomaM, table(actual=Class[const], predicted=predict(b2))))

          predicted
actual     glaucoma normal
  glaucoma       62     14
  normal         13     61
Misclassification rate = 0.18

> show(with(GlaucomaM, table(actual=Class[-const],
+      predicted=predict(b2, newdata=GlaucomaM[-const, ]))))
```

```
          predicted
actual     glaucoma normal
  glaucoma       18      4
  normal          3     21
Misclassification rate = 0.15

> X <- as.matrix(GlaucomaM[, -63])
> Y <- as.factor(GlaucomaM[, 63])
> l2 <- cv.glmnet(X[const, ], Y[const], family="binomial")
> show(table(actual=Y[const],
+      predicted=predict(l2, newx=X[const,], type="class")))

          predicted
actual     glaucoma normal
  glaucoma       64     12
  normal         13     61
Misclassification rate = 0.17

> show(table(actual=Y[-const],
+      predicted=predict(l2, newx=X[-const,], type="class")))

          predicted
actual     glaucoma normal
  glaucoma       18      4
  normal          6     18
Misclassification rate = 0.22
```