

Variable Selection and Regularization

Sanford Weisberg

October 24, 2011

Variable Selection

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

- Replace x_1, \dots, x_p , by z_1, \dots, 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 way 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

$$E(y|x_G, x_D) = E(y|x_G)$$

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.

Forward stepwise regression

We start with an initial class of models, e.g., $E(Y|X) = \alpha + \beta'X$, $\text{Var}(Y|X) = \sigma^2$, 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 $k \rightarrow k + 1$.
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 regression. If the criterion is not met, go to step 2.

Here is an example of the forward algorithm applied to the Australian Athletes data, in which we take lean body mass LBM as the response. The `step` function in base R can be used for stepwise regression. Next, we read the data, transform, center and scale (this is required for the lasso, following), and fit the null and full models and then forward stepwise regression.

```
> library(alr3)
> set.seed(1)
> test <- sample(1:202, 150, replace = FALSE)
> ais0 <- with(ais, data.frame(LBM = LBM, logSSF = log(SSF), logWt = log(Wt),
+   Sex = Sex, logHg = log(Hg), logHt = log(Ht), logRCC = log(RCC),
+   logHc = log(Hc), logWCC = log(WCC), logFerr = log(Ferr)))
> ais0 <- as.data.frame(scale(ais0))
> m0 <- lm(LBM ~ 1, ais0, subset = test)
> m1 <- update(m0, LBM ~ logWt + logSSF + logRCC + Sex + logHg +
+   logHt + logWCC + logHc + logFerr)
> (f <- as.formula(paste("~", paste(names(coef(m1))[-1], collapse = "+"))))

~logWt + logSSF + logRCC + Sex + logHg + logHt + logWCC + logHc +
  logFerr

> step(m0, scope = list(lower = ~1, upper = f), direction = "forward")
```

```
Start:  AIC=-2.18
LBM ~ 1
```

	Df	Sum of Sq	RSS	AIC
+ logWt	1	121.995	23.885	-271.606
+ logHt	1	95.002	50.878	-158.180
+ Sex	1	86.966	58.914	-136.182
+ logHg	1	57.015	88.866	-74.527
+ logHc	1	52.160	93.720	-66.548
+ logRCC	1	45.352	100.529	-56.029
+ logFerr	1	16.583	129.297	-18.279
+ logSSF	1	5.711	140.169	-6.168
<none>			145.880	-2.178
+ logWCC	1	0.450	145.430	-0.641

```
Step:  AIC=-271.61
LBM ~ logWt
```

	Df	Sum of Sq	RSS	AIC
+ logSSF	1	19.5004	4.3849	-523.87
+ Sex	1	14.8796	9.0057	-415.92
+ logHg	1	7.9754	15.9099	-330.55
+ logHc	1	7.8151	16.0701	-329.05
+ logRCC	1	6.8500	17.0352	-320.30
+ logHt	1	2.1569	21.7284	-283.80
+ logFerr	1	1.7648	22.1204	-281.12
<none>			23.8853	-271.61
+ logWCC	1	0.1081	23.7772	-270.29

```
Step: AIC=-523.87
LBM ~ logWt + logSSF
```

	Df	Sum of Sq	RSS	AIC
+ Sex	1	0.196906	4.1880	-528.76
+ logFerr	1	0.077894	4.3070	-524.56
<none>			4.3849	-523.87
+ logRCC	1	0.031962	4.3529	-522.97
+ logHt	1	0.028476	4.3564	-522.85
+ logHg	1	0.012304	4.3726	-522.29
+ logHc	1	0.007048	4.3779	-522.11
+ logWCC	1	0.000008	4.3849	-521.87

```
Step: AIC=-528.76
LBM ~ logWt + logSSF + Sex
```

	Df	Sum of Sq	RSS	AIC
<none>			4.1880	-528.76
+ logHt	1	0.0306762	4.1573	-527.86
+ logFerr	1	0.0181920	4.1698	-527.41
+ logRCC	1	0.0023285	4.1857	-526.85
+ logHc	1	0.0007616	4.1872	-526.79
+ logWCC	1	0.0005516	4.1874	-526.78
+ logHg	1	0.0003442	4.1877	-526.77

Call:

```
lm(formula = LBM ~ logWt + logSSF + Sex, data = ais0, subset = test)
```

Coefficients:

(Intercept)	logWt	logSSF	Sex
-0.003095	0.937928	-0.317341	-0.075316

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

To draw the graph that is shown in Figure ??, I'm going to repeat this calculation, but is the `stepAIC` function from the `MASS` package because it allows saving and manipulating computed values from each step in the process.

First is presented a function that will extract the information, given a model `m1` and a null model `m0`. The `keep` argument to `stepAIC` is used to keep only the coefficient estimates at each step. The function `FScoefs` calls `stepAIC` and keeps the estimated coefficient vector at each step, with zeroes filled-in for variables not included in the current mean function. Also, in the call the call to `stepAIC`, set `k = 0`. The criterion for selecting a subset is to minimize $RSS + kp$, where p is the number of predictors. The choice of $k = 2$ is the default gives AIC, while $k = \log(n)$ gives BIC. By setting $k = 0$, we minimize RSS , which means we also continue to the "full" model.

```
> 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
+ }

```

```
> print(coefs <- FScoefs(m0, m1, ais0), digits = 2)
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
(Intercept)	0.047	0.012	-0.0083	-0.0031	-0.0025	-0.0022	-0.0025	-0.0039	-0.0041
logWt	0.000	0.923	0.9899	0.9379	0.9599	0.9588	0.9626	0.9667	0.9666
logSSF	0.000	0.000	-0.3696	-0.3173	-0.3228	-0.3244	-0.3267	-0.3285	-0.3296
logRCC	0.000	0.000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0252	0.0319
Sex	0.000	0.000	0.0000	-0.0753	-0.0757	-0.0693	-0.0726	-0.0687	-0.0679
logHg	0.000	0.000	0.0000	0.0000	0.0000	0.0000	-0.0071	-0.0289	-0.0182
logHt	0.000	0.000	0.0000	0.0000	-0.0267	-0.0234	-0.0258	-0.0281	-0.0270
logWCC	0.000	0.000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
logHc	0.000	0.000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-0.0177
logFerr	0.000	0.000	0.0000	0.0000	0.0000	0.0096	0.0092	0.0109	0.0103
	[,10]								
(Intercept)	-0.0041								
logWt	0.9665								
logSSF	-0.3290								
logRCC	0.0322								
Sex	-0.0679								
logHg	-0.0182								
logHt	-0.0272								
logWCC	-0.0025								
logHc	-0.0172								
logFerr	0.0102								

```

> n <- length(coef(m1)) - 1
> steps <- 0:(dim(coefs)[2] - 1)

```

Finally, draw the plot:

```

> matplot(steps, t(coefs[-1, ]), lty = 1, type = "l", xlim = c(0,
+   n + 2), xlab = "Step number", ylab = "Coef est")
> xpos = rep(rev(steps)[1], 4)
> ypos = coefs[2:5, xpos[1]]
> text(xpos, ypos, rownames(coefs)[2:5], cex = 0.6, 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

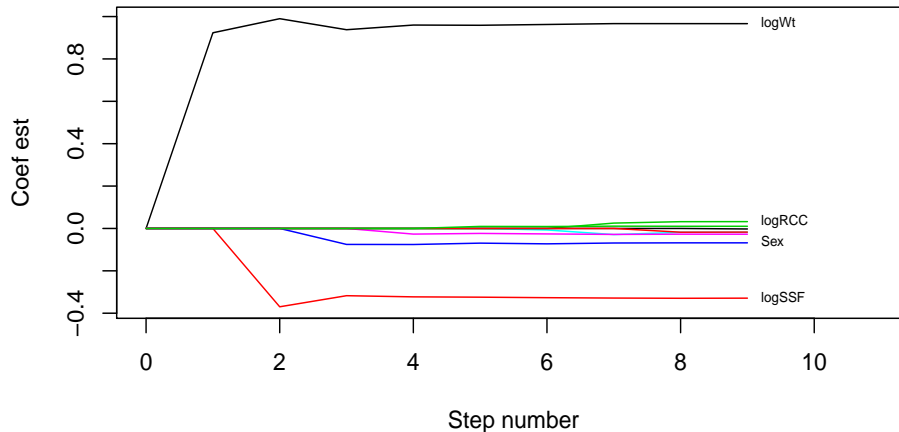


Figure 1: Variable trace for stepwise regression.

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 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 $E(y|X = x) = \alpha + \beta'x$ and $\text{Var}(Y|X) = \sigma^2$. 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

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

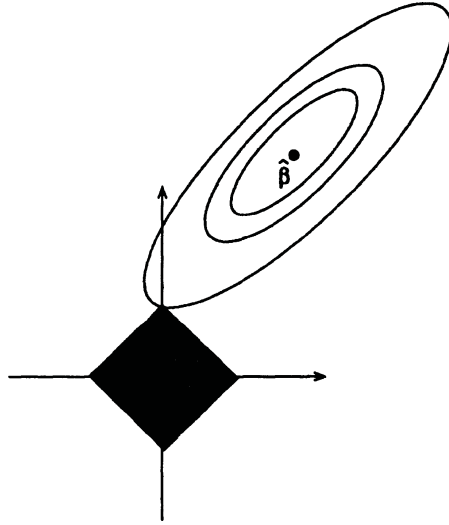


Figure 2: How lasso, works, from Tibshirani, 1996

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 2 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 .

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. As λ increases we seem to “add” variables whenever a new vertex is crossed, and so we get a search path as shown in Figure 3. In the call to `glmnet` the `standardize` argument is set to `FALSE` since the data frame `ais0` is already (nearly) standardized. The `glmnet` does not have an interface that will allow using a formula (one is given on page 417 of Fox and Weisberg, *An R Companion to Applied Regression*). You must provide the predictors and the response explicitly in a matrix and a vector, respectively, omitting a column of ones for the intercept.

```
> library(glmnet)
> X <- as.matrix(ais0[, -1])
> Y <- ais0[, 1]
> ans <- glmnet(X[test, ], Y[test], standardize = FALSE)
> plot(ans, "norm", label = TRUE)
```

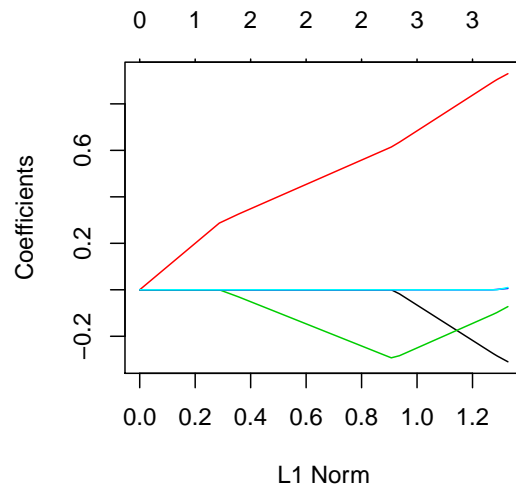


Figure 3: Variable trace for the lasso.

This is a variable trace, just as with stepwise regression, but of course the trace is different. Selection of λ , and hence of the variables to remove, can be done using generalized cross validation. The function `cv.glmnet` in the `glmnet` package does this automatically using ten-fold cross-validation by default:

```
> ans <- cv.glmnet(X[test, ], Y[test], standardize = FALSE)
> (coef(ans, s = "lambda.1se"))
```

10 x 1 sparse Matrix of class "dgCMatrix"

```
      1
(Intercept) 0.006909257
logSSF      -0.209076476
logWt       0.828613895
Sex         -0.150802426
logHg       .
logHt       0.000000000
logRCC      0.000000000
logHc       .
logWCC      .
logFerr     0.000000000
```

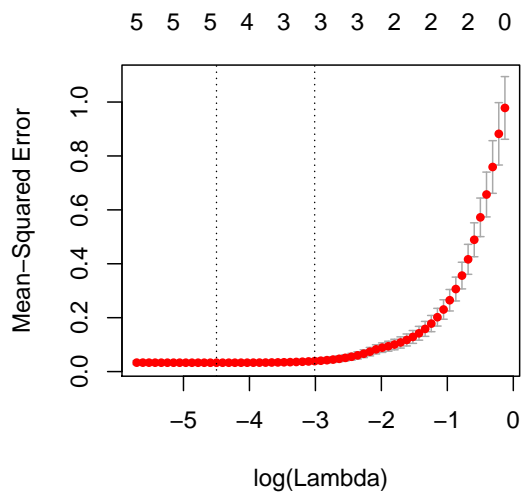
```
> (coef(ans, s = "lambda.min"))
```

10 x 1 sparse Matrix of class "dgCMatrix"

```
      1
(Intercept) -0.001759332
logSSF      -0.306013285
logWt       0.926107133
Sex         -0.076067469
logHg       .
logHt       0.000000000
```

```
logRCC      0.005152609
logHc       .
logWCC      .
logFerr     0.008422734
```

```
> plot(ans)
```



The standard error bars on the plot are estimated using the cross-validation. The minimum value of λ is

```
> ans$lambda.min
```

```
[1] 0.01111321
```

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

```
> ans$lambda.1se
```

```
[1] 0.04923843
```

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.lassomin <- sum((Y[-test] - predict(ans, X[-test, ], s = "lambda.min"))^2)
> rss.lasso1se <- sum((Y[-test] - predict(ans, X[-test, ], s = "lambda.1se"))^2)
> befit <- stepAIC(m0, scope = list(lower = ~1, upper = f), trace = F,
+   direction = "forward")
> rss.be <- sum((Y[-test] - predict(befit, ais0[-test, ]))^2)
> rss.ols <- sum((Y[-test] - predict(m1, ais0[-test, ]))^2)
> data.frame(lasso = rss.lassomin, lasso.1se = rss.lasso1se, be = rss.be,
+   ols = rss.ols)
```

```
      lasso lasso.1se      be      ols
1 2.027619  2.21864 2.10682 2.091182
```

Here are some advantages of the lasso method.

1. Large n , large p the linear model will be approximately OK.
2. $p \gg n$ 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.

Figure 4 summarizes a very small simulation comparing eleven methods of model fitting that I did for another purpose. The data were generated from

$$y|x \sim \text{Poisson}(\exp(x_1 + x_2 + x_3))$$

with either three or nine additional “noise” variables. This is unfavorable to the lasso because the linear model is not correct, although there is a linear predictor, and the Poisson will produce occasional outliers. My criterion is the angle between the true value of β and the estimated value of β .

The methods include Poisson log-linear regression and Poisson with backward elimination; Gaussian regression with a linear model with response either Y or \sqrt{Y} and with and without backward elimination; `sir`, with and without backward elimination; the lasso, among others. The two Poisson methods should be, and are, the most effective, and serve as a standard to judge the other methods. Gaussian or ols regression with response \sqrt{Y} should do well in this problem, and that is in fact the case; `sir`, though more variable, does almost as well on average. The lasso does no better than ols in these examples, which is no real surprise since both of these are incorrectly assuming an underlying linear model. The results are similar for $p = 6$ and $p = 12$, except for `save`, which seems to do poorly with $p = 12$.

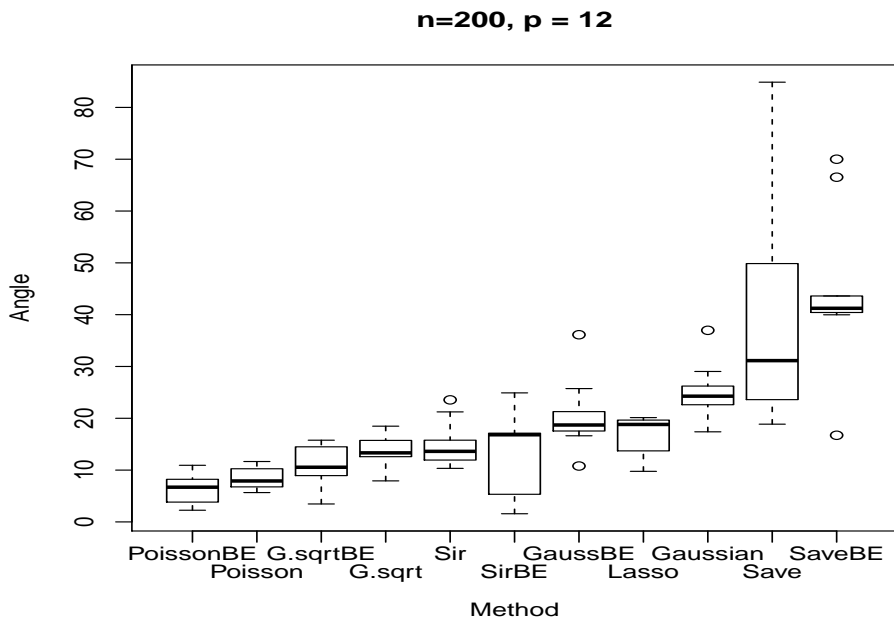
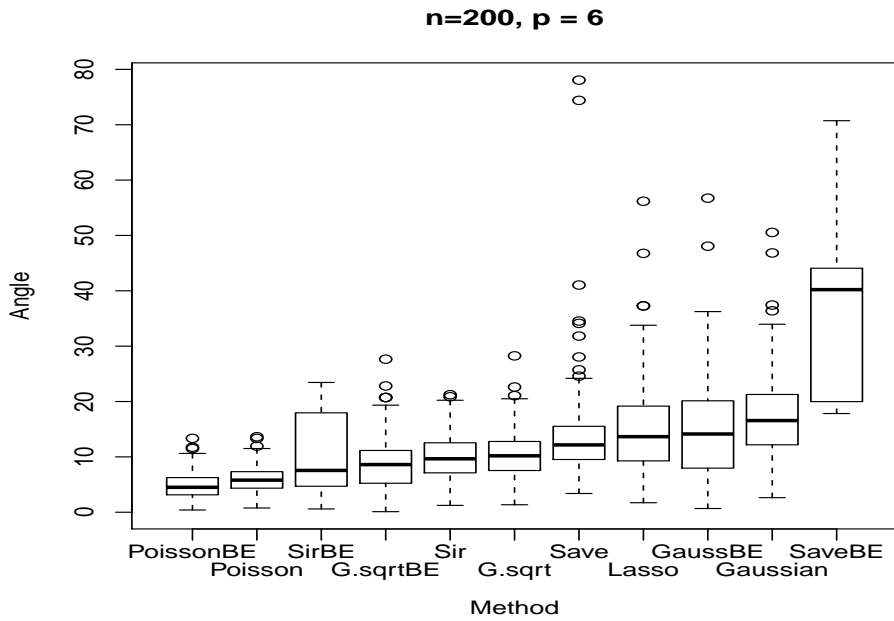


Figure 4: Results of a small simulation comparing eleven methods of estimating the single direction in a single-index model with Poisson errors. Shown in the graphs are boxplots of the angle between the estimated spanning vector and true central subspace, so small values are better.