MSG500/MVE190
Linear Models - Lecture 15

Rebecka Jörnsten
Mathematical Statistics
University of Gothenburg/Chalmers University of Technology

December 13, 2012

1 Regularized regression

In ordinary least squares we assume that $y = X\beta + \epsilon$, where X is the $n \times p$ design matrix, errors are uncorrelated and have constant error variance σ^2. If the x-variables are correlated, the estimated coefficients $\hat{\beta}$ have high estimation variance and are correlated. This means that it is difficult to get a direct interpretation of the x-variable effect on the response y. We called this phenomenon "the collinearity problem".

The source of the problem lies in the numerically unstable matrix inverse operation we perform to estimate β:

$$\hat{\beta} = (X'X)^{-1}X'y.$$

When the x's are correlated, the $X'X$ matrix is near singular (since columns are near perfectly correlated). Numerical instability of the inverse is something we encounter when x's are correlated, but also when the number of predictors (p) is large compared with the sample size n. Indeed, the inverse cannot be computed if $p > n$.

What can we do to fix the problem?

1.1 Principal component regression

Let us take a closer look at the design matrix X. The singular value decomposition of X is

$$X = UDV', \quad \text{where} U'U = I \text{ and} V'V = I.$$

The covariance matrix $Cov(X)$ is proportional to $X'X = VD^2V'$.
In Figure 1 I depict a data set where $p = 2$ and x_1 and x_2 are generated from a multivariate normal with covariance matrix $\Sigma = \begin{pmatrix} 1.5 & 1 \\ 1 & 1 \end{pmatrix}$. As you can see, the x-variables are correlated. I compute the SVD of the matrix and depict the right component direction $V : V_1, V_2$ (the principal component directions). The first component V_1 comprises the direction in the data with maximum variance, which is the first element of the diagonal matrix D. The second component V_2 comprises the direction with the second more variance, orthogonal to direction V_1.

```
> library(MASS)
> x <- mvrnorm(250, mu = c(0, 0), Sigma = matrix(c(1.5, 1, 1, 1),
+     2, 2))
> plot(x)
> prx <- prcomp(x)
> print(prx$sdev^2/sum(prx$sdev^2))

[1] 0.93025595 0.06974405

> abline(0, prx$rot[2, 1]/prx$rot[1, 1], col = 2)
> abline(0, prx$rot[2, 2]/prx$rot[1, 2], col = 3)
```

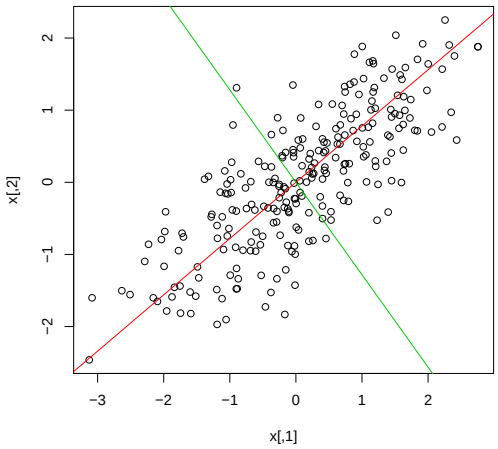

Figure 1: Rotating the coordinate system. 1st PC = red, 2nd PC = green

We can reexpress the data X in the new coordinate system comprised by the principal component direction. That means, we rotate the data so that the red line above is the new x_1 axis and the green is the new x_2 axis. The new x-variables in the rotated coordinate system are denoted $\tilde{X}$. Mathematically, we construct the as $\tilde{X} = XV$.

The new variables $\tilde{X}$ are "ideal" in some sense for doing regression; (1) they are uncorrelated so we don't have the numerical instabilities of the original x-data; (2) they are ordered in terms of decreasing spread. In the new coordinate system we have

$$\tilde{X}: \quad \tilde{X}'\tilde{X} = V'X'XV = V'VD^2V'V = D^2$$

and so $V(x_1) = d_1^2 \ > \ V(x_2) = d_2^2 \ > \cdots$.

In the new coordinate system we have

$$y = \tilde{X}\tilde{\beta} + \epsilon.$$

We use least squares to solve for $\tilde{\beta}$:

$$\hat{\tilde{\beta}} = (\tilde{X}'\tilde{X})^{-1}\tilde{X}'y.$$

In the new coordinate system, it follows that $\tilde{\beta} = V'\beta$, so we can always go from one model to the other using the PC directions in V.

What are the properties of the PC regression estimates?

$$V(\hat{\tilde{\beta}}) = \sigma^2(\tilde{X}'\tilde{X})^{-1} = \sigma^2 D^{-2}.$$

So, the PC regression coefficient estimate are uncorrelated. In addition, their precision is the order of the PCs. The first coefficient estimate $\hat{\tilde{\beta}}_1$ is the most precise, etc.

How do we use PC regression in practise? We can choose to select the top K PC variables a priori (before we see y) as the ones we can estimate with highest precision. However, the PC decomposition did not take y into account. It is therefore not at all guaranteed that the PCs with largest variance are the ones related to y. We can therefore use standard F-tests or model selection approaches to select between the PCs in the new coordinate system. (There is also the alternative to consider rotations of the coordinate system that enhances the correlation between y and the new coordinates $\tilde{X}$ - this is called Partial Least Squares (PLS).)

What are the pro's and con's of PC regression? Well, it is simple to use. The new coordinate are uncorrelated so the numerical performance of selection etc is better. However, we have lost the direct interpretation since each PC coordinate tends to involve *all* the x-variables. Recently, *sparse PCA* was presented (Zou, Hastie, Tibshirani, 2004: `http://www.stanford.edu/ hastie/Papers/sparsepc.pdf`). This allows you to explore a whole spectrum of rotation based methods. At one extreme, we use the original X, which may have collinearity issues. At the other extreme, we use $\tilde{X} = PC(X)$, which are uncorrelated but difficult to interpret. Between these extremes we have $\tilde{X} = sparsePC(X)$, where each PC component $\tilde{x}_j$ involves only a subset of x-variables, and as a result cannot be guaranteed to be completely uncorrelated with other $\tilde{x}_k$. The more sparse you make the PCs, the more collinearity remains and the easier to interpret the models. The less sparse you make them, the more uncorrelated the $\tilde{x}$ are, and each $\tilde{x}$ can now involve more x's and so the models are more difficult to interpret.

Let's try this out on the heart disease data.

```
> SA <- read.table("SA.dat", header = T)
> SA$famhist <- SA$famhist - 1
> ii <- sample(seq(1, dim(SA)[1]), 200)
> mm <- lm(ldl ~ age + sbp + adiposity + obesity + typea + alcohol +
+     alcind + tobacco + as.factor(tobind) + as.factor(chd) + as.factor(famhist),
+     data = SA, subset = ii)
> print(summary(mm))

Call:
lm(formula = ldl ~ age + sbp + adiposity + obesity + typea +
    alcohol + alcind + tobacco + as.factor(tobind) + as.factor(chd) +
    as.factor(famhist), data = SA, subset = ii)

Residuals:
    Min      1Q  Median      3Q     Max
-3.4995 -1.1611 -0.1929  0.8509  6.5493

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)         0.623119   1.270696   0.490 0.624440
age                -0.015561   0.012855  -1.211 0.227602
sbp                 0.003066   0.006014   0.510 0.610759
adiposity           0.109680   0.032128   3.414 0.000785 ***
obesity             0.023437   0.052892   0.443 0.658190
typea               0.003121   0.012498   0.250 0.803078
alcohol            -0.013047   0.005676  -2.298 0.022635 *
alcind             -0.012443   0.302454  -0.041 0.967227
tobacco            -0.003835   0.033456  -0.115 0.908861
as.factor(tobind)1  0.690793   0.329182   2.099 0.037196 *
as.factor(chd)1     0.730676   0.298585   2.447 0.015319 *
as.factor(famhist)1 0.287189   0.259048   1.109 0.269005
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.717 on 188 degrees of freedom
Multiple R-squared: 0.3225,        Adjusted R-squared: 0.2829
F-statistic: 8.136 on 11 and 188 DF,  p-value: 1.427e-11

> selmm <- step(mm, trace = F)
> print(summary(selmm))

Call:
lm(formula = ldl ~ adiposity + alcohol + as.factor(tobind) +
    as.factor(chd), data = SA, subset = ii)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-3.5136 -1.1473 -0.1425  0.8808  7.2094

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)         1.436500   0.422031   3.404 0.000807 ***
adiposity           0.107613   0.015182   7.088 2.43e-11 ***
alcohol            -0.012557   0.005058  -2.483 0.013886 *
as.factor(tobind)1  0.572534   0.302032   1.896 0.059491 .
as.factor(chd)1     0.684710   0.269206   2.543 0.011752 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.703 on 195 degrees of freedom
Multiple R-squared: 0.3088,        Adjusted R-squared: 0.2946
F-statistic: 21.78 on 4 and 195 DF,  p-value: 7.175e-15
```

I first fit the model to a training data of size 200. I then use stepwise model selection to eliminate some of the variables. I use these two models for prediction on the test data:

```
> pmm <- predict(mm, newdata = SA[-ii, -12])
> pselmm <- predict(selmm, newdata = SA[-ii, -12])
> print(sum(SA$ldl[-ii] - pmm)^2/length(pmm))

[1] 1.721037

> print(sum(SA$ldl[-ii] - pselmm)^2/length(pselmm))

[1] 0.5961549
```

Since this is a random exercise you may get slightly different results at home, but in general I expect the prediction error to be better for the selected model rather than the more complex one using all variables.

We will now use PC regression to analyze the data. The package `elasticnet` allows you to compute both regular and sparse PCA. I first standardize the variables, since PCA is easily dominated by the scale of individual variables otherwise.

```
> library(elasticnet)

Loaded lars 0.9-8

> SA2 <- SA
> standardize <- function(x) {
+     x <- (x - mean(x))/sd(x)
+ }
> SA2 <- apply(SA2, 2, standardize)
> SA2 <- as.data.frame(SA2)
> names(SA2) <- names(SA)
```

As you can see below, fitting the model to standardized data makes no difference in regression since this only means that the coefficients will be scaled accordinly:

```
> mm <- lm(ldl ~ age + sbp + adiposity + obesity + typea + alcohol +
+     alcind + tobacco + as.factor(tobind) + as.factor(chd) + as.factor(famhist),
+     data = SA2, subset = ii)
> print(summary(mm))

Call:
lm(formula = ldl ~ age + sbp + adiposity + obesity + typea +
    alcohol + alcind + tobacco + as.factor(tobind) + as.factor(chd) +
```

```
    as.factor(famhist), data = SA2, subset = ii)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7358 -0.5759 -0.0957  0.4221  3.2486

Coefficients:
                                    Estimate Std. Error t value Pr(>|t|)
(Intercept)                        -0.468744   0.154920  -3.026 0.002828 **
age                                -0.113570   0.093820  -1.211 0.227602
sbp                                 0.032815   0.064363   0.510 0.610759
adiposity                           0.430298   0.126047   3.414 0.000785 ***
obesity                             0.048468   0.109380   0.443 0.658190
typea                               0.015506   0.062094   0.250 0.803078
alcohol                            -0.149085   0.064863  -2.298 0.022635 *
alcind                             -0.002699   0.065613  -0.041 0.967227
tobacco                            -0.008483   0.074000  -0.115 0.908861
as.factor(tobind)0.561641714195945  0.342649   0.163282   2.099 0.037196 *
as.factor(chd)1.38193602756419      0.362432   0.148105   2.447 0.015319 *
as.factor(famhist)1.13576838927991  0.142452   0.128494   1.109 0.269005
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8516 on 188 degrees of freedom
Multiple R-squared: 0.3225,        Adjusted R-squared: 0.2829
F-statistic: 8.136 on 11 and 188 DF,  p-value: 1.427e-11
```

The `spca()` function computes K principal components. The option `para` allows you to control the sparsity. Below, I compute the ordinary PCs.

```
> newx <- spca(SA2[, -12], K = 3, para = c(0, 0, 0), type = c("predictor"))
> print(newx)

Call:
spca(x = SA2[, -12], K = 3, para = c(0, 0, 0), type = c("predictor"))

3 sparse PCs
Pct. of exp. var. : 27.5 13.5 11.3
Num. of non-zero loadings :  11 11 11
Sparse loadings
             PC1    PC2    PC3
age       -0.460  0.083 -0.189
sbp       -0.310  0.010  0.145
adiposity -0.450  0.361  0.191
obesity   -0.361  0.365  0.376
typea      0.000 -0.052  0.273
alcohol   -0.170 -0.530  0.353
alcind    -0.116 -0.525  0.433
tobacco   -0.330 -0.242 -0.420
tobind    -0.280 -0.325 -0.345
chd       -0.304 -0.070 -0.259
famhist   -0.194  0.036  0.127
```

In the new coordinate system, I fit the full model. Notice that the results in terms of R^2 is identical to the original fit, *but* which variables that are selected differs quite a lot. Notice, it is not the first PCs that are the most correlated with y necessarily as the stepwise model selection demonstrates.

```
> newx <- spca(SA2[, -12], K = 11, para = rep(0, 11), type = c("predictor"))
> SA3 <- SA2
> SA3[, -12] <- as.matrix(SA2[, -12]) %*% t(newx$load)
```

```
> names(SA3) <- c("pc1", "pc2", "pc3", "pc4", "pc5", "pc6", "pc7",
+     "pc8", "pc9", "pc10", "pc11", "ldl")
> mm <- lm(ldl ~ pc1 + pc2 + pc3 + pc4 + pc5 + pc6 + pc7 + pc8 +
+     pc9 + pc10 + pc11, data = SA3, subset = ii)
> print(summary(mm))

Call:
lm(formula = ldl ~ pc1 + pc2 + pc3 + pc4 + pc5 + pc6 + pc7 +
    pc8 + pc9 + pc10 + pc11, data = SA3, subset = ii)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7358 -0.5759 -0.0957  0.4221  3.2486

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.02207    0.06087  -0.363  0.71730
pc1          0.11670    0.07465   1.563  0.11966
pc2          0.18392    0.07331   2.509  0.01296 *
pc3          0.09603    0.07610   1.262  0.20858
pc4          0.17088    0.09162   1.865  0.06371 .
pc5          0.08590    0.10623   0.809  0.41976
pc6          0.22047    0.08455   2.608  0.00985 **
pc7          0.13369    0.08014   1.668  0.09693 .
pc8         -0.32729    0.06387  -5.124 7.38e-07 ***
pc9         -0.10389    0.06445  -1.612  0.10866
pc10        -0.04857    0.07417  -0.655  0.51337
pc11        -0.01539    0.09398  -0.164  0.87008
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8516 on 188 degrees of freedom
Multiple R-squared: 0.3225,        Adjusted R-squared: 0.2829
F-statistic: 8.136 on 11 and 188 DF,  p-value: 1.427e-11

> selmm <- step(mm, trace = F)
> print(summary(selmm))

Call:
lm(formula = ldl ~ pc2 + pc4 + pc6 + pc7 + pc8 + pc9, data = SA3,
    subset = ii)

Residuals:
    Min      1Q  Median      3Q     Max
-1.8616 -0.5766 -0.0973  0.4370  3.6279

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.01305    0.06054  -0.216   0.8296
pc2          0.17459    0.06976   2.503   0.0132 *
pc4          0.15793    0.07070   2.234   0.0266 *
pc6          0.19048    0.07394   2.576   0.0107 *
pc7          0.11713    0.07006   1.672   0.0962 .
pc8         -0.31248    0.04721  -6.619 3.48e-10 ***
pc9         -0.11635    0.05663  -2.054   0.0413 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8515 on 193 degrees of freedom
```

```
Multiple R-squared: 0.3047,        Adjusted R-squared: 0.2831
F-statistic: 14.09 on 6 and 193 DF,  p-value: 2.758e-13
```

Let's try sparse PCA instead. Below, I ask `spca` to generate new coordinate that involve fewer x-variables per component (controlled by `para`).

```
> newx <- spca(SA2[, -12], K = 11, para = rep(3, 11), type = c("predictor"))
> print(newx)

Call:
spca(x = SA2[, -12], K = 11, para = rep(3, 11), type = c("predictor"))

11 sparse PCs
Pct. of exp. var. : 16.0 13.6  8.2  9.1  8.7  8.0  7.1  7.3  5.1  4.9  1.6
Num. of non-zero loadings :  6 6 4 2 2 1 2 1 4 2 2
Sparse loadings
             PC1    PC2    PC3    PC4    PC5 PC6    PC7 PC8    PC9   PC10
age       -0.024  0.000  0.000  0.000  0.000   0  0.000   0  0.000  1.000
sbp        0.000  0.000  0.000  0.000  0.000  -1  0.000   0  0.000  0.000
adiposity -0.411  0.064  0.149  0.000  0.000   0  0.000   0  0.000  0.000
obesity   -0.830  0.134  0.313  0.000  0.000   0  0.000   0  0.000 -0.021
typea      0.000  0.000  0.000 -0.995 -0.103   0  0.000   0  0.000  0.000
alcohol   -0.093 -0.610  0.000  0.000  0.000   0 -0.411   0  0.673  0.000
alcind     0.000 -0.703  0.319  0.000  0.000   0  0.000   0 -0.639  0.000
tobacco   -0.363 -0.193 -0.882  0.000  0.000   0  0.000   0 -0.223  0.000
tobind    -0.042 -0.275  0.000  0.000  0.000   0  0.912   0  0.299  0.000
chd        0.000  0.000  0.000  0.000  0.000   0  0.000   1  0.000  0.000
famhist    0.000  0.000  0.000 -0.103  0.995   0  0.000   0  0.000  0.000
             PC11
age         0.000
sbp         0.000
adiposity -0.902
obesity     0.432
typea       0.000
alcohol     0.000
alcind      0.000
tobacco     0.000
tobind      0.000
chd         0.000
famhist     0.000
```

Which x-variables contribute to each component?

```
> SA3 <- SA2
> SA3[, -12] <- as.matrix(SA2[, -12]) %*% t(newx$load)
> names(SA3) <- c("pc1", "pc2", "pc3", "pc4", "pc5", "pc6", "pc7",
+     "pc8", "pc9", "pc10", "pc11", "ldl")
> mm <- lm(ldl ~ pc1 + pc2 + pc3 + pc4 + pc5 + pc6 + pc7 + pc8 +
+     pc9 + pc10 + pc11, data = SA3, subset = ii)
> print(summary(mm))

Call:
lm(formula = ldl ~ pc1 + pc2 + pc3 + pc4 + pc5 + pc6 + pc7 +
    pc8 + pc9 + pc10 + pc11, data = SA3, subset = ii)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7358 -0.5759 -0.0957  0.4221  3.2486

Coefficients:
```

```
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.022072   0.060869  -0.363 0.717296
pc1          0.177926   0.070451   2.526 0.012378 *
pc2          0.149085   0.064863   2.298 0.022635 *
pc3          0.047169   0.071865   0.656 0.512399
pc4          0.262358   0.113226   2.317 0.021575 *
pc5         -0.049810   0.107671  -0.463 0.644179
pc6          0.091491   0.069817   1.310 0.191643
pc7          0.017476   0.072191   0.242 0.808982
pc8         -0.380351   0.098401  -3.865 0.000153 ***
pc9          0.038246   0.065750   0.582 0.561474
pc10        -0.008483   0.074000  -0.115 0.908861
pc11         0.010445   0.065006   0.161 0.872524
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8516 on 188 degrees of freedom
Multiple R-squared: 0.3225,        Adjusted R-squared: 0.2829
F-statistic: 8.136 on 11 and 188 DF,  p-value: 1.427e-11
```

```
> selmm <- step(mm, trace = F)
> print(summary(selmm))

Call:
lm(formula = ldl ~ pc1 + pc2 + pc4 + pc8, data = SA3, subset = ii)

Residuals:
     Min       1Q   Median       3Q      Max
-1.60378 -0.54248 -0.07835  0.41333  3.13306

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.01767    0.05989  -0.295  0.76826
pc1          0.16330    0.06455   2.530  0.01220 *
pc2          0.15364    0.05709   2.691  0.00773 **
pc4          0.25372    0.08875   2.859  0.00472 **
pc8         -0.37762    0.04896  -7.713 6.16e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8435 on 195 degrees of freedom
Multiple R-squared: 0.3106,        Adjusted R-squared: 0.2964
F-statistic: 21.96 on 4 and 195 DF,  p-value: 5.604e-15
```

Try this at home: use PC regression and predict on the test data. Which degree of sparsity works best? Does this beat the prediction performance of the stepwise selection model in the original coordinate system?

1.2 Ridge regression

We have that

$$V(\hat{\beta}) = \sigma^2(X'X)^{-1} = \sigma^2 V D^{-2} V' = \sigma^2 \sum_{k=1}^{p} d_k^{-2} v_k v_k'.$$

The above expression is called the spectral decomposition of the estimation covariance matrix. If we look at the total $MSE(\hat{\beta})$ for all coefficients we have (since $\hat{\beta}$ are unbiased)

$$MSE(\hat{\beta}) = Trace(V(\hat{\beta})) = \sigma^2 Trace((X'X)^{-1}) = \sigma^2 \sum_{k=1}^{p} d_k^{-2}.$$

This total MSE is large if any of the eigen values of $X'X$, d_k^2s are small. When does this happen? Whenever there are x's that are correlated so that the information in X is redundant. In the figure above, you see that the second direction V_2 has a small eigen value d_2^2.

In least squares estimation we have that

$$E[\hat{\beta}] = \beta, \quad \text{and} \quad V(\hat{\beta}) = \sigma^2(X'X)^{-1}.$$

The question we now ask if we can reduce the variance by allowing for some bias, and then obtain a lower total MSE (since MSE is bias-squared + variance). The source of the high variance is the numerical instability of the inverse operation on $X'X$. We stabilize this by adding something to the diagonal of $X'X$ before taking the inverse (you probably recognize this "fix" from linear algebra). The resulting model is called *ridge regression*.

$$\hat{\beta}_R = (X'X + rI)^{-1} X'y.$$

Above, I have added the constant r to the diagonal element of $X'X$. What are the properties of $\hat{\beta}_R$?

$$E[\hat{\beta}_R] = (X'X + rI)^{-1} X'X\beta = (X'X + rI)^{-1}(X'X + rI - rI)\beta = \beta - r(X'X + rI)^{-1}\beta.$$

The bias of this estimator is thus $r(X'X + rI)^{-1}\beta$, which depends on β and is thus larger for large β. The bias is controlled by r and is 0 when r is 0. The estimation variance is

$$V(\hat{\beta}_R) = \sigma^2(X'X + rI)^{-1} X'X (X'X + rI)^{-1},$$

which decreases with r. Put this all together and we have

$$MSE(\hat{\beta}_R) = Trace(V(D^2 + rI)^{-1} V'[\sigma^2 V D^2 V' + r^2 \beta\beta'] V(D^2 + rI)^{-1} V') =$$

$$= Trace(V(D^2 + rI)^{-1}[\sigma^2 D^2 + r^2 \tilde{\beta}\tilde{\beta}'](D^2 + rI)^{-1} V') = \sum_{k=1}^{p} \frac{\sigma^2 d_k^2 + r^2 \tilde{\beta}_k^2}{(d_k^2 + r)^2} \leq \sum_{k=1}^{p} \sigma^2 d_k \text{ for some } r,$$

where $\tilde{\beta} = V\beta$.

In Figure 2 we see an example of the MSE dependency on r. For some sample sizes, true β and choices of r, the ridge estimator can have smaller MSE than the LS estimator. It can be especially beneficial for small n and high collinearity. You can choose r via cross-validation.

1.3 Shrinkage estimators

Another way of looking at this is to approach the regularization of the inverse $X'X$ from a model penalty. First, let's try to figure out what r is actually doing to the fit. We start with the special case when x's are uncorrelated. Then, $X'X = D^2$ and we have

$$(X'X + rI) = (D^2 + rI) = diag(d_1^2 + r, d_2^2 + r, \cdots, d_p^2 + r)$$

and you since $\hat{\beta}_R = (X'X + rI)^{-1} X'y$ you see that r has the largest impact on the coefficients j whose x-variables have small spread d_j. Essentially, the coefficient estimate $\hat{\beta}_j$ you would get from LS is shrunk by a factor that depends on r. We can write general shrinkage estimators as

$$\hat{\beta}_S = \frac{\hat{\beta}_{LS}}{1 + c},$$

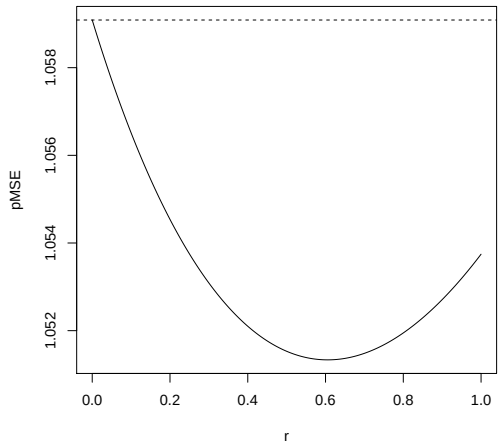

Figure 2: MSE of ridge estimator (solid) as a function of r, with MSE of least squares as dotted horizontal line.

where $c > 0$. The bias of $\hat{\beta}_S$ is $c\beta/(1+c)$ and variance $V(\hat{\beta}_{LS})/(1+c)^2$, which is less that the estimation variance of the LS estimator.
If the xs are correlated, then

$$X'X + rI = VD^2V' + rVV' = V(D^2 + rI)V'$$

and now the shrinkage factor has the largest effect in the PC directions k with the smallest d_k^2.

The idea behind shrinkage is that by not allowing $\hat{\beta}_S$ to get too big, we cover ourselves from risk of poor estimates having a huge impact on prediction. We just make every estimate smaller, which increases the bias but suppresses the variance. With appropriate choices of shrinkage, the pMSE may be reduced compared with ordinary LS.

1.4 Penalized regression

The least squares criterion is written as

$$\min_{\beta}(y - X\beta)'(y - X\beta) = ||y - X\beta||^2 = \sum_{i=1}^{n}(y_i - x_i\beta)^2.$$

The penalized least squares problem related to ridge regression can be written as

$$\min_{\beta}(y - X\beta)'(y - X\beta) \text{ subject to} ||\beta||^2 \leq \tau.$$

The constraint $||\beta||^2 = \beta'\beta = \sum_{j=1}^{p}\beta_j^2$. What the problem states is that we want to minimize LS *but* we don't want the sum of the estimates coefficients (squared) to be too large, i.e. we penalize the magnitudes of the $\hat{\beta}$s. This constraint can be achieved if all βs are small or perhaps if one is big but the rest are very small, and of course anything in between these extremes.

How do we solve such constrained optimization problems? We use *Lagrangian methods*, replacing the constraint with an added penalty

$$\min_{\beta}(y - X\beta)'(y - X\beta) + \lambda\beta'\beta.$$

We solve this penalized LS problem for different values of λ until the constraint $\beta'\beta \leq \tau$ is fulfilled. If $\lambda = 0$, we solve the LS problem. If the constraint is fulfilled for this solution we are done. If not, we increase λ and solve again. If the constraint is fulfilled we're done, otherwise we increase λ again. The

optimal solution to the constrained optimization problem is the one with the smallest λ that leads to a solution that satisfies the constraint.

How do we solve the penalized LS problem? Let's take the derivative with respect to each β_j and set it to 0. The derivative looks like

$$-2X'(y - X\beta) + 2\lambda\beta = 0$$

which we can solve for β:

$$\hat{\beta} = (X'X + \lambda I)^{-1}X'y = \hat{\beta}_R.$$

So, the ridge estimator is actually identical to the penalized regression estimate with a constraint on the sum of squares of the regression coefficients.

We can graphically represent what this constrained optimization routine does: In Figure 3 I illus-

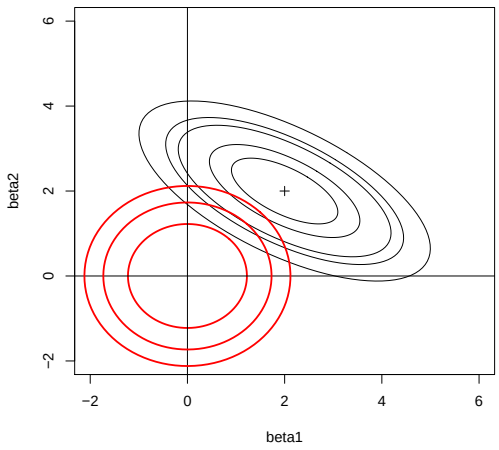

Figure 3: Ridge regression penalty (red circles) and likelihood contour (elipsoid)

trate this for a problem with $p = 2$ parameters. I depict the penalty function in red for various values of τ and compare with the likelihood (RSS) contours as a function of different values of β_1 and β_2, where $RSS(\beta_1, \beta_2) = \sum_i (y_i - \beta_1 x_{i1} - \beta_2 x_{i2})^2$. Note that to be inside the red constraint region, for some values of τ, the LS solution is not allowed and is instead shrunk toward the origin. Look for the values on the red circle that's the deepest inside the RSS contour plot - those are the regularized solutions.

Having formulated the penalized regression problem as above, we see that it is possible to explore other type of constraints. This is currently a 'hot topic' in statistical research. The LASSO estimator uses the constraint shown in Figure 4 which you can see is diamond shaped. Mathematically, we can write this penalized regression problem as

$$\min_{\beta}(y - X\beta)'(y - X\beta) + \lambda \sum_{j=1}^{p} |\beta_j|,$$

i.e. we penalize the sum of absolute values of the coefficients. The impact this type of penalty is seen in Figure 4. For some τ, it is possible that the solution inside the constraint region (diamond) that's the deepest inside the RSS contour (i.e. fits the data the best subject to satisfying the constraint) occurs at one point of the diamond. In that case, some of the β's are 0. In the above example, for the second largest diamond the solution occurs for $\beta_1 = 0$ so variable 1 is dropped from the model.

Of course, one can try any type of constraint region to encourage model fits of a certain kind. Below, I use a square constraint region. In Figure 5 I illustrate a square constraint region. This will encourage the βs to be equal.

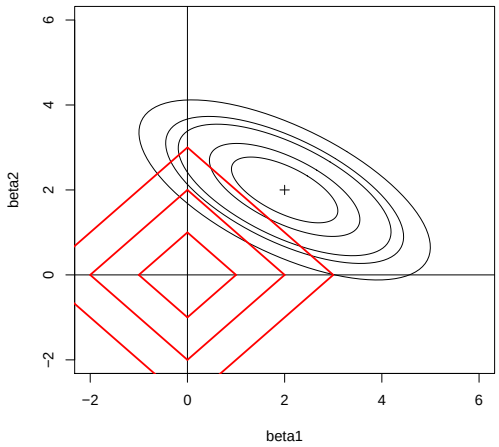

Figure 4: Lasso regression penalty (red circles) and likelihood contour (elipsoid)

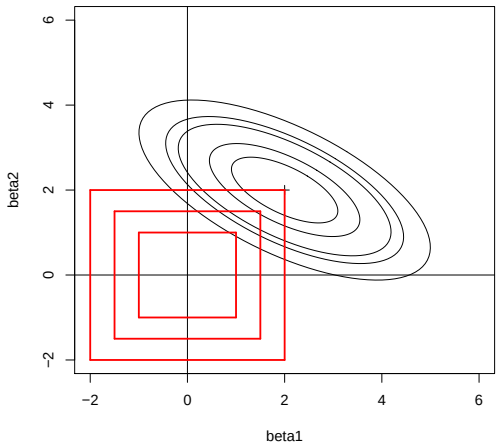

Figure 5: Oscar regression penalty (red circles) and likelihood contour (elipsoid)

In Figure 6 I illustrate what the regularization does to the regression coefficient estimate. I plot the regularized estimate as a function of the LS estimate (which is indicated by the black 45degree line in the plot). Ridge-regression is the red line, which as you see has a slope less than 1. That means, each ridge coefficient is a reduced LS coefficient by a shrinkage factor. For large β we get a large bias (gap between the black and red curves) and no β is ever set to 0. The green curve is the LASSO estimator. As you see, some βs are just set to 0 whereas others are shrunk by a constant factor by deducting a value from the estimate. Thus, the bias is constant and not larger for large β which is a huge benefit, and since some β are set to 0 we get both reduced estimation variance *and* variable selection. This is why the LASSO models have become so popular.

2 Demo 15

We try regularized regression on the heart disease data. The package `lars` include the LASSO models.

```
> library(lars)
> xx <- as.matrix(cbind(rep(1, dim(SA)[1]), as.matrix(SA[, -12])))
> colnames(xx) <- c("int", names(SA)[-12])
```

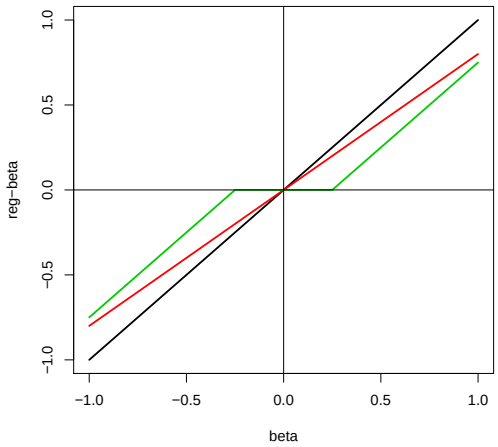

Figure 6: Regularized estimates of β as a function of the LS estimator. Red is ridge, green is lasso.

```
> yy <- SA[, 12]
> ll <- lars(xx[ii, ], yy[ii], intercept = F)
> print(summary(ll))

LARS/LASSO
Call: lars(x = xx[ii, ], y = yy[ii], intercept = F)
   Df    Rss        Cp
0   0 5120.7 1537.2339
1   1 4228.0 1236.3665
2   2 1034.2  154.8502
3   3  850.1   94.4032
4   4  767.6   68.4159
5   5  692.7   45.0121
6   6  691.8   46.6825
7   7  606.3   19.6774
8   8  584.7   14.3801
9   9  559.8    7.9224
10 10  559.1    9.6657
11 11  554.2   10.0169
12 12  554.2   12.0000

> print(ll)

Call:
lars(x = xx[ii, ], y = yy[ii], intercept = F)
R-squared: 0.892
Sequence of LASSO moves:
     adiposity obesity sbp tobind int chd typea famhist alcohol tobacco age
Var          4       5   3     10   1  11     6      12       7       9   2
Step         1       2   3      4   5   6     7       8       9      10  11
     alcind
Var       8
Step     12
```

As you can see, the lars output tells you which variables are added to the model as the size of the constraint region is increased (i.e. the constraint is relaxed by reducing the lagrange multiplier λ above). Here, the first variable to enter is adiposity. The summary function gives you the RSS for each λ corresponding to a constraint region where one more variable is added to the model. For each such model

you also get the Mallow's Cp, which can be used for model selection. Pick the λ that minimizes the Cp. This may correspond to a model where some βs are set to 0.

In Figure 7, I use the lasso plot provided with the lars package. To read this plot, you read the x-axis from left to right. This corresponds to the volume of the constraint region. To the left, we start with the diamond having volume 0, which sets all β to 0. As we go to the right, we increase the volume of the diamond, and more and more variables are allowed to enter the model. Values for λ corresponding to a new variable entering the model are represented by vertical lines. Look for model separated by a large gap between such lines. This indicates a region of stability in terms of model selection, i.e. you need to increase the volume of the diamond a lot for the next variable to enter. Such gaps are candidates for selected models for prediction.

```
> plot(ll)
```

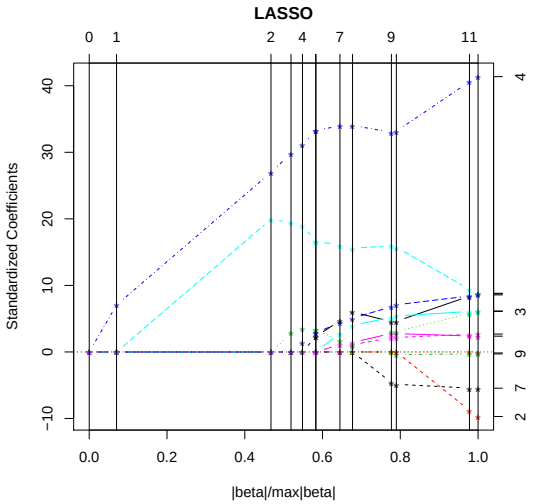

Figure 7: LARS trace for heart disease data.

We can now use the selected models for prediction. Below, I use the command `which.min()` to identify the β solution corresponding to the diamond constraint that results in the minimum Cp. I then use this to predict the test data. I then compute the prediction error of the model with no shrinkage (LS solution) and the selected model. In general, using regularization provides better predictions.

```
> selll <- ll$beta[which.min(ll$Cp), ]
> pll <- xx[-ii, ] %*% selll
> print(sum(SA$ldl[-ii] - xx[-ii, ] %*% ll$beta[dim(ll$beta)[1],
+     ])^2/length(pll))

[1] 1.721037

> print(sum(SA$ldl[-ii] - pll)^2/length(pll))

[1] 1.256441
```

Recently, Zou, Hastie and Tibshirani (2004) `http://http://www.stanford.edu/ hastie/Papers/elasticnet.pdf` proposed the *elastic net* which is implemented in an R package. This method uses a combination of the both the ridge and lasso penalties

$$\min_{\beta}(y - X\beta)'(y - X\beta) + \lambda_2 \sum_{j=1}^{p} \beta_j^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j|.$$

This helps when x's are competing to be in the model. LASSO tends to pick just one of two correlated x-variables to be in the model, and it is often quite by chance that one gets selected over another. By adding the ridge-penalty to the lasso problem we also encourage correlated x's to enter together (and at the same time) into the model. Of course, we have to pick the λ_1 and λ_2 carefully to balance the two constraints. In the elastic net package you only have to pick λ_2 as the function examines many values of λ_1 automatically.

```
> library(elasticnet)
> xx <- as.matrix(cbind(rep(1, dim(SA)[1]), as.matrix(SA[, -12])))
> colnames(xx) <- c("int", names(SA)[-12])
> yy <- SA[, 12]
> ll <- enet(xx[ii, ], yy[ii], intercept = F)
> print(summary(ll))

          Length Class  Mode
call        4    -none- call
actions    13    -none- list
allset     12    -none- numeric
beta.pure 156    -none- numeric
vn         12    -none- character
mu          1    -none- numeric
normx      12    -none- numeric
meanx      12    -none- numeric
lambda      1    -none- numeric
L1norm     13    -none- numeric
penalty    13    -none- numeric
df         13    -none- numeric
Cp         13    -none- numeric
sigma2      1    -none- numeric

> print(ll)

Call:
enet(x = xx[ii, ], y = yy[ii], intercept = F)
Cp statistics of the Lasso fit
Cp: 1529.993 1230.737  154.984   94.869   69.031   45.762   47.434   20.583   15.325    8.912   10.65
DF:  1  2  3  4  5  6  7  8  9 10 11 12 13
Sequence of  moves:
     adiposity obesity sbp tobind int chd typea famhist alcohol tobacco age
Var          4       5   3     10   1  11     6      12       7       9   2
Step         1       2   3      4   5   6     7       8       9      10  11
     alcind
Var       8 13
Step     12 13

> plot(ll)
```

In Figure 8 I show an example of a the elastic net applied to the heart disease data. In the above code I used the default setting of `enet()` where $\lambda_2 = 0$ so the solution is identical to the LASSO.

```
> selll <- ll$beta[which.min(ll$Cp), ]
> pll <- xx[-ii, ] %*% selll
> print(sum(SA$ldl[-ii] - xx[-ii, ] %*% ll$beta[dim(ll$beta)[1],
+     ])^2/length(pll))

[1] 1.721037

> print(sum(SA$ldl[-ii] - pll)^2/length(pll))

[1] 1.256441
```

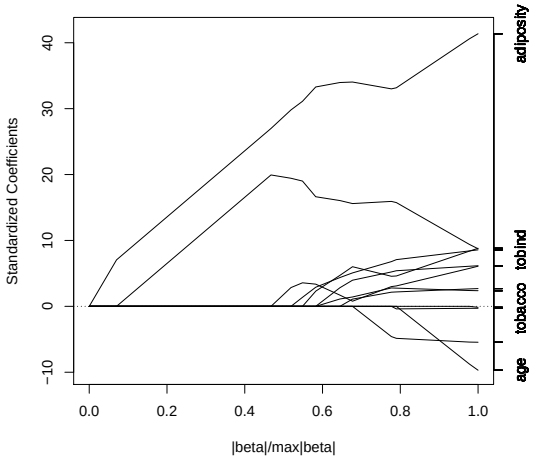

Figure 8: Elastic net for heart disease data.

Below, I now add the ridge-penalty.

```
> ll <- enet(xx[ii, ], yy[ii], intercept = F, lambda = 0.4)
> print(summary(ll))

          Length Class  Mode
call        5    -none- call
actions    13    -none- list
allset     12    -none- numeric
beta.pure 156    -none- numeric
vn         12    -none- character
mu          1    -none- numeric
normx      12    -none- numeric
meanx      12    -none- numeric
lambda      1    -none- numeric
L1norm     13    -none- numeric
penalty    13    -none- numeric
df         13    -none- numeric
Cp         13    -none- numeric
sigma2      1    -none- numeric

> print(ll)

Call:
enet(x = xx[ii, ], y = yy[ii], lambda = 0.4, intercept = F)
Sequence of  moves:
     adiposity obesity sbp int age typea tobind chd famhist alcind tobacco
Var          4       5   3   1   2     6     10  11      12      8       9
Step         1       2   3   4   5     6      7   8       9     10      11
     alcohol
Var        7 13
Step      12 13

> plot(ll)
```

From Figure 9 you can see that some of the "paths" (the trajectories of the coefficients as a function of the constraint volume on the x-axis) are more parallel than for the LASSO solution and correlated variables tend to enter the model at the same time (adiposity and obesity). Try this at home for other values of λ_2 and check with the code below if you can beat LASSO in terms of prediction performance.

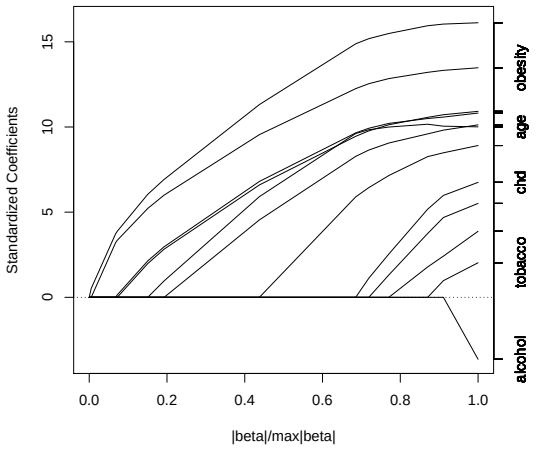

Figure 9: Elastic net for heart disease data

```
> selll <- ll$beta[which.min(ll$Cp), ]
> pll <- xx[-ii, ] %*% selll
> print(sum(SA$ldl[-ii] - xx[-ii, ] %*% ll$beta[dim(ll$beta)[1],
+     ])^2/length(pll))

[1] 261.7089

> print(sum(SA$ldl[-ii] - pll)^2/length(pll))

[1] 1.277788
```

2.1 Caveats

Remember, when you do this at home you may get different results since the training and test data are selected randomly. Also note, to set up CIs for LASSO coefficient estimates you have to use bootstrap since we don't have a closed form solution for the standard errors.