Reducing the dimensionality of X'X by different methods

Stefan

September 23, 2016

Multiple linear regression

linear model: $X\beta = y$

first column of X is ones, the other columns of X are forecasts produced by different models, y is the observation

OLS: minimise $||X\beta - y||^2$

OLS estimator: $\hat{\beta} = (X'X)^{-1}X'y$

if all X are independent, $(X'X)^{-1}$ is diagonal, and the elements of $\hat{\beta}$ are equal to the **correlations** between the columns of X and y.

but in general the forecasts are correlated with one another, **multi-collinearity**

as a result, a predictor can be positively correlated with y, but receive a negative weight

also, the estimated parameters have **high variance**

related is this interesting point from wikipedia entry on **Tikhonov regularization**: $X\beta$ is a like low-pass filter, filtering out the variances within the elements of X, producing a weighted average; therefore, the solution to the inverse problem acts as a high-pass filter, amplifying noise

Example:

```
load("../../data/enso-nao.Rdata")
X = cbind(1, enso[, -1])
y = enso[, 1, drop = FALSE]
# nao[, -1] = nao[, -1] / 1000 X = cbind(1, nao[,-1]) y = nao[, 1,
# drop=FALSE]
```

```
# X = cbind( 1,
# rowMeans(read.table('../../data/CMC1-CanCM3-NAO-hind-ens-members-ic-Nov-val-DJF-1982-20
# rowMeans(read.table('../../data/CMC2-CanCM4-NAO-hind-ens-members-ic-Nov-val-DJF-1982-20
# rowMeans(read.table('../../data/ECMWFS4-NAO-hind-ens-members-ic-Nov-val-DJF-1982-2010.t
# #rowMeans(read.table('../../data/CFS2-NAO-hind-ens-members-ic-Nov-val-DJF-1982-2010.txt
# rowMeans(read.table('../../data/MFS3-NAO-hind-ens-members-ic-Nov-val-DJF-1982-2010.txt
# ) y =
```

```r
# as.matrix(read.table('../../data/ERAINT-NAO-DJF-1982-2010.txt'))/1000
# i.nna = !is.na(rowSums(cbind(X,y))) X = X[i.nna, ] y = y[i.nna, ,
# drop=FALSE]
```

The correlations matrix is

```r
cor(cbind(y, X[, -1]))
```

```
##         obs    cfs    cmc   gfdl     mf   nasa     ec
## obs  1.0000 0.8140 0.8948 0.8493 0.8732 0.8805 0.9119
## cfs  0.8140 1.0000 0.7808 0.8932 0.9284 0.8889 0.8575
## cmc  0.8948 0.7808 1.0000 0.8165 0.8250 0.9035 0.9114
## gfdl 0.8493 0.8932 0.8165 1.0000 0.8680 0.9286 0.9160
## mf   0.8732 0.9284 0.8250 0.8680 1.0000 0.8991 0.8975
## nasa 0.8805 0.8889 0.9035 0.9286 0.8991 1.0000 0.9391
## ec   0.9119 0.8575 0.9114 0.9160 0.8975 0.9391 1.0000
```

Here are the OLS regression coeficients compared to the correlations

```r
cbind(beta = solve(crossprod(X)) %*% crossprod(X, y), cor = cor(X, y))
```

```
## Warning: the standard deviation is zero
```

```
##          obs    obs
##      -4.0686     NA
## cfs  -0.1044 0.8140
## cmc   0.3341 0.8948
## gfdl  0.1674 0.8493
## mf    0.6636 0.8732
## nasa -0.1630 0.8805
## ec    0.2754 0.9119
```

Simple linear regression on the multi-model ensemble mean

$y = (XM)\beta$ where M is the $((m+1) \times 2)$ matrix that transforms the row vector $(1, x_1, ..., x_m)$ into the row vector $(1, \sum x_i/m)$, i.e. calculates the multimodel ensemble mean

```r
m = ncol(X) - 1
M = rbind(c(1, 0), cbind(rep(0, m), rep(1/m, m)))
print(M)
```

```
##      [,1]   [,2]
## [1,]    1 0.0000
## [2,]    0 0.1667
## [3,]    0 0.1667
## [4,]    0 0.1667
## [5,]    0 0.1667
## [6,]    0 0.1667
## [7,]    0 0.1667
```

OLS estimator: $\hat{\beta} = ((XM)'(XM))^{-1}(XM)'y = (M'X'XM)^{-1}M'X'y$

```
cbind(beta = solve(crossprod(X %*% M)) %*% crossprod(X %*% M, y), cor = cor(X %*%
    M, y))
```

```
## Warning: the standard deviation is zero
```

```
##         obs    obs
## [1,] 2.3245     NA
## [2,] 0.9529 0.9166
```

Leave-one-out cross-validation

We want to check how well the two approaches perform at predicting the observation out-of-sample

```
N = length(y)
err_mlr = err_slr = err_clim = rep(NA_real_, N)
for (i in 1:N) {
    X_ = X[-i, ]
    y_ = y[-i, , drop = FALSE]
    beta_1 = solve(crossprod(X_)) %*% crossprod(X_, y_)
    beta_2 = solve(crossprod(X_ %*% M)) %*% crossprod(X_ %*% M, y_)
    y1 = sum(beta_1 * X[i, ])
    y2 = sum(beta_2 * (X %*% M)[i, ])
    err_mlr[i] = (y1 - y[i])^2
    err_slr[i] = (y2 - y[i])^2
    err_clim[i] = (mean(y_) - y[i])^2
}
cbind(MLR = mean(err_mlr), SLR = mean(err_slr), CLIM = mean(err_clim))
```

```
##         MLR    SLR  CLIM
## [1,] 0.3299 0.2795 1.621
```

Simple linear regression performs better than multiple linear regression out-of-sample.

How can this be explained?

Tikhonov regularisation

penalised least-squares: minimise $||X\beta - y||^2 + ||\Lambda\beta||^2 = (X\beta - y)'(X\beta - y) + \beta'\Lambda'\Lambda\beta$

Λ is called the Tikhonov matrix

The penalised least squares estimator of β is $\hat{\beta} = (X'X + \Lambda'\Lambda)^{-1}X'y$

in ridge regression, Λ is a multiple of the identity matrix, leading to shrinkage of $\hat{\beta}$ towards zero (because the norm of $\Lambda\beta$ can be interpreted as a penality on the magnitude of β)

We want the elements of β to be more similar to each other. Can we choose Λ so that it penalises the "roughness" of β? Yes we can.

$var(\beta) = 1/m\sum\beta_i^2 - (1/m\sum\beta_i)^2 = 1/m\beta'\beta - 1/m^2\beta'1_m'1_m\beta = \beta'(1/mI - 1/m^2 1_m'1_m)\beta$

where I is the identitiy matrix and 1_m is a $(m \times 1)$ matrix with all elements equal to 1.

So we choose $\Lambda'\Lambda = k(1/mI - 1/m^2 1_m'1_m)$, where k is the penalty parameter. We do not want to penalise the intercept, so we add a row and column of zeros

```
LtL = diag(1/m, nrow = m) - matrix(1/m^2, m, m)
LtL = rbind(0, cbind(0, LtL))
print(LtL)
```

```
##      [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## [1,]    0  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
## [2,]    0  0.13889 -0.02778 -0.02778 -0.02778 -0.02778 -0.02778
## [3,]    0 -0.02778  0.13889 -0.02778 -0.02778 -0.02778 -0.02778
## [4,]    0 -0.02778 -0.02778  0.13889 -0.02778 -0.02778 -0.02778
## [5,]    0 -0.02778 -0.02778 -0.02778  0.13889 -0.02778 -0.02778
## [6,]    0 -0.02778 -0.02778 -0.02778 -0.02778  0.13889 -0.02778
## [7,]    0 -0.02778 -0.02778 -0.02778 -0.02778 -0.02778  0.13889
```

Test if $\beta\Lambda\Lambda'\beta'$ is really equal to the variance of the last m elements of the vector β:

```
beta_test = runif(m + 1)
print(c(beta_test %*% LtL %*% beta_test, mean(beta_test[-1]^2) - mean(beta_test[-1])^2))
```

```
## [1] 0.07578 0.07578
```

Now we calculate the OLS estimator of β for the full multi-model ensemble and for the multi-model ensemble mean.

```
# the ordinary least squares estimator (unequal weighting)
beta_ols = solve(crossprod(X)) %*% crossprod(X, y)
# the ordinary least squares estimator for the multi-model ensemble mean
# (equal weighting)
beta_ols_mmm = solve(crossprod(X %*% M)) %*% crossprod(X %*% M, y)
```

```r
beta_ols_mmm = c(beta_ols_mmm[1], rep(beta_ols_mmm[2]/m, m))

plot(beta_ols[-1], type = "l", lwd = 7, col = "gray", lty = 2, ylab = "model weight",
    xlab = "model")
lines(beta_ols_mmm[-1], lty = 1, col = "gray", lwd = 7)

# the penalised least squares estimator
beta_pls = function(k) {
    res = solve(crossprod(X) + k * LtL) %*% crossprod(X, y)
    return(res[-1])
}
k_vec = c(0, 1, 10, 100, 1000)
lty = 0
for (k in k_vec) {
    lty = lty + 1
    lines(beta_pls(k), lty = lty)
}
legend("topleft", c("OLS (unequal)", "OLS (equal)", paste("PLS k =", k_vec)),
    lty = c(2, 1, 1:length(k_vec)), col = c(rep("gray", 2), rep("black", length(k_vec))),
    lwd = c(rep(3, 2), rep(1, length(k_vec))))
```

We see that for small regularisation constants k, the PLS estimator is equal to the OLS estimator with unequal weighting, and for large k, the PLS estimator is equal to the OLS estimator with equal weighting.

The leave-one-out error and the hat matrix

Call $\hat{y}_i = x_i'\hat{\beta}$ the fitted value at instance i

Call $\hat{\beta}_{-i}$ the estimate of β fitted to the data with the ith instance left-out

Call $\hat{y}_{i,-i} = x_i'\hat{\beta}_{-i}$ the predicted value of y_i using the parameters that were fitted without using y_i and x_i

Define the hat matrix by $\hat{y} = Hy$, i.e. $H = X(X'X + \Lambda'\Lambda)^{-1}X'$, and define by $h_i = H_{i,i}$ the ith diagonal element of the hat matrix

Then it can be shown that the leave-one-out cross validation error is given by

$$\sum_{i=1}^{N}(\hat{y}_{i,-i} - y_i)^2 = \sum_{i=1}^{N}\frac{(\hat{y}_i - y_i)^2}{(1-h_i)^2}$$

The proof uses $\hat{y}_{i,-i} = x_i'(X'X - x_ix_i' + \Lambda'\Lambda)^{-1}(X'y - x_iy)$, $h_i = x_i'(X'X + \Lambda'\Lambda)^{-1}x_i$, and $(X'X - x_ix_i' + \Lambda'\Lambda)^{-1} = (X'X + \Lambda'\Lambda)^{-1} + (X'X + \Lambda'\Lambda)^{-1}\frac{x_ix_i'}{1-h_i}(X'X + \Lambda'\Lambda)^{-1}$ (by the Woodbury formula). The rest is algebra.

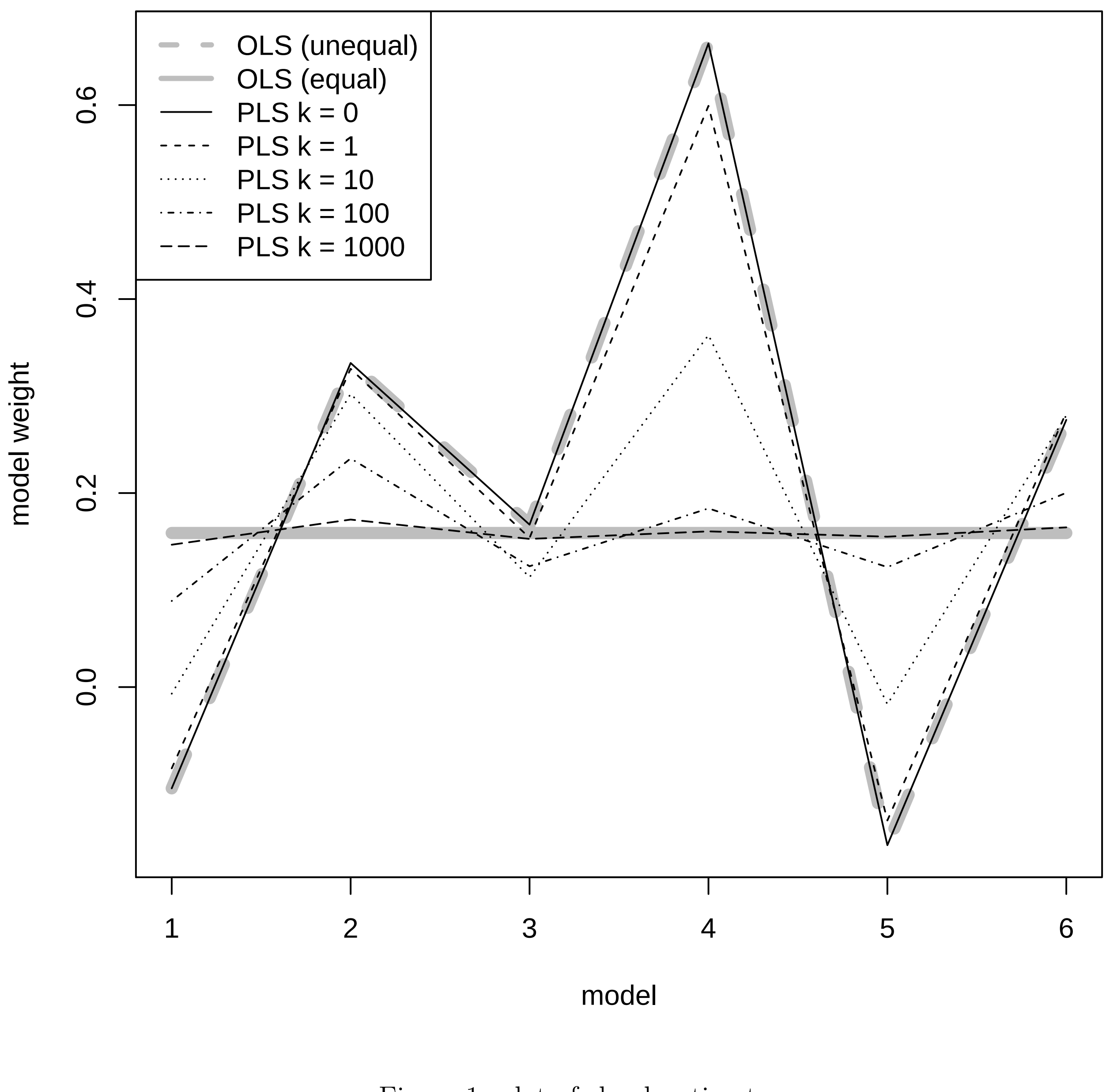

Figure 1: plot of chunk estimators

Can we improve upon equal weighting?

We repeat the leave-one-out crossvalidation for different values of k. Is there an intermediate value of k that yields a better combined forecast than the equally weighted forecasts? We calculate the leave-one-out error by brute force, repeatedly fitting the model to data with one data point left out, and also with the analytical expression.

```
beta_pls = function(kLtL, X, y) {
    beta = solve(crossprod(X) + kLtL) %*% crossprod(X, y)
    return(beta)
}
loocv_err = function(kLtL, X, y) {
    H = X %*% solve(crossprod(X) + kLtL) %*% t(X)
    yhat = H %*% y
    err = sum((y - yhat)^2/(1 - diag(H))^2)
    return(err)
}

err_pls = c()
err_pls_ana = c()

k_vec = seq(0, 1000, 10)
for (k in k_vec) {
    kLtL = k * LtL
    err_ = 0
    for (i in 1:N) {
        X_ = X[-i, ]
        y_ = y[-i, , drop = FALSE]
        beta = beta_pls(kLtL, X_, y_)
        y_pred = sum(beta * X[i, ])
        err_ = err_ + (y_pred - y[i])^2
    }
    err_pls = c(err_pls, err_/N)
    err_pls_ana = c(err_pls_ana, loocv_err(kLtL, X, y)/N)
}
plot(k_vec, err_pls)
lines(k_vec, err_pls_ana)
```

There is indeed an intermediate value of k for which the leave-one-out error with unequal weighting is smaller than the leave-one-out error with equal weighting.

This is good.

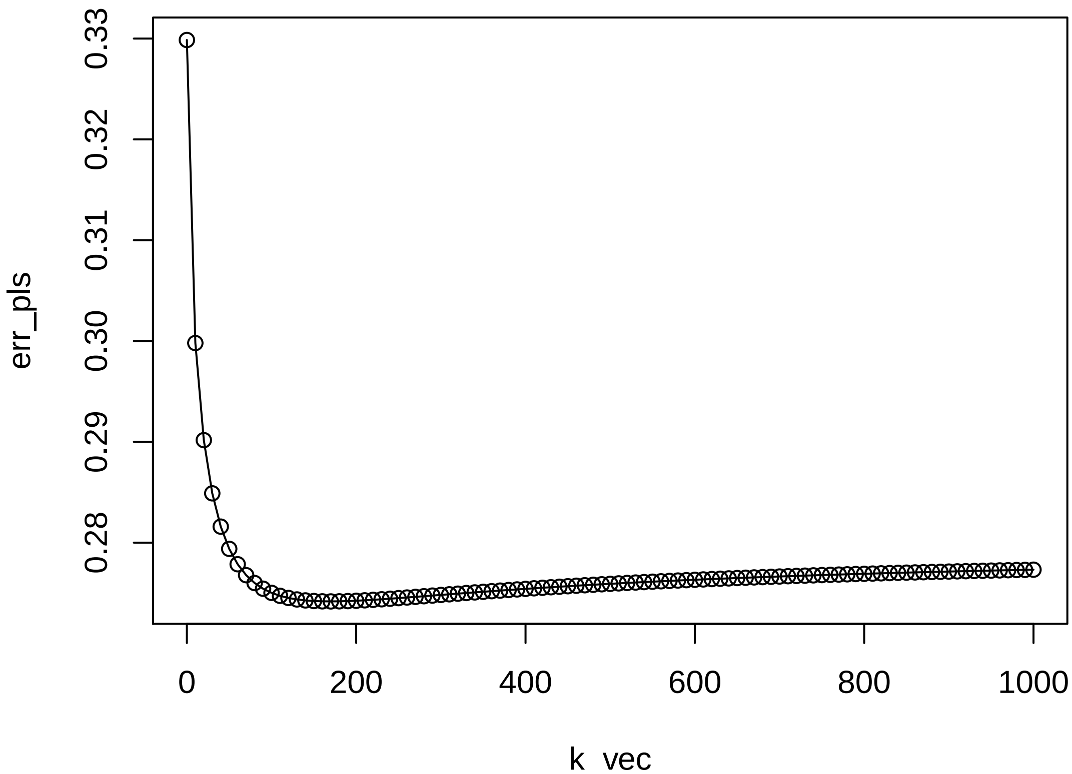

Figure 2: empirically and analytically calculated cross validation error, as function of the penality parameter

Choosing the optimal k

The regularisation parameter k is unknown in practice, and somehow has to be estimated from the data.

A common method is to choose the value of k that minimises the LOOCV error.

So we can use numerical optimisation and the analytical expression of the LOOCV error to select the best k:

```
loocv_err = function(k, LtL, X, y) {
    H = X %*% solve(crossprod(X) + k * LtL) %*% t(X)
    yhat = H %*% y
    err = sum((y - yhat)^2/(1 - diag(H))^2)
    return(err)
}
optim(par = 0, fn = loocv_err, method = "BFGS", LtL = LtL, X = X, y = y)
```

```
## $par
## [1] 162.3
##
## $value
## [1] 7.951
##
## $counts
## function gradient
##       17       16
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Putting it all together

```
beta_pls = function(X, y, k = NULL) {
    p = ncol(X) - 1
    LtL = diag(1/p, nrow = p) - tcrossprod(rep(1/p, p))
    LtL = rbind(0, cbind(0, LtL))  # intercept is not penalised

    if (is.null(k)) {
        loocv_err_ = function(k) {
            H = X %*% solve(crossprod(X) + k * LtL) %*% t(X)
            yhat = H %*% y
```

```
            err = sum((y - yhat)^2/(1 - diag(H))^2)
            return(err)
        }
        loocv_err_gr = function(k) {
            H = X %*% solve(crossprod(X) + k * LtL) %*% t(X)
            yhat = H %*% y
            h = diag(H)
            D = -X %*% solve(crossprod(X) + k * LtL) %*% LtL %*% solve(crossprod(X) +
                k * LtL) %*% t(X)
            d = diag(D)
            err = 2 * sum((y - yhat)^2 * (1 - h)^(-3) * d)
            return(err)
        }
        opt = optim(par = 0, fn = loocv_err_, gr = loocv_err_gr, method = "BFGS")
        k = opt$par
    }

    beta = solve(crossprod(X) + k * LtL) %*% crossprod(X, y)
    return(beta)
}
print(cbind(beta_ols, beta_ols_mmm, beta_pls(X, y)))
```

```
##          obs beta_ols_mmm    obs
##      -4.0686       2.3245 2.2797
## cfs  -0.1044       0.1588 0.1090
## cmc   0.3341       0.1588 0.2147
## gfdl  0.1674       0.1588 0.1340
## mf    0.6636       0.1588 0.1720
## nasa -0.1630       0.1588 0.1379
## ec    0.2754       0.1588 0.1860
```

Full cross validation

```
N = length(y)
err_mlr = err_slr = err_clim = err_plr = rep(NA_real_, N)
for (i in 1:N) {
    X_ = X[-i, ]
    y_ = y[-i, , drop = FALSE]
    beta_1 = solve(crossprod(X_)) %*% crossprod(X_, y_)
    beta_2 = solve(crossprod(X_ %*% M)) %*% crossprod(X_ %*% M, y_)
    beta_3 = beta_pls(X_, y_, k = 500)
    y1 = sum(beta_1 * X[i, ])
    y2 = sum(beta_2 * (X %*% M)[i, ])
```

```r
    y3 = sum(beta_3 * X[i, ])
    err_mlr[i] = (y1 - y[i])^2
    err_slr[i] = (y2 - y[i])^2
    err_clim[i] = (mean(y_) - y[i])^2
    err_plr[i] = (y3 - y[i])^2
}
print(colMeans(cbind(err_clim, err_mlr, err_slr, err_plr)))
```

```
## err_clim  err_mlr  err_slr  err_plr
##   1.6211   0.3299   0.2795   0.2759
```

TODO

- show that regression of the MMM is equivalent to equal-weights regression
- why does the penalised estimator converge to the equal-weights estimator for $k \to \infty$