Collinearity

Milica Cudina

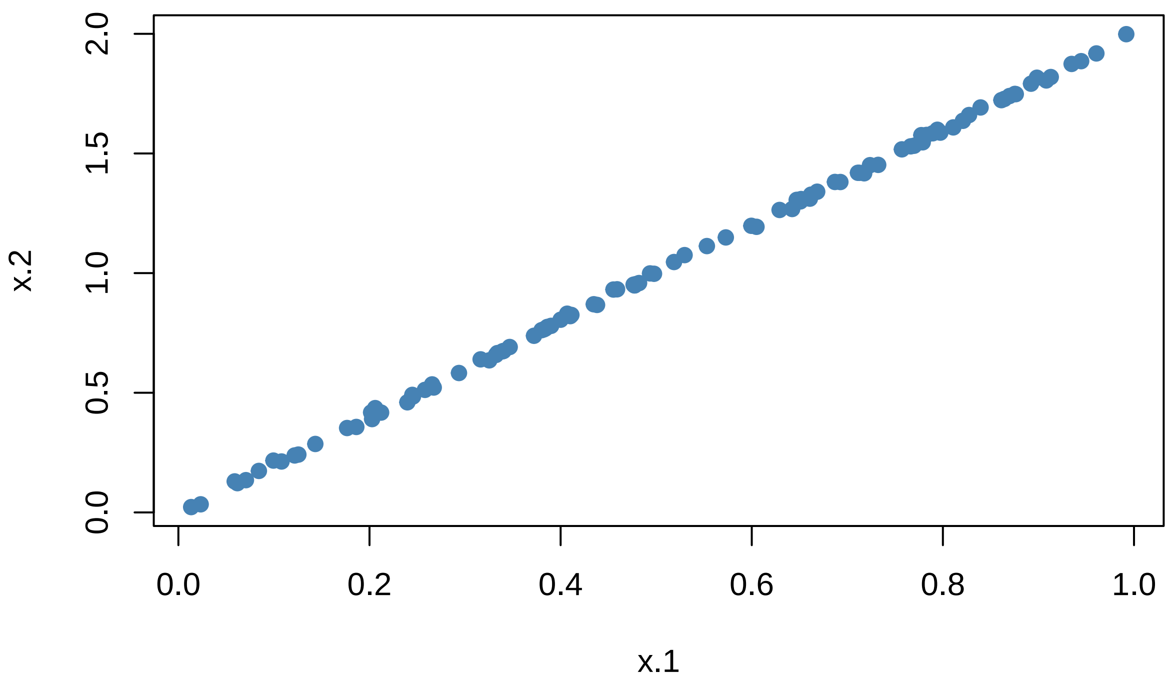
This notebook was inspired by **Problem 3.7.14** from the textbook.

Let us first create our data. First we specify some constants.

```
alpha=2

gamma.1=0.5
gamma.2=5

beta.0=0.5
beta.1=2
beta.2=3
beta.3=4
```

Now we simulate some data.

```
#setting the seed for replicability
set.seed(1)

#simulating the predictors
x.1=runif(100)
#x.1
x.2=alpha*x.1+rnorm(100, mean=0, sd=0.01)
#x.2
#cor(x.1,x.2)
plot(x.1,x.2, pch=19, col="steelblue")
```



```r
x.3=gamma.1*x.1+gamma.2*x.2+rnorm(100, mean=0, sd=0.01)

#simulating the response
y=beta.0+beta.1*x.1+beta.2*x.2+beta.3*x.3+rnorm(100)
```

Now, we arrange everyone into a data frame.

```r
df=data.frame(x.1, x.2, x.3, y)
df
```

```
##           x.1       x.2       x.3         y
## 1  0.26550866 0.5349984  2.812248 14.021227
## 2  0.37212390 0.7381275  3.876514 19.371854
## 3  0.57285336 1.1491179  6.028836 29.138748
## 4  0.90820779 1.8051219  9.470420 45.365797
## 5  0.20168193 0.4176941  2.174437 11.549744
## 6  0.89838968 1.8165834  9.521360 46.978197
## 7  0.94467527 1.8856783  9.910730 45.286207
## 8  0.66079779 1.3111542  6.879957 33.847628
## 9  0.62911404 1.2639253  6.620339 32.406085
## 10 0.06178627 0.1222220  0.660696  3.207755
## 11 0.20597457 0.4359653  2.287065 12.319118
## 12 0.17655675 0.3527211  1.849497  8.920029
## 13 0.68702285 1.3809431  7.258812 34.767791
## 14 0.38410372 0.7684875  4.043353 20.604493
## 15 0.76984142 1.5322501  8.039979 40.515976
## 16 0.49769924 0.9972864  5.257343 25.786683
## 17 0.71761851 1.4171874  7.442196 35.533400
## 18 0.99190609 1.9984677 10.474047 49.186289
## 19 0.38003518 0.7616029  3.996588 19.200198
## 20 0.77744522 1.5766166  8.273881 38.940434
## 21 0.93470523 1.8741656  9.861260 47.178015
## 22 0.21214252 0.4171856  2.193057 11.342450
## 23 0.65167377 1.3094548  6.877681 32.390578
## 24 0.12555510 0.2417692  1.270852  9.208993
## 25 0.26722067 0.5219050  2.739795 13.715349
##  [ reached 'max' / getOption("max.print") -- omitted 75 rows ]
```

What can we say about the correlations?

```r
cor(df)
```

```
##           x.1       x.2       x.3         y
## x.1 1.0000000 0.9998475 0.9998507 0.9972650
## x.2 0.9998475 1.0000000 0.9999929 0.9973901
## x.3 0.9998507 0.9999929 1.0000000 0.9973878
## y   0.9972650 0.9973901 0.9973878 1.0000000
```

Now, let's fit a regression on these data.

```r
lm.fit=lm(y ~ x.1+x.2+x.3, data=df)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = y ~ x.1 + x.2 + x.3, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.73347 -0.60458 -0.00543  0.65950  2.25890
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   0.9723     0.2134   4.556 0.0000154 ***
## x.1           3.9192    21.1345   0.185     0.853
## x.2          15.8052    48.4469   0.326     0.745
## x.3           1.3098     9.3303   0.140     0.889
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9705 on 96 degrees of freedom
## Multiple R-squared:  0.9948, Adjusted R-squared:  0.9946
## F-statistic:  6111 on 3 and 96 DF,  p-value: < 2.2e-16
```

Looking at the summary, we see an admirable R^2 coupled with horrible $p-$values. We cannot reject the null hypothesis that $\beta_i = 0$ for any $i = 1, 2, 3$. However, since we know the model we simulated with, we can assess how good the fit is by comparison. Remember that the actual βs: $(0.5, 2, 3, 4)$. The estimates we got are these:

```r
coef(lm.fit)
```

```
## (Intercept)         x.1         x.2         x.3
##   0.9722882   3.9192167  15.8051621   1.3097744
```

They are quite a bit "off".

What would happen for regressions on just the individual predictors?

```r
lm.fit.1=lm(y~x.1, data=df)
summary(lm.fit.1)
```

```
##
## Call:
## lm(formula = y ~ x.1, data = df)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.78461 -0.54813 -0.05748  0.65785  2.07916
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   0.9337     0.2151    4.34 0.0000346 ***
## x.1          49.3496     0.3695  133.57   < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9836 on 98 degrees of freedom
## Multiple R-squared:  0.9945, Adjusted R-squared:  0.9945
## F-statistic: 1.784e+04 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
lm.fit.2=lm(y~x.2, data=df)
summary(lm.fit.2)
```

```
##
## Call:
## lm(formula = y ~ x.2, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.71923 -0.60338  0.01538  0.66302  2.27424
##
## Coefficients:
##             Estimate Std. Error t value   Pr(>|t|)
## (Intercept)   0.9786     0.2098   4.664 0.00000983 ***
## x.2          24.6357     0.1801 136.752    < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9609 on 98 degrees of freedom
## Multiple R-squared:  0.9948, Adjusted R-squared:  0.9947
## F-statistic: 1.87e+04 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
lm.fit.3=lm(y~x.3, data=df)
summary(lm.fit.3)
```

```
##
## Call:
## lm(formula = y ~ x.3, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.75389 -0.60225 -0.02139  0.62294  2.27305
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  0.97097    0.20998   4.624 0.0000115 ***
## x.3          4.69368    0.03434 136.691   < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.9613 on 98 degrees of freedom
## Multiple R-squared:  0.9948, Adjusted R-squared:  0.9947
## F-statistic: 1.868e+04 on 1 and 98 DF,  p-value: < 2.2e-16
```

Now, the significance is blissfully back - **for all**. The problem we illustrated here is that of **collinearity**. There is no test for collinearity, but it would seem that a beneficial pre-processing step when dealing with multiple predictors could be to fit the predictors on other predictors. For instance, we have

```r
lm.fit.x<-lm(x.3 ~ x.1+x.2, data=df)
summary(lm.fit.x)
```

```
##
## Call:
## lm(formula = x.3 ~ x.1 + x.2, data = df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.028311 -0.007273 -0.000537  0.006338  0.023359
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.001305   0.002319   0.563    0.575
## x.1         0.356009   0.227133   1.567    0.120
## x.2         5.070967   0.113372  44.728   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01056 on 97 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 3.513e+06 on 2 and 97 DF,  p-value: < 2.2e-16
```

We would also wish to reduce the dimension of the array of predictors by getting rid of linear dependence (as much as is feasible). This can be done by dropping predictors or by creating a linear combination of predictors (see the discussion on p.103 of the textbook).