cls-lm-regression

April 21, 2022

1 LM & CLS prediction models

1.0.1 Author: Thomas Nørgaard

1.1 Multiple linear regression

[1]:
```
# Reading in data.
data1 <- read.table("PAHI.txt",header=TRUE)
pah <- data1
```

[2]:
```
# Making the linear model
modela1 <- lm(pah[,2] ~ ., data = subset(pah, select=-c(3:11)))

# Finding the rank by calling rank on the model using the qr function.
paste("The rank of the matrix is",qr(modela1)$rank)
```

```
'The rank of the matrix is 25'
```

A least squares solution always exists, because we can always find the projection from y into the subspace.

The least squares solution is not unique, because we have too many vectors for having just a basis and our vectors are not linearly independent. We can write our vectors in several ways and manipulate the beta vector. We have too many vectors/predictor variables and redundant information.

[3]:
```
# Finding the RSS using the deviance function.
paste("The residual sum of squares is",(deviance(modela1)))
```

```
'The residual sum of squares is 0'
```

With a RSS of 0, the model is basically a perfect fit. This almost certainly means that we overfit a lot. The overfitting is a result of us having a higher number predictor variables ($p = 27$) than our rank value ($\text{rank} = 25$).

A potential problem of using multiple linear regression in this situation would be that the model will automatically overfit to become a perfect fit, because our number of variables exceed our rank. This basically makes the model useless, as it cannot tell us anything new.

2 Multivariate and multiple linear regression with a restricted number of predictors

The feedback mentioned parts 2.5 + 2.6 + 2.7. In order to correct 2.6, we had to make our code work in the predict() function. This meant that we needed to make our lm() differently, without using x and y blocks, but instead subsetting inside the lm() command itself. We also needed to separate our testing data before centering.

A main reason our lm() and prediction did not work properly, is that the way we had initially split the data into training and test, made it unable run the lm() correctly. We assume this was due to a mix of matrices, dataframes, doubles, etc, making R unable to use the functions properly.

With a model that was not correct, the predict() function mentioned in the feedback could not work properly.

Due to these issues, we determined that we needed to do the entire part 2 from scratch. The old code is in the bottom, #'ed out.

[4]:
```
# Reading in data.
data2 <- read.table("PAHI.txt",header=TRUE)
pah2 <- data2
pah2test <- data2
```

[5]:
```
# Using scale() to center the data
pah2 <- scale(pah2,center=TRUE,scale=FALSE)
```

[6]:
```
# Removing the rows, that are not supposed to be in the training set, but will␣
 ↪be in the test set.
pah2 <- pah2[-c(1,6,17,19,22),]

# This variable is used to only pick observations that are integer multiple of␣
 ↪10.
int_selector <- seq(12,38,2)
```

[7]:
```
# Fitting the model, using the int_selector to limit the data available to lm()␣
 ↪in the pah2 dataset.
modela2 <- lm(pah2[,2:11]~.-1, data = as.data.frame(pah2[,int_selector]))
```

[8]:
```
# Finding the rank of the design matrix
paste("The rank of the matrix is",qr(modela2)$rank)
```

'The rank of the matrix is 14'

As stated earlier, a least squares solution always exists, because we can always find the projection from y into the subspace.

In this case, our rank (14) is the same as our number of predictor variables (14). With them being equal, we have a unique least squares solution.

[24]:
```r
# Estimating the coefficients of the model.
modela2$coefficients
```

	Py	Ace	Anth	Acy	Chry	Benz	Fluora	Flu
x220	-0.14930340	0.07434467	0.02448284	0.07626494	-0.07362480	-0.1787227	0.3728748	0.3
x230	-0.40141759	0.26902112	0.23241826	0.14285183	0.24639116	0.6842439	-1.1429160	0.8
x240	1.12638456	-0.86862880	-0.61465078	-0.84194644	-0.43239083	-1.9936403	1.9561911	-2.8
x250	0.06743632	-0.00944430	1.71722100	0.48198864	0.21057402	0.4270298	-0.8219911	0.8
x260	-2.11134443	1.48063865	-4.33903162	0.26780863	-0.71807864	-0.5136820	-0.3661484	0.7
x270	1.32435449	-0.66671264	2.97290706	-2.14093778	3.54085511	1.4336318	-2.7232631	-0.6
x280	1.04085635	-0.50690356	-1.87109763	1.60160432	-1.63818283	2.3457736	1.5176568	-4.8
x290	-1.22894106	-0.49895937	0.93111972	-0.24305931	0.06007888	0.9239939	0.7443119	3.3
x300	3.30809532	-0.75953636	3.26656551	1.73329673	-4.77205379	-3.8604583	5.9326598	22.
x310	-12.71097835	4.14518270	-2.71269608	-3.89542987	5.72615919	-0.4879462	-10.0251388	-32
x320	9.71265564	-5.94730085	-2.67425850	15.42454078	-5.29856222	-9.8001949	26.4132176	9.7
x330	8.27549529	4.63012024	2.47521697	-12.34126503	1.75372960	7.2434647	-26.5880858	9.1
x340	-15.83265704	3.17369711	4.16317934	-0.42304232	-0.23321423	17.2306689	-3.8273510	-3.7
x350	13.26122156	2.74032423	-5.23672816	-1.19446464	2.16114728	-17.5981500	9.3290328	-10

[10]:
```r
# Estimating the standard deviation using the sigma() function.
sigma(modela2)
```

Py 0.0601731467647676 **Ace** 0.0743407115141682 **Anth** 0.0202522497341119 **Acy**
0.0553698892605009 **Chry** 0.00980269098559937 **Benz** 0.0485215942370758 **Fluora**
0.0303182414470409 **Fluore** 0.0748833454061967 **Nap** 0.0233023397451102 **Phen**
0.0406722126932099

[11]:
```r
# Making the prediction on the test data, using the predict() function.
prediction <- predict(modela2, newdata = as.data.
↪frame(pah2test[c(1,6,17,19,22),]))
prediction
```

	Py	Ace	Anth	Acy	Chry	Benz	Fluora	Fluore	Nap
1	0.29416898	0.042182012	0.1686285	0.1994724	0.3126379	1.8650117	0.20489698	0.7951271	0.02
6	0.39896078	-0.048349013	0.1745521	0.1611428	0.3830086	2.3841348	0.18353904	1.0947763	0.04
17	0.01106472	0.041826024	0.0554367	0.1212910	0.4003510	1.2028930	0.14627445	1.0671226	0.02
19	0.12555903	0.032294384	0.2250922	0.1705583	0.1643319	2.7685813	0.09917329	0.6568584	-0.0
22	0.15754532	0.006704391	0.2293326	0.1868290	0.2493634	0.6780484	0.37067965	1.1023818	0.03

[12]:
```r
# Finding the mean squares error using the apply() function to make the␣
↪calculation.
MSE_MLR <- apply((pah2test[c(1,6,17,19,22),2:11] - prediction)^2,2,mean)
MSE_MLR
```

Py 0.0223795049708643 **Ace** 0.016450929047628 **Anth** 0.000522403664789312 **Acy**
0.00650298551003752 **Chry** 0.00362125635925967 **Benz** 0.029826118407214 **Fluora**
0.0123404826461349 **Fluore** 0.0286375958826755 **Nap** 0.00728744462653437 **Phen**
0.0323539734209463

3 Classical least squares using all available predictors

The feedback mentioned parts 3.1 + 3.2 + 3.3. This is pretty much the entire CLS part, which made us determine again to do from scratch, now also using the knowledge we had gained from redoing part 2. We had made the same mistakes with data preparation, which meant redoing the entire part was the best solution.

The old code can be found in the bottom, #'ed out.

[13]:
```
# Reading in data.
data3 <- read.table("PAHI.txt",header=TRUE)
pah3 <- data3
pah3test <- data3
```

[14]:
```
# Using scale() to center the data
pah3 <- scale(pah3,center=TRUE,scale=FALSE)
```

[15]:
```
# Removing the rows, that are not supposed to be in the training set, but will␣
 ↪be in the test set.
pah3 <- pah3[-c(1,6,17,19,22),]
```

[16]:
```
# Fitting the model
modela3 <- lm(pah3[,12:38]~.-1, data = as.data.frame(pah3[,2:11]))
```

[17]:
```
# Estimating the coefficients of the model.
modela3$coefficients
```

	x220	x225	x230	x235	x240	x245	x250	x2
Py	-0.01986882	0.051162285	0.12932753	0.16545072	0.26791466	0.13562661	0.009324648	0.0
Ace	0.32239837	0.681525019	0.49206925	0.08824934	-0.02103315	0.03178691	0.101814640	0.
Anth	0.00690426	-0.061213146	-0.11876708	0.14318982	0.32823445	0.67161524	0.960704515	0.8
Acy	0.58417989	0.823188335	0.86526462	0.48137662	0.22321112	0.11854579	0.118900976	0.0
Chry	0.22020513	0.214183357	0.17950716	0.12490871	0.11005657	0.12722903	0.188689554	0.2
Benz	0.17743732	0.170711725	0.14576581	0.09717913	0.08666005	0.10985165	0.140834507	0.
Fluora	0.16595385	0.085304121	0.08827611	0.30239989	0.30446513	0.23079431	0.236575352	0.2
Fluore	0.14498886	0.105527514	0.05952847	0.04460454	0.02201679	0.03678084	0.056460248	0.0
Nap	0.50997837	-0.013594981	-0.25189075	-0.14001066	-0.07251315	-0.04417309	0.017034640	-0
Phen	0.08066813	0.005026241	0.02435548	0.10047803	0.18267612	0.28891668	0.349403840	0.2

[18]:
```
# The CLS model is currently made to predict X using Y, but we want to predict␣
 ↪our Y (response variables).
# Some manipulation is needed, to go from X=YA+E  to  B=A^T(AA^T)^-1:
cls_model <-␣
 ↪t(modela3$coefficients)%*%solve(modela3$coefficients%*%t(modela3$coefficients))

# Since the predict() function requires a lm() input, the cls_model predictions␣
 ↪needs to be calculated manually.
```

```
# For this a variable x containing the test data is made (Those rows removed␣
 ↪from the traning set).
x <- as.matrix(pah3test[c(1,6,17,19,22),12:38])

# Predictions are made using the x and beta_predict variables.
cls_prediction <- x%*%cls_model
cls_prediction
```

	Py	Ace	Anth	Acy	Chry	Benz	Fluora	Fluore	Nap
1	0.4549077	0.1447793	0.11512053	0.16993843	0.3414104	1.8664816	0.2040679	0.593594585	0.08
6	0.4911077	0.2971919	0.04408487	0.05389204	0.4170784	2.3280755	0.2105776	0.735995202	0.14
17	0.1271329	0.1900362	-0.01444155	0.05575390	0.4778940	1.1444007	0.2148871	0.672380589	0.06
19	0.1264188	0.2297517	0.13568509	0.05236866	0.2865161	2.6244347	0.1685086	0.007700859	0.01
22	0.2174676	0.2494077	0.11970557	0.05255585	0.3674431	0.6464693	0.3572481	0.685021584	0.16

[19]:
```
# Finding the MSE of the CLS prediction.
MSE_CLS <- apply((pah3test[c(1,6,17,19,22),2:11] - cls_prediction)^2,2,mean)
MSE_CLS
```

```
Py      0.00448512605719092 Ace     0.0125825227038654 Anth     0.0113214079697619 Acy
 0.00176394123620277 Chry       0.00135519955854667 Benz       0.0220391616228199 Fluora
 0.0173838742244353 Fluore       0.112300359392475 Nap       0.000413492446056952 Phen
 0.0854047155417972
```

4 Comparison of predictors

[23]:
```
# Subtracting the two MSE from each other, to get an easy overview for␣
 ↪comparison.
# If there is a negative number, it means that the MLR performs better than CLS␣
 ↪and vice versa.
MSE_MLR - MSE_CLS
```

```
Py      0.0178943789136734 Ace      0.00386840634376254 Anth      -0.0107990043049726 Acy
 0.00473904427383475 Chry        0.002266056800713 Benz       0.00778695678439406 Fluora
 -0.00504339157830041 Fluore      -0.0836627635097994 Nap      0.00687395218047742 Phen
 -0.0530507421208509
```

From the comparison of the mean squared errors, we can see that MLR performs better in 4 out of 10 cases to predict the response variable, where the CLS model performs better in the remaining 6 out of 10 cases.

CLS performs better on predicting the Py, Ace, Acy, Chry, Benz and Nap responce variables.

MLR performs better on prediction the Anth, Fluora, Fluore and Phen response variables.

So, one model does not always perform better than the other. It depends on the response variable that is being predicted.

Why is this comparison of the accuracy of predictions reasonable? What are the problematic aspects of this comparison that might make the conclusions questionable?

It it reasonable to compare the predictions, because they are compared using MSE. So even though they don't use same amount of X (Predictor variables), the MSE makes a useable comparison. The MSE tells us how close the regression line of our model is to the actual data. Or, how close our prediction is to the actual values in our data.

It is a potential problem that the CLS model uses more predictors than the MLR model. It is possible (not assured) that some of the predictors that we filtered out in the MLR part, would have improved that model and made it equally good or better than the CLS model. This could however also hurt CLS, which due to deriving from the Beer-Lambert law always needs to use all predictor variables. In another case this might hurt the accuracy of the CLS, by introducing a lot of noise. This noise can potentially be limited in MLR, by removing noise predictors.