Exercise 10

Tobias Famos

Loading the data

Load the data

```
boston <- read.table("/home/tobias/unibe/statistical methods in R/Exercise 10/Boston.txt", header=T)
str(boston)
```

```
## 'data.frame':    506 obs. of  13 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : int  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

Drop the `medv`row

```
boston.medv <- boston$medv
boston.clean <- subset(boston, select=-medv)
str(boston.clean)
```

```
## 'data.frame':    506 obs. of  12 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : int  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
```

Task 1: Normalize and build PCA.

Normalize the Data with the z-score using the built in scale function. Note that scale returns a matrix not a data frame. For simplicity, I have converted it back to a data frame

```
boston.nrom <- as.data.frame(scale(boston.clean))
cor(boston.nrom)
```

```
##                crim          zn       indus         chas         nox
## crim     1.00000000 -0.20046922  0.40658341 -0.055891582  0.42097171
## zn      -0.20046922  1.00000000 -0.53382819 -0.042696719 -0.51660371
## indus    0.40658341 -0.53382819  1.00000000  0.062938027  0.76365145
## chas    -0.05589158 -0.04269672  0.06293803  1.000000000  0.09120281
## nox      0.42097171 -0.51660371  0.76365145  0.091202807  1.00000000
## rm      -0.21924670  0.31199059 -0.39167585  0.091251225 -0.30218819
## age      0.35273425 -0.56953734  0.64477851  0.086517774  0.73147010
## dis     -0.37967009  0.66440822 -0.70802699 -0.099175780 -0.76923011
## rad      0.62550515 -0.31194783  0.59512927 -0.007368241  0.61144056
## tax      0.58276431 -0.31456332  0.72076018 -0.035586518  0.66802320
## ptratio  0.28994558 -0.39167855  0.38324756 -0.121515174  0.18893268
## lstat    0.45562148 -0.41299457  0.60379972 -0.053929298  0.59087892
##                  rm         age         dis          rad         tax    ptratio
## crim    -0.21924670  0.35273425 -0.37967009  0.625505145  0.58276431  0.2899456
## zn       0.31199059 -0.56953734  0.66440822 -0.311947826 -0.31456332 -0.3916785
## indus   -0.39167585  0.64477851 -0.70802699  0.595129275  0.72076018  0.3832476
## chas     0.09125123  0.08651777 -0.09917578 -0.007368241 -0.03558652 -0.1215152
## nox     -0.30218819  0.73147010 -0.76923011  0.611440563  0.66802320  0.1889327
## rm       1.00000000 -0.24026493  0.20524621 -0.209846668 -0.29204783 -0.3555015
## age     -0.24026493  1.00000000 -0.74788054  0.456022452  0.50645559  0.2615150
## dis      0.20524621 -0.74788054  1.00000000 -0.494587930 -0.53443158 -0.2324705
## rad     -0.20984667  0.45602245 -0.49458793  1.000000000  0.91022819  0.4647412
## tax     -0.29204783  0.50645559 -0.53443158  0.910228189  1.00000000  0.4608530
## ptratio -0.35550149  0.26151501 -0.23247054  0.464741179  0.46085304  1.0000000
## lstat   -0.61380827  0.60233853 -0.49699583  0.488676335  0.54399341  0.3740443
##              lstat
## crim     0.4556215
## zn      -0.4129946
## indus    0.6037997
## chas    -0.0539293
## nox      0.5908789
## rm      -0.6138083
## age      0.6023385
## dis     -0.4969958
## rad      0.4886763
## tax      0.5439934
## ptratio  0.3740443
## lstat    1.0000000
```

Now building the PCA with the built in function `princomp`

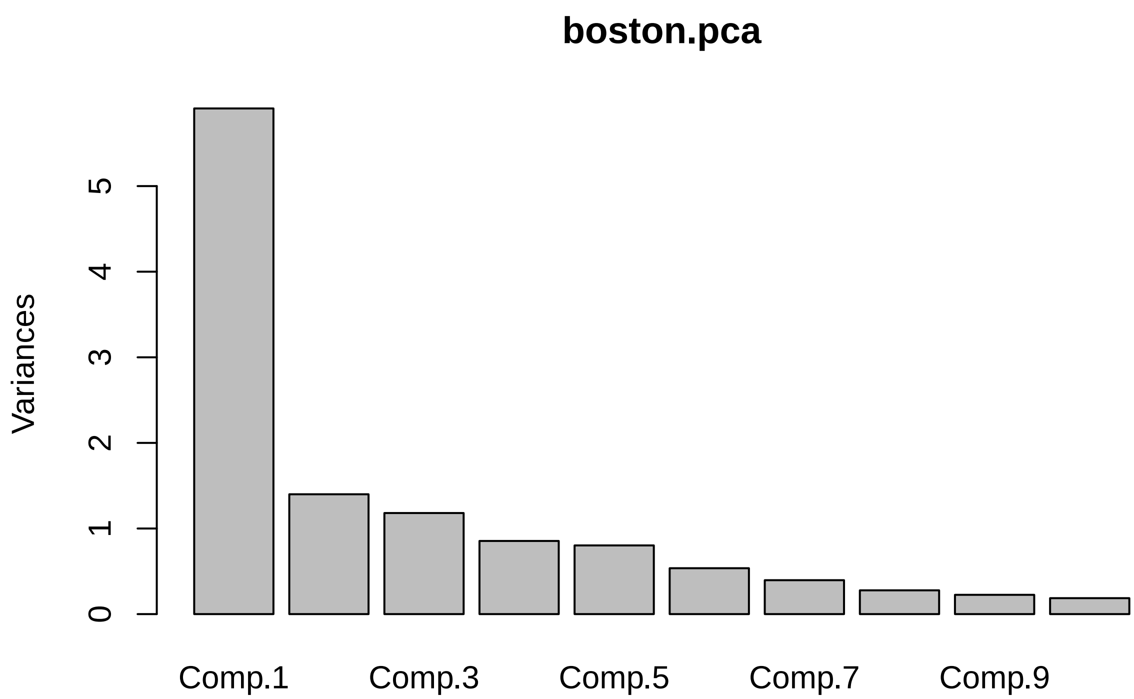
```
boston.pca <- princomp(boston.nrom, cor=T)
summary(boston.pca, loadings=T)
```

```
## Importance of components:
##                           Comp.1    Comp.2     Comp.3     Comp.4    Comp.5
## Standard deviation     2.4304497 1.1832423 1.08659863 0.92436016 0.8957392
```

```
## Proportion of Variance 0.4922571 0.1166719 0.09839138 0.07120348 0.0668624
## Cumulative Proportion  0.4922571 0.6089290 0.70732039 0.77852386 0.8453863
##                           Comp.6    Comp.7     Comp.8     Comp.9    Comp.10
## Standard deviation     0.7322458 0.6293692 0.52723411 0.47451400 0.43151563
## Proportion of Variance 0.0446820 0.0330088 0.02316465 0.01876363 0.01551715
## Cumulative Proportion  0.8900683 0.9230771 0.94624170 0.96500533 0.98052248
##                           Comp.11     Comp.12
## Standard deviation     0.41256242 0.252036760
## Proportion of Variance 0.01418398 0.005293544
## Cumulative Proportion  0.99470646 1.000000000
##
## Loadings:
##         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
## crim     0.251  0.274  0.351         0.192  0.760  0.155  0.272         0.109
## zn      -0.266  0.250  0.359  0.194  0.402 -0.295 -0.401  0.374  0.259 -0.268
## indus    0.355                             -0.345  0.174  0.633 -0.374  0.313
## chas           -0.503  0.201  0.811 -0.197
## nox      0.350 -0.233                0.215 -0.208                0.203 -0.136
## rm      -0.196 -0.273  0.561 -0.402 -0.285        -0.327        -0.441
## age      0.323 -0.293        -0.163         0.109 -0.600         0.392  0.456
## dis     -0.331  0.343         0.234               -0.122 -0.162 -0.166  0.690
## rad      0.322  0.231  0.408        -0.119 -0.149        -0.462
## tax      0.342  0.213  0.331               -0.329        -0.168         0.115
## ptratio  0.211  0.393 -0.184  0.109 -0.702        -0.318  0.255  0.126 -0.186
## lstat    0.315  0.128 -0.264  0.185  0.346        -0.425 -0.220 -0.598 -0.255
##         Comp.11 Comp.12
## crim
## zn
## indus   -0.116  -0.251
## chas
## nox      0.807
## rm       0.135
## age     -0.188
## dis      0.402
## rad     -0.115  -0.633
## tax     -0.221   0.721
## ptratio  0.214
## lstat
```

Plot the PCA to have a visual representation

```
plot(boston.pca)
```



Task 2

From the summary of the loadings of the PCA we can deduce the following on the influence of the predictors on the component 1: The `indus` (proportion of non-retail business acres per town.) has the highest influence on the component 1. The `chas` (view on the river) is not considered in the component 1. And the lowest influence on the component 1 (when not counting not considered predictors) is `rm`(average number of rooms per dwelling)

```r
summary(boston.pca, loadings=T)
```

```
## Importance of components:
##                           Comp.1    Comp.2     Comp.3     Comp.4    Comp.5
## Standard deviation     2.4304497 1.1832423 1.08659863 0.92436016 0.8957392
## Proportion of Variance 0.4922571 0.1166719 0.09839138 0.07120348 0.0668624
## Cumulative Proportion  0.4922571 0.6089290 0.70732039 0.77852386 0.8453863
##                           Comp.6    Comp.7     Comp.8     Comp.9    Comp.10
## Standard deviation     0.7322458 0.6293692 0.52723411 0.47451400 0.43151563
## Proportion of Variance 0.0446820 0.0330088 0.02316465 0.01876363 0.01551715
## Cumulative Proportion  0.8900683 0.9230771 0.94624170 0.96500533 0.98052248
##                           Comp.11     Comp.12
## Standard deviation     0.41256242 0.252036760
## Proportion of Variance 0.01418398 0.005293544
## Cumulative Proportion  0.99470646 1.000000000
##
## Loadings:
##         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
## crim     0.251  0.274  0.351         0.192  0.760  0.155  0.272         0.109
## zn      -0.266  0.250  0.359  0.194  0.402 -0.295 -0.401  0.374  0.259 -0.268
## indus    0.355                             -0.345  0.174  0.633 -0.374  0.313
## chas           -0.503  0.201  0.811 -0.197
## nox      0.350 -0.233                0.215 -0.208                0.203 -0.136
## rm      -0.196 -0.273  0.561 -0.402 -0.285        -0.327        -0.441
## age      0.323 -0.293        -0.163         0.109 -0.600         0.392  0.456
## dis     -0.331  0.343         0.234               -0.122 -0.162 -0.166  0.690
## rad      0.322  0.231  0.408        -0.119 -0.149        -0.462
```

```
## tax      0.342  0.213  0.331                -0.329        -0.168          0.115
## ptratio  0.211  0.393 -0.184  0.109 -0.702        -0.318  0.255  0.126 -0.186
## lstat    0.315  0.128 -0.264  0.185  0.346        -0.425 -0.220 -0.598 -0.255
##         Comp.11 Comp.12
## crim
## zn
## indus   -0.116  -0.251
## chas
## nox      0.807
## rm       0.135
## age     -0.188
## dis      0.402
## rad     -0.115  -0.633
## tax     -0.221   0.721
## ptratio  0.214
## lstat
```

Task 3

The portion of variance is not given explicitly by the `princomp` command but can be calculated using the standard deviations (portion of variance is the normalized standard deviation)

```
portion_of_variance <- boston.pca$sdev^2/sum(boston.pca$sdev^2)
portion_of_variance
```

```
##      Comp.1      Comp.2      Comp.3      Comp.4      Comp.5      Comp.6
## 0.492257148 0.116671856 0.098391382 0.071203475 0.066862395 0.044681998
##      Comp.7      Comp.8      Comp.9     Comp.10     Comp.11     Comp.12
## 0.033008799 0.023164650 0.018763628 0.015517145 0.014183979 0.005293544
```

```
sum(portion_of_variance[0:4])
```

```
## [1] 0.7785239
```

```
sum(portion_of_variance[0:5])
```

```
## [1] 0.8453863
```

The break of explaining 80% of the variance is reached by adding the component 5.

Task 4: Create new dataset and predict

All the values for the components are already provided by `princomp`. Thus we only have to select the stated components (1 - 5) from Task 3 and we have build the dataset.

```
boston.data.pca <- as.data.frame(boston.pca$scores[,1:5])
boston.data.pca$medv <- boston.medv
```

Now build a multiple linear regression

```
library(boot)
boston.mlr <- glm(medv~., data=boston.data.pca)
summary(boston.mlr)
```

```
##
## Call:
## glm(formula = medv ~ ., data = boston.data.pca)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -19.196   -2.867   -0.688    1.819   33.623
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 22.53281    0.22863  98.557  < 2e-16 ***
## Comp.1      -2.29837    0.09407 -24.433  < 2e-16 ***
## Comp.2      -2.85708    0.19322 -14.787  < 2e-16 ***
## Comp.3       3.12373    0.21041  14.846  < 2e-16 ***
## Comp.4      -1.77042    0.24733  -7.158 2.94e-12 ***
## Comp.5      -1.34565    0.25524  -5.272 2.01e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 26.44856)
##
##     Null deviance: 42716  on 505  degrees of freedom
## Residual deviance: 13224  on 500  degrees of freedom
## AIC: 3101.2
##
## Number of Fisher Scoring iterations: 2
```

```
boston.cv <- cv.glm(boston.data.pca, boston.mlr, K=10)
```

Task 5 Build an compare model with all components

First build the model using all the components.

```
boston.data.pca_all <- as.data.frame(boston.pca$scores)
boston.data.pca_all$medv <- boston.medv
boston.mlr_all <- glm(medv~., data=boston.data.pca_all)
summary(boston.mlr_all)
```

```
##
## Call:
## glm(formula = medv ~ ., data = boston.data.pca_all)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -15.1304   -2.7673   -0.5814    1.9414   26.2526
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 22.53281    0.21330 105.640  < 2e-16 ***
## Comp.1      -2.29837    0.08776 -26.189  < 2e-16 ***
## Comp.2      -2.85708    0.18027 -15.849  < 2e-16 ***
## Comp.3       3.12373    0.19630  15.913  < 2e-16 ***
## Comp.4      -1.77042    0.23075  -7.672 9.10e-14 ***
## Comp.5      -1.34565    0.23813  -5.651 2.70e-08 ***
## Comp.6      -0.18624    0.29129  -0.639  0.52289
## Comp.7       1.04909    0.33891   3.096  0.00208 **
## Comp.8       0.32326    0.40456   0.799  0.42465
```

```
## Comp.9       1.44544    0.44951   3.216  0.00139 **
## Comp.10     -1.26845    0.49430  -2.566  0.01058 *
## Comp.11     -3.19994    0.51701  -6.189 1.27e-09 ***
## Comp.12     -3.34030    0.84630  -3.947 9.07e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 23.02113)
##
##     Null deviance: 42716  on 505  degrees of freedom
## Residual deviance: 11349  on 493  degrees of freedom
## AIC: 3037.8
##
## Number of Fisher Scoring iterations: 2
```

```
boston.cv.all <- cv.glm(boston.data.pca_all, boston.mlr_all, K=10)
```

Now lets compare the cross validated mean square errors

```
boston.cv$delta[1]
```

```
## [1] 27.00816
```

```
boston.cv.all$delta[1]
```

```
## [1] 24.9549
```

As we acn see, the cross validated mean squared error is smaller for the set using all the components. Although this is not on truly fresh data, thus the danger of overfitting is still present. To eliminate this danger one might do a train / validation split for evaluating the models