Data Modelling and Analysis

Lecture 7 (Practice): Principal Component Analysis Demo in *iris*

Dr Mercedes Torres Torres

1 Questions

1. Apply Principal Component Analysis to iris in R
2. Study the results when using PCA for transformation
3. Study the results when using PCA for dimensionality reduction.

3.1 How many dimensions do you need to keep 95% of the variance?

3.2 And 99% of the variance?

2 Introduction

Principal Component Analysis (PCA) is a data transformation and data reduction technique which helps to:

1. Transform the data by finding relationships between attributes and decorrelating attributes
2. Reduce the dimensionality of the data by projecting it into a new space where the axes are ordered according to variance.

Let's see the effect of applying Principal Component Analysis (PCA) to the *iris* dataset in R.

3 Answers:

Q1: **Apply Principal Component Analysis to iris in R:**

First things first, let's get our chunk options out of the way:

```
knitr::opts_chunk$set(fig.align = 'center', out.width = '60%', fig.pos = 'H')
```

1. Analysis:

Let's look at the data, considering first overall statistics and then with visualisations:

1.1. Statistics:

```r
# 1. Load iris and show a summary of its values:
ir = iris
str(ir)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 ...
```

```r
summary(ir[1:4])
```

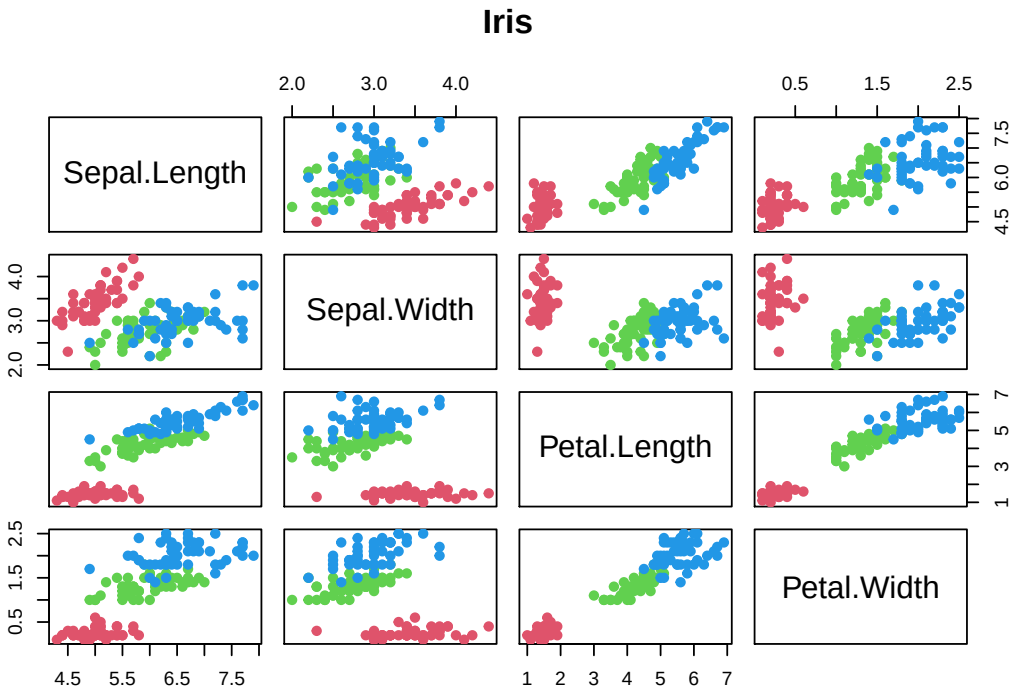
```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

```r
head(ir,5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

1.2. Visualisation:

```r
all = pairs(ir[1:4],main="Iris", pch=19, col=as.numeric(ir$Species)+1)
```



Let's examine the variability of all numeric variables

```
sapply(ir[1:4],var) # sapply: applies function var over list vector iris[1:4]
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.6856935    0.1899794    3.1162779    0.5810063
```

```
range(sapply(ir[1:4],var)) # gives us the range of values in iris
```

```
## [1] 0.1899794 3.1162779
```

The range in this case is not huge, but still worth considering.

2. Standardisation:

Before applying PCA, we must standardize our variables with *scale()* function. Do not forget to remove your label or class variable. PCA does not use the labels in any way.

```
iris.stand = as.data.frame(scale(iris[,1:4])) #Don't forget to remove your label!!
head(iris.stand) # here you can see how the scaled iris looks like
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1   -0.8976739  1.01560199    -1.335752   -1.311052
## 2   -1.1392005 -0.13153881    -1.335752   -1.311052
## 3   -1.3807271  0.32731751    -1.392399   -1.311052
## 4   -1.5014904  0.09788935    -1.279104   -1.311052
## 5   -1.0184372  1.24503015    -1.335752   -1.311052
## 6   -0.5353840  1.93331463    -1.165809   -1.048667
```

```
# After scaling, the st and mean of each attribute should be 1 and 0, respectively
sapply(iris.stand,sd) #now, standard deviations are 1
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##            1            1            1            1
```

```
sapply(iris.stand,mean) #now, mean should be 0 (or very very close to 0)
```

```
##  Sepal.Length   Sepal.Width  Petal.Length   Petal.Width
## -4.484318e-16  2.034094e-16 -2.895326e-17 -3.663049e-17
```

3. Calculate PCA:

For the sake of completeness, let's compare both PCA functions in R: *prcomp()* and *princomp()*.

Remember that, if we use *prcomp()* function, we indicate *scale=TRUE* to use the correlation matrix.

```
pca = prcomp(iris.stand,scale=T) #The preferred method
pca2 = princomp(iris.stand, scores=T)
```

```
pca #shows the dataframe pca with the principal components and the attributes:
```

```
## Standard deviations (1, .., p=4):
## [1] 1.7083611 0.9560494 0.3830886 0.1439265
##
## Rotation (n x k) = (4 x 4):
##                     PC1         PC2        PC3        PC4
## Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
## Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
## Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
## Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

```
summary(pca) # VERY IMPORTANT: looking at 'Cumulative Proportion', we can see
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.7084 0.9560 0.38309 0.14393
## Proportion of Variance 0.7296 0.2285 0.03669 0.00518
## Cumulative Proportion  0.7296 0.9581 0.99482 1.00000
```

```
#that how many PCs we need to take to achieve variance in data.
# In this sample, by taking the first two PCs, we get over 95% of the variance
```

The *summary()* function applied to the result of the PCA shows the effect that each PC has and how much variance they introduce. To see the total effect of the dimensions from PC1 to a particular dimension, take a look at the *Cumulative Proportion*.

Let's take a look at the other PCA:

```
summary(pca2)
```

```
## Importance of components:
##                           Comp.1    Comp.2     Comp.3      Comp.4
## Standard deviation     1.7026571 0.9528572 0.38180950 0.143445939
## Proportion of Variance 0.7296245 0.2285076 0.03668922 0.005178709
## Cumulative Proportion  0.7296245 0.9581321 0.99482129 1.000000000
```

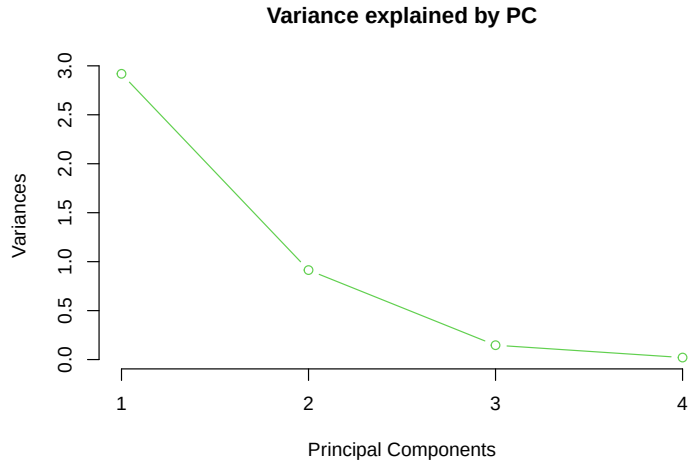
We can look at the standard deviation of each component and the proportion of variance explained by each component.

```
#The standard deviation is stored in (see 'str(pca)'):
pca$sdev
```

```
## [1] 1.7083611 0.9560494 0.3830886 0.1439265
```

You can use *screeplot()* to see how much variance each PC has in a visual way:

```
#it will be useful to decide how many principal components should be retained.
screeplot(pca, type="lines",col=3, main="Variance explained by PC")
title(xlab="Principal Components")
```



You can also access the *loadings* for the Principal Components (PC). The *loadings* are the weights of the original attribute in each of the new dimensions. They will be useful to generate the linear combination of original attributes that give the new set of axes.

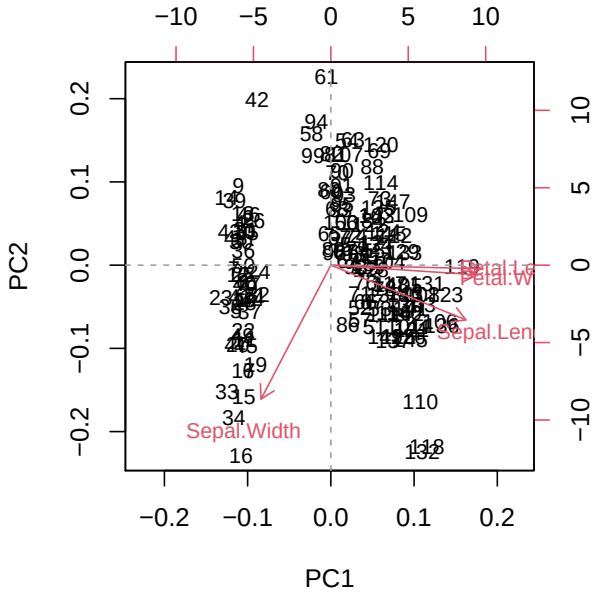
```
#The loadings for the principal components are stored in:
pca$rotation #This shows the linear combinations of each variable into each PC
```

```
##                     PC1         PC2        PC3        PC4
## Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
## Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
## Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
## Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

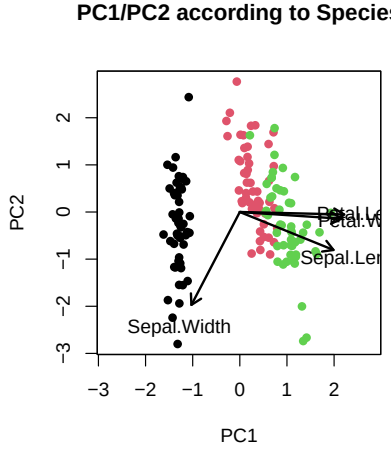
```
# with princomp(): pca2$loadings
#they will be slightly different due to the difference in the PCA calculation
```

Finally, you can also plot the information contained in the first two PCs. As the new axes with the most variance, it's always good to take a look and see how they compare with the original attributes:

```
biplot(pca,cex=0.8)
abline(h = 0, v = 0, lty = 2, col = 8)
```




```
#Using a different biplot function, developed by school and modified mby me
source("extra/bips.R")
bip(pca, col=iris$Species, main = "PC1/PC2 according to Species")
```



Q2: **Study the results when using PCA for transformation**

You can find your transformed data in the x field of the *pca* dataframe.

Remember that PCA decorrelates the data. Let's see if it really did so by comparing the overall correlations between the original standardised dataset and the new transformed dataset:

```
cor(iris.stand)
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

```
cor(pca$x)
```

```
##                PC1           PC2           PC3           PC4
## PC1  1.000000e+00 -9.760996e-17 -1.685228e-15  1.996979e-15
## PC2 -9.760996e-17  1.000000e+00  2.381560e-16  2.409194e-15
## PC3 -1.685228e-15  2.381560e-16  1.000000e+00 -6.157152e-16
## PC4  1.996979e-15  2.409194e-15 -6.157152e-16  1.000000e+00
```

Quite a difference, isn't it!

Q3:**Study the results when using PCA for dimensionality reduction.**

You can combine the Cumulative Proportion of PCs with the projections to make sure you keep the dimensions you need.

Of course, you may do this manually, just checking the results of the PCA and selecting that number of dimensions.

```
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.7084 0.9560 0.38309 0.14393
## Proportion of Variance 0.7296 0.2285 0.03669 0.00518
## Cumulative Proportion  0.7296 0.9581 0.99482 1.00000
```

```
#You can see that for 0.95 variance, you'll need to dimensions
#So, let keep those two dimensions:
```

```
iris_pca=pca$x[,1:2] # x for prcomp
iris_pca2=pca2$scores[,1:2] # scores for princomp
head(iris_pca,4)
```

```
##            PC1        PC2
## [1,] -2.257141 -0.4784238
## [2,] -2.074013  0.6718827
## [3,] -2.356335  0.3407664
## [4,] -2.291707  0.5953999
```

However, you may also create code that does that automatically for you, that way, you can vary the threshold automatically, if desired, and obtain different versions of the dataset. **Try and create your own function to do this.**

I have created one function, called *automatic_pca(data, thres)* that both calculates the pca of a dataset and also returns the projected version of the data with a cumulation proportion of the data of *thres*.

Let's take that function for a spin:

```
var_thres = 0.95
nu_iris = automatic_pca(ir[, 1:4], var_thres)
head(nu_iris)
```

```
##            PC1        PC2
## [1,] -2.257141 -0.4784238
## [2,] -2.074013  0.6718827
## [3,] -2.356335  0.3407664
## [4,] -2.291707  0.5953999
## [5,] -2.381863 -0.6446757
## [6,] -2.068701 -1.4842053
```

We need 2 dimensions to obtain 95% of variance.

```
var_thres = 0.99
nu_iris = automatic_pca(ir[, 1:4], var_thres)
head(nu_iris)
```

```
##            PC1        PC2         PC3
## [1,] -2.257141 -0.4784238  0.12727962
## [2,] -2.074013  0.6718827  0.23382552
## [3,] -2.356335  0.3407664 -0.04405390
## [4,] -2.291707  0.5953999 -0.09098530
## [5,] -2.381863 -0.6446757 -0.01568565
## [6,] -2.068701 -1.4842053 -0.02687825
```

We need 3 dimensions to obtain 99% of variance.