Statistics and Data Analysis in Physical Geography

Practical exercises 7

Multivariate Analysis

Geert Sterk, Niko Wanders, Philip Kraaijenbrink

Department of Physical Geography
Geosciences Faculty
Utrecht University

2023

7.1 Principal components

Principal component analysis (PCA) tries to find major direction of variability, or, after scaling the variables, major directions of linear correlation in the data. PCA can be seen as an exploratory technique that investigates multiple linear correlations among variables, without making the distinction between dependent and independent variables (unlike e.g. linear regression and discriminant analysis). In that sense the technique is sometimes called "unsupervised" – we don’t predict one specific variable from others and can’t measure its predictive capacity from looking at residuals from a single variable.

Principal components is not a statistical technique in the sense of inference, testing, confidence intervals etc; it is rather a way to see whether sets of variables contain redundant information and can be summarized by a smaller set of new variables, called principal components. In some sense you can explain PCA in terms of a projection: from *n* correlated variables, without loss of information the new set of *n* PC are new uncorrelated axes, that explain a decreasing amount of the total variability present. The projected values of the data are called *PC scores*, the rotation matrix to go from the data space to the PC space has coefficients called *loadings*.

7.1.1 Principal component analysis workflow practice

Load the file TABLE612.TXT from Davis book from the website in R, similarly as in the regression exercise 6.

```
# load the tidyverse packages
library(tidyverse)
# load the dataset table612 = read_delim(   file      =
'https://www.kgs.ku.edu/Mathgeo/Books/Stat/ASCII/TABLE612.TXT',   delim     =
"\t",
  )
```

To perform a PCA using matrix calculations by R, conduct the following steps. Check the intermediate outputs.

1. Make a scatter plot of the x1 and x2 data in table612 using ggplot
2. Create a covariance matrix of table612 (use function cov()),
3. Calculate the eigen values and vectors from the covariance matrix (use function eigen()) and store it as new variable eigen_table,
4. Project the data on the two principal components (see also lecture notes 7, Eq. 7.6). To do this in R:
   a. First change the class of table612 to matrix by using X = as.matrix(table612)
   b. U = eigen_table$vectors
   c. Use the function %*% to perform matrix multiplication in R (see https://rcoder.com/matrix-operations-r/ for more information on matrix calculations in R). Using a regular * would perform an item wise multiplication instead. S = X %*% U

5. Plot the first two columns of the matrix (X[,1:2]) and the transformed data (S[,1:2]) using the regular plot function of R.

6. Now project the data on the first principal component only using (Eq. 7.7):
   ```
   U1 = U[,1]
   S2 = X %*% U1
   ```
7. Plot the projected data:
   ```
   plot(S2, rep(0,25))
   ```

8. Create a new dataset of x1 and x2 with transformed data (reduced variance) using Eq. 7.9 from the lecture notes. The function t() allows you to transform a matrix:
   ```
   Xnew = S2 %*% t(U1)
   ```

9. Make again a plot of the x1 and x2 data plot(Xnew)

7.1.2 Principal component analysis on a test data set

Satellite imagery is crucial for various applications, but its high dimensionality can sometimes be challenging. Principal Component Analysis (PCA) is therefore a powerful tool that reduces this complexity by extracting key information from satellite images. In this exercise, you'll explore how PCA can help in satellite image analysis for feature extraction, noise reduction and data reduction. In this exercise, you will work with an ASTER satellite image of the Almere area in the Netherlands and apply PCA analysis to the data.

Create a new folder on your computer for this analysis, create a subfolder "data", and download the data set "aster_flevo.tif" (on blackboard) into it. Next, install the terra package to deal with spatial grids using install.packages(). Create a new script and start by loading the tidyverse and terra packages.

Load the ASTER image in R by using:

```
aster = rast('data/aster_flevo.tif')
```

The ASTER image consists of 6 different bands, or layers, that all have the reflectance of a different wavelength:

band1 = 560 nm (Green) band2
= 660 nm (Red) band3 = 820
nm (Infrared) band4 = 1650 nm
(Infrared) band5 = 2165 nm
(Mid Infrared) band6 = 2205 nm
(Mid Infrared)

Some bands show more information than others. Using PCA, you will evaluate which bands are most relevant to make an image classification, and how you can reduce the total information of all bands and remove information that is redundant.

First, start by having a look at the data by running `print(aster)` in the console to see the image information.

A satellite image basically consists of a stack of matrices describing the pixel values where the pixels are associated with specific coordinates. The dimensions are indicated in the image information under “dimensions”. Have a look at the statistics of each band or matrix using

```
summary(aster)
```

Next, have a look at a false color composite of the satellite image showing the near infrared, red and green bands by running the following command:

```
plotRGB(aster, r=3, g=2, b=1, stretch='lin')
```

Now, have a look at all bands separately in grayscale using

```
plot(aster, col=gray.colors(255))
```

Exercise 7.1.
Which band do you think is the best option to classify land use types and why?
# 7.1: NDVI (NIR - R) / (NIR + R), NDVI is used to detect vegetation density and is used as a indication for land use when other data isn't available.

In the next step look at the correlations between the different bands. To do this, the data first has to be transformed. It is now in a stack of matrices, but to calculate the correlation we need a matrix where the rows describe each observation (i.e. each image pixel) and the columns describe the variables (i.e. each image band). To transform the image to such a matrix use and get a matrix of the coordinates for each row, use:

```
aster_table = as_tibble(aster)
aster_coord = as.data.frame(aster, xy=T)[,1:2]
```

Have a look at the table using print(aster_table). Does this meet your expectations? How many observations are there?

Now apply the correlation function `cor` to the data.

Look at the correlations between the bands and create two scatter plots using `ggplot2`, one for the two bands with the lowest correlation and one for the two bands with the highest correlation. Since there are many observations, plotting all points will test your system so take a sample of the data instead, e.g. 10000 observations. You can use the tidyverse function sample_n() to sample a specific amount of rows.

Additionally, also make a plot between NIR and SWIR1 and inspect its output.

Exercise 7.2a
Looking at the variances in the NIR band and SWIR1 band, which one has the largest variance?
# 7.2a: The NIR band has the highest variance. Ranges from 0 to ~130

Exercise 7.2b
If we would only consider these two bands, how would you describe the first and the second principal component?
# 7.2b: first PC is dominant in NIR and goes from 0,0 to 125, 12.5 Second PC dominant in SWIR1 ranges from 7.5,75 to 50,12.5

Principal components analysis (PCA) is a method in which original data is transformed into a new set of data which may better capture the essential information. Often some variables are highly correlated such that the information contained in one variable is largely a duplication of the information contained in another variable. Instead of throwing away the redundant data, PCA condenses the information in inter-correlated variables into a few variables, called principal components (PC's).

So, PCA is a decorrelation procedure that reorganizes by statistical means the reflectance values from as many of the spectral bands as we choose to include in the analysis. In this case we will include all six bands. Through PCA we will create a new orthogonal coordinate system (the principal components), which can be used to transform the original data into new, uncorrelated data. Usually only a few principle components will be sufficient to capture the essential information.

In this case, the six bands have all the same unit (i.e. reflectance) and so we can use the covariance matrix. If they would have different units, one chooses the correlation matrix instead. Have a look at these outputs:

```
cov(aster_table) cov(scale(aster_table))
cor(aster_table)
all.equal(cor(aster_table), cov(scale(aster_table)))
```

The scale function subtracts the mean and divides by the standard deviation, so the result has mean zero and unit standard deviation (and variance). Note that the covariance of the scaled data equals the correlation of the data. This means that if we work with covariances of scaled data, we effectively work with correlations of the (unscaled) data.

Exercise 7.3
What do the values on the diagonal of the covariance matrix represent?

a. covariances between two different variables
b. variance of a single variable
c. standard deviation of a single variable
d. mean value of a single variable

Carry out a principal component analysis manually on all six variables. First calculate the eigen vectors (use command "eigen") from the covariance matrix. Create the matrix U which contains all six eigen vectors, similar as how you have done in the workflow above.

Exercise 7.4
How much of the total variance is explained by PC1? PC2? PC3? and PC4?
# 7.4: PC1 = 78,0%, PC2 = 21,7%, PC3 = 0.2% PC4 = 0.1%

Also, similar to the workflow before, convert the table to a matrix and project the original data on all six PC's by running the commands:

```
X = as.matrix(aster_table) S = X
%*% U
```

Create a correlation matrix of S and look at the correlations between the projected data.

Use the projected data in combination with the coordinates stored earlier to convert the matrix with transformed observations back into a 2D image using the code below. Create a plot of the projected data on the first PC and compare the result with the "best" band you have selected under exercise 7.1.

```
# convert to raster image
pc_img = rast(setNames(bind_cols(aster_coord, S), c('x','y',paste0('PC',1:6))))
# make plot of PC1
# (note, we take negative of image for visual purposes here) plot(-pc_img[[1]],
col=gray.colors(255, 0, 1))
```

Exercise 7.5a
What do you conclude about the difference or similarity between these two images?
# 7.5a: NDVI has better differences.

Exercise 7.5b
Look at the loadings of PC1 (in matrix U). Which band is the most important for PC1? Is this answer in agreement with your previous answer?
# 7.5b: band 3 (NIR) with red being second importand. Just like an NDVI exploits.
Now create plots of the projected data on all PCs using

```
plot(pc_img, col=gray.colors(255, 0, 1))
```

And look at the summary statistics of the PC matrix

```
summary(S)
```

Exercise 7.6
Which PC's contain useful information in your opinion?
# 7.6: PC1 and PC2, They have the largest variance. so they actually show useful data.

Convert the projected data S back to the original values (by using the inverse of U, which is U^T and calculated by t(U)) and store it as X_back. Compare the converted data with the original data:

```
all.equal(X[,1:5], X_back[,1:5], check.attributes=F)
```

Is the result what you expect?

Now, project the original data on PC1 only (also see workflow) and call the result S1. Convert it to a raster image again by adapting the code for exercise 7.5a. Inspect the projected image and compare it with the previous projected data on PC1 created for exercise 7.5a.

Now transform the data back to the original values and call the result X1. Compare the values of the new matrix with the original matrix X and compare the summary statistics using the summary() function.

Finally, convert the matrix to a raster again, plot it as the same color image as you did in the beginning of this exercise.

```
# create image from the matrix and check results
x1_img = rast(setNames(bind_cols(aster_coord, X1), c('x','y',names(aster_table))))
# plot false color plotRGB(x1_img, r=3, g=2, b=1, stretch='lin')
```

Also plot the bands separately again:

```
# plot all bands separately as grayscale plot(x1_img,
col=gray.colors(255, 0, 1))
```

Exercise 7.7
What can you say about the two matrices? Are they equal or different. If they are different, what is the reason for this?
# 7.7: The original images are a bit clearer/have more contrast but overall the images are very similar.

Repeat all the same steps as for Exercise 7.7, but now use the first three PC's in the transformation instead of just the first one.

Exercise 7.8a
What has happened to the projected and then back-converted data when three PC's are used? Is there a loss of information?
# 7.8a: The same as for 7.7

Exercise 7.8b
What is the advantage in terms of data storage when three principle components are used?
# 7.8b: The data is smaller in size. and is faster to load.

You have calculated the matrix, eigenvectors and eigenvalues "by hand" in the exercises above, but R also has an automatic routine for PCA using the function prcomp(). Apply a PCA to the satellite data using prcomp

```
# perform PCA with dedicated function pca_from_r =
prcomp(aster_table, scale=F)  pca_from_r
```

Exercise 7.9
How do we calculate eigenvalues for principal components from the "standard deviation" of each component?
# 7.9: by taking the square of the standard deviations

- taking the square root of the standard deviation
- eigenvalues are equal to the standard deviations
- by taking the square of the standard deviation

Note that the results of the prcomp are almost the same as when calculating the PC's manually, but the difference is that the prcomp command centres the projected data around the mean. It stores the centering (and scaling parameters when selected) as well as the transformed matrix as separate subitems, accessible using the dollar sign:

```
pca_from_r$center
pca_from_r$scale pca_from_r$x
```

Depending on the purpose of the image classification, different combinations of PC's can be made to obtain maximum separation between different feature classes. For instance, we could combine PC1 and PC2 in one image, where we plot PC1 as green and PC2 as red, but other combinations could be tried as well. However, in actual image classification often the real frequency bands are used instead of the projected data on the PC's. Only in case the image contains a lot of noise there can be a real advantage when the classification is done on the PC transformed data.

An example of plotting a PC image on multiple color channels could be:

```
plotRGB(pc_img, r=1, g=2, b=3, stretch='lin')
```

```r
# install.packages('terra')

# library(terra)

# library(tidyverse)

# library(ggthemes)

aster = rast('C:/Users/Lars/OneDrive/1Uni/1Master/Statistics/Excersises/Exercise 7/data/aster_flevo.tif')


print(aster)

summary(aster)

plotRGB(aster, r=3, g=2, b=1, stretch='lin')

plot(aster, col=gray.colors(255))

# 7.1: NDVI (NIR - R) / (NIR + R), NDVI is used to detect vegetation density and is used as a indication for
land use when other data isn't available.


aster_table = as_tibble(aster)

aster_coord = as.data.frame(aster, xy=T)[,1:2]


cor(aster_table)

ggplot(data = sample_n(aster_table, 10000),aes(x = Red, y = NIR))+

  geom_point()

ggplot(data = sample_n(aster_table, 10000),aes(x = Green, y = Red))+

  geom_point()

ggplot(data = sample_n(aster_table, 10000),aes(x = NIR, y = SWIR1))+

  geom_point()

# 7.2a: The NIR band has the highest varience. Ranges from 0 to ~130

# 7.2b: first PC is dominant in NIR and goes from 0,0 to 125, 12.5 Second PC dominant in SWIR1 ranges
from 7.5,75 to 50,12.5


covtable = cov(aster_table)

cov(scale(aster_table))

cor(aster_table)

all.equal(cor(aster_table), cov(scale(aster_table)))


# 7.3: b
```

```
eigen_table = eigen(covtable)
X = as.matrix(aster_table)
U = eigen_table$vectors
S = X %*% U

PC1percent = eigen_table$values[1]/sum(eigen_table$values)*100
PC2percent = eigen_table$values[2]/sum(eigen_table$values)*100
PC3percent = eigen_table$values[3]/sum(eigen_table$values)*100
PC4percent = eigen_table$values[4]/sum(eigen_table$values)*100

# 7.4: PC1 = 78,0%, PC2 = 21,7%, PC3 = 0.2% PC4 = 0.1%

cor(S)

# convert to raster image
pc_img = rast(setNames(bind_cols(aster_coord, S), c('x','y',paste0('PC',1:6))))
# make plot of PC1
# (note, we take negative of image for visual purposes here)
plot(-pc_img[[1]], col=gray.colors(255, 0, 1))

NDVI = ((aster[[3]]-aster[[2]])/(aster[[3]]+aster[[2]]))

plot(NDVI, col = gray.colors(255,0,1))

# 7.5a: NDVI has better differences.
# 7.5b: band 3 (NIR) with red being second importand. Just like an NDVI exploits.

plot(pc_img, col=gray.colors(255, 0, 1))
summary(S)
# 7.6: PC1 and PC2, They have the largest variance. so they actually show useful data.

X_back = t(U)
```

```
all.equal(X[,1:5], X_back[,1:5], check.attributes=F)
S1 = X %*% U
X1 = S1 %*% X_back
summary(X1)

# create image from the matrix and check results
x1_img = rast(setNames(bind_cols(aster_coord, X1), c('x','y',names(aster_table))))
# plot false color
plotRGB(x1_img, r=3, g=2, b=1, stretch='lin')

# plot all bands separately as grayscale
plot(x1_img, col=gray.colors(255, 0, 1))
# 7.7: The original images are a bit clearer/have more contrast but overall the images are very similar.
# 7.8a: The same as for 7.7
# 7.8b: The data is smaller in size. and is faster to load.

# perform PCA with dedicated function
pca_from_r = prcomp(aster_table, scale=F)
pca_from_r
# 7.9: by taking the square of the standard deviations

pca_from_r$center
pca_from_r$scale
pca_from_r$x

plotRGB(pc_img, r=1, g=2, b=3, stretch='lin')
```