PCA Revealed

Part 3: Preliminary Concepts

Gaston Sanchez

August 2014

Readme

Introduction

Introduction

About

Before starting our discussion about **Principal Components Analysis** (PCA), we need to review some conceptual elements of extreme importance.

Main Idea

The idea is to introduce some material that surrounds PCA, but at the same time, this material can be presented independently of PCA.

About the Data

About PCA

Briefly:

Principal Components Analysis (PCA) is a multivariate method that allows us to study and explore a data set of quantitative variables measured on a set of objects.

Data Structure

Analyzed Data

PCA is applied to study a data set under a rectangular format: table with rows and columns.

Conventionally

- Rows represent objects (i.e. observations, individuals, samples).
- Columns represent variables (i.e. features, characteristics, descriptors).

Example: Cereals Data Set

```
cereals
```

```
##                 Cups Calories Carbs Fat Fiber Potassium Protein Sodium Sugars
## CapnCrunch      0.75      120  12.0   2   0.0        35       1    220     12
## CocoaPuffs      1.00      110  12.0   1   0.0        55       1    180     13
## Trix            1.00      110  13.0   1   0.0        25       1    140     12
## AppleJacks      1.00      110  11.0   0   1.0        30       2    125     14
## CornChex        1.00      110  22.0   0   0.0        25       2    280      3
## CornFlakes      1.00      100  21.0   0   1.0        35       2    290      2
## Nut&Honey       0.67      120  15.0   1   0.0        40       2    190      9
## Smacks          0.75      110   9.0   1   1.0        40       2     70     15
## MultiGrain      1.00      100  15.0   1   2.0        90       2    220      6
## CracklinOat     0.50      110  10.0   3   4.0       160       3    140      7
## GrapeNuts       0.25      110  17.0   0   3.0        90       3    179      3
## HoneyNutCheerios 0.75     110  11.5   1   1.5        90       3    250     10
## NutriGrain      0.67      140  21.0   2   3.0       130       3    220      7
## Product19       1.00      100  20.0   0   1.0        45       3    320      3
## TotalRaisinBran 1.00      140  15.0   1   4.0       230       3    190     14
## WheatChex       0.67      100  17.0   1   3.0       115       3    230      3
## Oatmeal         0.50      130  13.5   2   1.5       120       3    170     10
## Life            0.67      100  12.0   2   2.0        95       4    150      6
## Maypo           1.00      100  16.0   1   0.0        95       4      0      3
## QuakerOats      0.50      100  14.0   1   2.0       110       4    135      6
## Muesli          1.00      150  16.0   3   3.0       170       4    150     11
## Cheerios        1.25      110  17.0   2   2.0       105       6    290      1
## SpecialK        1.00      110  16.0   0   1.0        55       6    230      3
```

Cereals Data

Download the data in R using the package `RCurl` as follows:

```r
# if you don't have RCurl installed
install.packages("RCurl")

# load package RCurl
library(RCurl)

# google docs spreadsheets url
google_docs = "https://docs.google.com/spreadsheet/"

# public key of data 'cereals'
cereals_key = "pub?key=0AjoVnZ9iB261dEljWDZiWS1reVh1UFNKdE5EcVRvSkE&output=csv"

# download URL of data file
cereals_csv = getURL(paste(google_docs, cereals_key, sep = ""))

# import data in R (through a text connection)
cereals = read.csv(textConnection(cereals_csv), row.names = 1, header = TRUE)
```

Cereals Data

Structure of data cereals

```
# check structure
str(cereals, vec.len = 1)

## 'data.frame': 23 obs. of  9 variables:
##  $ Cups     : num  0.75 1 ...
##  $ Calories : int  120 110 ...
##  $ Carbs    : num  12 12 ...
##  $ Fat      : int  2 1 ...
##  $ Fiber    : num  0 0 ...
##  $ Potassium: int  35 55 ...
##  $ Protein  : int  1 1 ...
##  $ Sodium   : int  220 180 ...
##  $ Sugars   : int  12 13 ...
```

Example: Cereals Data Set

Variables in Cereals

$X_1 = \texttt{Cups}$

$X_2 = \texttt{Calories}$

$\vdots$

$X_8 = \texttt{Sodium}$

$X_9 = \texttt{Sugars}$

Objects in Cereals

$obj_1 = \texttt{CapnCrunch}$

$obj_2 = \texttt{CocoaPuffs}$

$\vdots$

$obj_{22} = \texttt{Cheerios}$

$obj_{23} = \texttt{SpecialK}$

Matrix Structure

Data

The analyzed data can be expressed in matrix format $\mathbf{X}$:

$$\mathbf{X}_{n,p} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

- n objects in the rows
- p quantitative variables in the columns

Variables Considerations

Variables

We will denote the p variables in $\mathbf{X}$ by $X_1, X_2, \dots, X_p$

Mean-centered

We will assume that the data is centered

$$\bar{X}_j = \sum_{i=1}^{n} x_{ij} = 0$$

(i.e. *centered*: variables with mean $= 0$)

Mean-centered Data

Centered Data

For convenient reasons —to make computations easier and simplify notation— it is typically assumed that the variables are **mean-centered** which means that variables have mean $= 0$.

We often refer to this assumption by simply saying that the data is centered.

Centering Variables in Cereals Data

Centered data `cereals` using function `scale()`

```
# mean-centering variables
X = scale(cereals, center = TRUE, scale = FALSE)

# check mean values of variables
colMeans(X)

##        Cups   Calories      Carbs        Fat      Fiber  Potassium    Protein
##   2.896e-17  2.471e-15  6.951e-16  8.689e-17 -4.827e-17  3.707e-15 -5.792e-17
##      Sodium     Sugars
## -2.471e-15  3.475e-16
```

More about the variables

Variables assumption

When performing PCA, there's an implicit assumption that no single feature or variable is more important than any other.

Interdependent Variables

This implies that we make no distinctions of variables as being preditors or predictands. Some authors say that all variables are interdependent.

Looking at Rows and Columns

Data Concerns

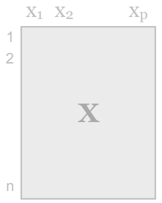
Two sides of the same coin

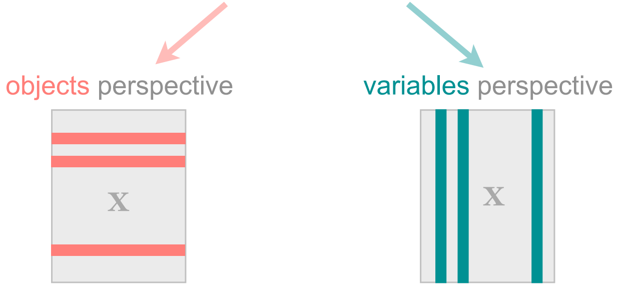
When the analyzed data can be expressed as a matrix with objects in rows, and variables in columns, we commonly care for two issues:

- Study the resemblance between objects
- Study the relationships among variables

Data Perspectives

looking at a data matrix from two perspectives





Data Matrix Products

Matrix Products

There are two fundamental matrix products that play a crucial role when the data is in a matrix X with objects in rows, and variables in columns:

- $X'X$ association matrix for the variables
- XX' association matrix for the objects

(keep in mind we are assuming centered data)

Association Matrix of Variables

Association between Variables

Having data in a matrix X with n rows and p variables, the association matrix for variables $X'X$ is a matrix of size $p \times p$

Cross-Products of Variables

This matrix contains the cross-products between all pairs of variables. Moreover, this matrix represents the relationships between variables. It is the matrix containing the information of the variances and covariances.

Cross-Products among variables

```
# association matrix of variables
assoc_variables = t(X) %*% X

# dimensions
dim(assoc_variables)

## [1] 9 9

# first 5 rows and 5 columns
assoc_variables[1:5,1:5]

##             Cups Calories   Carbs      Fat   Fiber
## Cups      1.2779   -1.713   4.402  -0.8791  -2.395
## Calories -1.7130 4486.957  56.957 140.8696 140.435
## Carbs     4.4020   56.957 284.457 -24.6304   1.935
## Fat      -0.8791  140.870 -24.630  18.6087  10.804
## Fiber    -2.3946  140.435   1.935  10.8043  37.152
```

Association Matrix of Objects

Associations between Objects

Having data in a matrix X with n rows and p variables, the association matrix for variables XX' is a matrix of size $n \times n$

Cross-Products of Objects

This matrix contains the cross-products between all pairs of objects. Moreover, this matrix represents the relationships between objects.

Cross-products between objects

```
# association matrix of objects
assoc_objects = X %*% t(X)

# dimensions
dim(assoc_objects)

## [1] 23 23

# first 5 rows and 5 columns
assoc_objects[1:5, 1:5]

##            CapnCrunch CocoaPuffs   Trix AppleJacks CornChex
## CapnCrunch     3619.3     1325.5 1655.4      958.9   5791.0
## CocoaPuffs     1325.5     1133.8 2462.7     2469.2    990.3
## Trix           1655.4     2462.7 6293.5     6746.1   -760.9
## AppleJacks      958.9     2469.2 6746.1     7459.6  -2442.3
## CornChex       5791.0      990.3 -760.9    -2442.3  11948.7
```

Variation in Data

Total Variation in Data

Variation and Inertia

A major concept is that of **total variation** in data, also known as total inertia

About Inertia

Inertia reflects the total amount of dispersion (variability) in the data. Two ways to compute inertia using association matrices:

- $trace(X'X)$
- $trace(XX')$

Inertia

The inertia is a numeric value that indicates the amout of dispersion in a dataset. Think of it as a generalization of the concept of variance.

By itself the inertia doesn't tell us much, but it's important to know that it can be calculated from both association matrices (objects and variables)

```
# inertia
sum(diag(assoc_variables))

## [1] 188340

# inertia
sum(diag(assoc_objects))

## [1] 188340
```

Matrix Decompositions

Decompositions

Decompositions

Matrix decompositions, also known as matrix factorizations, are another topic that plays a relevant role. They allow us to express a matrix $\mathbf{Y}$ as the product of a number of simpler matrices —usually 2 or 3—:

$$\mathbf{Y} = \mathbf{AB}$$

or

$$\mathbf{Y} = \mathbf{ABC}$$

Data Matrix Products

What for?

A matrix decomposition is just a means of expressing a matrix as a product of two or more simpler matrices.

EVD and SVD

There are many types of matrix decompositions but for our PCA understanding purposes, we are interested in two decompositions:

- Eigen-Value Decomposition (EVD)
- Singular Value Decomposition (SVD)

EVD

Eigenvalue Decomposition

This decomposition applies to **symmetric** matrices such as the cross-product association matrices $X'X$ and XX'

Symmetric Matrices

The attractive thing about EVD is that when applied to symmetric matrices the results have a “simple” nice structure.

Eigen-Value Decomposition

EVD

An $n \times n$ symmetric matrix $\mathbf{Y}$ can be decomposed as:

$$\mathbf{Y} = \mathbf{U \Lambda U'}$$

where

- $\mathbf{U}$ is a $n \times p$ column **orthonormal** matrix containing the eigen-vectors of $\mathbf{Y}$
- $\mathbf{\Lambda}$ is a $p \times p$ **diagonal** matrix containing the eigen-values of $\mathbf{Y}$

EVD in R

`eigen()` function

R provides the function `eigen()` to perform a eigen-value decomposition of a given matrix

`eigen()` output

A list with the following components

- `values` a vector containing the eigen-values
- `vectors` a matrix whose columns contain the eigen-vectors

EVD example in R

```r
# create symmetric matrix Y
set.seed(22)
M = matrix(rnorm(20), 5, 4)
Y = t(M) %*% M

# eigen-value decomposition
EVD = eigen(Y)

# elements returned by svd()
names(EVD)
```

```
## [1] "values"  "vectors"
```

```r
# vector of singular values
(lambda = EVD$values)
```

```
## [1] 15.615  4.090  2.175  0.187
```

```r
# matrix of eigen-vectors
(U = EVD$vectors)
```

```
##         [,1]    [,2]     [,3]    [,4]
## [1,]  0.5708  0.7407 -0.33863  0.1043
## [2,] -0.2742  0.5295  0.76797  0.2338
## [3,]  0.2772 -0.3206  0.04462  0.9046
## [4,]  0.7226 -0.2612  0.54181 -0.3408
```

```r
# U osrthonormal (U'U = I)
t(U) %*% U
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,]  1.000e+00 -3.331e-16 -1.110e-16 -2.776e-17
## [2,] -3.331e-16  1.000e+00  3.608e-16  1.665e-16
## [3,] -1.110e-16  3.608e-16  1.000e+00  5.274e-16
## [4,] -2.776e-17  1.665e-16  5.274e-16  1.000e+00
```

EVD example in R (con't)

```
# Y equals U L U'
U %*% diag(lambda) %*% t(U)
```

```
##        [,1]   [,2]   [,3]   [,4]
## [1,]  7.584 -1.401  1.485  5.244
## [2,] -1.401  3.614 -1.767 -2.769
## [3,]  1.485 -1.767  1.778  3.466
## [4,]  5.244 -2.769  3.466  9.092
```

```
# compare to Y
Y
```

```
##        [,1]   [,2]   [,3]   [,4]
## [1,]  7.584 -1.401  1.485  5.244
## [2,] -1.401  3.614 -1.767 -2.769
## [3,]  1.485 -1.767  1.778  3.466
## [4,]  5.244 -2.769  3.466  9.092
```

SVD

Singular Value Decomposition

This decomposition applies for any rectangular matrix, meaning that we can apply SVD to any data table X.

Principal Axis

The real importance behind SVD comes from the SVD

- Eigen-Value Decomposition (EVD)
- Singular Value Decomposition (SVD)

Singular Value Decomposition

SVD

An $n \times p$ matrix $\mathbf{X}$ can be decomposed as:

$$\mathbf{X} = \mathbf{U \Lambda V}'$$

where

- $\mathbf{U}$ is a $n \times p$ column **orthonormal** matrix containing the left singular vectors
- $\mathbf{\Lambda}$ is a $p \times p$ **diagonal** matrix containing the singular values of $\mathbf{X}$
- $\mathbf{V}$ is a $p \times p$ column **orthonormal** matrix containing the right singular vectors

`svd()` in R

`svd()` function

R provides the function `svd()` to perform a singular value decomposition of a given matrix

`svd()` output

A list with the following components

- `d` a vector containing the singular values
- `u` a matrix whose columns contain the left singular vectors
- `v` a vector whose columns contain the right singular vectors

SVD example in R

```r
# X matrix
set.seed(22)
X = matrix(rnorm(20), 5, 4)

# singular value decomposition
SVD = svd(X)

# elements returned by svd()
names(SVD)
```

```
## [1] "d" "u" "v"
```

```r
# vector of singular values
(d = SVD$d)
```

```
## [1] 3.9516 2.0224 1.4748 0.4324
```

```r
# matrix of left singular vectors
(U = SVD$u)
```

```
##         [,1]     [,2]    [,3]      [,4]
## [1,] -0.4251 -0.53913 -0.7233  0.009794
## [2,]  0.5269 -0.76863  0.2860  0.056100
## [3,]  0.5753  0.05000 -0.4421  0.131072
## [4,]  0.2215  0.05273 -0.1702 -0.951234
## [5,] -0.4021 -0.33655  0.4131 -0.273371
```

```r
# matrix of right singular vectors
(V = SVD$v)
```

```
##         [,1]    [,2]     [,3]    [,4]
## [1,]  0.5708 -0.7407  0.33863  0.1043
## [2,] -0.2742 -0.5295 -0.76797  0.2338
## [3,]  0.2772  0.3206 -0.04462  0.9046
## [4,]  0.7226  0.2612 -0.54181 -0.3408
```

SVD example in R (con't)

```r
# U orthonormal  (U'U = I)
t(U) %*% U
```

```
##              [,1]       [,2]       [,3]       [,4]
## [1,] 1.000e+00  1.388e-16  2.776e-17  0.000e+00
## [2,] 1.388e-16  1.000e+00 -2.776e-17 -8.327e-17
## [3,] 2.776e-17 -2.776e-17  1.000e+00  5.551e-17
## [4,] 0.000e+00 -8.327e-17  5.551e-17  1.000e+00
```

```r
# V orthonormal  (V'V = I)
t(V) %*% V
```

```
##               [,1]       [,2]       [,3]       [,4]
## [1,]  1.000e+00 -1.110e-16 -5.551e-17  1.110e-16
## [2,] -1.110e-16  1.000e+00  8.327e-17  1.943e-16
## [3,] -5.551e-17  8.327e-17  1.000e+00 -8.327e-17
## [4,]  1.110e-16  1.943e-16 -8.327e-17  1.000e+00
```

```r
# X equals U D V'
U %*% diag(d) %*% t(V)
```

```
##         [,1]     [,2]     [,3]    [,4]
## [1,] -0.5121  1.85809 -0.76391 -0.9222
## [2,]  2.4852 -0.06603  0.08196  0.8616
## [3,]  1.0078 -0.16276  0.74303  2.0029
## [4,]  0.2928 -0.19986 -0.08402  0.9366
## [5,] -0.2090  0.30056 -0.79289 -1.6157
```

```r
# compare to X
X
```

```
##         [,1]     [,2]     [,3]    [,4]
## [1,] -0.5121  1.85809 -0.76391 -0.9222
## [2,]  2.4852 -0.06603  0.08196  0.8616
## [3,]  1.0078 -0.16276  0.74303  2.0029
## [4,]  0.2928 -0.19986 -0.08402  0.9366
## [5,] -0.2090  0.30056 -0.79289 -1.6157
```