post02-yanisa-cheeppensuk

Yanisa

11/25/2017

Linear Algebra and Principal Component Analysis

Introduction

As we approach the end of the end of this course, I couldn't help but notice some functions in R which are directly related to linear algebra: matrix, row and column operations, eigenvalues etc. Upon further research, I found that R also has many built-in operations that could ease matrix operations. In this post, I will discuss some of these operations as well as go through some simple linear algebra examples and how it relates to Principal Component Analysis (PCA).

{base_r} Matrix Functions

Let A and B be matrices with dimensions such that operations below are defined.

- `A * B` : element-wise multiplication
- `A %*% B` : matrix multiplication
- `t(A)` : transpose of matrix A
- `diag(A)` : returns a diagonal matrix with elements of A on the diagonal. If k is scalar, `diag(k)` returns a k x k identity matrix
- `det(A)` : returns the determinant of the matrix
- `solve(A,b)` : solves for the vector x in the equation Ax = b
- `solve(A)` : returns A-inverse

```
# let A and B be 2x2 matrices for easy examples
A <- matrix(c(1,2,3,4), nrow = 2, ncol = 2, byrow = TRUE)
B <- matrix(c(1,1,1,1), nrow = 2, ncol = 2, byrow = TRUE)

A * B # element-wise multiplication
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
A %*% B # matrix multiplication
```

```
##      [,1] [,2]
## [1,]    3    3
## [2,]    7    7
```

```
t(A) # A transposed
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
diag(2) # a 2x2 identity matrix
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
det(A) # returns determinant of matrix A
```

```
## [1] -2
```

```
b <- c(0,1)
solve(A,b) # solves the system of equations
```

```
## [1]  1.0 -0.5
```

```
solve(A) # finds A-inverse
```

```
##      [,1] [,2]
## [1,] -2.0  1.0
## [2,]  1.5 -0.5
```

Matrix Diagonalization

> Let A and be a square matrix. A is diagonalizable if and only if A is invertible. A can therefore be expressed as a

> product of a digonal matrix D with its eigenvalues in the diagonal and a matrix P with its corresponding eigenvectors in columns. $A = P \cdot D \cdot P^{-1}$

R has a built-in function `eigen()` which returns a list of eigenvalues and eigenvectors. We will now build a function that diagonalizes a square matrix and returns matrices D and P.

Let's take a look at the function `eigen()` applied onto a 3x3 matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

```
# A is a 3x3 matrix
A = matrix(c(1,1,1,
             0,1,0,
             0,1,2), nrow = 3, ncol = 3, byrow = TRUE)
y <- eigen(A)

# y is a list with two elements: first element is a vector of eigenvalues and
# second element is a matrix of eigenvectors
y
```

```
## eigen() decomposition
## $values
## [1] 2 1 1
##
## $vectors
##           [,1] [,2]       [,3]
## [1,] 0.7071068    1  0.0000000
## [2,] 0.0000000    0  0.7071068
## [3,] 0.7071068    0 -0.7071068
```

In this case, the eigenvalues of A are 2,1,1 and the eigenvectors are $\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$, scaled to integers.

Now, we will write a function to diagonalize A. `diagonal()` will return a list of three matrices, D for the diagonal matrix, P for the invertible matrix with eigenvectors as its columns and Pin for the inverse matrix of P. Together, matrix multiplication of the three matrices will be equal to A. It will throw an error with the message, 'non-diagonalizable matrix' if the operation cannot be carried out, i.e. determinant of the matrix is zero.

```
diagonal <- function(A) {
  # First, check if the matrix is diagonalizable, if not, throw an error
  if (det(A) == 0) {stop('non-diagonalizable matrix')}

  D <- matrix(c(0), ncol = ncol(A), nrow = nrow(A), byrow = FALSE)
  P <- matrix(c(0), ncol = ncol(A), nrow = nrow(A), byrow = FALSE)
  Pin <- matrix(c(0), ncol = ncol(A), nrow = nrow(A), byrow = FALSE)

  # use function `eigen` to find the eigenvalues and eigenvectors of A
  y <- eigen(A)
  D <- diag(y[[1]])
  P <- y[[2]]
  Pin <- solve(y[[2]])
  list(D = D, P = P, Pin = Pin)
}

diagonal(A)
```

```
## $D
##      [,1] [,2] [,3]
## [1,]    2    0    0
## [2,]    0    1    0
## [3,]    0    0    1
##
## $P
##           [,1] [,2]       [,3]
## [1,] 0.7071068    1  0.0000000
## [2,] 0.0000000    0  0.7071068
## [3,] 0.7071068    0 -0.7071068
##
## $Pin
##      [,1]      [,2]      [,3]
## [1,]    0  1.414214  1.414214
## [2,]    1 -1.000000 -1.000000
## [3,]    0  1.414214  0.000000
```

```
# We can check if the diagonalization is done correctly
A == diagonal(A)$P %*% diagonal(A)$D %*% diagonal(A)$Pin
```

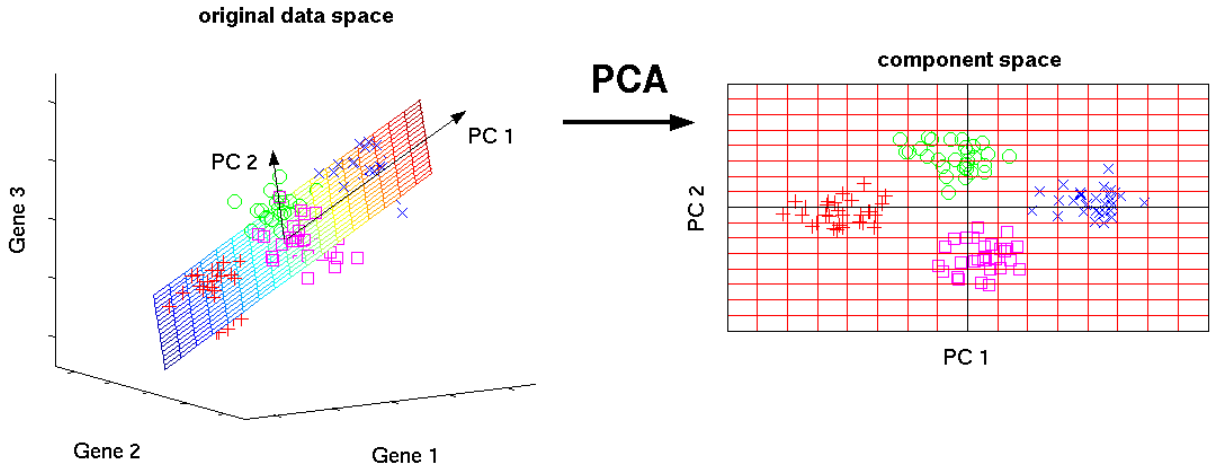
```
##      [,1] [,2] [,3]
## [1,] TRUE TRUE TRUE
## [2,] TRUE TRUE TRUE
## [3,] TRUE TRUE TRUE
```

In real life, diagonalization allows us to raise matrices to a power quickly and accurately. This is already built into R however, diagonalization still has important applications for Principle Component Analysis (PCA) which we have touched upon briefly in class and maching learning. For further reading on eigenvalues, PCA and Machine Leaning, follow this thread for a brief introducion on the 'Curse of Dimensionality': https://www.quora.com/What-is-the-curse-of-dimensionality.

Let's now take a closer look at PCA.

Principal Component Analysis (PCA)

PCA allows us to assess many variables at once by reducing the dimensionality of the data set. In simpler terms, assessing how two variables are related to each other is easy because we can plot data points onto a 2D graph, three variables is slightly trickier but we can still plot them onto a 3D graph - data visualization forms which we can physically assess. What happens when we want to observe more than three variables at once? We have to reduce its 'dimensions' so we can still plot them onto a data plot which we can actually see. We typically attempt to plot them onto a 2D graph. In the diagram below, a 3D data set is compressed onto a 2D representation.



Source: https://www.linkedin.com/pulse/make-predictions-test-data-set-using-principal-r-manish-saraswat

To provide better intuition as to how PCA works, we will be looking at PCA from a linear algebra perspective.

The axis PC1 is the eigenvector with the largest eigenvalue - this represents the axis in the data set with the largest covariance. PC2 is the second largest eigenvalue and so on… The intuition behind this is that, the eigenvector with the largest eigenvalue will be most impacted when there is a small change along that axis. Similarly, when two variables are very related, when one changes, the other chagnes significantly. To study the 'relatedness' of variables, we look at the covariance between each pair of variables.

Eigenvalues and eigenvectors are computed from the covariance matrix of the data set. Let's consider the Iris data set - it has five columns (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width and Species) and 150 instances.

> Covariance Matrix is a matrix whose i,jth element is the covariance between the ith and jth variables in the data set. Since the covariance between the ith and jth element is equal to that between the jth and ith element, the Covariance Matrix is symmetric.

`iris0` is the matrix of the Iris data set with the mean of each column subtracted from the entries of each column.

```
iris0 <- cbind(SLength0 = iris$Sepal.Length - mean(iris$Sepal.Length),
               SWidth0 = iris$Sepal.Width - mean(iris$Sepal.Width),
               PLength0 = iris$Petal.Length - mean(iris$Petal.Length),
               PWidth0 = iris$Petal.Width - mean(iris$Petal.Width))
head(iris0) # mean of each column is now 0
```

```
##        SLength0     SWidth0 PLength0    PWidth0
## [1,] -0.7433333  0.44266667   -2.358 -0.9993333
## [2,] -0.9433333 -0.05733333   -2.358 -0.9993333
## [3,] -1.1433333  0.14266667   -2.458 -0.9993333
## [4,] -1.2433333  0.04266667   -2.258 -0.9993333
## [5,] -0.8433333  0.54266667   -2.358 -0.9993333
## [6,] -0.4433333  0.84266667   -2.058 -0.7993333
```

To find the covariance matrix, apply the `cov()` function to X:

```
coviris <- cov(iris0)
coviris # symmetric matrix with variance of the variables along the diagonal
```

```
##              SLength0     SWidth0    PLength0     PWidth0
## SLength0   0.6856935 -0.0424340  1.2743154  0.5162707
## SWidth0   -0.0424340  0.1899794 -0.3296564 -0.1216394
## PLength0   1.2743154 -0.3296564  3.1162779  1.2956094
## PWidth0    0.5162707 -0.1216394  1.2956094  0.5810063
```

Now we can find the eigenvalues and eigenvectors of this matrix using the function `eigen`.

```
coviris_eigen <- eigen(coviris)
coviris_eigen
```

```
## eigen() decomposition
## $values
## [1] 4.22824171 0.24267075 0.07820950 0.02383509
##
## $vectors
##              [,1]        [,2]        [,3]       [,4]
## [1,]  0.36138659 -0.65658877 -0.58202985  0.3154872
## [2,] -0.08452251 -0.73016143  0.59791083 -0.3197231
## [3,]  0.85667061  0.17337266  0.07623608 -0.4798390
## [4,]  0.35828920  0.07548102  0.54583143  0.7536574
```

Compare this with the built-in function `prcomp` which calculates all the Principal Components in a data set:

```
# iris[1:4] are the numeric columns of iris
# look at the second element $rotation for the Principal Component Values
PCA <- prcomp(iris[1:4])
PCA$rotation
```

```
##                      PC1         PC2         PC3        PC4
## Sepal.Length  0.36138659 -0.65658877  0.58202985  0.3154872
## Sepal.Width  -0.08452251 -0.73016143 -0.59791083 -0.3197231
## Petal.Length  0.85667061  0.17337266 -0.07623608 -0.4798390
## Petal.Width   0.35828920  0.07548102 -0.54583143  0.7536574
```

Evidently, PC1 is a numeric vector equivalent to the eigenvector with the largest eigenvalue obtained from the covariance matrix. PC2 corresponds to the eigenvector with the second largest eigenvalue and so on…

Conclusion

The function `prcomp` in R efficiently calculates the PCA for us, however, to understand the inner workings of PCA, we need to understand linear algebra concepts such as eigenvalues and eigenvectors. Some might find this mathematically challenging but R also has tools which can make Principal Component visualization easier, otherwise, this website is also helpful in understanding how PCA works: http://setosa.io/ev/principal-component-analysis/

References

- https://www.statmethods.net/advstats/matrix.html
- http://setosa.io/ev/eigenvectors-and-eigenvalues/
- https://www.quora.com/What-is-the-curse-of-dimensionality
- https://en.wikipedia.org/wiki/Curse_of_dimensionality
- https://en.wikipedia.org/wiki/Principal_component_analysis
- https://georgemdallas.wordpress.com/2013/10/30/principal-component-analysis-4-dummies-eigenvectors-eigenvalues-and-dimension-reduction/
- https://klevas.mif.vu.lt/~tomukas/Knygos/principal_components.pdf