Tutorial 7: Principal Component Analysis through R

Miss. Na Zhao

Outline

- PCA theory
- Singular Value Decomposition
- Manually PCA realization
- Built-in PCA Functions
- Functional PCA

PCA theory

what is PCA ?

PCA is a type of linear transformation on a given data set that has values for a certain number of variables (coordinates) for a certain amount of spaces. This linear transformation fits this dataset to a new coordinate system in such a way that the most significant variance is found on the first coordinate, and each subsequent coordinate is orthogonal to the last and has a lesser variance. In this way, you transform a set of x correlated variables over y samples to a set of p uncorrelated principal components over the same samples.

Mathematically, suppose a dataset $X_1, X_2, \ldots, X_n$ with sample covariance matrix S, X_i are i.i.d. from random vector X with covariance matrix Σ, which will be used in the derivation. The goal of the derivation is to find $\mathbf{a}^T$ that maximizes the variance of $\mathbf{a}^T X$. For this, we will consider the vector $\mathbf{a}^T$ that maximizes $Var(\mathbf{a}^T X) = \mathbf{a}^T \Sigma \mathbf{a}$ as the first principal component.

PCA theory

derivation of PCA

We consider the first principal components. To do the above maximization, we will need a constraint to rein in otherwise unnecessarily large values of $\mathbf{a}$. The constraint in this example is the unit length vector $\mathbf{a}^T\mathbf{a} = 1$, otherwise the maximum will be infinite. This constraint is employed with a Lagrange multiplier λ so that the function is maximized at an equality constraint of g(x)=0. Thus the Lagrangian function is defined as:

$$\mathbf{a}^T\Sigma\mathbf{a} - \lambda(\mathbf{a}^T\mathbf{a} - 1).$$

PCA theory

Brief Aside: Lagrange Multipliers

The Lagrange mulitiplier method is used for finding a maximum or minimum of a multivariate function with some constraint on the input values. As we are interested in maximization, the problem can be briefly stated as "maximize $f(x)$ subject to $g(x) = c$". here, $g(\mathbf{a}) = \mathbf{a}^T\mathbf{a} = 1$ and $f(\mathbf{a}) = \mathbf{a}^T\Sigma\mathbf{a}$. The Lagrange multiplier, defined as λ allows the combination of $f(\mathbf{a})$ and $g(\mathbf{a})$ into a new function $L(\mathbf{a}, \lambda)$, defined as:

$$L(\mathbf{a}, \lambda) = f(\mathbf{a}) - \lambda(g(\mathbf{a}) - 1).$$

The sign of λ can be positive or negative.

The new function is then solved through partial derivatives:

$$\frac{\partial L(\mathbf{a}, \lambda)}{\partial \mathbf{a}} = 2\Sigma\mathbf{a} - 2\lambda\mathbf{a} = 0,$$

indicates λ is an eigenvector of the covariance matrix Σ and $\mathbf{a}$ is the corresponding eigenvector.

PCA theory

Eigenvalues and Eigenvectors

In linear algebra, an eigenvector of a linear transformation is a nonzero vector that changes at most by a scalar factor when that linear transformation is applied to it. The corresponding eigenvalue, often denoted by λ, is the factor by which the eigenvector is scaled. In a finite-dimensional vector space V, one matrix A represents one linear transformation, and $\mathbf{v}$ is a nonzero vector in V, then $\mathbf{v}$ is an eigenvector of A if $A\mathbf{v}$ is a scalar multiple of $\mathbf{v}$. This can be written as

$$A\mathbf{v} = \lambda\mathbf{v}.$$

PCA theory

Important Property of symmetric matrix

- Eigenvector of different eigenvalues are linear independent with each others: suppose $A\mathbf{v}_i = \lambda_i \mathbf{v}_i, i = 1, 2$ and $\lambda_1 \neq \lambda_2$ are nonzero, then $\mathbf{v}_1^T \mathbf{v}_2 = 0$.

This property guarantees that the k-th principal component direction is automatically the eigenvector corresponding to the largest k eigenvalue. **Try to check with the Lagrange multiplier by yourself if interested!**

So the standard PCA prosedures are:

- Obtain the variance covariance matrix Σ (known or estimated by sample covariance matrix);
- Obtain the eigenvalues λ_k descendingly and corresponding eigenvectors $\mathbf{a}_k$.
- $\mathbf{a}_k X_i$ will be the k-th principal component projection of observation X_i.

PCA theory

Variance Accounts

- The total variance is the sum of variances of all individual principal components.
- The fraction of variance explained by a principal component is the ratio between the variance of that principal component and the total variance.
- For several principal components, add up their variances and divide by the total variance.

By definition, $Var(\mathbf{a}_k^T X) = \mathbf{a}_k^T \Sigma \mathbf{a}_k = \lambda_k \mathbf{a}_k^T \mathbf{a}_k = \lambda_k$, so the k-th principal component explains $\frac{\lambda_k}{\sum \lambda_i} \times 100\%$ of the total variance.

You can decide how many principal components to take by observing the distribution of eigenvalues.

Singular Value Decomposition

definition

A singular value decomposition takes a rectangular matrix A, where A is a n by p matrix, in which the n rows represents the number of observations, and the p columns represents the number of covariates. The SVD theorem states:

$$A_{n\times p} = U_{n\times n} S_{n\times p} V^T_{p\times p},$$

where $U_{n\times n}$, $V_{p\times p}$ are identical matrix and $S_{n\times p}$ is a diagonal matrix.
$s_i = S_{i,i}$ are known as the singular values of $A_{n\times p}$.

Relationship with Eigen-Decompostion

For any $A \in R^{n\times p}$, $A^T A$ and AA^T are both symmetric matrices. The eigenvectors of $A^T A$ (should be vector of length p) make up the columns of V, the eigenvectors of AA^T (should be vector of length n) make up the columns of U; and the singular values in S are square roots of eigenvalues from $A^T A$ and AA^T.

Singular Value Decomposition

application of SVD in PCA

Since Σ is estimated by $\frac{X^T X}{n-1}$, from the previous statement, we will find the eigenvectors of Σ are the singular vectors of X (since they are all scaled to be of length 1), while the eigenvalues λ_k of Σ and the singular values s_k of X satisfying $\lambda_k / s_k^2 = \frac{1}{n-1}$.

Remember to centralized X before calculation, otherwise the relationship does not hold anymore!

Advantages and Disadvantages

- SVD does not need to calculate $X^T X$, can improve accuracy and stability. (Eg. Lauchli Matrix)
- In most case, computational complexity of standard PCA is $O(np^2 + p^3)$, SVD depends on the construction of matrix, but generally much faster.

Manually PCA realization

-Use dataset *USArrests*, which contains four variables that represent the number of arrests per 100,000 residents for *Assault*, *Murder*, and *Rapein* each of the fifty US states in 1973. The data set also contains the percentage of the population living in urban areas, *UrbanPop*. In addition to loading the set, we'll also use a few packages that provide added functionality in graphical displays and data manipulation.

```
library(tidyverse)  # data manipulation and visualization
library(gridExtra)  # plot arrangement
data("USArrests")
head(USArrests, 5)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
```

Manually PCA realization

- obtain the eigenvectors of the covariance matrix

```
scaled_df <- apply(USArrests, 2, scale)# scaling and centerinf the ol
arrests.cov <- cov(scaled_df) # obtain the covariance matrix
arrests.eigen <- eigen(arrests.cov)
arrests.eigen
```

```
## eigen() decomposition
## $values
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
##
## $vectors
##            [,1]       [,2]       [,3]        [,4]
## [1,] -0.5358995  0.4181809 -0.3412327  0.64922780
## [2,] -0.5831836  0.1879856 -0.2681484 -0.74340748
## [3,] -0.2781909 -0.8728062 -0.3780158  0.13387773
## [4,] -0.5434321 -0.1673186  0.8177779  0.08902432
```

Manually PCA realization

- Obtain the proportion of variance explained (PVE) for the k-th principal component.
- By plotting PVE and cumulative PVE, we take the first two sets of loadings and store them in the matrix *phi*

```
PVE <- arrests.eigen$values / sum(arrests.eigen$values)
round(PVE, 2)
```

```
## [1] 0.62 0.25 0.09 0.04
```

```
cumsum(PVE)
```

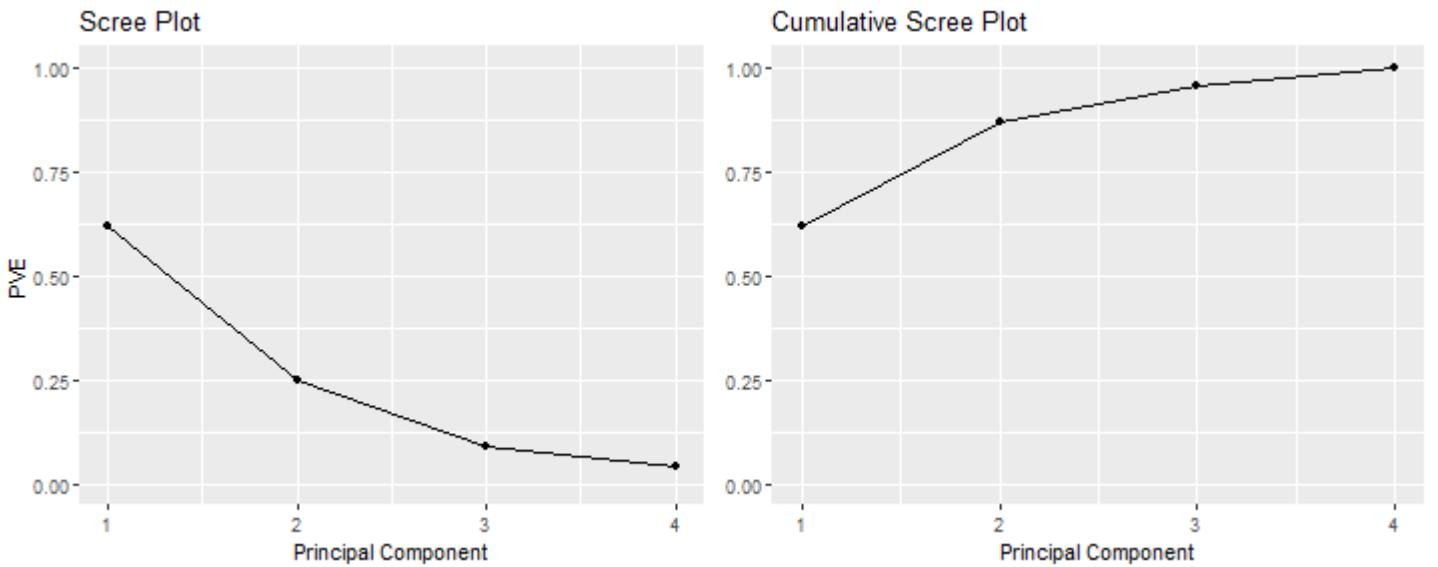
```
## [1] 0.6200604 0.8675017 0.9566425 1.0000000
```

```
phi <- arrests.eigen$vectors[,1:2] # Extract the loadings
```

Manually PCA realization

- We can directly plot PVE and cumulative PVE for visualization.

```
# PVE plot
PVEplot <- qplot(c(1:4), PVE) +  geom_line() +  xlab("Principal Compo
# Cumulative PVE plot
cumPVE <- qplot(c(1:4), cumsum(PVE)) +  geom_line() + xlab("Principal
grid.arrange(PVEplot, cumPVE, ncol = 2)
```



Manually PCA realization

Eigenvectors that are calculated in any software package are unique up to a sign flip. By default, eigenvectors in *R* point in the negative direction. For this example, we'd prefer the eigenvectors point in the positive direction because it leads to more logical interpretation of graphical results as we'll see shortly. To use the positive-pointing vector, we multiply the default loadings by -1. The set of loadings for the first principal component (PC1) and second principal component (PC2) are shown below:

```
phi <- -phi
row.names(phi) <- c("Murder", "Assault", "UrbanPop", "Rape")
colnames(phi) <- c("PC1", "PC2")
phi
```

```
##                PC1        PC2
## Murder   0.5358995 -0.4181809
## Assault  0.5831836 -0.1879856
## UrbanPop 0.2781909  0.8728062
## Rape     0.5434321  0.1673186
```

Each principal component vector defines a direction in feature space.

Manually PCA realization

- Projection the original data in two PC directions

```
PC1 <- as.matrix(scaled_df) %*% phi[,1]
PC2 <- as.matrix(scaled_df) %*% phi[,2]
# Create data frame with Principal Components scores
PC <- data.frame(State = row.names(USArrests), PC1, PC2)
head(PC)
```

```
##        State        PC1        PC2
## 1    Alabama  0.9756604 -1.1220012
## 2     Alaska  1.9305379 -1.0624269
## 3    Arizona  1.7454429  0.7384595
## 4   Arkansas -0.1399989 -1.1085423
## 5 California  2.4986128  1.5274267
## 6   Colorado  1.4993407  0.9776297
```

Manually PCA realization

- Plot Principal Components for each State

```
ggplot(PC, aes(PC1, PC2)) +
  modelr::geom_ref_line(h = 0) +
  modelr::geom_ref_line(v = 0) +
  geom_text(aes(label = State), size = 3) +
  xlab("First Principal Component") +
  ylab("Second Principal Component") +
  ggtitle("First Two Principal Components of USArrests Data")
```

Built-in PCA Functions

- Using above process to repeatedly perform PCA may be a bit tedious. Fortunately R has several built-in functions (along with numerous add-on packages) that simplifies performing PCA. One of these built-in functions is *prcomp*. The *prcomp* function centers the variables to have mean zero by default.

```
library(stats)
pca_result <- prcomp(USArrests, scale = TRUE)
summary(pca_result)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.5749 0.9949 0.59713 0.41645
## Proportion of Variance 0.6201 0.2474 0.08914 0.04336
## Cumulative Proportion  0.6201 0.8675 0.95664 1.00000
```

try to find the difference when scale=0

Built-in PCA Functions

- The output from *prcomp* contains a number of useful quantities.
- The *center* and *scale* components correspond to the means and standard deviations of the variables that were used for scaling prior to implementing PCA.

```
names(pca_result)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

```
pca_result$center # means
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```
pca_result$scale  # standard deviations
```

```
##    Murder   Assault  UrbanPop      Rape
##  4.355510 83.337661 14.474763  9.366385
```

Built-in PCA Functions

- The *rotation* matrix provides the principal component loadings, each column contains the corresponding principal component loading vector.
- We can also obtain the principal components scores from our results as these are stored in the *x* list item of our results. However, we also want to make a slight sign adjustment to our scores to point them in the positive direction.

```
pca_result$rotation
```

```
##                   PC1        PC2        PC3         PC4
## Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
## Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
## UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
## Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
```

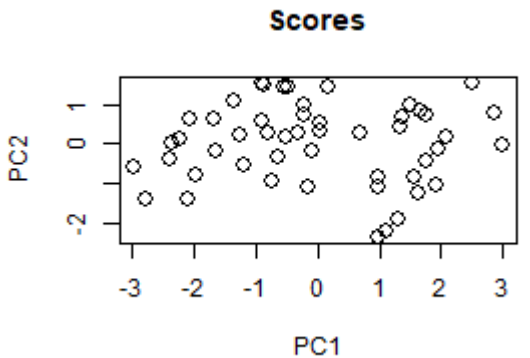
```
pca_result$rotation <- -pca_result$rotation
pca_result$x <- - pca_result$x
head(pca_result$x)
```

```
##                  PC1        PC2         PC3          PC4
## Alabama     0.9756604 -1.1220012  0.43980366 -0.154696581
## Alaska      1.9305379 -1.0624269 -2.01950027  0.434175454
```

Built-in PCA Functions

PCA plot

```
plot(pca_result$x[,1:2],
     pch=21, cex=1.5,    # point
     main="Scores"       # title
)
```




```
plot(pca_result$rotation,
     pch=21, cex=1.5,    # point
     main="Loadings"      # title
)
text(pca_result$rotation,
     labels=rownames(pca_result$
)
```

Built-in PCA Functions

- *sdev* stores the standard deviation of each principal component.
- Also indicates the variance explained for each pcs.

```
(VE <- pca_result$sdev^2)
```

```
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
```

```
PVE <- VE / sum(VE)
round(PVE, 2)
```

```
## [1] 0.62 0.25 0.09 0.04
```

```
cumsum(PVE)
```

```
## [1] 0.6200604 0.8675017 0.9566425 1.0000000
```

Built-in PCA Functions

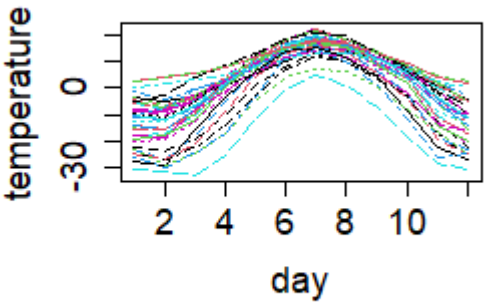
PCA biplot, including loading and score together

```
biplot(pca_result, scale = 0)
```

Functional PCA

- Deal with it as normal n by p dataset
- Using the canadian weather data in the package

```
library(fda)
tke=seq(1,365,length.out = 12) #choose 12 days of the year
tempfd=daily$tempav[tke,]
par(mfrow=c(1,1),mar = c(8, 8, 4, 2))
matplot(tempfd,xlab='day',ylab='temperature',cex.lab=1.5,cex.axis=1.5
```



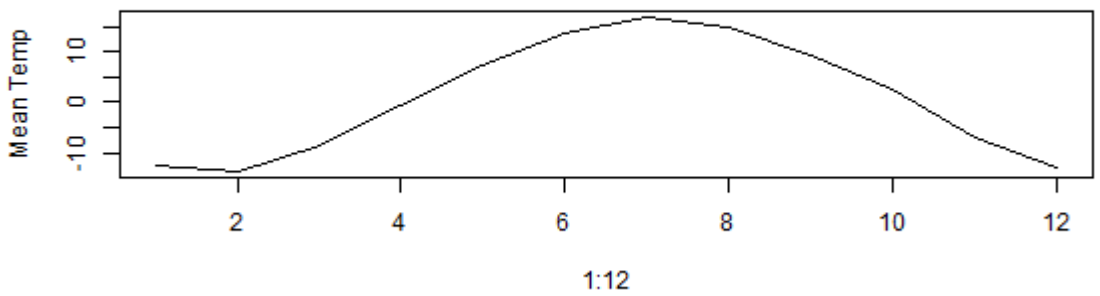
Functional PCA

- Do functional principal component analysis on the 35 temperature curves
- We choose 4 FPCs here

```
temppca = prcomp(t(tempfd),scale=TRUE)
temppca$scale  # standard deviations
```

```
##  [1] 9.112751 9.863217 9.207604 7.685911 5.527087 4.070011 3.881083 3.7206
##  [9] 4.684577 6.575368 9.072498 8.827342
```

```
plot(1:12,temppca$center,type="l",ylab="Mean Temp") # means
```



Functional PCA

- Check the direction of scores

```
head(temppca$x[,1:3])
```

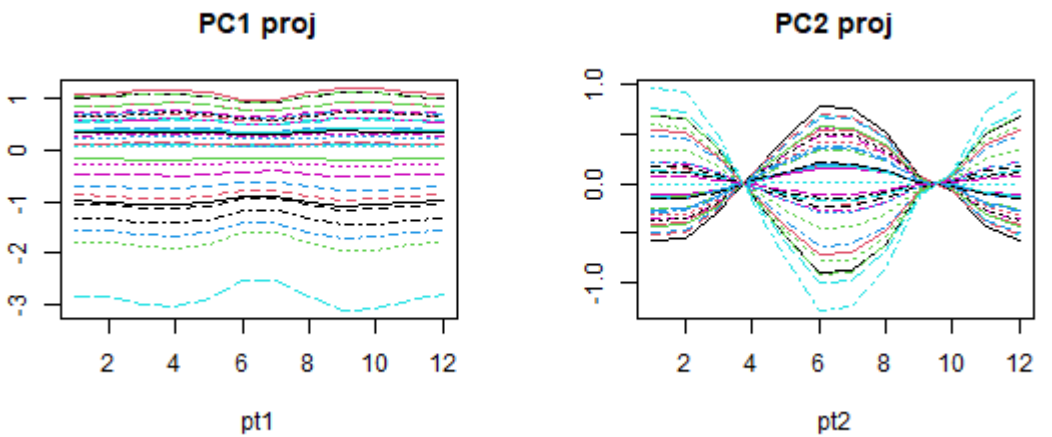
```
##                   PC1        PC2        PC3
## St. Johns   -1.162101  2.0013039 -0.2454867
## Halifax     -2.297653  0.4924047 -0.4663359
## Sydney      -1.938340  1.0015595 -0.6339339
## Yarmouth    -2.444384  1.4147753 -0.2159883
## Charlottvl  -1.916215  0.3744154 -0.5700962
## Fredericton -1.940858 -0.3509284 -0.2866007
```

```
temppca$x <- -temppca$x
temppca$rotation <- -temppca$rotation
```

Functional PCA

- plot the first FPC loadings.

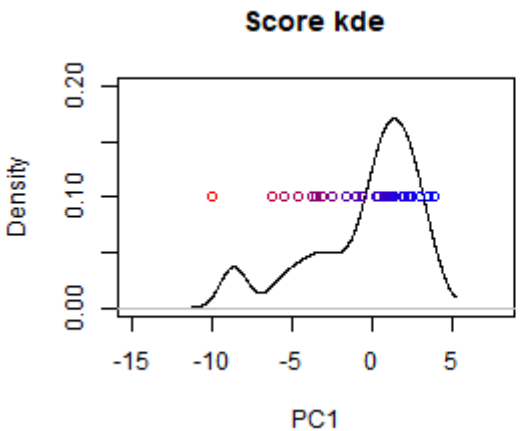
```
pt1=matrix(temppca$rotation[,1],ncol=1)%*%matrix(temppca$x[,1],nrow=1
pt2=matrix(temppca$rotation[,2],ncol=1)%*%matrix(temppca$x[,2],nrow=1
par(mfrow=c(1,2))
matplot(pt1,type="l",main="PC1 proj")
matplot(pt2,type="l",main="PC2 proj")
```

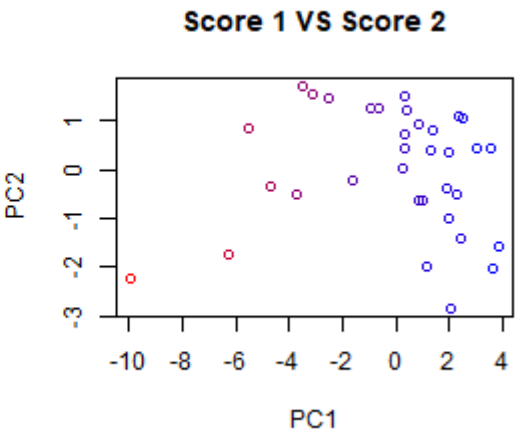


Functional PCA

- plot the first FPC scores, similar for other pcs plot.

```
library("kdensity")
kde = kdensity(temppca$x[,1], start = "gumbel", kernel = "gaussian")
rbPal <- colorRampPalette(c('red','blue'))
Col <- rbPal(10)[as.numeric(cut(temppca$x[,1],breaks = 10))]
par(mfrow=c(1,2))
plot(kde, main = "Score kde",xlim=c(-15,8),ylim=c(0,0.2),xlab="PC1")
points(temppca$x[,1],rep(0.1,35),col=Col)
plot(temppca$x[,1],temppca$x[,2], main = "Score 1 VS Score 2",xlab="P
```





Q&A