Yonsei University, Department of Applied Statistics

Chapter 10 실습

12기 YBIGTA 신보현

March 20, 2019

```
knitr::opts_chunk$set(comment=NA, fig.width=4, fig.height=4,fig.align='center',message=FALSE)
```

1. Principal Components Analysis

USArrests 데이터에 PCA를 수행해본다. PCA를 하기 이전에, 표준화를 반드시 진행해야, 스케일의 차이에 따른 차이를 방지할 수 있다. 옵션에 scale=TRUE을 설정함으로써 표준화 후에 PCA를 수행한다.

```
data = USArrests
apply(data,2,mean)
```

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

```
apply(data,2,var)
```

```
    Murder    Assault   UrbanPop       Rape
  18.97047 6945.16571  209.51878   87.72916
```

```
pca.result = prcomp(data,scale=TRUE)
names(pca.result)
```

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

center와 scale은 표준화 하기 이전의 변수들에 대한 평균과 표준오차를 의미한다. rotation은 principal component loadings matrix을 보여준다. 행렬의 각 칼럼은 principal component loading vector이다.

```
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
```

principal component score vectors을 얻기 위해서 데이터와 principal component loading vectors을 곱할 필요가 없다. PCA 결과의 x값에 principal component score vectors가 칼럼으로 있다. 다시 말해서 k번째 칼럼이 k번째 principal component score vector이다.

```
dim(pca.result$x)
```

```
[1] 50  4
```

```
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
Arizona    -1.7454429 -0.7384595  0.05423025 -0.826264240
Arkansas    0.1399989  1.1085423  0.11342217 -0.180973554
California -2.4986128 -1.5274267  0.59254100 -0.338559240
Colorado   -1.4993407 -0.9776297  1.08400162  0.001450164
```

즉, 예를 들어서 X 칼럼의 선형 결합으로 만들어진 first principal component은 $Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p$ 인데, 모든 관측치에 대해서 first principal component가 있으므로 이를 첨자 i번째 관측치에 대한 first principal component은 $z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + \cdots + \phi_{p1}x_{ip}$ 이다. 이 데이터에 대해서는 칼럼이 4개 이므로 $z_{i1} = \phi_{11}x_{i1} + \cdots + \phi_{41}x_{i4}$ 이다. 여기서 i번째 원래 데이터에 대응되는 x 값들을 대입하면 그 값이 바로 principal component scores이다. 즉, principal component loadings vector와 i번째 관측치들 간의 내적이 바로 principal component scores이다. 첫 번째 관측치의 첫번째 principal scores 값을 확인해보면 아래와 같다.

```
scale(data)[1,] %*% pca.result$rotation[,1]
```

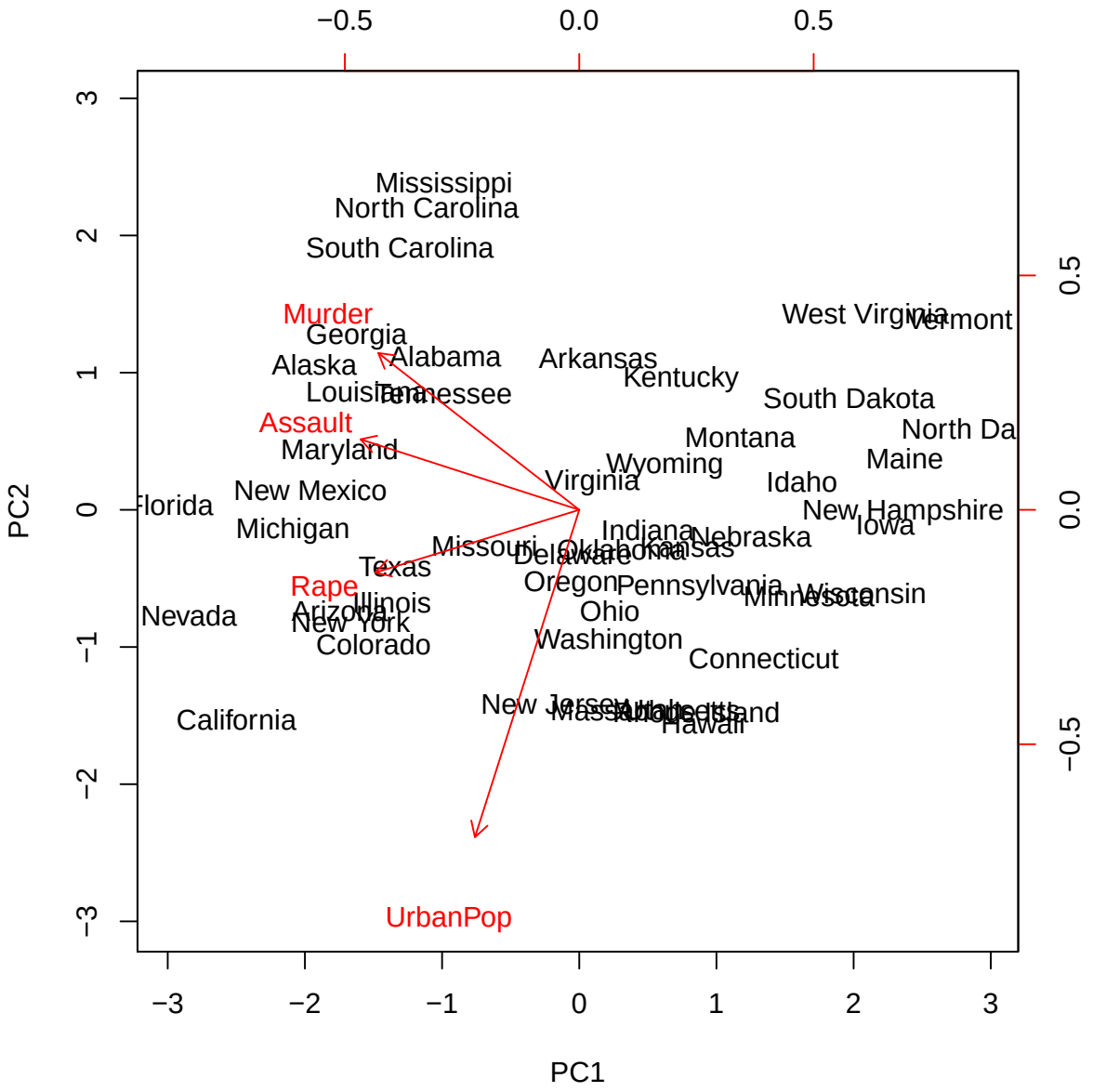
```
           [,1]
[1,] -0.9756604
```

```
pca.result$x[1,1]
```

```
[1] -0.9756604
```

이는 $\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \end{bmatrix} \begin{bmatrix} \phi_{11} \\ \phi_{21} \\ \phi_{31} \\ \phi_{41} \end{bmatrix}$ 을 한 것과 같다. 그리고 이 값을 pca.result$x의 (1,1) 요소와 비교해보면 값이 일치함을 알 수 있다.

```
biplot(pca.result,scale=0)
```



위 그림은 처음의 두 principal components에 대한 biplot이다. scale=0 옵션은 화살표가 loadings을 나타내도록 스케일 해준다.

sdev 값을 통해서 각각의 principal component의 standard deviation을 출력해준다.

```
pca.result$sdev
```

```
[1] 1.5748783 0.9948694 0.5971291 0.4164494
```

각각의 principal component에 의해서 설명된 분산은 이를 제곱함으로써 얻어진다.

```
pca.var = pca.result$sdev^2
pca.var
```

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

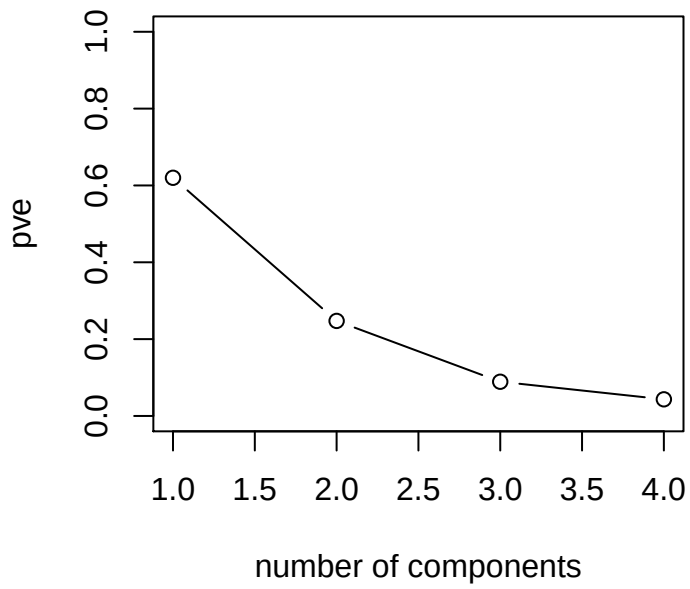
각 principal component에 의해서 설명되는 분산의 비율을 계산하기 위해서는 전체 분산 합으로 나누어주면 된다.

```
pve = pca.var/sum(pca.var)
pve
```

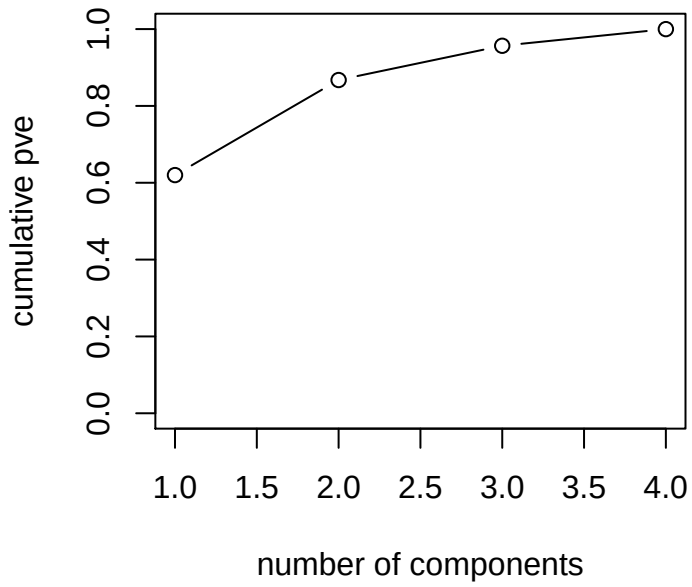
```
[1] 0.62006039 0.24744129 0.08914080 0.04335752
```

first principal component가 이 데이터의 분산 중 62%를 설명하고 second principal component가 24.7%정도를 설명하고 있음을 확인할 수 있다.
각각의 principal component에 대한 PVE 또는 누적 PVE 그림을 아래와 같이 그릴 수 있다.

```
plot(pve,xlab='number of components',type='b',ylim=c(0,1))
```



```r
plot(cumsum(pve),xlab='number of components',type='b',ylab='cumulative pve',ylim=c(0,1))
```



2. K-Means Clustering

우선 두 개의 군집을 가지도록 데이터를 생성하고 K=2로 하여 k-means를 시행해보자.

```r
set.seed(2)
x = matrix(rnorm(50*2),ncol=2)
x[1:25,1] = x[1:25,1]+3
x[1:25,2] = x[1:25,2]-4
km.result = kmeans(x,2,nstart=20)
names(km.result)
```

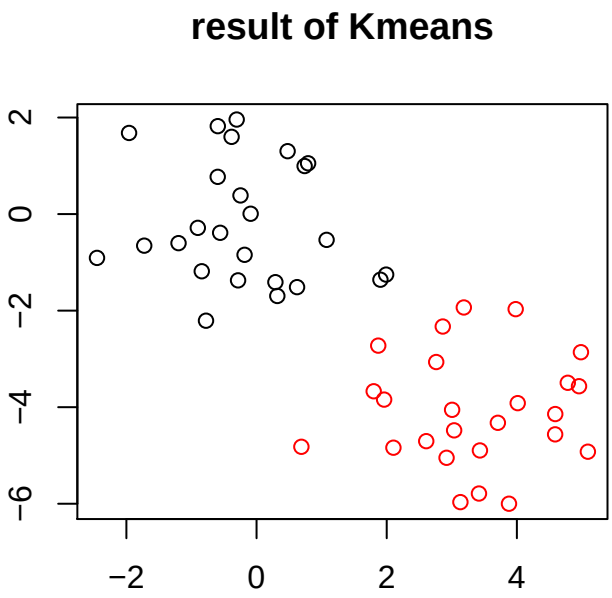
```
[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

```r
km.result$cluster
```

```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1
[36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

K-means가 데이터를 완벽하게 분리하였다. 이를 그림으로 나타내보자.

```
plot(x,col=km.result$cluster,xlab='',ylab='',main='result of Kmeans')
```



kmeans 함수를 실행할 때, multiple initial cluster assignments를 위해서 nstart argument을 사용한다. 만약 nstart 값이 1보다 크다면 알고리즘에서 multiple random assignments을 할 것이고 kmeans 함수는 가장 좋은 결과만 보고하할 것이다. nstart=1과 nstart=20의 결과를 비교해보자.

```
set.seed(3)
km.result = kmeans(x,3,nstart=1)
km.result$tot.withinss
```

```
[1] 104.3319
```

```
km.result = kmeans(x,3,nstart=20)
km.result$tot.withinss
```

```
[1] 97.97927
```

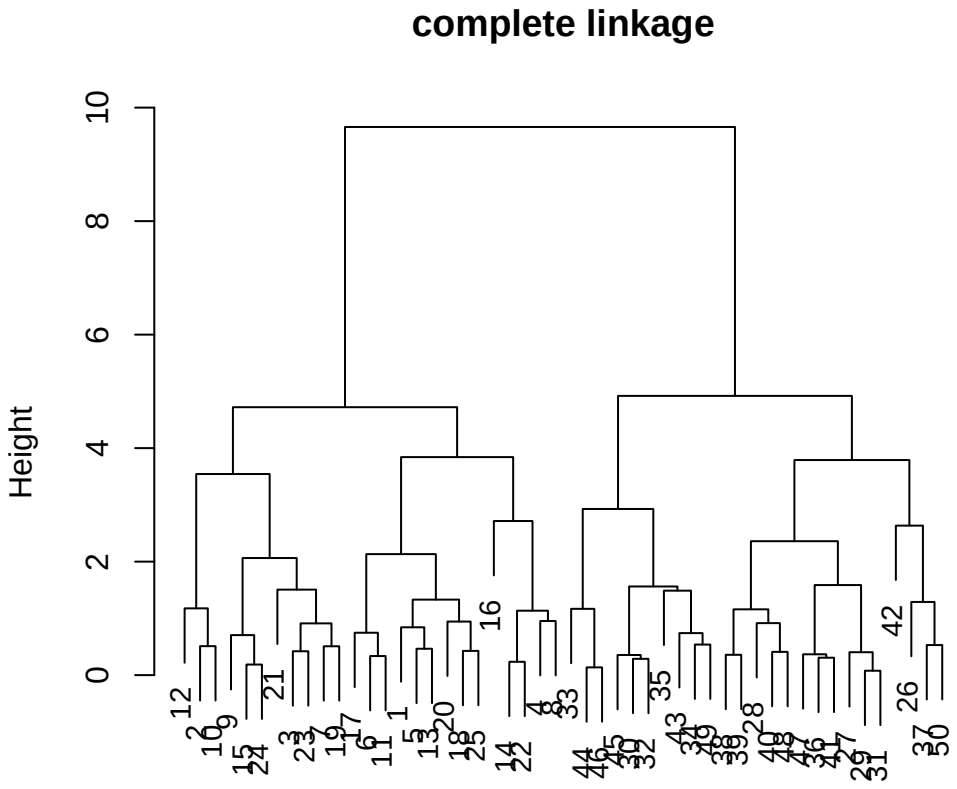
tot.withinss는 k-means에서 최소화하려는, 클러스터 내의 변동성의 총합이다. 각 개별의 클러스터의 변동성 합은 withinss을 통해 확인할 수 있다.

이 책에서는 항상 큰 nstart 값으로 k-means를 시행하기를 강력하게 추천한다. 그렇지 않다면 부적절한 local optimum이 달성될 수 있기 때문이다.
kmeans를 시행할 때에는 nstart 옵션을 넣는 것 뿐만 아니라 set.seed() 함수를 통해 random seed 설정하는 것이 중요하다. 이렇게 함으로써 k-means 결과물을 완전히 동일하게 재생산할수 있기 때문이다.

3. Hierarchical Clustering

dissimilarity measure로 유클리디안 거리를 사용하여 complete, single, average linkage 클러스터링을 수행해본다. dist() 함수는 유클리디안 거리 행렬을 반환해준다.

```
hc.complete = hclust(dist(x),method='complete')
hc.average = hclust(dist(x),method='average')
hc.single = hclust(dist(x),method='single')
plot(hc.complete,main='complete linkage',xlab='',sub='',cex=0.9)
```



plot 함수를 통해 dendrogram을 그려보았다.
각 관측치에 대해서 클러스터의 라벨을 결정하기 위해서 cutree 함수를 사용한다.

```
cutree(hc.complete,2)
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
[36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
cutree(hc.average,2)
```

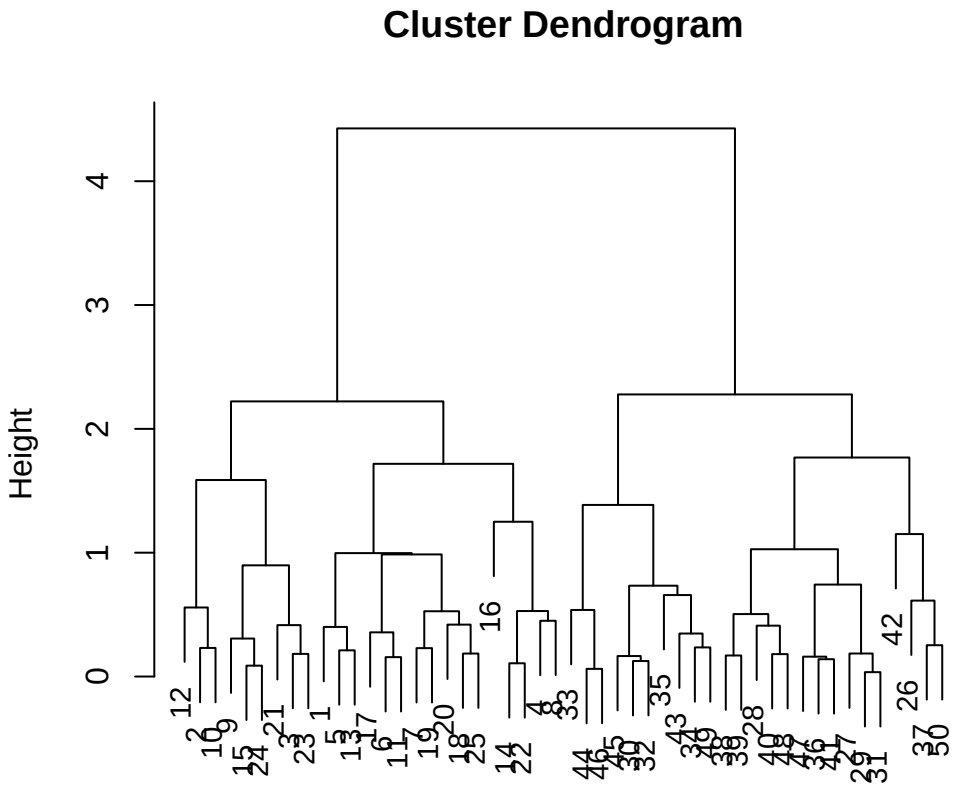
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 2 2
[36] 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2
```

```
cutree(hc.single,2)
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

complete, average linkage을 사용하였을 때에는 나름 클러스터 분류가 잘 되었지만 single linkage는 단 하나를 제외한 모든 관측치가 하나의 클러스터에 있다고 말하는, 다소 이상한 결과를 보여주었다.
클러스터를 하기 이전에, pca처럼 표준화하는 것이 중요하다고 언급했었다. 표준화를 한 이후에 클러스터링을 하면 아래와 같다(근데 사실 x는 표준정규분포에서 발생시킨 값이었으므로 큰 차이는 없을 듯 하다)

```
xsc = scale(x)
plot(hclust(dist(xsc),method='complete'),xlab='',sub='',cex=0.9)
```



상관계수에 기반한 거리는 as.dist 함수를 통해 계산되는데, 이는 임의의 정사각 행렬을 hclust 함수가 거리 행렬로 인식하게끔 변환하는 역할을 한다. 하지만 이는 최소 세 개의 변수를 가지는 데이터에 대해서 적용이 가능하므로 이에 맞는 데이터를 우선 생성하자.

```
x = matrix(rnorm(30*3),ncol=3)
dd = as.dist(1-cor(t(x)))
class(1-cor(t(x)))
```

```
[1] "matrix"
```

```
class(dd)
```

```
[1] "dist"
```

```
hclust(dd,method='complete')
```

```
Call:
hclust(d = dd, method = "complete")

Cluster method   : complete
Number of objects: 30
```

as.dist 함수를 통해 dist 객체로 변환되었음을 확인할 수 있다. dist 객채가 아닌 matrix 객체는 hclust 함수가 인식을 못하므로 주의하자.