REPORT

다변량자료분석 기말과제

32201805 박정민

단국대학교 DANKOOK UNIVERSITY

주성분분석을 이용한 분류분석 및 군집분석

1. 주성분 분석

---

o 훈련용 데이터의 공분산행렬과 상관행렬에 대해 주성분을 실시하고 적절한 주성분의 수 확인

[R코드] : 데이터 불러오기 & 공분산행렬, 상관행렬 주성분

R코드

```
#데이터 불러오기
train<-read.csv("training.csv", header=T)
test<-read.csv("test.csv", header=T)

#train,test 차원확인
dim(train) #[1] 5000   785
dim(test)   #[1] 5000   785

#결측값 확인
sum(is.na(train)) #0
sum(is.na(test)) #0

#class label
table(train[,1])
table(test[,1])
#2      3      5      8      9
#1000 1000 1000 1000 1000

options(max.print = 10000)

##-공분산행렬-##
train.pca<-prcomp(train[,-1])
attributes(train.pca)
summary(train.pca)

##-상관행렬-##
train.pcas<-prcomp(train[,-1],scale.=T)
```

attributes(train.pcas)

summary(train.pcas)

실행결과

```
> ##-공분산분석-##
> train.pca<-prcomp(train[,-1])
> attributes(train.pca)
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"

> summary(train.pca)
Importance of components:
                            PC1       PC2       PC3       PC4       PC5
Standard deviation     354.32253 340.08887 304.94404 286.56817 244.41861
Proportion of Variance   0.02702   0.01986   0.01557   0.01375   0.01000
Cumulative Proportion    0.02702   0.04688   0.06245   0.07620   0.08620
                            PC6       PC7       PC8       PC9      PC10
Standard deviation     224.87764 2.20e+02 204.17556 196.05987 184.17589
Proportion of Variance   0.00889 8.71e-03   0.00699   0.00644   0.00569
Cumulative Proportion    0.09510 9.62e-02   0.10320   0.10964   0.11532
                           PC11      PC12      PC13      PC14      PC15
Standard deviation     182.46127 177.09466 173.86978 169.32182 164.99896
Proportion of Variance   0.00557   0.00525   0.00503   0.00478   0.00456
Cumulative Proportion    0.12090   0.12615   0.13118   0.13596   0.14054
                           PC16      PC17      PC18      PC19      PC20
Standard deviation     156.64667 153.09485 151.89826 148.26905 144.52490
Proportion of Variance   0.00411   0.00392   0.00386   0.00368   0.00349
Cumulative Proportion    0.14465   0.14857   0.15244   0.15612   0.15960
                           PC21      PC22      PC23      PC24      PC25
Standard deviation     143.28707 140.74254 138.12942 135.84706 134.82517
Proportion of Variance   0.00344   0.00332   0.00319   0.00309   0.00304
Cumulative Proportion    0.16305   0.16637   0.16956   0.17265   0.17569
```

```
> ##-공분산행렬-##
> train.pcas<-prcomp(train[,-1],scale.=T)
> attributes(train.pcas)
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"

> summary(train.pcas)
Importance of components:
                          PC1     PC2     PC3     PC4     PC5     PC6     PC7
Standard deviation     3.84451 3.66691 3.33317 3.15401 2.70119 2.46093 2.41949
Proportion of Variance 0.01885 0.01715 0.01417 0.01269 0.00931 0.00772 0.00747
Cumulative Proportion  0.01885 0.03600 0.05017 0.06286 0.07217 0.07989 0.08736
                          PC8     PC9    PC10    PC11    PC12    PC13    PC14
Standard deviation     2.24370 2.16174 2.03972 2.01348 1.94540 1.92550 1.86482
Proportion of Variance 0.00642 0.00596 0.00531 0.00517 0.00483 0.00473 0.00444
Cumulative Proportion  0.09378 0.09974 0.10505 0.11022 0.11505 0.11978 0.12421
                         PC15    PC16    PC17    PC18    PC19    PC20    PC21
Standard deviation     1.82937 1.74692 1.70476 1.67451 1.63104 1.60433 1.58245
Proportion of Variance 0.00427 0.00389 0.00371 0.00358 0.00339 0.00328 0.00319
Cumulative Proportion  0.12848 0.13237 0.13608 0.13966 0.14305 0.14633 0.14953
                         PC22    PC23    PC24    PC25    PC26    PC27    PC28
Standard deviation     1.5589 1.53621 1.50557 1.49302 1.47179 1.45044 1.43564
Proportion of Variance 0.0031 0.00301 0.00289 0.00284 0.00276 0.00268 0.00263
Cumulative Proportion  0.1526 0.15564 0.15853 0.16137 0.16413 0.16682 0.16945
                         PC29    PC30    PC31    PC32    PC33    PC34    PC35
Standard deviation     1.4270 1.41281 1.4005 1.38536 1.3718 1.36179 1.35636
Proportion of Variance 0.0026 0.00255 0.0025 0.00245 0.0024 0.00237 0.00235
Cumulative Proportion  0.1720 0.17459 0.1771 0.17954 0.1819 0.18431 0.18665
                         PC36    PC37    PC38    PC39    PC40    PC41    PC42
Standard deviation     1.34953 1.3427 1.32804 1.32566 1.32210 1.3134 1.31021
Proportion of Variance 0.00232 0.0023 0.00225 0.00224 0.00223 0.0022 0.00219
Cumulative Proportion  0.18898 0.1913 0.19353 0.19577 0.19800 0.2002 0.20239
```

[분석]

train,test 데이터는 각각 785개의 열과 5000개의 행으로 이루어져 있고 결측치는 없고 각각의 class label은 2,3,5,8,9 각각 1000개의 행이 존재한다.

공분산행렬은 scale=F로 주성분분석을 진행한 것이고 상관행렬은 scale=T로 진행하였다.

prcomp()를 통해 얻는 값은 각각 sdev(표준편차), rotation(회전변환식의 계수), center(각 변수들의 중심 위치), scale(표준화), x(pc.score)을 알 수 있다.

[R코드] : 적절한 주성분 수 확인 : 누적 설명력

R코드
##시각화 : 모든 주성분의 수

```
par(mfrow = c(1, 2))
plot(train.pca$x[,1:2],col=train[,1])
sort(train.pca$rotation[,1], decreasing=T)[1:20]

plot(train.pcas$x[,1:2],col=train[,1])
sort(train.pcas$rotation[,1], decreasing=T)[1:20]

##적절한 주성분 수
#누적설명력
cum_var <- cumsum(train.pca$sdev^2)/sum(train.pca$sdev^2)
cum_vars <- cumsum(train.pcas$sdev^2)/sum(train.pcas$sdev^2)

plot(1:length(cum_var), cum_var, type = "l", xlab = "주성분 번호", ylab = "누적 설명력", main = "누적 설명력")

plot(1:length(cum_vars), cum_vars, type = "l", xlab = "주성분 번호", ylab = "누적 설명력", main = "누적 설명력")

#임계값
thresholds <- c(0.7, 0.8, 0.9)
results <- data.frame(Threshold = thresholds, Components = NA, VarianceExplained = NA)

# 임계값 별로 주성분의 수와 설명력 계산
for (i in 1:3) {
  threshold <- thresholds[i]
  sel_com <- which(cum_var >= threshold)[1]
  variance_explained <- cum_var[sel_com]
  results[i, "Components"] <- sel_com
  results[i, "VarianceExplained"] <- variance_explained
}
results

for (i in 1:3) {
  threshold <- thresholds[i]
  sel_com <- which(cum_vars >= threshold)[1]
  variance_explained <- cum_var[sel_com]
  results[i, "Components"] <- sel_com
  results[i, "VarianceExplained"] <- variance_explained
}
```

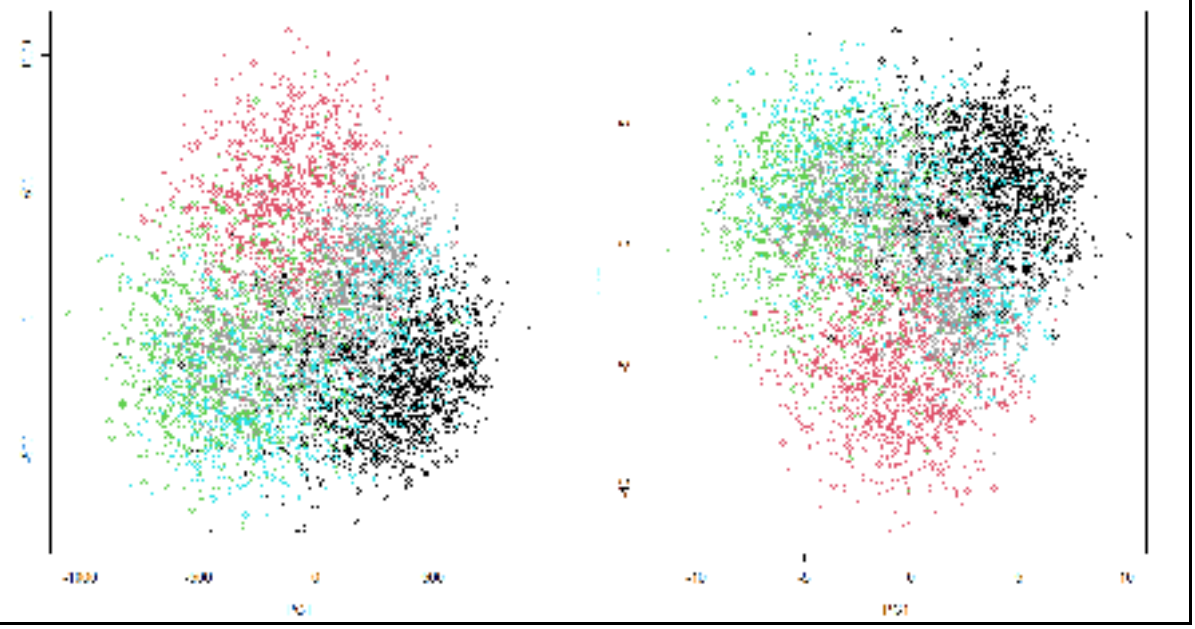
results

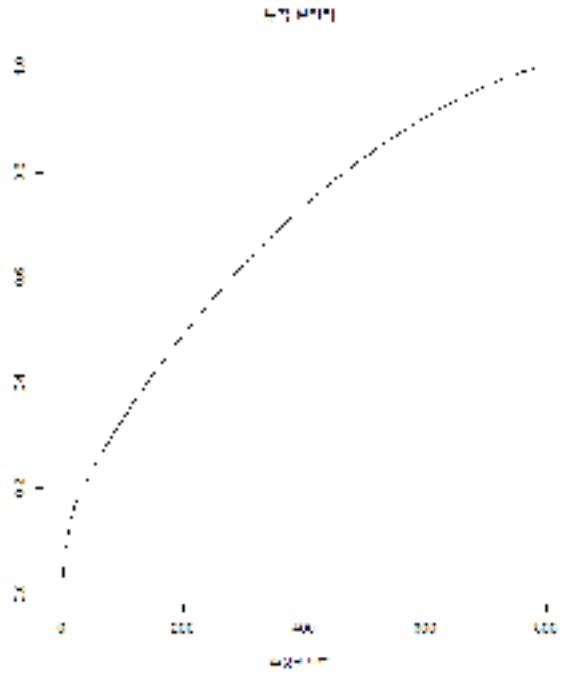
#중간인 80% 선택

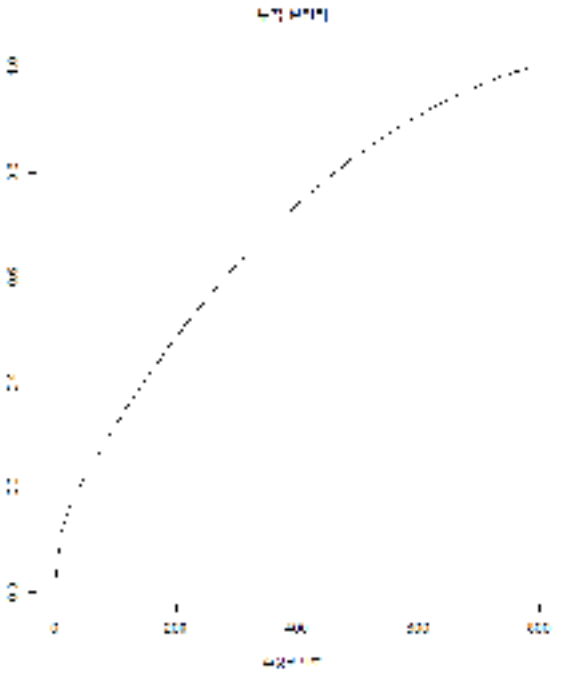
```
train_80.pca<-prcomp(train[,-1],rank.=results[2,2])
train_80.pcas<-prcomp(train[,-1],rank.=results[2,2])

dim(train_80.pca$x)
dim(train_80.pcas$x)
```

실행결과

```
> ##시각화 : 두 개 주성고의 수
> par(mfrow = c(1, 2))
> plot(train.pca$x[,1:2],col=train[,1])
> sort(train.pca$rotation[,1], decreasing=T)[1:20]
      V266       V493       V521       V267       V494       V239       V465
0.10271021 0.09915463 0.09914947 0.09115530 0.08853026 0.08670183 0.08464516
      V265       V240       V520       V293       V492       V522       V549
0.08359681 0.08193741 0.08135952 0.08091662 0.07620960 0.07609749 0.07443200
      V464       V548       V466       V294       V238       V292
0.07408668 0.07287010 0.07286101 0.06986976 0.06706253 0.06652846
> plot(train.pcas$x[,1:2],col=train[,1])
> sort(train.pcas$rotation[,1], decreasing=T)[1:20]
      V193       V266       V521       V194       V165       V239       V267
0.09987698 0.09911045 0.09815430 0.09132609 0.09002726 0.08800609 0.08800035
      V240       V265       V520       V166       V522       V164       V192
0.08403591 0.08121497 0.08046871 0.07947190 0.07795840 0.07777355 0.07715978
      V293       V519       V518       V238       V163       V241
0.07624639 0.07374839 0.07189858 0.06836413 0.06729395 0.06668536
```







```
> #임계값
> thresholds <- c(0.7, 0.8, 0.9)
> results <- data.frame(Threshold = thresholds, Components = NA, VarianceExplained
= NA)
> # 임계값 별로 주성분의 수와 설명력 계산
> for (i in 1:3) {
+   threshold <- thresholds[i]
+   sel_com <- which(cum_var >= threshold)[1]
+   variance_explained <- cum_var[sel_com]
+   results[i, "Components"] <- sel_com
+   results[i, "VarianceExplained"] <- variance_explained
+ }
> results
  Threshold Components VarianceExplained
1       0.7        366         0.7000075
2       0.8        467         0.8005360
3       0.9        593         0.9000804
> for (i in 1:3) {
+   threshold <- thresholds[i]
+   sel_com <- which(cum_vars >= threshold)[1]
+   variance_explained <- cum_var[sel_com]
+   results[i, "Components"] <- sel_com
+   results[i, "VarianceExplained"] <- variance_explained
+ }
> results
  Threshold Components VarianceExplained
1       0.7        365         0.6995245
2       0.8        463         0.7969290
3       0.9        588         0.8996362
> train_80.pca<-prcomp(train[, 1],rank.=results[2,2])
> train_80.pca<-prcomp(train[, 1],rank.=results[2,2])
> dim(train_80.pca$x)
[1] 5000  463
> dim(train_80.pca$x)
[1] 5000  463
```

[분석]

공분산행렬과 상관행렬 둘 다 PC1,PC2에 대해 class label로 군집을 나누어 시각화하였다.

누적 설명력이 70%, 80%, 90% 이상인 것들을 많이 쓴다고 하여 임계값을 이로 지정하고 구성요소의 개수와 누적 설명력을 데이터프레임으로 반환하였다.

누적 설명력이 중간인 80%로 지정하여 rank를 통해 주성분의 수를 463개로 지정하였다.

공분산행렬, 상관행렬 둘다 적절한 주성분의 수가 같다.

---

o 상위 주성분에 대해 주성분점수를 이용하여 시각화

[R코드] : 적절한 주성분의 수 확인 : Scree plot

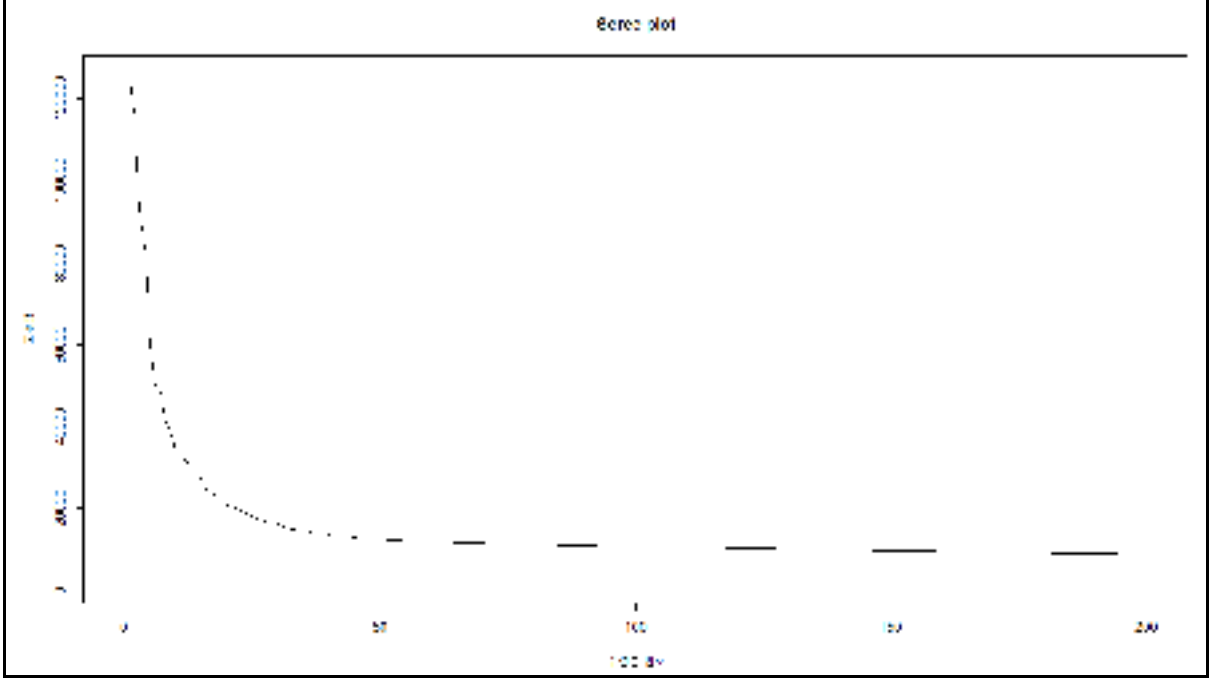
R코드

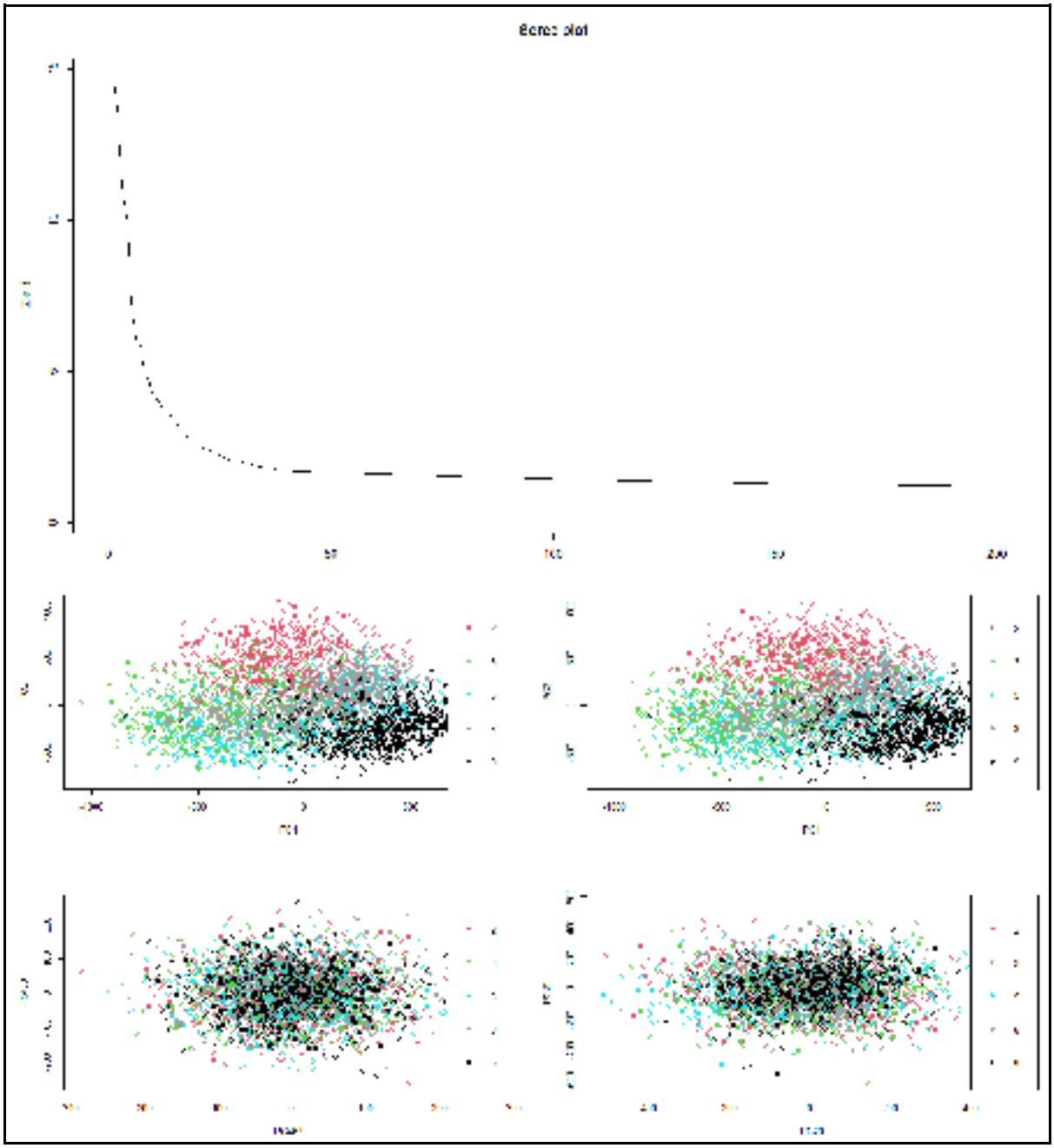
```
#Scree plot
#공분산
eigenvalues=train.pca$sdev^2

#20씩 기울기 계산
slope_values <- seq(1, length(eigenvalues), by = 20)
slope_values

slopes<-c()
for (i in slope_values) {
  slope<-(eigenvalues[i+20]-eigenvalues[i])/20
  slopes<-c(slopes,slope)
}

#기울기 변화
rate_slopes<-diff(slopes)
opt<-which.max(rate_slopes)
opt<-opt*20+1
opt #21

# 최적의 주성분의 수 표시 -> 21
```

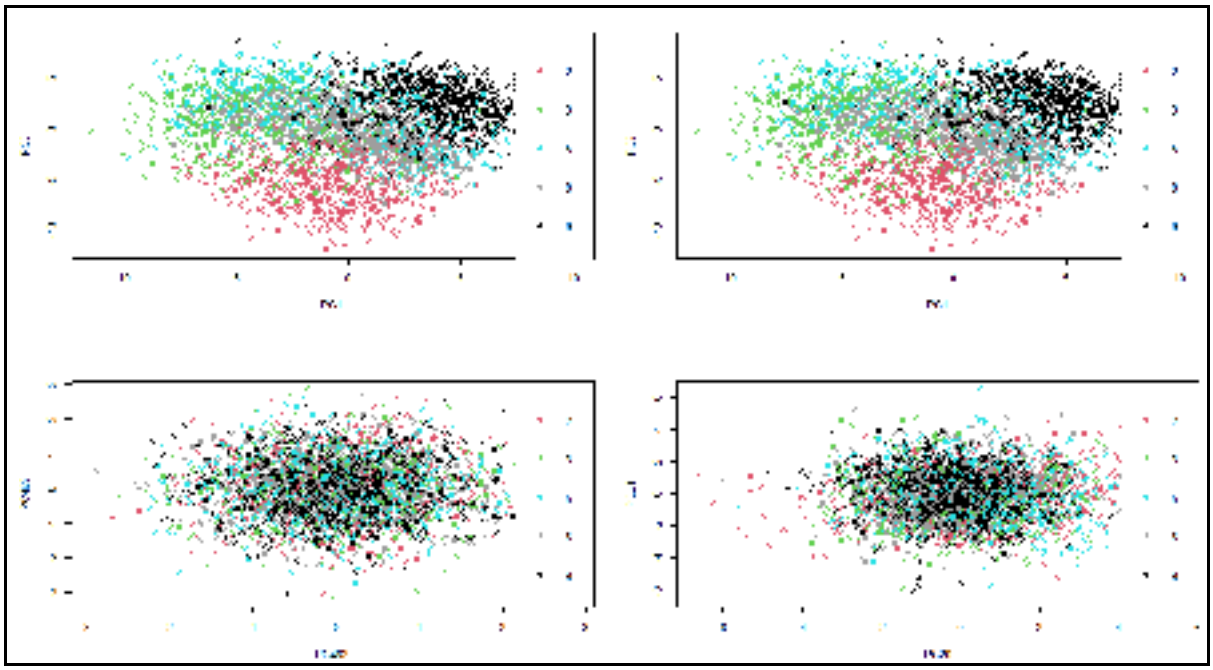
```
plot(1:length(eigenvalues), eigenvalues, type = "l", xlab = "주성분 번호", ylab = "고윳값",main = "Scree plot", xlim=c(0,200))
abline(v = opt, col = "red", lty = 2)

train_sc.pca<-prcomp(train[,-1],rank.=opt)

#상관
eigenvalues=train.pcas$sdev^2

#20씩 기울기 계산
slope_values <- seq(1, length(eigenvalues), by = 20)
slope_values

slopes<-c()
for (i in slope_values) {
  slope<-(eigenvalues[i+20]-eigenvalues[i])/20
  slopes<-c(slopes,slope)
}
slopes

#기울기 변화
rate_slopes<-diff(slopes)
opt<-which.max(rate_slopes)
opt<-opt*20+1
opt

# 최적의 주성분의 수 표시 -> 21
plot(1:length(eigenvalues), eigenvalues, type = "l", xlab = "주성분 번호", ylab = "고윳값",main = "Scree plot", xlim=c(0,200))
abline(v = opt, col = "red", lty = 2)

train_sc.pcas<-prcomp(train[,-1],center=TRUE,rank.=opt)
train_sc.pcas<-prcomp(train[,-1],rank.=opt,scale.=T)

# 상위 주성분 점수를 이용한 시각화
par(mfrow = c(2, 2))
#공분산
plot(train_80.pca$x[,1:2],col=train[,1])
legend("topright", legend=unique(train[,1]),col=unique(train[,1]),pch=16)
```

```
plot(train_sc.pca$x[,1:2],col=train[,1])
legend("topright", legend=unique(train[,1]),col=unique(train[,1]),pch=16)
plot(train_80.pca$x[,462:463],col=train[,1])
legend("topright", legend=unique(train[,1]),col=unique(train[,1]),pch=16)
plot(train_sc.pca$x[,20:21],col=train[,1])
legend("topright", legend=unique(train[,1]),col=unique(train[,1]),pch=16)

#상관
plot(train_80.pcas$x[,1:2],col=train[,1])
legend("topright", legend=unique(train[,1]),col=unique(train[,1]),pch=16)
plot(train_sc.pcas$x[,1:2],col=train[,1])
legend("topright", legend=unique(train[,1]),col=unique(train[,1]),pch=16)
plot(train_80.pcas$x[,462:463],col=train[,1])
legend("topright", legend=unique(train[,1]),col=unique(train[,1]),pch=16)
plot(train_sc.pcas$x[,20:21],col=train[,1])
legend("topright", legend=unique(train[,1]),col=unique(train[,1]),pch=16)
```

실행결과




Scree plot



[분석]

누적 설명력은 총변동의 누적비율로 정보의 손실이 없도록 적절하게 주성분 수를 정할 수 있고 스크리 그래프는 각 데이터의 분산을 확인하는 것이다.

적절한 주성분의 수를 확인하기 위해 스크리 그래프를 그려 경사가 완만해지는 지점을 찾았다.

주성분 번호가 21인 고윳값에서 완만해졌다.

따라서, 상위 주성분의 수에 대해 시각화를 하였는데,

첫번째 줄은 공분산 행렬에서 누적설명력과 스크리 그래프를 통해 구한 상위 주성분 수에 대해 PC1,PC2로 시각화하였다.

두번째 줄은 공분산 행렬에서 상위 주성분의 수 중 마지막 주성분 점수에 대해 시각화하였다.

세번째, 네번째는 각각 상관행렬을 통해 시각화하였고 주성분의 번호가 뒤로 갈수록 군집이 잘 나누어지지 않고 설명력이 떨어지는 것을 볼 수 있다.

---

o 훈련용 데이터의 주성분분석 결과(회전변환 행렬 등)를 이용하여 테스트 데이터의 주성분 점수(PC scores)계산

[R코드] : 테스트 데이터의 주성분 점수 계산 및 시각화

R코드

```
#테스트 데이터 주성분 검사
#공분산
#훈련 데이터 주성분 점수 계산 비교
train_scaled<-scale(train[,-1], train.pca$center, train.pca$scale)
test_scaled<-scale(test[,-1], train.pca$center, train.pca$scale)

train_score1 <- as.data.frame(train.pca$x)
train_score2<-train_scaled %*% train.pca$rotation
train_score1[1:10,1:10]
train_score2[1:10,1:10]

#상위 주성분으로 테스트 데이터 계산
test_scores1 <- test_scaled %*% train_80.pca$rotation
test_scores1[1:10]

test_scores2 <- test_scaled %*% train_sc.pca$rotation
test_scores2[1:10]

#상관
#훈련 데이터 주성분 점수 계산 비교
train_scaled<-scale(train[,-1], train.pcas$center, train.pcas$scale)
test_scaled<-scale(test[,-1], train.pcas$center, train.pcas$scale)

train_score1 <- as.data.frame(train.pcas$x)
train_score2<-train_scaled %*% train.pcas$rotation
train_score1[1:10,1:10]
train_score2[1:10,1:10]

#상위 주성분으로 테스트 데이터 계산
test_scores3 <- test_scaled %*% train_80.pcas$rotation
test_scores3[1:10]

test_scores4 <- test_scaled %*% train_sc.pcas$rotation
test_scores4[1:10]

par(mfrow = c(2, 2))
```

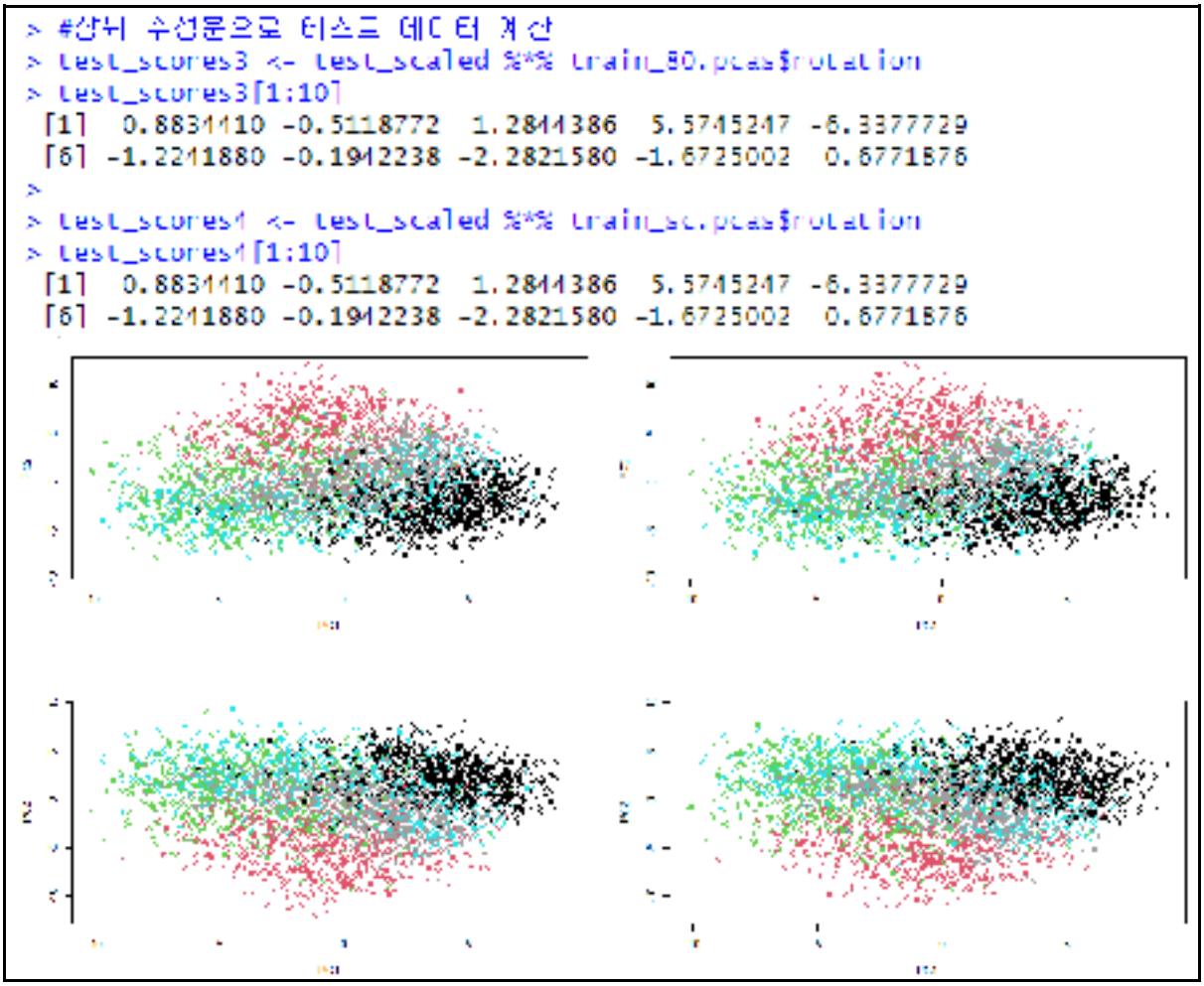
```
plot(test_scores1[,1:2],col=test[,1])
plot(test_scores2[,1:2],col=test[,1])
plot(test_scores3[,1:2],col=test[,1])
plot(test_scores4[,1:2],col=test[,1])
```

실행결과

```
> train_score1[1:10,1:10]
          PC1        PC2      PC3        PC4        PC5
1   1.6042466  6.7211142 4.351410  1.2983182  2.9432980
2   1.3997534  1.9692821 5.302732  1.9952998  0.2559857
3   2.8683939  5.2675268 4.743857  2.3742221  2.6884250
4   0.9137533  5.4827098 4.488188  0.6657843  2.1163765
5   2.8071802  1.9301159 4.712905  1.8090712  1.0429148
6   1.1259992  4.0944344 6.551578  0.5803599  2.3832009
7   0.2830472  5.6788790 2.363668  1.8146120  2.8812209
8   4.9279230  0.1229316 1.403166  2.0074026  2.5442242
9   2.6720374 -4.8552409 5.968347 -4.9139215 -1.2425215
10 -0.2063593 -5.1358496 5.033096 -2.6022721 -4.1803549
          PC6        PC7         PC8        PC9        PC10
1   1.0632835 -2.0121551  0.2755581  0.9316265  3.8385150
2  -0.8641091  3.2769532 -3.5226065 -0.7486613 -3.1557728
3  -1.6615605 -0.7130931  0.3322917 -2.9415523 -0.8396874
4   0.2147735 -0.7139694  2.3504806 -0.6640210  1.9462736
5   1.9713524  4.0804577 -4.2334002 -0.3127792 -0.7058129
6   0.9853542 -0.0483107  1.8023765 -1.7536450  2.0916140
7   0.4872536 -0.3904286  1.2641534  1.2101204  0.5887643
8   2.0354371  3.1758268 -4.1915792  2.0174966  1.2704604
9   2.6496255 -0.3145738 -1.5799979  2.2415042 -2.5698751
10 -2.4155785 -1.1125930  0.1860392  4.7466676 -1.8653781
> train_score2[1:10,1:10]
             PC1        PC2      PC3        PC4        PC5
 [1,] -1.6042466 -6.7211142 4.351410  1.2983182  2.9432980
 [2,]  1.3997534  1.9692821 5.302732  1.9952998 -0.2559857
 [3,]  2.8683939 -5.2675268 4.743857 -2.3742221  2.6884250
 [4,] -0.9137533 -5.4827098 4.488188  0.6657843  2.1163765
 [5,] -2.8071802 -1.9301159 4.712905  1.8090712  1.0429148
 [6,] -1.1259992 -4.0944344 6.551578 -0.5803599  2.3832009
 [7,]  0.2830472 -5.6788790 2.363668  1.8146120  2.8812209
 [8,] -4.9279230  0.1229316 1.403166  2.0074026  2.5442242
 [9,]  2.6720374 -4.8552409 5.968347 -4.9139215 -1.2425215
[10,] -0.2063593 -5.1358496 5.033096 -2.6022721 -4.1803549
             PC6        PC7         PC8        PC9        PC10
 [1,]  1.0632835 -2.0121551  0.2755581  0.9316265  3.8385150
 [2,] -0.8641091  3.2769532 -3.5226065 -0.7486613 -3.1557728
 [3,] -1.6615605 -0.7130931  0.3322917 -2.9415523 -0.8396874
 [4,]  0.2147735 -0.7139694  2.3504806 -0.6640210  1.9462736
 [5,]  1.9713524  4.0804577 -4.2334002 -0.3127792 -0.7058129
 [6,]  0.9853542 -0.0483107  1.8023765 -1.7536450  2.0916140
```

```
> #상위 주성분까지 테스트 데이터 계산
> test_scores1 <- test_scaled %*% train_sc.pca$rotation
> test_scores1[1:10]
 [1]  0.6951506 -0.6489774  7.0898800  5.4409578 -6.3406660
 [6] -1.5515590 -0.1778693 -2.4478480 -7.9219651  0.4451728
>
> test_scores2 <- test_scaled %*% train_sc.pca$rotation
> test_scores2[1:10]
 [1]  0.6951506 -0.6489774  7.0898800  5.4409578 -6.3406660
 [6] -1.5515590 -0.1778693 -2.4478480 -7.9219651  0.4451728
> train_score1[1:10,1:10]
          PC1        PC2      PC3        PC4        PC5
1  -1.6042466 -6.7211142 4.351410  1.2983182  2.9432980
2   1.3997534  1.9692821 5.302732  1.9952998 -0.2559857
3   2.8683939 -5.2675268 4.743857 -2.3742221  2.6884250
4  -0.9137533 -5.4827098 4.488188  0.6657843  2.1163765
5  -2.8071802 -1.9301159 4.712905  1.8090712  1.0429148
6  -1.1259992 -4.0944344 6.551578 -0.5803599  2.3832009
7   0.2830472 -5.6788790 2.363668  1.8146120  2.8812209
8  -4.9279230  0.1229316 1.403166  2.0074026  2.5442242
9   2.6720374 -4.8552409 5.968347 -4.9139215 -1.2425215
10 -0.2063593 -5.1358496 5.033096 -2.6022721 -4.1803549
          PC6        PC7        PC8        PC9       PC10
1   1.0632835 -2.0121551  0.2755581  0.9316265  3.8385150
2  -0.8641091  3.2769532 -3.5226065 -0.7186613 -3.1557728
3  -1.6615605 -0.7130931  0.3322917 -2.9445523 -0.8396874
4   0.2147735 -0.7139694  2.3504806 -0.6640210  1.9462736
5   1.9713524  4.0801577 -1.2334002 -0.3127792 -0.7058129
6   0.9853542 -0.0483107  1.8023765 -1.7536450  2.0916140
7   0.4872536 -0.3901286  1.2641534  1.2101204  0.5887643
8   2.0354371  3.1758268 -1.1915792  2.0174966  1.2701604
9   2.6496255 -0.3145738 -1.5799979  2.2115042 -2.5698751
10 -2.4155785 -1.1125930  0.1860392  1.7166676 -1.8653781
> train_score2[1:10,1:10]
           PC1        PC2      PC3        PC4        PC5
 [1,] -1.6042466 -6.7211142 4.351410  1.2983182  2.9432980
 [2,]  1.3997534  1.9692821 5.302732  1.9952998  0.2559857
 [3,]  2.8683939  5.2675268 4.743857  2.3742221  2.6884250
 [4,]  0.9137533  5.4827098 4.488188  0.6657843  2.1163765
 [5,]  2.8071802  1.9301159 4.712905  1.8090712  1.0429148
 [6,]  1.1259992  4.0944344 6.551578  0.5803599  2.3832009
 [7,]  0.2830472  5.6788790 2.363668  1.8146120  2.8812209
 [8,]  4.9279230  0.1229316 1.403166  2.0074026  2.5442242
 [9,]  2.6720374  4.8552409 5.968347  4.9139215  1.2425215
[10,]  0.2063593  5.1358496 5.033096  2.6022721  4.1803549
           PC6        PC7        PC8        PC9       PC10
 [1,]  1.0632835  2.0121551  0.2755581  0.9316265  3.8385150
 [2,]  0.8641091  3.2769532  3.5226065  0.7486613  3.1557728
 [3,]  1.6615605  0.7130931  0.3322917  2.9445523  0.8396874
 [4,]  0.2147735  0.7139694  2.3504806  0.6640210  1.9462736
```

```
> #상위 주성분으로 테스트 데이터 계산
> test_scores3 <- test_scaled %*% train_80.pcas$rotation
> test_scores3[1:10]
 [1]  0.8831110 -0.5118772  1.2844386  5.5745247 -6.3377729
 [6] -1.2211880 -0.1942238 -2.2821580 -1.6725002  0.6771876
>
> test_scores4 <- test_scaled %*% train_80.pcas$rotation
> test_scores4[1:10]
 [1]  0.8831110 -0.5118772  1.2844386  5.5745247 -6.3377729
 [6] -1.2211880 -0.1942238 -2.2821580 -1.6725002  0.6771876
```



[분석]

훈련데이터를 주성분 분석한 결과 x가 주성분 점수를 의미한다.

원데이터에서 rotation을 곱해주면 같은 주성분 점수를 나타내기 때문에 이를 훈련데이터에서 같은 값이 나오는지 확인하였다.

같은 방법으로 테스트 데이터를 훈련 데이터에서 구한 주성분 분석 결과인 rotation(회전행렬)과 scale(표준화 여부), center(중심 위치)를 이용하여 주성분 점수를 계산하였다.

이를 PC1, PC2로 시각화하였다.

2. 주성분점수를 이용한 분류분석

---

o 주성분의 수 k를 변화시켜가면서 k개의 주성분점수를 svm()함수에 적용하여 분류분석을 진행하고 784개의 모든 변수를 다 사용하였을 때의 분류분석 결과와 비교

[R코드] : 모든 변수 사용하여 svm(svc, polynomial, radial) 모델 생성 후 평가

R코드

```
library(MASS)
library(ggplot2)
library(e1071)
library(caret)

#데이터 불러오기
train<-read.csv("training.csv", header=T)
test<-read.csv("test.csv", header=T)

#공분산
train.pca <- prcomp(train[,-1],center=T)
dim(train.pca$x)

#테스트 목표변수+주성분 점수
train_new<-c()
train_new<-cbind(train[,1],train.pca$x)
colnames(train_new)[1]<-'trny'

# 주성분 결과 적용하여 테스트 데이터의 주성분 계산
test_centered <- scale(test[,-1], center = train.pca$center, scale = train.pca$scale)
test_pc_scores <- test_centered %*% train.pca$rotation

#테스트 목표변수+주성분 점수
test_new<-cbind(test[,1],test_pc_scores)
colnames(test_new)[1]<-'tsny'

#모든 주성분 분석 사용 - SVM
model1 <- svm(trny~., data = train_new, kernel='linear', type = "C-classification")
```

```
model2 <- svm(trny~., data = train_new, kernel='polynomial', type = "C-classification")
model3 <- svm(trny~., data = train_new, kernel='radial', type = "C-classification")

# 분류 모델 평가
predictions1 <- predict(model1, test_new[,-1])
predictions2 <- predict(model2, test_new[,-1])
predictions3 <- predict(model3, test_new[,-1])

#모든 주성분 분석 사용
#Linear
confusionMatrix(as.factor(predictions1), as.factor(test[,1]),mode = "everything", positive='1')
#Polynomial
confusionMatrix(as.factor(predictions2), as.factor(test[,1]),mode = "everything", positive='1')
#Radial
confusionMatrix(as.factor(predictions3), as.factor(test[,1]),mode = "everything", positive='1')
```

실행결과

```
          Reference
Prediction   2   3   5   8   9
         2 785  83  46  89  42
         3  64 729 139 107  59
         5  49  98 708 106  58
         8  69  56  87 654  75
         9  33  34  20  44 766

Overall Statistics

               Accuracy : 0.7284
                 95% CI : (0.7158, 0.7407)
    No Information Rate : 0.2
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6605

 Mcnemar's Test P-Value : 5.299e-10

Statistics by Class:

                     Class: 2 Class: 3 Class: 5 Class: 8
Sensitivity            0.7850   0.7290   0.7080   0.6540
Specificity            0.9350   0.9077   0.9223   0.9283
Pos Pred Value         0.7512   0.6639   0.6948   0.6930
Neg Pred Value         0.9456   0.9305   0.9267   0.9148
Precision              0.7512   0.6639   0.6948   0.6930
Recall                 0.7850   0.7290   0.7080   0.6540
F1                     0.7677   0.6949   0.7013   0.6739
Prevalence             0.2000   0.2000   0.2000   0.2000
Detection Rate         0.1570   0.1458   0.1416   0.1308
Detection Prevalence   0.2090   0.2196   0.2038   0.1882
Balanced Accuracy      0.8600   0.8184   0.8151   0.7911
                     Class: 9
Sensitivity            0.7660
Specificity            0.9673
Pos Pred Value         0.8540
Neg Pred Value         0.9430
Precision              0.8540
Recall                 0.7660
F1                     0.8076
Prevalence             0.2000
Detection Rate         0.1532
```

```
          Reference
Prediction   2   3   5   8   9
         2 144   0   0   1   1
         3 328 860 416  77 164
         5   0   0  10   0   0
         8 512 135 561 919 336
         9  16   5  13   3 499

Overall Statistics

               Accuracy : 0.4864
                 95% CI : (0.4725, 0.5004)
    No Information Rate : 0.2
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.358

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 2 Class: 3 Class: 5 Class: 8
Sensitivity            0.1440   0.8600   0.0100   0.9190
Specificity            0.9995   0.7538   1.0000   0.6140
Pos Pred Value         0.9863   0.4661   1.0000   0.3731
Neg Pred Value         0.8237   0.9556   0.8016   0.9681
Precision              0.9863   0.4661   1.0000   0.3731
Recall                 0.1440   0.8600   0.0100   0.9190
F1                     0.2513   0.6046   0.0198   0.5308
Prevalence             0.2000   0.2000   0.2000   0.2000
Detection Rate         0.0288   0.1720   0.0020   0.1838
Detection Prevalence   0.0292   0.3690   0.0020   0.4926
Balanced Accuracy      0.5717   0.8069   0.5050   0.7665
                     Class: 9
Sensitivity            0.4990
Specificity            0.9908
Pos Pred Value         0.9310
Neg Pred Value         0.8878
Precision              0.9310
Recall                 0.4990
F1                     0.6497
```

```
          Reference
Prediction   2   3   5   8   9
         2 863  81  17  63  20
         3  12 653  31  33  11
         5  60 108 923 105  57
         8  28  26  20 650  16
         9  28  42   9  50 806

Overall Statistics

               Accuracy : 0.797
                 95% CI : (0.7856, 0.8081)
    No Information Rate : 0.2
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7162

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 2 Class: 3 Class: 5 Class: 8
Sensitivity            0.8630   0.6530   0.9230   0.6500
Specificity            0.9547   0.9782   0.8702   0.9775
Pos Pred Value         0.8266   0.8821   0.6101   0.8781
Neg Pred Value         0.9651   0.9185   0.9781   0.9178
Precision              0.8266   0.8821   0.6101   0.8781
Recall                 0.8630   0.6530   0.9230   0.6500
F1                     0.8444   0.7506   0.7559   0.7471
Prevalence             0.2000   0.2000   0.2000   0.2000
Detection Rate         0.1726   0.1306   0.1846   0.1300
Detection Prevalence   0.2088   0.1480   0.2881   0.1480
Balanced Accuracy      0.9089   0.8156   0.8966   0.8137
                     Class: 9
Sensitivity            0.8960
Specificity            0.9855
Pos Pred Value         0.8665
Neg Pred Value         0.9738
Precision              0.8665
Recall                 0.8960
F1                     0.8810
```

[분석]

모든 주성분 수 (734개)를 이용하여 SVM(linear, polynomial, radial)을 진행하였다.

Linear커널로 모델 생성 후 평가한 결과는

정확도 : 0.7284

정밀도 : 0.7512 0.6639 0.6948 0.6950 0.8540

재현율 : 0.7850 0.7290 0.7080 0.6540 0.7660

F1- score : 0.7677 0.6949 0.7013 0.6739 0.8076

Polynomial 커널은

정확도 : 0.4864

정밀도 : 0.9863 0.4661 1.0000 0.3731 0.9310

재현율 : 0.1440 0.8600 0.0100 0.9190 0.4990

F1- score : 0.2513 0.6046 0.0198 0.5308 0.6497

Radial 커널은

정확도 : 0.797

정밀도 : 0.8266 0.8824 0.6401 0.8784 0.8665

재현율 : 0.8630 0.6530 0.9230 0.6500 0.8960

F1-score : 0.8444 0.7506 0.7559 0.7471 0.8810

로 결과가 도출되었다.

정확도는 radial 커널로 모델을 돌린 결과가 0.797 로 가장 높고 polynomial 커널을 통해 돌린 결과가 0.49 로 가장 낮다. 정밀도, 재현율, f1-score 는 test class label 인 2,3,5,8,9 에 대해 각각 결과가 도출되었다. Polynomial 커널로 모델을 평가하였을 때에는 class label 이 8 일 때 정밀도가 1 이 나오고 재현율과 f1-score 가 각각 0.01 이 나오는 것과 같이 분류분석이 제대로 되지 않음을 알 수 있다.

[R 코드] : 주성분의 수 k 를 10%씩 변화시켜가며 분류분석 (공분산행렬)

R 코드

```
# 공분산행렬-k 를 변화시키면서 분류 분석 수행: 10%씩
end<-round(dim(train.pca$x)[2]/10)*10
len<-seq(end/10,end,end/10)
len

#Linear
train_new<-c()
test_new<-c()
predictions<-c()

for (k in len) {
  train_new<-cbind(train[,1],train.pca$x)
  colnames(train_new)[1]<-'trny'
  train_sel<-train_new[,c(1:k)]
```

```
# 주성분 결과 적용하여 테스트 데이터의 주성분 계산
test_centered <- scale(test[,-1], center = train.pca$center, scale = train.pca$scale)
test_pc_scores <- test_centered %*% train.pca$rotation

test_new<-cbind(test[,1],test_pc_scores)
colnames(test_new)[1]<-'tsny'
test_sel<-test_new[,c(1:k)]

model <- svm(trny~., data = train_sel, kernel='linear', type = "C-classification")

# 분류 모델 평가
predictions <- predict(model, test_sel[,-1])

accuracy <- sum(predictions == test[,1]) / length(predictions)  # 분류 정확도 계산

# 클래스 수 계산
num_classes <- length(unique(test[,1]))

# 각 클래스별로 TP, FP, FN 계산
tp <- numeric(num_classes)
fp <- numeric(num_classes)
fn <- numeric(num_classes)

actual<-test[,1]
for (i in c(2,3,5,8,9)) {
  tp[i] <- sum(actual == i & predictions == i)
  fp[i] <- sum(actual != i & predictions == i)
  fn[i] <- sum(actual == i & predictions != i)
}



# 정밀도 계산
precision <- tp / (tp + fp)
precision<-precision[c(2,3,5,8,9)]

# 재현율 계산
recall <- tp / (tp + fn)
recall<-recall[c(2,3,5,8,9)]
```

```
  # F-1 점수 계산
  f1_score <- (2 * precision * recall) / (precision + recall)


  # 결과 출력
  cat("Number of principal components:", k,"₩n","| Accuracy:", accuracy, "₩n")
  cat("| Precision:", precision, "₩n")
  cat("| Recall:", recall, "₩n")
  cat("| F1-score:", f1_score, "₩n")
}

#Polynomial
train_new<-c()
test_new<-c()
predictions<-c()

for (k in len) {
  train_new<-cbind(train[,1],train.pca$x)
  colnames(train_new)[1]<-'trny'
  train_sel<-train_new[,c(1:k)]

  # 주성분 결과 적용하여 테스트 데이터의 주성분 계산
  test_centered <- scale(test[,-1], center = train.pca$center, scale = train.pca$scale)
  test_pc_scores <- test_centered %*% train.pca$rotation

  test_new<-cbind(test[,1],test_pc_scores)
  colnames(test_new)[1]<-'tsny'
  test_sel<-test_new[,c(1:k)]

  model <- svm(trny~., data = train_sel, kernel='polynomial', type = "C-classification")

  # 분류 모델 평가
  predictions <- predict(model, test_sel[,-1])

  accuracy <- sum(predictions == test[,1]) / length(predictions)  # 분류 정확도 계산

  # 클래스 수 계산
  num_classes <- length(unique(test[,1]))
```

```
  # 각 클래스별로 TP, FP, FN 계산
  tp <- numeric(num_classes)
  fp <- numeric(num_classes)
  fn <- numeric(num_classes)

  actual<-test[,1]
  for (i in c(2,3,5,8,9)) {
    tp[i] <- sum(actual == i & predictions == i)
    fp[i] <- sum(actual != i & predictions == i)
    fn[i] <- sum(actual == i & predictions != i)
  }


  # 정밀도 계산
  precision <- tp / (tp + fp)
  precision<-precision[c(2,3,5,8,9)]

  # 재현율 계산
  recall <- tp / (tp + fn)
  recall<-recall[c(2,3,5,8,9)]

  # F-1 점수 계산
  f1_score <- 2 * precision * recall / (precision + recall)


  # 결과 출력
  cat("Number of principal components:", k,"₩n","| Accuracy:", accuracy, "₩n")
  cat("| Precision:", precision, "₩n")
  cat("| Recall:", recall, "₩n")
  cat("| F1-score:", f1_score, "₩n")
}

#Radial
train_new<-c()
test_new<-c()
predictions<-c()

for (k in len) {
```

```
train_new<-cbind(train[,1],train.pca$x)
colnames(train_new)[1]<-'trny'
train_sel<-train_new[,c(1:k)]

# 주성분 결과 적용하여 테스트 데이터의 주성분 계산
test_centered <- scale(test[,-1], center = train.pca$center, scale = train.pca$scale)
test_pc_scores <- test_centered %*% train.pca$rotation

test_new<-cbind(test[,1],test_pc_scores)
colnames(test_new)[1]<-'tsny'
test_sel<-test_new[,c(1:k)]

model <- svm(trny~., data = train_sel, kernel='radial', type = "C-classification")

# 분류 모델 평가
predictions <- predict(model, test_sel[,-1])

accuracy <- sum(predictions == test[,1]) / length(predictions)  # 분류 정확도 계산

# 클래스 수 계산
num_classes <- length(unique(test[,1]))

# 각 클래스별로 TP, FP, FN 계산
tp <- numeric(num_classes)
fp <- numeric(num_classes)
fn <- numeric(num_classes)

actual<-test[,1]
for (i in c(2,3,5,8,9)) {
  tp[i] <- sum(actual == i & predictions == i)
  fp[i] <- sum(actual != i & predictions == i)
  fn[i] <- sum(actual == i & predictions != i)
}


# 정밀도 계산
precision <- tp / (tp + fp)
precision<-precision[c(2,3,5,8,9)]
```

```
  # 재현율 계산
  recall <- tp / (tp + fn)
  recall<-recall[c(2,3,5,8,9)]

  # F-1 점수 계산
  f1_score <- 2 * precision * recall / (precision + recall)



  # 결과 출력
  cat("Number of principal components:", k,"₩n","| Accuracy:", accuracy, "₩n")
  cat("| Precision:", precision, "₩n")
  cat("| Recall:", recall, "₩n")
  cat("| F1-score:", f1_score, "₩n")
}
```

실행결과

```
> len
 [1]  78 156 234 312 390 468 546 624 702 780
Number of principal components: 78
 | Accuracy: 0.8734
| Precision: 0.8876953 0.8441558 0.8250478 0.8899676 0.9231537
| Recall: 0.909 0.845 0.863 0.825 0.925
| F1-score: 0.8982213 0.8445777 0.8435973 0.8562532 0.9240759
Number of principal components: 156
 | Accuracy: 0.8518
| Precision: 0.8688363 0.8171206 0.8122568 0.8542977 0.9098361
| Recall: 0.881 0.84 0.835 0.815 0.888
| F1-score: 0.8748759 0.8284024 0.8234714 0.8341863 0.8987854
Number of principal components: 234
 | Accuracy: 0.8168
| Precision: 0.8444882 0.772381 0.7799416 0.812433 0.8788501
| Recall: 0.858 0.811 0.801 0.758 0.856
| F1-score: 0.8511905 0.7912195 0.7903305 0.7842732 0.8672716
Number of principal components: 312
 | Accuracy: 0.7988
| Precision: 0.8187919 0.7218201 0.777336 0.7916129 0.8907741
| Recall: 0.854 0.806 0.782 0.712 0.84
| F1-score: 0.8360255 0.7632576 0.779661 0.7510549 0.8646123
Number of principal components: 390
 | Accuracy: 0.7888
| Precision: 0.8121442 0.7145138 0.7700205 0.771033 0.8887689
| Recall: 0.856 0.791 0.75 0.724 0.823
| F1-score: 0.8334956 0.7508306 0.7598784 0.7467767 0.854621
```

```
Number of principal components: 468
 | Accuracy: 0.7874
| Precision: 0.8074157 0.7188645 0.7121181 0.7845982 0.8778706
| Recall: 0.847 0.785 0.736 0.703 0.841
| F1-score: 0.8267448 0.7504278 0.7241646 0.7415612 0.8590198
Number of principal components: 546
 | Accuracy: 0.7684
| Precision: 0.7861248 0.7078004 0.7120901 0.7565789 0.8800088
| Recall: 0.828 0.778 0.746 0.69 0.8
| F1-score: 0.8066245 0.7384907 0.7289797 0.7171571 0.8387151
Number of principal components: 624
 | Accuracy: 0.76
| Precision: 0.7652011 0.7002775 0.7251462 0.75 0.8749015
| Recall: 0.818 0.757 0.744 0.684 0.797
| F1-score: 0.7907202 0.7275348 0.7344571 0.7154817 0.8311682
Number of principal components: 702
 | Accuracy: 0.7494
| Precision: 0.7599962 0.6811989 0.7221095 0.7001587 0.8688709
| Recall: 0.801 0.75 0.717 0.69 0.794
| F1-score: 0.7799416 0.7139457 0.7170141 0.6945116 0.8296761
Number of principal components: 780
 | Accuracy: 0.7286
| Precision: 0.7611165 0.6552347 0.6941215 0.7011914 0.8520578
| Recall: 0.784 0.726 0.701 0.666 0.766
| f1 score: 0.7724138 0.6888046 0.6961271 0.6834274 0.8067404
```

```
Number of principal components: 78
 | Accuracy: 0.9148
| Precision: 0.9886284 0.8572751 0.919598 0.8974104 0.924757
| Recall: 0.869 0.975 0.915 0.901 0.964
| F1-score: 0.9249601 0.8898509 0.9173913 0.8992016 0.9443702
Number of principal components: 156
 | Accuracy: 0.8712
| Precision: 0.9873897 0.8042713 0.8880676 0.8814501 0.8772482
| Recall: 0.783 0.886 0.847 0.881 0.961
| F1-score: 0.8733965 0.8431079 0.8670912 0.85645 0.9184549
Number of principal components: 234
 | Accuracy: 0.835
| Precision: 0.9903587 0.7472246 0.8931947 0.7670753 0.8512412
| Recall: 0.719 0.875 0.749 0.876 0.956
| F1-score: 0.8331402 0.8060807 0.8150161 0.8179777 0.9006123
Number of principal components: 312
 | Accuracy: 0.8066
| Precision: 0.9854444 0.6995305 0.9154728 0.7191617 0.8471223
| Recall: 0.677 0.894 0.619 0.881 0.942
| F1-score: 0.8026082 0.7848992 0.7386502 0.7919101 0.8920455
Number of principal components: 390
 | Accuracy: 0.787
| Precision: 0.97609 0.6621788 0.9128841 0.7117117 0.8744849
| Recall: 0.694 0.914 0.574 0.869 0.914
| F1-score: 0.8112215 0.7680672 0.7049142 0.7825304 0.8786453
```

```
Number of principal components: 468
 | Accuracy: 0.7492
| Precision: 0.9790323 0.6226931 0.9552239 0.8348637 0.8401461
| Recall: 0.607 0.911 0.448 0.885 0.895
| f1 score: 0.7493827 0.7397483 0.6099387 0.7303484 0.8714703
Number of principal components: 546
 | Accuracy: 0.713
| Precision: 0.9739837 0.5685786 0.9606061 0.8073019 0.8515241
| Recall: 0.599 0.912 0.317 0.871 0.866
| f1 score: 0.7417957 0.7004608 0.4766917 0.7156043 0.858701
Number of principal components: 624
 | Accuracy: 0.6634
| Precision: 0.9736347 0.5447304 0.9765258 0.5334123 0.8533755
| Recall: 0.517 0.889 0.208 0.894 0.809
| f1 score: 0.6753756 0.6755319 0.3429514 0.6881614 0.8305055
Number of principal components: 702
 | Accuracy: 0.5808
| Precision: 0.9846547 0.4869565 0.96875 0.4544073 0.8025487
| Recall: 0.385 0.896 0.124 0.897 0.602
| f1 score: 0.5535586 0.6309859 0.2198582 0.603228 0.7222555
Number of principal components: 780
 | Accuracy: 0.498
| Precision: 0.989418 0.4682366 1 0.3797101 0.820082
| Recall: 0.187 0.855 0.013 0.917 0.518
| f1 score: 0.31455 0.6050955 0.02566634 0.5370425 0.6653821
```

```
Number of principal components: 78
 | Accuracy: 0.9204
| Precision: 0.9660657 0.8930693 0.8937198 0.9330544 0.9204545
| Recall: 0.911 0.902 0.925 0.892 0.972
| F1-score: 0.9377252 0.8975124 0.9090909 0.9120654 0.9455253
Number of principal components: 156
 | Accuracy: 0.8836
| Precision: 0.9199444 0.8503937 0.8344186 0.9033298 0.8915663
| Recall: 0.854 0.864 0.897 0.841 0.962
| F1-score: 0.8994207 0.8571429 0.8645783 0.8710513 0.9254449
Number of principal components: 234
 | Accuracy: 0.868
| Precision: 0.943757 0.8255361 0.8273585 0.888165 0.8080594
| Recall: 0.839 0.847 0.877 0.818 0.959
| F1-score: 0.8883007 0.8361303 0.8514563 0.8516398 0.911397
Number of principal components: 312
 | Accuracy: 0.8664
| Precision: 0.9346622 0.8216374 0.8293384 0.8789809 0.8766114
| Recall: 0.844 0.843 0.865 0.828 0.952
| F1-score: 0.8870205 0.8321816 0.8467939 0.8527291 0.9127317
Number of principal components: 390
 | Accuracy: 0.8632
| Precision: 0.9243421 0.807619 0.8326848 0.8744681 0.8850467
| Recall: 0.843 0.848 0.856 0.822 0.947
| F1-score: 0.8817992 0.8273171 0.8441815 0.8474227 0.9149758
Number of principal components: 468
 | Accuracy: 0.8582
| Precision: 0.9166667 0.8047847 0.8272816 0.8541667 0.8835361
| Recall: 0.847 0.841 0.843 0.82 0.94
| f1 score: 0.8804574 0.8224939 0.8350669 0.8367347 0.9161703
Number of principal components: 546
 | Accuracy: 0.85
| Precision: 0.8963675 0.8118227 0.8123772 0.8383838 0.8833718
| Recall: 0.839 0.824 0.827 0.83 0.93
| f1 score: 0.8667355 0.817866 0.8196234 0.8341709 0.911315
Number of principal components: 624
 | Accuracy: 0.842
| Precision: 0.8946809 0.8121827 0.7911962 0.8290472 0.8640996
| Recall: 0.841 0.8 0.847 0.805 0.928
| F1-score: 0.8670103 0.8060453 0.8176049 0.8168442 0.8938511
Number of principal components: 702
 | Accuracy: 0.8372
| Precision: 0.8778271 0.8455647 0.7884071 0.8461538 0.8744053
| Recall: 0.857 0.777 0.857 0.781 0.919
| F1-score: 0.8673469 0.8071093 0.8206948 0.8122725 0.8961482
Number of principal components: 780
 | Accuracy: 0.8048
| Precision: 0.8302797 0.8845744 0.7594517 0.8691171 0.8567806
| Recall: 0.861 0.674 0.914 0.671 0.904
| F1-score: 0.8453608 0.7650397 0.7661158 0.7574161 0.8849781
```

[분석]

svm 모델로 주성분의 수인 k 를 10%씩 늘려가며 총 10 번을 반복하여 평가하였다.

주성분의 수를 10%인 78 개를 사용하였을 때가 3 개의 커널 모두 정확도가 가장 높고 주성분의 수가 늘어갈수록 점점 낮아지는 것을 볼 수 있다.

<k 개의 주성분 점수로 분류분석 vs 모든 변수 다 사용하였을 때의 분류분석>
3 개의 커널 모두 상위 주성분을 사용할수록 정확도가 높아진다. 하지만, polynomial 커널을 사용하였을 때 상위 주성분으로 모델 평가를 하였을 때와 모든 변수를 다 사용하였을 때의 정확도의 차이가 크게 나타난다. 그 이유는 데이터셋이 복잡하지 않고 차원이 매우 높아 과적합의 문제가 발생하였다고 해석할 수 있다. 3 개의 커널 중 가장 정확도가 높은 것은 radial 커널을 사용하였을 때이므로 데이터셋은 비선형성을 가지고 있다고 할 수 있다.

3. 군집분석

---

o k-평균 군집분석을 하기 위한 함수 작성

R 코드

```
set.seed(1805)
#데이터 불러오기
train<-read.csv("training.csv", header=T)
df<-train[,-1]
df<-df[,c(1:10)]

trnx<-as.matrix(df)
row<-nrow(trnx)
col<-ncol(trnx)

k_means<-function(data,k){
  sample<-sample(row,k,replace=FALSE)
  w<-data[sample,]
  temp<-w
  w2<-0
  while(identical(w,w2)==F){
    w<-temp
    Gdata<-disW(data,w,row,k)
    w2<-divG(Gdata,w,k,row,col)
    temp<-w2
```

```
  }
  size<-0
  wss<-0
  for(i in 1:k){
    size[i]<-length(which(Gdata[,col+1]==i))

  }
  wss<-wss_cal(Gdata,w,k)

  cat("K-means clustering with",k,"clusters of sizes:",size)
  cat("₩n","Clustering vector:","₩n",Gdata[,col+1])
  cat("₩n","wss:",wss)
  Gdata;
}

#최단거리 계산
disW<-function(data,w,row,k){
  gNum<-0
  dis<-0
  for(i in 1:row){
    for(j in 1:k){
      dis[j]<-dist(rbind(data[i,],w[j,]))
    }
    gNum[i]<-which.min(dis)
  }
  Gdata<-cbind(data,gNum)
  Gdata;
}

#군집의 평균 구하기
divG<-function(data,w,k,row,col){
  group<-col+1
  for(i in 1:k){
    for(j in 1:col){
      if(length(which(data[,group]==i))==0){
        w[i,j]<-w[i,j]
      }
      else if(length(which(data[,group]==i))==1){
        w[i,j]<-data[which(data[,group]==i),-(group)][j]
```

```
        }else{
          w[i,j]<-mean(data[which(data[,group]==i),-(group)][,j])
        }
      }
    }
    w;
}
wss_cal<-function(data,w,k){
    wss<-rep(0,k)
    group<-col+1
    for(i in 1:k){
      cluster_points<-data[data[,group]==i,-(group)]
      center<-w[i,]
      wss[i]<-sum(rowSums((cluster_points-center)^2))
    }
    wss;
}
```

[분석]

k-평균 군집분석을 데이터들간에 최단거리를 계산하는 disW function, 군집내에 데이터들간에 평균을 구하는 divG function, 군집내 오차제곱합을 구하는 wss_cal function 을 통해 최종 function 을 구성하였다.

disM 는 열의 개수가 784 개인 원데이터에 군집의 번호를 나타내는 열을 하나 추가하여 각 행들이 어느 군집에 속하는지 Gdata 로 나타내는 함수이다.

wss_cal 은 군집내에 중심점과 데이터들간에 오차제곱합을

divG 는 군집내에 데이터가 0 과 1 이 있을 때에는 평균을 구하지 않고 2 이상일 때부터 같은 군집안에 있는 데이터들간에 평균을 구한 후 중심점을 업데이트하는데 사용하는 함수이다.

최종 함수인 k_means 에서는 처음에 초기값을 랜덤한 점으로 지정해준다. 초기 중심점을 w, 바뀌는 중심점을 w2 라고 해주고 이 중심점들이 더 이상의 변화가 없을 때까지 disM( ), divG( ) 함수를 이용하여 반복하여준다.

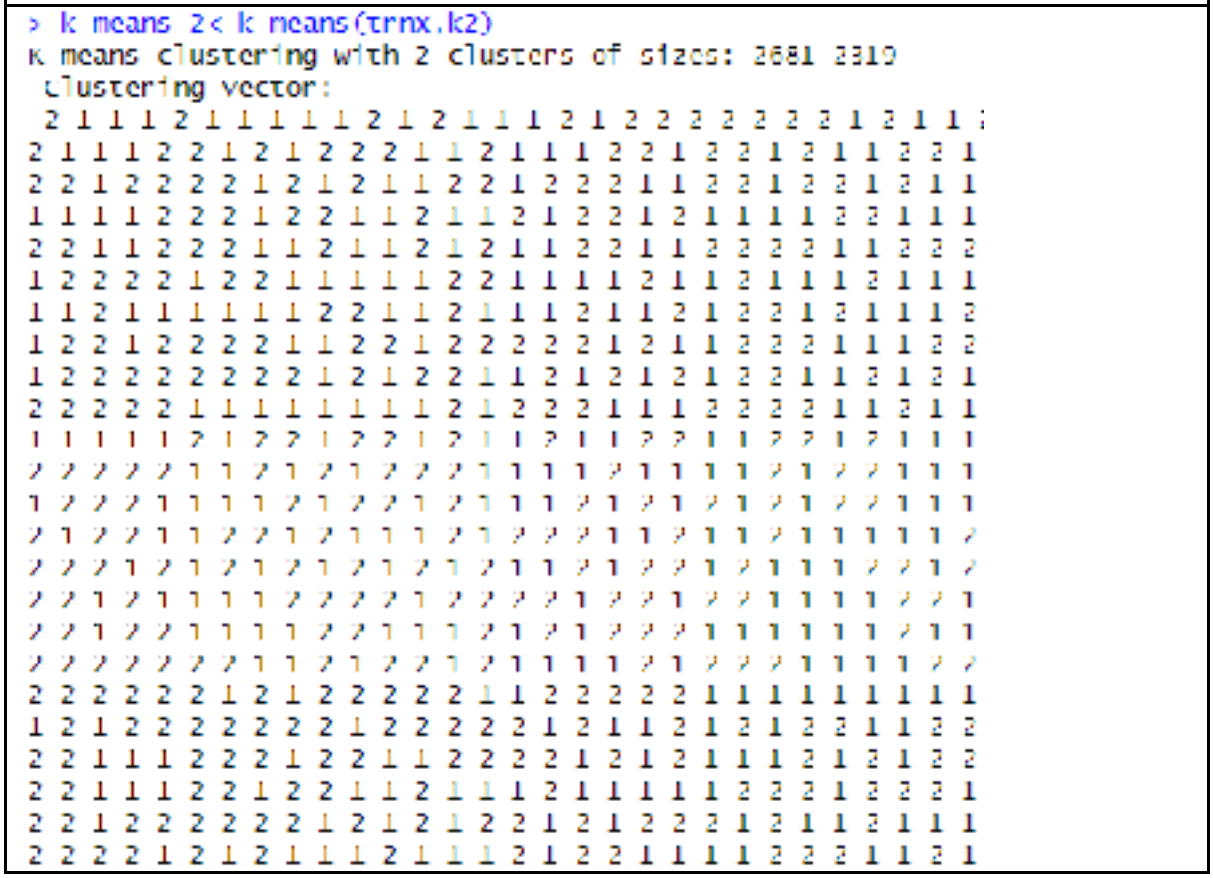
따라서, k_means 함수에 dataframe 과 군집의 수 k 를 입력하면 E-M 알고리즘을 이용하여 군집을 나누고 각 관측값의 clustering vector 와 각 군집의 wss 가 반환된다.

[R 코드] : 내가 작성한 k-means 함수와 kmeans( ) 비교

R 코드
#군집 수 : 2

```
k2<-2
k_means_2<-k_means(trnx,k2)
kmeans(trnx,k2)

#군집 수 : 3
k3<-3
k_means_3<-k_means(trnx,k3)
kmeans_3<-kmeans(trnx,k3)

#군집 수 : 4
k4<-4
k_means_4<-k_means(trnx,k4)
kmeans_4<-kmeans(trnx,k4)
```

실행결과



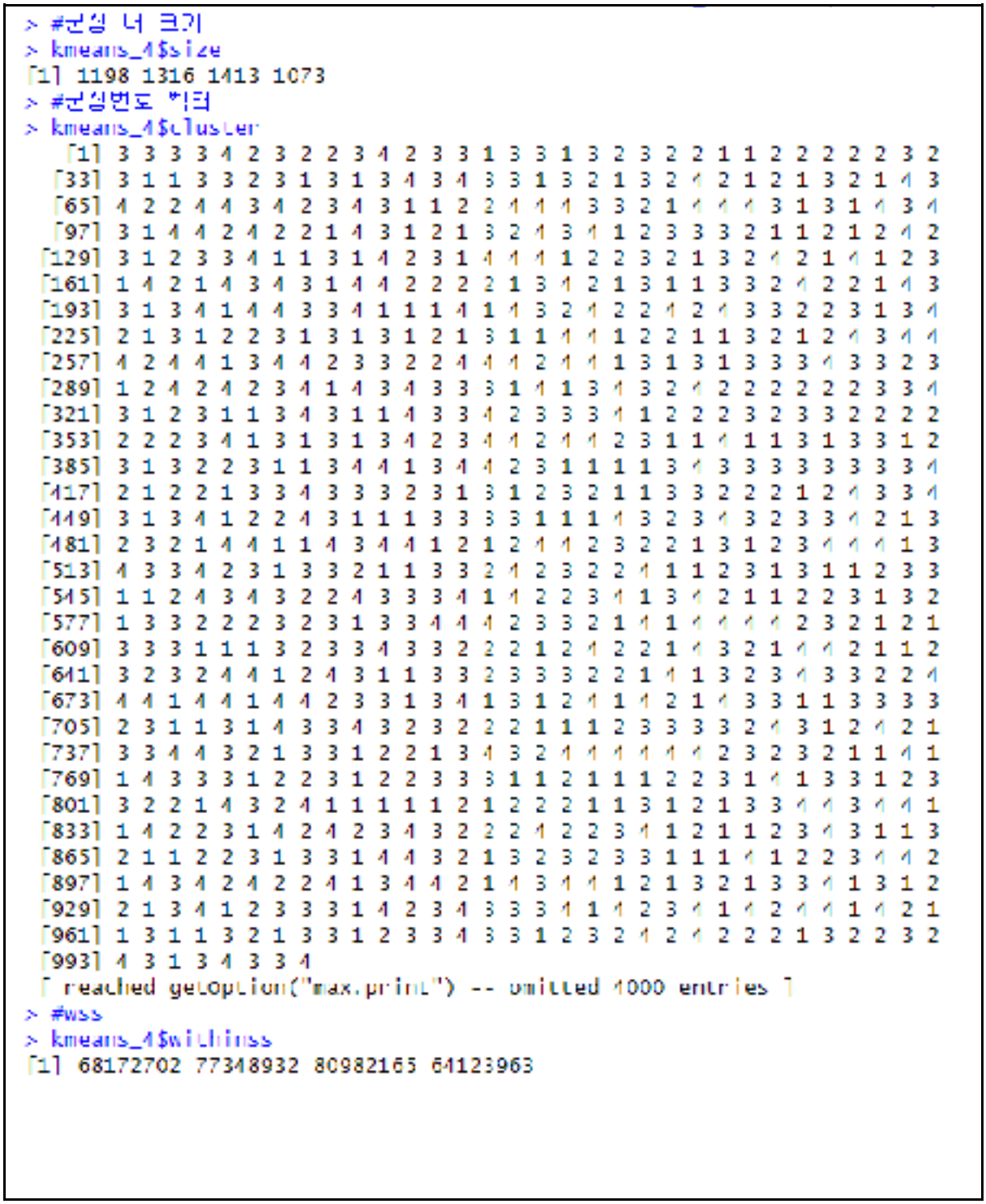
```
 2  2  2  2  1  2  2  1  2  2  1  2  2  1  1  1  2  1  1  1  2  1  1  1  1  2  2  2  1  1
 2  2  1  1  1  1  1  2  1  1  1  1  2  2  1  2  2  2  2  2  1  1  2  1  1  1  1  2  2  2  2
 1  1  2  1  2  1  1  2  1  2  1  2  1  2  1  1  1  2  2  2  2  2  1  2  1  2  1  1  2  1  2
 2  2  1  2  1  2  1  2  2  1  2  2  1  1  1  2  2  2  2  2  2  1  1  2  1  1  2  1  1  2  1
 2  2  1  1  1  1  1  2  1  1  1  2  2  1  2  1  1  1  2  2  2  1  1  1  1  1  1  2  1  2  2
 1  2  1  1  1  1  2  1  2  2  2  1  2  2  2  2  2  2  1  1  1  1  2  2  2  2  2  2  2  1  1
 1  2  2  1  2  2  1  2  2  1  2  1  1  2  1  1  1  1  2  1  1  2  1  2  1  2  1  1  2  2
 2  1  1  1  1  1  2  2  1  1  1  1  1  1  1  1  1  1  1  2  2  2  1  2  1  1  1  1  1  1
 1  1  2  1  1  2  1  2  2  1  2  2  1  1  2  1  1  1  1  2  2  1  1  2  2  2  2  1  2  2
 1  1  2  1  1  2  1  2  2  2  1  2  1  1  1  2  1  2  1  1  1  2  1  1  1  1  2  1  1  2
 2  2  1  2  1  1  2  2  2  1  1  2  1  1  2  2  2  2  1  1  1  1  1  1  2  1  1  2  1  1
 1  1  2  2  2  1  2  1  2  1  1  1  1  1  1  1  1  1  1  2  1  2  2  1  2  1  1  2  1  2
 2  2  1  1  2  1  1  2  1  1  2  2  2  2  1  1  2  2  2  1  2  1  2  1  2  1  1  2  1  1
 1  1  2  1  1  2  2  1  1  1  2  2  1  1  1  1  2  1  1  1  1  2  1  1  2  2  1  1  2
 2  2  1  2  2  1  1  1  2  1  1  1  1  1  1  2  1  1  2  1  1  1  2  1  1  2  2  2  1
 2  2  2  1  2  1  1  2  1  1  2  1  1  2  2  1  2  2  1  1  2  2  1  1  1  2  2  2  2  2
 1  2  1  2  2  1  2  1  1  2  1  1  2  1  2  1  1  1  2  1  1  2  1  1  1  2  1  2  2  1
 2  2  1  1  2  1  1  1  1  1  2  1  1  1  2  2  1  2  2  1  1  2  2  2  1  1  1  2  1  1
 2  2  2  1  1  1  1  1  1  2  2  2  2  1  2  1  2  2  1  2
 wss: 193000311 175488115
> #군집 내 크기
> kmeans_2$size
[1] 2327 2673
> #군집도표 벡터
> kmeans_2$cluster
   [1] 1 2 2 2 1 2 2 2 2 2 1 2 1 2 2 2 1 2 1 1 1 1 1 1 1
  [26] 2 1 2 2 1 1 2 2 2 1 1 2 1 2 1 1 1 2 2 1 2 2 2 1 1
  [51] 2 1 1 2 1 2 2 1 1 2 1 1 2 1 1 1 1 2 1 2 1 2 2 1 1
  [76] 2 1 1 1 2 2 1 1 2 1 1 2 1 2 2 2 2 2 2 1 1 1 2 1 1
 [101] 2 2 1 2 2 1 2 1 1 2 1 2 2 2 2 1 1 2 2 2 1 1 2 2 1
 [126] 1 1 2 2 1 2 2 1 2 1 2 2 1 1 2 2 1 1 1 1 2 2 1 1 1
 [151] 2 1 1 1 1 2 1 1 2 2 2 2 2 1 1 2 2 2 2 1 2 2 1 2 2
 [176] 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1 1 2 2 1 2 1 2 1 2 2
 [201] 1 2 1 1 2 1 2 2 2 1 2 1 2 2 1 1 1 1 2 2 1 1 2 1 1
 [226] 1 1 1 2 1 2 2 1 1 1 2 2 2 1 1 2 1 1 1 1 1 1 1 1 2
 [251] 1 2 1 1 2 2 1 2 1 2 1 2 1 1 2 2 1 2 1 2 1 1 1 1 1
 [276] 2 2 2 2 2 2 2 2 2 1 2 1 1 1 2 2 2 1 1 1 1 2 2 1 2 2
 [301] 2 2 2 2 2 2 1 2 1 1 2 1 1 2 1 2 2 1 2 2 1 1 2 2 1 1
 [326] 2 1 2 2 2 2 1 1 1 1 1 2 2 1 2 1 2 1 1 1 2 2 2 2 1 2
 [351] 2 2 2 2 1 2 1 1 2 2 1 2 1 1 1 2 2 2 2 1 2 1 1 2 1 2
 [376] 2 2 1 2 1 2 1 2 1 2 1 1 2 2 2 1 2 1 1 2 2 1 1 2 1
 [401] 2 2 2 2 1 2 1 1 1 2 2 1 2 2 1 2 2 2 2 2 1 1 1 1 2 1
 [426] 2 1 2 1 2 1 2 1 2 1 2 2 1 2 1 1 2 1 2 2 2 1 1 2 1
 [451] 1 1 2 1 2 2 2 2 2 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 2 2
 [476] 2 2 1 1 2 1 1 2 1 1 2 2 2 2 1 1 2 2 2 1 2 1 2 1 1
 [501] 1 2 2 2 2 2 2 2 1 2 2 1 1 1 1 1 1 1 2 2 1 2 1 1 2 1
 [526] 2 2 2 2 2 1 2 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 2 1 2 1
 [551] 1 1 1 1 1 2 2 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1
 [576] 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 2 1 2 1 2 1 1 2 2 1 1
 [601] 1 1 2 2 2 1 1 1 2 1 1 2 2 1 1 1 1 2 1 2 1 2 2 2 1
```

```
 [551] 1 1 1 1 2 2 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 2 1 1
 [576] 1 1 1 1 1 2 1 1 1 1 1 2 1 2 2 1 2 1 2 1 1 2 2 1 1
 [601] 1 1 2 2 2 1 1 1 2 1 1 2 2 1 1 1 1 2 1 2 1 2 2 2 1
 [626] 2 1 2 1 1 1 1 2 2 2 1 1 2 1 1 2 2 1 2 2 2 1 2 2 2
 [651] 2 2 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1 1 1 2 1 2 1 2 1
 [676] 1 2 1 2 1 1 1 2 1 2 2 1 2 2 2 1 1 1 1 2 1 2 1 2 2
 [701] 2 1 2 2 2 1 2 1 1 2 2 2 2 1 1 1 2 2 1 2 2 1 1 2 1
 [726] 1 2 1 1 1 1 1 2 1 1 2 1 2 2 2 1 1 2 2 1 1 2 2 2 1
 [751] 1 2 2 1 2 1 2 2 1 1 2 1 1 2 2 1 1 2 2 1 1 2 1 1 2
 [776] 2 1 1 2 1 1 2 1 1 2 1 2 2 2 1 2 1 2 2 2 2 2 2 1 1
 [801] 1 1 1 1 1 2 1 1 1 2 2 2 2 2 2 2 2 1 1 2 1 1 1 1 1
 [826] 1 1 1 1 1 1 1 2 2 1 1 1 2 2 2 1 1 2 2 2 1 2 1 1 2
 [851] 1 2 1 2 2 1 1 1 2 1 2 2 1 1 1 1 2 2 1 1 1 2 2 1 1
 [876] 2 2 1 2 1 2 2 2 2 2 2 2 1 1 1 2 2 2 1 2 2 1 1 1 1
 [901] 1 2 1 1 2 2 1 2 2 1 2 1 1 1 2 2 1 1 2 2 2 1 2 1 2
 [926] 2 1 1 1 1 2 2 2 2 2 2 2 2 2 1 1 2 2 1 1 1 2 2 2 1
 [951] 1 1 2 2 2 2 2 1 1 2 1 2 1 1 2 1 1 1 2 1 2 1 1 1 2
 [976] 2 1 2 1 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 1 1 2 2 1 2
 [ reached getOption("max.print") -- omitted 4000 entries ]
> #wss
> kmeans_2$withinss
[1] 150365215 170270530
```

```
> k_means_3<-k_means(trnx,k3)
K-means clustering with 3 clusters of sizes: 1779 1644 1577
 Clustering vector:
 3 2 1 1 1 1 1 1 3 3 2 3 3 3 3 3 2 1 3 3 1 3 2 3 1 1 2 1 1 2 2 3 1 2 2 1
2 1 1 2 2 1 1 1 3 1 2 3 1 2 1 1 2 2 3 1 1 2 1 2 2 3 3 3 2 3 2 2 3 2 1 3 2
1 2 2 1 1 1 3 3 2 2 1 2 1 1 1 3 2 1 2 1 3 2 3 3 2 1 3 2 1 1 2 1 1 3 1 2 3
3 1 2 3 3 3 2 3 1 1 1 1 3 3 2 1 2 1 1 3 3 2 1 2 1 3 3 1 1 2 1 3 2 1 3 2 3
2 2 2 3 3 1 3 1 2 3 2 2 1 1 1 2 3 2 1 1 3 1 2 3 2 3 1 3 3 1 2 2 1 3 2 1 3
2 1 3 1 2 3 2 2 3 3 1 1 1 1 2 3 3 3 2 3 3 3 1 3 2 3 1 1 1 1 1 2 3 1 1 3 1
1 2 2 3 1 2 2 1 3 1 3 2 3 2 1 3 2 3 1 1 3 2 2 3 2 2 2 2 1 3 3 1 3 2 2 1 3
1 2 3 2 1 1 1 1 3 3 1 3 3 1 3 3 3 3 2 3 2 1 3 2 3 3 3 3 1 2 2 2 2 3 1 2 3
3 3 3 1 3 3 2 2 1 1 3 3 3 3 1 2 2 1 2 2 2 1 3 1 3 2 1 3 1 2 3 3 3 2 2 2 3
1 3 2 1 1 2 1 3 2 3 1 2 3 1 1 2 3 3 3 2 3 1 1 3 1 3 2 2 1 3 3 2 2 2 1 3 3
3 2 3 2 2 1 2 2 2 2 3 2 2 2 2 1 1 2 3 1 2 1 3 1 3 3 1 1 1 2 2 2 3 1 2 3 1
3 3 1 3 2 3 1 1 3 2 3 1 3 2 3 2 2 2 1 2 1 2 3 1 2 2 1 3 3 1 2 2 3 3 2 1 1
1 3 1 2 1 3 3 2 1 3 2 1 3 1 1 3 3 3 1 3 1 1 2 2 1 1 3 3 3 3 2 2 3 2 3 2 1
2 1 1 3 1 1 1 2 2 3 2 2 1 2 3 2 3 2 3 3 3 2 3 2 1 3 2 3 2 3 3 2 1 2 1 2 3
3 1 3 3 2 3 3 3 1 2 2 3 1 1 1 1 3 2 3 2 1 2 3 1 2 3 3 2 1 1 2 2 2 3 2 1 1
1 2 1 2 1 1 1 2 3 2 1 3 1 2 2 2 2 3 2 3 2 2 3 1 1 1 3 3 2 1 3 3 2 1 1 1 1
2 1 2 1 3 2 3 1 1 3 3 2 2 3 3 1 2 2 1 2 1 3 2 1 1 2 2 3 1 3 3 1 1 3 3 1 2
2 2 3 1 2 1 1 1 3 2 2 1 1 1 2 1 2 1 1 1 2 2 1 3 2 3 3 2 1 3 2 2 3 1 1 3 3
2 3 3 1 3 1 3 1 1 1 2 3 1 1 1 1 3 1 2 1 3 2 1 1 2 1 1 3 1 1 3 3 1 2 2 3 3
3 1 3 1 2 1 2 2 3 3 2 2 3 2 1 2 3 2 2 3 3 1 1 3 3 2 2 1 1 2 1 2 1 1 1 2 2
2 3 1 3 3 2 3 2 3 3 1 2 3 1 1 2 3 1 3 3 2 2 2 2 2 2 2 1 2 3 2 2 2 3 3 2 3
2 2 2 3 1 2 1 1 3 1 2 2 2 1 3 2 3 3 2 2 1 2 2 1 1 1 3 2 2 2 3 1 1 1 2 3 3
3 1 2 2 2 2 2 1 2 3 2 2 3 3 2 3 1 1 1 3 3 3 3 3 2 2 2 2 1 1 1 1 1 2 1 3 1
1 3 2 3 2 2 1 1 2 2 2 1 1 1 3 2 2 2 3 2 3 2 1 1 1 1 1 2 1 3 2 1 1 3 3 2 3
1 3 3 2 1 2 2 2 3 1 1 2 3 2 3 2 2 1 2 2 3 2 1 2 3 2 3 2 2 1 3 1 1 3 2 2 3
3 2 2 3 1 1 1 3 2 2 1 3 2 1 3 3 1 3 1 1 1 1 1 3 1 1 1 3 2 1 1 3 1 3 1 2 2
2 3 3 3 3 3 3 2 3 1 3 1 2 2 2 2 2 3 1 1 3 3 2 3 3 3 2 3 1 3 2 1 1 2 3 3 1
2 1 1 1 2 3 1 1 3 2 1 1 3 3 2 1 3 3 3 3 3 1 3 2 2 1 2 3 2 1 1 1 2 1 1 2 3
1 2 3 2 3 1 2 2 2 1 2 1 1 3 2 1 3 2 2 3 1 2 2 2 1 2 2 1 3 1 2 3 1 3 3 3 3
2 3 1 2 1 3 2 1 1 1 3 1 1 1 1 1 2 3 3 2 1 2 2 3 2 3 3 1 1 3 1 2 2 2 3 3 1
1 3 3 1 2 2 1 3 3 1 2 1 2 2 1 3 3 1 1 3 1 3 2 1 1 1 3 2 1 3 1 2 2 2 3 3 2
1 3 3 1 3 3 3 1 1 3 2 2 3 1 1 1 3 1 1 3 1 2 2 2 3 1 1 2 2 2 1 2 3 1 3 2 2
1 1 3 2 1 3 3 2 2 3 2 3 1 2 3 3 2 1 3 3 2 1 3 1 2 3 2 3 3 2 2 2 3 1 1 2 3
2 2 1 3 2 2 2 3 2 2 2 1 1 1 1 1 1 2 2 3 1 3 1 2 2 3 3 3 3 3 3 1 2 3 2 1 3
2 1 1 2 3 3 3 1 3 2 3 1 1 1 3 3 3 2 3 3 3 1 1 1 2 3 3 3 3 2 2 2 3 2 2 1 1
3 2 1 3 2 1 1 1 1 1 2 2 1 1 1 3 3 2 1 3 1 2 1 1 3 2 3 1 2 2 1 3 3 2 1 1 3
2 1 2 1 2 1 2 3 1 3 3 1 2 2 3 3 1 1 2 1 3 1 3 1 3 3 3 1 2 3 3 3 2 3 2 1 3
1 1 1 3 2 1 1 3 2 3 2 2 2 2 3 1 3 3 1 3 1 3 1 1 3 1 3 1 2 3 1 3 3 2 3 1 2
3 2 1 2 3 2 1 3 3 3 2 3 2 1 1 3 3 2 1 2 3 3 1 1 3 3 1 1 3 2 3 3 1 3 3 1 1
1 2 2 1 3 1 3 3 1 1 3 2 2 2 3 2 2 3 3 3 1 2 3 2 1 2 1 2 3 1 2 1 1 3 1 2 2
1 2 2 1 1
 wss: 140477985 118854613 126943222
```

```
> kmeans_3$size
[1] 1781 1518 1671
> #군집번호 센터
> kmeans_3$cluster
   [1] 1 2 2 1 2 3 1 2 1 1 2 2 2 2 1 1 3 3 1 2 1 2 2 2 1 3 3 2 2 3 3 1
  [33] 3 2 3 3 1 2 3 2 2 3 3 1 3 1 1 2 1 1 2 2 2 2 2 1 2 3 3 2 2 2 1 1
  [65] 1 3 1 3 3 1 3 2 1 2 2 3 2 3 2 3 1 2 3 2 2 3 3 3 3 1 3 2 3 3 1 3
  [97] 1 1 2 2 1 3 1 2 3 2 1 1 3 3 1 1 3 2 2 1 1 2 2 2 3 3 3 1 1 2 2 3
 [129] 1 1 1 2 2 1 2 2 1 2 1 2 3 2 1 3 3 1 2 1 3 3 2 1 1 3 1 3 2 1 3 2
 [161] 1 2 1 2 1 3 3 3 1 1 3 1 2 1 3 2 1 3 2 2 1 1 3 3 1 2 1 1 3 3 1 3
 [193] 3 1 1 2 2 3 2 2 1 1 1 3 2 1 1 2 1 2 1 2 1 3 2 3 3 2 1 2 1 2 1 3
 [225] 2 1 1 2 2 1 1 3 1 3 1 3 3 1 3 1 2 3 1 2 3 1 3 3 3 2 3 1 1 3 1 3
 [257] 3 1 1 3 2 2 3 3 3 3 3 2 1 2 1 1 2 1 1 1 2 3 1 3 1 1 3 1 1 1 1 3
 [289] 3 2 3 2 1 1 3 1 1 1 1 1 1 1 3 2 1 1 1 1 1 2 3 3 3 1 2 2 2 1 1 2
 [321] 3 3 1 2 2 2 1 1 1 3 3 2 1 2 1 3 2 2 2 3 1 2 1 2 2 1 3 3 2 1 1 1
 [353] 3 1 3 1 2 3 1 2 3 1 1 1 3 2 1 3 1 1 1 2 1 3 3 1 2 2 2 3 1 3 3 3
 [385] 3 1 2 3 1 3 2 3 1 1 2 1 3 1 3 3 3 3 1 2 2 1 1 1 1 2 1 2 1 1 2 1
 [417] 2 1 3 1 2 2 2 3 3 1 3 2 3 1 1 3 3 3 2 1 3 2 3 1 1 2 2 3 3 1 2 2
 [449] 3 1 1 3 1 2 2 3 1 2 1 1 1 1 3 2 2 2 3 2 3 1 1 1 1 1 2 3 1 3 1 3
 [481] 2 3 2 3 1 1 3 2 2 2 1 2 3 3 2 1 3 1 2 1 1 1 3 1 3 2 1 3 1 3 1 1
 [513] 3 2 2 1 2 1 1 2 1 2 2 2 1 1 1 3 2 2 1 2 2 3 3 1 2 1 3 3 3 1 3 3 2
 [545] 1 2 1 2 3 2 3 1 3 3 2 3 2 1 3 2 1 3 3 2 3 3 1 3 1 3 3 3 1 2 1 3
 [577] 2 2 3 1 1 1 1 2 3 1 2 2 3 1 1 2 2 3 2 2 1 2 1 2 1 1 1 3 3 1 1 1
 [609] 2 3 3 2 3 1 2 3 1 2 3 1 2 2 1 2 2 2 1 1 1 2 3 2 1 2 3 3 3 3 1 3 2
 [641] 2 1 3 2 1 3 1 3 2 3 2 2 1 2 1 3 3 1 1 2 2 3 3 1 1 1 3 1 2 1 1 2
 [673] 1 2 1 3 2 1 2 1 3 2 1 2 2 2 1 2 2 1 3 1 3 1 2 3 1 1 2 3 3 1 2 2
 [705] 3 1 1 3 3 3 2 1 1 3 2 1 3 1 3 1 3 3 2 2 2 1 1 1 3 3 3 1 2 3 2 2
 [737] 1 2 3 3 2 1 2 1 1 3 1 2 1 1 3 3 1 3 1 2 1 2 1 1 2 3 3 3 3 2 2 2
 [769] 2 1 3 2 3 1 1 2 1 1 2 1 3 2 2 3 2 2 3 1 2 3 3 1 2 1 2 3 1 1 2 1
 [801] 2 3 3 1 1 1 2 3 2 3 1 2 3 2 3 2 3 3 3 2 2 3 1 2 3 2 3 3 1 1 3 1
 [833] 1 3 2 2 3 3 1 3 3 2 3 2 1 2 1 1 1 3 1 1 2 3 1 2 1 3 1 2 3 3 2 1
 [865] 1 2 3 1 1 2 2 3 1 2 3 2 3 2 3 3 1 2 3 3 3 3 3 2 1 3 2 1 1 3 3 3
 [897] 2 1 1 1 2 1 2 3 2 1 3 2 2 1 2 3 2 3 3 2 2 1 2 1 2 1 3 3 2 3 3 1
 [929] 1 2 2 3 2 3 2 1 1 1 3 3 1 3 1 1 1 3 3 1 2 1 1 1 3 3 3 2 1 2 2 2
 [961] 3 2 3 2 3 3 3 1 1 2 2 1 2 2 2 2 3 2 1 1 3 1 2 2 2 1 2 1 3 3 1 1
 [993] 1 1 3 3 3 1 3 3
 [ reached getOption("max.print") -- omitted 4000 entries ]
> #wss
> kmeans_3$withinss
[1] 110163068  93931307  99101666
```

```
> k means 4< k means(trnx.k4)
k means clustering with 4 clusters of sizes: 1341 1153 1119 1387
 clustering vector:
 3 4 1 1 3 1 1 1 4 4 2 1 3 4 4 4 2 1 3 3 3 3 2 2 1 1 2 1 1 2 2 4 1 4 2 3 4
2 1 1 2 2 1 1 2 1 1 1 3 3 1 2 1 4 2 4 1 2 2 1 2 2 4 3 3 2 3 2 2 3 2 1 4 2
3 4 2 1 1 1 3 3 2 4 1 2 1 2 1 4 4 1 2 1 3 2 3 4 2 2 3 1 3 4 4 3 1 2 2 4 3
4 1 4 4 3 3 4 4 1 2 1 1 1 3 2 2 4 1 3 4 4 2 1 2 1 4 3 3 1 1 2 3 2 1 4 4 3
4 2 4 3 3 3 3 1 2 3 4 4 1 1 1 2 3 4 1 1 4 3 4 3 4 4 1 4 3 1 4 4 1 4 2 1 3
4 1 4 1 2 3 4 4 3 3 1 1 2 1 4 3 3 3 2 1 3 4 1 4 4 3 2 1 1 2 2 2 3 1 1 3 2
1 2 2 3 3 2 4 3 3 1 3 2 3 4 1 4 2 3 1 3 3 2 2 3 2 2 2 4 2 4 3 1 3 4 2 1 3
1 2 4 2 2 1 4 2 4 3 1 3 3 3 3 3 4 4 4 3 4 1 3 1 3 3 3 3 2 4 4 2 2 3 3 2 4
4 3 4 1 4 3 4 1 1 3 4 3 3 4 2 2 4 3 4 4 2 1 1 2 2 4 1 2 2 4 3 4 4 4 2 2 3
3 3 4 1 2 4 2 4 2 3 2 4 4 1 1 2 1 4 4 4 2 1 3 3 1 4 4 4 1 3 3 4 4 4 1 3 4
3 2 4 2 4 1 4 4 2 2 4 2 4 2 4 3 2 4 4 1 2 1 4 3 4 3 2 3 1 2 4 4 4 3 4 3 3
3 1 1 1 4 4 1 1 4 4 4 1 3 2 3 2 2 2 1 2 1 2 4 3 2 2 1 3 4 1 2 4 3 3 4 1 1
1 4 1 2 1 1 1 2 1 1 4 1 3 1 1 3 3 4 1 3 1 1 2 4 1 3 3 3 3 4 4 4 4 2 1 4 1
2 1 2 1 1 1 1 4 2 1 4 4 1 4 3 2 4 2 3 3 1 4 3 1 1 4 2 4 4 3 3 2 2 2 1 2 1
4 2 1 2 2 4 1 4 1 4 4 2 1 3 3 2 4 4 4 4 3 2 3 2 2 3 3 2 3 2 4 2 2 3 2 1 1
1 2 1 2 1 1 1 4 4 4 1 1 1 4 2 4 2 4 2 3 2 2 2 2 3 1 3 3 2 2 3 4 2 1 1 1 1
2 1 2 3 3 4 3 1 3 3 1 1 4 3 3 3 4 2 3 4 1 3 2 1 3 4 2 1 2 4 4 1 2 4 3 1 2
2 2 3 1 4 1 2 1 4 2 2 1 4 1 4 1 4 3 1 1 2 4 1 3 2 3 4 2 1 3 4 2 3 1 3 3 3
2 3 3 1 3 1 3 1 3 2 4 3 1 3 2 2 4 3 4 1 3 4 1 1 2 1 2 3 1 1 4 2 1 4 4 3 4
4 1 3 1 2 2 4 4 4 4 2 4 3 4 1 2 4 4 2 3 4 2 3 4 3 2 2 2 1 2 2 2 1 3 1 4 1
4 2 1 4 3 2 4 4 4 3 3 4 4 1 1 2 4 1 3 3 4 2 2 2 4 2 2 1 1 3 2 2 2 3 4 4 3
4 1 3 2 4 1 2 1 2 1 1 4 2 4 3 1 1 4 1 1 4 2 3 3 2 2 3 3 4 3 2 2 1 1 4 1 4
1 4 1 2 2 1 2 4 3 3 2 2 2 2 3 2 3 3 3 1 2 2 1 4 1 4 2 2 1 4 3 2 1 3 3 4 3
4 2 1 4 1 3 2 1 3 4 1 2 3 1 2 4 1 3 2 2 1 4 2 2 4 4 2 4 2 3 1 4 1 4 1 4 2
1 1 3 1 1 4 1 1 4 3 1 3 1 1 2 1 3 2 3 4 2 2 2 1 4 2 3 2 1 4 2 1 1 3 4 2 3
4 4 4 3 2 2 3 1 3 1 2 4 4 2 4 4 4 4 2 2 3 2 4 1 1 1 4 4 2 2 4 1 1 3 4 3 2
3 1 2 2 2 2 2 2 4 3 2 4 1 3 2 1 3 1 1 1 4 4 3 4 2 2 4 1 1 2 1 1 1 4 1 3 1
1 3 1 3 2 4 2 3 4 4 4 3 2 1 3 2 2 2 4 1 4 2 2 2 3 3 1 2 2 4 2 1 1 4 3 2 3
1 3 3 1 3 4 2 2 3 1 2 4 4 2 3 2 4 2 4 4 4 2 3 2 3 2 3 4 4 3 4 3 1 4 2 2 3
3 2 2 3 1 2 2 4 2 2 1 3 2 1 4 4 1 4 1 1 2 1 1 1 1 2 2 3 4 1 2 3 1 4 1 4 4
2 4 4 4 4 3 3 4 3 3 3 1 1 4 2 1 4 4 1 2 4 3 2 3 4 3 4 3 1 3 2 1 2 4 3 4 3
4 1 1 3 2 3 2 1 3 4 2 1 4 3 4 1 3 3 4 3 4 2 4 2 2 1 2 3 4 1 1 2 2 1 2 2 4
1 2 1 2 3 1 4 2 1 2 2 1 1 4 2 1 3 4 4 4 1 2 4 2 1 4 4 1 3 1 1 4 1 4 3 3 4
2 2 1 2 4 4 4 1 3 1 3 2 3 1 1 3 2 3 3 4 1 4 4 3 2 3 3 3 2 4 1 4 2 2 3 4 1
3 3 3 2 4 2 1 1 3 1 4 1 1 2 1 4 3 1 2 4 3 4 4 3 2 3 4 4 1 4 1 2 2 2 4 3 4
1 4 4 1 3 1 4 3 3 3 4 2 4 1 3 1 4 1 1 4 1 4 2 2 1 3 3 4 2 2 1 4 3 1 3 1 4
2 1 4 4 3 3 3 2 2 3 2 4 1 2 4 4 2 1 3 3 4 3 4 1 4 4 2 4 3 2 4 4 3 1 1 4 4
2 4 1 3 2 2 4 3 4 4 4 2 2 1 1 2 1 4 2 3 3 3 1 4 4 3 4 3 3 1 3 3 4 4 4 1 3
2 1 1 2 3 3 4 1 4 4 4 1 1 1 4 4 4 4 3 3 1 3 1 1 2 4 3 3 3 4 1 2 4 4 2 3 1
3 2 1 3 2 1 1 2 1 1 2 2 1 1 1 4 3 2 3 4 1 4 2 1 4 2 3 1 4 4 2 3 1 4 1 1 4
2 1 4 1 4 2 1 1 2 3 3 1 2 2 3 4 3 2 4 1 4 2 1 1 4 3 3 1 4 3 4 4 2 4 4 1 3
1 1 1 3 2 1 1 3 2 3 2 4 4 2 4 1 3 4 1 3 1 3 3 1 4 1 3 1 4 1 1 3 3 2 3 3 4
3 4 3 2 4 2 1 4 3 4 4 3 4 3 1 4 4 2 1 4 2 4 1 1 3 1 3 1 3 2 3 4 1 3 3 1 1
1 4 2 1 1 1 3 3 1 3 2 4 4 2 3 2 4 1 4 3 1 1 3 2 2 4 1 4 1 1 4 1 3 3 2 4 2
1 2 2 1 3
 wss: 108948262 92385431 89962144 103097136> kmeans_4<-kmeans(trnx,k4)
```



[분석]

군집 수 k 를 2 라고 하였을 때,

내가 작성한 함수인 k_means 는

size : 2681 2319

wss: 193600311 175488115

kmeans( ) 함수를 이용하여 구한 결과는

size : 2327 2673
wss : 150365215 170270530
가 나온다.

군집 수 k 를 3 이라고 하였을 때,
k_means( )
size : 1779 1644 1577
wss: 140477985 118854613 126943222
kmeans( )
size : 1781 1548 1671
wss : 110463068 93934307 99101666

군집 수 k 를 4 라고 하였을 때,
k_means( )
size: 1341 1153 1119 1387
wss: 108948262 92385431 89962144 103097136

kmeans( )
size : 1198 1316 1413 1073
wss : 68172702 77348932 80982165 64123963

내가 작성한 k-평균 군집분석 함수인 k_means 와 R studio 에 내장된 함수인 kmeans 함수를 wss(군집 내 오차제곱합)으로 비교해보면 군집수인 k 를 변환시켜주어도 kmeans 함수의 wss 가 더 작게 나오는 것을 알 수 있다.
이는 내가 작성한 함수인 k_means 에서는 초기 중심값을 무작위로 선택한 후 반복 조건을 while 문을 사용하여 중심값이 수렴할 때까지 반복시키는 것으로 작성하였는데
kmeans 함수는 k-means++ 초기화 방법을 사용하여 초기화 방법을 이용하여 초기 중심값을 선책하는데 이 값은 데이터를 잘 대표하는 중심값을 선택하므로 알고리즘의 성능과 결과가 더 좋게 나온다. 또한, kmeans 는 반복 중지 임계값인 tol=1e-06 이 기본값으로 설정되어있어 수렴 조건인 중심값의 이동 거리나 할당 변화량이 임계값보다 작을 때까지 반복한다.
따라서, E-M 알고리즘을 통해 중심값을 갱신하며 k-평균 알고리즘을 진행하는 것을 동일하지만 파라미터값들의 기준값 지정의 차이로 인하여 결과값에도 차이가 있음을 알 수 있다.

4. 결론

변수들을 회전변환하여 선형식인 주성분으로 표현하는 주성분 분석을 진행하여 차원을 축소해주었고 주성분 분석을 통해 차원축소한 데이터셋들을 svm 모델에 적용하였을 때의 평가지표를 확

인해보았다. 또한, 주성분 분석을 하지 않은 차원이 큰 데이터셋을 k-means 군집분석을 진행해보았다.

이 과정을 통하여 데이터셋이 간단하고 차원이 크다면 비선형성에 의해 과적합 문제가 발생할 수 있어 모델을 적용할 때 정확도가 낮게 나올 수 있다. 따라서, 주성분 분석을 통해 적절한 주성분의 수를 확인하고 차원을 축소한 후 분류분석과 군집분석을 진행한다면 더 좋은 성능과 결과가 나올 수 있다는 것을 알게 되었다.