hw8

Jiayi

2024-10-31

```r
knitr::opts_chunk$set(echo = TRUE)
library(faraway)      # Contains the datasets
library(ggplot2)      # For data visualization
library(pls)          # For principal component regression (PCR)
```

```
##
## Attaching package: 'pls'

## The following object is masked from 'package:stats':
##
##     loadings
```

```r
library(glmnet)       # For ridge and lasso regression
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```r
library(MASS)
library(ModelMetrics) # For RMSE calculation
```

```
##
## Attaching package: 'ModelMetrics'

## The following object is masked from 'package:base':
##
##     kappa
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':
##
##     select
## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Problem 1:

1.(a)

```
data(seatpos)
?seatpos

# Define response and explanatory variables
y <- seatpos$hipcenter
X <- seatpos %>% dplyr::select(-hipcenter) # Exclude hipcenter for predictors
```

PCA without Scaling

```
# Perform PCA without scaling
pca_no_scale <- prcomp(X, scale = FALSE)
U_no_scale <- pca_no_scale$rotation

# Display the loading matrix U without scaling
print("Loading matrix U without scaling:")
```

```
## [1] "Loading matrix U without scaling:"
```

```
U_no_scale
```

```
##                 PC1          PC2        PC3         PC4          PC5
## Age     -0.02656516 -0.973286219 -0.1993345  0.07358807 -0.061513592
## Weight  -0.92053301 -0.055430205  0.3855962 -0.02766099 -0.008009736
## HtShoes -0.25254069  0.133886012 -0.5807696  0.14807810  0.019193825
## Ht      -0.25317632  0.141520720 -0.5773155  0.06234507  0.033276939
## Seated  -0.10445187  0.085258885 -0.2152331  0.34707440 -0.550865336
## Arm     -0.06360299 -0.059300292 -0.1585311 -0.17479834  0.597921590
## Thigh   -0.06177741 -0.001172272 -0.2173435 -0.90376485 -0.318988403
## Leg     -0.07256890  0.030014565 -0.1493290  0.01412876  0.481621965
##                  PC6          PC7          PC8
## Age      0.048324829  0.024698396  0.011381619
## Weight   0.002826413 -0.004297615 -0.002508137
## HtShoes  0.218015623 -0.184489007 -0.690754566
## Ht       0.186899554 -0.155203232  0.720115741
## Seated  -0.631124887  0.336498173  0.005703586
## Arm     -0.708809326 -0.275978866 -0.025813714
## Thigh   -0.051116030  0.157142748 -0.055612712
## Leg      0.108868432  0.852735648 -0.019077860
```

Perform PCA with scaling

```
pca_scaled <- prcomp(X, scale = TRUE)
U_scaled <- pca_scaled$rotation

# Display the loading matrix U with scaling
print("Loading matrix U with scaling:")
```

```
## [1] "Loading matrix U with scaling:"
```

```
U_scaled
```

```
##                  PC1        PC2         PC3         PC4        PC5         PC6
## Age     -0.007219379 -0.8763467 -0.16383976 -0.16522774 -0.3349932 -0.25464449
## Weight  -0.366979122 -0.0448877 -0.42981137 -0.60025209  0.5537489  0.09798202
## HtShoes -0.411460536  0.1055831 -0.03375209  0.02577245 -0.2204816 -0.05101900
## Ht      -0.412057421  0.1119799 -0.01116858  0.02294603 -0.1887759 -0.04369735
## Seated  -0.381270226  0.2178995 -0.17138740 -0.15033847 -0.6171009  0.23019712
## Arm     -0.348771387 -0.3742641  0.01670980  0.55358297  0.2380225  0.60781701
## Thigh   -0.327523319 -0.1251793  0.86246173 -0.31151283  0.1038969 -0.06344739
## Leg     -0.389747512  0.0555930 -0.11688322  0.43024468  0.2205229 -0.70326773
##                 PC7          PC8
## Age      0.02269849 -0.015528966
## Weight  -0.04435483  0.008082356
## HtShoes  0.53650776  0.691876699
## Ht       0.50884054 -0.721493879
## Seated  -0.56689080 -0.002844309
## Arm     -0.07347462  0.007539785
## Thigh   -0.14492761  0.018814564
## Leg     -0.32092126  0.005274142
```

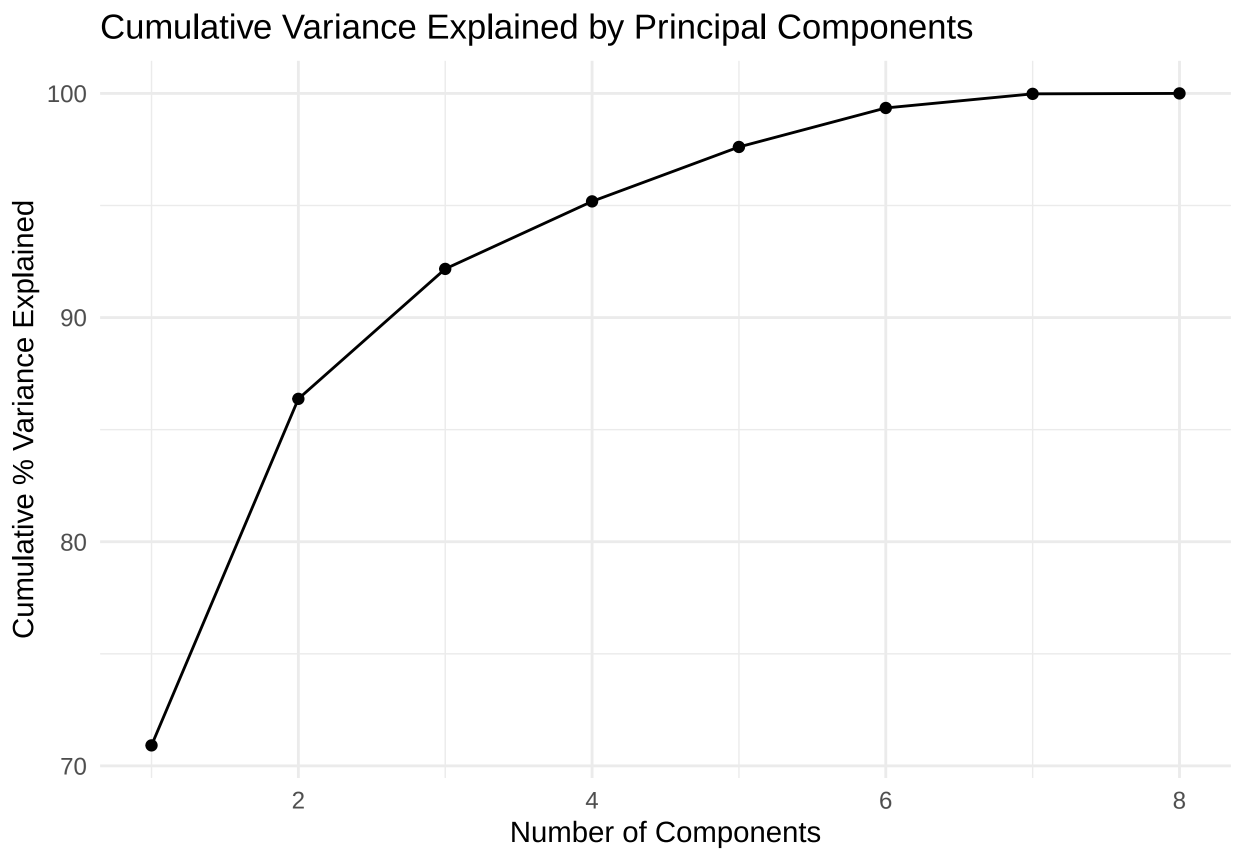
1. (b) Yes, scaling helps in this case. Since the variables likely have different units and ranges (e.g., Weight versus Arm or Leg length), scaling normalizes their contribution to the principal components. This prevents variables with larger scales from dominating the analysis, providing a more interpretable and balanced principal component structure.

I'll explain this by looking at two charts. Chart1: Without Scaling: The loadings for each principal component are heavily influenced by the variables with larger absolute values (like Weight and hipcenter). This is because variables with larger variances tend to dominate the first few components in unscaled PCA, leading to biased interpretations. Chart2: With Scaling: After scaling, each variable has a mean of zero and a standard deviation of one, so each variable contributes equally to the principal components regardless of their original units. The loadings are now more balanced, and the PCA reflects the underlying structure of the data rather than being skewed by the scale of individual variables.

1. (c)

```
# Calculate % variance explained by each component
var_explained <- pca_scaled$sdev^2 / sum(pca_scaled$sdev^2) * 100
cum_var_explained <- cumsum(var_explained)
```

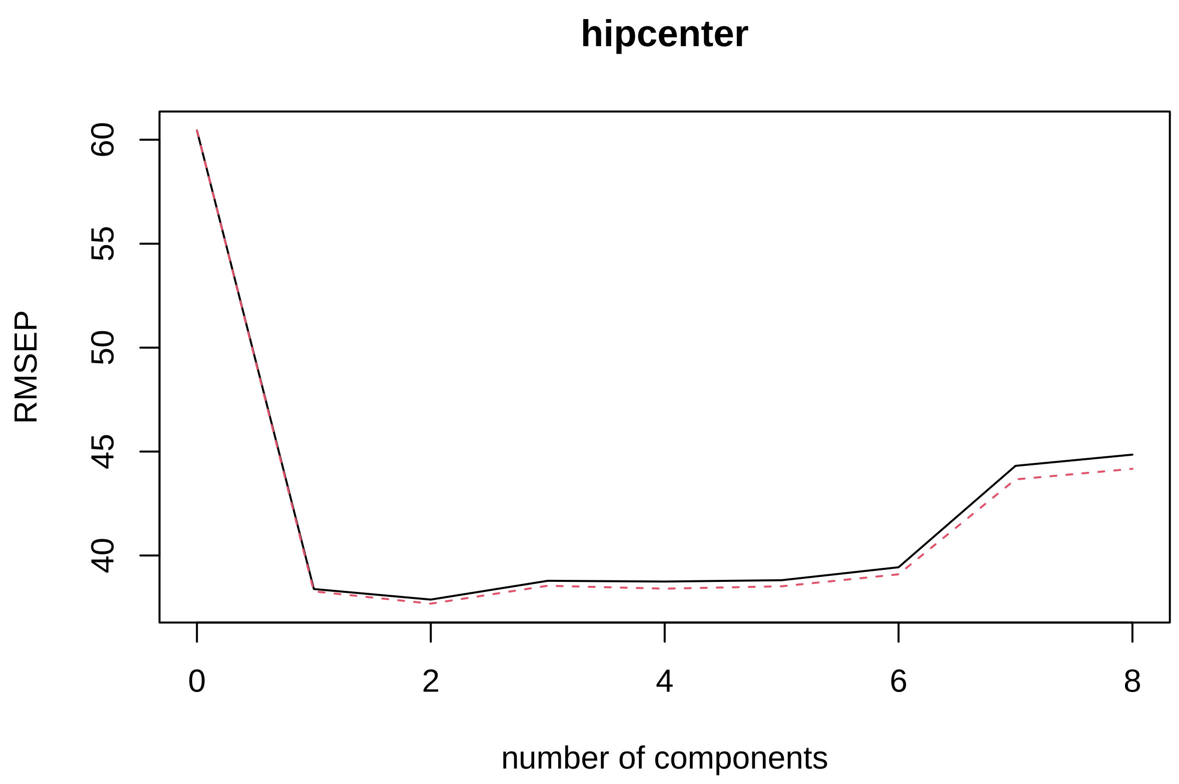
```r
# Plot cumulative variance explained
ggplot(data.frame(Components = 1:length(var_explained), Variance = cum_var_explained), aes(x = Componen
  geom_line() + geom_point() +
  labs(title = "Cumulative Variance Explained by Principal Components", x = "Number of Components", y =
  theme_minimal()
```



1.(d)

```r
# Perform PCR with cross-validation to select optimal number of components
pcr_model <- pcr(hipcenter ~ ., data = seatpos, scale = TRUE, validation = "CV")

# Plot RMSEP to find optimal components
validationplot(pcr_model, val.type = "RMSEP")
```



```r
summary(pcr_model)
```

```
## Data:    X dimension: 38 8
##  Y dimension: 38 1
## Fit method: svdpc
## Number of components considered: 8
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           60.45    38.39    37.88    38.78    38.75    38.81    39.43
## adjCV        60.45    38.27    37.69    38.55    38.41    38.52    39.10
##        7 comps  8 comps
## CV       44.31    44.85
## adjCV    43.66    44.17
##
## TRAINING: % variance explained
##            1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X            70.91    86.37    92.17    95.18    97.61    99.35    99.98
## hipcenter    61.89    66.34    66.48    67.82    68.02    68.58    68.63
##            8 comps
## X           100.00
## hipcenter    68.66
```

Problem 2:

```r
data(fat)

X_fat <- as.matrix(fat[, c("neck", "chest", "abdom", "hip", "thigh", "knee", "ankle", "biceps", "forearn
y_fat <- fat$brozek

# Split data into training and testing sets
set.seed(123)
train_indices <- sample(1:nrow(X_fat), 0.7 * nrow(X_fat))
X_train <- X_fat[train_indices, ]
y_train <- y_fat[train_indices]
X_test <- X_fat[-train_indices, ]
y_test <- y_fat[-train_indices]
```

```r
# Ridge Regression with Cross-Validation
ridge_cv_fat <- cv.glmnet(X_train, y_train, alpha = 0)  # alpha = 0 for Ridge
ridge_model_fat <- glmnet(X_train, y_train, alpha = 0, lambda = ridge_cv_fat$lambda.min)

# Ridge Regression Coefficients
print("Ridge Regression Coefficients:")
```

```
## [1] "Ridge Regression Coefficients:"
```

```r
print(coef(ridge_model_fat))
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) -10.87440261
## neck         -0.40743254
## chest         0.11237909
## abdom         0.59985233
## hip          -0.07569097
## thigh        -0.02074726
## knee          0.03687377
## ankle         0.04798041
## biceps        0.08261190
## forearm       0.15170094
## wrist        -1.22185867
```

```r
# Predict and calculate RMSE for Ridge Regression on the test set
ridge_pred <- predict(ridge_model_fat, X_test, s = ridge_cv_fat$lambda.min)
ridge_rmse <- sqrt(mean((y_test - ridge_pred)^2))
paste("RMSE for Ridge Regression on Test Set:", ridge_rmse)
```

```
## [1] "RMSE for Ridge Regression on Test Set: 3.86956312259966"
```

Results for Ridge Regression: - Coefficients: Ridge regression retained all predictors with shrunk coefficients. - RMSE: 3.86956312259966.

All predictors have non-zero coefficients because Ridge regression shrinks the coefficients towards zero but doesn't set them to zero. Ridge retains all predictors with reduced magnitudes, useful when we believe most predictors contribute to the response.

```r
# LASSO Regression with Cross-Validation
lasso_cv_fat <- cv.glmnet(X_train, y_train, alpha = 1)  # alpha = 1 for LASSO
lasso_model_fat <- glmnet(X_train, y_train, alpha = 1, lambda = lasso_cv_fat$lambda.min)

# LASSO Regression Coefficients
print("LASSO Regression Coefficients:")
```

```
## [1] "LASSO Regression Coefficients:"
```

```r
print(coef(lasso_model_fat))
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  1.398040861
## neck        -0.600860845
## chest       -0.030657704
## abdom        0.882113568
## hip         -0.248854629
## thigh        .
## knee         .
## ankle        0.025843845
## biceps       0.009725825
## forearm      0.213693733
## wrist       -1.117521657
```

```r
# Predict and calculate RMSE for LASSO Regression on the test set
lasso_pred <- predict(lasso_model_fat, X_test, s = lasso_cv_fat$lambda.min)
lasso_rmse <- sqrt(mean((y_test - lasso_pred)^2))
paste("RMSE for LASSO Regression on Test Set:", lasso_rmse)
```

```
## [1] "RMSE for LASSO Regression on Test Set: 3.80129174946318"
```

Results for LASSO Regression: - Coefficients: LASSO set thigh and knee coefficients to zero. - RMSE: 3.80129174946318 (LASSO shows a better fit than Ridge in this case. The model performs well.)

LASSO eliminates less important predictors, performing both variable selection and regularization, which is helpful when you suspect that only a few predictors are truly important.