Statistical Method Illustration - PCA

Daniel Shang

```r
# Load the packages necessary for the project
library(dplyr)
```

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

## The following objects are masked from 'package:stats':
##
##     filter, lag

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

```r
library(psych)
```

```
## Warning: package 'psych' was built under R version 4.0.3
```

```r
library(fansi)
```

```
## Warning: package 'fansi' was built under R version 4.0.3
```

```r
library(devtools)
```

```
## Warning: package 'devtools' was built under R version 4.0.3

## Loading required package: usethis

## Warning: package 'usethis' was built under R version 4.0.3
```

```r
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.0.3
```

```
##
## Attaching package: 'ggplot2'

## The following objects are masked from 'package:psych':
##
##     %+%, alpha
```

```
# Import the data and show a few grow to get a quick glance at the data
data = read.csv('heart.csv')
head(data)
```

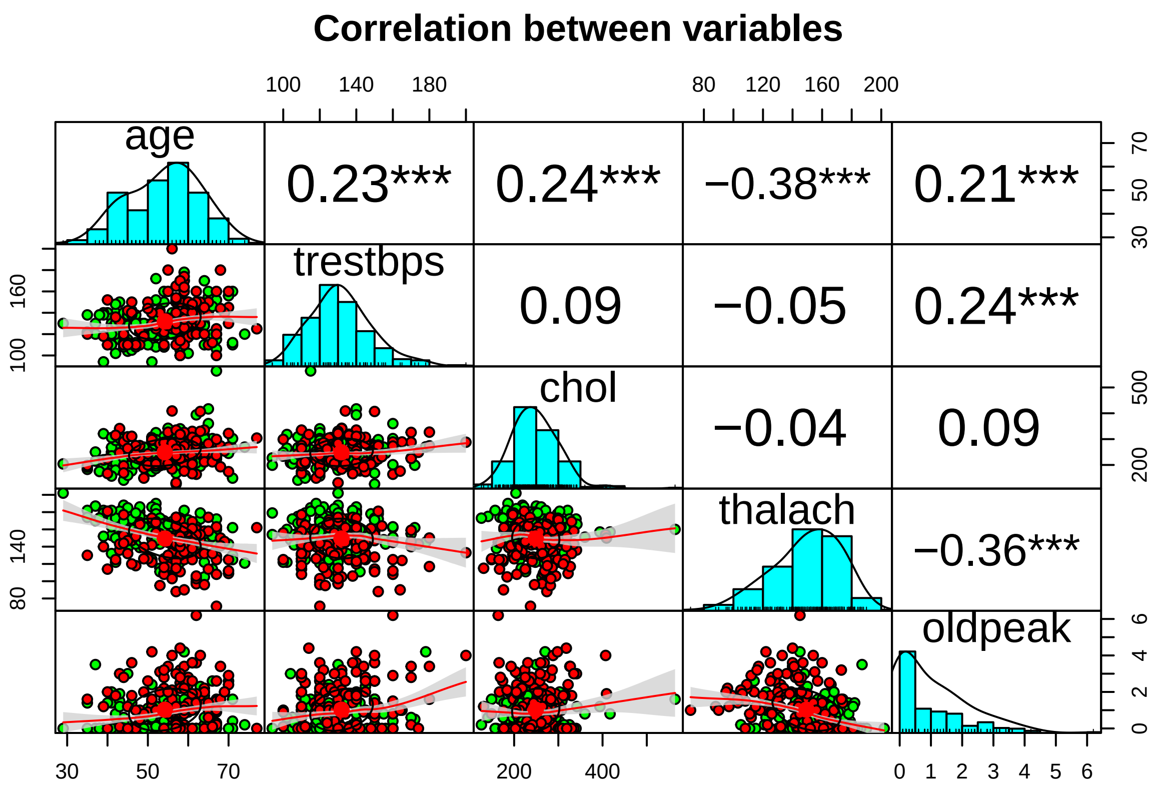
```
##   ï..age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 1     63   1  3      145  233   1       0     150     0     2.3     0  0    1
## 2     37   1  2      130  250   0       1     187     0     3.5     0  0    2
## 3     41   0  1      130  204   0       0     172     0     1.4     2  0    2
## 4     56   1  1      120  236   0       1     178     0     0.8     2  0    2
## 5     57   0  0      120  354   0       1     163     1     0.6     2  0    2
## 6     57   1  0      140  192   0       1     148     0     0.4     1  0    1
##   target
## 1      1
## 2      1
## 3      1
## 4      1
## 5      1
## 6      1
```

```
# Clean the data as necessary. Since PCA only works with numerical variables, we
## remove all the categorical/dummy variables. Doing this would lose us information,
## making the model's performance not as good as others that consider all the variabels.
## However, this example is for illustration purpose. Thus, I will go with a dataset
## that does not include all the data.
colnames(data)[1] = 'age'
data_clean = na.omit(mutate_all(data, ~ifelse(. %in% c('N/A', 'null', 'Null', 'NULL',
                                                        '', NULL), NA, .)))
to_drop = c('sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal')
data_clean = data_clean[, !(colnames(data_clean) %in% to_drop)]
data_clean$target = as.factor(data_clean$target)
head(data_clean)
```

```
##   age trestbps chol thalach oldpeak target
## 1  63      145  233     150     2.3      1
## 2  37      130  250     187     3.5      1
## 3  41      130  204     172     1.4      1
## 4  56      120  236     178     0.8      1
## 5  57      120  354     163     0.6      1
## 6  57      140  192     148     0.4      1
```

```
# Split the dataset into train and test for further model performance evaluation
set.seed(123)
index_1 = sample(2, nrow(data_clean), replace = TRUE, prob = c(0.8, 0.2))
train = data_clean[index_1 == 1, ]
test = data_clean[index_1 == 2, ]
```

```
# Build a correlation matrix to show the relationship between each pair of the
## variables, except for target (the dependent variable). Sometimes, two independent
## variables are highly correlated. When that is the case, the colinearity problem
## could happen. PCA does a good job handling that. Not only that, PCA prioritize
## the most important variables when there are hundreds of independent variable.

## This correlation matrix shows that no two variables are highly correlated to
## each other, indicating a remote possibility of colinearity problem.
pairs.panels(x = train[, -6], gap = 0, bg = c('red', 'green')[train$target],
             pch = 21, stars = TRUE, ci = TRUE, alpha = 0.1,
             main = 'Correlation between variables')
```



```
# Build a PCA model, calculate the average and standard deviation of each variables,
## and show them.
pca1 = prcomp(train[, -6], center = TRUE, scale. = TRUE)
pca1$center
```

```
##        age   trestbps       chol    thalach    oldpeak
##  54.202429 131.910931 248.906883 149.421053   1.027935
```

```
pca1$scale
```

```
##       age  trestbps      chol   thalach   oldpeak
##  8.897309 17.173675 53.570441 23.070254  1.156314
```

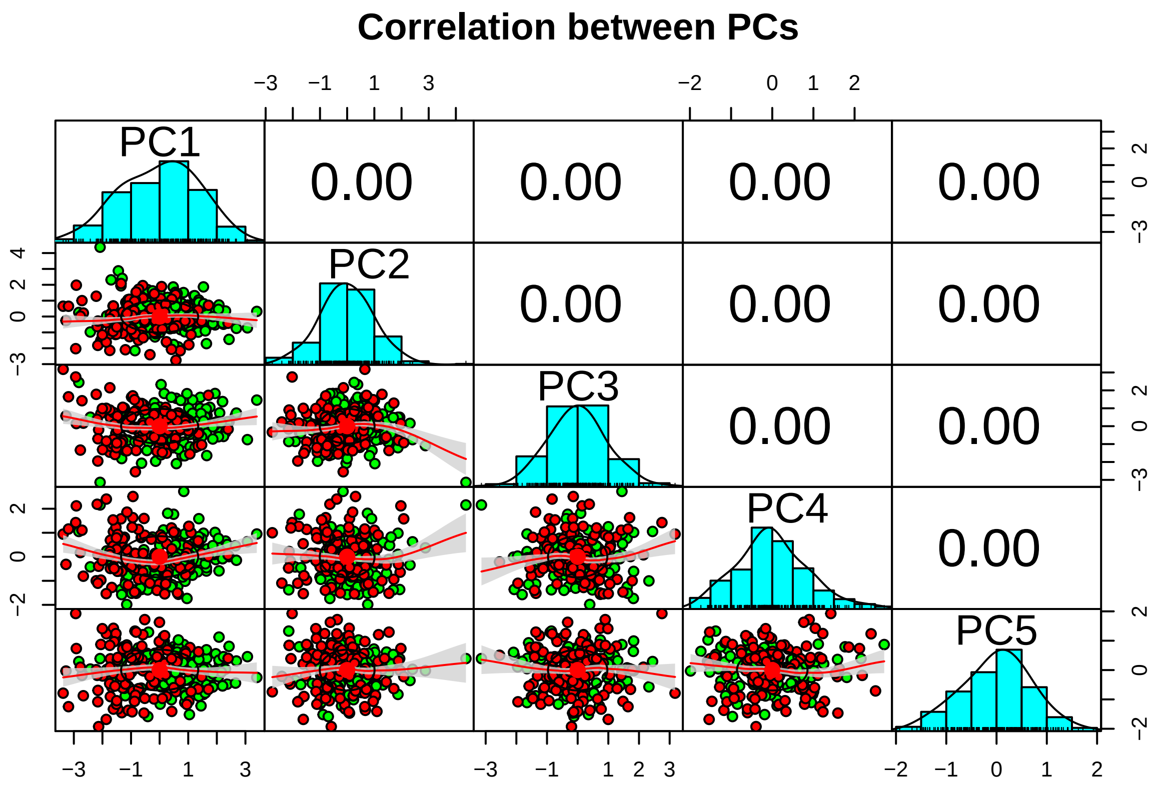
```r
# Print the PCA model to see the detail. The value in the matrix below is called
## loading score. Loading score is the coordinates for the unit vector. Practically,
## since age has the greatest loading score in this case, it is responsible for the
## most variation along PC1.
print(pca1)
```

```
## Standard deviations (1, .., p=5):
## [1] 1.3501271 1.0100964 0.9688530 0.8574901 0.6949077
##
## Rotation (n x k) = (5 x 5):
##                 PC1        PC2        PC3         PC4        PC5
## age      -0.5436206  0.1392706 -0.2360410 -0.58134930  0.5398129
## trestbps -0.3611250  0.3046759  0.7840986 -0.22036132 -0.3367368
## chol     -0.2789604  0.7478540 -0.3919299  0.40918842 -0.2045761
## thalach   0.5014419  0.4847984  0.3370119  0.09146374  0.6257667
## oldpeak  -0.4947753 -0.3057143  0.2495758  0.66156866  0.4022127
```

```r
# Summarize the principle component model. The number shows that PC1 explains
## 36.72% of the variability. Also, PC2 explains 20.47% of the variability. Since
## each principle component has a fairly large of proportion, each of them
## contributes to the variability.
summary(pca1)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5
## Standard deviation     1.3501 1.0101 0.9689 0.8575 0.69491
## Proportion of Variance 0.3646 0.2041 0.1877 0.1471 0.09658
## Cumulative Proportion  0.3646 0.5686 0.7564 0.9034 1.00000
```

```r
# Then, we plot the principle component in a correlation matrix. Since each PC is
## orthogonal to the next PC, they are not correlated at all. This is how PC model
## helps handle multi-colinearity problem.
pairs.panels(pca1$x, gap = 0, bg = c('red', 'green')[train$target], stars = TRUE,
             main = 'Correlation between PCs', pch = 21, ci = TRUE, alpha = 0.1)
```



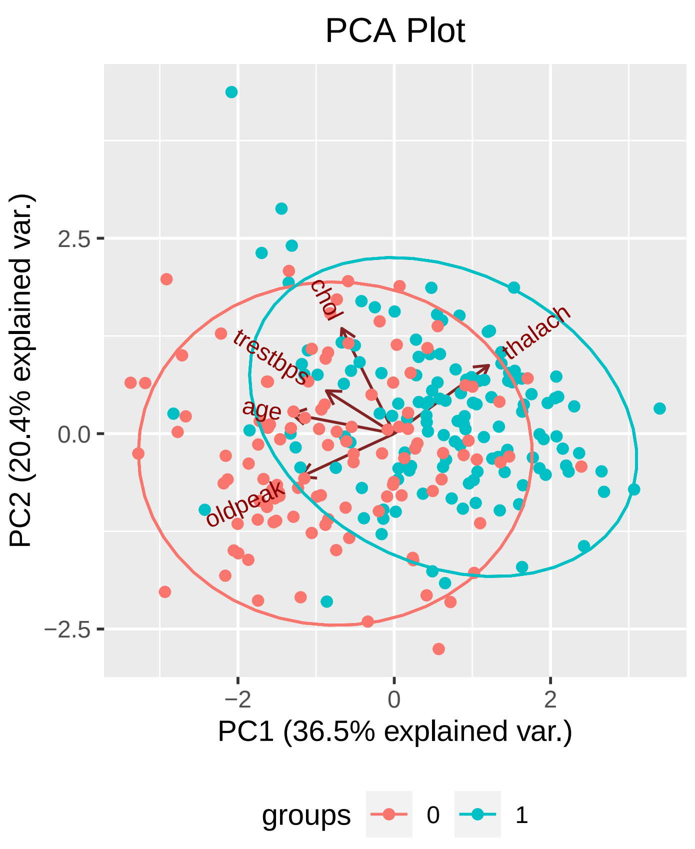
```
# This chunk is to define the 'ggbiplot' function. The function and resources can be
#found at https://github.com/vqv/ggbiplot | Copyright 2011 Vincent Q. Vu.

ggbiplot <- function(pcobj, choices = 1:2, scale = 1, pc.biplot = TRUE,
                     obs.scale = 1 - scale, var.scale = scale,
                     groups = NULL, ellipse = FALSE, ellipse.prob = 0.68,
                     labels = NULL, labels.size = 3, alpha = 1,
                     var.axes = TRUE,
                     circle = FALSE, circle.prob = 0.69,
                     varname.size = 3, varname.adjust = 1.5,
                     varname.abbrev = FALSE, ...)
{
  library(ggplot2)
  library(plyr)
  library(scales)
  library(grid)

  stopifnot(length(choices) == 2)

 if(inherits(pcobj, 'prcomp')){
    nobs.factor <- sqrt(nrow(pcobj$x) - 1)
    d <- pcobj$sdev
    u <- sweep(pcobj$x, 2, 1 / (d * nobs.factor), FUN = '*')
    v <- pcobj$rotation
  } else if(inherits(pcobj, 'princomp')) {
    nobs.factor <- sqrt(pcobj$n.obs)
    d <- pcobj$sdev
    u <- sweep(pcobj$scores, 2, 1 / (d * nobs.factor), FUN = '*')
    v <- pcobj$loadings
  } else if(inherits(pcobj, 'PCA')) {
    nobs.factor <- sqrt(nrow(pcobj$call$X))
    d <- unlist(sqrt(pcobj$eig)[1])
    u <- sweep(pcobj$ind$coord, 2, 1 / (d * nobs.factor), FUN = '*')
    v <- sweep(pcobj$var$coord,2,sqrt(pcobj$eig[1:ncol(pcobj$var$coord),1]),FUN="/")
  } else if(inherits(pcobj, "lda")) {
      nobs.factor <- sqrt(pcobj$N)
      d <- pcobj$svd
      u <- predict(pcobj)$x/nobs.factor
      v <- pcobj$scaling
      d.total <- sum(d^2)
  } else {
    stop('Expected a object of class prcomp, princomp, PCA, or lda')
  }

  choices <- pmin(choices, ncol(u))
  df.u <- as.data.frame(sweep(u[,choices], 2, d[choices]^obs.scale, FUN='*'))

  v <- sweep(v, 2, d^var.scale, FUN='*')
  df.v <- as.data.frame(v[, choices])

  names(df.u) <- c('xvar', 'yvar')
  names(df.v) <- names(df.u)

  if(pc.biplot) {
    df.u <- df.u * nobs.factor
  }

  r <- sqrt(qchisq(circle.prob, df = 2)) * prod(colMeans(df.u^2))^(1/4)

  v.scale <- rowSums(v^2)
  df.v <- r * df.v / sqrt(max(v.scale))

  if(obs.scale == 0) {
    u.axis.labs <- paste('standardized PC', choices, sep='')
  } else {
    u.axis.labs <- paste('PC', choices, sep='')
  }

  u.axis.labs <- paste(u.axis.labs,
                       sprintf('(%0.1f%% explained var.)',
                               100 * pcobj$sdev[choices]^2/sum(pcobj$sdev^2)))

  if(!is.null(labels)) {
    df.u$labels <- labels
  }

  if(!is.null(groups)) {
    df.u$groups <- groups
  }
```

```r
if(varname.abbrev) {
  df.v$varname <- abbreviate(rownames(v))
} else {
  df.v$varname <- rownames(v)
}

df.v$angle <- with(df.v, (180/pi) * atan(yvar / xvar))
df.v$hjust = with(df.v, (1 - varname.adjust * sign(xvar)) / 2)

g <- ggplot(data = df.u, aes(x = xvar, y = yvar)) +
        xlab(u.axis.labs[1]) + ylab(u.axis.labs[2]) + coord_equal()

if(var.axes) {

  if(circle)
  {
    theta <- c(seq(-pi, pi, length = 50), seq(pi, -pi, length = 50))
    circle <- data.frame(xvar = r * cos(theta), yvar = r * sin(theta))
    g <- g + geom_path(data = circle, color = muted('white'),
                       size = 1/2, alpha = 1/3)
  }

  g <- g +
    geom_segment(data = df.v,
                 aes(x = 0, y = 0, xend = xvar, yend = yvar),
                 arrow = arrow(length = unit(1/2, 'picas')),
                 color = muted('red'))
}

if(!is.null(df.u$labels)) {
  if(!is.null(df.u$groups)) {
    g <- g + geom_text(aes(label = labels, color = groups),
                       size = labels.size)
  } else {
    g <- g + geom_text(aes(label = labels), size = labels.size)
  }
} else {
  if(!is.null(df.u$groups)) {
    g <- g + geom_point(aes(color = groups), alpha = alpha)
  } else {
    g <- g + geom_point(alpha = alpha)
  }
}

if(!is.null(df.u$groups) && ellipse) {
  theta <- c(seq(-pi, pi, length = 50), seq(pi, -pi, length = 50))
  circle <- cbind(cos(theta), sin(theta))

  ell <- ddply(df.u, 'groups', function(x) {
    if(nrow(x) <= 2) {
      return(NULL)
    }
    sigma <- var(cbind(x$xvar, x$yvar))
```

```
      mu <- c(mean(x$xvar), mean(x$yvar))
      ed <- sqrt(qchisq(ellipse.prob, df = 2))
      data.frame(sweep(circle %*% chol(sigma) * ed, 2, mu, FUN = '+'),
                 groups = x$groups[1])
    })
    names(ell)[1:2] <- c('xvar', 'yvar')
    g <- g + geom_path(data = ell, aes(color = groups, group = groups))
  }

  if(var.axes) {
    g <- g +
    geom_text(data = df.v,
              aes(label = varname, x = xvar, y = yvar,
                  angle = angle, hjust = hjust),
              color = 'darkred', size = varname.size)
  }
  return(g)
}
```

```
# Here, we plot the PCA model using the 'ggbiplot' package that is available from
## GitHub. The five arrows in the plow indicate both the sign and the magnitude of
## the loading score in the previous model detail.
ggbiplot(pca1, obs.scale = 1, groups = train$target, ellipse = TRUE,
         ellipse.prob = 0.9) +
  labs(title = 'PCA Plot') +
  theme(legend.direction = 'horizontal', plot.title = element_text(hjust = 0.5),
        legend.position = 'bottom')
```

```
## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##
## Attaching package: 'plyr'

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

##
## Attaching package: 'scales'

## The following objects are masked from 'package:psych':
##
##     alpha, rescale
```



```
# We predict the loading scores using the train and test sets.
train_1 = predict(pca1, train)
train_1 = data.frame(train_1, train[6])
test_1 = predict(pca1, test)
test_1 = data.frame(test_1, test[6])
```

```
# Here, we fit a logistic regression model using loading scores as variables.
logistic_1 = glm(target~., data = train_1, family = 'binomial')
summary(logistic_1)
```

```
##
## Call:
## glm(formula = target ~ ., family = "binomial", data = train_1)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2672  -0.7459   0.4127   0.7934   2.3689
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.20172    0.15801   1.277 0.201733
## PC1          1.05283    0.14833   7.098 1.27e-12 ***
## PC2          0.59962    0.15461   3.878 0.000105 ***
## PC3          0.08346    0.16676   0.500 0.616735
## PC4         -0.34423    0.18318  -1.879 0.060224 .
## PC5          0.15154    0.22824   0.664 0.506729
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 340.27  on 246  degrees of freedom
## Residual deviance: 246.58  on 241  degrees of freedom
## AIC: 258.58
##
## Number of Fisher Scoring iterations: 4
```

```
# Here, we set the threshold of categorizing the target as 0.5 and make the prediction
## using the logistic regression model. Unsurprisingly, the predictive power is
## low. This is mainly because, for the illustration purpose, I removed those
## categorical variables. Those fields could contained critical information to
## make the prediction. But once again, the purpose of this project is to illustrate
## the code to build a PCA model, not to make a prediction as accurate as possible.

pred = predict(logistic_1, newdata = test_1)

pred_response = c()
for (i in 1:length(pred)) {
  if (pred[i] > 0.5) {
    pred_response = c(pred_response, 1)
  } else {
    pred_response = c(pred_response, 0)
  }
}

pred_response = as.factor(pred_response)
confusionMatrix(pred_response, test_1$target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 18 15
##          1  8 15
##
##                Accuracy : 0.5893
##                  95% CI : (0.4498, 0.719)
##     No Information Rate : 0.5357
##     P-Value [Acc > NIR] : 0.2522
##
##                   Kappa : 0.1889
##
##  Mcnemar's Test P-Value : 0.2109
##
##             Sensitivity : 0.6923
##             Specificity : 0.5000
##          Pos Pred Value : 0.5455
##          Neg Pred Value : 0.6522
##              Prevalence : 0.4643
##          Detection Rate : 0.3214
##    Detection Prevalence : 0.5893
##       Balanced Accuracy : 0.5962
##
##        'Positive' Class : 0
##
```