Normal Distribution Sampling

Tarek El-Hajjaoui

2023-04-13

```
library(tidyr) #the pipe (%>%) tool is extremely useful
library(MASS) # used for mvrnorm
```

Q1

Suppose Suppose X_1, X_2, Y_1, Y_2 are mutually independent.

- X_1 and X_2 are iid from $N(\mu = 0, \sigma^2 = 2^2)$
- Y_1 and Y_2 are iid from $N(\mu = 0, \sigma^2 = 1^2)$

a) Calculate $P(|X_1 - X_2| > 4)$

- Let $X = |X_1 - X_2|$ then $P(|X| > 4) \sim N(0, 8)$
- Transform X -> Z_X then $P(|Z_X| > 4/\sqrt{8}) \sim N(0, 1)$
- Calculation in R below:

```
2 * (1 - pnorm(4/sqrt(8)))
```

```
## [1] 0.1572992
```

b) Calculate $P(|Y_1 - Y_2| > 4)$

- Let $Y = Y_1 - Y_2$ then $P(|Y| > 4) \sim N(0, 2)$
- Transform Y -> Z_y then $P(|Z_Y| > 4/\sqrt{2}) \sim N(0, 1)$
- Calculation of in R below:

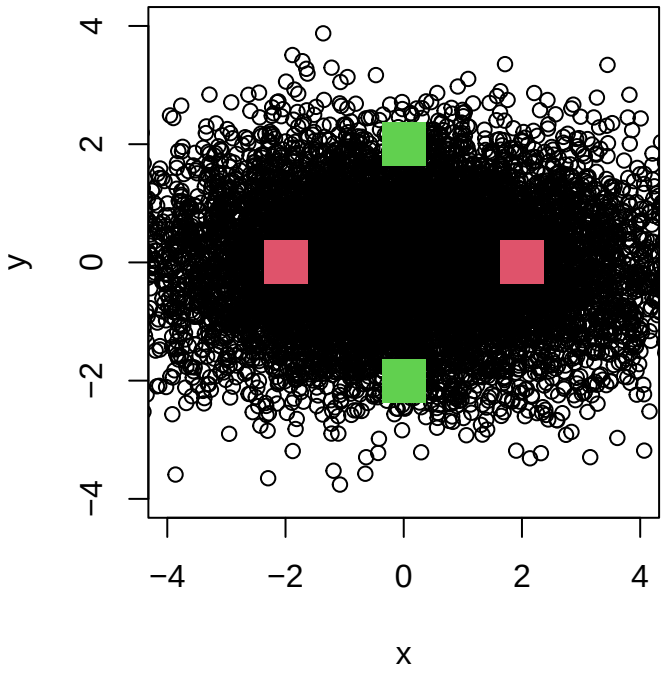
```
2 * (1 - pnorm(4/sqrt(2)))
```

```
## [1] 0.004677735
```

Q1 continued

c) Estimate the two probabilities using simulations. The code in the previous page generates 1000 random samples. Change the sample size from n=1000 to n=10000 and then estimate the two probabilities. To do that, you need to examine all pairs of data points and then calculate the proportion of pairs satisfying a certain condition.

```
set.seed(20230404)
n <- 10000 # number of samples
# bivariate normal random sample parameters
bivariate_mu <- c(0,0)
cov_matrix <- matrix(c(4,0,0,1),2,2)
# bivariate normal sample
sample <- mvrnorm(n=n, mu=bivariate_mu, Sigma=cov_matrix)
# Extract the X values, X ~ N(0, 4)
x <- sample[, 1]
# Extract the Y values, X ~ N(0, 1)
y <- sample[, 2]
# plot
par(pty="s") #to make sure the shape of figure is a square
sample %>%
  plot(xlab="x", ylab="y", xlim=c(-4,4), ylim=c(-4,4))
points(x=c(-2, 0, 0, 2), y=c(0, -2, 2, 0), pch=15, col=c(2,3,3,2),cex=3)
```



Calculate $P(|X| > 2)$ and calculate $P(|Y| > 2)$

```
# Calculate the proportion of pairs satisfying |X| > 2
mean(abs(x) > 2)
```

```
## [1] 0.3196
```

```
# Calculate the proportion of pairs satisfying |Y| > 2
mean(abs(y) > 2)
```

```
## [1] 0.05
```

Q2

a) Find a matrix A such that AY gives the difference of mean vectors between iris setosa and iris versicolor

```
# Extracting the continuous feature columns from the dataset, 1 - 4.
# and select only setosa and versicolor species
iris_sub <- subset(iris, Species %in% c("setosa", "versicolor"))

# Compute the mean vector for setosa and versicolor
mean_setosa <- colMeans(subset(iris_sub, Species == "setosa")[, 1:4])
mean_versicolor <- colMeans(subset(iris_sub, Species == "versicolor")[, 1:4])

# Compute the difference of mean vectors
diff_mean <- mean_setosa - mean_versicolor
diff_mean
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##       -0.930        0.658       -2.798       -1.080
```

```
# Create a matrix A such that AY = diff_mean
A <- diag(4) # Identity matrix with dimension 4x4
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

```
dim(A)
```

```
## [1] 4 4
```

```
# Put Y into a matrix
Y <- as.matrix(rbind(mean_setosa, mean_versicolor))
Y
```

```
##                 Sepal.Length Sepal.Width Petal.Length Petal.Width
## mean_setosa            5.006       3.428        1.462       0.246
## mean_versicolor        5.936       2.770        4.260       1.326
```

```
dim(Y)
```

```
## [1] 2 4
```

```
AY <- A %*% t(Y)
AY
```

```
##      mean_setosa mean_versicolor
## [1,]       5.006           5.936
## [2,]       3.428           2.770
## [3,]       1.462           4.260
## [4,]       0.246           1.326
```

b) Find a matrix B such that YB is column-standardized, i.e., the standard deviation of each column/feature is 1.

```
# Apply sd() function to the columns (2nd dim)
Z_y <- apply(Y, 2, sd)
Z_y
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.6576093    0.4652763    1.9784848    0.7636753
```

```
# Matrix B such that YB is column-standardized
B <- diag(2) / Z_y
B
```

```
##          [,1]     [,2]
## [1,] 1.52066 0.000000
## [2,] 0.00000 1.309457
```

```
dim(Y)
```

```
## [1] 2 4
```

```
dim(B)
```

```
## [1] 2 2
```

```
# Compute YB
YB <- t(Y) %*% B

YB
```

```
##                   [,1]     [,2]
## Sepal.Length 7.6124227 7.772937
## Sepal.Width  5.2128216 3.627196
## Petal.Length 2.2232045 5.578287
## Petal.Width  0.3740823 1.736340
```

c) Check the following

- Let $C = \mathbf{I}_{150} - \frac{1}{150}J$, where $J_{150\times 150}$ is an all-ones matrices . Use R to verify that CY centers each column/feature.

```
C=diag(1,150) - (1/150) * matrix(1, 150, 150)

# define Y to be all of Iris dataset
Y=as.matrix(iris[, 1:4])

dim(C)
```

```
## [1] 150 150
```

```
dim(Y)
```

```
## [1] 150   4
```

```
CY <- C %*% Y

dim(CY)
```

```
## [1] 150   4
```

```
# Compute column means of CY
col_means_CY <- colMeans(CY)

# Verify that column means of CY are all approximately zero
all.equal(col_means_CY, rep(0, 4), check.names = FALSE)
```

```
## [1] TRUE
```

- Let S be the sample covariance matrix. Use R to verify that each column of $CYS^{-1/2}$ has been centered and standardized (in fact, the columns have also been de-correlated).

```
# Load the qtl2pleio package
#library(qtl2pleio)

Y <- as.matrix(iris[, 1:4])

# Compute the sample covariance matrix S
S <- cov(Y)

# Using eigen decomposition of the covariance matrix S
eig <- eigen(S)
D <- diag(1/sqrt(eig$values))
# Compute S^(-1/2)
S_inv_sqrt <- eig$vectors %*% D %*% t(eig$vectors)

#S_inv_sqrt <- calc_invsqrt_mat(S)
CYS_inv_sqrt <- C %*% Y %*% S_inv_sqrt

colMeans(CYS_inv_sqrt)
```

```
## [1]  2.691447e-15 -4.232170e-15 -9.582844e-16 -1.500466e-15
```

```
# Verify that column means of CY are all approximately zero
all.equal(colMeans(CYS_inv_sqrt), rep(0, 4), check.names = FALSE)
```

```
## [1] TRUE
```

```
apply(CYS_inv_sqrt, 2, sd)
```

```
## [1] 1 1 1 1
```

```
# Verify that column means of CY are all approximately zero
all.equal(apply(CYS_inv_sqrt, 2, sd), rep(1, 4), check.names = FALSE)
```

```
## [1] TRUE
```

Q3

Choose a picture you like and conduct approximations using singular value decomposition (SVD).

```
library(jpeg)
img=readJPEG("./lion.jpg", native = FALSE)
img=img[,,1]
dim(img)
```

```
## [1] 524 800
```

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

image(img, col=gray(c(0:10)/10))
obj.bm=svd(img)
n=10
img.approx=obj.bm$u[,1:n]%*% diag(obj.bm$d[1:n]) %*% t(obj.bm$v[,1:n])
image(img.approx, main="rank 10", col=gray(c(0:10)/10))
n=20
img.approx=obj.bm$u[,1:n]%*% diag(obj.bm$d[1:n]) %*% t(obj.bm$v[,1:n])
image(img.approx, main="rank 20", col=gray(c(0:10)/10))
n=30
img.approx=obj.bm$u[,1:n]%*% diag(obj.bm$d[1:n]) %*% t(obj.bm$v[,1:n])
image(img.approx, main="rank 30", col=gray(c(0:10)/10))
```