HW: Week 9

36-350 – Statistical Computing

Week 9 – Fall 2020

Name: Kimberly Zhang

Andrew ID: kyz

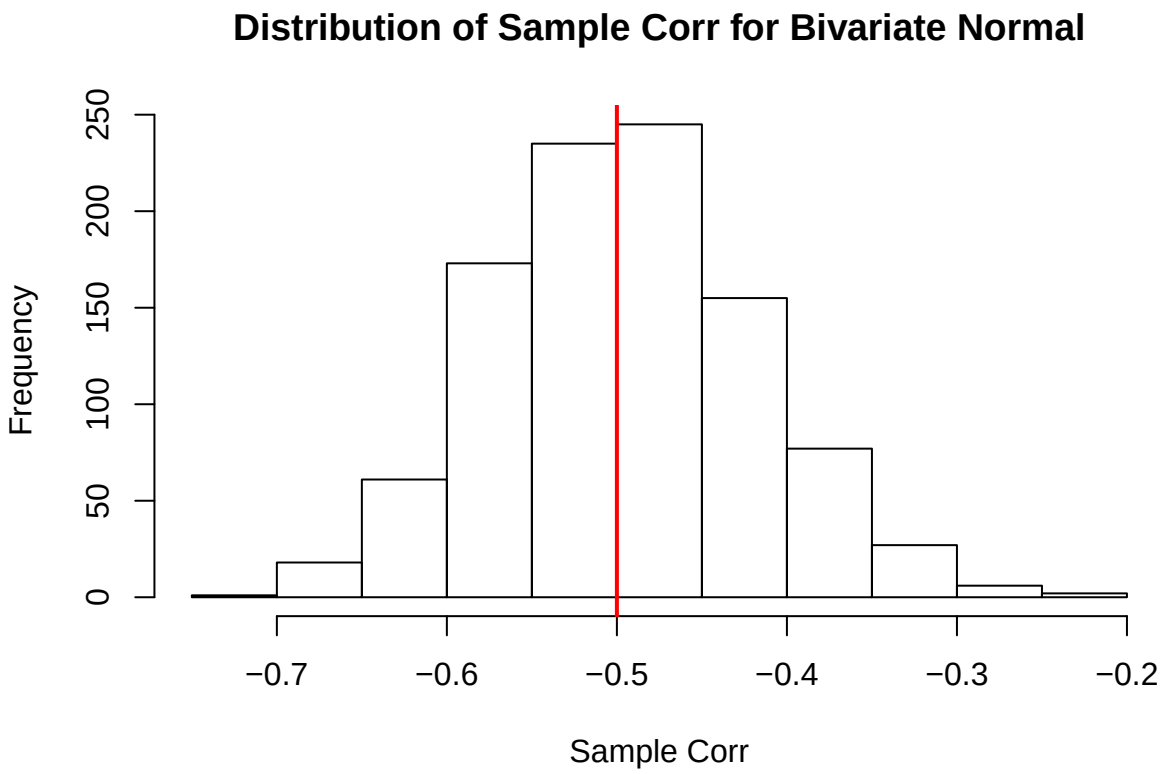
You must submit **your own** lab as a PDF file on Gradescope.

Question 1

(10 points)

Display the sampling distribution for $R_{1,2}$, the off-diagonal element of a two-dimensional sample correlation matrix for a bivariate normal, given that $\mu_1 = \mu_2 = 2$, $\sigma_1 = 1$, $\sigma_2 = 2$, $\rho_{1,2} = -0.5$, and $n = 100$. Sample 1000 values and display them in a histogram; include a vertical line for the population value (-0.5). (Reminder: if I don't specify how exactly to visualize something, i.e., if I don't specify base `R` versus `ggplot`, then you can choose whichever.) Note that the final distribution that you see will not be normal.

```
suppressMessages(library(MASS))
mu = c(2,2)
sigma.1 = 1
sigma.2 = 2
rho.12  = -0.5
Sigma = matrix(c(sigma.1^2,rho.12*sigma.1*sigma.2,rho.12*sigma.1*sigma.2,
                 sigma.2^2),nrow=2)
n = 100
res = rep(NA, 1000)
for (i in 1:1000){
 data = mvrnorm(n,mu,Sigma)
 res[i] = cor(data[,1], data[,2])
}
hist(res, main= "Distribution of Sample Corr for Bivariate Normal",
     xlab = "Sample Corr")
abline(v=-0.5, col = "red", lwd= 2)
```

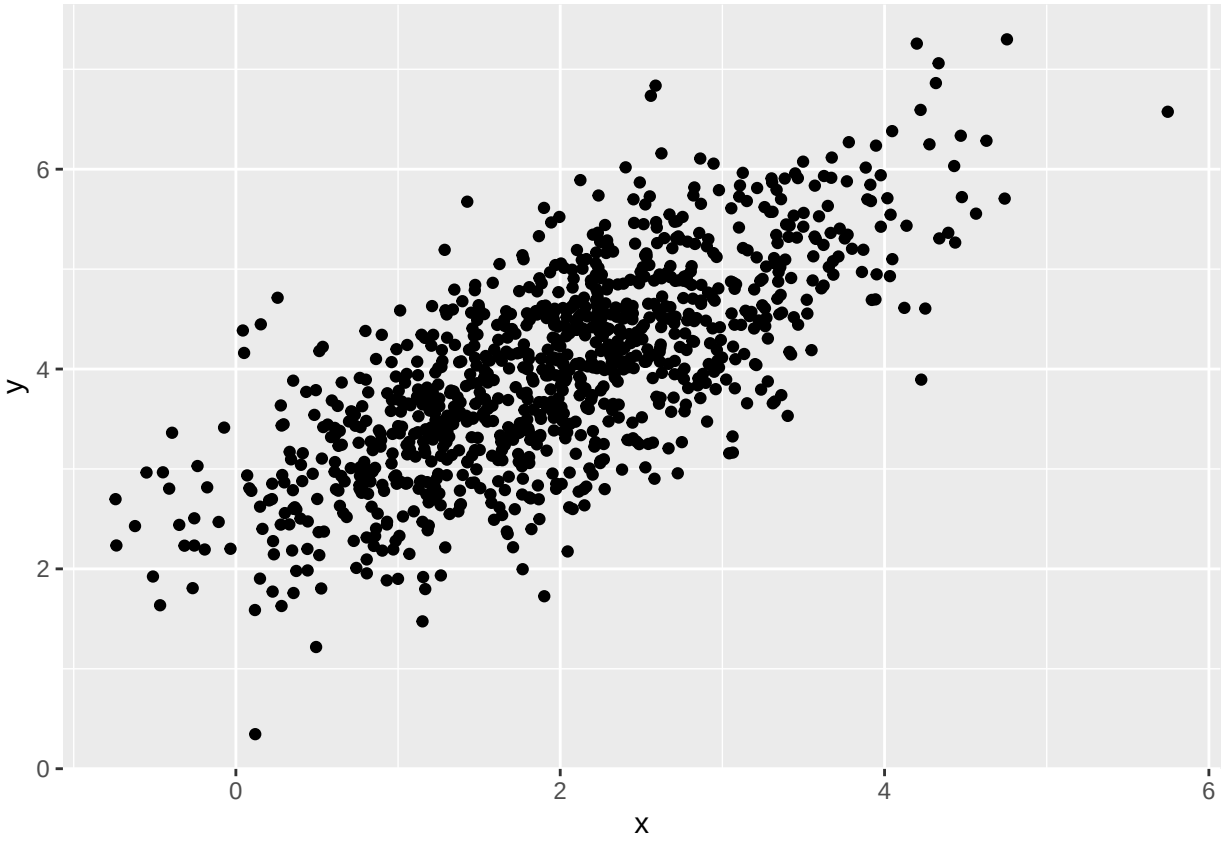


Question 2

(10 points)

Assume you have a trivariate multivariate normal with $\mu = \{2, 3, 4\}$ and $\sigma = \{1, 2, 1\}$, and $\rho_{1,2} = 0.4$, $\rho_{1,3} = 0.7$, and $\rho_{2,3} = -0.2$. Sample 1000 data from the marginal distribution for (x_1, x_3), and display them via `ggplot`. Compute the sample means along each marginal axis: they should be approximately 2 and 4.

```
suppressMessages(library(tidyverse))
mu = c(2,3,4)
sigma.1 = 1
sigma.2 = 2
sigma.3 = 1
rho.12  = 0.4
rho.13  = 0.7
rho.23  = -0.2
Sigma = matrix(c(sigma.1^2,rho.12*sigma.1*sigma.2,rho.13*sigma.1*sigma.3,
                 rho.12*sigma.2*sigma.1, sigma.2^2, rho.23*sigma.2*sigma.3,
                 rho.13*sigma.3*sigma.1, rho.23*sigma.3*sigma.2, sigma.3^3),
               nrow=3)
p= c(1,3)
mu.marginal = mu[p]
Sigma.marginal= Sigma[p, p]
data = mvrnorm(1000,mu.marginal,Sigma.marginal)
df = data.frame(x=data[,1],y=data[,2])
ggplot(data=df,mapping=aes(x=x,y=y)) + geom_point()
```




```
colMeans(df)
```

```
##        x        y
## 1.969600 3.988518
```

Question 3

(15 points)

Repeat Q2, except here you should sample from the conditional distribution $f(x_1, x_3|x_2 = 1)$. Compute the conditional correlation coefficient ρ_{cond} between the data along the x_1 and x_3 axes, given $x_2 = 1$, and display the sample correlation matrix.

```
k = c(1,3)
d.minus.k = c(2)
x.2 = 1
Sigma.kk = Sigma[k,k]
Sigma.kd = Sigma[k,d.minus.k]
Sigma.dk = Sigma[d.minus.k,k]
Sigma.dd = Sigma[d.minus.k,d.minus.k]
mu.cond = mu[k] +
  Sigma.kd %*% solve(Sigma.dd) %*% matrix(c(x.2)-mu[d.minus.k],nrow=1)
Sigma.cond = Sigma.kk - Sigma.kd %*% solve(Sigma.dd) %*% Sigma.dk
data = mvrnorm(1000,mu.cond,Sigma.cond)
df = data.frame(x=data[,1],y=data[,2])
cor(df)
```

```
##           x         y
```

```
## x 1.0000000 0.8784434
## y 0.8784434 1.0000000
```

Question 4

(15 points)

Assume that you have a mixture model: you have 100 data sampled from a bivariate normal with $\mu = \{1, 1\}$ and $\sigma = \{1.2, 1.2\}$, with $\rho = 0.4$, and another 100 data sampled from a bivariate normal with $\mu = \{3, 3\}$, $\sigma = \{1, 1\}$, and $\rho = -0.6$. Plot your sampled data with separate colors for each component of the mixture. Then perform logistic regression to try to classify each sampled point as being from component 1 or component 2, and output the proportion of times you misclassify a point. (Don't worry about breaking your data up into training and testing sets, as this is a simple academic exercise; just use all 200 points to train your classifier, then output the training misclassification error.)

How to train your logistic classifier and get the misclassification rate?

- Assuming you already have a data frame with sampled x_1 and x_2 values in the first and second columns, add a third column with the labeled class. (Name this column `class`.) Use 0 for the first class, and 1 for the second.
- Use `glm()` with model formula `class~.`, your data frame, and the argument `family=binomial`.
- Use `predict()` with the output of `glm()` and with the argument `type="response"`. This will generate 200 predictions between 0 and 1.
- Round off all predictions to 0 or 1.
- Create a `table()` with the arguments being your rounded-off predictions and the labeled classes.
- Compute the proportion of table elements that are "off-diagonal" (upper-right and lower-left). Done.

```
mu = c(1,1)
sigma.1 = 1.2
sigma.2 = 1.2
rho.12  = 0.4
Sigma = matrix(c(sigma.1^2,rho.12*sigma.1*sigma.2,rho.12*sigma.1*sigma.2,
                 sigma.2^2),nrow=2)
sample1= mvrnorm(100,mu,Sigma)
sample1 = cbind(sample1, rep(0, 100))
mu = c(3,3)
sigma.1 = 1
sigma.2 = 1
rho.12  = -0.6
Sigma = matrix(c(sigma.1^2,rho.12*sigma.1*sigma.2,rho.12*sigma.1*sigma.2,
                 sigma.2^2),nrow=2)
sample2= mvrnorm(100,mu,Sigma)
sample2 = cbind(sample2, rep(1, 100))
sample = rbind(sample1, sample2)
df = data.frame(x.1= sample[,1], x.2 = sample[,2], class = sample[,3])
model = glm(formula = class~., data = df, family = "binomial")
preds = predict(model, type= "response")
first.class = round(preds)[1:100]
second.class = round(preds)[101:200]
(sum(first.class !=0) + sum(second.class !=1))/200
```

```
## [1] 0.08
```

In the following code chunk, we input seven measurements for each of 5000 asteroids. The data frame is `df`. There is another variable, `q`, that is also loaded and which we will utilize later.

```
load(url("http://www.stat.cmu.edu/~pfreeman/HW_09_PCA.Rdata"))
names(df)
```

```
## [1] "diameter" "albedo"   "a"        "e"        "i"        "per_y"    "H"
```

Question 5

(10 points)

Perform PCA on the data frame `df`. Use the rule-of-thumb in the notes to determine the number of principal components (or PCs) to retain, and for those PCs, indicate the mapping from PC to original variables. (Also, display the proportion of variance explained by the PCs you retain.) Are any of the original variables "unimportant" within the context of the retained PCs?

```
p = prcomp(df, scale = TRUE)
p$rotation
```

```
##                  PC1         PC2          PC3         PC4         PC5
## diameter -0.10427784  0.64018993 -0.381807654 -0.06374223  0.08430670
## albedo    0.02926049  0.27356501  0.891476176 -0.02430516 -0.21790935
## a        -0.61875917 -0.08069235  0.091831781 -0.14969568  0.25344191
## e        -0.30222711  0.04700618 -0.006640299  0.94592379 -0.10576940
## i        -0.37669959 -0.08308185 -0.183319322 -0.21654799 -0.87665888
## per_y    -0.60414428 -0.08470000  0.131810599 -0.16680797  0.31781953
## H         0.08469313 -0.70179878 -0.005997209  0.05871658  0.02116336
##                   PC6          PC7
## diameter -0.648700557  0.039023087
## albedo   -0.285476999 -0.001974438
## a        -0.029727335 -0.717403265
## e        -0.006606603  0.020061597
## i        -0.003403910  0.046405205
## per_y     0.035335507  0.693093653
## H        -0.703919686  0.029514963
```

```
cumsum(p$sdev^2)/sum(p$sdev^2)
```

```
## [1] 0.3354716 0.5970100 0.7429421 0.8663518 0.9740229 0.9965365 1.0000000
```

```
The first 5 PCs should be retained; together they explain up to 97.4% of the
variance.
PC1: dominated by a and per_y in the opposite direction (both around -0.6)
PC2: primarily diameter in same direction (0.64) and
H in opposite direction (-0.7)
PC3: mainly albedo in same direction (0.89)
PC4: primarily e in same direction (0.945)
PC5: mainly i in opposite direction (-0.876)

There doesn't seem to be any unimportant variables; all of them
are represented in the retained PCs.
```

Question 6

(10 points)

Something that one can do with principal components is regression: after all, all you've done is transform your data to a new coordinate system. Below, linearly regress the variable `q` upon all the variables in `df`, and

print the adjusted R^2 and the sum of squared errors for the model. Then repeat linear regression, except now regress the variable `q` upon only the retained PCs. Again, print out the adjusted R^2 and the sum of squared errors. Are the second value close to the first? (They often will be, but don't have to be.) (Hint: look at the names of the list elements that are output by the *summary* of your linear regression fits, as one of those list elements may help you with extracting the adjusted R^2. As far as the sum of squared errors, you need to simply compute the `sum()` of the difference between the observed values of `q` and the predicted values from `predict()`.)

```
lin.model = lm(q~., data = df)
summary(lin.model)$adj.r.squared
```

```
## [1] 0.7390329
```

```
sum((q - predict(lin.model, type= "response"))^2)
```

```
## [1] 2469.368
```

```
scores = p$x[, 1:5]
pc.model = lm(q~., data = as.data.frame(scores))
summary(pc.model)$adj.r.squared
```

```
## [1] 0.1842183
```

```
sum((q - predict(pc.model, type= "response"))^2)
```

```
## [1] 7722.323
```

```
The second value is not close to the first.
```

Question 7

(10 points)

See how well you can reconstruct the data using the first six PCs.

How to do this:

1. Note that the principal component scores are $z = X_s v$, where $X_s = (X - \bar{X})/s_X$ is the scaled data, and v is the PCA rotation matrix. That means that $zv^{-1} = X_s v v^{-1} = X_s$.
2. Also note that $\bar{X}$ and s_X are recorded in the output of `prcomp()`.
3. Also note that $v^{-1} = v^T$, i.e., the inverse of v is also its transpose.
4. Let z' be the first six columns of z. Then $X'_s = z'(v')^T$, where v' is the first six columns of v.

Call the reconstructed data X'. Finish off the problem by computing the mean of the reconstruction error in each column, i.e., compute the mean of the absolute value of $(X' - X)/X$ within each column of X. For what columns is the reconstruction error still large?

```
v.prime = p$rotation[,1:6]
z.prime = p$x[, 1:6]
x_s.prime = z.prime%*%t(v.prime)
scaled_df= scale(df)
d = dim(x_s.prime)
x.prime = x_s.prime*t(matrix(rep(p$scale, times= d[1]), nrow= d[2])) +
          t(matrix(rep(p$center, times= d[1]), nrow= d[2]))
colMeans(abs((x.prime - df)/df))
```

```
## diameter albedo a e i per_y
## 0.0046795464 0.0001655792 0.1039421553 0.0016866978 0.0037124089
0.9652642864
## H
```

```
## 0.0003016961
```

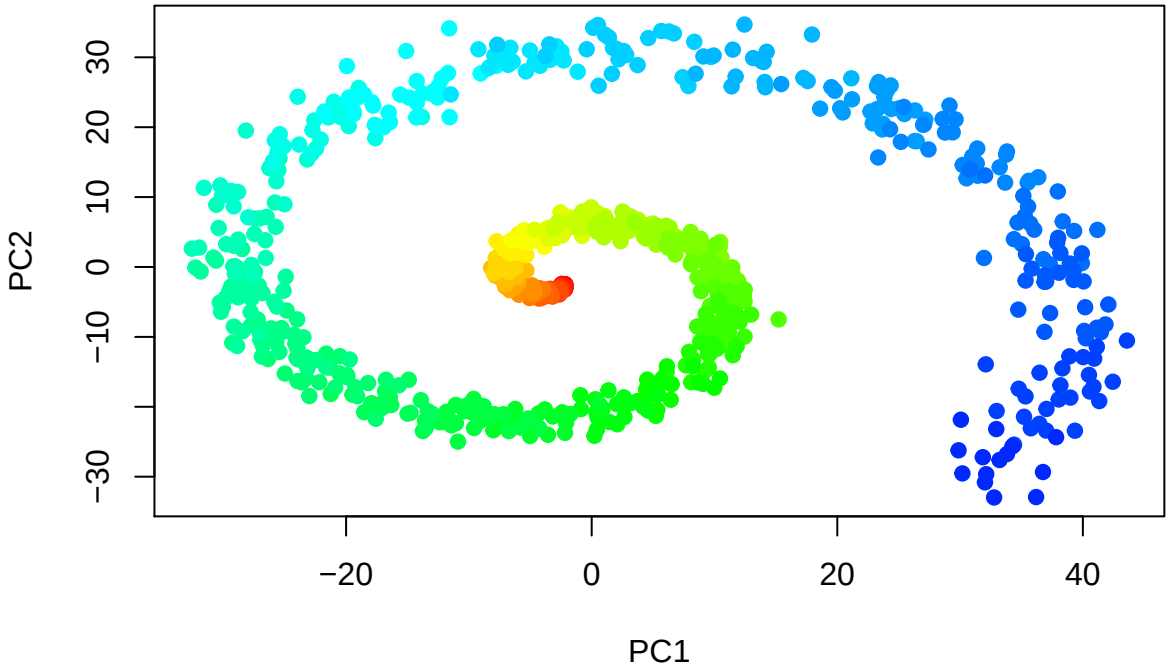
```
The reconstruction error is still large for per_y.
```

Question 8

(10 points)

Plot the first two principal component score vectors, color-coded by the `color` variable defined above. If it doesn't seem like PCA did much here, that's OK... see the next question below.

```
p = prcomp(data)
plot(p$x[,1], p$x[,2], col = color, pch = 19, xlab= "PC1", ylab= "PC2")
```



Question 9

(10 points)

In PCA, the eigendecomposition targets the covariance matrix of the scaled data. The limitation of PCA is that it is a linear algorithm, meaning that it maps the data to hyperplanes in the original space. Hence, PCA will not perform very well with data that have obvious non-linear structure, like the spiral data shown above. One way to deal with these particular data is to attempt to construct an "axis" that follows the spiral, and then another "axis" that lies orthogonal to the first one.

Here we will implement a simplified (and incomplete!) form of the *diffusion map* algorithm.

The bulk of the code is given below. What you need to do is the following:

- Pick a value of epsilon. If epsilon is too small, then neighboring data points are determined to be dissimilar and your output plot will look messy. If epsilon is too large, then all the data are considered

to be similar (you are oversmoothing here) and your output plot will look messy. If epsilon is just right, then the vast majority of the data in your final plot will lie along a line. (This means that the coordinate of the first eigenvector is following the spiral! At least approximately.)

- Once A is computed, apply eigendecomposition to it and plot the second eigenvector (along the y-axis) versus the first eigenvector, with points color-coded using the `color` variable defined above.
- Note: if there is one or more outlier points, you may need to zoom in on your plot!
- Also note: when I run my code chunk, I'm getting the spinning spinner. If you get this, just make an edit in your code chunk, and the spinning spinner will go away. Knitting works fine.

The take-away point here: if your data exhibit non-linear structure, don't trust that PCA will do a good job for you as a dimension reducer! Look into other algorithms, like diffusion map, local linear embedding, fast ICA, etc.

```
# pick a value of epsilon
epsilon = 1

# uncomment below
D = as.matrix(dist(data))  # compute pairwise Euclidean distance
K = exp(-D^2/epsilon)      # compute "similarity" matrix
v  = sqrt(apply(K,1,sum))
A = K/(v %*% t(v))         # apply graph Laplacian normalization

# apply eigendecomposition to A and plot the second vector vs. the first
eA = eigen(A)$vectors
plot(eA[,1], eA[,2], col = color, pch = 19, xlab = "First Eigenvector",
     ylab = "Second Eigenvector")
```