STAT542 HW1

Name: Chun Yin Ricky Chue (chue2@illinois.edu)

Question 1.

From the sample R markdown file (line 22 - 30) to generate $\mathbf{X}$ and $\mathbf{Y}$.

```
library(MASS)
set.seed(1)
P = 4
N = 200
rho = 0.5
V <- rho^abs(outer(1:P, 1:P, "-"))
X = as.matrix(mvrnorm(N, mu=rep(0,P), Sigma=V))
beta = as.matrix(c(1, 1, 0.5, 0.5))
Y = X %*% beta + rnorm(N)
```

The code should generate a 200×4 matrix $\mathbf{X}$.

Part a.

The maximum likelihood estimator (MLE) of the sample variance-covariance matrix $\widehat{\Sigma}$ of $\mathbf{X}$ is $\frac{1}{N}\sum_{i=1}^{N}(X_i - \bar{X})(X_i - \bar{X})$, where $N = 200$, the sample size. Since the `cov()` function in `R` calculates the unbiased covariance matrix, we would calculate the MLE of the covariance matrix by $\frac{N-1}{N}$`cov()`.

```
Sigma_X = (N-1.)/N * cov(X)
```

$\widehat{\Sigma}$ is

```
Sigma_X              ## The sample variance-covariance matrix.
```

```
##            [,1]      [,2]      [,3]       [,4]
## [1,] 0.97247606 0.3909800 0.2105751 0.04279479
## [2,] 0.39097995 0.9679975 0.4298725 0.11644234
## [3,] 0.21057509 0.4298725 1.0022939 0.38896065
## [4,] 0.04279479 0.1164423 0.3889606 0.92359611
```

The covariance matrix $\widehat{\Sigma}$ of $\mathbf{X}$ is symmetric and positive definite (Since if $\mathbf{v}^T\widehat{\Sigma}\mathbf{v} = 0$ for any vector $\mathbf{v}$, twidehat implies $\widehat{\Sigma}$ is drawn from a scalar (variance $= 0$), which is not the case here.)
For a symmetric and nonsingular matrix $\mathbf{A}$,

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I} \quad \text{and} \quad \mathbf{I} = \mathbf{I}^{\mathbf{T}}, \text{ so we have}$$

$$\mathbf{A}\mathbf{A}^{-1} = (\mathbf{A}\mathbf{A}^{-1})^T \quad \Rightarrow \mathbf{A}\mathbf{A}^{-\mathbf{1}} = (\mathbf{A}^{-1})^T\mathbf{A}^T$$

Since $\mathbf{A} = \mathbf{A}^T$, we have

$$\mathbf{A}^{-1}\mathbf{A} = (\mathbf{A}^{-1})^T\mathbf{A} \quad \Rightarrow \mathbf{A}^{-1}\mathbf{A}\mathbf{A}^{-1} = (\mathbf{A}^{-1})^T\mathbf{A}\mathbf{A}^{-1}$$

$$\mathbf{A}^{-1}\mathbf{I} = (\mathbf{A}^{-1})^T\mathbf{I} \quad \Rightarrow \mathbf{A}^{-1} = (\mathbf{A}^{-1})^T$$

Hence, the inverse of the symmetric matrix is also symmetric. For a positive definite symmetric matrix $\mathbf{A}$, define some vectors $\mathbf{u}$ and $\mathbf{v}$ such twidehat $\mathbf{u} = \mathbf{A}\mathbf{v}$.

$$\mathbf{u}^T\mathbf{A}^{-1}\mathbf{u} = \mathbf{v}^T\mathbf{A}^T\mathbf{A}^{-1}\mathbf{A}\mathbf{v} = \mathbf{v}^T\mathbf{A}^T\mathbf{v} = \mathbf{v}^T\mathbf{A}\mathbf{v} > 0$$

So, the inverse of $\mathbf{A}$ is also symmetric and positive definite. Hence, $\widehat{\Sigma}^{-1}$ exists and it is symmetric and positive definite. Using `solve()` function to solve for the inverse of $\widehat{\Sigma}$.

```
inv_Sigma_X = solve(Sigma_X)
```

Moreover, $\widehat{\Sigma}$ is diagonalizable. *i.e.* $\widehat{\Sigma}^{-1}\mathbf{v} = \lambda_i\mathbf{I}$, for λ_i being the i-th element of a diagonal matrix $\mathbf{D}$.

$$\Rightarrow (\widehat{\Sigma} - \lambda_i\mathbf{I})\mathbf{v} = 0 \quad \Rightarrow (\widehat{\Sigma} - \lambda_i\mathbf{I}) = 0$$

Using the `eigen()` function to calculate the eigenvalues and eigenvectors of $\widehat{\Sigma}^{-1}$.

```
eigen_value_Sigma_X = eigen(inv_Sigma_X)
```

Denote $\mathbf{U}$ be the matrix storing the eigenvectors of $\widehat{\Sigma}^{-1}$.
Rewrite the equation $\widehat{\Sigma}^{-1}\mathbf{U} = \mathbf{UD}$.

$$\Rightarrow \mathbf{U}^{-1}\widehat{\Sigma}^{-1}\mathbf{U} = \mathbf{U}^{-1}\mathbf{UD}$$
$$\mathbf{U}^{-1}\widehat{\Sigma}^{-\frac{1}{2}}\Sigma^{-\frac{1}{2}}\mathbf{U} = \mathbf{D}$$
$$\mathbf{U}^{-1}\widehat{\Sigma}^{-\frac{1}{2}}\mathbf{U}\mathbf{U}^{-1}\Sigma^{-\frac{1}{2}}\mathbf{U} = \mathbf{D}$$
$$(\mathbf{U}^{-1}\widehat{\Sigma}^{-\frac{1}{2}}\mathbf{U})^2 = (\mathbf{D}^{\frac{1}{2}})^2$$
$$\mathbf{U}^{-1}\widehat{\Sigma}^{-\frac{1}{2}}\mathbf{U} = \mathbf{D}^{\frac{1}{2}}$$
$$\Rightarrow \widehat{\Sigma}^{-\frac{1}{2}} = \mathbf{U}\mathbf{D}^{\frac{1}{2}}\mathbf{U}^{-1}$$

The diagonal elements of $\mathbf{D}^{\frac{1}{2}}$ is just the square roots of the eigenvalues.

```
### Create the square root of the diagonal matrix, eigenvector and the inverses.
diagonal_matrix_sqrt = diag(sqrt(eigen_value_Sigma_X$values), P, P)
U_egvector = eigen_value_Sigma_X$vectors
inv_U_egvector = solve(U_egvector)
inv_Sigma_X_sqrt = U_egvector %*% diagonal_matrix_sqrt %*% inv_U_egvector
```

$\widehat{\Sigma}^{-\frac{1}{2}}$ is

```
inv_Sigma_X_sqrt                    ### Square root of Inverse Sigma_hat.
```

```
##              [,1]         [,2]        [,3]         [,4]
## [1,]  1.087195001 -0.214630433 -0.05304807  0.009189527
## [2,] -0.214630433  1.164390335 -0.24338412  0.006920835
## [3,] -0.053048066 -0.243384122  1.15864541 -0.237377262
## [4,]  0.009189527  0.006920835 -0.23737726  1.115351840
```

Sanity check: As $\widehat{\Sigma}^{-\frac{1}{2}}\widehat{\Sigma}^{-\frac{1}{2}} = \widehat{\Sigma}^{-1}$ and $\widehat{\Sigma}^{-1}\widehat{\Sigma} = \mathbf{I}$, we check if $\widehat{\Sigma}^{-\frac{1}{2}}\widehat{\Sigma}^{-\frac{1}{2}}\widehat{\Sigma} = \mathbf{I}$

```
matrix_test = inv_Sigma_X_sqrt %*% inv_Sigma_X_sqrt %*% Sigma_X
matrix_test
```

```
##              [,1]          [,2]          [,3]          [,4]
## [1,]  1.000000e+00 -6.548581e-17  3.972517e-16  0.000000e+00
## [2,] -3.369700e-16  1.000000e+00 -2.012279e-16  6.522560e-16
## [3,]  5.481726e-16  4.024558e-16  1.000000e+00 -6.661338e-16
## [4,] -3.677614e-16  0.000000e+00 -5.551115e-16  1.000000e+00
```

The matrix product is indeed an identity matrix (with round-off errors at the off-diagonal elements).

Part b.

The Euclidean distance is defined by $\sqrt{||\mathbf{x1} - \mathbf{x2}||^2}$.

```
### Euclidean function definition.
mydist <- function(x1, x2) {
  dist_sum = 0
  for (l in 1:P) {
      dist_sum = dist_sum + (x1[l] - x2[l])**2
  }
  return(dist_sum**0.5)
}
```

Then to calculate the distance of the target point to all the entires in $\mathbf{X}$.

```
x_target = c(0.5,0.5,0.5,0.5)            ### Target point.
### A list to store the Euclidean distances of each point in X.
Eucd_dist_list = as.list(rep(0, length(N)))
for (i in 1:N) {
  Eucd_dist_list[i] = mydist(X[i,], x_target)
}
Eucd_dist_list = do.call(rbind, Eucd_dist_list)   ### Write the list in matrix form.
Eucd_dist = sort(Eucd_dist_list, index.return = TRUE)   ### Sort the distances in ascending order.
Eucd_dist_value = Eucd_dist$x[1:5]
Eucd_dist_index = Eucd_dist$ix[1:5]
```

The row numbers of the closest 5 subjects are:

```
Eucd_dist_index
```

```
## [1] 188 165 144  36 154
```

And the corresponding distances of those 5 subjects are:

```
Eucd_dist_value
```

```
## [1] 0.4312105 0.6101556 0.6155711 0.6491776 0.7045383
```

The corresponding $\mathbf{Y}$ values of the 5-NN are:

```
Eucd_nearY = Y[Eucd_dist_index]
Eucd_nearY
```

```
## [1]  1.8228239 -0.3023569  1.1638351  1.9163523  2.1325402
```

Hence, the 5-NN estimation at the target point is the average of these 5 values, *i.e.*

```
Eucd_5NN = mean(Eucd_nearY)
Eucd_5NN
```

```
## [1] 1.346639
```

Part c.

The Mahalanobis distance is defined by $\sqrt{(\mathbf{x1} - \mathbf{x2})^T \widehat{\Sigma}^{-1} (\mathbf{x1} - \mathbf{x2})}$.

```
### Mahalanobis function definition.
mydist2 <- function(x1, x2, s) {
  dist_mod = as.list(rep(0, length(P)))
  for (l in 1:P) {
```

```
      dist_mod[l] = x1[l] - x2[l]
  }
  dist_mod = do.call(rbind, dist_mod)
  Maha_distt = sqrt(t(dist_mod) %*% s %*% dist_mod)
  return(Maha_distt)
}
```

Then to calculate the Mahalanobis distance of the target point to all the entires in $\mathbf{X}$.

```
x_target = c(0.5,0.5,0.5,0.5)            ### Target point.
Maha_dist_list = as.list(rep(0, length(N)))
### A list to store the Euclidean distances of each point in X.
for (i in 1:N) {
  Maha_dist_list[i] = mydist2(X[i,], x_target, inv_Sigma_X)
}
Maha_dist_list = do.call(rbind, Maha_dist_list)   ### Write the list in matrix form.
Maha_dist = sort(Maha_dist_list, index.return = TRUE)   ### Sort the distances in ascending order.
Maha_dist_value = Maha_dist$x[1:5]
Maha_dist_index = Maha_dist$ix[1:5]
```

The row numbers of the closest 5 subjects are:

```
Maha_dist_index
```

```
## [1] 188  49 144 165  36
```

And the corresponding distances of those 5 subjects are:

```
Maha_dist_value
```

```
## [1] 0.4681211 0.7241747 0.7342852 0.7680334 0.8337267
```

The corresponding $\mathbf{Y}$ values of the 5-NN are:

```
Maha_nearY = Y[Maha_dist_index]
Maha_nearY
```

```
## [1]  1.8228239 -1.7928659  1.1638351 -0.3023569  1.9163523
```

Hence, the 5-NN estimation at the target point is the average of these 5 values, *i.e.*

```
Maha_5NN = mean(Maha_nearY)
Maha_5NN
```

```
## [1] 0.5615577
```

Part d.

In this case, the Euclidean distance estimator seems to perform better, that the estimated average is much closer to the majority of the 5-NN's.

The reason behind could be that there is one neighbor (index 165, value -0.3023569) whose value is far from the remaining four of the neighbors. The use of covariance function in the Mahalanobis distance could emphasize the effect of this "outlier" to the estimation.

Question 2.

```
set.seed(Sys.time())              ### Remove seed
library(class)
library(kknn)
```

Part a.

For k-NN, the estimated response $\widehat{\mathbf{y}}_\mathbf{i}$ of a data point x_i is given by:

$$\widehat{\mathbf{y}}_\mathbf{i}(x_i) = \sum_{j=1}^{n} w(x_i, x_j) \cdot y_j$$

where n is the total number of training data, (x_j, y_j) are the j-th training data, and $w(x_i, x_j)$ is the weight of the pairs (x_i, x_j).

When computing the averages by the kNN algorithm, each training data is treated as an independent entry, so we can write

$$\widehat{\mathbf{y}}_\mathbf{i} = \mathbf{D}\mathbf{y}$$

where $\mathbf{D}$ is a diagonal $n \times n$ matrix, defined as $\mathbf{D}_{ij} = w(x_i, x_j)$

Therefore, for degree of freedom:

$$\mathrm{df}(\widehat{\mathbf{y}}_\mathbf{i}) = \frac{1}{\sigma^2} \sum_{i=1}^{n} \mathrm{cov}(\mathbf{D}\mathbf{y}, \mathbf{y})) = \frac{1}{\sigma^2} \mathrm{tr}(\mathrm{cov}(\mathbf{D}\mathbf{y}, \mathbf{y})) \quad \text{Since } \mathbf{D} \text{ is diagonal}$$

$$= \frac{1}{\sigma^2} \mathrm{tr}(\mathbf{D}\mathrm{cov}(\mathbf{y}, \mathbf{y})) = \mathrm{tr}(\mathbf{D}) = \sum_{i=1}^{n} w(x_i, x_j)$$

Now, consider the kNN algorithm.

$$w(x_i, x_j) = \frac{1}{k} \quad \text{if } x_j \text{ is one of the neightbors to } x_i$$

$$w(x_i, x_j) = 0 \quad \text{otherwise}$$

So, degree of freedom of kNN is

$$\mathrm{df}(\widehat{\mathbf{y}}_\mathbf{i}) = \sum_{i=1}^{n} \frac{1}{k} = \frac{n}{k}$$

For $k = 5$,

$$\mathrm{df}(\widehat{\mathbf{y}}_\mathbf{i}(k = 5)) = \frac{n}{5}$$

Part b.

Generate a design matrix $\mathbf{X}$ from independent standard normal distribution with $N = 200$ and $p = 4$.

```
  P = 4
  N = 200
  V <- diag(P)      ### 4 x 4 identity matrix, since no covariance between entries in X.
  X = as.matrix(mvrnorm(N, mu=rep(0,P), Sigma=V))
```

Now, define a true model $Y = (X_1 + 2X_2)\sin(X_3 + X_4)$, the true Y's.

```
Y_true = (X[,1] + 2 * X[,2]) * sin(X[,3] + X[,4])
```

The response variables of the 200 observations are generated by adding independent standard normal noise to the true model, and repeat by 20 times.

```
TRIAL_NUM = 20
Y_data = matrix(NA, TRIAL_NUM, N)
for (l in 1:TRIAL_NUM) {
     Y_data[l,] = Y_true + rnorm(N)
}
```

Apply 5NN algorithm for 20 times to obtain the $\widehat{y_i}$'s. The observations (x_i, y_i)'s are taken away from the training data and instead used as a testing data to estimate $\widehat{y_i}$.

```
Y_estimate = matrix(NA, N, TRIAL_NUM)
for (l in 1:TRIAL_NUM) {
  for (k in 1:N) {
    d = seq(1,N)
    d = d[d!=k]     ### Remove the point itself as a training data.
    knn_reg = kknn(Y_data[l,d] ~ ., train = data.frame(x = X[d,], y = Y_data[l,d]),
              test = data.frame(x = t(X[k,]), y = Y_data[l,k]), k = 5, kernel = "rectangular")
    Y_estimate[k,l] = knn_reg$fitted.values
  }
}
```

Calculate the covariance for each observation. Construct matrix for storing the kNN result and model value of each observation. Finally, the code sums up the trace of covariance for the 200 observations.

For the definition of degrees of freedom, in this case $\sigma^2 = 1$ (since the noise, $\epsilon \sim N(0, 1)$).

So, $\text{df}_{estimated} = \sum_{i=1}^{n} \text{cov}(\widehat{y_i}, y_i)$.

```
Y_true = matrix(Y_true, 1, N)
y_cov_tr = 0
for (l in 1:N) {
  y_obs = data.frame(y_est = Y_estimate[l,], y_real = matrix(c(Y_true[,l],Y_true[,l])))
  y_cov = cov(y_obs)              ### Covariance of matrix
  y_cov_tr = y_cov_tr + sum(diag(y_cov))     ### Trace of matrix.
}
```

Therefore, the estimated degree of freedom is:

```
y_cov_tr
```

```
## [1] 99.9258
```

The theoretical degree of freedom for 5NN algorithm for 200 observations is $\frac{n}{k} = \frac{200}{5} = 40$. The estimated value is ~ 2 times higher than the theoretical value.

Part c.

Say if $\mathbf{X}$ is a $n \times p$ predictor matrix. Write the degree of freedom of linear regression in the following form.

$$\mathrm{df}(\widehat{\mathbf{y}}) = \frac{1}{\sigma^2}\mathrm{Tr}(\mathrm{cov}(\widehat{\mathbf{y}}, \mathbf{y})) = \frac{1}{\sigma^2}\mathrm{Tr}(\mathrm{cov}(\mathbf{X}(\mathbf{X^T X})^{-1}\mathbf{X^T y}, \mathbf{y})) = \frac{1}{\sigma^2}\mathrm{Tr}(\mathbf{X}(\mathbf{X^T X})^{-1}\mathbf{X^T}\mathrm{cov}(\mathbf{y}, \mathbf{y}))$$

For $\epsilon_i \in N(0, \sigma^2)$ and are i.i.d., the cross terms in $\mathrm{cov}(\mathbf{y}, \mathbf{y})$ are 0, and so $\mathrm{cov}(\mathbf{y}, \mathbf{y}) = \sigma^2$. So,

$$\mathrm{df}(\widehat{\mathbf{y}}) = \mathrm{Tr}(\mathbf{X}(\mathbf{X^T X})^{-1}\mathbf{X^T}) = \mathrm{Tr}(\mathbf{X^T X}(\mathbf{X^T X})^{-1}) = p$$

So, the theoretical degree of freedom for linear regression is p, where p is the number of parameters.

```
set.seed(Sys.time())              ### Remove seed
```

Question 3.

Load the `SAheart` dataset from the `ElemStatLearn` package, the kNN and the classification packages.

```
library(ElemStatLearn)
library(class)
library(kknn)
data(SAheart)
```

Assign the Cross-Validation (CV) groups of the observations.

```
SAheart_len = length(SAheart$age)          ### Length of the data list.
nfold = 10
infold = sample(rep(1:nfold, length.out = SAheart_len))
```

Select the predictive features, `tobacco` and `age`.

```
SAheart_table = subset(SAheart,select = c(tobacco,age))
SAheart_chd = SAheart$chd
```

The next step is to normalize the predictive features, it transforms all the values to a common scale.

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
SAheart_n <- as.data.frame(lapply(SAheart_table, normalize))
```

Check if the normalization has been conducted properly or not.

```
summary(SAheart_n)
```

```
##     tobacco              age
##  Min.   :0.000000   Min.   :0.0000
##  1st Qu.:0.001683   1st Qu.:0.3265
##  Median :0.064103   Median :0.6122
##  Mean   :0.116527   Mean   :0.5677
##  3rd Qu.:0.176282   3rd Qu.:0.8163
##  Max.   :1.000000   Max.   :1.0000
```

Apply kNN algorithm for 10-fold cross validation. The error matrix stores the fraction of misclassified data, *i.e.*$1 - \frac{\text{misclassified}}{\text{total}}$.

```
K = 50 # maximum number of k considered.
errorMatrix = matrix(NA, K, nfold) # save the prediction error of each fold

for (l in 1:nfold)
{
  for (k in 1:K)
    {
      knn.fit = knn(train = SAheart_n[infold != l, ], test = SAheart_n[infold == l, ],
                    cl = SAheart_chd[infold != l], k = k)
      errorMatrix[k, l] = 1 - mean(knn.fit == SAheart_chd[infold == l])
    }
}
```

Find the averaged CV error.

```
error_mean = rowMeans(errorMatrix)
min_error_k = which.min(error_mean)
```

The best k which gives the lowest CV error:

```
min_error_k
```

```
## [1] 13
```

To refit the model of best k with all the data to get the training error.

```
errorList = list(NA, nfold) # save the prediction error of each fold

for (l in 1:nfold)
{
     knn.fit = knn(train = SAheart_n[infold != l, ], test = SAheart_n,
                   cl = SAheart_chd[infold != l], k = min_error_k)
     errorList[l] = 1 - mean(knn.fit == SAheart_chd)
}

trainerror = matrix(errorList, nrow = 1, ncol = nfold)
trainerror = as.numeric(as.character(unlist(trainerror)))
trainerror_mean = mean(trainerror)
```

The training error for the model with best k:

```
trainerror_mean
```

```
## [1] 0.2683983
```

Plot the averaged cross-validation error curve for different k's.

```
plot(error_mean, xlab = "k", ylab = "CV_error", col = "orange", type = "l",
     lwd = 2.5, main = "Averaged CV error with different k's", ylim = c(0.2,0.5))
points(x = min_error_k, y = error_mean[min_error_k], col = "red", pch = 19)
legend(x = 42, y = 0.50, legend = "Best k", col = "red", pch = 19)
```