STAT542 Statistical Learning Homework 1

Huamin Zhang

Sep. 15 2017

Name: Huamin Zhang (huaminz2@illinois.edu)

Question 1

Download the R markdown file from our course website (at the bottom of the google site, or click here). Follow the exact same code (line 22-30) to generate X and Y (do not change the random seed), then perform the following tasks. In this question, you are NOT allowed to use any additional R package (except the "MASS" package which is already used for generating the multivariate normal samples).

a) [10 points]

Calculate the sample variance-covariance matrix $\hat{\Sigma}$ of X (using the maximum likelihood estimator, not the unbiased version). Then calculate $\hat{\Sigma}^{-\frac{1}{2}}$.

Answer:

```
library(MASS)
set.seed(1)
#create the dataset
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)

#compute the variance-covariance matrix
one = as.matrix(rep(1,N))
x = X - 1/N * one %*% t(one) %*% X
var_cov_a = t(x) %*% x * (1/N)
var_cov_a
```

```
##            [,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
```

```
#var_cov_a = cov(X)*(N-1)/N
#var_cov_a = cov.wt(X, wt = rep(1/nrow(X), nrow(X)), method = "ML")$cov
```

So the sample variance-covariance matrix of X is

$$\hat{\Sigma} = \begin{pmatrix} 1.0479268 & 0.5699420 & 0.3043565 & 0.1353540 \\ 0.5699420 & 1.1175618 & 0.5781821 & 0.2928669 \\ 0.3043565 & 0.5781821 & 1.0271222 & 0.5250593 \\ 0.1353540 & 0.2928669 & 0.5250593 & 1.0102321 \end{pmatrix}$$

```
#compute the sigma^(-1/2)
eigenvector <- eigen(var_cov_a)
var_cov_b = eigenvector$vectors %*% diag(eigenvector$values^(-1/2)) %*% t(eigenvector$vectors)
var_cov_b
```

```
##              [,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
```

Thus, we get

$$\hat{\Sigma}^{-\frac{1}{2}} = \begin{pmatrix} 1.087195001 & -0.214630433 & -0.05304807 & 0.009189527 \\ -0.214630433 & 1.164390335 & -0.24338412 & 0.006920835 \\ -0.053048066 & -0.243384122 & 1.15864541 & -0.237377262 \\ 0.009189527 & 0.006920835 & -0.23737726 & 1.115351840 \end{pmatrix}$$

b) [15 points]

We want to perform a 5-NN estimation at the target point $x = (0.5, 0.5, 0.5, 0.5)^T$. To do this, lets first write a function `mydist <- function(x1, x2)` that will return the Euclidean distance between any two vectors x1 and x2. Calculate the distance from all sample points to the target point x and output the row numbers of the closest 5 subjects. Use their Y values to obtain the 5-NN estimation at the target point.

Answer:

```
mydist <- function(x1, x2){
  return(sqrt(sum((x1-x2)^2)))
}

x = c(0.5,0.5,0.5,0.5)
Euclidean_distance = rep(0,N)
for(i in 1:N){
  Euclidean_distance[i] = mydist(x,X[i,])
}
#The row numbers of the closest 5 subjects
closet_id = order(Euclidean_distance)[1:5]
closet_id
```

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

```
#The Euclidean distance of the closest 5 subjects
Euclidean_distance[closet_id]
```

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

```r
y = mean(Y[closet_id])
y
```

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

The distance from all sample points to the target point x is stored in the array `Euclidean_distance`. The row numbers of the closest 5 subjects is 188, 165, 144, 36, 154. Using their Y values to obtain the 5-NN estimation at the target point, we get the estimation `y` = 1.3466389

c) [10 points]

Write another function `mydist2 <- function(x1, x2, s)` that returns the Mahalanobis distance between any two vectors $x1$ and $x2$ using any covariance matrix `s`. Redo the steps in b) to find the 5-NN estimation based on this Mahalanobis distance with $s = \hat{\Sigma}$.

Answer:

```r
mydist2 <- function(x1, x2, s){
  x = x1 - x2
  d2 = t(x) %*% solve(var_cov_a) %*% x
  return(sqrt(d2))
}
Mahalanobis_distance = rep(0,N)
for(i in 1:N){
  Mahalanobis_distance[i] = mydist2(x,X[i,],var_cov_a)
}
#The row numbers of the closest 5 subjects
closet_id2 = order(Mahalanobis_distance)[1:5]
closet_id2
```

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

```r
#The Mahalanobis distance of the closest 5 subjects
Mahalanobis_distance[closet_id2]
```

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

```r
y2 = mean(Y[closet_id2])
y2
```

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

The distance from all sample points to the target point x is stored in the array `Mahalanobis_distance`. The row numbers of the closest 5 subjects is 188, 49, 144, 165, 36. Using their Y values to obtain the 5-NN estimation at the target point, we get the estimation `y` = 0.5615577

d) [5 points]

Which estimator seems to perform better? Can you give any explanation?

Answer:

The estimation based on the Mahalanobis distance is better.

The Euclidean distance assumes the data to be isotropically Gaussian and treat each feature equally. It means all variables are measured in the same units of length and it is blind to correlated variables. On the other hand, the Mahalanobis distance relaxes the assumption of the Euclidean distance and takes the covariances into account. For example, in a 2D case, the Mahalanobis distance leads to elliptic decision boundaries and on the contrary the Euclidean distance leads to the circular boundary. When using Euclidean distance, the set of points equidistant from a given location is a sphere. So when the distribution to be decidedly non-spherical, for instance ellipsoidal, Euclidean distance doesn't work well. However, the Mahalanobis distance performs better because it stretches this sphere to correct for the respective scales of the different variables, and to account for correlation among variables.

In this problem, we take the covariances(the matrix `V`)into account when building the dataset(X = as.matrix(mvrnorm(N, mu=rep(0,P), Sigma=V))). Also we can find that the variance-covariance matrix we calculated in question(a) seems to be approaching to the matrix `V`. So I think the estimation based on the Mahalanobis distance is better.

Here we can also get the result from simulation. We redo the steps to generate X and Y and create 2200 instances, including 2000 instances for training set and 200 instances for test set. Then we use square-error loss to compare their performance. We repeat the steps for 50 times. Here is the code.

```
set.seed(10086)
a = rep(0,50)
b = rep(0,50)

Euclidean <- function(X1,Y1,X,Y,z){
  y1 = rep(0,200)
  for(k in 1:200){
    Euclidean_distance = rep(0,2000)
    for(i in 1:2000){
      Euclidean_distance[i] = mydist(X1[k,],X[i,])
    }
    closet_id = order(Euclidean_distance)[1:z]
    y = mean(Y[closet_id])
    y1[k] = y
  }
  dis = sum(Y1-y1)^2 / 200
  return(dis)
}

Mahalanobis <- function(X1,Y1,X,Y,s,z){
  y1 = rep(0,200)
  for(k in 1:200){
    Mahalanobis_distance = rep(0,2000)
    for(i in 1:2000){
      Mahalanobis_distance[i] = mydist2(X1[k,],X[i,],s)
    }
    closet_id = order(Mahalanobis_distance)[1:z]
    y = mean(Y[closet_id])
    y1[k] = y
  }
  dis = sum(Y1-y1)^2 / 200
  return(dis)
}
for(p in 1:50){
  P = 4
  N = 2200
```

```
  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)
  X1 = X[1:200,]
  Y1 = Y[1:200]
  X = X[201:2200,]
  Y = Y[201:2200]
  z = 5

  one = as.matrix(rep(1,2000))
  x = X - 1/2000 * one %*% t(one) %*% X
  var_cov_a = t(x) %*% x * (1/2000)

  a[p] = Euclidean(X1,Y1,X,Y,z)
  b[p] = Mahalanobis(X1,Y1,X,Y,var_cov_a,z)
}
```

Then we compare the mean of square-error loss in the 30 experiments. We can find the square-error loss of the estimation based on the Mahalanobis distance is 1.2194768 which is smaller than that based on the Euclidean distance 1.2310721.

```
mean(a)
```

```
## [1] 1.231072
```

```
mean(b)
```

```
## [1] 1.219477
```

Question 2

You already know how to perform kNN on any target point x. Now, perform a simulation study to estimate the degrees of freedom of a k-nearest neighbor method for regression. The degrees of freedom of a fit is defined as $\sum_{i=1}^{n} Cov(\hat{y}_i, y_i)/\sigma^2$. You should proceed as follows:

a) [10 points]

If we are interested in using k = 5, derive the degrees of freedom of this model using the given formula.

Answer:

Assume $y_i = f(x_i) + \epsilon_i$ where where the errors $\epsilon_i (i = 1, 2, \ldots, n)$ are uncorrelated with common variance $\sigma^2 > 0$, and the fitted value $\hat{Y} = \hat{f}(X)$. Then, for kNN algorithm, we have

$$\hat{y}_i = \frac{1}{k} \sum_{x_j \in N_k(x_i)} y_j$$

where where $N_k(x_i)$ defines the k samples from the training data that are closest to x_i. So the degree of freedom is

$$\begin{aligned} df(\hat{(f)}) &= \frac{1}{\sigma^2}\sum_{i=1}^{n} Cov(\hat{y}_i, y_i) \\ &= \frac{1}{\sigma^2}\sum_{i=1}^{n} Cov(\frac{1}{k}\sum_{x_j \in N_k(x_i)} y_j, y_i) \\ &= \frac{1}{\sigma^2}\sum_{i=1}^{n} \frac{\sigma^2}{k} \\ &= \frac{n}{k} \end{aligned}$$

So using k = 5, the degrees of freedom is $df = \frac{n}{5}$

b) [20 points]

Perform the simulation study:

-Generate a design matrix X from independent standard normal distribution with $n = 200$ and $p = 4$. Now, Fix these X values for the rest of this problem.

-Define an appropriate true model `f(X)` (choose whatever function you want) as the mean of Y .

-Using your model, generate the response variables for these 200 observations by adding an independent standard normal noise ϵ. Fit 5-nearest neighbor to the data (you can use existing package if you like). Obtain $\hat{y}_i$.

-To get a good estimate of $Cov(\hat{y}_i, yi)$, you need to perform this experiment multiple times and calculate a sample covariance. Repeat the previous step 20 times to calculate the estimation. Keep in mind that you do not change X values, only re-generate Y for each run.

-Compare your estimated degrees of freedom with the theoretical value that you derived in (a).

Answer:

Generate a design matrix X from independent standard normal distribution with $n = 200$ and $p = 4$.

```
library(kknn)
set.seed(0)
#create dataset
P = 4
N = 200
I = diag(nrow = 4)
X = as.matrix(mvrnorm(N, mu=rep(0,P), Sigma=I))
```

Define an model $f(X) = X\beta$ where $\beta = (1, 1, 0.5, 0.5)^T$

```
beta = as.matrix(c(1, 1, 0.5, 0.5))
Y = X %*% beta + rnorm(N)
```

Generate the response variables for these 200 observations by adding an independent standard normal noise ϵ. Fit 5-nearest neighbor to the data (you can use existing package if you like). Obtain $\hat{y}_i$.

The fitted values are stored in `test.pred`.

```
k = 5
knn.fit = kknn(Y ~ X, train = data.frame(x = X, y = Y), test = data.frame(x = X, y = Y),
               k = k, kernel = "rectangular")
test.pred = knn.fit$fitted.values
```

Repeat the previous step 20 times to calculate the estimation. The response variables and fitted values are stored in `Y.train` and `Y.pred`.

```
Y.pred = NULL
Y.train = NULL
for(i in 1:20){
  Y = X %*% beta + rnorm(N)
  Y.train = cbind(Y.train, Y)
  knn.fit = kknn(Y ~ X, train = data.frame(x = X, y = Y), test = data.frame(x = X, y = Y),
                 k = k, kernel = "rectangular")
  Y.pred = cbind(Y.pred, knn.fit$fitted.values)
}
```

Compare your estimated degrees of freedom with the theoretical value

```
sum_cov = 0
for(j in 1:N){
  sum_cov = sum_cov + cov(Y.pred[j,],Y.train[j,])
}
df = sum_cov / 1
df
```

```
## [1] 39.40965
```

Thus, the estimated degree of freedom is $df = 39.409648$. According to the theoretical value derived in (a), the degree of freedom is $df = \frac{n}{k} = 40$. The estimated degree of freedom is close to the theoretical value.

c) [10 points]

Consider the a linear model $y = X\beta + \epsilon$, and the fitted value from linear regression $\hat{y} = X\hat{\beta} = X(X^TX)^{-1}X^Ty$. For simplicity, lets assume that ϵ_i are i.i.d. normal with mean 0 and variance σ^2. Recall the alternative definition of the degrees of freedom: $df(\hat{f}) = \frac{1}{\sigma^2}Trace(Cov(\hat{y}, y))$. What is the theoretical degrees of freedom for this linear regression?

Ansewer:

Assume there is a linear model $y = X\beta + \epsilon$ where X is a $n * p$ matrix. Since $E[\epsilon] = 0$ and $Var(\epsilon) = \sigma^2$, we get $Cov(y, y) = E[\epsilon^2] = Var(\epsilon) + E^2[\epsilon] = \sigma^2$. Thus,

$$\begin{aligned}
df((\hat{f})) &= \frac{1}{\sigma^2}tr(Cov(X(X^TX)^{-1}X^Ty, y)) \\
&= \frac{1}{\sigma^2}tr(X(X^TX)^{-1}X^TCov(y, y)) \\
&= \frac{1}{\sigma^2}tr(X(X^TX)^{-1}X^T\sigma^2) \\
&= tr(X(X^TX)^{-1}X^T) \\
&= tr(X^TX(X^TX)^{-1}) \\
&= tr(I_{p*p}) \\
&= p
\end{aligned}$$

So the degrees of freedom for this linear regression is p which is the number of features in dataset X

Question 3

Load the SAheart dataset from the ElemStatLearn package. Consider kNN model using two variables, age and tobacco to model the binary outcome chd (coronary heart disease). Use 10-fold cross validation to select the best k. Also report your training error and plot the averaged cross-validation error curve for difference choices of k. Note: you can find some examples in our intro.r file, but feel free to improve it.

Answer:

Here are 462 instances in the dataset. Since we do the 10-fold cross validation, there will be as least $462 - 47 = 415$ instances in the training data. So we try the value of k from 1 to 415.

```
set.seed(0)
library(ElemStatLearn)
library(class)
head(SAheart)
```

```
##   sbp tobacco  ldl adiposity famhist typea obesity alcohol age chd
## 1 160   12.00 5.73     23.11 Present    49   25.30   97.20  52   1
## 2 144    0.01 4.41     28.61  Absent    55   28.87    2.06  63   1
## 3 118    0.08 3.48     32.28 Present    52   29.14    3.81  46   0
## 4 170    7.50 6.41     38.03 Present    51   31.99   24.26  58   1
## 5 134   13.60 3.50     27.78 Present    60   25.99   57.34  49   1
## 6 132    6.20 6.47     36.21 Present    62   30.77   14.14  45   0
```

```
x = SAheart[,c('age','tobacco')]
y = SAheart[,'chd']

nfold = 10
infold = sample(rep(1:nfold, length.out=dim(x)[1]))
```

```
K = floor(dim(x)[1] *9 / 10) # maximum number of k that I am considering
train_errorMatrix = matrix(NA, K, nfold) # save the prediction error of each fold
test_errorMatrix = matrix(NA, K, nfold) # save the prediction error of each fold

for (l in 1:nfold){
  for (k in 1:K){
    knn.train <- knn(train = x[infold != l, ], test = x[infold != l, ], cl = y[infold != l], k=k)
    knn.test <- knn(train = x[infold != l, ], test = x[infold == l, ], cl = y[infold != l], k=k)
    train_errorMatrix[k,l] = mean(knn.train != y[infold != l])
    test_errorMatrix[k, l] = mean(knn.test != y[infold == l])
  }
}
```

The mean training error and cross-validation error are stored in `avg_train_error` and `avg_test_error`

```
avg_train_error = apply(train_errorMatrix,1,mean)
avg_test_error = apply(test_errorMatrix,1,mean)
```

Here is some part of the average training error($k = 1, \ldots, 6$)(Here we only report part of the average training error to avoid producing an excessively long report). We can find that when k is large enough, the training error will not change as k increases.

```r
head(avg_train_error)
```

```
## [1] 0.03439354 0.19938485 0.21140350 0.25973703 0.26455456 0.26215883
```

```r
best_k = order(avg_test_error)[1]
best_k
```

```
## [1] 74
```

```r
avg_train_error[best_k]
```

```
## [1] 0.2890842
```

```r
avg_test_error[best_k]
```

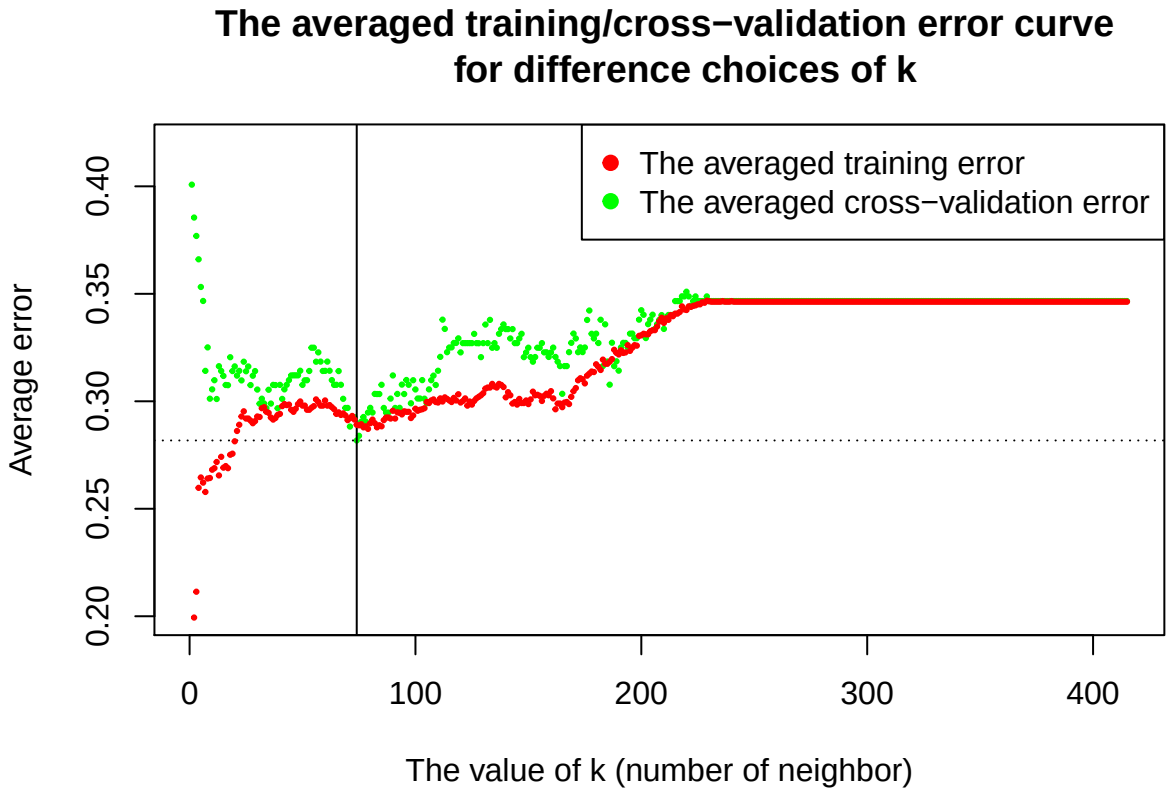
```
## [1] 0.2817761
```

```r
#use all data refit the model with best k
knn.best_k <- knn(train = x, test = x, cl = y, k=best_k)
knn_train_error.best_k = mean(knn.best_k != y)
knn_train_error.best_k
```

```
## [1] 0.2922078
```

According to the cross-validation error, we think $k = 74$ is the best value of k. When we use $k = 74$, the averaged training error in 10-fold cross validation is 0.2890842 and the cross-validation error is 0.2817761. We refit the KNN model with the best k, and the train error is 0.2922078. Here is the plot of the averaged training error and the averaged cross-validation error curve for difference choices of k.

```r
plot(avg_test_error,pch = 19,cex=0.3,col='green',xlab = 'The value of k (number of neighbor)',
     ylab = 'Average error', main = ' The averaged training/cross-validation error curve
     for difference choices of k',ylim=c(0.2,0.42))
abline(v = best_k)
abline(h = avg_test_error[best_k], lty = 3)
points(avg_train_error,pch = 19,cex=0.3,col='red')
legend( "topright", c("The averaged training error ","The averaged cross-validation error"),
        pch = 19, col = c('red','green'), cex = 1)
```


The averaged training/cross−validation error curve
for difference choices of k
The averaged training error
The averaged cross−validation error
Average error
0.20
0.25
0.30
0.35
0.40
0
100
200
300
400
The value of k (number of neighbor)