STAT34800 HW1

Sarah Adilijiang

Problem A

(a)

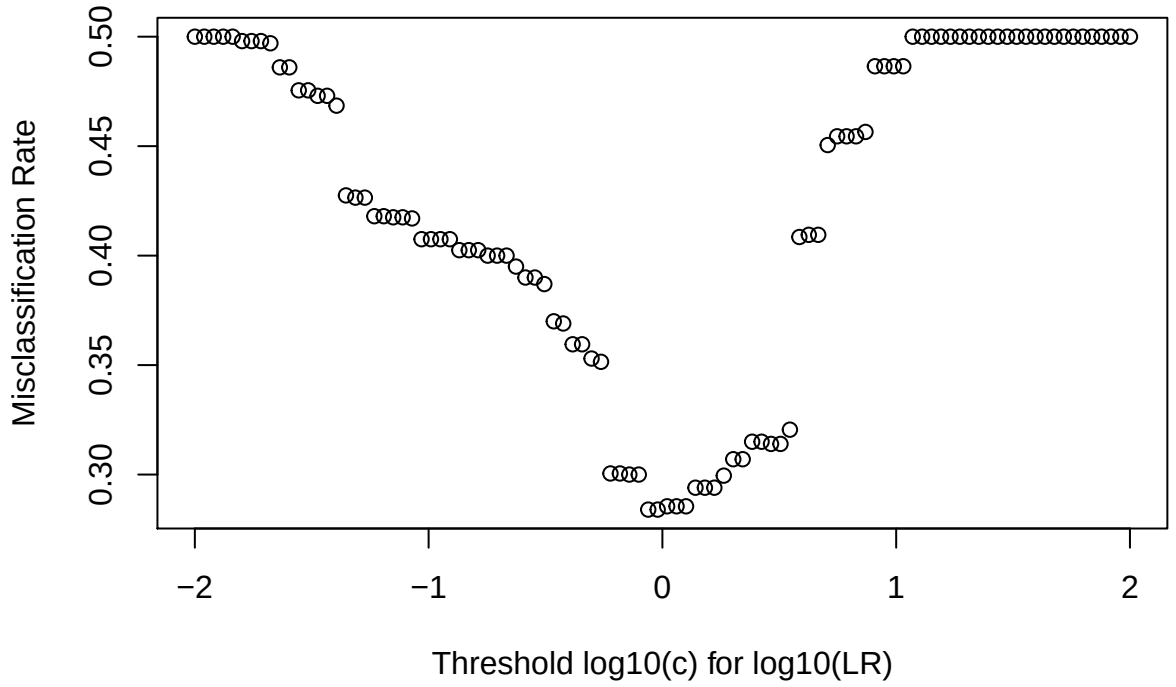
```
#' Simulation study
#'
#' Simulates data from forest and savanna tusks from prespecified frequencies, and uses LR to classify
#'
#' @param nS number of savanna tusks to use
#' @param nF number of forest tusks to use
#' @param fS allele frequencies of savanna elephants
#' @param fF allele frequencies of forest elephants
#' @param lc vector of log(c) values of LR cutoff to use
#' @param seed set a random seed using set.seed() for reproducibility
#'
#' @return Plot of misclassification rate against threshold
```

```
simulation_study = function(nS, nF, fS, fF, lc, seed){
    # simulate data
    set.seed(seed)
    x = matrix(NA,nS+nF,6)
    for (i in 1:6) {
        x[1:nS,i] = rbinom(nS,1,fS[i])
        x[(nS+1):(nS+nF),i] = rbinom(nF,1,fF[i])
    }

    # compute LR
    L = function(f,x){ prod(f^x * (1-f)^(1-x)) }
    LR = rep(NA, nS+nF)
    for (i in 1:length(LR)) {
        LR[i] = L(fS,x[i,])/L(fF,x[i,])
    }

    # calculate misclassification rate
    true_class = rep(c("S","F"),c(nS,nF))
    mis_rate = rep(NA,length(lc))
    for (i in 1:length(lc)) {
        LR_class = rep(NA, nS+nF)
        for (j in 1:length(LR)) {
            LR_class[j] = ifelse(LR[j]>10^lc[i],"S","F")
        }
        mis_rate[i] = mean(LR_class!=true_class)
    }

    # min misclassification rate & plot
    print(paste0("c = ", round(10^lc[which.min(mis_rate)],6), " (i.e. log10(c) = ",
                 round(lc[which.min(mis_rate)],6), ") minimizes the misclassification rate."))
    print(paste0("The minimum misclassification rate is ", min(mis_rate)))
```

```
    plot(mis_rate~lc,xlab="Threshold log10(c) for log10(LR)",ylab="Misclassification Rate")
}
```

```
fS = c(0.40,0.12,0.21,0.12,0.02,0.32)
fF = c(0.80,0.20,0.11,0.17,0.23,0.25)
lc = seq(-2,2,length=100)
simulation_study(nS=1000, nF=1000, fS=fS, fF=fF, lc=lc, seed=100)
```

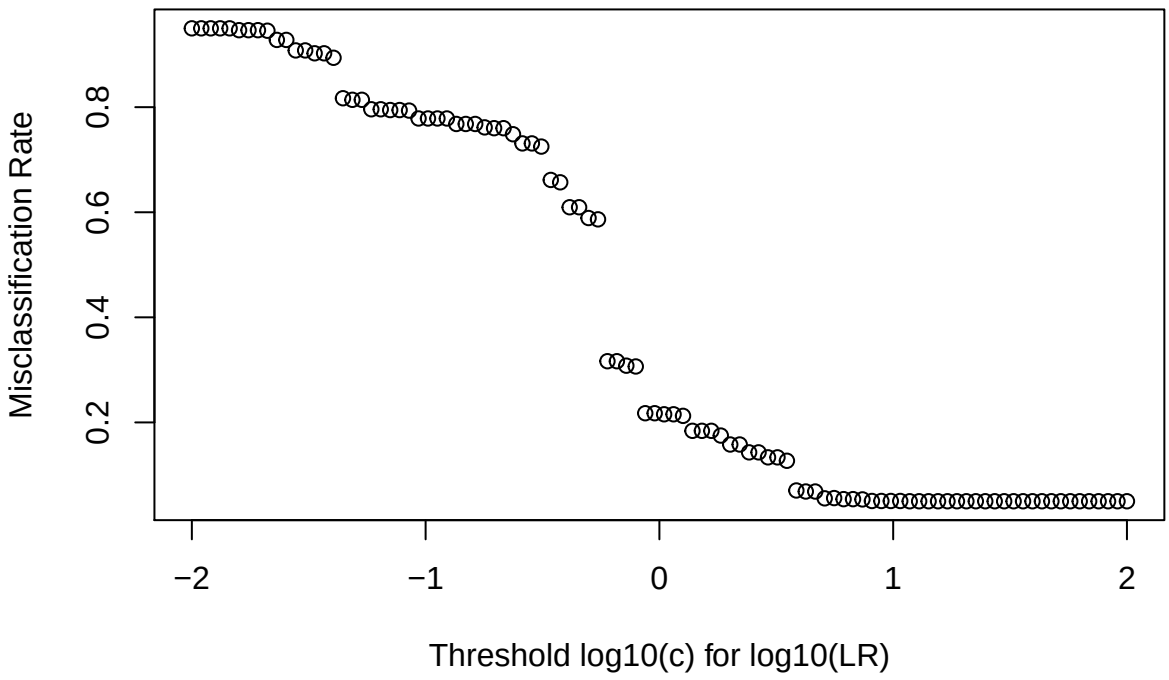
```
## [1] "c = 0.869749 (i.e. log10(c) = -0.060606) minimizes the misclassification rate."
## [1] "The minimum misclassification rate is 0.284"
```



(b)

```
simulation_study(nS=100, nF=1900, fS=fS, fF=fF, lc=lc, seed=100)
```

```
## [1] "c = 11.76812 (i.e. log10(c) = 1.070707) minimizes the misclassification rate."
## [1] "The minimum misclassification rate is 0.05"
```



Comments:

(1) In question (a), we simulate 1000 tusks from M_S and 1000 tusks from M_F, so the prior probability of two models are equal (50% for both M_S and M_F). In this case, as the threshold c increases, the miscalssificate rate starts from about 50%, and begins to decrease until reaches a minimum value, then starts to increase again until reaches about 50%. When c = 0.869749, i.e. log10(c) = -0.060606, the misclassification rate reaches its minimum value, which is about 28.4%.

(2) In question (b), we simulate 100 tusks from M_S and 1900 tusks from M_F, so the prior probability of two models are very different (5% for M_S and 95% for M_F). In this case, as the threshold c increases, the miscalssificate rate starts from near 95%, then generally keeps decreasing until reaches a minimum value about 5% and stops. When c = 11.76812, i.e. log10(c) = 1.070707, the misclassification rate reaches its minimum value, which is about 5%.

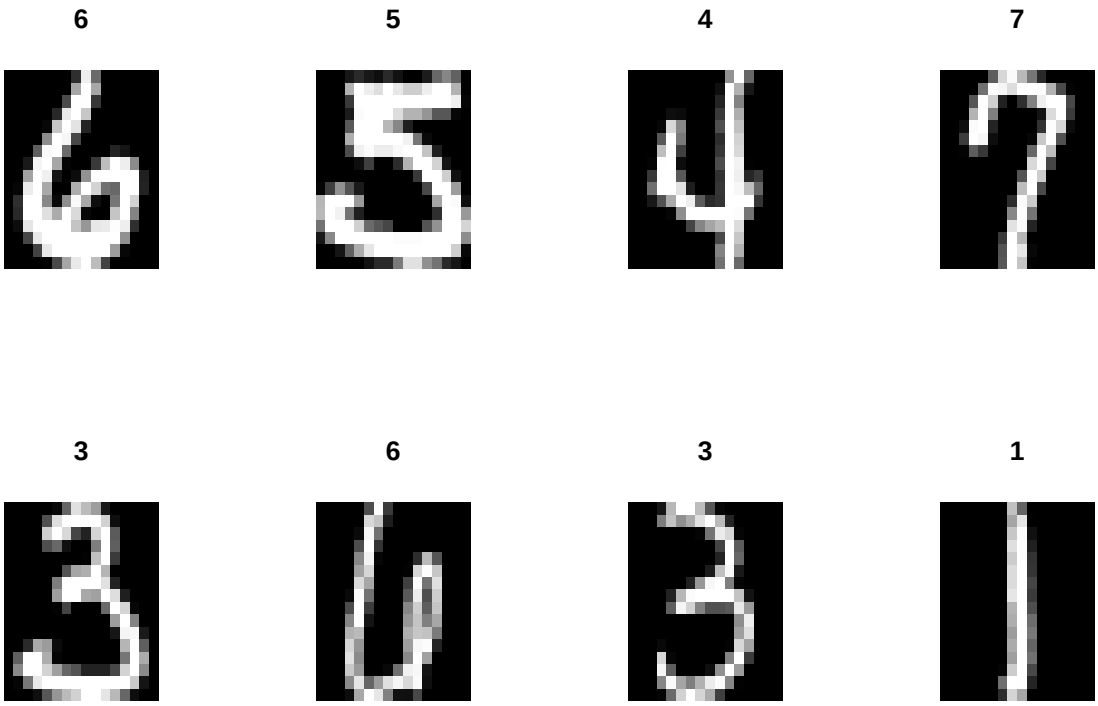
(3) Comparing the two procedures, the optimal c value is much larger in question (b) than in question (a). This is because in question (b), the prior probability of model M_S is much smaller than that of model M_F. Since the threshold c is for the classification rule: "classify as model M_S if LR > c, otherwise classify as model M_F", so a convincing LR to classify the data as from model M_S requires much stronger evidence in question (b), which means much higher LR and its corresponding optimal threshold c.

Problem B

(a) Ploting some digits as 16*16 images

```
# read data
# number of columns = 257 = 1 digit identifier (0-9) + 256 grayscale values
zipcode_train = read.table("zip.train.gz") # 7291*257
zipcode_test = read.table("zip.test.gz")   # 2007*257
```

```
# plot the first 8 digit images in the training dataset
par(mfrow=c(2,4))
for (i in 1:(2*4)) {
    label = zipcode_train[i,1]
    pixels = as.numeric(zipcode_train[i,-1])
    fig = matrix(pixels, 16, 16, byrow=TRUE)
    fig = t(apply(fig,2,rev))  # reverse the image to the right direction
    image(fig, col=gray(seq(0,1,length=16*16)), axes=FALSE, main=label)
}
```



(b) Classification by Logistic Regression & kNN

```
# subset data of digit 2 & 3
train = subset(zipcode_train, V1==2 | V1==3)
test = subset(zipcode_test, V1==2 | V1==3)
```

Logistic Regression

```r
# learn the classifier on the training data
train_logistic = train
train_logistic$V1 = ifelse(train_logistic$V1==3,1,0)  # y = I(digit=3)
logistic = glm(V1~., family=binomial, data=train_logistic)

# misclassification rate of training data
logistic.train.pred = predict(logistic, type="response")
logistic.train.pred = ifelse(logistic.train.pred>0.5,1,0)
mis_train.logistic = mean(logistic.train.pred != train_logistic$V1)
mis_train.logistic
```

```
## [1] 0
```

```r
# apply on the test data & misclassification rate of test data
test_logistic = test
test_logistic$V1 = ifelse(test_logistic$V1==3,1,0)  # y = I(digit=3)
logistic.test.pred = predict(logistic, newdata=test_logistic[,-1], type="response")
logistic.test.pred = ifelse(logistic.test.pred>0.5,1,0)
mis_test.logistic = mean(logistic.test.pred != test_logistic$V1)
mis_test.logistic
```

```
## [1] 0.05494505
```

kNN (k=1,3,5,7,15)

```r
library(class)
set.seed(123)
train.X = train[,-1]
test.X = test[,-1]
train.label = as.factor(train[,1])
test.label = as.factor(test[,1])
K = c(1,3,5,7,15)

# misclassification rate of training data
mis_train.knn = rep(NA,length(K))
for (i in 1:length(K)) {
    knn.train.pred = knn(train.X, train.X, train.label, k=K[i])
    mis_train.knn[i] = mean(knn.train.pred != train.label)
}
mis_train.knn
```

```
## [1] 0.000000000 0.005039597 0.005759539 0.006479482 0.009359251
```

```r
K[which.min(mis_train.knn)]
```

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

```r
mis_train.knn[which.min(mis_train.knn)]
```

```
## [1] 0
```

```r
# misclassification rate of test data
mis_test.knn = rep(NA,length(K))
for (i in 1:length(K)) {
    knn.test.pred = knn(train.X, test.X, train.label, k=K[i])
    mis_test.knn[i] = mean(knn.test.pred != test.label)
}
mis_test.knn
```

```
## [1] 0.02472527 0.03021978 0.03021978 0.03296703 0.03846154
```

```
K[which.min(mis_test.knn)]
```

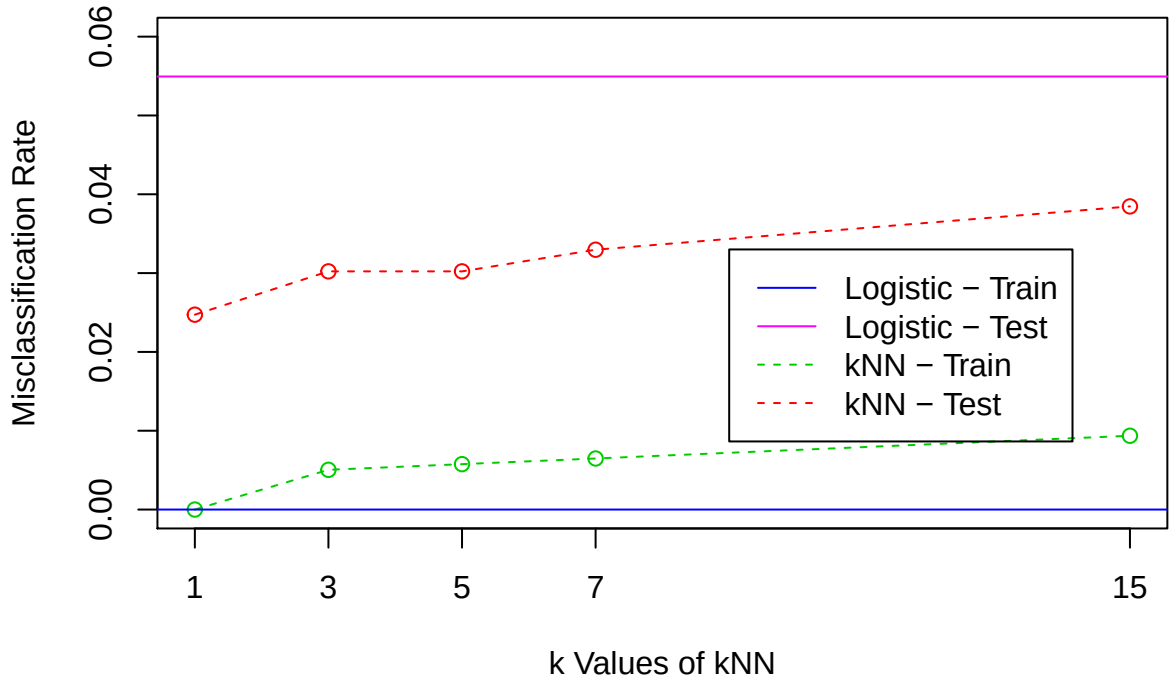
```
## [1] 1
```

```
mis_test.knn[which.min(mis_test.knn)]
```

```
## [1] 0.02472527
```

Plot misclassification rates

```
# plot misclassification rates
plot(mis_test.knn ~ K, col=2, xlim=c(1,15), ylim=c(0,0.06), xaxt='n',
     xlab="k Values of kNN", ylab="Misclassification Rate")
axis(side=1, at=c(1,3,5,7,15))
lines(mis_test.knn ~ K, lty=2, col=2)

points(mis_train.knn ~ K, col=3)
lines(mis_train.knn ~ K, lty=2, col=3)

abline(h=mis_train.logistic, col=4)
abline(h=mis_test.logistic, col=6)

legend(9,0.033,legend=c("Logistic - Train","Logistic - Test","kNN - Train","kNN - Test"),
       col=c(4,6,3,2), lty=c(1,1,2,2))
```



Comments:

(1) Logistic Regression

The misclassification rate of Logistic Regression model is 0 on the training data set, and about 0.05494505 on the test data set.

(2) kNN (k=1,3,5,7,15)

The misclassification rate of kNN model on the training data set are (0.000000000, 0.005039597, 0.005759539, 0.006479482, 0.009359251) for k=(1,3,5,7,15) respectively. And the minimal misclassification rate is 0 at k=1. As k increases, the misclassification rate also slightly increases.

The misclassification rate of kNN model on the test data set are (0.02472527, 0.03021978, 0.03021978, 0.03296703, 0.03846154) for k=(1,3,5,7,15) respectively. And the minimal misclassification rate is 0.02472527 also at k=1. As k increases, the misclassification rate also slightly increases.

(3) Comparison

We can see in the plot that the kNN models here all works better than Logistic Regression model for having lower misclassification rates on the test data set.

(c) Cross-Validation for kNN

k-flod Cross-Validation for **kNN**

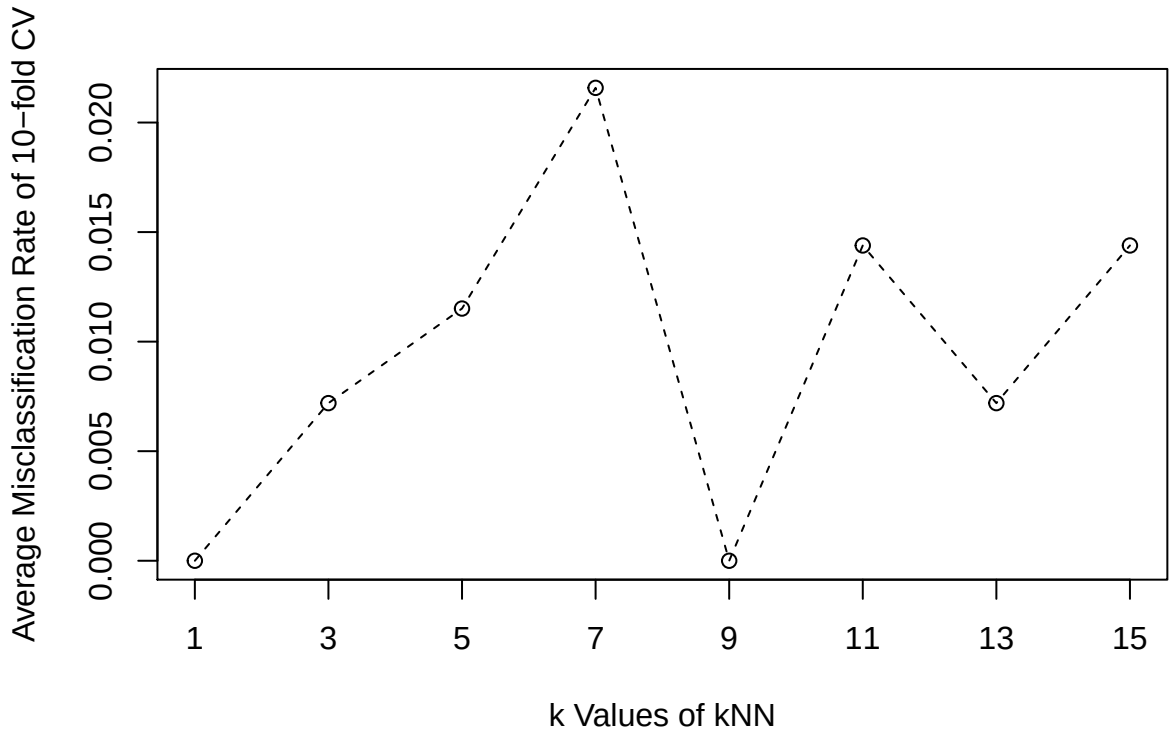
```
library(class)
K = seq(1,15,2)    # k in kNN
fold = 10          # 10-fold CV

# reshuffle training data
set.seed(123)
index = sample(nrow(train.X), nrow(train.X))
data = train[index, ]
data.X = data[,-1]
data.label = as.factor(data[,1])

# calculate Average Misclassification Rate
ave_mis_rate = rep(NA,length(K))
for (i in 1:length(K)) {
    store_mis_rate = rep(NA,fold)
    for (j in 1:fold) {
        valid_data = data.X[(1:nrow(data))%%10==(i-1), ]
        valid_label = data.label[(1:nrow(data))%%10==(i-1)]
        train_data = data.X[(1:nrow(data))%%10!=(i-1), ]
        train_label = data.label[(1:nrow(data))%%10!=(i-1)]

        knn_pred = knn(train_data, valid_data, train_label, k=K[i])
        store_mis_rate[j] = mean(knn_pred != valid_label)
    }
    ave_mis_rate[i] = mean(store_mis_rate)
}

# find the best k for kNN
K[which.min(ave_mis_rate)]
```

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

```
ave_mis_rate[which.min(ave_mis_rate)]
```

```
## [1] 0
```

```r
# plot the Average Misclassification Rate of 10-fold CV
plot(ave_mis_rate ~ K, xaxt='n',
     xlab="k Values of kNN", ylab="Average Misclassification Rate of 10-fold CV")
axis(side=1, at=K)
lines(ave_mis_rate ~ K, lty=2)
```



Leave-one-out Cross-Validation for **kNN**

```r
library(class)
K = seq(1,15,2)    # k in kNN
ave_mis_rate = rep(NA,length(K))

for (i in 1:length(K)) {
    store_mis_rate = rep(NA,nrow(train.X))
    for (j in 1:nrow(train.X)) {
        valid_data = train.X[j,]
        valid_label = train.label[j]
        train_data = train.X[-j,]
        train_label = train.label[-j]
        knn_pred = knn(train_data, valid_data, train_label, k=K[i])
        store_mis_rate[j] = mean(knn_pred != valid_label)
    }
    ave_mis_rate[i] = mean(store_mis_rate)
}

# find the best k for kNN
```

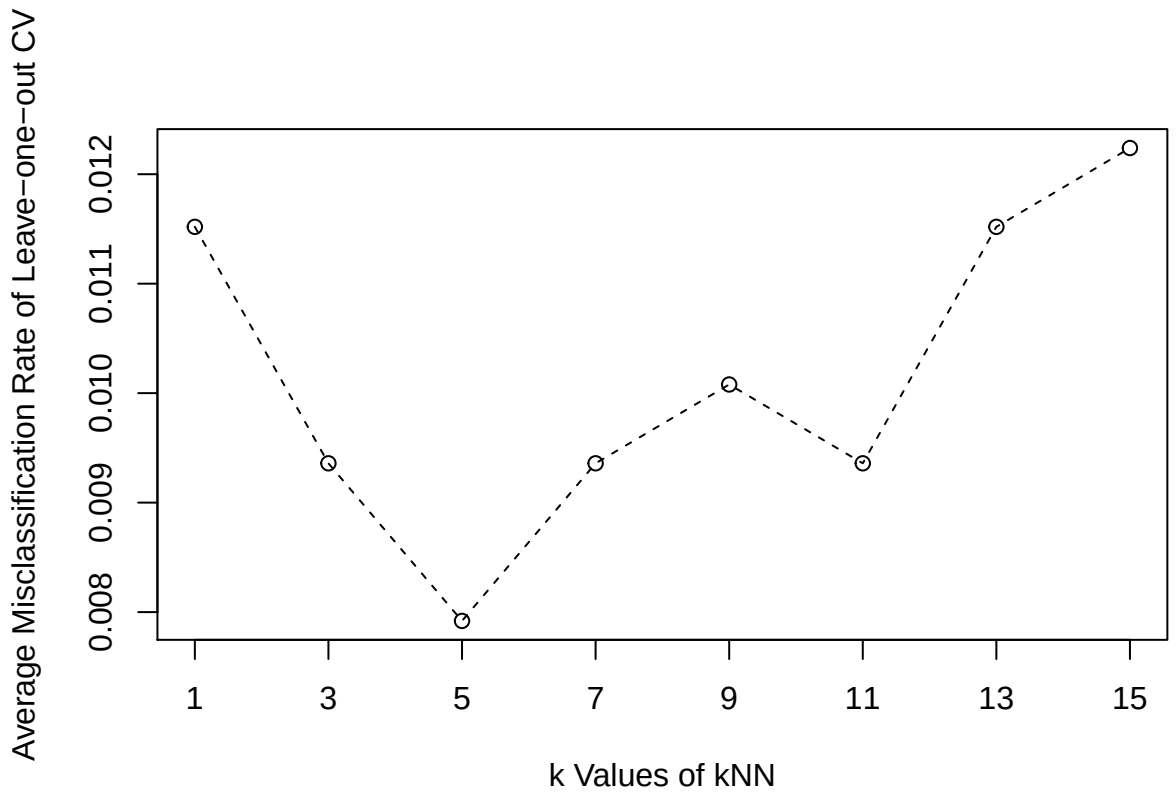
```
K[which.min(ave_mis_rate)]
```

```
## [1] 5
```

```
ave_mis_rate[which.min(ave_mis_rate)]
```

```
## [1] 0.007919366
```

```
# plot the Average Misclassification Rate of 10-fold CV
plot(ave_mis_rate ~ K, xaxt='n',
     xlab="k Values of kNN", ylab="Average Misclassification Rate of Leave-one-out CV")
axis(side=1, at=K)
lines(ave_mis_rate ~ K, lty=2)
```



Comments:

(1) Using 10-fold Cross-Validation, the misclassification rate fluctuates between 0 and 0.02, and reaches the minimum value 0 at k=1.

The method is: randomly split the training data into 10 equal-sized parts; at each time, pick one part as validation set and use the other 9 parts as training set; train the model on the training set and compute the misclassification rate on the validation set; repeat for each part thus get 10 misclassification rate and calculate its average value.

(2) Using Leave-one-out Cross-Validation, the misclassification rate fluctuates between 0.008 and 0.012, and reaches the minimum value 0.007919366 at k=5.

The method is similar with 10-fold Cross-Validation, only that every time use only one example as the validation set and all the other examples as the training set.

(3) Recall that in question (b), the kNN model works best when k=1 for both training data set and test data set. Here we got two answers from cross-validation methods, which is k=1 from 10-fold cross-validation and k=5 from leave-one-out cross-validation. It might because that for this specific data set, the k value of kNN does not make a big change, all the misclassification rate for k equals 1 to 15 are all quite small. Therefore, different methods may get different optimal k values for the kNN model.

(d) Weighted Loss Function

(1) Logistic Regression with Weighted Loss Fuction

```
train_logistic = train
test_logistic = test
weight_logistic = ifelse(train_logistic$V1==2,5,1) # 5 for digit 2; 1 for digit 3
train_logistic$V1 = ifelse(train_logistic$V1==3,1,0)  # y = I(digit=3)
test_logistic$V1 = ifelse(test_logistic$V1==3,1,0)    # y = I(digit=3)

# original classifier
logistic_ori = glm(V1~., family=binomial, data=train_logistic)

# modified classifier with weighted loss function
logistic_new = glm(V1~., family=binomial, data=train_logistic,
                   weights=weight_logistic)
```

```
# new loss function
new_loss = function(pred, true){
    loss = rep(NA, length(pred))
    for (i in 1:length(pred)) {
        if (pred[i]!=true[i] & true[i]==0){
            loss[i] = 5}
        else if (pred[i]!=true[i] & true[i]==1) {
            loss[i] = 1}
        else {
            loss[i] = 0}
    }
    print(sum(loss))
}


# compute new loss on test data for original classifier
test.pred_ori = predict(logistic_ori, newdata=test_logistic[,-1], type="response")
test.pred_ori = ifelse(test.pred_ori>0.5,1,0)
new_loss(test.pred_ori, test_logistic$V1)
```

```
## [1] 56
```

```
# compute new loss on test data for modified classifier
test.pred_mod = predict(logistic_new, newdata=test_logistic[,-1], type="response")
test.pred_mod = ifelse(test.pred_mod>0.5,1,0)
new_loss(test.pred_mod, test_logistic$V1)
```

```
## [1] 50
```

Discussion:

Method:

Here we lose 5 points every time we misclassify a 2 as a 3, but 1 point every time we misclassify a 3 as a 2.

So in the glm() function, we need to gives weight=5 to digit 2 and weight=1 to digit 3 to penalize 5 times more for misclassifying digit 3 as digit 2.

Result:

The new loss on the test data set using original Logistic classifier is 56, while the new loss on the test data set using modified weighted Logistic classifier is 50. Since the modified classifier is just based on the new loss fuction, so it will give smaller new loss result.

(2) kNN with Weighted Loss Fuction

```
library(pdist)
train.X = train[,-1]
test.X = test[,-1]
train.label = train[,1]
test.label = test[,1]
K = c(1,3,5,7,15)

# dis = pdist(test.X, train.X)
# dis[i,j] = L2 distance from (ith row of test.X) to (jth row of train.X)
#          = L2 distance from (ith obs. of test.X) to (jth obs. of train.X)


# weighted knn function
knn_weighted = function(v2, v3, k) {
    pred = rep(NA,nrow(test.X))
    for (i in 1:nrow(test.X)) {
        dis = pdist(test.X[i,], train.X)
        index = order(as.matrix(dis)[1,])[1:k]
        labels = train.label[index]
        n2 = length(labels[which(labels==2)])
        n3 = length(labels[which(labels==3)])

        # digit 2 has v2 votes; digit 3 has v3 votes
        if (v2*n2 > v3*n3) { pred[i]=2 }
        else { pred[i]=3 }
    }
    return(pred)
}


# new loss function
new_loss = function(pred, true){
    loss = rep(NA, length(pred))
    for (i in 1:length(pred)) {
        if (pred[i]!=true[i] & true[i]==2){
            loss[i] = 5}
        else if (pred[i]!=true[i] & true[i]==3) {
            loss[i] = 1}
        else {
            loss[i] = 0}
    }
    return(sum(loss))
}
```

```r
# compute new loss on test data for original classifier
new_loss_ori = rep(NA,length(K))
for (i in 1:length(K)) {
    test.pred_ori = knn_weighted(v2=1, v3=1, k=K[i]) # both digits have one vote
    new_loss_ori[i] = new_loss(test.pred_ori, test.label)
}
new_loss_ori
```

```
## [1] 33 39 39 48 58
```

```r
K[which.min(new_loss_ori)]
```

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

```r
new_loss_ori[which.min(new_loss_ori)]
```

```
## [1] 33
```

```r
# compute new loss on test data for modified classifier with weighted loss function
new_loss_mod = rep(NA,length(K))
for (i in 1:length(K)) {
    test.pred_mod = knn_weighted(v2=5, v3=1, k=K[i]) # digit 2 has 5 votes
    new_loss_mod[i] = new_loss(test.pred_mod, test.label)
}
new_loss_mod
```

```
## [1] 33 26 22 30 26
```

```r
K[which.min(new_loss_mod)]
```

```
## [1] 5
```

```r
new_loss_mod [which.min(new_loss_mod)]
```

```
## [1] 22
```

Discussion:

Method:

In the kNN algorithm, the classification is decided by the majority votes from k nearest neighbors, where each data point has equal vote. But here we lose 5 points every time we misclassify a 2 as a 3, and 1 point every time we misclassify a 3 as a 2. It is equivalent to giving digit 2 five votes and giving digit 3 only one vote, so the we are penalizing five times more for misclassifying digit 3 as digit 2.

Result:

The new loss on the test data set using original kNN classifier is (33, 39, 39, 48, 58) for k = (1,3,5,7,15) respectively. When k=1, it gives the minimum new loss, which is 33.

While the new loss on the test data set using modified weighted kNN classifier is (33, 26, 22, 30, 26) for k = (1,3,5,7,15) respectively. When k=5, it gives the minimum new loss, which is 22.

Again, since the modified classifier is just based on the new loss fuction, so it will give smaller new loss result.

Challenges:

The most challenge here is that we need to rewrite the kNN algorithm by ourselves instead of just giving weights like what I did for Logistic model. And it is a little tricky to identify the k nearest neighbors by calculating the Euclidean disctances between all the observations in the training data set and all the observations in the test data set. Luckily, the pdist() function helps me out, which computes the Lp distance (i.e. for Euclidean disctance, ues Lp=2) between rows of a matrix X and rows of another matrix Y. Then the

other part of the programs is just to follow what the kNN algorithm does and finally got a modified weighted kNN classifier.

Problem C

Run the given code first.

```
orig_data = read.table("four_salmon_pops.csv",header=TRUE,colClasses="character",sep=",")
# or        read.csv("four_salmon_pops.csv")
set.seed(100) # to ensure reproducibility

# Convert the data at each locus to a factor
# Note that we have to be careful to include all the levels from *both* columns for each locus (marker)
mylevels= function(locus){levels(factor( c(orig_data[,(1+2*locus)],
                                           orig_data[,(2+2*locus)]) ))}

# now set up four_salmon_pops
four_salmon_pops = orig_data
for(locus in 1:12){
  four_salmon_pops[,1+2*locus]= factor(four_salmon_pops[,1+2*locus],levels = mylevels(locus))
  four_salmon_pops[,2+2*locus]= factor(four_salmon_pops[,2+2*locus],levels = mylevels(locus))
}

# Randomly divide the data into a training set and a test set
nsamp = nrow(four_salmon_pops)
subset = (rbinom(nsamp,1,0.5)==1) # include each fish in subset with probability 0.5
train = four_salmon_pops[subset,]
test = four_salmon_pops[!subset,]

# this function computes a table of the alleles and their counts at a given locus (locus= 1...12)
# in a given data frame (data)
compute_counts = function(data,locus){
  return(table(data[,1+2*locus]) + table(data[,2+2*locus]))
}

# Here's an example of how this can be used
# to compute the allele frequencies (both the counts and the proportions) in the training set
trainc = list()
for(i in 1:locus){trainc[[i]]= compute_counts(train,i)} # counts

normalize= function(x){x/sum(x)}
trainf = lapply(trainc,normalize) # compute the proportions, or relative frequencies, from counts
```

(1) Estimate the allele frequencies (proportions) in the training set

```
sub.allele.f = function(salmon){
    allele.c = list()
    subdata = subset(train, Population==salmon)
    for(i in 1:12){allele.c[[i]]= compute_counts(subdata,i)}
    allele.f = lapply(allele.c, normalize)
    return(allele.f)
}
```

```
salmon1.f = sub.allele.f(salmon="EelR")
salmon2.f = sub.allele.f(salmon="FeatherHfa")
salmon3.f = sub.allele.f(salmon="FeatherHsp")
salmon4.f = sub.allele.f(salmon="KlamathRfa")
```

```
# show some examples:
# the allele frequencies at the first marker(locus) "Ogo2" in the salmon "EelR"
salmon1.f[[1]]
```

```
##
##        208        214        216        218        220        222
## 0.00000000 0.00000000 0.01538462 0.05384615 0.53846154 0.34615385
##        224        226        228        230        232        234
## 0.03076923 0.00000000 0.00000000 0.00000000 0.01538462 0.00000000
##        240        242        248        252        254
## 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
```

```
# the allele frequencies at the second marker(locus) "Ogo4" in the salmon "FeatherHsp"
salmon3.f[[2]]
```

```
##
##        132        134        136        138        140        144
## 0.22000000 0.00000000 0.52666667 0.19333333 0.00000000 0.00000000
##        150        156        158        162        164
## 0.03333333 0.00000000 0.01333333 0.01333333 0.00000000
```

(2) Compute the posterior probabilities in the test set

```
test2 = na.omit(test)
print(paste0("There are ",nrow(test)," rows in the test data set, among which ",
             nrow(test)-nrow(test2), " rows have NA missing values."))
```

```
## [1] "There are 268 rows in the test data set, among which 102 rows have NA missing values."
```

```
# compute likelihoods of each salmon subpopulation for each individual
salmon = list(salmon1.f, salmon2.f, salmon3.f, salmon4.f)
Lik = matrix(NA, nrow(test), 4)
for (i in 1:nrow(test)) {
    for (j in 1:4) {
        probs = rep(NA,12)
        for (k in 1:12) {
            freq = salmon[[j]][[k]]
            allele1 = ifelse(is.na(test[i,1+2*k]), 1, as.numeric(freq[names(freq)==test[i,1+2*k]]))
            allele2 = ifelse(is.na(test[i,2+2*k]), 1, as.numeric(freq[names(freq)==test[i,2+2*k]]))
            probs[k] = allele1 * allele2
        }
        Lik[i,j] = prod(probs)
    }
}

# compute posterior probabilites of each salmon subpopulation for each individual
indvidual.post.probs = matrix(NA, nrow(test), 4)
for (i in 1:nrow(test)) {
    for (j in 1:4) {
        indvidual.post.probs[i,j] = Lik[i,j]/sum(Lik[i,])
```

```
    }
}
indvidual.post.probs = as.data.frame(indvidual.post.probs)
colnames(indvidual.post.probs) = c("EelR","FeatherHfa","FeatherHsp","KlamathRfa")
indvidual.post.probs$Ture.Population = test$Population
```

```
# show some examples:
# posterior probabilities for some individuals (3 from each subpopulations)
indvidual.post.probs[c(1:3,70:72,140:142,210:212),]
```

```
##             EelR   FeatherHfa   FeatherHsp KlamathRfa Ture.Population
## 1   1.000000e+00 0.000000e+00 0.000000e+00  0.0000000             EelR
## 2   1.000000e+00 0.000000e+00 0.000000e+00  0.0000000             EelR
## 3            NaN          NaN          NaN        NaN             EelR
## 70  0.000000e+00 6.123403e-01 3.876597e-01  0.0000000      FeatherHfa
## 71           NaN          NaN          NaN        NaN      FeatherHfa
## 72  0.000000e+00 1.957892e-01 8.042108e-01  0.0000000      FeatherHfa
## 140 0.000000e+00 7.303107e-01 2.696893e-01  0.0000000      FeatherHsp
## 141          NaN          NaN          NaN        NaN      FeatherHsp
## 142 0.000000e+00 6.149498e-01 3.850502e-01  0.0000000      FeatherHsp
## 210 0.000000e+00 0.000000e+00 0.000000e+00  1.0000000      KlamathRfa
## 211 0.000000e+00 1.485371e-07 0.000000e+00  0.9999999      KlamathRfa
## 212 1.043801e-07 1.354416e-11 6.994333e-11  0.9999999      KlamathRfa
```

(3) Compute prediction error rate in the test set

```
# removing individuals with four NaN's
indvidual.post.probs2 = na.omit(indvidual.post.probs)

# calulate error rate
for (i in 1:nrow(indvidual.post.probs2)) {
    indvidual.post.probs2$Pred.Population[i] = names(which.max(indvidual.post.probs2[i,-5]))
}
mean(indvidual.post.probs2$Ture.Population!=indvidual.post.probs2$Pred.Population)
```

```
## [1] 0.3296703
```

(4) Comments:

Since we assume that all four populations are equally likely a priori, thus $P(S_1) = P(S_2) = P(S_3) = P(S_4)$, so the posterior probability is calculated as below:

$$P(S_i|X) = \frac{P(X|S_i)P(S_i)}{\sum_{j=1}^{4} P(X|S_j)P(S_j)} = \frac{P(X|S_i)}{\sum_{j=1}^{4} P(X|S_j)} = \frac{\prod_{k=1}^{12} p(X_{k1}|S_i)p(X_{k2}|S_i)}{\sum_{j=1}^{4}\left(\prod_{k=1}^{12} p(X_{k1}|S_j)p(X_{k2}|S_j)\right)}$$

where $P(X|S_i)$ is the likelihood under model S_i (i.e. the salmon is from subpopulation S_i), and $p(X_{k1}|S_i)$ and $p(X_{k2}|S_i)$ are the allele frequencies of the two allele types at marker locus k.

Problems:

(1) There are 268 rows in the test data set, among which 102 rows have NA missing values. Thus it is not a good idea to remove these observations. To solve this problem, I used "1" as their frequencies at the marker locus with NA values. Note that the NA's appear in pairs at both of two allele sites for each

locus, and for all the four subpopulations likelihoods $P(X|S_i)$ $(i = 1, 2, 3, 4)$ they are all used as "1", i.e. $p(X_{k1}|S_i) = p(X_{k2}|S_i) = 1$. Since $P(S_i|X) = \frac{P(X|S_i)}{\sum_{j=1}^{4} P(X|S_j)}$, as a result, this method won't affect the posterior probabilities $P(S_i|X)$ $(i = 1, 2, 3, 4)$ for comparing between different subpopulations.

(2) In the posterior probabilities output in question (2), we can see that there are many NaN values. This is because we are estimating the allele frequencies using the training data set, and using them on the test set to compute posterior probabilities. However, there are some alleles only appearing in the test set but not in the training set. Therefore, the frequencies of these "new" alleles in the test set are "0"'s for all the subpopulations, so $p(X_{k1}|S_i)p(X_{k2}|S_i) = 0$ $\forall i = 1, 2, 3, 4$, which generates zero likelihoods for all the subpopulations, i.e. $P(X|S_i) = 0$ $\forall i = 1, 2, 3, 4$. As a result, the zero denominators in the function of posterior probabiliy $P(S_i|X)$ generate NaN values.

(3) When evaluating classification error rate in question (3), I ignored these individuals with four NaN posterior probabilities. Because they have same posterior probabilites for all the four salmon subpopulations, it is not possible to assign its population class. If I randomly assign one of four populations to them with same probability, I think it may not reflect how well our trained classifier is working. So I only used the individuals without NaN values and calculated the error rate, which is about 32.97%.