Math 5365
Data Mining 1
Homework 6
Mary Barker

1. (a) This problem will apply knn to the wdbc.data data set Standardize the data and verify that the column means are equal to zero and the column standard deviations are equal to one afterwards.

   The code for this and all of the following problems is attached at the end of the document.
   For this case, each column was successfully scaled such that the mean value of all of the values in each column is 0 and the standard deviation 1.

   (b) Split the data into 70% training and 30% test data.

   As in homework 5, the data was successfully split into training and testing data.

   (c) Calculate the test error rate for predicting breast cancer diagnosis using knn with k = 3, and find a 95% confidence interval for this error rate.

   The error when using knn with k = 3 is 0.04093567. The exact confidence interval for this error rate is (0.9399628, 0.9742064).

   (d) Compare the test error rates of knn with k = 3 and rpart, and determine if there is a statistically significant difference between them.

   The error when using rpart is 0.04678363, with an exact confidence interval of (0.9317086, 0.9684996).
   Using an exact McNemar test on the two prediction models resulted in a p-value of roughly 0.003 which is below the threshold $\alpha = 0.05$. Thus there is a statistically significant difference between the two.

2. (a) Use knn.cv to estimate the error when k = 3.

   The error was computed as 0.03514938 and therefore the error rate is approximately 3.5%.

   (b) Use knn.cv to find the value of k = 1, 2, ... , 10 that minimizes the error rate. Let the optimal value be k_0.

   The optimal value k_0 is 4.

   (c) Estimate the error rate of knn with $k = k_0$ using 10-fold cross-validation.

   The average error rate using 10-fold cross-validation is 0.03342732 and therefore the average error rate is approximately 3.3%.

(d) Estimate the error rate of knn with $k = k_0$ using the bootstrap with $b = 100$.

The average error is 0.02666667, or $2.\bar{6}\%$.

3. Bonus: Write your own function for performing k-nearest neighbors classification and compare the results you obtain with knn. Here are some guidelines.

   (a) To keep things simple, you can assume there are only two class labels and k is odd, so you don't have to worry about ties. On the other hand, breaking ties isn't too hard. It might be interesting to figure out how to do it. (hint:runif).

   (b) A good starting point would be to write a function called distancematrix, whcih accepts matrices x_{train} and x_{test}. It returns a matrix D such that D_{ij} is the euclidean distance between the ith row of x_{train} and the jth row of x_{test}. Once you have the distance matrix D, finding nearest neighbors and so on should be pretty easy.

   Using my own nearest neighbors algorithm with $k = 3$ on multiple runs with as many subdivisions of the data, and also over a few hand-manufactured test cases for hands-on validation resulted in the exact same error as with the knn function every time, and the correct distribution for each of the hand-computable test cases. The exact values were to be expected, since the value of k used each time was odd, so there were no ties to break. However, in case of potential ties, there is logic in place to arbitrate the outcome in a nonbiased way. In case of a tie the program chooses the value of the class to choose based on the value of a randomly generated number.

```
# Data Mining hw 6
library(class)
library(stats)
library(exact2x2)
library(exactci)
library(rpart)

wdbc <- read.table("~/Dropbox/Tarleton/data_mining/hw05/wdbc.data",
header=FALSE,sep=",")
#1 This problem will apply knn to the wdbc.data data set
# a) Standardize the data and verify that the column means
#    are equal to zero and the column standard deviations
#    are equal to one afterwards.
wdbc <-wdbc[ ,-c(1) ]
n = nrow(wdbc)
x = seq(from=2,to=ncol(wdbc),by=1)

sdized <- standardize(wdbc, x)
apply(sdized[,x],2,sd)
apply(sdized[,x],2,mean)
```

```
# b) Split the data into 70% training and 30% test data.
splitset <- splitdata(sdized, 0.7, FALSE)
training <- splitset$traindata
testing <- splitset$testdata
train = splitset$train

# c) Calculate the test error rate for predicting breast
#    cancer diagnosis using knn with k = 3, and find a 95%
#    confidence interval for this error rate.
pred_knn = knn(train=sdized[train,x],test=sdized[-train,x],
cl=sdized$V2[train], k=3)

M_list = confmatrix(wdbc$V2[-train], pred_knn)

error_knn = M_list$error

exact_knn = binom.test(x=n - round(error_knn*n, digits=0), n=n, p=0.05)

# d) Compare the test error rates of  knn with k = 3 and
#    rpart, and determine if there is a statistically
#    significant difference between them.

tree_rpart = rpart(V2~., sdized[train,])
pred_rpart = predict(tree_rpart, newdata=sdized[-train,],type="class")
M_list = confmatrix(sdized$V2[-train], pred_rpart)
error_rpart = M_list$error

exact_rpart = binom.test(x=n-round(error_rpart*n,digits=0),n=n,p=0.05)

accv1 <- (sdized[-trainv,]$V2 == predict(tree_rpart, sdized[-trainv,], type='class'))
accv2 <- (sdized[-trainv,]$V2 == pred_knn)
mcnemartable = table(accv1, accv2)
mcnemar.exact(mcnemartable)


#2
# a) Use knn.cv to estimate the error when k = 3
V2 = sdized$V2
pred_knn_2 = knn.cv(train = sdized[,x], cl = V2, k = 3)
M_list = confmatrix(V2, pred_knn_2)
error_knn_2 = M_list$error

exact_knn_2 = binom.test(x=n-round(error_rpart*n,digits=0),n=n,p=0.05)
```

```
print(c(error_knn, error_rpart, error_knn_2))
print(c(exact_knn, exact_rpart, exact_knn_2))

# b) Use knn.cv to find the value of k = 1, 2, ... , 10 that
#    minimizes the error rate. Let the optimal value be k_0
accvect = rep(0,10)
for(k in 1:10){
  pred_knn_2 = knn.cv(train = sdized[,x], cl = V2, k = k)
  accvect[k] = confmatrix(V2, pred_knn_2)$accuracy
}
k_0 = which.max(accvect)

# c) Estimate the error rate of knn with k = k_0 using 10-fold
#    cross-validation.
k = 10
size = ceiling(n/k)
folds = sample(rep(1:k,size))
myvec = folds[1:n]
tmperror10 = 0
for(i in 1:10){
  trn <- sdized[myvec!=i,]
  tst <- sdized[myvec==i,]
  tmpknn = knn(trn[,x],tst[,x],V2[myvec!=i],k_0)

  M_list = confmatrix(V2[myvec==i], tmpknn)

  tmperror10 = tmperror10 + M_list$error
}
tmperror10 = tmperror10 / 10
print(tmperror10)

# d) Estimate the error rate of knn with k = k_0 using the
#    bootstrap with b = 100.
k = 100
size = ceiling(n/k)
folds = sample(rep(1:k,size))
myvec = folds[1:n]
tmperror100 = 0
for(i in 1:100){
  trn <- sdized[myvec!=i,]
  tst <- sdized[myvec==i,]
  tmpknn = knn(trn[,x],tst[,x],V2[myvec!=i],k_0)
```

```
  M_list = confmatrix(V2[myvec==i], tmpknn)

  tmperror100 = tmperror100 + M_list$error
}
tmperror100 = tmperror100 / 100
print(tmperror100)

# 3. Bonus

distancematrix = function(x_train, x_test){
  if(ncol(x_train) != ncol(x_test)){
    print("error in function distancematrix: incompatible matrix sizes")
  }

  d_mat <- matrix(,nrow=nrow(x_train),ncol=nrow(x_test))

  for(i in 1:nrow(x_train)){
    for(j in 1:nrow(x_test)){
      x1 = as.numeric(x_train[i,])
      x2 = as.numeric(x_test[j,])
      d = sqrt( t(x1 - x2) %*% (x1 - x2))
      d_mat[i,j] = d
    }
  }
  return(d_mat)
}

# quick hand-verifiable test for distancematrix
# a <- matrix( seq(1:6),3)
# b <- matrix( seq(from=2,to=16,by=2),4)
# d <- distancematrix(a, b)


find_closest_nbrs = function(dset, trainidx, vidx, k){
  if(k > length(trainidx)){
    print("Error in function find_closest_nbrs:k too big")
    print("... resetting k = size of training data")
    k = length(trainidx)
  }
  train <- dset[trainidx,-c(vidx)]
  test <- dset[-trainidx,-c(vidx)]

  D <- distancematrix(train, test)
  closest_vals <- matrix(,nrow(test),k)
```

```
  testvals <- rep('+', nrow(test))

  for(i in 1:nrow(test)){
    # get row numbers for the k traindata rows closest to each testdata column i
     mylist <- sort(D[,i])[1:k]
     nidx <- vector()

     for(j in 1:k){
       nidx <- c(nidx, which(D[,i] == mylist[j]) )
     }

     # then find predominating class value for that set of closest rows
     myvals <- dset[trainidx[nidx],vidx]
     t1 <- levels(dset[,vidx])[1]
     t2 <- levels(dset[,vidx])[2]
     n1 <- sum( (myvals == t1) * 1)
     n2 <- length(myvals) - n1

     if(n1 > n2){
       testvals[i] <- t1
     }else if(n2 > n1){
       testvals[i] <- t2
     } else {
       myrand <- runif(1)
       if(myrand > 0.5){
         testvals[i] <- t1
       }else{
               testvals[i] <- t2
       }
     }
   }
   print(testvals)
   mytable <- confmatrix(dset[-trainidx, vidx], testvals)
   return(mytable)
}
t <- find_closest_nbrs(wdbc, trainv, 1, 3)
tknn <- knn(wdbc[trainv,-c(1)],wdbc[-trainv,-c(1)], wdbc$V2[trainv], 3)
confmatrix(wdbc$V2[-trainv],tknn)$error
t$error

# test case that is hand-verifiable for closest_nbrs routine
M <- c("M", "M","B","M","B","B","B","M")
V2 <- c(20, 3, 4, 19, 38, 6, 15, 47)
V3 <- c(6, 8, 1, 5, 2, 3, 0, 9)
```

```
V4 <- c(150, 200, 108, 215, 198, 170, 101, 155)
d1 <- data.frame(M, V2, V3, V4)

M <- c("M", "M","B","M")
V2 <- c(10, 5, 26, 2)
V3 <- c(4, 1, 6, 5)
V4 <- c(190, 201, 105, 177)
d2 <- data.frame(M, V2, V3, V4)

total <- rbind(d1, d2)
tt1 <- 1:8
val = 1

t_test <- find_closest_nbrs(total, tt1, 1, 3)

x1 = 2:4
t_pred_knn = knn(train=total[tt1,x1],test=total[-tt1,x1],cl=total$M[tt1], k=3)
t_test_knn <-confmatrix(total[-tt1, 1], t_pred_knn)

#####################################################
# test case that is hand-verifiable for closest_nbrs routine
M <- c("M", "B","B","M","B","B","B","M","M", "B","B","M","B","B","B","M")
V2 <- c(20, 3, 4, 3.3, 38, 6, 15, 47,20, 3, 4, 3.3, 38, 6, 15, 47)
V3 <- c(6, 8, 1, 7.5, 2, 3, 0, 9,6, 8, 1, 7.5, 2, 3, 0, 9)
V4 <- c(150, 200, 108, 201, 198, 170, 101, 155,150, 200, 108, 201, 198, 170, 101, 155)
d1 <- data.frame(M, V2, V3, V4)

M <- c("M", "M","B","M","M", "M","B","M")
V2 <- c(10, 5, 26, 2, 11, 15, 46, 12)
V3 <- c(4, 1, 6, 5, 4, 1, 6, 5)
V4 <- c(190, 201, 105, 177, 186, 197, 101, 174)
d2 <- data.frame(M, V2, V3, V4)

total <- rbind(d1, d2)
tt2 <- 1:8
val = 1

t_test <- find_closest_nbrs(total, tt2, 1, 3)
```