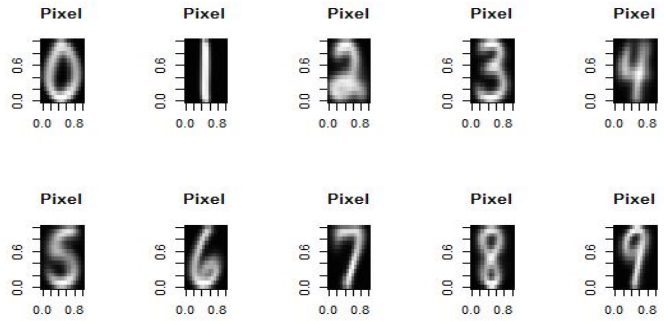
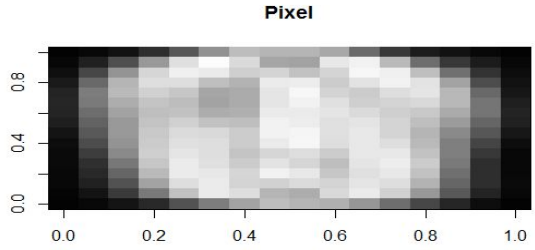
STA 141A
Final Project Report
Group #: 20 (Shivang Soni, Min Kim, Neva (Shuhua) He)

1. See CODES (#1) in APPENDIX: read_digits()
2. See CODES (#2) in APPENDIX: view_digit()
3. 
    a. Graphical Displacement of digits (0 through 9) : Below is the images of digits based the their averages



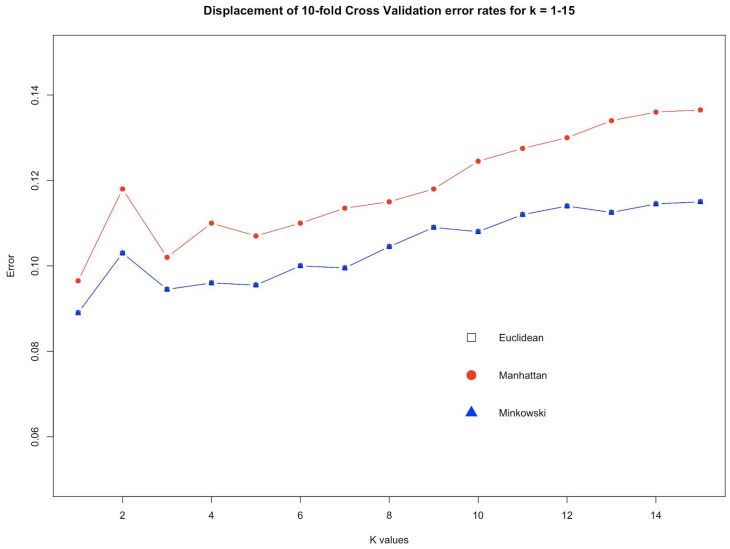

    b. Below is the picture of pixel, the darker the pixel is, less likely the pixel to be useful due to variance close 0. The variance close to 0 indicates that the sum of squared distance between an observed value and the mean of values is close to 0. Therefore, darker pixels indicate that variance close to 0 is less useful to classify different numbers of zip codes from 0 to 9. For instance, the whiter region in this picture indicates higher variance than the region of darker pixel, which is more useful to classify zip codes.




4. See CODES (#4) in APPENDIX: predict_knn()
5. See CODES (#5) in APPENDIX: cv_error_knn()
    a. Strategies used to make the function run efficiently
        i. In the function, we split the indexes instead of the entire observations into folds. The test and train data are read in a single function. So we don't have to repeatedly read the data and change the distance metric every time, and this saves execution time.

ii. For computational efficiency, using sapply() function instead of using for-loop to calculate the k-fold Cross Validation will be more efficient to solve the time-consuming computation and writing longer code.

6. 10-fold Cross Validation
   a. Displacement of 10-fold Cross Validation error rates for k = 1,...,15
      i. Below is the graph showing error rates calculated for k = 1,..., 15 by using 3 distance metrics: Euclidean, Manhattan, and Minkowski.




   b. Conclusion drawn from the results
      1. The error rates represented from the graph above indicate the proportion of number of misplaced values over total number of placements. The best combinations of k and distance metric is the one resulted lowest error rate. The table below shows the error rates calculated by using 3 distance metric.

| Metric | Error Rates (# of misplaced value / total number of placements) with respect to k values | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Euclidean | 0.089 | 0.103 | 0.09545 | 0.09600 | 0.09555 | 0.01 | 0.09555 | 0.1045 | 0.10900 | 0.10080 | 0.11200 | 0.11400 | 0.11255 | 0.1145 | 0.1150 |
| Minkowski | 0.089 | 0.103 | 0.09545 | 0.09600 | 0.09555 | 0.01 | 0.09555 | 0.1045 | 0.10900 | 0.10080 | 0.11200 | 0.11400 | 0.11255 | 0.1145 | 0.1150 |
| Manhattan | 0.09665 | 0.118 | 0.1002 | 0.111 | 0.1007 | 0.11 | 0.11335 | 0.115 | 0.118 | 0.12445 | 0.12755 | 0.13 | 0.134 | 0.136 | 0.1365 |
| The error rates are based on the training dataset. | | | | | | | | | | | | | | | |

2. According to the graph and the table, the combination of k=1 and the Euclidean/Minkowski methods give the best result (error = 0.089). The combination of k=3 and the Euclidean/Minkowski methods also give a relatively lower error rate than Manhattan method.
3. According to the graph, overall, Euclidean/Minkowski always perform better than Manhattan because they always have lower error rates given the same k.
4. According to the graph, there is a gradual increasing trend of error rates when k is increasing.

ii. Not useful to consider additional values of k
   1. Determining whether the additional values of k is useful or not depends more on the how to choose the optimum value of k over the amount of numbers of k
   2. As k increases, the error rates seem to become gradually higher. However, additional k's will result in consistent rates on a certain rate therefore, the error rates do not change dynamically and will not reduce the error rates. Hence, additional k is not useful.

7. 10-fold Cross Validation of the 3 best k and distance metric combinations to estimate confusion matrix
   a. The three confusion matrix below indicate the 3 best k and distance metric combinations:
      i. k = 1 & Euclidean/Minkowski

K: 1 Method: euclideanerr: 0.089

| predict_list \ actual_list | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 351 | 0 | 5 | 2 | 1 | 4 | 4 | 0 | 5 | 1 |
| 1 | 0 | 257 | 1 | 0 | 4 | 0 | 0 | 1 | 2 | 0 |
| 2 | 0 | 0 | 173 | 5 | 3 | 2 | 2 | 2 | 4 | 0 |
| 3 | 0 | 0 | 3 | 141 | 0 | 13 | 0 | 0 | 8 | 0 |
| 4 | 1 | 4 | 1 | 1 | 173 | 2 | 1 | 4 | 0 | 8 |
| 5 | 2 | 0 | 2 | 12 | 0 | 131 | 0 | 0 | 5 | 0 |
| 6 | 2 | 1 | 2 | 0 | 2 | 2 | 162 | 0 | 0 | 0 |
| 7 | 0 | 0 | 4 | 1 | 2 | 0 | 0 | 136 | 2 | 4 |
| 8 | 0 | 0 | 6 | 2 | 0 | 3 | 0 | 0 | 137 | 2 |
| 9 | 1 | 1 | 1 | 1 | 13 | 3 | 1 | 4 | 3 | 161 |

      ii. k = 3 & Euclidean/Minkowski

K: 3 Method: euclideanerr: 0.0945

| predict_list \ actual_list | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 351 | 1 | 13 | 4 | 2 | 11 | 3 | 0 | 8 | 1 |
| 1 | 0 | 256 | 3 | 1 | 4 | 0 | 0 | 4 | 3 | 0 |
| 2 | 0 | 0 | 167 | 3 | 4 | 1 | 2 | 1 | 4 | 0 |
| 3 | 0 | 0 | 3 | 142 | 0 | 11 | 1 | 0 | 6 | 0 |
| 4 | 1 | 4 | 2 | 1 | 176 | 1 | 1 | 4 | 0 | 7 |
| 5 | 1 | 0 | 0 | 11 | 0 | 129 | 0 | 0 | 4 | 0 |
| 6 | 2 | 1 | 2 | 0 | 2 | 1 | 162 | 0 | 1 | 0 |
| 7 | 0 | 0 | 5 | 2 | 4 | 1 | 0 | 134 | 2 | 4 |
| 8 | 1 | 0 | 3 | 1 | 0 | 2 | 0 | 1 | 132 | 2 |
| 9 | 1 | 1 | 0 | 0 | 6 | 3 | 1 | 3 | 6 | 162 |

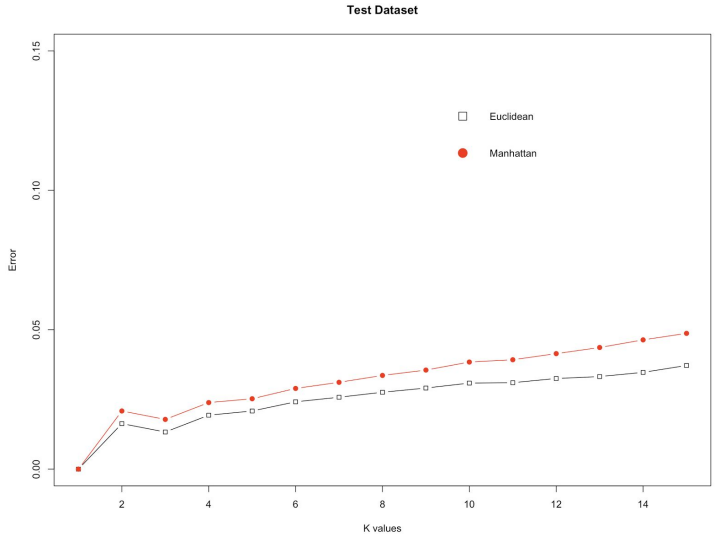
      iii. k = 1 & Manhattan

```
K: 1 Method: manhattanerr: 0.0965          actual_list
predict_list   0   1   2   3   4   5   6   7   8   9
           0 350   0   6   4   0   8   5   0   3   2
           1   0 259   4   1   5   0   1   2   3   0
           2   1   0 169   4   4   1   1   2   4   0
           3   1   0   3 140   0  14   0   0  13   0
           4   0   3   1   0 175   1   1   3   0  10
           5   1   0   2  12   0 128   0   0   4   0
           6   3   1   2   0   2   2 161   0   0   0
           7   0   0   7   2   4   0   0 136   3   6
           8   0   0   4   1   0   3   0   0 131   0
           9   1   0   0   1   8   3   1   4   5 158
```

b. Conclusion drawn from the results
    i. The best combination, which gives the lowest error rate, is still the combination of k = 1 and Euclidean, which is corresponding to the result we got from Q6.
    ii. The numbers of diagonal in the confusion matrix represents the correct frequencies. There are some error happen, which may due to the similar handwriting. For example, there are 14 of 5s were incorrectly recognized as 3s. The bias is less than 1% .
    iii. The reason that the combination of k = 1 and Euclidean came out to be best is that k = 1 is a special case where class is predicted to be the class of the closest training sample, therefore, it results in lowest error rates.
    iv. Even though the combination of k = 1 and Euclidean came out to be best, we still need to consider other optimum value of k since k = 1 generally implies overfitting of dataset.

8. Using knn as the classifier to compute the error rates and finding the best k and distance metrics combination based the error rates computed.
    a. The Best K and Distance metrics combination occurs at k = 1 & Euclidean/Minkowski , k = 3 & Euclidean/Minkowski and k = 1 & Manhattan as these have lowest error rates values evaluated using K Nearest Neighbor. Based on evaluation, the error rates computed by using different metrics are almost or more precisely exactly the same.
    b. The numbers of diagonal in the confusion matrix represents the correct frequencies. There are some error happen, which may due to the similar handwriting. For example, there are 14 of 5s were incorrectly recognized as 3s. The bias is less than 1% .

9. Test set error rates
    a. Displacement of test set error rates for k = 1,..., 15




| Metric | Error Rates (# of misplaced value / total number of placements) with respect to k values | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Euclidean | 0.00 | 0.02 | 0.017 | 0.024 | 0.0225 | 0.028 | 0.031 | 0.034 | 0.035 | 0.038 | 0.039 | 0.041 | 0.044 | 0.046 | 0.049 |
| Manhattan | 0.00 | 0.01632 | 0.01330 | 0.0193 | 0.020847 | 0.024139 | 0.025785 | 0.027 | 0.02 | 0.0308 | 0.031 | 0.032 | 0.033 | 0.035 | 0.037 |
| The error rates are based on the test dataset. | | | | | | | | | | | | | | | |

b. Comparison between results and 10-fold Cross Validation error rates
   i. In order to consider a comparison between the results and 10-fold Cross Validation error rates, the mean error rates for each distance metric of test and train dataset are calculated.
   ii. According to the graph and the table, for both Euclidean and Manhattan, the mean error rates from the training dataset came out to be smaller than the error rates from the 10-fold CV dataset. However, both datasets show an increasing trend of error rates when k is increased.
   iii. Overall, the error rates are relatively smaller in training dataset than the 10-fold Cross Validation dataset. That is because we are using the whole Training dataset in this question, which outnumbers the 10-fold Cross Validation dataset.

10. Summary of group members' contribution
    a. Min Kim
        i. has contributed to write codes for numbers 1-3
        ii. found relevant and useful resources
        iii. analysis of results
        iv. has contributed to format ,write, and edit the final report and APPENDIX
    b. Shivang Soni
        i. has contributed write codes for numbers 1-3, 6,8, 9.
        ii. found relevant and useful resources
        iii. edit & finalize code
        iv. edit final report.
        v. Contribution for analysing data in result.
    c. Neva He
        i. has contributed to write majority of predict_knn() and cv_error_knn() functions (Q4, 5, 7)
        ii. redo Q6, 8 & 9
        iii. edit & finalize code
        iv. found relevant and useful resources
        v. edit final report & analysis of results

**APPENDIX: References**


1.
How to read text file in R
https://www.datacamp.com/community/tutorials/r-data-import-tutorial#txt

How to change the column names of data frame in R

https://stackoverflow.com/questions/6081439/changing-column-names-of-a-data-frame

How to access the column of data frame in R

https://stackoverflow.com/questions/15297746/r-accessing-a-column-vector-of-a-matrix-by-name

2.

a. How to rotate the matrix in R

   https://stackoverflow.com/questions/16496210/rotate-a-matrix-in-r

3. knn function algorithm

a. http://dataaspirant.com/2017/01/02/k-nearest-neighbor-classifier-implementation-r-scratch/
b. http://dataaspirant.com/2017/01/09/knn-implementation-r-using-caret-package/
c. https://www.analyticsvidhya.com/blog/2015/08/learning-concept-knn-algorithms-programming/
d. http://www.learnbymarketing.com/tutorials/k-nearest-neighbors-in-r-example/

4. Plotting legend in R
a. https://stat.ethz.ch/R-manual/R-devel/library/graphics/html/legend.html

7. Choosing k value
https://stats.stackexchange.com/questions/107870/does-k-nn-with-k-1-always-implies-overfitting

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
9. Plotting legend in R
a. https://stat.ethz.ch/R-manual/R-devel/library/graphics/html/legend.html


See Codes in following pages

CODES

```
# 1. Function to load the dataset
read_digits = function (input = "~/Desktop/final project/digits/test.txt") {
  name = read.table(input)  ## dataframe & function to read text file
  ## Conversion to appropriate data
  colnames(name)[1] = "Zip codes"
  return(name)
}

Test <- read_digits("~/Desktop/final project/digits/Test.txt")
Train <- read_digits("~/Desktop/final project/digits/Train.txt")
############################
#2
# view the digit
view_digit <- function(s) {
  print(s[1]) #print its actual number
  s<-matrix(as.numeric(s[2:257]),16,16) #put it into a matrix
  rotate <- function(x) t(apply(x, 2, rev)) #define rotation function
  s<-rotate(s) #rotation 3 times
  s<-rotate(s)
  s<-rotate(s)
  image(t(s),col=paste("gray",1:99,sep="")) #plot it
}
view_digit(Test[1,]) #call function

############################
#3a
par(mfrow=c(2,5))
ShowMean <- function(DataName) for(i in 0:9) view_digit(sapply(DataName[which(DataName[
,1] == i),],mean)) #define function print these numbers
ShowMean(Test)

#3b
#plot variance
ShowVar <- function(DataName) view_digit(sapply(DataName,var))
ShowVar(Test)
############################
#4
Matrix_generation<- function(Test, Train, Method) as.matrix(dist(rbind(Test[,2:257],
Train[,2:257]),method = Method))
predict_knn <- function(Test, Train, K, DistMatrix) {
  #browser()  #debug tool
  EDist <- function(i,j) DistMatrix[i,j + nrow(Test)]
```

```
  PerDict <- function(i) {
    num = sapply(1:nrow(Train),function(j) Train[j,1]) #initialize reslut vector for the test
    dist = sapply(1:nrow(Train),function(j) EDist(i,j))
    temp <- data.frame(num,dist) #put into data frame for sorting and geting Kth
    temp <- table(head(temp[order(temp$dist),],K)$num)#get frequncy
    return(as.numeric(names(temp[order(temp,decreasing = TRUE)][1])))
  }
  return(sapply(1:nrow(Test),PerDict))
}

#calculate the distance matrix first
Dist <- as.matrix(dist(rbind(Test[,2:257], Train[,2:257]),method = "euclidean"))#
#define Eludian dist function
predict_knn(Test,Train,5,"euclidean",Dist)

##################################
#5
cv_error_knn <- function(Data,Matrix,k) {
  chunk <- as.integer(nrow(Data)/10)
  Fold_knn <- function(i) {
    #browser()  #debug tool
    GetJ <- function(j) ifelse(i*chunk<j & j < (i+1)*chunk,j + chunk,j)
    EDist <- function(i,j) Matrix[i,GetJ(j)]
    PerDict <- function(i) {
      num = sapply(1:(chunk*9),function(j) Data[GetJ(j),1]) #initialize reslut vector for the test
      dist = sapply(1:(chunk*9),function(j) EDist(i,j))
      temp <- data.frame(num,dist) #put into data frame for sorting and geting Kth
      temp <- table(head(temp[order(temp$dist),],k)$num)#get frequncy
      return(as.numeric(names(temp[order(temp,decreasing = TRUE)][1])))
    }
    return(sapply((i*chunk + 1) : ((i+1)*chunk),PerDict))
  }
  actual_list <- sapply(0:9,function(i) Data[(i*chunk + 1) : ((i+1)*chunk),1]) #output actual result
list
  predict_list <- sapply(0:9,Fold_knn) #output knn result
  E <- function(i) { #get the error rate of the test
    m1 <- table(predict_list[,i],actual_list[,i])
    return((sum(m1)-sum(diag(m1)))/sum(m1))
  }
  return(mean(sapply(1:10,E))) #return mean error rate
}
Dist_10 <- as.matrix(dist(rbind(Test[,2:257]),method = "euclidean")) #Calculate distance to itself
cv_error_knn(Test,Dist_10,5)
```

```
cv_error_knn(Test,Dist_10,2)

##################################
#6
par(mfrow=c(1,1))#initialize plot
err <- c() #initialize list
for(k in 1:15) {
  print(k)
  err <- c(err,cv_error_knn(Test,Dist_10,k)) #add err rate to list
}
#put into data frame with method
err = data.frame(1:15,err)
err$method = "euclidean"
names(err) = c("k","err", "method")
#plot
plot(err$err,type="b",ylab="Error",xlab="K
values",ylim=c(0.05,0.15),lty=1,lwd=1,pch=22,main="Displacement of 10-fold Cross Validation
error rates for k = 1-15")
err$err #obtain error rates

Dist_10_manhattan <- as.matrix(dist(rbind(Test[,2:257]),method = "manhattan")) #calculate
distance matrix
err1=c()#initialize list
for(k in 1:15){
  print(k)
  err1=c(err1,cv_error_knn(Test,Dist_10_manhattan,k))#add err rate to list
}
#put into data frame with method
err1 = data.frame(1:15,err1)
err1$method = "manhattan"
names(err1) = c("k","err", "method")
lines(err1$err,col="red",lty=1,lwd=1,type="b",pch=19)
err1$err #obtain error rates

Dist_10_minkowski <- as.matrix(dist(rbind(Test[,2:257]),method = "minkowski")) #calculate
distance matrix
err2=c()#initialize list
for(k in 1:15){
  print(k)
  err2=c(err2,cv_error_knn(Test,Dist_10_minkowski,k))#add err rate to list
  print(err2)
}
err2 = data.frame(1:15,err2) #put into data frame with method
```

```
err2$method = "manhattan"
names(err2) = c("k","err", "method")
#add line
lines(err2$err,col="blue",lty=1,lwd=1,type="b",pch=17)
err2$err #obtain error rates
legend("bottomright",
       legend = c("Euclidean", "Manhattan","Minkowski"),
       col = c("black",
               "red","blue"), pch=c(22,19,17),
       bty = "n",
       pt.cex = 2,
       cex = 1,
       text.col = "black",
       horiz = F ,
       inset = c(0.1, 0.1))

############################
#7 & 8
#put all k and method into combination
err = data.frame(1:15,err)
err$method = "euclidean"
names(err) = c("k","err", "method")

ERR <- rbind(err,err1,err2)
DM <- data.frame(c("euclidean",
"manhattan","minkowski"),list(Dist_10,Dist_10_manhattan,Dist_10_minkowski))
names(DM) <- c("method","matrix")
MA <- function(Data,err,euclideanMatrix,manhattanMatrix,minkowskiMatrix) {
  #browser()  #debug tool
  tablelist <- c()
  confusionMatrix <- function(Data,k,Matrix) {
    chunk <- as.integer(nrow(Data)/10)
    Fold_knn <- function(i) {
      #browser()  #debug tool
      GetJ <- function(j) ifelse(i*chunk<j & j < (i+1)*chunk,j + chunk,j)
      EDist <- function(i,j) Matrix[i,GetJ(j)]
      PerDict <- function(i) {
        #browser()  #debug tool
        num = sapply(1:(chunk*9),function(j) Data[GetJ(j),1]) #initialize reslut vector for the test
        dist = sapply(1:(chunk*9),function(j) EDist(i,j))
        temp <- data.frame(num,dist) #put into data frame for sorting and geting Kth
        temp <- table(head(temp[order(temp$dist),],k)$num)#get frequncy
        return(as.numeric(names(temp[order(temp,decreasing = TRUE)][1])))
```

```
    }
    return(sapply((i*chunk + 1) : ((i+1)*chunk),PerDict))
  }
  actual_list <- sapply(0:9,function(i) Data[(i*chunk + 1) : ((i+1)*chunk),1]) #output actual result
list
  predict_list <- sapply(0:9,Fold_knn) #output knn result
  E <- function(i) { #get the error rate of the test
    m1 <- table(predict_list[,i],actual_list[,i])
    return((sum(m1)-sum(diag(m1)))/sum(m1))
  }
  cat("err:", mean(sapply(1:10,E)))
  return(table(predict_list,actual_list)) #return mean error rate
 }
 err = err[order(err$err),] #order by err
 for(i in 1:3) {
  cat("K:", err$k[i],"Method:",err$method[i])
  if(err$method[i] == "euclidean") Matrix <- euclideanMatrix
  if(err$method[i] == "manhattan") Matrix <- manhattanMatrix
  if(err$method[i] == "minkowski") Matrix <- minkowskiMatrix
  temp <- confusionMatrix(Data,err$k[i],Matrix)
  print(temp)
  tablelist <- c(tablelist,temp)
 }
 return(tablelist)
}
#get the confusion matrix in a list of three
tablelist <- MA(Test,ERR,Dist_10,Dist_10_manhattan,Dist_10_minkowski) #also used in Q8

################################
#9. Display test set error rates for k = 1; : : : ; 15 and at least 2 different
#distance metrics in one plot.
#Compare your results to the 10-fold CV error rates.
par(mfrow=c(1,1))

Plot1to15 <- function(Test,Train,method) {
 #browser()  #debug tool
 Matrix_generation <- function(Test, Train, Method) as.matrix(dist(rbind(Test[,2:257],
Train[,2:257]),method = Method))
 DistMatrix <- Matrix_generation(Test,Train, method) #get dismatrix
 F_knn <- function() {
  #browser()  #debug tool
  EDist <- function(i,j) DistMatrix[i,j + nrow(Test)] #distance function O(1)
  PerDict <- function(i) {
```

```
    num = sapply(1:nrow(Train),function(j) Train[j,1]) #initialize reslut vector for the test
    dist = sapply(1:nrow(Train),function(j) EDist(i,j))
    temp <- data.frame(num,dist) #put into data frame for sorting and geting Kth
    temp <- temp[order(temp$dist),] #sort
    getRes <- function(k) {
      temp <- table(head(temp,k)$num)#get frequncy
      return(as.numeric(names(temp[order(temp,decreasing = TRUE)][1])))
    }
    return(sapply(1:15,getRes))
  }
  return(sapply(1:nrow(Test),PerDict))
 }
 perdict_list <- F_knn() #get result list
 actual_list <- sapply(1:nrow(Test),function(i) Test[i,1]) #get actual list
 err <- function(i) { #calculate err
  m1 <- table(perdict_list[i,],actual_list)
  return((sum(m1)-sum(diag(m1)))/sum(m1))
 }
 return(sapply(1:15,err))
}
p9 <- Plot1to15(Test,Train,"euclidean") #get error
plot(p9,type="b",ylab="Error",xlab="K values",ylim=c(0,1),lty=1,lwd=1,pch=22,main="Test
Dataset")
man <- Plot1to15(Test,Train,"manhattan")
lines(man,col="red",lty=1,lwd=1,type="b",pch=19) #graph
legend("topright",
       legend = c("Euclidean", "Manhattan","Minkowski"),
       col = c("black",
               "red","blue"), pch=c(22,19,17),
       bty = "n",
       pt.cex = 2,
       cex = 1,
       text.col = "black",
       horiz = F ,
       inset = c(0.1, 0.1))
```