




# DATA ANALYSIS PROJECT

Description: The U.S. Postal Services processes an average of 21.1 million pieces of mail per hour. Outbound mail must be sorted according to the zip code of its destination. In this project, I implemented the k-nearest neighbors algorithm for classifying handwritten digits in zip codes from R. R codes are attached in the end.



### Question 3

- a) To observe what each digit looks like on average, we subset the train dataset based on each class level. Then we took the average of columns from 2th to 257th. We ended up getting one row for each digit with 257 columns. We found the following image for each digit. The image of digit 1 is brightest compared with other images, meaning the digit has least variation.

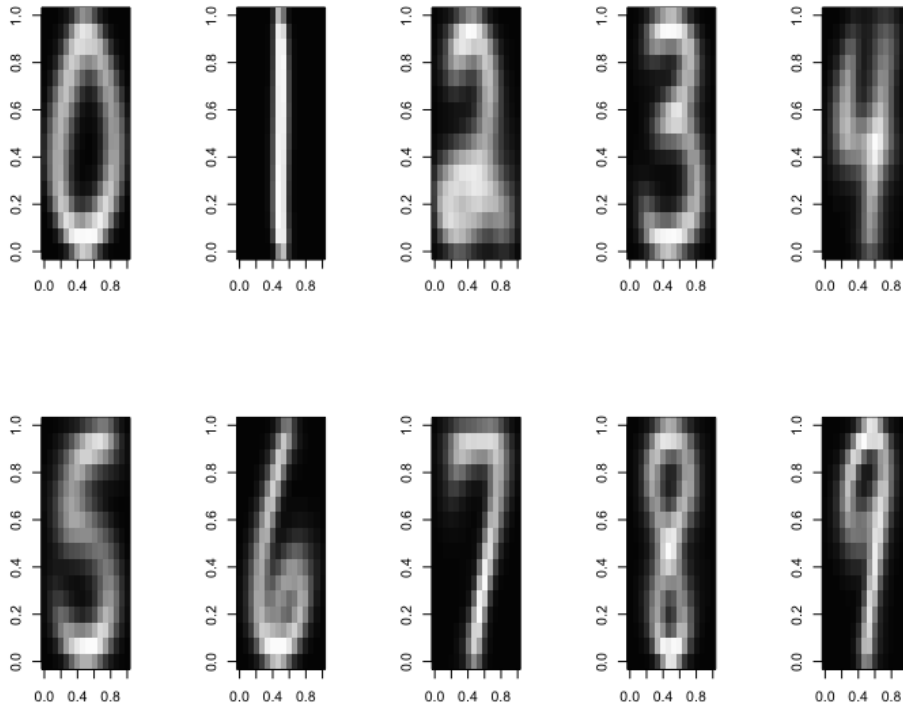


Figure 1. Images of each class level (i.e.0-9) on average.

- b) [citation: Got idea from Gupta's office hour]

We calculated variance of the mean for each column to measure the usefulness of pixel values. The reason is that for classification we want to find the different pixel values based on class labels. The lower the variance could mean the similar values among each column for each observation. We display the image of variance of each column. Each square represents the variance of each column. The darker the square means the more variation and the brighter means the less variation.

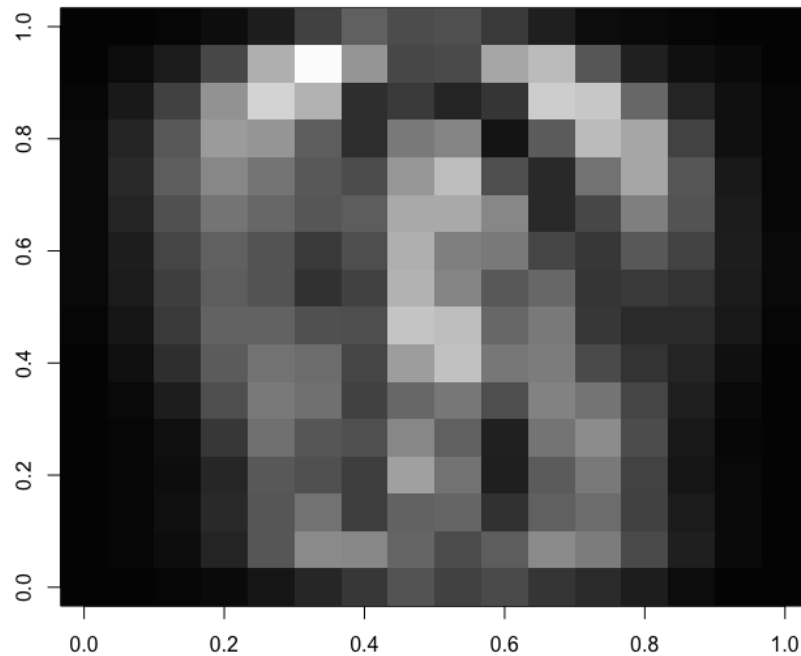


Figure 2. Variance of Mean for Each Feature.

**Question 4** is marked in the appendix.

### Question 5

In problem 5, the whole progress of cross validation, calculating `dist()` takes longer time than any other parts in this project. It usually costs about 55 seconds to 1 min to calculate one distance matrix of given train or test dataset. Thus, calculating 10 distance matrices three times to find for 15 Ks in KNN takes about 450 minutes. To include distance in any function would cause inefficiency. To avoid such inefficiency, we decide to isolate `dist()` function and calculate the distance matrix outside any loop. To do so, instead of writing one `cv_error_knn()`, we separate the processes and write several functions.

First, we write a shuffle data function which samples the train dataset and splits it into 10 folds. The observation of train is not divisible by 10, thus, one of the ten fold would contain 730 observations and the rest folds would contain 729 observations.

Second, we write a `find.subset.dist.matrix` function. The input arguments are train, test and `dis_method`. This function calculates all the distance between the train and the test set. And it subsets a matrix which has the dimension of `[1:6561, 6562:7291]` from the distance matrix. We save the new matrix into a list. Then, we obtain a list called new dist which has 10 lists. In each list, there are nine  $6561 \times 729$  matrix and one  $6561 \times 730$  matrix.

Thirds, we write a `cv_error_knn` function, in which find the new test and new train dataset, and apply `find.subset.dist.matrix` function inside the `cv_error_knn` to find the distance between new test and new train. Then, similarly, as problem 4, we find error rate for  $k$  from 1 to 15 using `lapply` and 3 number of distance methods. This return the error rate for each  $k$  for 10 different folds.

### Question 6

The three distance methods we used are Manhattan, Euclidean, and Canberra. Since we have 10 folds for each  $k$ , we took the average of error rates under each  $k$ .

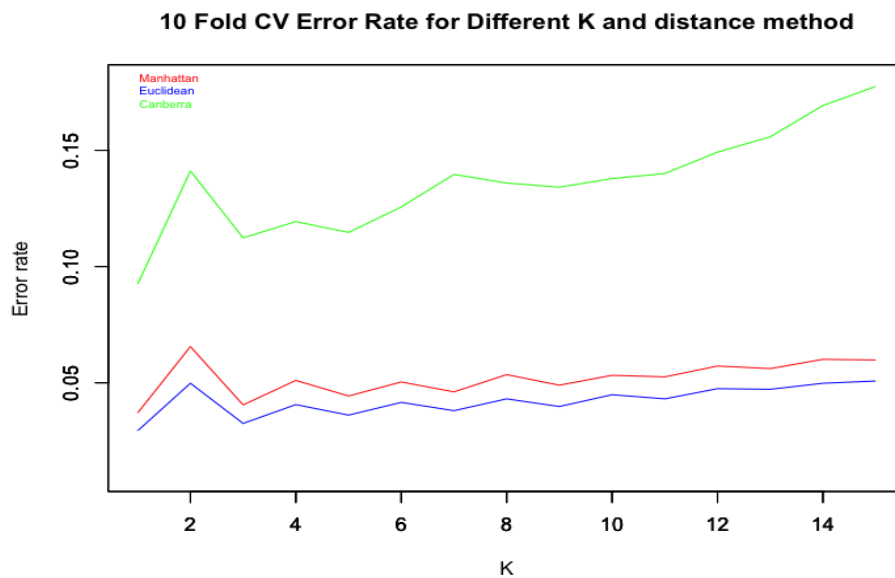


Figure 3

From Figure 3, we observed that for each  $k$ , Euclidean method shows lower error rate than Manhattan and Canberra does. Euclidean works better in our data set.

Using Euclidean method, it turns out that when  $k = 1$ , error rate is the lowest 2.935115%;  $k = 3$  has second lowest error rate: 3.250578%;  $k = 5$  has the third lowest error rate: 3.607231% (see Table 1). Therefore, the best combination is when  $k = 1$  and Euclidean method.

**Table 1.**

	Error rate
K = 1	2.935115%
K = 3	3.250578%
K = 5	3.607231%

As we can see from Figure 3, as K increase, the error rate shows an overall increasing trend and every rise in error rate is following by a smooth fall. It may or may not be useful to add more k values since it depends on what k we will choose. Large k does not necessarily decrease the error rate. For example, picking a k that is close to number of observations the maximum value k can choose will have very lower error rate; however the reason is that it almost includes every observation.

### Question 7

From question 6, we found  $k = 1, 3, 5$  with Euclidean method as the top three best combination, we construct three confusion matrices with corresponding combination. The diagonal elements represent the number of values for which the predicted label is equal to the training label. After calculating the test error rate, we find that  $k = 1$  still gives us the smallest error rate. Thus, the result is consistent with result from question 6.

#### Matrix 1

predict	0	1	2	3	4	5	6	7	8	9
0	1186	0	4	3	3	7	7	0	4	1
1	0	1002	2	0	8	0	1	3	5	0
2	5	0	703	5	2	4	0	0	1	0
3	1	0	5	634	1	12	0	0	13	1
4	0	2	0	0	615	2	0	4	1	6
5	1	0	1	9	0	519	3	1	6	0
6	1	0	1	0	2	6	652	0	4	0
7	0	1	14	0	1	1	0	633	3	8
8	0	0	1	6	1	3	1	1	505	1
9	0	0	0	1	19	2	0	3	0	627

We calculated the misclassification rate: 2.948841% and accuracy rate is 97.05116%. The values are very close to error rate we got from 10-fold cross validation.

We investigated the misclassification numbers of each digit. For example, digit 3 is most likely to be recognized as either 5 or 8; Digit 7 is misclassified 14 times as digit 2. Digit 9 is misclassified 19 times as digit 4.

#### Matrix 2.

predict	0	1	2	3	4	5	6	7	8	9
0	1183	0	3	2	0	6	8	0	4	1
1	1	1003	6	5	14	1	2	4	7	2
2	3	1	698	2	4	7	2	1	3	1
3	4	0	9	637	5	12	2	0	11	2
4	0	0	1	0	608	1	0	4	1	5
5	1	0	0	7	0	523	2	0	5	0
6	2	0	1	0	4	3	648	0	2	0
7	0	1	10	0	0	0	0	630	4	11
8	0	0	2	4	0	1	0	0	503	1
9	0	0	1	1	17	2	0	6	2	621

In Matrix 2, the misclassification rate is 3.250583% and accuracy rate is 96.74942%.

Digit 1 is mostly misclassified as Digit 4; Digit 3 is mostly misclassified as Digit 5 and 8; Digit 9 is misclassified mostly as 4.

### Matrix 3

predict	0	1	2	3	4	5	6	7	8	9
0	1182	0	8	2	2	8	13	0	4	1
1	1	1003	6	2	13	1	2	4	7	5
2	5	1	692	3	8	9	1	0	1	2
3	1	0	7	632	2	10	0	1	16	3
4	0	0	1	1	602	3	0	6	1	2
5	1	0	0	8	0	518	2	0	5	0
6	4	0	2	0	5	4	644	0	1	0
7	0	1	10	1	0	0	1	624	4	10
8	0	0	4	8	0	0	1	0	499	0
9	0	0	1	1	20	3	0	10	4	621

In Matrix 3, the misclassification rate is 3.758058% and accuracy rate is 96.24194%.

Digit 1 is misclassified as digit 4 most often; Consistent with matrix 1 and 2, digit 3 is falsely recognized as digit 5 and 8. Digit 9 is most likely to be misclassified as 4.

### Question 8:

From above questions, we choose  $k = 1$  with Euclidean distance method as our best combination. We calculated the number of each digit from the training set that is misclassified (see Table 2). For example, digit 1 is misclassified 19 times in total.

**Table 2.**

Digit	0	1	2	3	4	5	6	7	8	9
Frequency of misclassification	29	19	17	33	15	21	14	28	14	25
Rank	2	6	7	1	8	5	9	3	9	4

We find that the digit 3 has most misclassification rate. From Matrix 1, digit 3 is falsely recognized as digit 2, 5 and 8.



Here is standard writing track for 3:

- a) The shape of 3 is very similar to the 2. 3 has two radian, 2 has one radian. When people write scribbled 3, the machine may misclassify it as the 2, because if the second radian



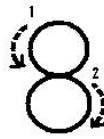
are not significant, the machine will recognize only one radian.

- b) The first and second stroke for writing 5 is very similar to 3, the second and third stroke in 5, will be read to one stroke. The first stroke in 3 is a curve, this curve is



similarly to the the second and third stroke in 5.

- c) For misclassification number 8, 8 like a combination of two 3, one of 3 is normal; another



is rotation  $180^\circ$ . And the shape in 8 is

### Question 9:

In problem 9, we display test set error rates for  $k = 1, \dots, 15$  and at three different distance methods which are euclidean, manhattan, canberra in one plot as the following:

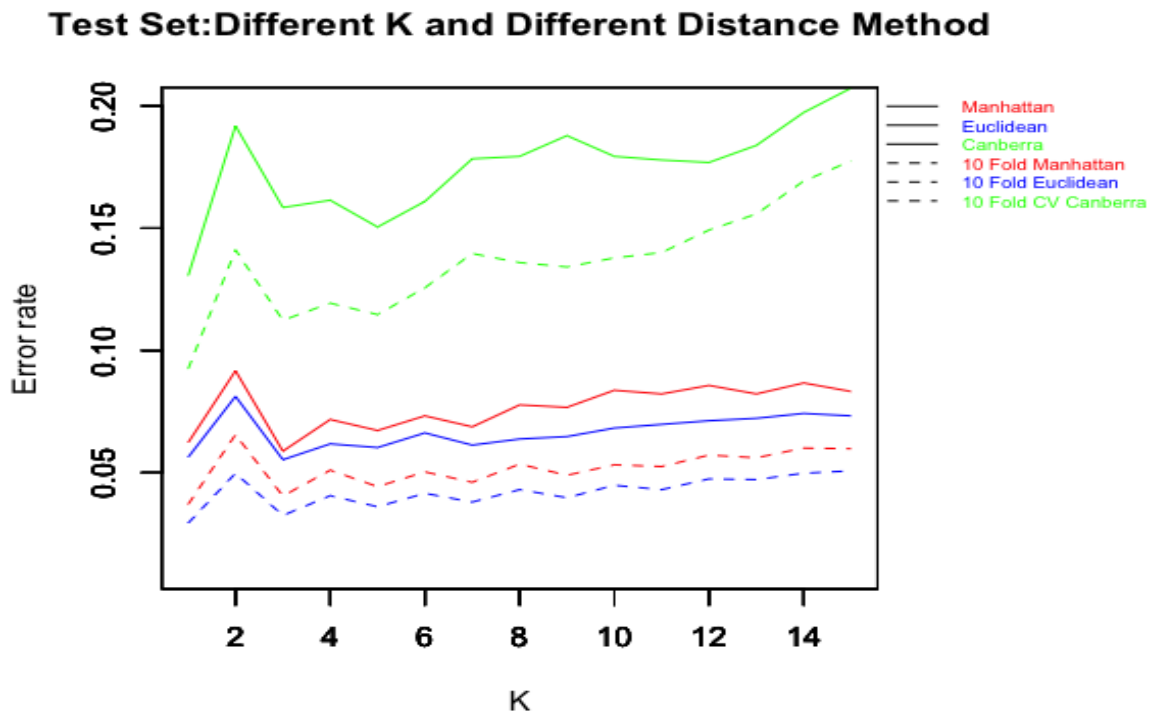


Figure 4.

In general, 10 Fold cross validation is splitting the training dataset into 10 fold randomly and predicting the class label for each fold by k nearest neighbor. Test set prediction is using training dataset to predict the label by K nearest neighbor.

Canberra distance method produce the highest error rate. Then is Manhattan distance. Euclidean produces least error rate. Euclidean distance is the best method to predict label in the test set with k nearest neighbor algorithm.

The above plot demonstrates the relationship between test set and 10 fold cross validation in 3 different distance methods. Dash line represent the error rate using 10 fold cross validation. It is clear that for each distance method, 10 fold cross validation always have lower error rate comparing with test set prediction.

Table of Test Set Error Rates and CV Error rates

	Test Set	CV Avg
Min	5.530643%	2.935115%
Max	20.72745%	17.74822%

The test set, the error rate for  $k = 1, \dots, 15$  at euclidean and manhattan distance, the range is (5.530643%, 20.72745%). In the 10 fold cross validation average error rates for  $k = 1, \dots, 15$  at euclidean and manhattan, canberra distance, the range is (2.935115%, 17.74822%). This agree with the above argument that 10 fold cross validation always have lower error rate comparing with test set prediction.



## Appendix

### ##Problem 1

#read digits into rds

library(readr)

read\_digits = function (in\_dir = "./",out\_file = "") {

  readname = function(){

    name = readline(prompt="Enter the number of the file:\n 1.test.txt\n 2.train.txt")

  }

  userchose =print(readname())

  #loop that can load different data depending on user's choice

  if (userchose == 1 ) {

    test=read.table("./test.txt")

    out\_file = "./test.rds"

  }

  else if (userchose == 2) {

    test=read.table("./train.txt")

  out\_file = "./train.rds"

  }

  else {

    print("Ops, Can't find it!")

    break

    return()

  }

  saveRDS(test, file = out\_file)

}

read\_digits()

test<-readRDS("./test.rds")

train<-readRDS("./train.rds")

### #Problem 2

view\_digit= function (obs){

  read\_observation <- function()

  {

    obs = readline(prompt="Which observation would you like to view : ")

  }

  obs =print(as.integer(read\_observation()))

  x=train[obs,2:257]

  x=data.matrix(x, rownames.force = NA)

  m=matrix(x,16,16)

#citation:<https://blog.snap.uaf.edu/2012/06/08/matrix-rotation-for-image-and-contour-plots-in-r/>

  f <- function(m) t(m)[,nrow(m):1]#function that rotate the image

```

image(f(t(m)),col=paste("gray",1:99,sep=""))
#heatmap(t(matrix(x,16,16),Rowv=NA,Colv=NA,col=paste("gray",1:99,sep="")))
}
view_digit()

```

#problem 3

#Display each digit looks like on average

#Function that takes the average of each column

```

meanfx<-function(x){
  y<-colMeans(x)
  is.vector(y)
  y<-y[-1]
  mat1<-matrix(y, nrow=16)
  f <- function(n) t(n)[,nrow(n):1]
  return(mat1)
}

```

#split into a list by 10 digits

```
splitbydigits = split(train, train[,1])
```

#compute averages for each digit

```
res = lapply(splitbydigits, meanfx)
```

#display

```

f <- function(n) t(n)[,nrow(n):1]
par(mfrow = c(2,5))
noreturn <- lapply(res, function(x){
  print(image(f(t(x)),col=paste("gray",1:99,sep="")))
})

```

#b

#To find which pixels seem the most likely to be useful for classification

#We use the average of each column from 3a.

```
res2 = lapply(res, as.vector)
```

```
res3 = do.call('rbind', res2)
```

```
colnames(res3) = NULL
```

```
rownames(res3) = 0:9
```

#Find the variance of each column

```
vars = apply(res3, 2, var)
```

```
matrix_vars = matrix(vars, 16,16)
```

#Display the image of variance of column means

```
image(matrix_vars, col = paste("gray", 1:99, sep = ""))
```

```
names(vars) = paste0("V",1:256)
```

##problem 4

```

train.subset = train[,2:257]
test.subset = test[,2:257]
predict_knn = function(train,test,k,dis_method){
  # k-nearest neighbors classification
  # train: train dataset
  # test:test dataset
  # k:number of group
  # dis_method:Distance method used in dis()
  train.label = as.data.frame(train$V1)
  colnames(train.label) = c("label")
  dist= as.matrix(dist(rbind(train,test), method = dis_method, diag = FALSE, upper = FALSE, p
= 2))
  dist.subset= dist[1:7291,7292:9298] #subset the dist matrix into 7291*2007 matrix
  subset = as.data.frame(dist.subset)
  #try function that run each column of subset
  try <- function(col){ #input a column of a data frame
    newdata = cbind(col,train.label)
    newdata = newdata[order(col)[1:k],]
    y = as.data.frame(table(newdata$label))
    #In case there is a tie in the dataset, do a sample
    winner=which(y$Freq==max(y$Freq))#Once the max freq is a tie, set these val1 as winner and
run the if loop
    if(length(winner)>1){ winner=sample(winner,1)}
    else{ winner= y[t(which.max(y$Freq))]}
    winner
  }
  a=as.data.frame(unlist(lapply(subset,try)))
  a
  #pred.test = cbind(test,a)
}

```

#Problem 5 Cross validation:

```
#dis_method="euclidean"#"manhattan"
```

```
shuffle.data = function(train){
```

```
  n.train <- nrow(train)
```

```
  p.cv <- matrix(NA,n.train,length(10))
```

```
  train$index <- 1:nrow(train)
```

```
  s <- split(sample(train$index), rep(1:10, length = n.train))#split 7291 data in to 10 folds by
sampling the index
```

```

#Find weight for each fold
#length.newtest = vector()
#for (i in 1:10) {length.newtest[i] = length(s[[i]])}
}
shuffle.data(train)
find.subset.dist.matrix = function(train,test,dis_method){
  # k-nearest neighbors classification
  # train: train dataset
  # test:test dataset
  # k:number of group
  # dis_method:Distance method used in dis()
  train.label = as.data.frame(train$V1)
  colnames(train.label) = c("label")
  train.subset = train[,2:257]
  test.subset = test[,2:257]
  dist= as.matrix(dist(rbind(train.subset,test.subset), method = dis_method, diag = FALSE, upper
= FALSE, p = 2))
  dist.subset=
dist[1:nrow(train.subset),(nrow(train.subset)+1):(nrow(train.subset)+nrow(test.subset))] #subset
the dist matrix into 7291*2007 matrix
  subset = as.data.frame(dist.subset)
}
for (i in 1:10){
  new.test.index = s[[i]]##list
  new.test = train[new.test.index,]
  new.train = train[-new.test.index,]
  new.dist[[i]]= find.subset.dist.matrix(new.train,new.test,dis_method)
}
cv_error_knn = function(train,n_folds =10,dis_method){
  for(i in 1:10){
    new.test.index = s[[i]]##list
    new.test = train[new.test.index,]
    new.train = train[-new.test.index,]
    predict[i,] = predict_knn(new.train,new.test,dis_method)
  }
}
find.subset.dist.matrix(new.train,new.test,dis_method)
cv_error_knn(train)
#problem 7

```

```

cv_error_knn = function(train,n_folds =10,dis_method){
  n.train <- nrow(train)
  p.cv <- matrix(NA,n.train,length(10))
  train$index <- 1:nrow(train)
  s <- split(sample(train$index), rep(1:10, length = n.train))
  predict = matrix(0,nrow = 10,ncol= 15)
  table_mat. = list()
  for(i in 1:10){
    new.test.index = s[[i]]##list
    new.test = train[new.test.index,]
    new.train = train[-new.test.index,]
    predict= predict_knn(new.train,new.test,dis_method)
    table_mat.e.3[[i]] = table(predict,new.test$V1)
  }
  return(table_mat.e.3)
}
#problem 8 confusion matrix
#Based on problem 7
#Problem 9
#test set error rates
dis_method = "euclidean" #or choose manhattan/canberra
true.test.error = function(train,test,dis_method){
  train.label = as.data.frame(train$V1)
  colnames(train.label) = c("label")
  train.subset = train[,2:257]
  test.subset = test[,2:257]
  dist= as.matrix(dist(rbind(train.subset,test.subset), method = dis_method, diag = FALSE, upper
= FALSE, p = 2))
  dim(dist)
  dist.subset=
dist[1:nrow(train.subset),(nrow(train.subset)+1):(nrow(train.subset)+nrow(test.subset))] #subset
the dist matrix into 7291*2007 matrix
subset = as.data.frame(dist.subset)
for (k in 1:15){
  a=as.vector(as.integer(unlist(lapply(subset,try))))
  error[k] = 1-sum(a == test$V1)/nrow(test)
}
return(error)
}
error.9.manhattan=true.test.error(train,test,"manhattan")

```

```

max(error.9.euclidean,error.9.manhattan,error.9.canberra)
min(error.9.euclidean,error.9.manhattan,error.9.canberra)
#make plot for error (for distance:manhattan,euclidean,canberra)
par(mar=c(5.1, 4.1, 4.1, 8.1), xpd=TRUE)
plot(error.9.manhattan,ylim = c(0.01,0.2),type = "l",col ="red",xlab = "K",ylab="Error
rate",main = "Test Set:Different K and Different Distance Method")
par(new=TRUE)
plot(error.9.euclidean,ylim = c(0.01,0.2),type = "l",col ="blue",xlab = "", ylab = "")
par(new=TRUE)
plot(error.9.canberra,,ylim = c(0.01,0.2),type = "l",col ="green",xlab = "", ylab = "")
par(new=TRUE)
plot(small.k.manhattan,ylim = c(0.01,0.2),type = "l",col ="red",pch=22, lty=2,xlab = "",ylab="")
par(new=TRUE)
plot(small.k.euclidean,ylim = c(0.01,0.2),type = "l",col ="blue",pch=22, lty=2,xlab = "", ylab =
"")
par(new=TRUE)
plot(small.k.canberra,ylim = c(0.01,0.2),type = "l",col ="green",pch=22, lty=2,xlab = "",ylab="")
legend("topright",inset=c(-0.5,0),legend = c("Manhattan", "Euclidean", "Canberra", "10 Fold
Manhattan", "10 Fold Euclidean", "10 Fold CV Canberra"),text.col
=c("red", "blue", "green", "red", "blue", "green"),bty = "n",cex = 0.6,lty=c(1,1,1,2,2,2))

```