STA141 Assignment III

Weitong(Jessie) Lin

November 3, 2015

Step 1

1.

- set the direction and read data

```
setwd('~/Desktop/UC Davis/141/STA141 Assignment 3')
data_image=read.csv('digitsTrain.csv', header=TRUE)
dim(data_image)
```

```
## [1] 5000  785
```

Here we can see that this data set it's a 5000 x 785 dataset

- suffle the order of the original dataset

```
# I use the set.seed() to fix a suffle data. Thus, the test later on will not change
set.seed(95616)
# randomly sample the index
change_order=sample(1:dim(data_image)[1],dim(data_image)[1],replace=FALSE)
# apply the suffling index to the dataset
data_image_new=data_image[change_order,]
data_labels=data_image_new[,1]
```

- knn function for one row dataset

This function is to apply knn rule to one row of a block of testdata.

```
#The input for this function is one block of distance matrix, lables for training set and labels for te
knn_oneBlock_oneRow=function(dist_block,k,block_train_labels,block_test_labels,rownum){
  #since the test data may be only one row, we need to use if statemetn to seperate the statement
  if (class(dist_block)=="matrix") {sort_dist=order(dist_block[rownum,])} #order the distance between o
  if (class(dist_block)!="matrix") {sort_dist=order(dist_block)}

  #Then, find the corresponsed first kth labels for the ordered training data. The result will show by
  sort_labels=lapply(k,  function(i) block_train_labels[sort_dist[1:i]])
  #count the 'labels'
  count_labels=lapply(1:length(k), function(i) table(sort_labels[[i]]))
  #we consider the label who are the majority as the predicted labels. The result will show by diff k
  pred_labels=unlist(lapply(1:length(k), function(i) names(which.max(count_labels[[i]]))))
}
```

- apply knn to one block

This function is to apply knn_oneBlock_oneRow() that we just write to every row within one block.

```
# The input for this function is one block of distance matrix, lables for training set and labels for t
knn_oneBlock=function(dist_block,block_train_labels,block_test_labels,k){
#go through all rows to knn_oneBlock_oneRow()
knn_oneblock=lapply(1:(5000/cv_fold_num), function(rownum) knn_oneBlock_oneRow(dist_block,k,block_train_
#what we get from the last part is a list, now combine the result into a data.frame
knn_oneBlock_list=do.call(rbind,knn_oneblock)
}
```

- knn for five block

This function is to apply knn_oneBlock() to all possible blocks.

```
# The input for this function is distance matrix, all data labels, number of cross validation fold and 
knn_AllBlock=function(dist_matrix,data_labels,cv_fold_num,k){
  # split index into groups
  groups=rep(1:cv_fold_num,each=floor(dim(dist_matrix)[1]/cv_fold_num))

  #pass the different block into knn_oneBlock()
  knn_each_block=function(cv_fold_num){
  # get the index for one block
  blocknum=which(groups==cv_fold_num)
  # get the distance matrix for this block
  dist_block=dist_matrix[blocknum,-blocknum]
  # get the traning set labels for this block
  block_train_labels=data_labels[-blocknum]
  # get the real test set labels for this block
  block_test_labels=data_labels[blocknum]
  # apply knn_oneBlock() for this block
  knn_allblock=knn_oneBlock(dist_block,block_train_labels,block_test_labels,k)
  }

  # go through all the blocks
  knn_Allblock=lapply(1:cv_fold_num, function(i) knn_each_block(i))
  # Here we can get a data frame with all predicted labels for all ks for all data
  knn_Allblock_list=as.data.frame(do.call(rbind,knn_Allblock))
  # we change the colnames
  colnames(knn_Allblock_list)=gsub("V", "K", colnames(knn_Allblock_list))
  # add the real labels to the data frame
  knn_Allblock_list=cbind(knn_Allblock_list,data_labels)
  # return the result
  return(knn_Allblock_list)
}
```

- apply knn to all 5 blocks

Now we will apply knn_AllBlock() to all blocks.

```
# The input for this function is the whole dataset, all data labels, number of cross validation fold an
# The output is alist which contains two data frame. One is a data frame with the predicted labels for 
My_knn=function(dataset,data_labels,cv_fold_num,k,dist_method){
```

```r
  # get the distance matrix
  dist_matrix=as.matrix(dist(dataset[,-1], method = dist_method))
  # get the data labels
  data_labels=dataset[,1]
  # get the predicted labels and real labels list for all blocks
  get_knn=knn_AllBlock(dist_matrix,data_labels,cv_fold_num,k)

  #this function will calculate misclassification_rate for each k
  misclassification_rate=function(i){
  # get the confusion_matrix
  confusion_matrix=table(get_knn$data_labels,get_knn[,i])
  # calculate the misclassification rate
  misclass_rate=1-(sum(diag(confusion_matrix))/dim(get_knn)[1])
  misclass_rate
  }
  # apply misclassification_rate() to all Ks. We will get a list
  misclassification_rate_list=lapply(1:length(k),function(i) misclassification_rate(i))
  # combine all lists into a data.frame.
  misclassification_rate_list=as.data.frame(do.call(cbind,misclassification_rate_list))
  # give the column names to misclassification_rate_list
  colnames(misclassification_rate_list)=gsub("V", "K", colnames(misclassification_rate_list))
  # return a list of two dataframe. One is a data frame with the predicted labels for diff k and the re
  newlist=list("pred_table"=get_knn,"misclassification_rate"=misclassification_rate_list)
  return (newlist)
}
```

- apply all possible distance methods Now let's apply some different distance methods. Also k and number of cv fold will be also assign. What we get is a table which reflect the error rate after cross validation for all pair k and distance methods.

```r
# some different distance methods
dist_method=c("euclidean","manhattan","maximum","canberra")
# k will be change from 1 to 30
k=1:30
# it's 5-fold cv
cv_fold_num=5
# apply all distance methods to the My_knn()
All_results=lapply(dist_method,function(dist_method) My_knn(data_image_new,data_labels,cv_fold_num,k,dis
# get all error rate after cross validation for all pair k and distance methods
error_table=lapply(1:length(dist_method),function(i) All_results[[i]]$misclassification_rate)
error_table=do.call(rbind,error_table)
# give the rownames as distance methods
rownames(error_table)=sapply(1:length(dist_method),function(i) dist_method[i])
error_table
```

```
##               K1     K2     K3     K4     K5     K6     K7     K8     K9
## euclidean 0.0644 0.0774 0.0636 0.0684 0.0708 0.0714 0.0708 0.0750 0.0758
## manhattan 0.0756 0.0918 0.0736 0.0796 0.0824 0.0862 0.0854 0.0866 0.0888
## maximum   0.3716 0.4044 0.3972 0.3876 0.3788 0.3758 0.3780 0.3806 0.3736
## canberra  0.0746 0.0788 0.0710 0.0702 0.0682 0.0666 0.0680 0.0678 0.0694
##              K10    K11    K12    K13    K14    K15    K16    K17    K18
## euclidean 0.0798 0.0780 0.0798 0.0832 0.0850 0.0840 0.0850 0.0864 0.0884
## manhattan 0.0918 0.0928 0.0956 0.0968 0.0984 0.1002 0.1010 0.1014 0.1056
## maximum   0.3798 0.3826 0.3826 0.3852 0.3822 0.3882 0.3888 0.3900 0.3914
## canberra  0.0706 0.0730 0.0724 0.0750 0.0752 0.0756 0.0758 0.0786 0.0782
##              K19    K20    K21    K22    K23    K24    K25    K26    K27
## euclidean 0.0894 0.0904 0.0920 0.0944 0.0954 0.0958 0.0962 0.0974 0.0962
## manhattan 0.1060 0.1066 0.1086 0.1106 0.1130 0.1156 0.1164 0.1174 0.1196
## maximum   0.3924 0.3964 0.3942 0.3966 0.3998 0.3982 0.4012 0.4016 0.4014
## canberra  0.0820 0.0812 0.0840 0.0860 0.0880 0.0868 0.0884 0.0882 0.0892
##              K28    K29    K30
## euclidean 0.0984 0.0982 0.1006
## manhattan 0.1210 0.1218 0.1240
## maximum   0.4038 0.3994 0.4008
## canberra  0.0900 0.0900 0.0896
```

- find the best model

```
# find the minimum error rate
min_error_position=which(error_table == min(error_table), arr.ind = TRUE)
# get the position, which can reflect the best K
best_dist=rownames(error_table)[min_error_position[1,1]]
# get the position, which can reflect the best distance methods
best_k=colnames(error_table)[min_error_position[1,2]]
# show the best model
best_model=data.frame(best_k,best_dist, min_error=min(error_table))
best_model
```

```
##   best_k best_dist min_error
## 1     K3 euclidean    0.0636
```

2.

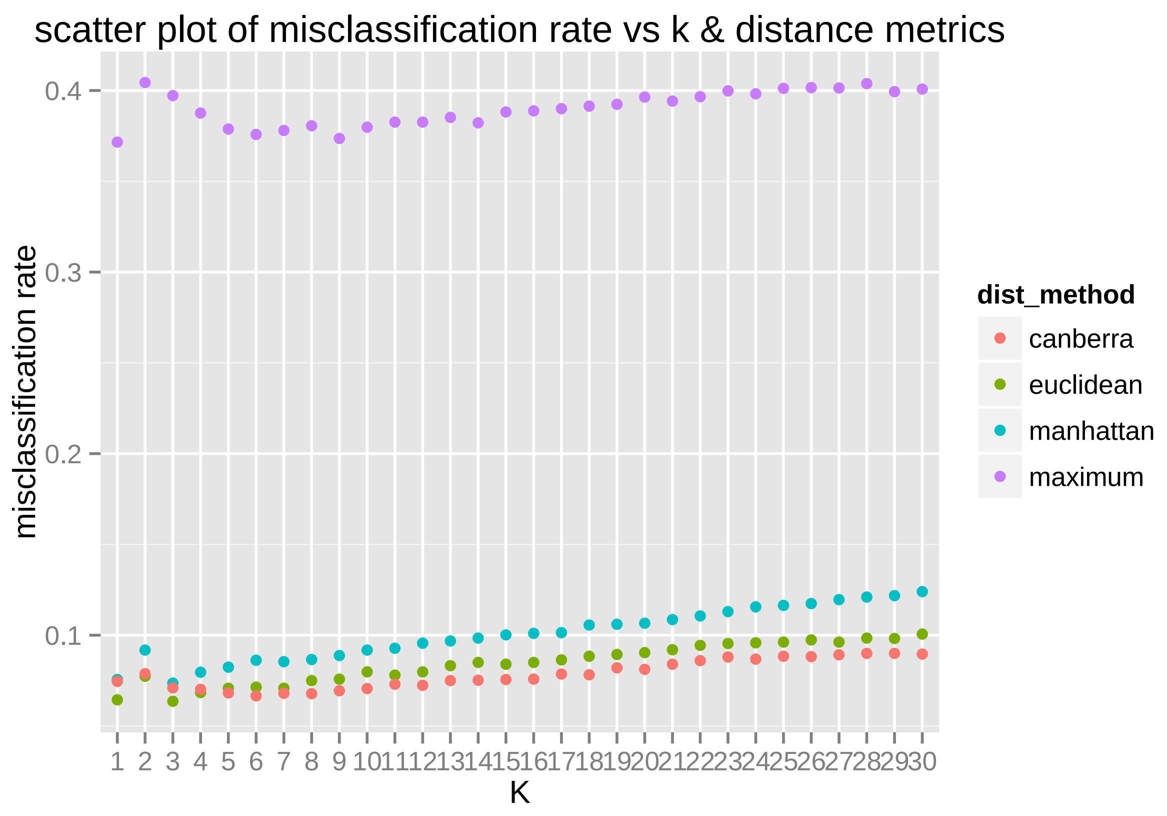
Now we use the result we get from the previous question to draw plots. First we convert the error table into a form which is convience for us to draw plot

```
new_error_table=data.frame(dist_method=rep(rownames(error_table),each=length(k)), K=as.factor(rep(1:len
#show how new form likes
head(new_error_table)
```

```
##   dist_method K error_rate
## 1   euclidean 1     0.0644
## 2   euclidean 2     0.0774
## 3   euclidean 3     0.0636
## 4   euclidean 4     0.0684
## 5   euclidean 5     0.0708
## 6   euclidean 6     0.0714
```

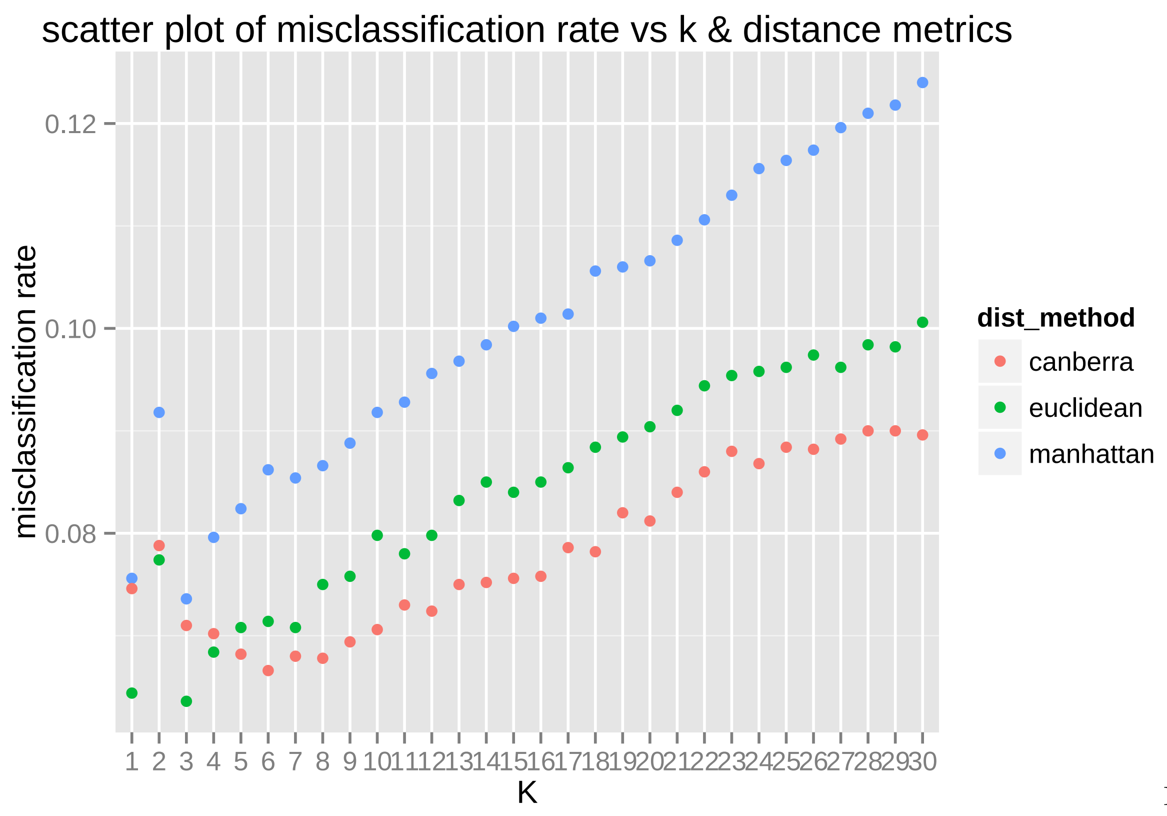
Now let's draw the plot:

```
library(ggplot2)
ggplot(new_error_table,aes(x=new_error_table$K,y=new_error_table$error_rate,col=dist_method))+
  geom_point()+
  labs(list(title = "scatter plot of misclassification rate vs k & distance metrics", x = "K", y = "mis
```



From the plot, we can see that the 'maximum' distance method has a relative high error rate. So now let drop this method and see other methods.

```
ggplot(new_error_table[-which(new_error_table$dist_method == 'maximum'),],aes(x=K,y=error_rate,col=dist_
  geom_point()+
  labs(list(title = "scatter plot of misclassification rate vs k & distance metrics", x = "K", y = "misc
```



For the general performance, the degree of good classification is canberra better than eculidean better than manhattan. However, we can clearly find that the best model is when k=3 and distance methods is 'euclidean'. #3. From the question, we've got the best model:

```
best_model
```

```
##   best_k best_dist min_error
## 1     K3 euclidean    0.0636
```

Let's get the confushion matrix for the whole dataset with k=3 and method 'euclidean' Now let's compute the confusion_matrix for the whole dataset

```
# from the best model
k=3
dist_method='euclidean'
# we see the whole data set, just like to do a test for each row
cv_fold_num=dim(data_image_new)[1]
#run  My_knn()
result_for_whole=My_knn(data_image_new,data_labels,cv_fold_num,k,dist_method)
#get the table which contains my prediction and the real labels
class_table=result_for_whole$pred_table
#get the confusion_matrix
confusion_matrix=table(class_table$data_labels,class_table[,1])
confusion_matrix
```

```
##
##       0   1   2   3   4   5   6   7   8   9
##   0 487   0   0   0   0   0   3   0   0   0
##    1    0 588   1   1   1   0   1   2   0   0
##    2    6  15 471   4   1   0   1  11   3   2
##    3    1   1   6 504   1   5   0   1   1   3
##    4    1   9   2   0 436   0   3   2   0  23
##    5    4   3   0  14   1 413   7   0   2   4
##    6    6   3   2   0   1   3 479   0   2   0
##    7    0  14   1   0   3   1   0 525   0  10
##    8    2  10   3  14   3  14   1   4 368   7
##    9    3   4   3   8   9   0   0   8   1 443
```

We can use confusion_matrix to calculate the misclassification rate:

```
1-sum(diag(confusion_matrix))/5000
```

```
## [1] 0.0572
```

which is the same result from the My_knn():

```
result_for_whole$misclassification_rate
```

```
##       K1
## 1 0.0572
```

From the misclassification_rate, we can see that knn method with k=4 and 'euclidean' distance method indeed has a good performance.

4.

Now Let's see the best/worst digit:

- split the data by different real digit

```
for_each_label=lapply(0:9,function(i) class_table[which(class_table$data_labels==i),])
```

- calculate whether we get the right prediction

```
#the right rate for each labels
count_for_each=lapply(1:10, function(i) table(for_each_label[[i]]$K1 == for_each_label[[i]]$data_labels
count_for_each=as.data.frame(do.call(cbind,count_for_each))
colnames(count_for_each)=0:9
# right rate
count_for_each
```

```
##              0        1         2         3         4        5         6
## TRUE 0.9938776 0.989899 0.9163424 0.9636711 0.9159664 0.921875 0.9657258
##              7         8         9
## TRUE 0.9476534 0.8638498 0.9248434
```

```r
# best prediction
names(which.max(count_for_each))
```

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

```r
# worst prediction
names(which.min(count_for_each))
```

```
## [1] "8"
```

From the result, we can see that '0' were generally classified best, '8' were generally classified worst.

5.

Since we already know that '8' has the worst prediction, we use '8' as an example. Here shows what the real '8' has been predict to:

```r
low_predict=class_table[which(class_table$data_labels==8),]
count_for_8=lapply(0:9,function(i) table(low_predict$K1==i)[2])
count_for_8=as.data.frame(do.call(cbind,count_for_8))
colnames(count_for_8)=0:9
count_for_8
```

```
##      0  1 2  3 4  5 6 7   8 9
## TRUE 2 10 3 14 3 14 1 4 368 7
```

Here we can find that '8' can be confused with '1','3','5','9'

6.

Again let's take '8' as example:

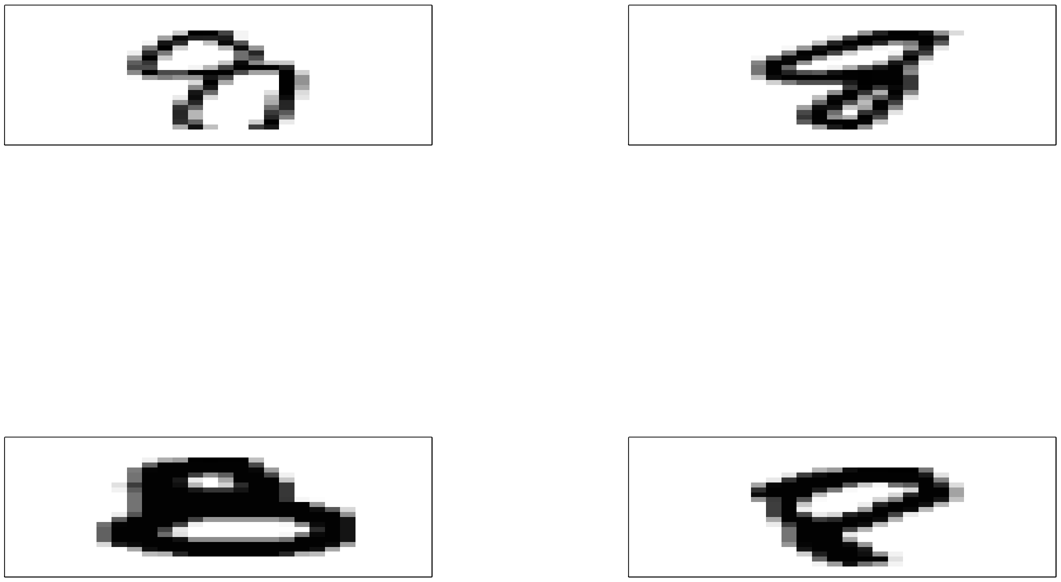
```r
# to draw the image according to the pixels
labels=data_image_new[,1]
pixels=data_image_new[,-1]
getImage =
  function(vals)
  {
    matrix(as.integer(vals), 28, 28, byrow = TRUE)
  }

draw = function(vals, colors = rgb((255:0)/255, (255:0)/255, (255:0)/255))
{
  if(!is.matrix(vals))
    vals = getImage(vals)

  m = t(vals)  # transpose the image
  m = m[,nrow(m):1]  # turn up-side-down
  image(m, col = colors, xaxt = "n", yaxt = "n")
}
```

Let's see some images of '8' which are misclassified.

```
low_not_right=low_predict[which(low_predict$K1!=8),]
# show some images that are misclassified
par(mfrow = c(2, 2))
invisible(lapply(c(3,16,22,40), function(i) draw(pixels[as.numeric(rownames(low_not_right)[i]),])))
```



From the plot, we can see that these 4 are hard to read. Now let's see how they are predicted:

```
low_not_right[c(3,16,22,40),]
```

```
##      K1 data_labels
## 162   3           8
## 1208  9           8
## 1789  3           8
## 2934  9           8
```

Now you can find that, the right two image is really quite like '9', even by human eyes. The left two are also hard to read that it's not surprise that it's recognized as '3'. Since we use pixels to detect the digit, when people write, some inks may overlap so that the pixels will be varied. Like the bottom right picture, the'o' below is narrowed that it almost changes to 'l', thus the pixels of below part of this '8' image may be very close to the pixels of below part of this '9' image. I believe that's the main reason why this '8' is misclassified as '9'.

step.2

This DTA method actually is very close to knn method. We can use 'knn_oneBlock_oneRow()'(get the closet point for one row of one block, k=1) and 'knn_oneBlock()'(apply previous function to all rows in one block) directly. The difference may happen when computing the distance matrix. The size for distance matrix changes to 5010 by 5010.

- DTA for one block First, we will get the average pixels for each image'0-9'

```
#compute the average pixels, from '0' to '9'
pixel_avg=lapply(0:9, function(i) colMeans(data_image_new[which(data_labels==i),-1]))
# combine them into a dataframe
pixel_avg=do.call(rbind,pixel_avg)
```

- DTA for five block

This function is to apply knn_oneBlock() to all possible blocks.

```
# The input for this function is distance matrix, all data labels, number of cross validation fold and 
DTA_AllBlock=function(dist_matrix,data_labels,cv_fold_num,k=1){
  # split index into groups
  groups=rep(1:cv_fold_num,each=floor((dim(dist_matrix)[1]-10)/cv_fold_num))

  #pass the different block into knn_oneBlock()
  knn_each_block=function(cv_fold_num){
  # get the index for one block
  blocknum=which(groups==cv_fold_num)
  # get the distance matrix for this block:1000 by 10
  dist_block=dist_matrix[blocknum,(dim(dist_matrix)[1]-9):dim(dist_matrix)[1]]
  # get the traning set labels for this block
  block_train_labels=0:9
  # get the real test set labels for this block
  block_test_labels=data_labels[blocknum]
  # apply knn_oneBlock() for this block
  knn_allblock=knn_oneBlock(dist_block,block_train_labels,block_test_labels,k)
  }

  # go through all the blocks
  knn_Allblock=lapply(1:cv_fold_num, function(i) knn_each_block(i))
  # Here we can get a data frame with all predicted labels for all ks for all data
  knn_Allblock_list=as.data.frame(do.call(rbind,knn_Allblock))
  # we change the colnames
  colnames(knn_Allblock_list)=gsub("V", "K", colnames(knn_Allblock_list))
  # add the real labels to the data frame
  DTA_Allblock_list=cbind(knn_Allblock_list,data_labels)
  # return the result
  return(DTA_Allblock_list)
}
```

- apply DTA to all 5 blocks

Now we will apply DTA_AllBlock() to all blocks.

```
# The inputs for this function are pixel_avg, the whole dataset, all data labels, number of cross valid
# The output is alist which contains two data frame. One is a data frame with the predicted labels for
My_DTA=function(pixel_avg,dataset,data_labels,cv_fold_num,k=1,dist_method){
  # get the distance matrix: add pixel_avg into the matrix
  datasetnew=rbind(dataset[,-1],pixel_avg)
  dist_matrix=as.matrix(dist(datasetnew, method = dist_method))
  # get the data labels
  data_labels=dataset[,1]
  # get the predicted labels and real labels list for all blocks
  get_knn=DTA_AllBlock(dist_matrix,data_labels,cv_fold_num,k)

  #this function will calculate misclassification_rate for each k
  misclassification_rate=function(i){
  # get the confusion_matrix
  confusion_matrix=table(get_knn$data_labels,get_knn[,i])
  # calculate the misclassification rate
  misclass_rate=1-(sum(diag(confusion_matrix))/dim(get_knn)[1])
  misclass_rate
  }
  # apply misclassification_rate() to all Ks. We will get a list
  misclassification_rate_list=lapply(1:length(k),function(i) misclassification_rate(i))
  # combine all lists into a data.frame.
  misclassification_rate_list=as.data.frame(do.call(cbind,misclassification_rate_list))
  # give the column names to misclassification_rate_list
  colnames(misclassification_rate_list)=gsub("V", "K", colnames(misclassification_rate_list))
  # return a list of two dataframe. One is a data frame with the predicted labels for diff k and the re
  newlist=list("pred_table"=get_knn,"misclassification_rate"=misclassification_rate_list)
  return (newlist)
}
```

- apply all possible distance methods Now let's apply some different distance methods. Also number of cv fold will be also assign. What we get is a table which reflect the error rate after cross validation for all distance methods.

```
# some different distance methods
dist_method=c("euclidean","manhattan","maximum","canberra")

# it's 5-fold cv
cv_fold_num=5
k=1
# apply all distance methods to the My_DTA()
All_results=lapply(dist_method,function(dist_method) My_DTA(pixel_avg,data_image_new,data_labels,cv_fold
# get all error rate after cross validation for all pair k and distance methods
error_table_DTA=lapply(1:length(dist_method),function(i) All_results[[i]]$misclassification_rate)
error_table_DTA=do.call(rbind,error_table_DTA)
# give the rownames as distance methods
rownames(error_table_DTA)=sapply(1:length(dist_method),function(i) dist_method[i])
```

Now we have the misclassification rate for DTA method:

```
error_table_DTA
```

```
##                K1
## euclidean 0.1848
## manhattan 0.3386
## maximum   0.3254
## canberra  0.4502
```

Here is the best model for knn method:

```
best_model
```

```
##   best_k best_dist min_error
## 1     K3 euclidean    0.0636
```

From the above result, we can find that the euclidean still holds the best classification. However, in general, DTA method is not as good as knn methods. Thus, I still recommand people can use knn method.