Man Pan

Student ID: 914656278

Email: manpan@ucdavis.edu

1 . Digits Data Exploration

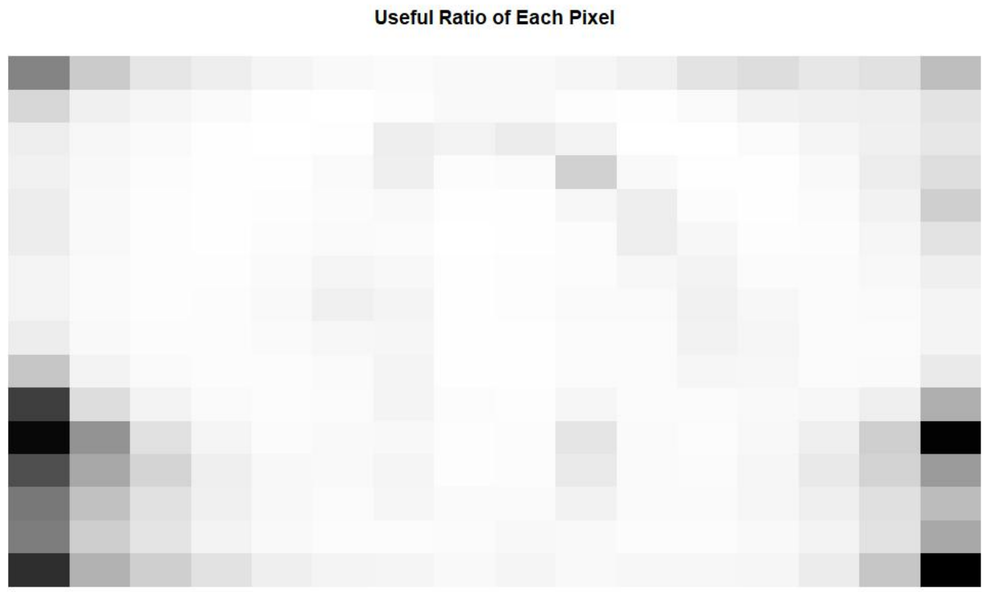
• Display graphically what each digit (0 through 9) looks like on average.

Picture 1

• Which pixels seem the most/least likely to be useful for classification?

Personally, I plan to use variance within group and variance by group to distinguish which pixels is the most/least useful for classification. I definite a variable named “a” , which is calculated by variance within group/variance by group, and I calculate “a” by each pixels. The pixel has the largest value in “a” is the least useful one, while the pixel has the smallest value in “a” is the most useful one. Finally, I find pixel V231 is the most useful, V17 is the least useful.

Then, in order to show the result much more understandable, I plot the useful ratio of each pixel. As the picture 2 shows, the write area means the least useful pixel, while the dark area means the most useful pixel.



Picture 2

2. Strategies of function cv_error_knn()

2.1 Create cv_error_knn() function

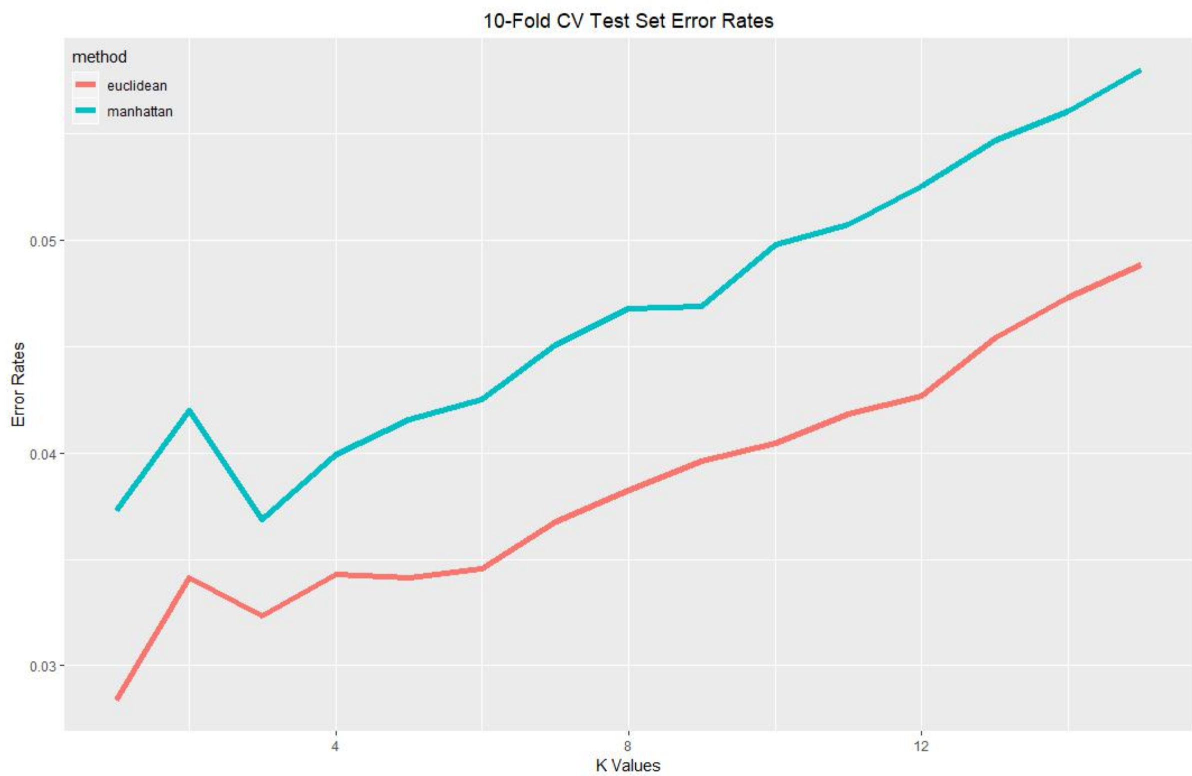
Strategies I used to make my function run efficiently:

(1) Split indexes (row numbers) into subsets rather than splitting entire observations into subsets for cross-validation;

(2) Calculating the distance matrix outside of the cv_error_knn() function, which will avoid calculating the distance in for loops.

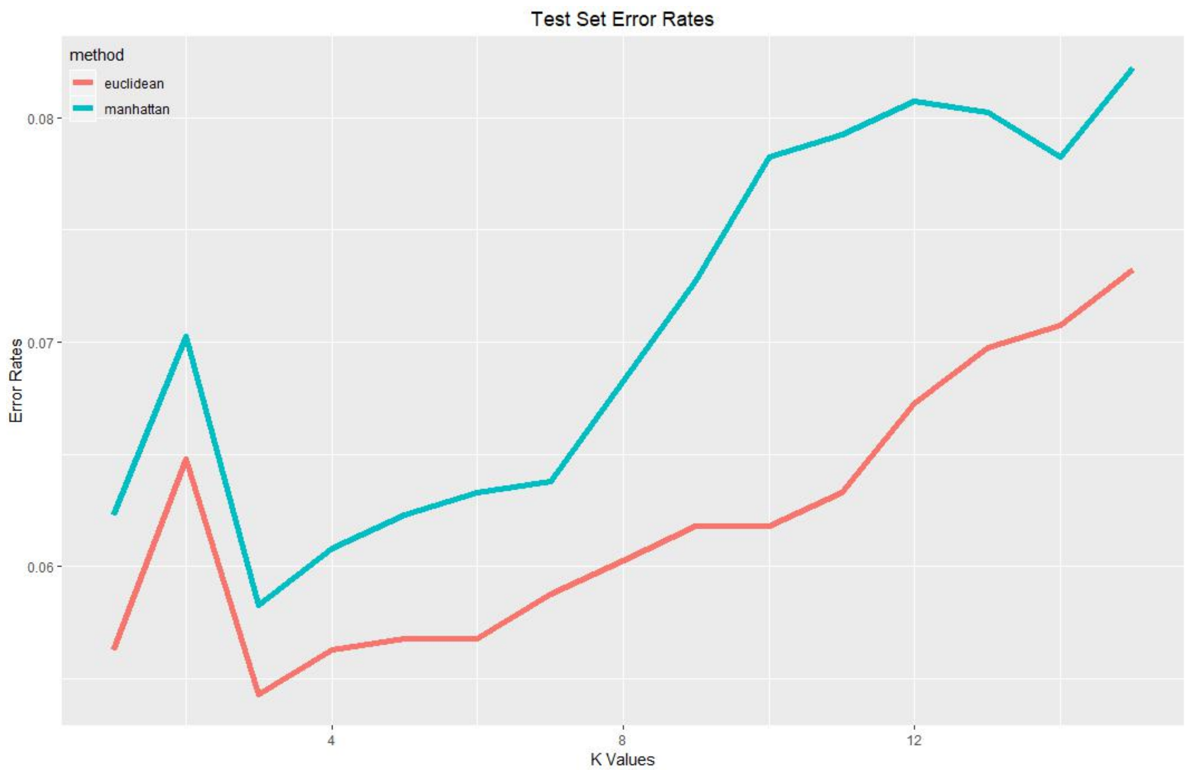
2.2 Error rates of 10 fold cross validation test set

For Euclidean distance, the error rate is lowest when k = 1. For Manhattan distance, the error rate is lowest when k = 3. Also, from picture 3, the error rates of Euclidean distance are lower compared to the error rates of Manhattan distance overall. Thus, I conclude that KNN model of Euclidean distance and k=1 is the best for this data set. And we I improve k value, the error rates will improve for most situation, expect k = 2, so it is not useful to consider additional values of k.



Picture 3

3. **Error Rates of Test Set**



Picture 4

For Euclidean distance, the error rate is lowest when k = 3. For Manhattan distance, the error rate is also lowest when k = 3. From picture 4, the error rates of Euclidean distance are lower compared to the error rates of Manhattan distance overall. Thus, for test data set, KNN model of Euclidean distance and k=3 has the lowest error rate.

Also, form picture 3 and 4, I find that 10-fold CV error rates are lower to the test set error rates overall. Because 10-fold CV error rates for Euclidean distance is about 0.03 ~ 0.05, while test set error rates for Euclidean distance is about 0.055 ~ 0.075. And 10-fold CV error rates for Manhattan distance is about 0.035 ~ 0.06, while test set error rates for Euclidean distance is about 0.06 ~ 0.08.

4. Mis - Classification of Each Digits

Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	335	0	2	0	0	0	0	1	0	1
1	0	255	0	0	6	0	2	1	0	0
2	6	1	183	2	1	0	0	2	3	0
3	3	0	2	154	0	5	0	0	0	2
4	0	3	1	0	182	1	2	2	1	8
5	2	1	2	4	0	145	2	0	3	1
6	0	0	1	0	2	3	164	0	0	0
7	0	1	1	1	4	0	0	139	0	1
8	5	0	1	6	1	1	0	1	148	3
9	0	0	1	0	2	0	0	4	1	169

Table 1

Mis-Classification Table

label	0	1	6	9	7	3	2	4	5	8
Accuracy Classification	0.011	0.034	0.035	0.045	0.054	0.072	0.076	0.090	0.094	0.108

Table 2

From the confusion matrix and mis-classification table, I find that label “1” has the highest accuracy classification while the label “8” has the lowest accuracy classification. The mis-classification rates of each digits from lowest to highest are “0”, “1”, “6”, “9”, “7”, “3”, “2”, “4”, “5”, “8”.

5. Conclusion

- I use image() to display graphically what each digit (0 through 9) looks like on average.
- I use image() to display ratio of variance within group and variance by group graphically to show Which pixels seem the most/least likely useful for classification.
- Compared to Manhattan distance, Euclidean distance is better to do classification for this data set.
- 10-fold CV error rates are lower to the test set error rates overall.
- Euclidean distance when k=1 KNN model is the “best” model for this data set.
- It is not useful to consider additional values of k, because the error rate will increase when we choose a high k value which is larger than 15.
- Label “1” has the highest accuracy classification while the label “4” has the lowest accuracy classification, when I use the “best” model to classify test data set.

Appendix

```
# Q1
set.seed(111)
read_digits = function(file_name){
  # INPUTS: file name
  # OUTPUTS: data frame
  path = 'digits/'
  temp = paste('digits/',file_name,sep="")
  data = read.table(paste(temp,'.txt',sep=""))
}
train = read_digits('train')
test = read_digits('test')
# Q2
# Display graphically what each digit (0 through 9) looks like on average
by_group = split(train,train$V1) # each digit 0 through 9

result = NULL
get_column_mean = function(group) {
  # INPUTS: each digit group
  # OUTPUTS: each column mean of each group
  for (i in (2:257)){
    temp1 = mean(group[,i])
    result = cbind(result,temp1)
  }
  return(result)
}
result = sapply(by_group,get_column_mean) #   column mean of all 10 groups
```

```
# change to 16*16 matrix and rotate
group = list()
rotate_clockwise = function(x) { t(apply(x, 2, rev))} # #rotated 90 degrees
for (i in (1:10)){
  group[[i]] = rotate_clockwise(t(matrix(result[,i],16,16)))
}

par(mfrow=c(2,5))
plot = list()
for (i in (1:10)){plot[[i]] = image(group[[i]],axes = F, col = grey(seq(1,0,length=512)))}

#Which pixels seem the most/least likely to be useful for classification?

# var_in_group
var_in_group = aggregate(train,list(train$V1),var)
var_in_group = var_in_group[,3:258]
var_in_group = apply(var_in_group,2,mean)

# var_between_group
temp1 = aggregate(train[,-1],list(train$V1),mean)
grand_mean = apply(temp1,2,mean)

var_between_group = function(by_group,grand_mean){
  # INPUTS: group , grand_mean
  # OUTPUTS: 256 variance between groups
  between_group_var = rep(0,256)
  for (j in (2:257)){
    ntotal = 0
```

```
    for (i in (1:10)){
      temp1 = mean(by_group[[i]][,j])
      temp2 = (temp1-grand_mean[j])^2
      temp3 = temp2*nrow(by_group[[i]])
      ntotal = ntotal + temp3
    }
    temp4 = ntotal/9
    between_group_var[j-1] = temp4
  }
  return (between_group_var)
}
var_between_group = var_between_group(by_group, grand_mean)


a = var_in_group/var_between_group # V231 is the most useful, V17 is the least useful
b = matrix(a,16,16)
image(b,axes = F, col = grey(seq(1,0,length=512)),main="Useful Ratio of Each Pixel")


# Q3
label = train[,1]
dataset   = train[,2:257]
test_label = test[,1]
test_dataset   = test[,2:257]


predict_KNN = function(test_dataset,k,dataset,label,dis){
  # INPUTS: predict points, k parameter,train dataset, train dataset's label, euclidean/manhattan distance
  # OUTPUTS: predict label
  sorteddisindex = apply(dis,2,order)
```

```
  temp2 = sorteddisindex[1:k,]
  if (k==1){temp4=label[temp2]}
  else {
    temp4 = rep(0,nrow(test_dataset))
    for (i in (1:ncol(temp2))){
      temp3 = table(label[temp2[,i]])[order(table(label[temp2[,i]]))]
      temp4[i] = as.numeric(names(temp3)[length(temp3)])
    }
  }
  return(temp4)
}

# Use the training set to check that your function works correctly
d_E_train = as.matrix(dist(dataset,method = "euclidean", upper = TRUE, diag =
TRUE))
predict_KNN(dataset[1:5,],2,dataset,label,d_E_train[,1:5])


# Q4
library(caret)
subsets = createFolds(1:7291, k = 10, list = TRUE, returnTrain = FALSE)
cv_error_knn = function(subsets,dataset,label,k,dis){
  # INPUTS: m-fold subsets, all train dataset, all train dataset's label, k, distance
method
  # OUTPUTS: mean error rate
  err_rate = rep(0,10)
  for (i in (1:10)){
    test_temp = dataset[subsets[[i]],]
    train_temp = dataset[-subsets[[i]],]
    get_label                                                              =
```

```
predict_KNN(test_temp,k,train_temp,label[-subsets[[i]]],dis[-subsets[[i]],subsets[[i]]])
      ori_label = label[subsets[[i]]]
      err_rate[i] = sum(get_label != ori_label)/length(ori_label)
   }
   return(mean(err_rate))
}
cv_error_knn(subsets,dataset,label,2,d_E_train)   # 0.03415003


# Q5
d_E_train = as.matrix(dist(dataset,method = "euclidean", upper = TRUE, diag = TRUE))
d_M_train = as.matrix(dist(dataset,method = "manhattan", upper = TRUE, diag = TRUE))


# euclidean distance
err_rate_cv_E = rep(0,15)
for (i in (1:15)){
   err_rate_cv_E[i] = cv_error_knn(subsets,dataset,label,i,d_E_train)
}
index = c(1:15)
method = rep("euclidean",15)
errrate_CV_E = data.frame(index,err_rate_cv_E,method)
colnames(errrate_CV_E)[2] <- "error"


err_rate_cv_E
# 0.02839134 0.03415003 0.03236677 0.03428890 0.03415210 0.03456400 0.03675879 0.03826658 0.03963927 0.04046006 0.04183143
# 0.04265447 0.04539871 0.04731840 0.04882788
```

```
# manhattan distance

err_rate_cv_M = rep(0,15)

for (i in (1:15)){

   err_rate_cv_M[i] = cv_error_knn(subsets,dataset,label,i,d_M_train)

}

err_rate_cv_M

# 0.03730881 0.04197010 0.03689352 0.03991211 0.04155990 0.04251767 0.04512549 0.04677064 0.04690838 0.04978754 0.05074927

# 0.05253141 0.05472714 0.05609738 0.05801782

index = c(1:15)

method = rep("manhattan",15)

errrate_cv_M = data.frame(index,err_rate_cv_M,method)

colnames(errrate_cv_M)[2] <- "error"

data1 = rbind(errrate_CV_E,errrate_cv_M)


library(ggplot2)

ggplot(data1,aes(x = index, y=error,color = method)) + geom_line(size = 2)+ labs(title = "10-Fold CV Test Set Error Rates", x= "K Values", y = "Error Rates")+

   theme(plot.title=element_text(hjust=0.5)) +

   theme(legend.background = element_blank(),legend.justification=c(0,1), legend.position=c(0, 1))


# Q6

predict_KNN = function(test_dataset,k,dataset,label, dis){

   # INPUTS: predict points, k parameter,train dataset, train dataset's label, euclidean/manhattan distance

   # OUTPUTS: predict label

   distance = dis[(nrow(test_dataset)+1):nrow(dis),1:nrow(test_dataset)]

   sorteddisindex = apply(distance,2,order)
```

```
  temp2 = sorteddisindex[1:k,]

  if (k==1){temp4=label[temp2]}
  else {
    temp4 = rep(0,nrow(test_dataset))
    for (i in (1:ncol(temp2))){
      temp3 = table(label[temp2[,i]])[order(table(label[temp2[,i]]))]
      temp4[i] = as.numeric(names(temp3)[length(temp3)])
    }
  }
  return(temp4)
}

d_E = as.matrix(dist(rbind(test_dataset, dataset),method = "euclidean", upper = TRUE, diag = TRUE))
d_M = as.matrix(dist(rbind(test_dataset, dataset),method = "manhattan", upper = TRUE, diag = TRUE))

err3 = function(test_dataset,k,dataset,label,test_label,dis){
  # INPUTS: predict points, k, train dataset, train label, test label, distance method
  # OUTPUTS: error rate estimator
  get_label = predict_KNN(test_dataset,k,dataset,label,dis)
  ori_label = test_label
  err = sum(get_label != ori_label)/length(ori_label)
  return(err)
}
#####   euclidean distance
err_rate3 = rep(0,15)
for (i in (1:15)){
```

```
  err_rate3[i] = err3(test_dataset,i,dataset,label,test_label,d_E)
}
err_rate3
#  0.05630294  0.06477329  0.05430992  0.05630294  0.05680120  0.05680120
0.05879422  0.06028899  0.06178376  0.06178376  0.06327853  0.06726457
0.06975585 0.07075237
# 0.07324365
index = c(1:15)
method = rep("euclidean",15)
err_rate_3 = data.frame(index,err_rate3,method)
colnames(err_rate_3)[2] = "error"

########## manhattan distance
err_rate4 = rep(0,15)
for (i in (1:15)){
  err_rate4[i] = err3(test_dataset,i,dataset,label,test_label,d_M)
}
err_rate4
#  0.06228201  0.07025411  0.05829596  0.06078724  0.06228201  0.06327853
0.06377678  0.06826109  0.07274539  0.07822621  0.07922272  0.08071749
0.08021923 0.07822621
# 0.08221226
#
index = c(1:15)
method = rep("manhattan",15)
err_rate_4 = data.frame(index,err_rate4,method)
colnames(err_rate_4)[2] = "error"
data2 = rbind(err_rate_3,err_rate_4)
ggplot(data2,aes(x = index, y=error, color = method)) + geom_line(size = 2)+
```

```
  labs(title = "Test Set Error Rates", x= "K Values", y = "Error Rates")+
  theme(plot.title=element_text(hjust=0.5))+
  theme(legend.background = element_blank(),legend.justification=c(0,1),
legend.position=c(0, 1))

# confusion matrix
get_label = predict_KNN(test_dataset,1,dataset,label, d_E)
ori_label = test_label
tb = as.matrix(table(ori_label, get_label))

# mis-classification for each label
mistake = ((rowSums(tb)) - diag(tb))/rowSums(tb)
sort(round(mistake,3))
```