Report 6

Jie Gu

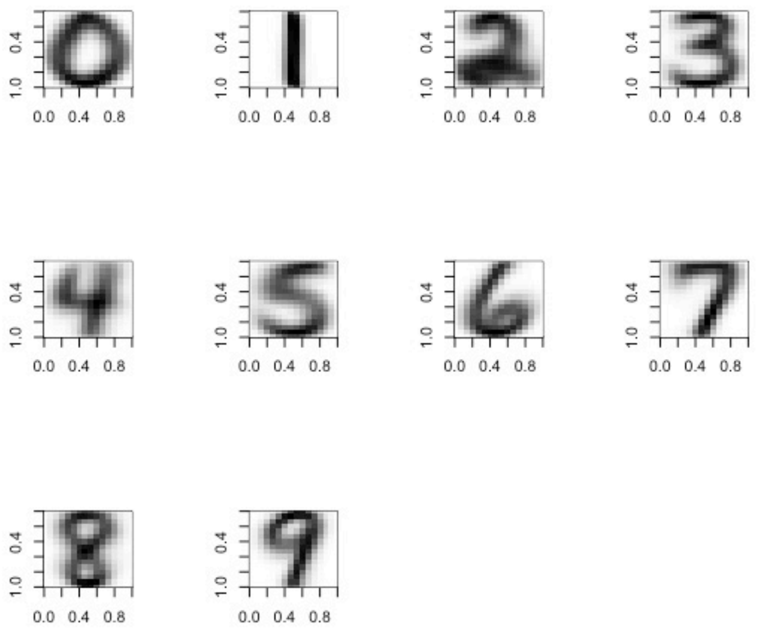
SID: 913953707

Introduction

In this report, I will investigate the digits data set which can show images of scanned zip code digits. For each observation of the data set, the first entry is the class label for the digit (0–9) and the remaining 256 entries are the pixel values in the 16 × 16 grayscale image of the digit. I will also use k-nearest neighbors to predict the label for a point or collection of points and do cross validation to estimate the error rate. Now, let's do some basic investigation on the data set.

Explore the Data Set

First of all, I will show you what each digit looks like on average by graph.



From the graph, we can see that for these digits, some black part and white part appear in the similar part of each digits. For instance, some white parts appear on the margin of each digits. Since these parts appear in the similar location of each digits, their pixel values tend to be similar. Moreover, they may be the least likely to be useful for classification since the color reflected by these pixels tend to be similar. By contrast, if some images reflected by pixels highly vary in each digit, they are more likely to reflect what the digits are. For these images, their pixels values are quite different. Therefore, we can use the variance of pixels to measure their ability for classification.

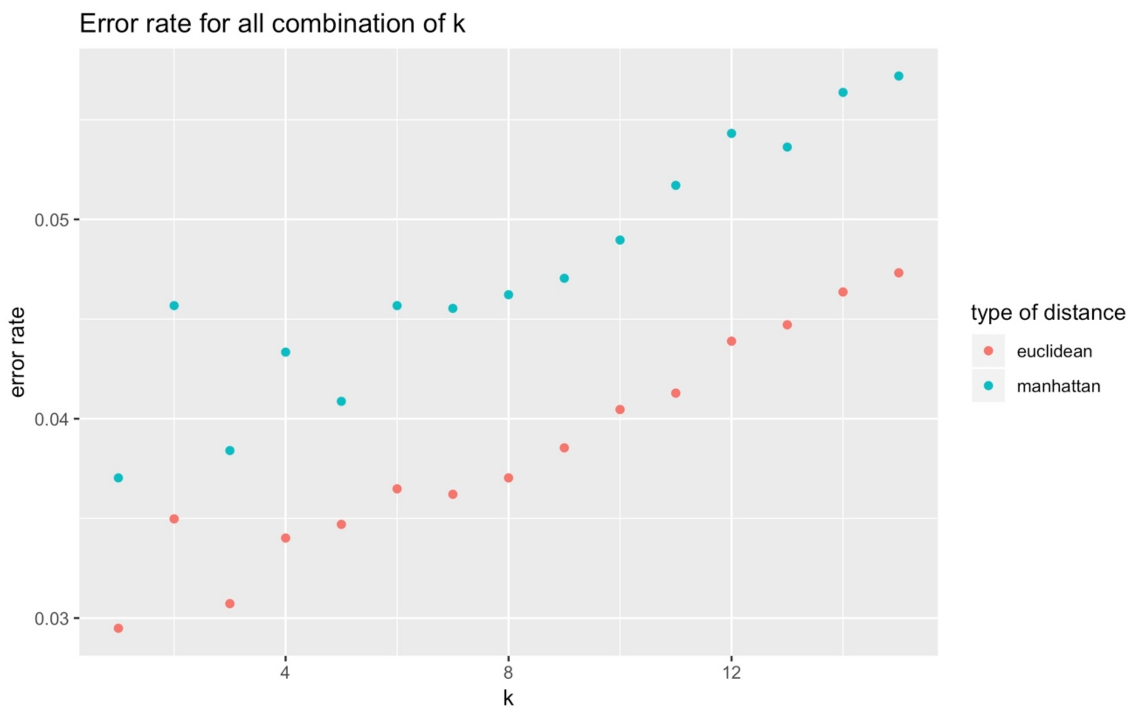
Hence, the 230th, 219th, 105th, 121st and 185th pixels seem the most likely to be useful for classification since their variances are the largest and the images reflected by them are quite different in each digit. These images are likely to determine the shape of digits. The 241st, 1st,256th,16th and 17th pixels seem the least likely to be useful for classification since their variances are the smallest and the images reflected by them are similar in each digit.

Strategies for 10-fold Cross-Validation

To make the function run efficiently, I calculate the distance from train to train at first, and then make it as an input. Therefore, for each fold, I only need to draw the distance matrixes of the fold to the rest of the folds from the entire distance matrix. This saves much time for the function to run. Otherwise, we will calculate the distance matrixes of each fold to the rest of the folds ten times, which cost much time. Also, Since I will call the predict_knn() function in the loop, I use the apply function in predict_knn() to order the distance matrix instead of a loop, by which I can also save time.

10-Fold CV Error Rates

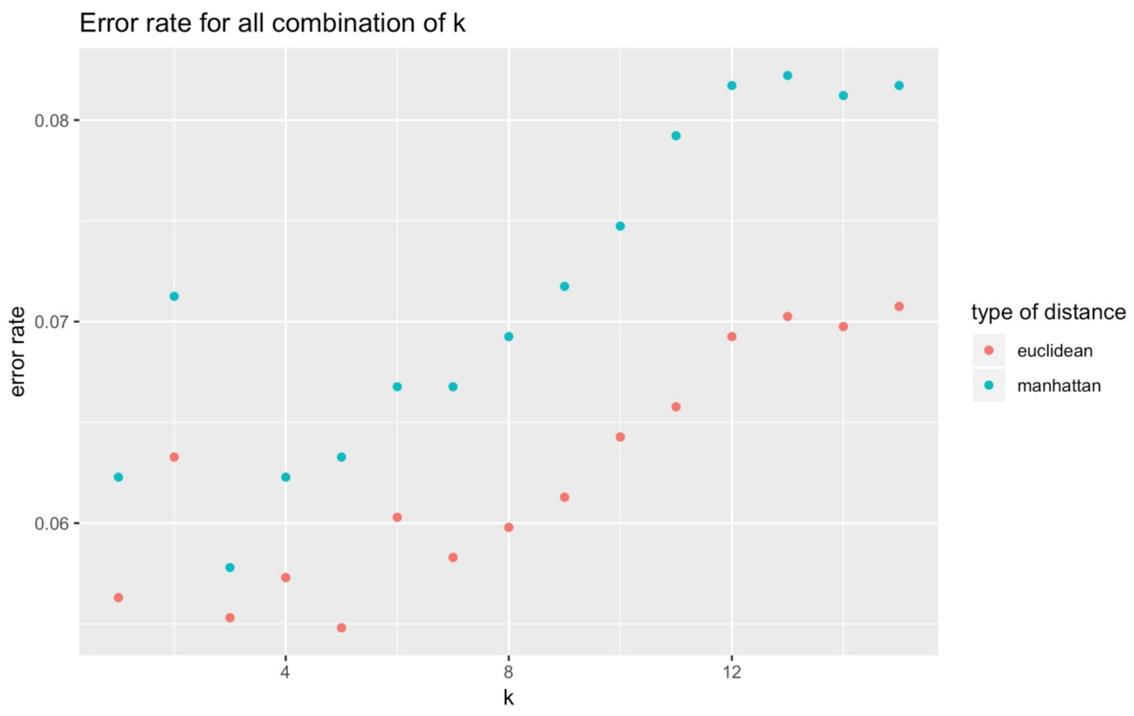
In this part, I do 10-fold CV error rates for all combinations of k from 1 to 15 and two different distance metrics: Euclidean distance metric and Manhattan distance metric. The following is a plot of the 10-fold CV error rate for each k and metric.



In this plot, we can see that the general trends for each type of distance metrices are similar: with the increase of k, the error rate decreases first and attain a minimum, and then the error rate keeps increasing. In general, the Euclidean distance metric performs better than Manhattan distance metric, since for the same k, the error rate of Euclidean distance metric is always smaller than Manhattan distance metric. We can see that the best combination working best for this data set is k = 1 and Euclidean distance metric. Also, when k equals 3, the Euclidean distance metric also performs very well. It would not be useful to consider additional values of k. The reason is that usually the CV error rate decrease at first and then keeps increasing. In my plot, the CV error has kept increasing after k = 3, so even we consider additional value of k, the error rate will keeps increasing for those k in general.

Test Set Error Rates

In this part, I use the train data set to predict the test data set with k-nearest neighbors. More precisely, the value of k is from 1 to 15, and I also use two distance metrics: Euclidean distance metric and Manhattan distance metric. In addition, I calculate the test set error rate for all the combination. Now, let's see how the error changes from a plot.



From the plot, we can see that the general trend for the test set error rate is similar to the 10-fold CV error rates. More precisely, with the increase of k, the test error rate also decreases firstly for both of the distance metric, and then the test error rate keeps increasing for both of the distance metric. Additionally, the Euclidean distance metric performs better than Manhattan distance metric, which is similar to the situation in 10-fold CV error rates. However, in general, the test error rate is higher when compared with the 10-fold CV error rate.

Considering on both plots, the best model seems to be the model with k = 3 and

Euclidean distance metric. The reason is that the error rate of k = 3 seems to be a local minimum attained in 10-fold CV error rate plot, and the test error rate of k = 3 also performs very well and can be considered as a local minimum. By counting the amount of each digit that are got wrong by the best model, 4 and 5 are most easily to be got wrong. Also, 0 and 1 are less likely to be got wrong. Compared with 0 and 1, the shape of 4 and 8 are more complex. So, we can guess that maybe the best models tend to get some digits with more complex shapes wrong.

Appendix

```
library(ggplot2)



#1.Write a function read_digits() that reads a digits file
into R.
#=====================================================

read_digits <- function(file){
  pixels = read.table(file)
  pixels$V1 = as.character(pixels$V1)
  names(pixels)[1] = 'label'
  return(pixels)
}

#Read the training and test files
train = read_digits('train.txt')
test = read_digits('test.txt')
```

```
#2.Explore the digits data:
#======================================
data = rbind(train,test)

#image for each label with average pixels
par(mfrow = c(3,4))
for(i in (0:9)){
  pix = data[data$label == i,][-1]
  pix_mean = sapply(pix,mean)
  pix_mean = as.numeric(pix_mean)
  pix_mean = matrix(pix_mean,16,16)
  image(pix_mean,ylim=c(1,0),xlim =
c(0,1),col=grey(seq(1,0,length=256)))
}


#Which pixels seem the most likely to be useful,and which
not?
#we can see this by variance
sort(sapply(data[2:257],var))
```

```
#3.Write a function predict_knn() that uses k-nearest
neighbors
#to predict the label for a point or collection of points.
#=====================================================

#The funciton of getting the distance matrix
distance <- function(prediction,training){
  distances = dist(rbind(prediction[-1],training[-1]))
  distances = as.matrix(distances)
  distances =
distances[1:nrow(prediction),(nrow(prediction)+1):(nrow(pre
diction)+nrow(training))]
  return(distances)
}
```

```
#Function of finding the winning label
winning_labels <- function(count){
  maximum = max(count)
  maximum = count[count == maximum]
  #If having ties, sample(names(maximum),1) choose one
randomly
  #otherwise, sample(names(maximum),1) just give the unique
result.
  label = sample(names(maximum),1)
  return(label)
}



#Prediction function
predict_knn <- function(prediction,training,k,distance){
  #order the distance matrixs and rearrange the labels
  train_index = apply(distance,1,order)
  train_label = training$label[train_index]
  train_label = matrix(train_label,nrow(training))
  new_label = train_label[1:k,]
  #initilize labels
  labels = rep(1,nrow(prediction))
  #a for loop to get the labels
  for(i in (1:nrow(prediction))){
    if (k == 1){
      count = table(new_label[i])
      labels[i] = winning_labels(count)
    }else{
      count = table(new_label[,i])
      labels[i] = winning_labels(count)
    }
  }
  return(labels)
}


#Check how it works
#choose k = 5
```

```
predict_train1 = predict_knn(train,train,5,distances)
#choose extreme case k = 1
predict_train2 = predict_knn(train,train,1,distances)
#choose extreme case k = n,which is 7291.
predict_train3 = predict_knn(train,train,7291,distances)
```

```
#4.Write a function cv_error_knn()
#=======================================================
#Rivise the distance function since I will add group to the
data frame,
#And I do not want to group be calcaulated.
distance2 <- function(prediction,training){
  distances = dist(rbind(prediction[-ncol(prediction)][-
1],training[-ncol(training)][-1]))
  distances = as.matrix(distances)
  distances =
distances[1:nrow(prediction),(nrow(prediction)+1):(nrow(pre
diction)+nrow(training))]
  return(distances)
}


#Since we focus on train, we could calcaulate the distance
outside the loop
train_shuffled = train[sample(nrow(train)), ]
train_shuffled$group = rep(c(1:10),length.out =
length(train_shuffled$label))
distance_whole = distance2(train_shuffled,train_shuffled)



#Corss validation function
cv_error_knn <- function(training,k,distance_whole){
```

```r
  each_error = rep(1,10)#initialize each_error
  for(j in (1:10)){
    each_group = training[training$group == j,]
    other_group = training[training$group != j,]
    #get the distance for each k from the distance_whole
matrix
    distance =
distance_whole[(training$group==j),(training$group!=j)]
    label = predict_knn(each_group,other_group,k,distance)
    each_error[j] = sum(label !=
each_group$label)/length(label)
  }

  error = mean(each_error)
  return(error)
}
```

```r
#5. In one plot, display 10-fold CV error rates for all
combinations
#of k = 1, . . . , 15 and two different distance metrics.
#===========================================================

#initilize error_train0
error_train = rep(1,15)

#Calculate the euclidean distance and k = 1 to k = 15
for(k in (1:15)){
  error_train[k] =
cv_error_knn(train_shuffled,k,distance_whole)
```

```r
}


#Another distance matrics with manhattan method
distance_man <- function(prediction,training){
   distances = dist(rbind(prediction[-ncol(prediction)][-
1],training[-ncol(training)][-1]),method = 'manhattan')
   distances = as.matrix(distances)
   distances =
distances[1:nrow(prediction),(nrow(prediction)+1):(nrow(pre
diction)+nrow(training))]
   return(distances)
}

#Do the same thing as we did in the former method of
distance matrics
distance_whole2 =
distance_man(train_shuffled,train_shuffled)

error_train_man = rep(1,15)
for(k in (1:15)){
  error_train_man[k] =
cv_error_knn(train_shuffled,k,distance_whole2)
}

#Combine them into a data frame so that we can plot them
train_error1 = cbind(1:15,error_train)
train_error2 = cbind(1:15,error_train_man)
train_error = rbind(train_error1,train_error2)
train_error = as.data.frame(train_error)
train_error$distance_type[1:15] = rep('euclidean',15)
train_error$distance_type[16:30] = rep('manhattan',15)
names(train_error) = c('k','error_rate','type_of_distance')


#plot the test set error rates for all combinations of
#k = 1, . . . , 15 and two different distance metrics.
ggplot(train_error,aes(x = k,y = error_rate,color =
type_of_distance)) +
  geom_point() + labs(x = 'k',y = 'error rate',
                     title = 'Error rate for all combination
```

```
of k',
                  color = 'type of distance')
```

```
#6.In one plot, display the test set error rates for all
combinations of
#k = 1, . . . , 15 and two different distance metrics.
#===========================================================

error_test = rep(1,15)#Initialize the error_text

#For the distance metrics in the former problem.
#Calculate the distance out of the loop for efficiency.
distances = distance(test,train)
#calculate each error with k from 1 to 15
for(k in (1:15)){
  predict_label = predict_knn(test,train,k,distances)
  error_test[k] = sum(predict_label !=
test$label)/length(test$label)
}


#For manhattan distance matrics,
#Do the similar thing as the first distacne matrics
distance_man2 <- function(prediction,training){
  distances = dist(rbind(prediction[-1],training[-1]),method
= 'manhattan')
  distances = as.matrix(distances)
  distances =
```

```r
distances[1:nrow(prediction),(nrow(prediction)+1):(nrow(pre
diction)+nrow(training))]
  return(distances)
}

distances2 = distance_man2(test,train)
error_test2 = rep(1,15)
for(k in (1:15)){
  predict_label = predict_knn(test,train,k,distances2)
  error_test2[k] = sum(predict_label !=
test$label)/length(test$label)
}



#Put k,error,type of distance into a data frame
test_error1 = cbind(1:15,error_test)
test_error2 = cbind(1:15,error_test2)
test_error = rbind(test_error1,test_error2)
test_error = as.data.frame(test_error)
test_error$distance_type[1:15] = rep('euclidean',15)
test_error$distance_type[16:30] = rep('manhattan',15)
names(test_error) = c('k','error_rate','type_of_distance')

#plot the test set error rates for all combinations of
#k = 1, . . . , 15 and two different distance metrics.
ggplot(test_error,aes(x = k,y = error_rate,color =
type_of_distance)) +
  geom_point() + labs(x = 'k',y = 'error rate',
                     title = 'Error rate for all combination
of k',
                     color = 'type of distance')

#From the both plot,the best model is  k = 3 with euclidean
method for distance.
predict_train_k3 = predict_knn(test,train,3,distances)
test$corret = (test$label == predict_train_k3)
sort(table(test[test$corret == FALSE,]$label))#4,5 are the
digits tend to be wrong
```