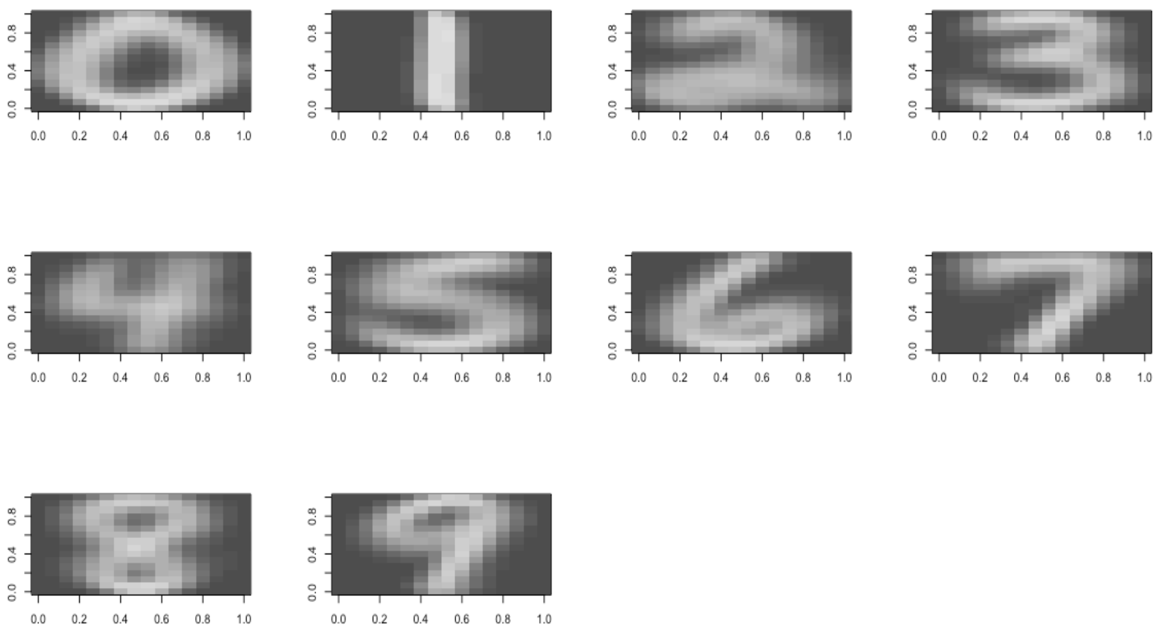
FINAL PROJECT

STA 141A

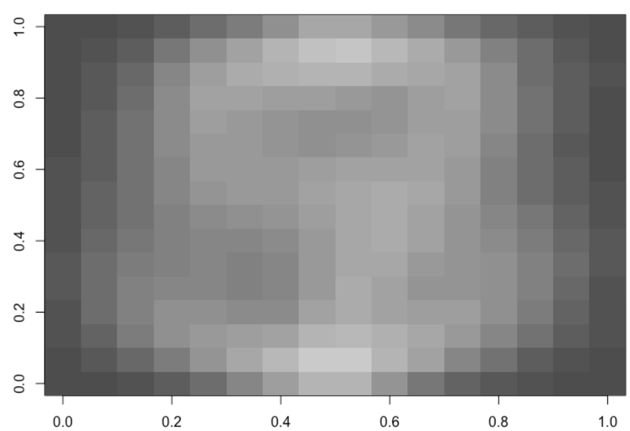
Ricardo Rendon (913892431), Arthur Wu (912779937)

1) and 2) In code

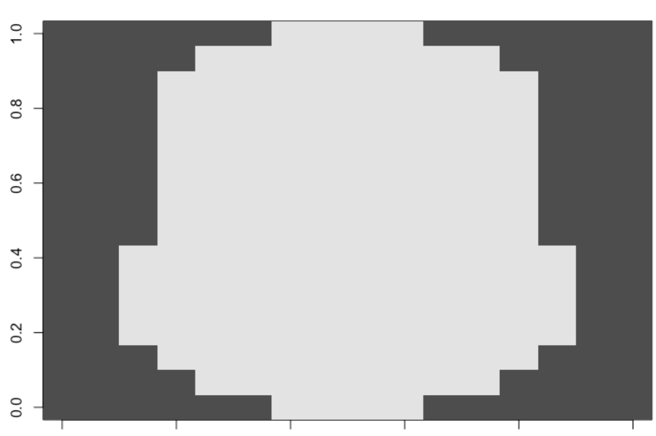
3) a)



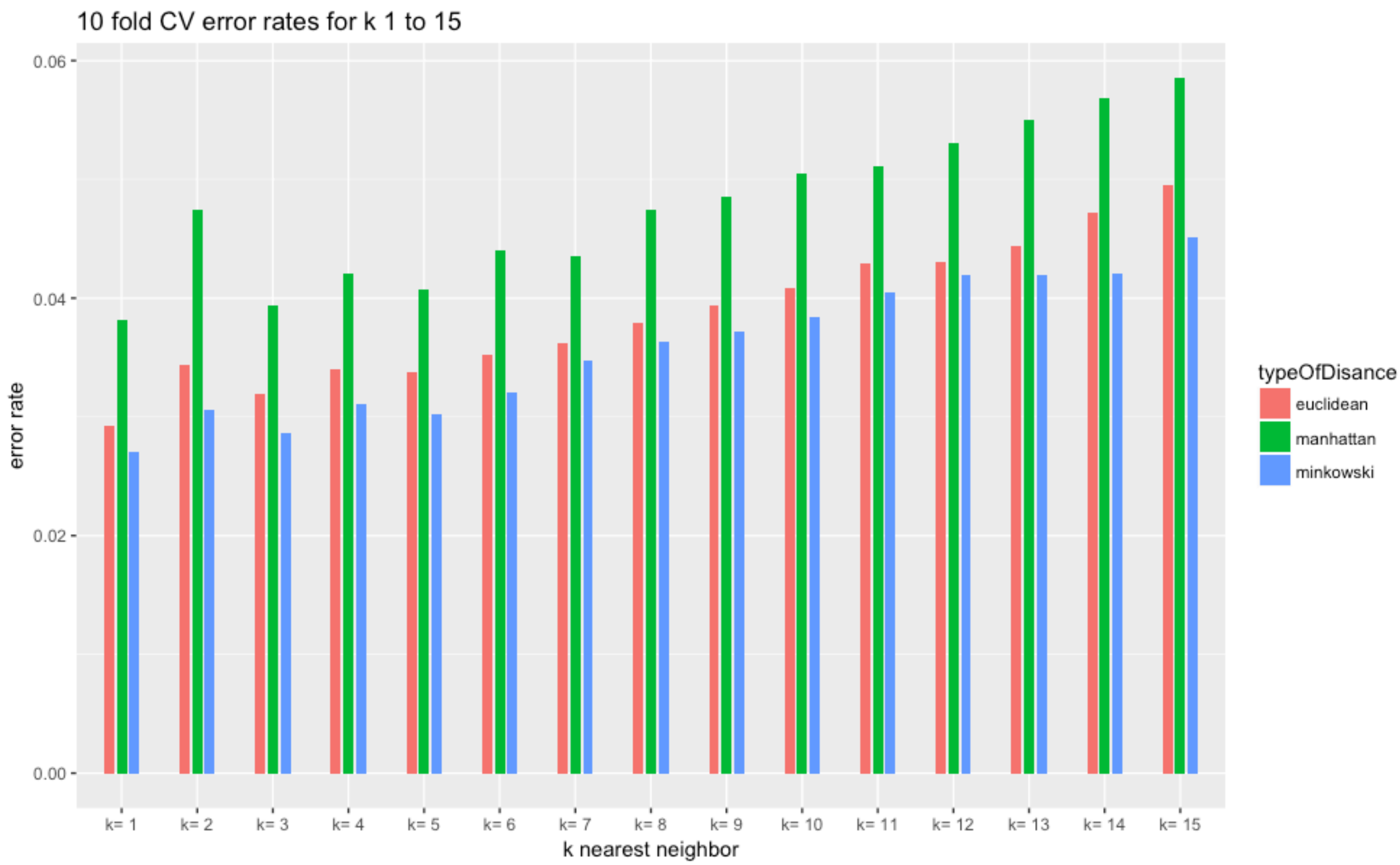
b)
The pixels that are most useful for classification are the ones that change the most (have high variation). For example, the pixels that are always black or white does not give us any input on what number the image is displaying. The pixels that have the most variance are the ones that determine which number is showing. If we take the average of each pixel on all observations, we get an image that tells us that the closer the pixel is to the color gray (the middle between black and white) the more variation it has.



We created a plot to show the variance of each pixel and set a cut-off point for variance which is 0.5 to determine which pixels are useful and colored



them in white (high variance) and which ones are in black (low variance).
4) In code
5) In code
6)



Our plot shows that as the value of k increases our error rates also increases. This may be because we are considering more observations (too many) to categorize each data point; therefore, the other extra values that are probably affecting in a negative way our classification. Also, we have to take in consideration that when k = 1, our model is only utilizing the closest distance that might not perform well with the new data since it does not have much space for noise (susceptible to random errors). For example, if we try to over-fit a model it will work perfectly for our data; however, it will not work well when you generalize the model since it does not work well for random noise.
The best combination of k and distance metric is when k = 1 and Minkowski distance (with p = 3).
Considering additional values of k do not help the categorization. This can be shown by looking at the plot; as the value of k increases in our plot, the error rates also increase.

7) The 3 best k and distance metric combinations are shown in the respective order:
Minkowski when k=1

	Reference									
Prediction	0	1	2	3	4	5	6	7	8	9
0	1187	0	2	3	3	11	7	0	4	1
1	0	1000	1	0	5	0	1	0	5	0
2	4	0	709	4	3	2	0	0	2	0
3	1	0	2	637	0	12	0	0	11	1
4	0	3	0	1	624	0	0	2	2	4
5	0	0	1	9	0	519	3	1	9	0
6	2	0	1	0	1	7	652	0	3	0
7	0	1	11	0	2	1	0	633	2	7
8	0	0	2	3	1	2	1	3	503	1
9	0	1	2	1	13	2	0	6	1	630

Minkowski when k=3

	Reference									
Prediction	0	1	2	3	4	5	6	7	8	9
0	1184	0	7	4	2	11	7	0	6	3
1	0	1002	3	1	8	0	1	3	5	0
2	3	1	703	3	2	3	0	2	1	0
3	2	0	2	641	0	10	0	0	9	1
4	0	0	1	0	613	2	0	4	4	3
5	2	0	0	5	0	521	3	0	4	0
6	3	0	1	0	5	6	653	0	1	0
7	0	1	11	0	0	0	0	629	4	6
8	0	0	2	3	0	1	0	0	506	1
9	0	1	1	1	22	2	0	7	2	630

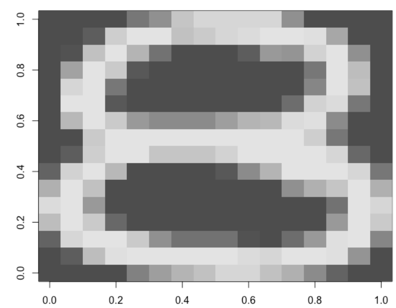
Euclidean when k=1

	Reference									
Prediction	0	1	2	3	4	5	6	7	8	9
0	1185	0	4	3	3	10	7	0	4	1
1	0	1001	2	0	7	0	1	1	5	0
2	5	0	703	4	2	3	0	0	2	0
3	1	0	3	636	1	13	0	0	15	1
4	0	3	0	1	620	2	0	3	1	4
5	1	0	1	8	0	516	2	1	7	0
6	2	0	1	0	1	7	653	0	3	0
7	0	1	13	0	2	1	0	634	3	9
8	0	0	2	5	1	2	1	1	502	1
9	0	0	2	1	15	2	0	5	0	628

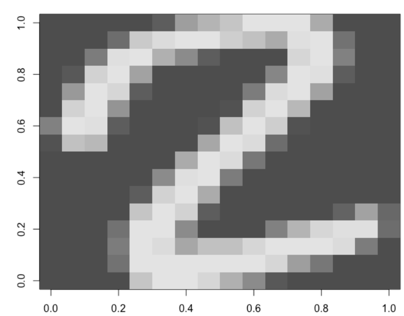
Having the smallest error rate does not always mean it is the best model to choose. Even if it has the smallest error rate it could classify one type of number always wrong and that model would not be useful. We check the confusion matrix to make sure there is not a big peak in the error rate for a number. Overall, we want an error rate that is small but stable enough to give us a reasonable error. Therefore, after analyzing the confusion matrices, we would still choose the Minkowski distance metric with k = 1 as our best combination because it has the smallest error rate and all the error rates of each observation are still distributed evenly across all categories.

8)
From the previous question, we concluded that Minkowski with k = 1 is our best combination. To explore our misclassified data points, we found some of the misclassified data points and plotted them

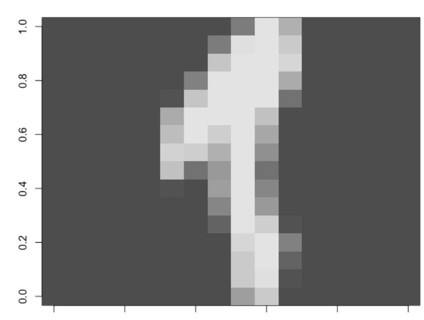
Exmaple1:
The true value of this observation is 8, however our model classified it as a 3. This happened because the person that wrote the number wrote 8 wider than the regular number 8 that we have in our data and its more similar to a 3 according to our model.

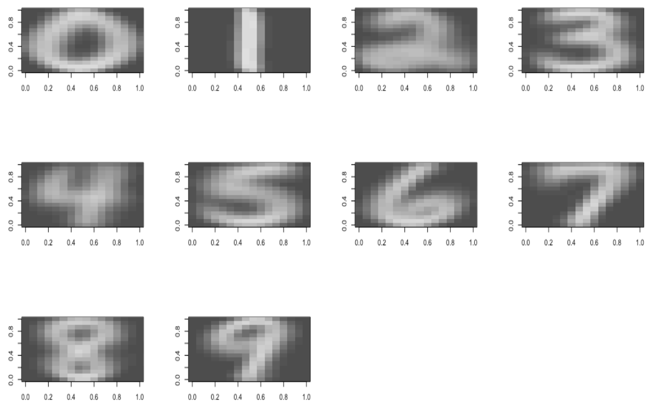


Example 2:
The true value of this observation is a 2, however our model classified it as an 8. This is because this person wrote his 2 in a slanted way that makes it look like an 8 that has not been completely finished.



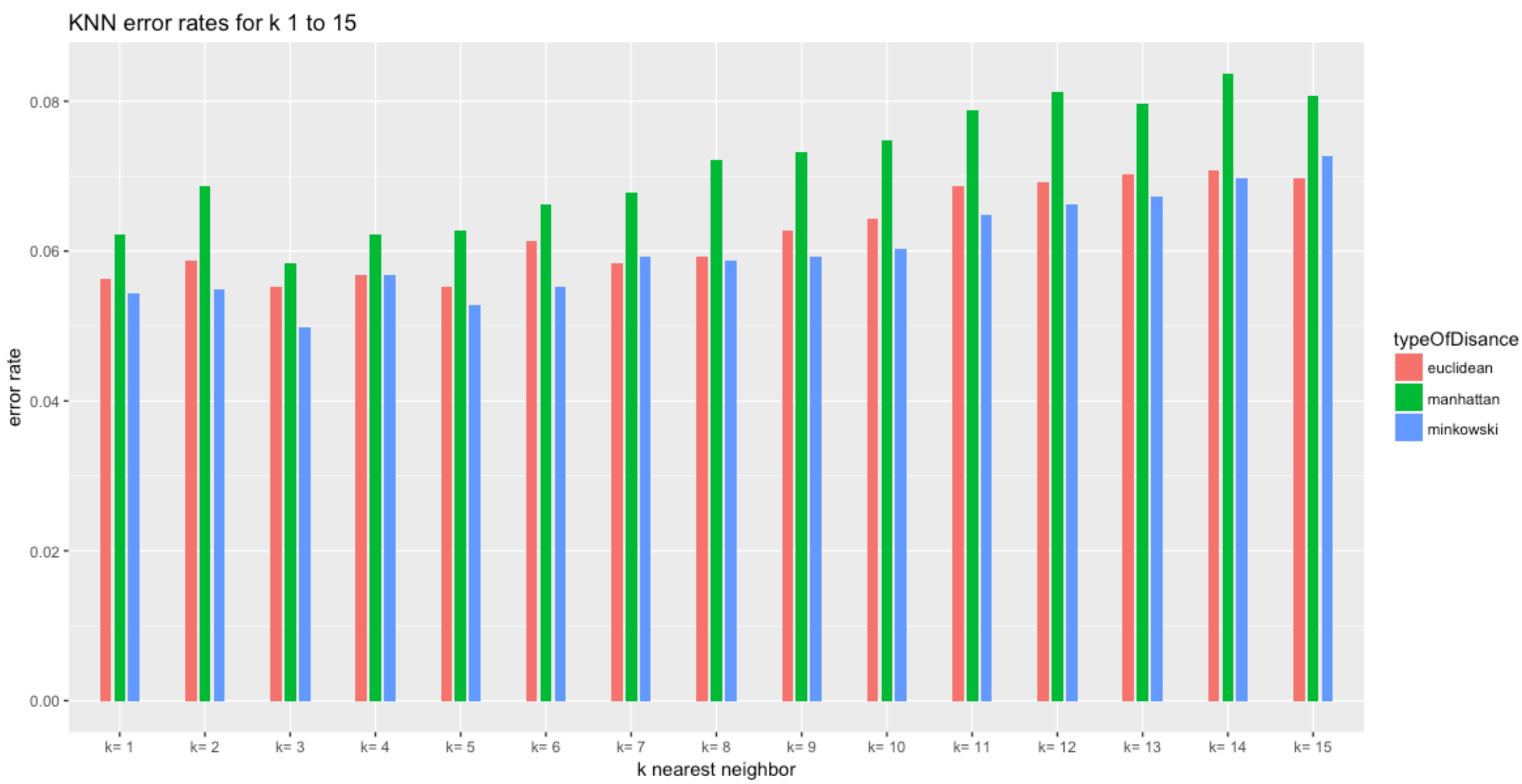
Example3:
This was supposed to be a 1 but our model classified it as a 9. The person who wrote this number one, wrote it in an unusual way.

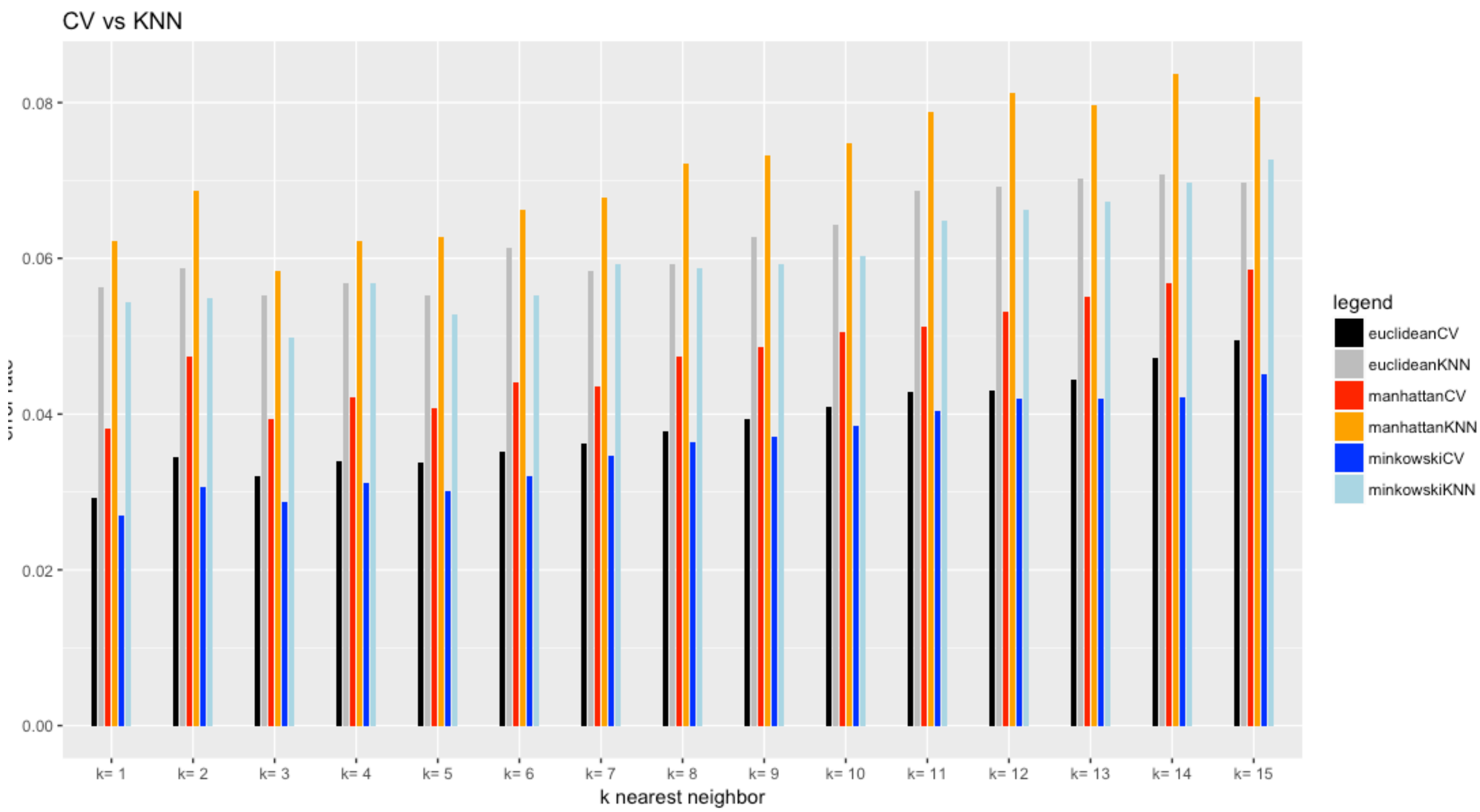




As we can see the misclassified numbers are special cases in our data set, so we can assume that our model works fine (assuming some errors from our model that is not the fault of the user). The issue is that the user writing the number is writing it in an unusual way. Most of these error were between 3 and 8, 9 and 4, and 7 and 2. (this sounds right since these numbers can be easliy written in a similar way with each other). These pairs of numbers usually look alike so it is harder to classify when the user writes the number in a unique way.

9)





Here we can see that the top 4 distances (the ones tht give the smalles error rate) do not differ from cross validation, which is good, however the best one does. Also, the trend does not change. In general, as the value of k increases, the higher the error rate in CV and KNN. However, we can see that there is a drop in error rate when when k approaches to 3 in KNN. We can then consider choosing the Minkowski distance with p = 3 as our best distance metric. Cross validation is an estimate for KNN. In our case, our cross validation gave us good estimates for our classifier. However, in reality, we may not get good estimates since we do not know the true values like we do here.

10)
One of the members left since we couldn't work toghether due to time issues. Ricardo and Arthur worked on all the problems together, we did not separate the assignment into parts. Thus, the contribution was equal for the both of us.

Appendix:

```
install.packages("caret")
install.packages("tidyr")
install.packages("reshape2")
library(caret)
library(tidyr)
library(ggplot2)
library(reshape2)
library(tidyr)
library(ggplot2)
setwd("/Users/rire948/Downloads/digits")

## 1 DONE
read_digits = function(name){

  data = read.table(name)
  data[,1] = as.factor(data[,1])

  return(data)
}
data  = read_digits("train.txt")
data23123=read_digits("test.txt")


#2 DONE
data2=data[,-1]##data train with no lables

## note data need to not have lables in our function,. we get rid of lab on the top
view_digits = function(selection,use_data){
  data=use_data
  data=as.matrix(data)
  if(dim(data)[2]==1){
    data=t(data)
  }
  pixel_graph=data
  rotate=function(x) {t(apply(x,2,rev))}

  pixel_graph=apply(t(matrix(as.numeric(data[selection,]),16,16)),1,rev)
  pixel_graph=rotate(rotate(pixel_graph))

  image(pixel_graph,zlim =c(-1,1),col=grey.colors(n = length(pixel_graph)))

}

view_digits(5,data2)
```

```
##3 DONE

lables=data[,1]

  avrg=sapply(0:9, function(l){colMeans(data2[lables==l,])})
  avrg=t(avrg)
  par(mfrow=c(3,4))

 sapply(1:10, function(a){view_digits(a,avrg)})
 par(mfrow=c(1,1))


 ##b

new_Avrg=colMeans(avrg)

view_digits(1,new_Avrg)

#######
variance_3b=apply(data2, 2, var)

variance_3b<-ifelse(variance_3b>0.5, 1,-1)

view_digits(1,variance_3b)

## so the ones that are grey are the most important one since they change the most



#4##
#Note" train first colum should be the lables and then the pixels
Knn=function(train,test,dist_method,k){

  data_train  = read.table(train)[,-1]
  data_train=as.matrix(data_train)
  data_trainlabels=read.table(train)[1]
  data_test  = read_digits(test)[,-1]
  data_test=as.matrix(data_test)


  data_usefull=rbind(data_train,data_test)
  distFinal=dist(data_usefull,method=dist_method)

  new123123=as.matrix(distFinal)
```

```
  new123123=new123123[-c(1:7291),]
  new123123=new123123[,-c(7292:9298)]##selecting the data points we need


  new123123457=as.vector(t(new123123))


  dist=new123123457

  chunk2 <- function(x,n) split(x, cut(seq_along(x), n, labels = FALSE))
  new1=chunk2(dist,2007)


  new2=lapply(new1,function(x){which(x %in% sort(x)[1:k])})
  new2=lapply(new2,function(x){x[1:k]})

  ## we need the labels for those locations(which we have in our train data)
  matrix1=matrix(ncol = k,nrow = 2007)

  for (i in 1:2007) {
   matrix1[i,]=data_trainlabels[c(as.vector(new2[[i]])),]
  }
  matrix1

  #to get the most comon values of each row
  new123=as.vector(as.numeric(apply(matrix1,1,function(x)
names(which.max(table(x))))))
  matrix1=as.data.frame(matrix1)
  matrix1$new=new123
  matrix1$new

  return( matrix1$new)

}#2minutes

#example
question4=Knn("train.txt","test.txt","euclidean",15)

#5

#Note" train first colum should be the lables and then the pixels
cv_error_knn=function(train,dist_method,k,fold){

  train5=read.table(train)
  train_NoLables5=read.table(train)[,-1]
  train_Lables5=read.table(train)[,1]
```

```
  dist_train5=dist(train_NoLables5,method=dist_method)##takes 2 min to run
  dist_train5=as.matrix(dist_train5)


rows_cross5=split(sample(nrow(train_NoLables5)),rep(1:fold,length=nrow(train_NoLable
s5)))

  ##implement function or loop to repeat this fold_test number of times
  get_errorrate=function(fold_test){

   dimension_row_test=dim(dist_train5[c(rows_cross5[[fold_test]]),
as.vector(unlist(rows_cross5[c(-fold_test)]))])[1]

  #so all distance for training row vs test row for 10 fold_test when using group 10 as  test
   dist_train6=dist_train5[c(rows_cross5[[fold_test]]), as.vector(unlist(rows_cross5[c(-
fold_test)]))]

   matrix123123=matrix(nrow = dimension_row_test,ncol = k)
   for (i in 1:dimension_row_test) {
     matrix123123[i,]= as.numeric(names(head(sort(dist_train6[i,]),k)))
    }
   matrix123123

   matrix567567=matrix(nrow=dimension_row_test,ncol=k)
   for (i in 1:nrow(matrix123123)) {
     matrix567567[i,]=train5[matrix123123[i,],1]

   }
   matrix567567

   new567567=as.vector(as.numeric(apply(matrix567567,1,function(x)
names(which.max(table(x))))))##predicted ones
   train5[c(rows_cross5[[fold_test]]),1]#real ones

  ##so to chek if is right
  ##so for the eerrror rate
   sum(new567567!=train5[c(rows_cross5[[fold_test]]),1])/length(new567567)
   return(sum(new567567!=train5[c(rows_cross5[[fold_test]]),1])/length(new567567))
  }
  rate_array=c(1:fold)

  for(i in 1:fold){
    rate_array[i]=get_errorrate(i)
  }
return(mean(rate_array))
```

```
} ## 2.5minutes

##6 DONE
#so now that we create the functions we can use it for the rest pf questions
# so since we want to use the same randomisation across all the next question we save
some global variables
#so first we run our data set in a new cvv function that saves some global variables that we
are going to reuse
#in later questions, since fold stays constant as 10 and the separation of the data set in 10
groups
#is also going to be the same we save folds ans wich obs is in which separation globaly.:
fold =10
cv_error_knn_GLOBALSAVED=function(train,fold){

  train5=read.table(train)
  train_NoLables5=read.table(train)[,-1]
  train_Lables5=read.table(train)[,1]


rows_cross5=split(sample(nrow(train_NoLables5)),rep(1:fold,length=nrow(train_NoLable
s5)))
  return(rows_cross5)

}
rows_cross5=cv_error_knn_GLOBALSAVED("train.txt",10) ## WE RN THIS TO SAVE OUR
separation
##rows_cross5, K DOSNT CHANGE THE rowcross so it dosnt matter the vlaue of k and fold
is alwasy 10 from now on


##crating more globals for distance
train_NoLables5=read.table("train.txt")[,-1]
dist_train5_eucl=dist(train_NoLables5,method="euclidean")##takes 2 min to run
dist_train5_man=dist(train_NoLables5,method="manhattan")##takes 2 min to run
dist_train5_min=dist(train_NoLables5,method="minkowski",p = 3)##takes 4-5 min to run

##using our data form the start of 6( the globals we created on start of 6)
## get 1:15k

cv_error_knn6=function(train,trainlabels,dist_method,k,fold){


  train5=read.table(train)
  train_NoLables5=read.table(train)[,-1]
  train_Lables5=read.table(train)[,1]
```

```
 #dist_train5=dist(train_NoLables5,method=dist_method)##takes 2 min to run
 if(dist_method=="euclidean"){dist_train5=dist_train5_eucl
 } else if(dist_method=="manhattan"){dist_train5=dist_train5_man
 } else if(dist_method=="minkowski"){dist_train5=dist_train5_min}
 dist_train5=as.matrix(dist_train5)


#rows_cross5=split(sample(nrow(train_NoLables5)),rep(1:fold,length=nrow(train_NoLabl
es5)))

 ##implement function or loop to repeat this fold_test number of times
 new_forquestion6=matrix(ncol = 15,nrow = 10)## were we gona save the rates for each k
when using 10 fold

 for(a in 1:k){

  for(b in 1:10){
 get_errorrate=function(fold_test){

  dimension_row_test=dim(dist_train5[c(rows_cross5[[fold_test]]),
as.vector(unlist(rows_cross5[c(-fold_test)]))])[1]
  dimension_row_test


  #so all distance for training row vs test row for 10 fold_test when using group 10 as  test
  dist_train6=dist_train5[c(rows_cross5[[fold_test]]), as.vector(unlist(rows_cross5[c(-
fold_test)]))]

  matrix123123=matrix(nrow = dimension_row_test,ncol = a)
  for (i in 1:dimension_row_test) {
   matrix123123[i,]= as.numeric(names(head(sort(dist_train6[i,]),a)))
  }
  matrix123123
  #rownames(matrix123123)=rows_cross5[[fold_test]]##if problem in herere deleat it
  ## so there are the min locations of each row(test) and column(train). now find the
lables of each
  ## so i think the first row mean the first location(index,row) that i used for test so
c(rows_cross5[[10]])

  matrix567567=matrix(nrow=dimension_row_test,ncol=a)
  for (i in 1:nrow(matrix123123)) {
   matrix567567[i,]=train5[matrix123123[i,],1]

  }
  matrix567567
```

```
   #rownames(matrix567567)=rows_cross5[[fold_test]]#if problem in herere deleat it

   new567567=as.vector(as.numeric(apply(matrix567567,1,function(x)
names(which.max(table(x))))))##predicted one

   train5[c(rows_cross5[[fold_test]]),1]#real one
   ##so to chek if is right
   ##so this is for fold_test number 1. we need to repeat this 9 more times
   ## so could just create another function that envelops this one that make i repeat 10
times and then we get the
   ##error rate

   ##SO FOR THe eerrror rate in this one is=
   sum(new567567!=train5[c(rows_cross5[[fold_test]]),1])/length(new567567)

   return(  sum(new567567!=train5[c(rows_cross5[[fold_test]]),1])/length(new567567))
  }


  new_forquestion6[b,a]= get_errorrate(b)
   }
}
  return(colMeans(new_forquestion6))

}  ## we redid function to make

aewq123=c(1:15)
aewq123=cv_error_knn6("train.txt","train.txt","euclidean",15,fold)##these are the avrg
eerror rates for 10 fold changing k form 1:15 euclidean

matrix123123r4=matrix(ncol=15,nrow=3)
matrix123123r4[1,]=aewq123


a45645k=c(1:15)
a45645k=cv_error_knn6("train.txt","train.txt","manhattan",15,fold)##these are the avrg
eerror rates for 10 fold changing k form 1:15 manhatan
matrix123123r4[2,]=a45645k

ab45645k=c(1:15)
ab45645k=cv_error_knn6("train.txt","train.txt","minkowski",15,fold)##these are the avrg
eerror rates for 10 fold changing k form 1:15 manhatan
matrix123123r4[3,]=ab45645k

colnames(matrix123123r4)=paste("k=",c(1:15))
rownames(matrix123123r4)=c("euclidean","manhattan","minkowski")
```

```
matris_2=matrix123123r4
matris_2=as.data.frame(matris_2)

matris_2$typeOfDisance = c("euclidean","manhattan","minkowski")
dat.g = melt(matris_2, id.vars = "typeOfDisance")
ggplot(dat.g, aes(variable, value)) + geom_bar(aes(fill = typeOfDisance),width = 0.4,
position = position_dodge(width=0.5), stat="identity")+
  labs(size = "Frequency", y = "error rate", x = "k nearest neighbor",title = "10 fold CV error
rates for k 1 to 15")



#7
##
sort(matrix123123r4)

cv_error_knn7=function(train,dist_method,k,fold){

  train5=read.table(train)
  train_NoLables5=read.table(train)[,-1]
  train_Lables5=read.table(train)[,1]

  #dist_train5=dist(train_NoLables5,method=dist_method)##takes 2 min to run
  #dist_train5=as.matrix(dist_train5)
  if(dist_method=="euclidean"){dist_train5=dist_train5_eucl
  } else if(dist_method=="manhattan"){dist_train5=dist_train5_man
  } else if(dist_method=="minkowski"){dist_train5=dist_train5_min}
  dist_train5=as.matrix(dist_train5)


  ##implement function or loop to repeat this fold_test number of times
  get_errorrate=function(fold_test){

    dimension_row_test=dim(dist_train5[c(rows_cross5[[fold_test]]),
as.vector(unlist(rows_cross5[c(-fold_test)]))])[1]
    dimension_row_test

    #so all distance for training row vs test row for 10 fold_test when using group 10 as  test
    dist_train6=dist_train5[c(rows_cross5[[fold_test]]), as.vector(unlist(rows_cross5[c(-
fold_test)]))]

    matrix123123=matrix(nrow = dimension_row_test,ncol = k)
    for (i in 1:dimension_row_test) {
      matrix123123[i,]= as.numeric(names(head(sort(dist_train6[i,]),k)))
    }
```

```
  matrix123123
  #rownames(matrix123123)=rows_cross5[[fold_test]]##if problem in herere deleat it
  ## so there are the min locations of each row(test) and column(train). now find the lables of each
  ## so i think the first row mean the first location(index,row) that i used for test so c(rows_cross5[[10]])

  matrix567567=matrix(nrow=dimension_row_test,ncol=k)
  for (i in 1:nrow(matrix123123)) {
   matrix567567[i,]=train5[matrix123123[i,],1]

  }
  matrix567567
  #rownames(matrix567567)=rows_cross5[[fold_test]]#if problem in herere deleat it

  new567567=as.vector(as.numeric(apply(matrix567567,1,function(x) names(which.max(table(x))))))##predicted one

  train5[c(rows_cross5[[fold_test]]),1]#real one
  ##so to chek if is right

  return(new567567)
 }


 rate_array=c(1:fold)
 rate_array=as.list(rate_array)

 asder123=c(1:fold)
 asder123=as.list(rate_array)

 for(i in 1:fold){
 rate_array[[i]]=get_errorrate(i)

 asder123[[i]]=train5[c(rows_cross5[[i]]),1]
}
 rate_array=unlist(rate_array)## predicted
 asder123=unlist(asder123)##true


 confusionMatrix(rate_array,asder123)[[2]]

 return(  confusionMatrix(rate_array,asder123)[[2]])

}
```

```
finalmatrix7=cv_error_knn7("train.txt","euclidean",1,fold)
finalmatrix7
finalmatrix71=cv_error_knn7("train.txt","minkowski",1,fold)
finalmatrix71
finalmatrix72=cv_error_knn7("train.txt","minkowski",3,fold)
finalmatrix72



##8

cv_error_knn8=function(train,dist_method,k,fold){


  train5=read.table(train)
  train_NoLables5=read.table(train)[,-1]
  train_Lables5=read.table(train)[,1]

  #dist_train5=dist(train_NoLables5,method=dist_method)##takes 2 min to run
  #dist_train5=as.matrix(dist_train5)
  if(dist_method=="euclidean"){dist_train5=dist_train5_eucl
  } else if(dist_method=="manhattan"){dist_train5=dist_train5_man
  } else if(dist_method=="minkowski"){dist_train5=dist_train5_min}
  dist_train5=as.matrix(dist_train5)


  ##implement function or loop to repeat this fold_test number of times
  get_errorrate=function(fold_test){

    dimension_row_test=dim(dist_train5[c(rows_cross5[[fold_test]]),
as.vector(unlist(rows_cross5[c(-fold_test)]))])[1]
    dimension_row_test

    #so all distance for training row vs test row for 10 fold_test when using group 10 as  test
    dist_train6=dist_train5[c(rows_cross5[[fold_test]]), as.vector(unlist(rows_cross5[c(-
fold_test)]))]

    matrix123123=matrix(nrow = dimension_row_test,ncol = k)
    for (i in 1:dimension_row_test) {
      matrix123123[i,]= as.numeric(names(head(sort(dist_train6[i,]),k)))
    }
    matrix123123
    #rownames(matrix123123)=rows_cross5[[fold_test]]##if problem in herere deleat it
    ## so there are the min locations of each row(test) and column(train). now find the
lables of each
```

```
   ## so i think the first row mean the first location(index,row) that i used for test so
c(rows_cross5[[10]])

   matrix567567=matrix(nrow=dimension_row_test,ncol=k)
   for (i in 1:nrow(matrix123123)) {
     matrix567567[i,]=train5[matrix123123[i,],1]

   }
   matrix567567
   #rownames(matrix567567)=rows_cross5[[fold_test]]#if problem in herere deleat it

   new567567=as.vector(as.numeric(apply(matrix567567,1,function(x)
names(which.max(table(x))))))##predicted one

   train5[c(rows_cross5[[fold_test]]),1]#real one
   ##so to chek if is right

   return(new567567)
  }


  rate_array=c(1:fold)
  rate_array=as.list(rate_array)

  asder123=c(1:fold)
  asder123=as.list(rate_array)

  for(i in 1:fold){
   rate_array[[i]]=get_errorrate(i)

   asder123[[i]]=train5[c(rows_cross5[[i]]),1]
  }
  rate_array=unlist(rate_array)## predicted
  asder123=unlist(asder123)##true

  q8=matrix(c(rate_array,asder123),ncol = 2,nrow = length(rate_array))

  return( q8)

}

Q8=cv_error_knn8("train.txt","minkowski",1,fold)

## we know which one are mistaken so we will plot those to see what was the input to
determine if the
#user wrote the number in a wierd way or our model is just wrong
```

```
colnames(Q8)=c("predicted","true")
Q8
#input the row from matrix(Q8)that does not match predicted with true
view_digits(as.vector(unlist(rows_cross5)[125]),data2)
view_digits(as.vector(unlist(rows_cross5)[212]),data2)
view_digits(as.vector(unlist(rows_cross5)[495]),data2)

# as we can see here  our model is not wrong is just that these people wrote the number in
a
#wierd way

##9
#we set globals to make it run faster


train="train.txt"
trainlabels="train.txt"
test="test.txt"
dist_method="euclidean"

data_train  = read.table(train)[,-1]
data_train=as.matrix(data_train)
data_trainlabels=read.table(trainlabels)[1]
data_test  = read_digits(test)[,-1]
data_test_labels  = read_digits(test)[,1]

data_test=as.matrix(data_test)

data_usefull=rbind(data_train,data_test)
distFinal_euc=dist(data_usefull,method="euclidean")##2min to run
distFinal_manh=dist(data_usefull,method="manhattan")##2m to run
distFinal_min=dist(data_usefull,method="minkowski",p = 3)##8 min to run

Knn9=function(train,test,dist_method,k){

 if(dist_method=="euclidean"){distFinal=distFinal_euc}
 else if(dist_method=="manhattan"){distFinal=distFinal_manh}
 else if(dist_method=="minkowski"){distFinal=distFinal_min}

  new123123=as.matrix(distFinal)
  new123123=new123123[-c(1:7291),]
  new123123=new123123[,-c(7292:9298)]


  new123123457=as.vector(t(new123123))
```

```
 ##
 #euc.dist <- function(x1, x2) sqrt(sum((x1 - x2) ^ 2))
 #euc.dist(data_train[1,],data_test[1,])
 #euc.dist(data_train[1,],data_test[2,])
 #euc.dist(data_train[1,],data_test[3,])
 ##

 #####

 #so thiss shwos us:##after his dont change!!!
 dist=new123123457
 chunk2 <- function(x,n) split(x, cut(seq_along(x), n, labels = FALSE))
 new1=chunk2(dist,2007)
 ## this gives us all the distance of each test compared to each train
 ## this gives us the location of each samll one
 new2=lapply(new1,function(x){which(x %in% sort(x)[1:k])})
 new2=lapply(new2,function(x){x[1:k]})
 head(new2)
 new1[[1]]
 ## we need the labels for those locations(which we have in our train data)
 matrix1=matrix(ncol = k,nrow = 2007)
 for (i in 1:2007) {
  matrix1[i,]=data_trainlabels[c(as.vector(new2[[i]])),]

 }
 matrix1
 #to get the most comon values of each row
 new123=as.vector(as.numeric(apply(matrix1,1,function(x)
names(which.max(table(x))))))
 matrix1=as.data.frame(matrix1)
 matrix1$new=new123
 matrix1$new
 return( matrix1$new)
}##2m


eucl9=sapply(1:15, function(x) Knn9("train.txt","test.txt","euclidean",x))

manhat9=sapply(1:15, function(x) Knn9("train.txt","test.txt","manhattan",x))

minkiu9=sapply(1:15, function(x) Knn9("train.txt","test.txt","minkowski",x))

eucl9
manhat9
minkiu9
```

```
true_lables=as.numeric(as.vector(data_test_labels))

eucl_errorrate=apply(eucl9,2,function(x){x==true_lables})
eucl_errorrate=colMeans(eucl_errorrate)
eucl_errorrate=t(as.matrix(eucl_errorrate))
row.names(eucl_errorrate)="euclidean"
colnames(eucl_errorrate)=paste("k=",c(1:15))
eucl_errorrate

manhat9_errorrate=apply(manhat9,2,function(x){x==true_lables})
manhat9_errorrate=colMeans(manhat9_errorrate)
manhat9_errorrate=t(as.matrix(manhat9_errorrate))
row.names(manhat9_errorrate)="manhattan"
colnames(manhat9_errorrate)=paste("k=",c(1:15))
manhat9_errorrate

minkiu9_errorrate=apply(minkiu9,2,function(x){x==true_lables})
minkiu9_errorrate=colMeans(minkiu9_errorrate)
minkiu9_errorrate=t(as.matrix(minkiu9_errorrate))
row.names(minkiu9_errorrate)="minkowski"
colnames(minkiu9_errorrate)=paste("k=",c(1:15))
minkiu9_errorrate

FINALEE=rbind(eucl_errorrate,manhat9_errorrate,minkiu9_errorrate)
FINALEE=1-FINALEE
FINALEE=as.data.frame(FINALEE)

FINALEE$typeOfDisance = c("euclidean","manhattan","minkowski")
dat.g = melt(FINALEE, id.vars = "typeOfDisance")
ggplot(dat.g, aes(variable, value)) + geom_bar(aes(fill = typeOfDisance),width = 0.4,
position = position_dodge(width=0.5), stat="identity")+
  labs(size = "Frequency", y = "error rate", x = "k nearest neighbor",title = "KNN error rates
for k 1 to 15")

FINENENELA=rbind(matris_2,FINALEE)
FINENENELA$typeOfDisance=c("euclideanCV","manhattanCV","minkowskiCV","euclideanK
NN","manhattanKNN","minkowskiKNN")
FINENENELA
dat.g = melt(FINENENELA, id.vars = "typeOfDisance")
ggplot(dat.g, aes(variable, value)) + geom_bar(aes(fill = typeOfDisance),width = 0.4,
position = position_dodge(width=0.5), stat="identity")+
  labs(size = "Frequency", y = "error rate", x = "k nearest neighbor",title = "KNN error rates
for k 1 to 15")+scale_fill_manual("legend", values = c( "manhattanCV" = "red",
"minkowskiCV" = "blue","euclideanKNN" = "grey", "manhattanKNN" = "orange",
"minkowskiKNN" = "lightblue","euclideanCV" = "black"))
```

#10

#One of te group memeber left. arthur and ricardo worked on evrything else.

##https://stackoverflow.com/questions/13458702/determining-minimum-values-in-a-vector-in-r
#https://www.r-bloggers.com/for-loops-and-how-to-avoid-them/
#https://stackoverflow.com/questions/19982938/how-to-find-the-most-frequent-values-across-several-columns-containing-factors
#https://stackoverflow.com/questions/5559384/euclidean-distance-of-two-vectors
#http://members.cbio.mines-paristech.fr/~jvert/svn/tutorials/practical/knnregression/knnregression.R
#https://stats.stackexchange.com/questions/61090/how-to-split-a-data-set-to-do-10-fold-cross-validation
#https://stackoverflow.com/questions/17606906/find-row-and-column-index-of-maximum-value-in-a-matrix
#https://stackoverflow.com/questions/19982938/how-to-find-the-most-frequent-values-across-several-columns-containing-factors
#https://stackoverflow.com/questions/13458702/determining-minimum-values-in-a-vector-in-r
#https://www.r-bloggers.com/for-loops-and-how-to-avoid-them/