Final Project - Dataset 2

AUTHOR
dm5153

Final Project

dm5153

Project 2 - Handwriting Data

Question 1.

Use PCA to reduce dimensions. How many components do you need to keep to reproduce the digits reasonably well? what is your final matrix?

To capture ~94% of the variance, 75 PCs is enough, and based on plots of the cumulative variance, the elbow at which diminishing returns for including additional PCs starts about here. The drawn images are recognizable at this level, and even at lower thresholds around 20~30 PCs the images can be visually interpreted. However, the background is not correctly reflected as completely white, and the label assignments are not correct even at 75 PCs or higher, despite cumulative variance totaling ~99%. To reach the highest level of accuracy at which the background and labels are correct, 576 PCs are needed. This amount is still 75% of the original number of components, which isn't an excellent reduction of dimensions. Excluding the labels could potentially improve the dimensionality reduction, though they are a single column with fairly low variance.

```
#import data, libraries, and set working directory
#current_path = rstudioapi::getActiveDocumentContext()$path
#setwd(dirname(current_path ))
#print( getwd() )

library(readxl)
library(png)
train <- read.csv("train.csv")
```

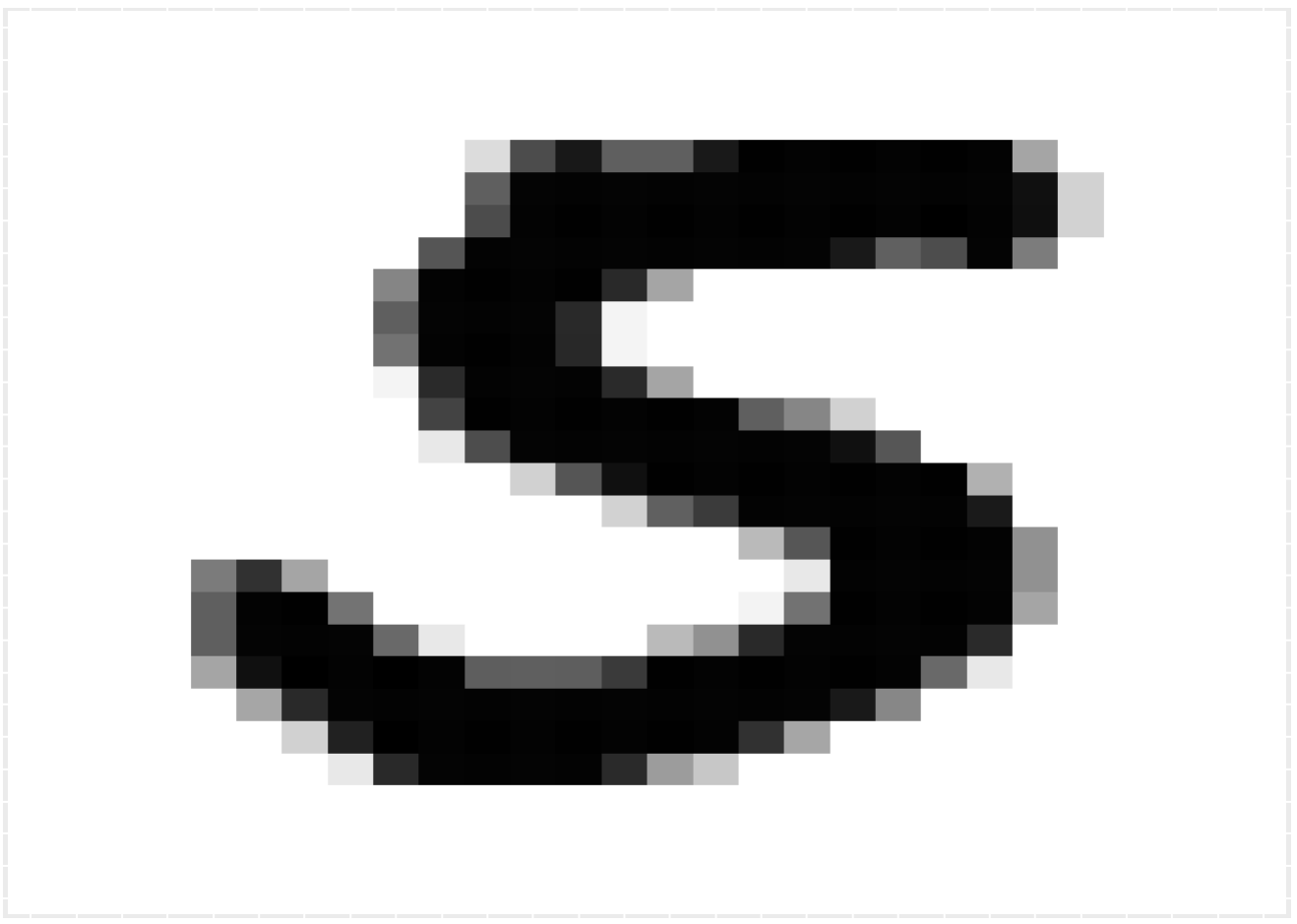
```
#function to draw the digit
draw_digit<-function(data,row){
  #import the relevant libraries
  library(ggplot2)
  library(reshape2)

  sqdim<-sqrt(ncol(data))
  #intialize the matrix with the first 28 pixels
  pixel_grid<-data[row,2:(sqdim+1)]
  #rename the columns
  colnames(pixel_grid) <- paste("Col", 1:sqdim)


  #put every 28 entries into a new row, starting at second row
  for(x in 1:(sqdim-1)){
    #define first pixel in the row
    start<-x*sqdim+2
    #define last pixel in the row
    end<-start+sqdim-1
    #hold the data from those pixels temporarily
    temp_row<-data[row,start:end]
    #make the column names match the full matrix
    colnames(temp_row) <- paste("Col", 1:sqdim)
    #add the temp row to the full matrix
    pixel_grid<-rbind(pixel_grid,temp_row)
  }
  #flip the matrix
  pixel_grid<-pixel_grid[nrow(pixel_grid):1,]
  #name the rows
  rownames(pixel_grid) <- paste("Row", 1:sqdim)
  #melt the data so ggplot can interpret it
  #also transpose at this point
  m<-melt(as.matrix(t(pixel_grid)))
  #give column names to the melted data
  colnames(m) <- c("x", "y", "value")
  #define the theme for the heatmap - remove axis etc
  theme<-theme(legend.position="none",axis.title.x=element_blank(),axis.text.x=element_blank(),axis.ticks.x=element_blank(),axis.ti
  #plot the data as a greyscale heatmap
  ggplot(m, aes(x=x,y=y,fill=value))+scale_fill_gradient(limits = c(min(m$value), max(m$value)), low = 'white', high = 'black')+geo
}
```

```r
#define a row for use as tester
testrow<-100
```

```r
#call the function on a row of your choice for reference image
draw_digit(train, testrow)
```

```
Warning: package 'ggplot2' was built under R version 4.2.2
```

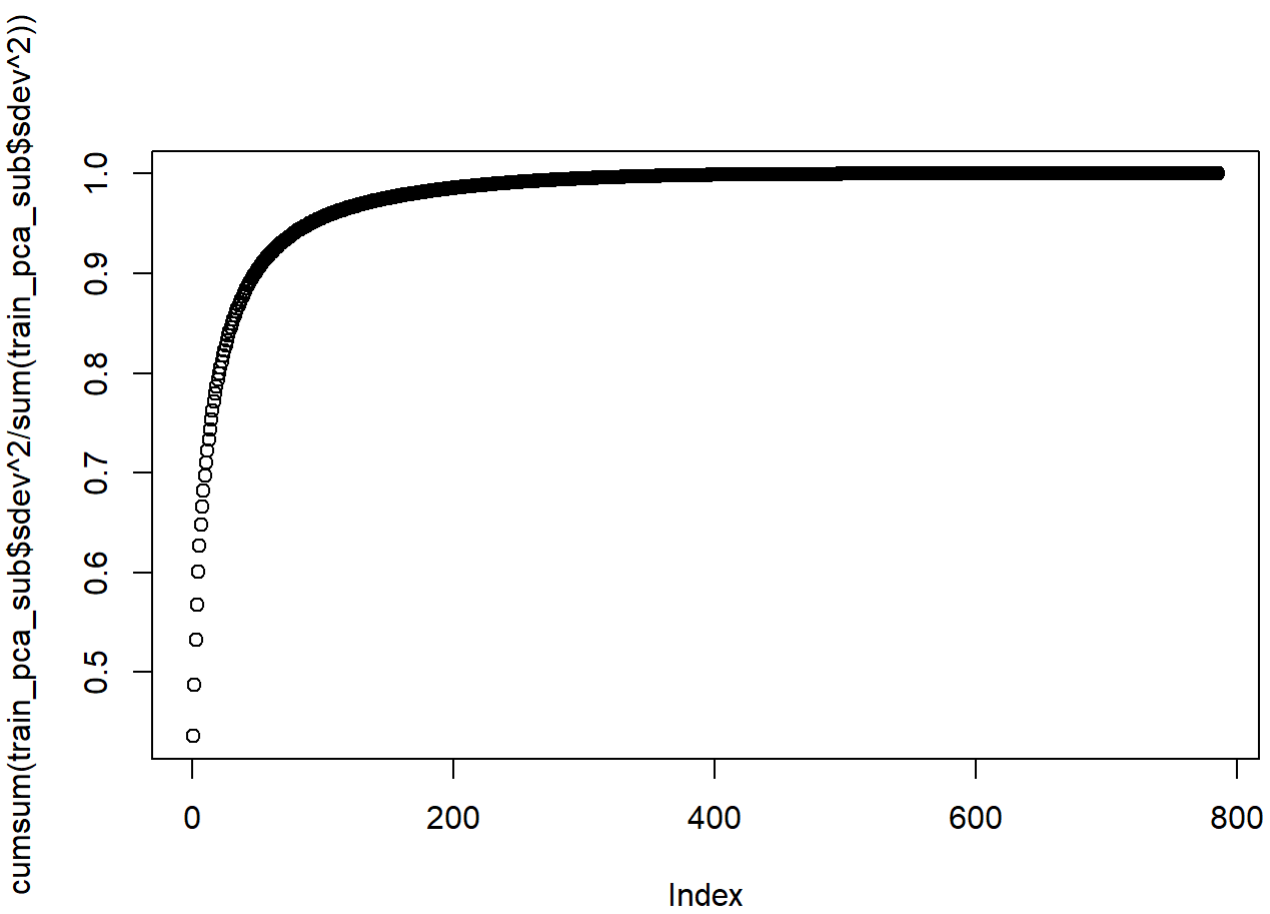
```
Warning: package 'reshape2' was built under R version 4.2.2
```



```r
#run PCA on the full dataset
full_pca<-prcomp(train, center=FALSE)
```

```r
#run PCA on a subset of the data
train_pca_sub<-prcomp(train[1:1000,], center=FALSE)
```

```r
#view the plot of PCs versus variance explained
plot(cumsum(train_pca_sub$sdev^2/sum(train_pca_sub$sdev^2)))
```

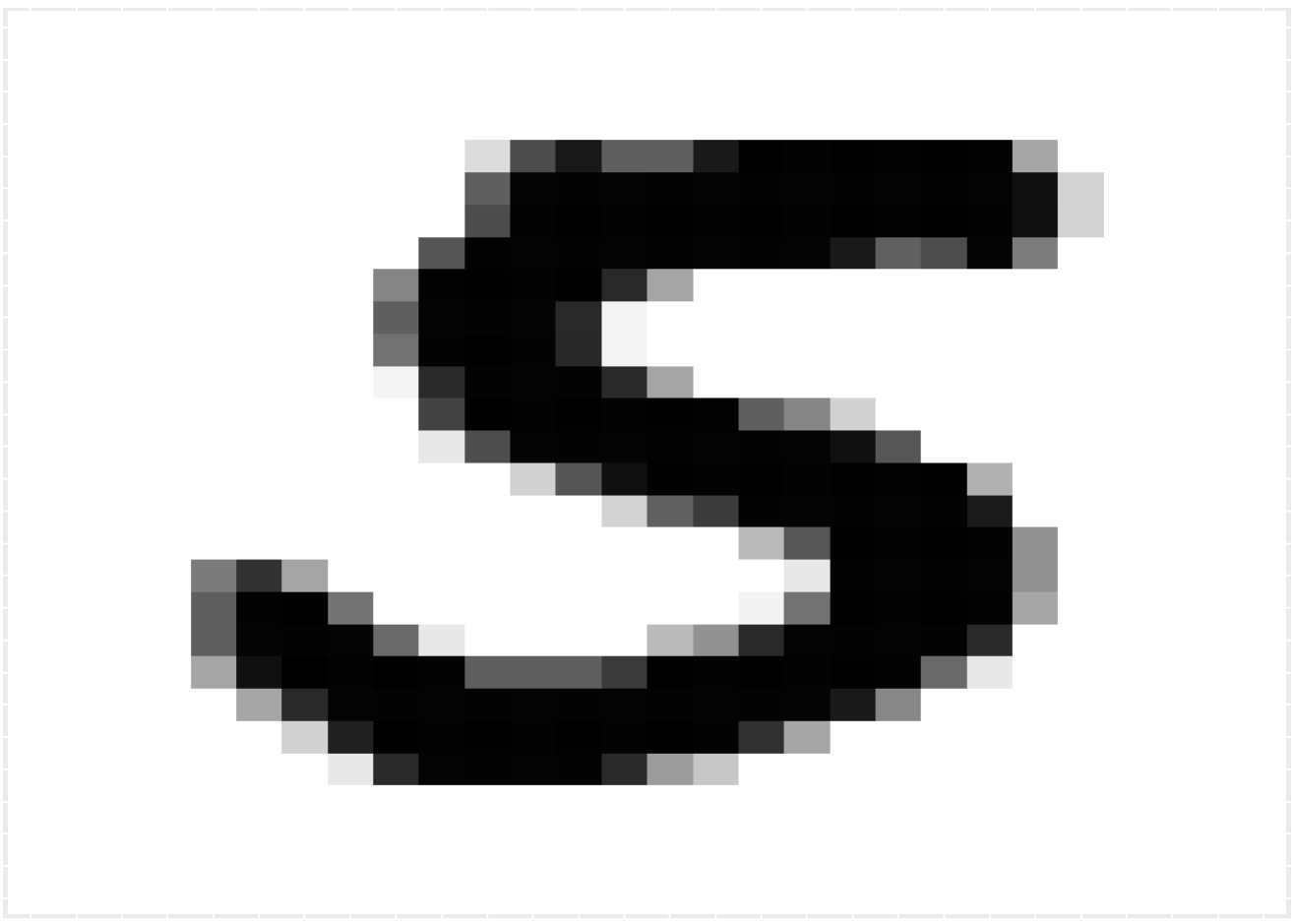



```r
#plot(train_pca_sub)
#summary(train_pca_sub)
```

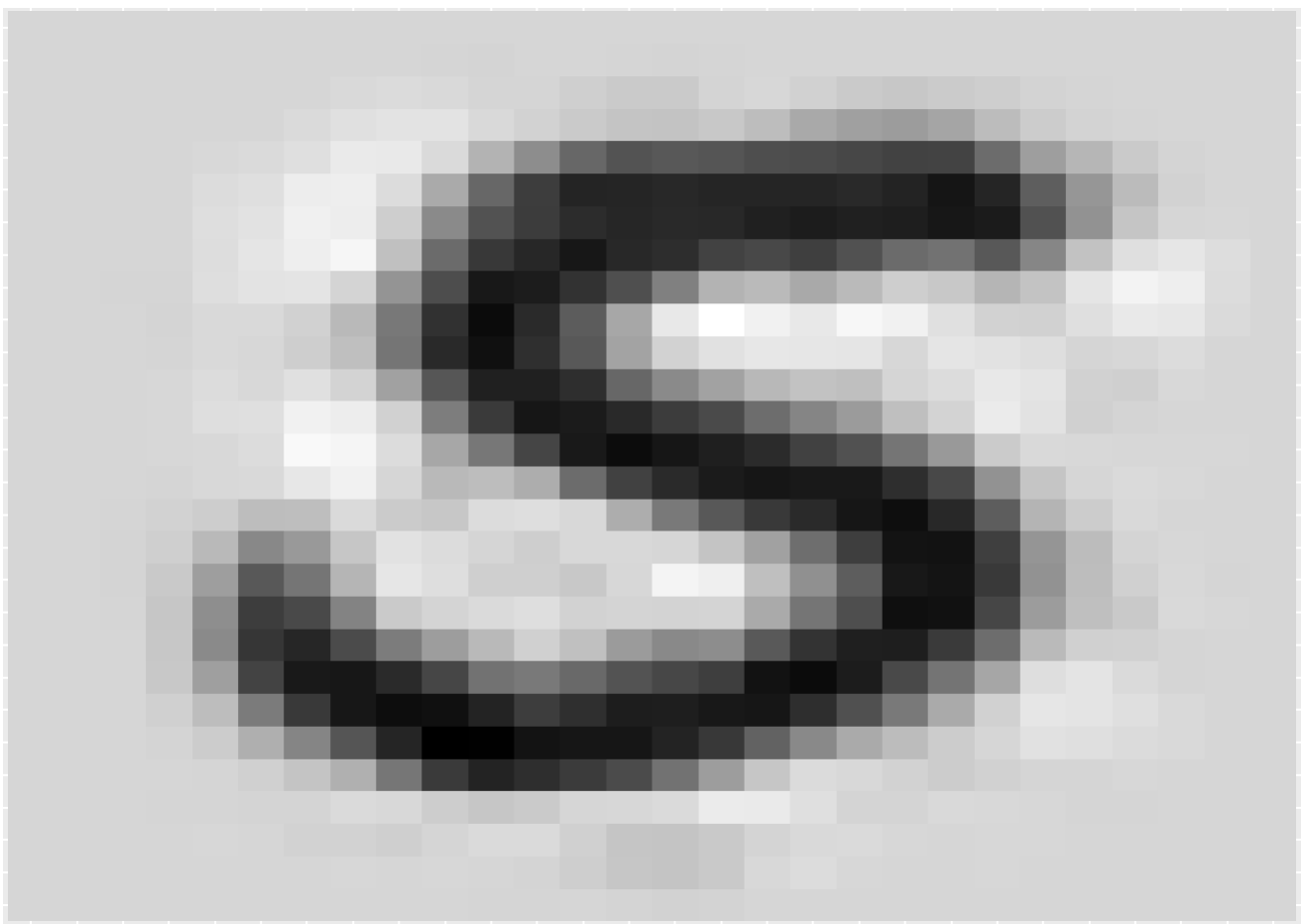
```r
#find the number of PCs needed for all the labels to return correctly
for(x in 1:ncol(train)){
   pc.use<-x
   train_trunc <- data.frame(round(train_pca_sub$x[,1:pc.use] %*% t(train_pca_sub$rotation[,1:pc.use]),0))
   if(all.equal(train_trunc$label,train$label[1:1000])==TRUE){
     PCs_needed<-pc.use
     break
   }
}
cat("Number of PCs for accurate labels:",PCs_needed)
```

```
Number of PCs for accurate labels: 567
```

```r
pc.use <- 576 # number of PCs needed to correctly assign labels and make background white
train_trunc <- data.frame(round(train_pca_sub$x[,1:pc.use] %*% t(train_pca_sub$rotation[,1:pc.use]),0))
draw_digit(train_trunc,testrow)
```



```r
train_pca_low<-prcomp(train[1:1000,], center=FALSE, rank. = 75)
test<-data.frame(train_pca_low$x[testrow,] %*% t(train_pca_low$rotation))
draw_digit(test,1)
```

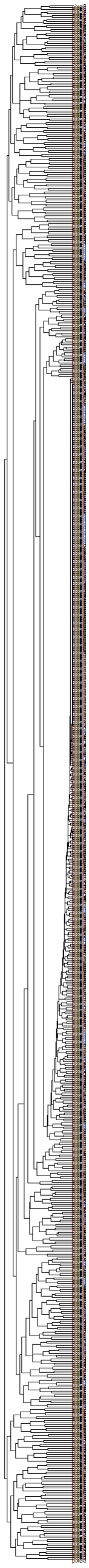


Question 2.

Draw a tree of the pixels, and see if you can explain the results based on geometry of the pixels (how far apart are they in the 2-d space). Try to Explain the PCA results in light of this.

By creating the dendrogram, we can see that the pixels with little to no data in them (mostly the edges of the images) cluster together and make up about 25% of the data. This is consistent with the above finding that 75% of the PCs were required to recreate the images.

```
Warning: package 'ggdendro' was built under R version 4.2.2

Warning: package 'dendextend' was built under R version 4.2.2


---------------------
Welcome to dendextend version 1.16.0
Type citation('dendextend') for how to cite the package.

Type browseVignettes(package = 'dendextend') for the package vignette.
The github page is: https://github.com/talgalili/dendextend/

Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
You may ask questions at stackoverflow, use the r and dendextend tags:
     https://stackoverflow.com/questions/tagged/dendextend

    To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
---------------------


Attaching package: 'dendextend'

The following object is masked from 'package:ggdendro':

    theme_dendro

The following object is masked from 'package:stats':

    cutree
```



Question 3.

Can you use some of the tools you have learnt to build a classifier,so if you get a new set of pixels you can predict what is in the picture. This is a the start of a real project, but you don't have all the tools (such as neural networks) which might be more suited for this task. Split your dataset into two (a training set and a test set), build your classifier and figure out how well it does with the test data in predicting the digits. Define the sensitivity and specificity of your classifier. How well does it recognize your own handwriting? (make sure your handwriting is not in the training set)

```
#split data into 70% test and 30% train
split1<- sample(c(rep(0, 0.7 * nrow(train)), rep(1, 0.3 * nrow(train))))
hw_train<-train[split1 == 0,]
hw_test<-train[split1 == 1,]
```

This will be be addressed using three different levels of algorithmic complexity.

3.a. Machine learning algorithms/neural networks are the obvious choice for this task, but first we will try to classify by simply using the average for each digit and correlating against that. Using this method is about 80% accurate. When comparing to the handwriting samples I created, it was limited by the fact that there was only a single comparison, between the average and the sample for each digit. It was able to identify five of the digits correctly.

3.b. The results of the correlative approach seemed insufficient, so using a machine learning model with PCA was the next step. By reducing dimensions and using LDA, we hope to classify each digit using a smaller number of the PC values instead of all 784 pixel columns. However, the performance of this model is poor despite various attempts to optimize, including using up to 500 PCs. The specificity and recall (the term sensitivity only applies to binary classification) were both far below acceptable standards, approximately between 0 and 0.2 depending on the digit being evaluated.

3.c. The PCA model was disappointing as well. Thus, a more proper machine learning algorithm seems to be required. We next evaluate a random forest model. This model uses much less time and lines of code, and has an accuracy of 95% when applied to the test set. However, it is completely unable to identify any of the experimental handwritten digit samples. When applied to the averages for each digit, it can correctly identify 4 out of the 10. The failure to identify the handwriting samples may be due to the scale of the images in both position and intensity - the amount of whitespace outside the writing is not exactly the same, and the intensity of the black color is less in the samples. It may perform better if each image in both sets had any row or column in the 28x28 image with a maximum value below a certain threshold removed, and with the color normalized so that the darkest black part is always equal to the maximum.

As a final observation, the basic correlative approach performed surprisingly well compared to more advanced algorithmic techniques.

3.a. Correlation Based Classification

```
#create digit averages from the training set

#create empty dataframe for the averages
digit_averages<-train[FALSE,]
#loop to get the averages for each digit 0-9
for(x in 0:9){
  #subset the data for the digit
  digit_subset<- hw_train[which(hw_train[,1]==x),]
  #average the columns
  digit_subset<-colMeans(digit_subset)
  #add it to the dataset of averages
  digit_averages<-rbind(digit_averages,digit_subset)
}
#rename the rows to the digit they represent, otherwise the labels start at 1 instead of 0
row.names(digit_averages)<-0:9
colnames(digit_averages)<-colnames(train)
#call the function on the average data for the digit of your choice
```

```
#sample a row from the test set
sample<-sample(1:nrow(hw_test),1)
#store the correlation and digit with the highest correlation
highest_cor<-0
digit<-as.integer()
#compare the correlation of each average digit to the test
for(x in 1:10){
  test<-as.numeric(hw_test[sample,2:785])
  avg<-as.numeric(digit_averages[x,2:785])
  test_cor<-cor(test, avg)
  if(test_cor>highest_cor){
    highest_cor<-test_cor
    digit<-digit_averages[x,"label"]
  }
}
#print the guess, correct answer, and correlation
cat("Guess:",digit,"\n")
```

```
Guess: 5
```

```
cat("Actual:",hw_test[sample,"label"],"\n")
```

```
Actual: 5
```

```
cat("Correlation:",highest_cor,"\n")
```

```
Correlation: 0.5173597
```

```
#function to do the above, returns true or false and the test information
test_classifier<-function(test_data,digit_averages){
  sample<-sample(1:nrow(test_data),1)
  highest_cor<-0
  digit<-as.integer()
  for(x in 1:10){
    test<-as.numeric(test_data[sample,2:785])
    avg<-as.numeric(digit_averages[x,2:785])
    test_cor<-cor(test, avg)
    if(test_cor>highest_cor){
      highest_cor<-test_cor
      digit<-digit_averages[x,"label"]
    }
  }
  result<-c(digit,test_data[sample,"label"],digit==test_data[sample,"label"],highest_cor)
  return(result)
}
```

```
#test the function
test_classifier(hw_test,digit_averages)
```

```
[1] 5.0000000 5.0000000 1.0000000 0.7884396
```

```
#test accuracy of the method
count<-0
total<-100
all_results<-test_classifier(hw_test,digit_averages)
for(x in 1:total){
  result<-test_classifier(hw_test,digit_averages)
  all_results<-rbind(all_results,result)
  if(result[3]){
    count<-count+1
    #cat(count)
  }
}
success_percent<-count/total
cat(success_percent*100,"% Correct")
```

```
75 % Correct
```

```
colnames(all_results)<-c("Prediction","Actual","Accuracy","Correlation")
head(all_results, 10)
```

```
            Prediction Actual Accuracy Correlation
all_results          3      3        1   0.7752446
result               0      0        1   0.6150235
result               2      2        1   0.6584950
result               6      6        1   0.6891298
result               6      6        1   0.6645137
result               3      3        1   0.7083832
result               6      6        1   0.8036371
result               6      6        1   0.6203995
result               0      0        1   0.6206056
result               0      0        1   0.8038243
```

```
## average over a small square (fac x fac)
ave_by_fac <- function(i1,fac,ii,jj){
  ave=0;
  cnt=0;
  for(i in c(1:fac)){
    for(j in c(1:fac)){
      cnt = cnt +1;
      x = (ii-1)*fac+i;
      y = (jj-1)*fac+j;
      ##         cat("i,j,ii,jj,x,y=",i,j,ii,jj,x,y,"\n");
      ave = ave+     i1[x,y];
    }}
  ave = ave/cnt;
  return(ave);
}
```

```r
## function I wrote to scale down a square image to a 28 x 28 image
## uses the averaging function above
scale_down_image <- function(img_in) {
  ## fac is the factor by which you have to scale the image to become a
  ## 28 x 28 square
  fac <- as.integer(dim(img_in)[1]/28);
  im_out <- matrix(0,nrow=28,ncol=28);
  for(i in c(1:28)){
    for(j in c(1:28)){
      im_out[i,j] = ave_by_fac(img_in,fac,i,j);
    }}
  return(im_out);
}
```

```r
#Get data
library(png)
library(vctrs)
```

```
Warning: package 'vctrs' was built under R version 4.2.2
```

```r
library(ggplot2)
library(reshape2)
```

```r
#function to take png image and convert it to same format as train.csv data
print_HW_digit<-function(img, label){

  #apply image scaling function
  img_scaled<-scale_down_image(img[,,2])

  #rescale values in the data to match given data, 0=white, 255=black
  img_scaled<-abs(img_scaled-1)
  img_scaled<-img_scaled-min(img_scaled)
  img_scaled<-img_scaled*255
  img_scaled<-round(img_scaled,0)
  #transpose data into correct orientation
  img_scaled<-t(img_scaled)

  #create the label as a dataframe
  label<-data.frame(label)

  #melt the image data so it is in long format
  img_m<-melt(img_scaled)
  #select only the values, excluding the x y coordinates
  img_m<-img_m$value
  #convert the linearized data into a data frame and transpose it so it is a row not a column
  img_lin<-data.frame(img_m)
  img_lin<-t(img_lin)
  #put the label in the first column
  img_lab<-cbind(label, img_lin)
  #label the columns and the row
  colnames(img_lab)<-colnames(train)
  rownames(img_lab)<-label
  #return the transformed data
  return(img_lab)
}
```

```r
#create empty dataframe to store results
HW_digits<-train[FALSE,]
#call the function for each digit and store the results
image<-print_HW_digit(readPNG("zero.png"),"0")
HW_digits<-rbind(HW_digits,image)
image<-print_HW_digit(readPNG("one.png"),"1")
HW_digits<-rbind(HW_digits,image)
image<-print_HW_digit(readPNG("two.png"),"2")
HW_digits<-rbind(HW_digits,image)
image<-print_HW_digit(readPNG("three.png"),"3")
HW_digits<-rbind(HW_digits,image)
image<-print_HW_digit(readPNG("four.png"),"4")
HW_digits<-rbind(HW_digits,image)
image<-print_HW_digit(readPNG("five.png"),"5")
HW_digits<-rbind(HW_digits,image)
image<-print_HW_digit(readPNG("six.png"),"6")
HW_digits<-rbind(HW_digits,image)
image<-print_HW_digit(readPNG("seven.png"),"7")
HW_digits<-rbind(HW_digits,image)
image<-print_HW_digit(readPNG("eight.png"),"8")
```

```
HW_digits<-rbind(HW_digits,image)
image<-print_HW_digit(readPNG("nine.png"),"9")
HW_digits<-rbind(HW_digits,image)
```

```
#test output of function with handwriting data
test_classifier(HW_digits,digit_averages)
```

```
[1] "5"                 "5"                 "TRUE"
[4] "0.420263727478807"
```

```
#test accuracy of method on handwriting data
digit_accuracy<-data.frame(matrix(ncol = 4, nrow = 10))
digit_names <- c(0:9)
rownames(digit_accuracy) <- digit_names
colnames(digit_accuracy) <-c("Correct Correlation", "Highest Correlation","Correct","Total")
count<-0
total<-100
digit_accuracy[,]<-0
for(x in 1:total){
  result<-test_classifier(HW_digits,digit_averages)
  y<-result[2]
  digit_count<-digit_accuracy[y,4]
  digit_accuracy[y,4]<-digit_count+1
  if(result[3]){
    count<-count+1
    digit_accuracy[y,3]<-digit_accuracy[y,3]+1
    digit_accuracy[y,1]<-result[4]
  }
  if(result[4]>digit_accuracy[y,2]){
    digit_accuracy[y,2]<-result[4]
  }
}
success_percent<-count/total
cat(success_percent*100,"% Correct")
```

```
54 % Correct
```

```
digit_accuracy
```

	Correct Correlation	Highest Correlation	Correct	Total
0	0.328203206619366	0.328203206619366	15	15
1	0.492162218351691	0.492162218351691	10	10
2	0.420922621417808	0.420922621417808	7	7
3	0.392198106242828	0.392198106242828	11	11
4	0	0.321264753878853	0	10
5	0.420263727478807	0.420263727478807	11	11
6	0	0.238191463905309	0	9
7	0	0.0960134307167561	0	13
8	0	0.418489289780749	0	9
9	0	0.289234230606411	0	5

3.b. PCA model

```
#PCA model

#for this method we will perform the PCA dimensionality reduction on the testing and training data, excluding the labels
library(MASS)
train_pca<-prcomp(hw_train[2:785], center=FALSE)
test_pca<-prcomp(hw_test[2:785], center=FALSE)
```

```
#once reduced to principal components, the labels are added back in
train_pca_df<-data.frame(as.numeric(hw_train$label),cbind(train_pca$x))
colnames(train_pca_df)[1]<-"label"

test_pca_df<-data.frame(as.numeric(hw_test$label),cbind(test_pca$x))
colnames(test_pca_df)[1]<-"label"

#the model includes up to PC10
pca_model<-lda(label~PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC9+PC10, data=train_pca_df)
```

```
#use the model to predict the values in the test set
hw_pca_predict<-predict(pca_model, newdata=test_pca_df)
```

```
#compare the counts of each number predicted
summary(hw_pca_predict$class)
```

```
   0    1    2    3    4    5    6    7    8    9
 915  731 1420 1151 1254 2314 1388 1004 1174 1249
```

```r
table(hw_test$label)
```

```
   0    1    2    3    4    5    6    7    8    9
1232 1402 1270 1291 1229 1203 1173 1277 1244 1279
```

```r
#count how many the model got right
comp<-hw_pca_predict$class==hw_test$label
table(comp)
```

```
comp
FALSE  TRUE
10597  2003
```

```r
# Evaluate the model by performance metrics
library(mltest)
model_eval <- ml_test(hw_pca_predict$class, hw_test$label, output.as.table = TRUE)
model_eval
```

```
  balanced.accuracy         DOR        F0.5           F1           F2       FDR
0         0.3436843 0.001779344 0.001022077 0.0009315324 0.0008557248 0.9989071
1         0.3676388 0.005891178 0.003467406 0.0028129395 0.0023663038 0.9958960
2         0.4946206 0.955225617 0.339568345 0.3509293680 0.3630769231 0.6676056
3         0.3244222 0.021969077 0.013570823 0.0131040131 0.0126682502 0.9860990
4         0.3774986 0.246450632 0.125700560 0.1264599275 0.1272285251 0.8748006
5         0.4191051 0.518782627 0.220862415 0.2627239124 0.3241650295 0.8003457
6         0.5149810 1.134390173 0.352416357 0.3701679032 0.3898026316 0.6585014
7         0.3717679 0.150787771 0.081239373 0.0754055239 0.0703534031 0.9143426
8         0.3334419 0.059018385 0.034511785 0.0339123242 0.0333333333 0.9650767
9         0.4343600 0.526349598 0.231872510 0.2302215190 0.2285938727 0.7670136
        FNR       FOR       FPR geometric.mean      Jaccard           L
0 0.9991883 0.3807609 0.3134431     0.02360657 0.0004659832 0.002589588
1 0.9978602 0.4115916 0.2668622     0.03960768 0.0014084507 0.008018372
2 0.6283465 0.3426363 0.3824123     0.47909151 0.2128043282 0.971866175
3 0.9876065 0.3908645 0.3635490     0.08881358 0.0065952185 0.034090296
4 0.8722539 0.3673749 0.3727489     0.28307049 0.0674978504 0.342713651
5 0.6159601 0.3247152 0.5458296     0.41763565 0.1512274959 0.703589299
6 0.5959079 0.3137343 0.3741302     0.50290062 0.2271202683 1.080084170
7 0.9326547 0.3832046 0.3238095     0.21339700 0.0391799544 0.207978258
8 0.9670418 0.3800948 0.3660743     0.14454428 0.0172486327 0.090031445
9 0.7724785 0.3659259 0.3588015     0.38195084 0.1300849352 0.634115249
     lambda         MCC          MK       NPV          OP   precision
0 1.4553612 -0.34452305 -0.37966801 0.6192391 -0.83867002 0.001092896
1 1.3610813 -0.32843737 -0.40748768 0.5884084 -0.83521136 0.004103967
2 1.0174206 -0.01049716 -0.01024196 0.6573637 -0.08964065 0.332394366
3 1.5517401 -0.36383077 -0.37696354 0.6091355 -0.80283001 0.013900956
4 1.3905976 -0.24358505 -0.24217555 0.6326251 -0.50263015 0.125199362
5 1.3562314 -0.14224475 -0.12506088 0.6752848  0.07530136 0.199654278
6 0.9521276  0.02884216  0.02776427 0.6862657 -0.05635792 0.341498559
7 1.3792780 -0.27624304 -0.29754726 0.6167954 -0.65988286 0.085657371
8 1.5254813 -0.33909021 -0.34517145 0.6199052 -0.74218935 0.034923339
9 1.2047416 -0.13210716 -0.13293954 0.6340741 -0.31722315 0.232986389
        recall specificity      Youden
0 0.0008116883   0.6865569 -0.31263138
1 0.0021398003   0.7331378 -0.26472237
2 0.3716535433   0.6175877 -0.01075872
3 0.0123934934   0.6364510 -0.35115551
4 0.1277461351   0.6272511 -0.24500276
5 0.3840399002   0.4541704 -0.16178975
6 0.4040920716   0.6258698  0.02996190
7 0.0673453406   0.6761905 -0.25646418
8 0.0329581994   0.6339257 -0.33311611
9 0.2275215012   0.6411985 -0.13128000
```

```r
#repeat this process for the handwritten digit samples
HW_digit_pca<-prcomp(HW_digits[,2:785], center = FALSE)
HW_digit_pca_df<-data.frame(cbind(HW_digit_pca$x[,1:10]),as.numeric(HW_digits$label))
colnames(HW_digit_pca_df)[11]<-"label"

hw_digit_pca_predict<-predict(pca_model, newdata=HW_digit_pca_df)
digit_model_eval<-ml_test(hw_digit_pca_predict$class,HW_digits$label, output.as.table = TRUE)
```

3.c. Random Forest model

```r
#random forest method
#set the parameters for the random forest model
library(randomForest)
```

```
Warning: package 'randomForest' was built under R version 4.2.2

randomForest 4.7-1.1

Type rfNews() to see new features/changes/bug fixes.


Attaching package: 'randomForest'

The following object is masked from 'package:ggplot2':

    margin
```

```r
numTrees <- 25
rf_labels<-as.factor(hw_train$label)
rf_test<-hw_test[,-1]
rf_train<-hw_train[,-1]
#create the model, testing it on the test data
rf <- randomForest(rf_train, rf_labels, xtest=rf_test,
                   ntree=numTrees)
```

```r
#count how many the model got right by comparing the predictions to the labels
comp<-data.frame(rf$test$predicted,hw_test$label,rf$test$predicted==hw_test$label)
colnames(comp)<-c("Predicted","Actual","Accuracy")
#calculate and report the performance metrics and summary
rf_comp<-table(comp$Accuracy)
rf_accuracy<-rf_comp[2]/(rf_comp[1]+rf_comp[2])*100
head(as.matrix(comp),20)
```

```
   Predicted Actual Accuracy
2  "0"       "0"    "TRUE"
3  "1"       "1"    "TRUE"
10 "3"       "3"    "TRUE"
13 "1"       "1"    "TRUE"
14 "3"       "3"    "TRUE"
19 "7"       "7"    "TRUE"
33 "4"       "4"    "TRUE"
35 "2"       "2"    "TRUE"
38 "1"       "1"    "TRUE"
39 "1"       "1"    "TRUE"
44 "9"       "4"    "FALSE"
46 "6"       "6"    "TRUE"
47 "3"       "3"    "TRUE"
50 "4"       "4"    "TRUE"
52 "5"       "5"    "TRUE"
56 "2"       "2"    "TRUE"
59 "9"       "9"    "TRUE"
60 "1"       "1"    "TRUE"
64 "0"       "0"    "TRUE"
70 "0"       "0"    "TRUE"
```

```r
rf_comp
```

```
FALSE  TRUE
  596 12004
```

```r
cat("\nRandom Forest accuracy:",rf_accuracy[[1]],"%")
```

```
Random Forest accuracy: 95.26984 %
```

```r
#applying the RF classifier to the handwritten data
rf2<- randomForest(rf_train, rf_labels, xtest=HW_digits[-1],
                   ntree=numTrees)
```

```r
#count how many the model got right by comparing the predictions to the labels
comp2<-data.frame(rf2$test$predicted,rf2$test$predicted==HW_digits$label, row.names = HW_digits$label)
colnames(comp2)<-c("Predicted","Accuracy")
#calculate and report the performance metrics and summary
rf_comp2<-table(comp2$Accuracy)
```

```
rf_accuracy2<-rf_comp2[2]/(rf_comp2[1]+rf_comp2[2])*100
as.matrix(comp2)
```

```
  Predicted Accuracy
0 "0"       "TRUE"
1 "5"       "FALSE"
2 "6"       "FALSE"
3 "5"       "FALSE"
4 "8"       "FALSE"
5 "5"       "TRUE"
6 "8"       "FALSE"
7 "5"       "FALSE"
8 "5"       "FALSE"
9 "5"       "FALSE"
```

```
rf_comp2
```

```
FALSE  TRUE
    8     2
```

```
cat("\nRandom Forest accuracy:",rf_accuracy2[[1]],"%")
```

```
Random Forest accuracy: 20 %
```

Question 4.

You can try simple things like take average of all data for each number and then take a “dot” product with your test set, and identify the pixels. This might work, maybe for some digits, and not others.

Here we will see which of the digit averages is best classified by the Random Forest model. This model is only about to correctly identify 4 of the 10 digits.

```
#applying the RF classifier to the handwritten data
rf3<- randomForest(rf_train, rf_labels, xtest=digit_averages[-1],
                   ntree=numTrees)
```

```
#count how many the model got right by comparing the predictions to the labels
comp3<-data.frame(rf3$test$predicted,rf3$test$predicted==HW_digits$label,row.names = HW_digits$label)
colnames(comp3)<-c("Predicted","Accuracy")

#calculate and report the performance metrics and summary
rf_comp3<-table(comp3$Accuracy)
rf_accuracy3<-rf_comp3[2]/(rf_comp3[1]+rf_comp3[2])*100
as.matrix(comp3)
```

```
  Predicted Accuracy
0 "0"       "TRUE"
1 "8"       "FALSE"
2 "2"       "TRUE"
3 "8"       "FALSE"
4 "8"       "FALSE"
5 "8"       "FALSE"
6 "6"       "TRUE"
7 "8"       "FALSE"
8 "8"       "TRUE"
9 "8"       "FALSE"
```

```
rf_comp3
```

```
FALSE  TRUE
    6     4
```

```
cat("\nRandom Forest accuracy:",rf_accuracy3[1],"%")
```

```
Random Forest accuracy: 40 %
```