

Ryan-Oblivion / Biostatistics_final

Public

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#)[main](#) [...](#)

Biostatistics_final / final_project_biostatistics.ipynb

**Ryan-Oblivion** final project ...[History](#)[1 contributor](#)

1607 lines (1607 sloc) | 225 KB

...

```
In [ ]: # NOTE: THIS PART REUSES THE MIDTERM
# CELL 5 IS WHERE THE FINAL PROJECT ANALYSIS BEGINS
```

```
In [1]: # cell1
data <- read.csv("train.csv.gz")
# Taking the train file and giving it the variable name data
#data

#length(data[,1])
#42000

pixels <- data.frame(data[,-1])
# we want to remove the first column from the file, so we only have pixels
head(data,10)
#smoothScatter(pixels)
```

label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	p
1	0	0	0	0	0	0	0	0	0	...	0	
0	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	...	0	
0	0	0	0	0	0	0	0	0	0	...	0	
0	0	0	0	0	0	0	0	0	0	...	0	
7	0	0	0	0	0	0	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	...	0	
5	0	0	0	0	0	0	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	...	0	

```
In [2]: #cell2
# This cell is to test out how to turn one row into a matrix and turn it
# into the image
# Then we will make a function and a for loop to do this for every row

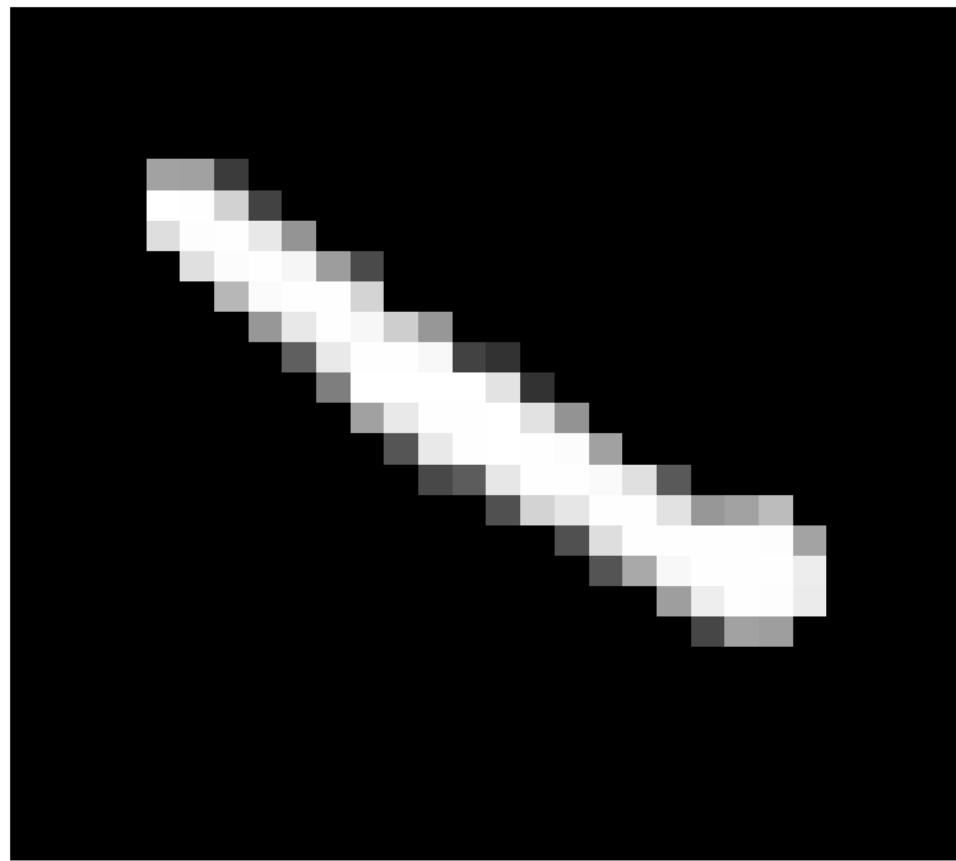
label1 <- pixels[1,]
# now getting the first row into a variable called label1
# this is just to visualize the frist row as its own data set

#mode(label1)
mode(label1) = "numeric"
# now we need to make the mode of this label1 data into numeric so it works
# the image method

my_label1 <- matrix(label1, nrow=28, byrow= TRUE)
# this turns our label into a 28x28 matrix

#par(mar= c(5,5,5,5))
#image(my_label1, axes = FALSE, col = grey.colors(256), start = 0, end = 1)
```

```
Biostatistics_final/final_project_biostatistics.ipynb at main · Ryan-Oblivion/Biostatistics_final  
# and now we can print the values in the matrix as an image  
# where 0 is black and 256 is white, and anything inbetween as shades of  
# this prints the image also  
  
#my_label1  
  
# next we print the matrix  
  
#matplotlib((my_label1), type= "l")
```



In [3]:

```
#cell13  
# number 1  
  
# this cell is where we do the same as the above cell but for every row  
  
r <- c(1:length(pixels[,1]))  
# we want to make a variable that keeps track of how many rows are  
# in the pixels dataframe  
  
for (x in r){  
    # now we want to loop through each number in r, which represents the
```

```
# current row in the dataframe pixels

label1 <- pixels[x,]
# next we want to pass that entire row into the variable
# called label1

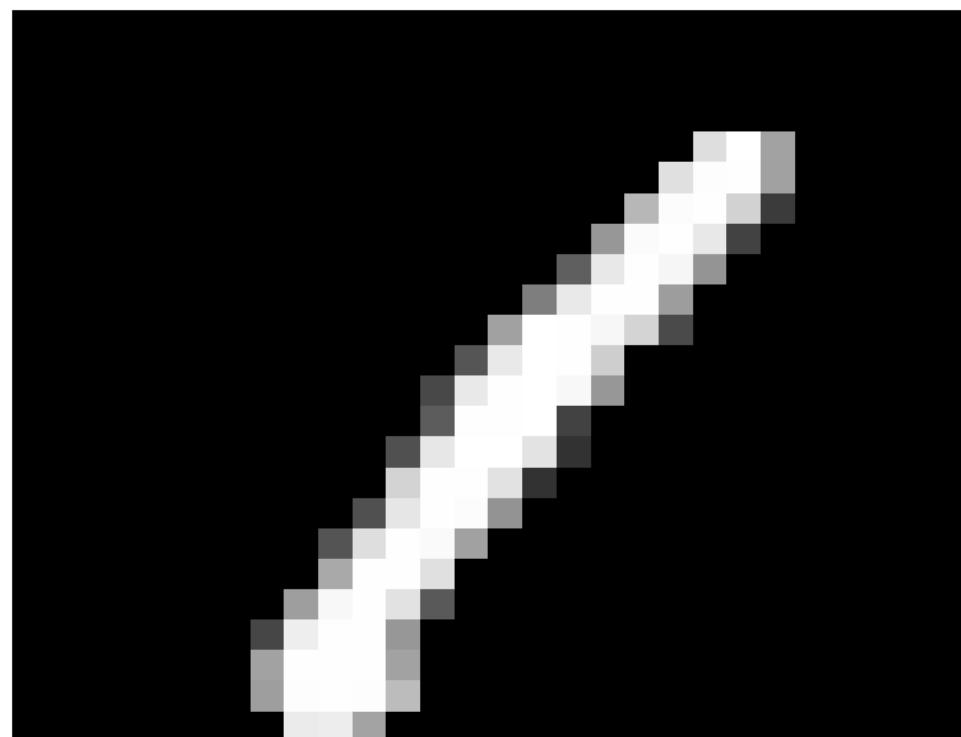
mode(label1) = "numeric"
# we need to change the mode of each number in that row to numeric
# so the image method can work with it

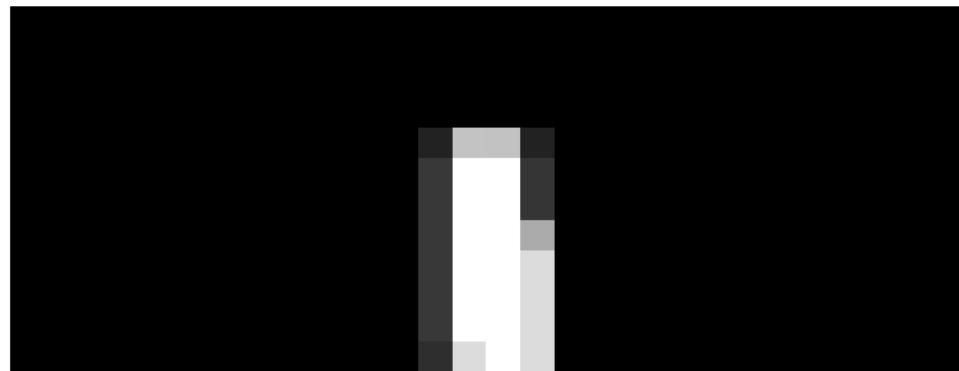
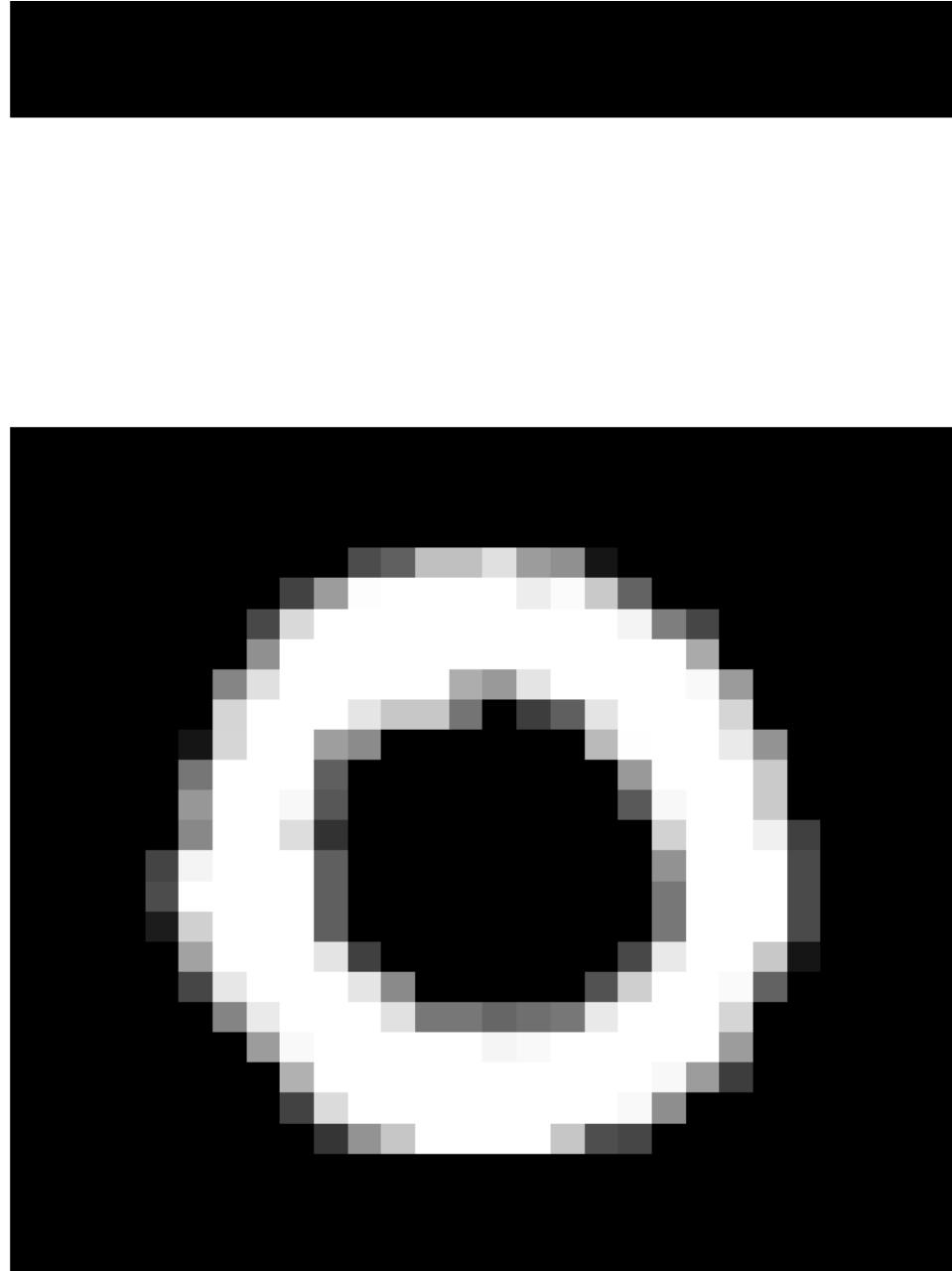
my_label1 <- matrix(rev(label1), nrow=28, byrow= FALSE)
# now we make that row into a 28x28 matrix called my_label1

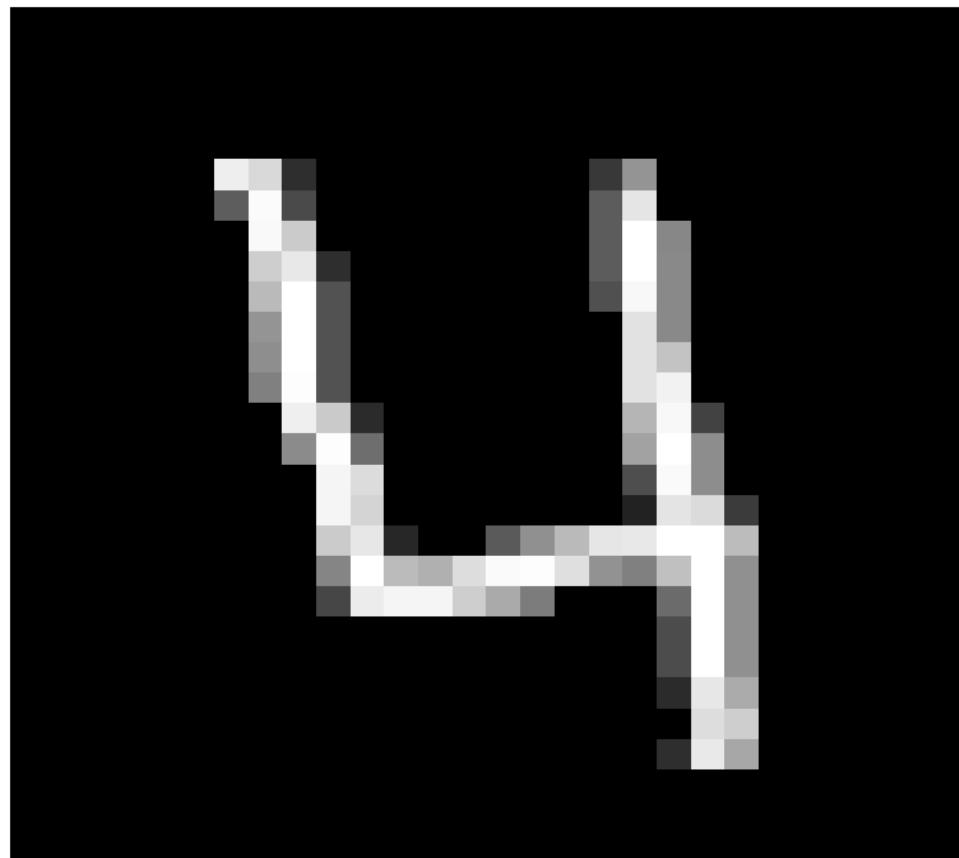
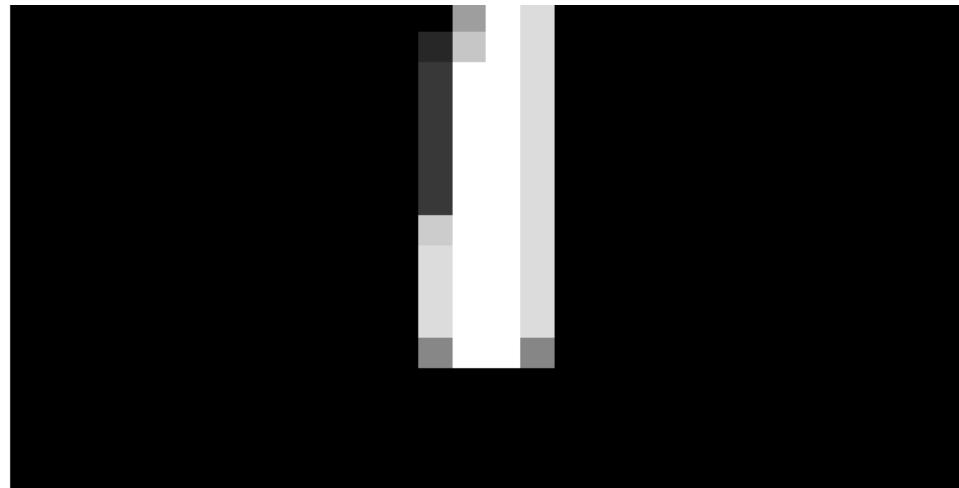
my_label1 <- apply(my_label1, 2, rev)

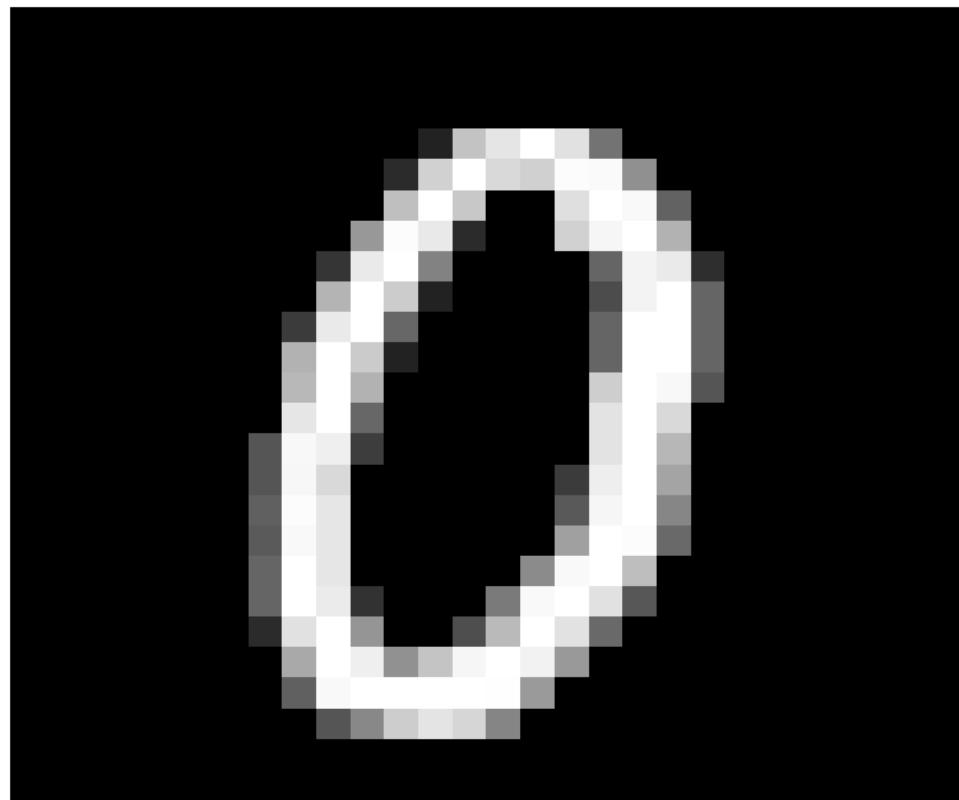
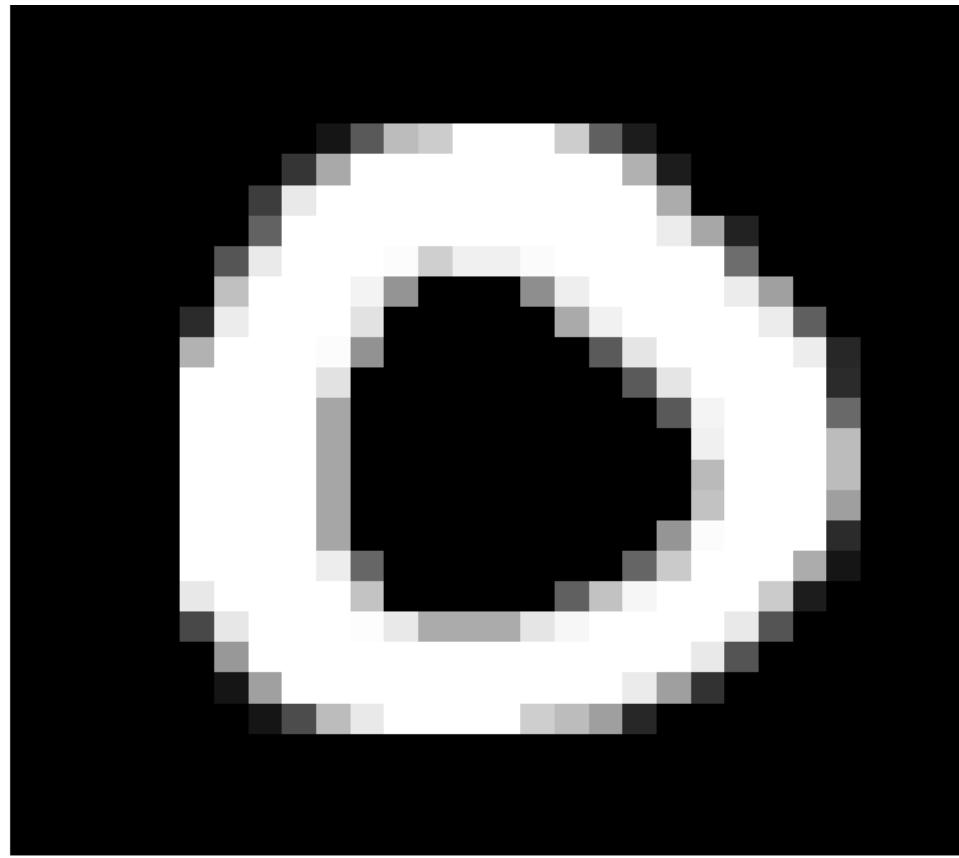
#par(mar= c(5,5,5,5))
image(my_label1, axes =FALSE, col = grey.colors(256, start =0, end=1
# now we use image to plot all the points in the matrix
# 0 being black and 256 being white, and everything inbetween being
# shades of grey
#my_label1
if (x ==10){ break }
# since there are 64k rows, we used a break statement at 10
# just to see how the first 10 hand written digits looked
# the code will run much longer if we had not done this
# in the next few cells we take the average of each number
# which is much better at combining all of the data into
# smaller parts

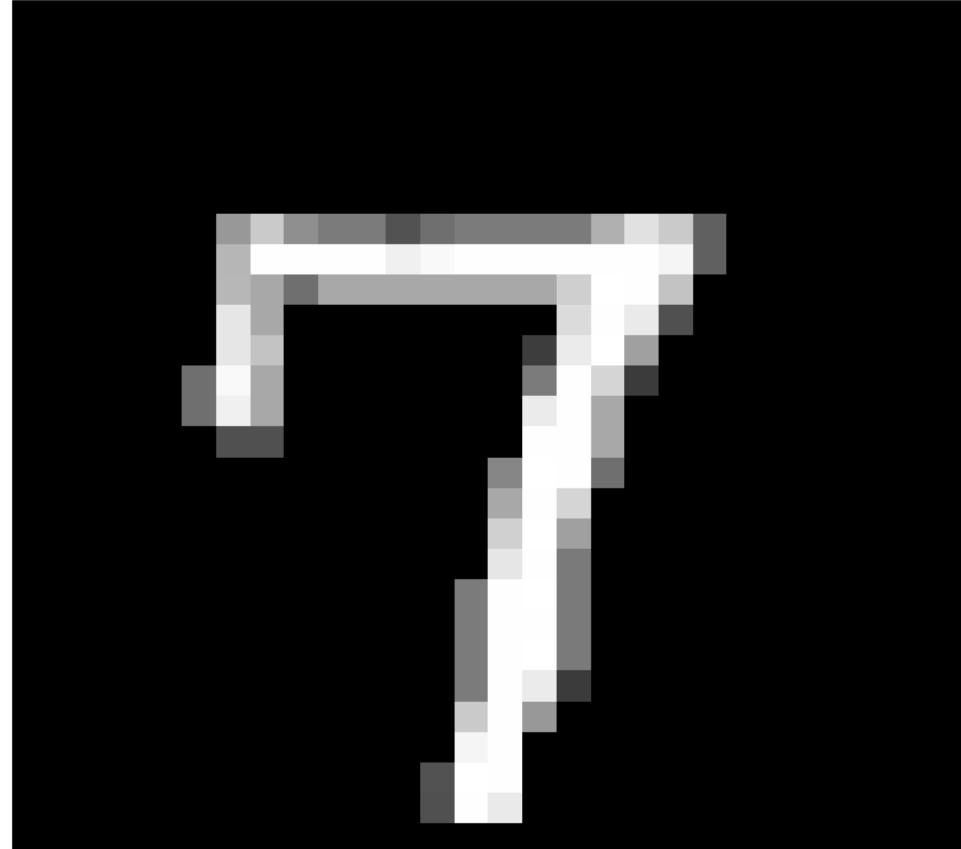
}
```

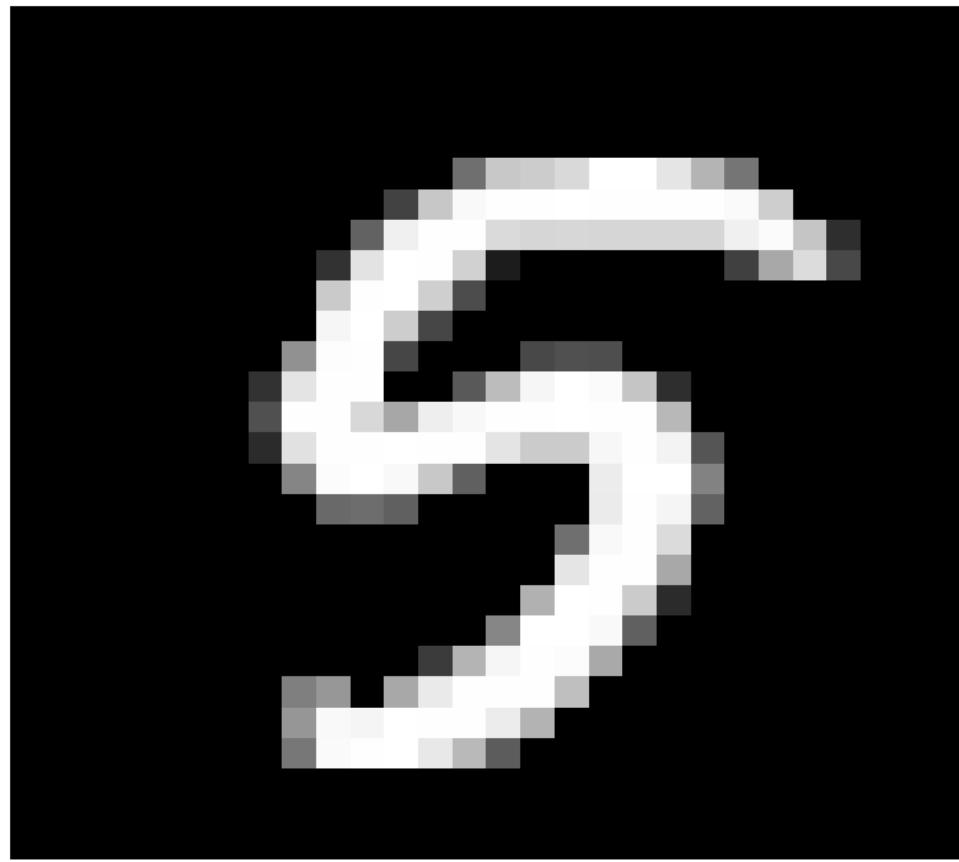
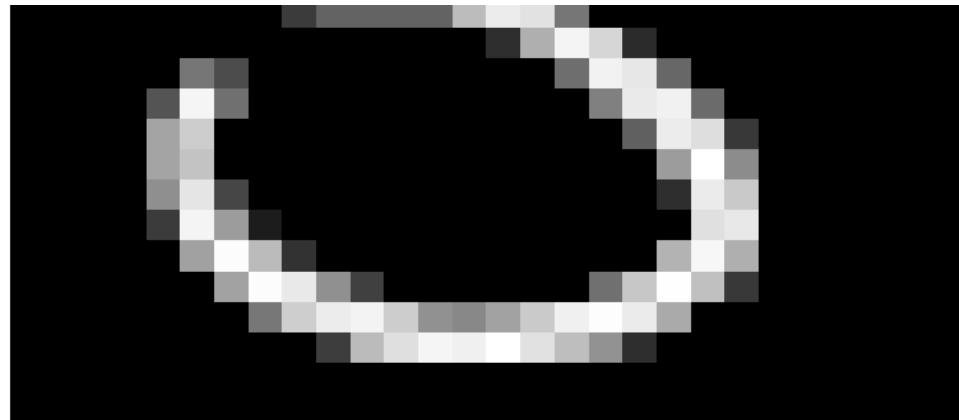


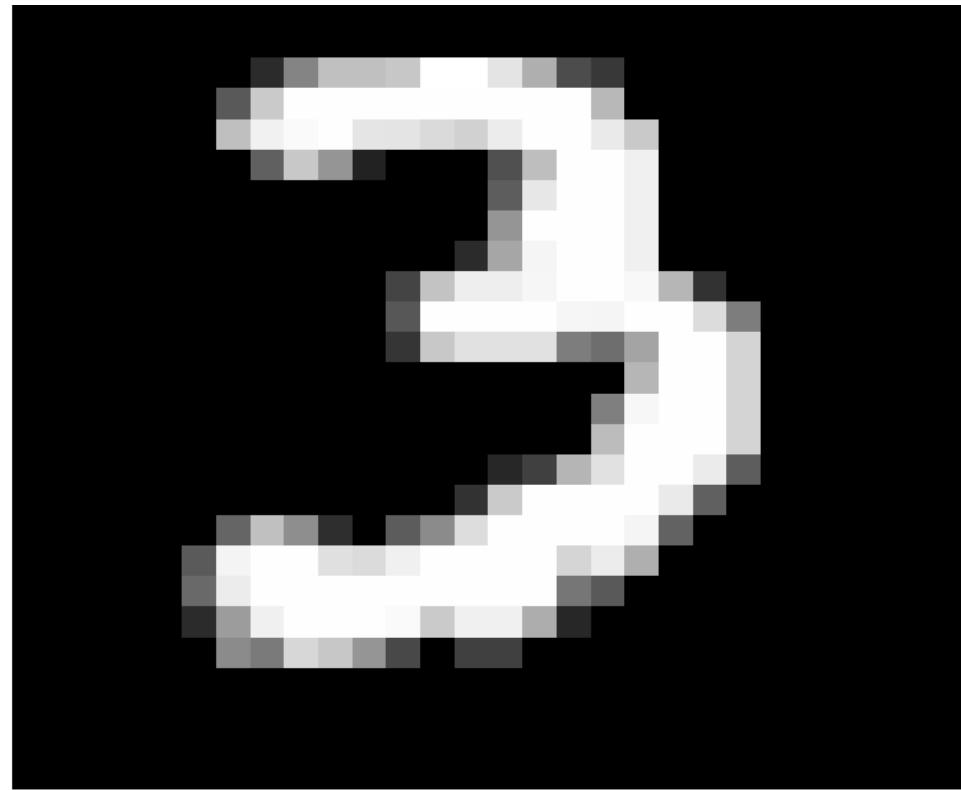












In [4]:

```
#cell14
# number 2
n <- c(0:9)
# we want to keep track of all the digits that correspond to
# a handwriting. from 0 to 9 as the variable n

for (x in n){
  # next we loop through each digit

  assign(paste0("label_",x), data[grep(x,data[,1]),])
  # we use assign to put all the data that we wish to grep into
  # a variable. we use paste0 to create variable names dynamically
  # then grep to get the digit which is the pattern that
  # the current loop is on from the first column. we then put
  # that entire row that matches the current digit into a variable
  # then the loop moves onto the next digit.

}
#label_0
#label_1
# here we show that it works for label_1 and before that we checked
# to see if it works for label_0 which it does.
# So this code will work for all label_(0-9)

#data[grep("0", data[,1]),]
```

In [5]:

```
#cell 5

# final project number 1
test <- data.frame(my_label1)
#test
# note: i am using "my_label1" variable from the 3rd cell. this variable
# uses code that gets information from the first 10 rows in the pixel da
# the last pixel data ends up being the digit "3"

# so the digit we are looking to run pca on is 3

list_to_remove <- c()

for (x in c(1:length(test))){

  if (var(test[,x]) == 0) {
    list_to_remove <- append(list_to_remove, x)
  }

}

list_to_remove
# I used this for loop to find all the columns that has a variance of 0
# the column number into the list_to_remove variable. now i will edit th
# matirx test and remove those columns

test_update <- test[,-(list_to_remove)]

test_update
# this is now only the data frame that consists of only columns with var
```

1. 1

2. 2

3. 3

4. 4

5. 25

6. 26

7. 27

8. 28

X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	5	36	26	0	0	0	0	0	0	0	0	0	0	0	0	1
67	87	215	231	33	0	0	0	0	0	0	0	0	0	0	0	1

50	223	253	253	136	0	0	0	0	0	0	0	0	0	0	0	0	0	30
176	253	253	253	70	0	0	0	0	0	0	0	0	0	0	0	0	0	149
148	253	253	191	6	0	0	0	0	0	0	0	0	0	0	0	0	0	78
78	253	253	183	0	0	0	0	0	0	0	0	0	0	0	0	0	0	;
16	244	253	223	27	0	0	0	0	0	0	8	24	14	0	0	0	0	1
0	152	253	253	67	0	0	0	0	0	0	150	253	142	0	0	0	0	1
12	223	253	253	186	7	0	0	0	0	0	194	253	220	5	0	0	0	1
12	223	253	253	253	153	4	0	0	0	0	194	253	219	99	78	28	21	
0	109	253	253	253	253	12	0	0	0	0	194	253	236	234	253	206	13	
0	4	47	172	253	253	120	0	0	0	0	53	235	253	253	253	253	25	
0	0	25	216	253	253	193	130	55	0	40	233	253	253	253	253	253	25	
0	0	0	112	234	253	253	253	237	122	97	253	240	224	224	224	22	2	
0	0	0	0	31	212	253	253	253	253	253	121	0	0	0	0	0	1	
0	0	0	0	0	30	214	253	253	253	253	185	7	0	0	0	0	1	
0	0	0	0	0	0	28	170	170	170	170	53	0	0	0	0	0	1	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	

In [6]:

```

length(label_1)

pca_label_1 <- prcomp(test_update[,c(2:length(test_update))], center = T
#summary(pca_label_1)

# from pc1- pc496? the standard deviation goes below 0.05 after that

#plot(pca_label_1x[,1],pca_label_1x[,2])

#biplot(pca_label_1, scale =0)
# use screeplot, find the pc cut off that contains 90% of the variance
# and convert it back to the data to see how the digits look

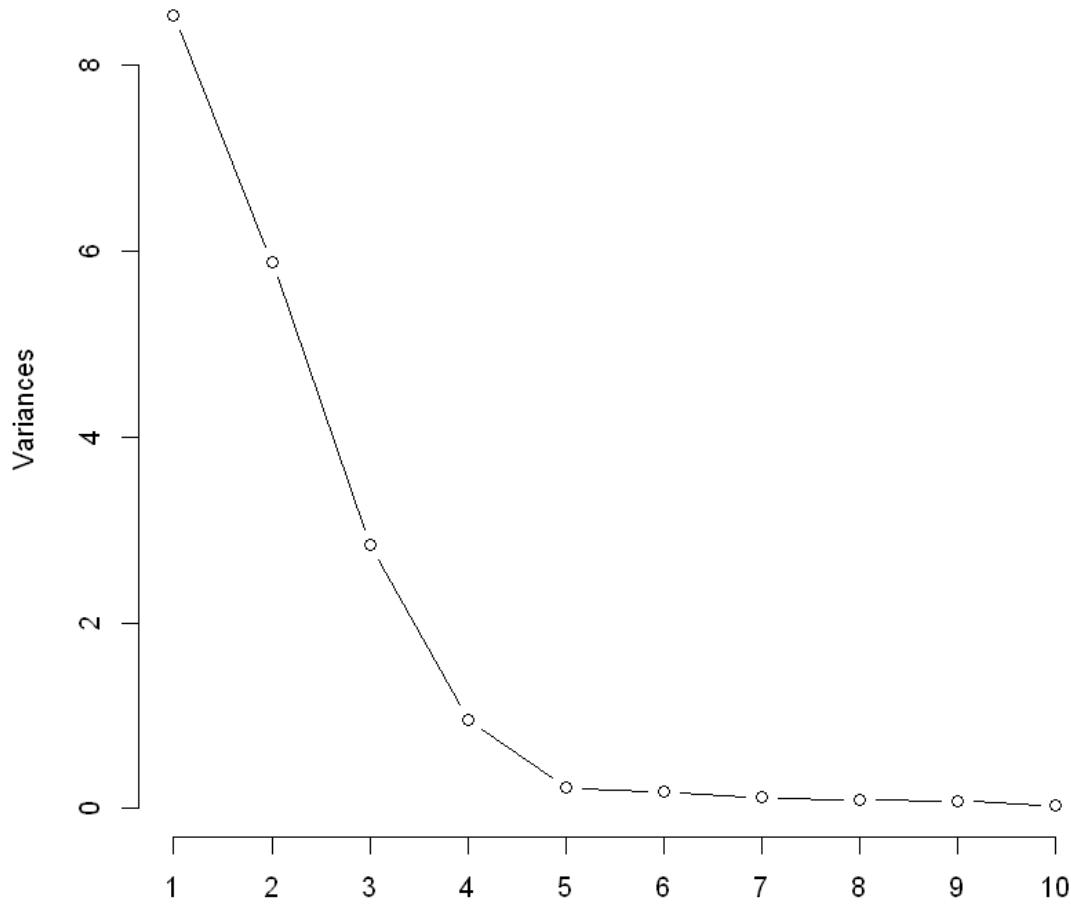
screeplot(pca_label_1, type = "lines")

#new_label_1 <- label_1[,c(2:496)]
#new_label_1

```

785

pca_label_1



In [7]:

```
standard_dev <- pca_label_1$sdev
# this is getting the standard deviation

var_label_1 <- standard_dev^2
# this is to get the variance of the pc
#var_label_1[1:10]

prop_var <- var_label_1/sum(var_label_1)
prop_var[1:3]
# this shows the proportion of variance explained

sum(prop_var[1:3])
# the first 3 pc (principal components) consist of 90% of the variance

# so now i need to take the first 35 pc and convert them back into the c
# data and see how the digit looks.
```

1. 0.449323565287185

2. 0.309497345718064

3. 0.149361985750576

0.908182896755825

In [8]:

```
# number 1

pc.use <- 4

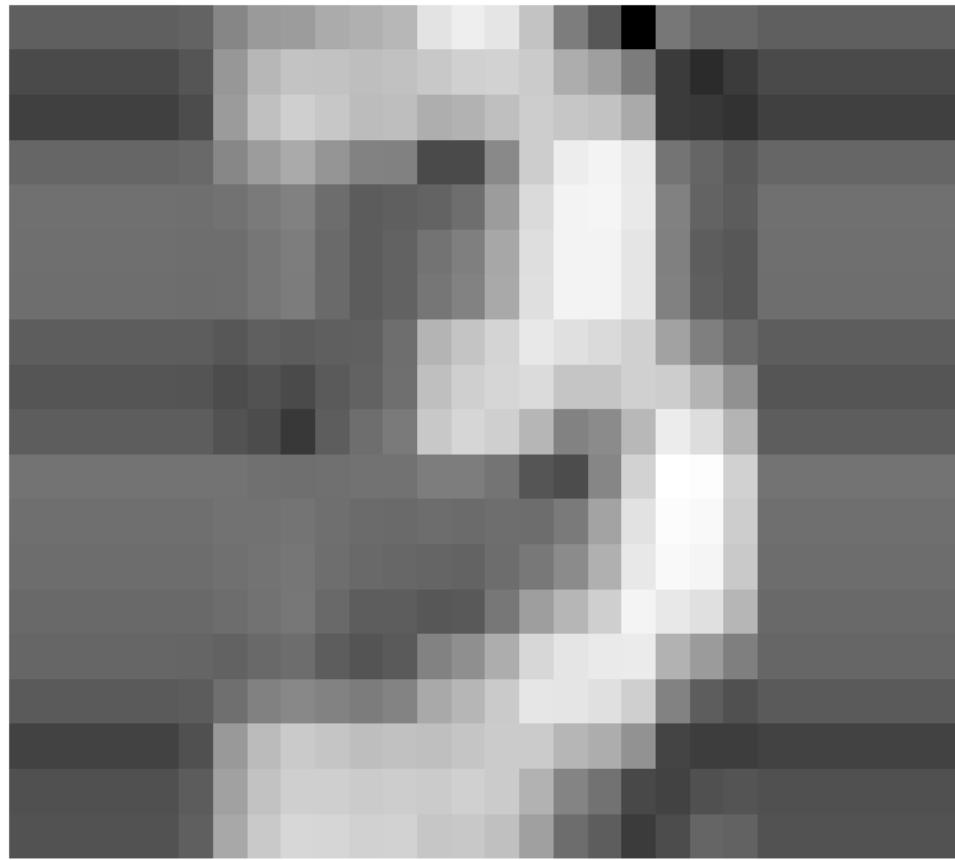
reconstructed <- pca_label_1$x[,1:pc.use] %*% t(pca_label_1$rotation[,1:

reconstructed
# this is the final matrix, within the variable reconstructed

image(reconstructed, axes =FALSE, col = grey.colors(256, start =0, end=1
# even though the above cell shows that 3 principal components account for
# 90% of the variance, i think using 4 is able to reproduce the digit re
# well
```

X6	X7	X8	X9	X10	X11	X12
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.6048609	-0.6352856	-0.7529194	-0.6427501	-0.54699252	-0.4957419	-0.44111247
0.5476336	0.4146480	0.2466798	-0.4225703	-0.57401439	-0.4407446	-0.39299063
1.3405545	1.1564015	0.9922401	-0.1620609	-0.49785690	-0.3792599	-0.37371643
1.7183426	1.4916277	1.3530434	-0.0540572	-0.44703767	-0.2977867	-0.32735642
1.6943396	1.4996481	1.2421715	-0.1505307	-0.61437851	-0.4939950	-0.43300597
1.5558440	1.3890948	1.0626934	-0.2431010	-0.72097520	-0.6195837	-0.50062432
1.5798620	1.4441968	1.1246396	-0.1301549	-0.64839389	-0.6271294	-0.51854009
1.2508881	1.4055468	1.0775944	0.5657874	-0.14822160	-0.6831950	-0.55047231
1.3023923	1.5323816	1.2381569	0.8645526	0.06421055	-0.6632490	-0.56560740
1.0922526	1.3794699	1.3853598	1.3612551	0.65300398	-0.3081666	-0.45154279
0.3905018	0.7791963	1.3940030	2.1718257	1.73478171	0.3470158	-0.28329655
-0.4492012	-0.1154588	0.8718181	2.1325264	2.13816448	0.8704012	-0.00499395
-0.6207145	-0.3757007	0.6651227	1.9782084	2.28846752	1.5023073	0.72436895
-0.9049414	-0.8120450	0.1237109	1.4949165	2.33832781	2.6120066	2.22950665
-0.7747248	-0.8636943	-0.8457336	-0.1472938	0.76679925	2.2288035	2.78199994
-0.5475356	-0.7546981	-0.8914260	-0.6249043	0.27778595	1.9939113	2.64941015
-0.5680083	-0.7144890	-0.8965326	-0.7487676	-0.18840910	0.8811843	1.32831858
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
0.7275113	0.7473491	0.8536929	0.6584437	0.53411468	0.4933435	0.44275862

```
Biostatistics_final/final_project_biostatistics.ipynb at main · Ryan-Oblivion/Biostatistics_final  
-0.1275113 -0.7473491 -0.8536929 -0.6584437 -0.53411468 -0.4933435 -0.44275862  
-0.7275113 -0.7473491 -0.8536929 -0.6584437 -0.53411468 -0.4933435 -0.44275862
```



In [9]:

```
# number 2

dist_recon <- dist(reconstructed)

# finding the distances of each row to use in the hclust function

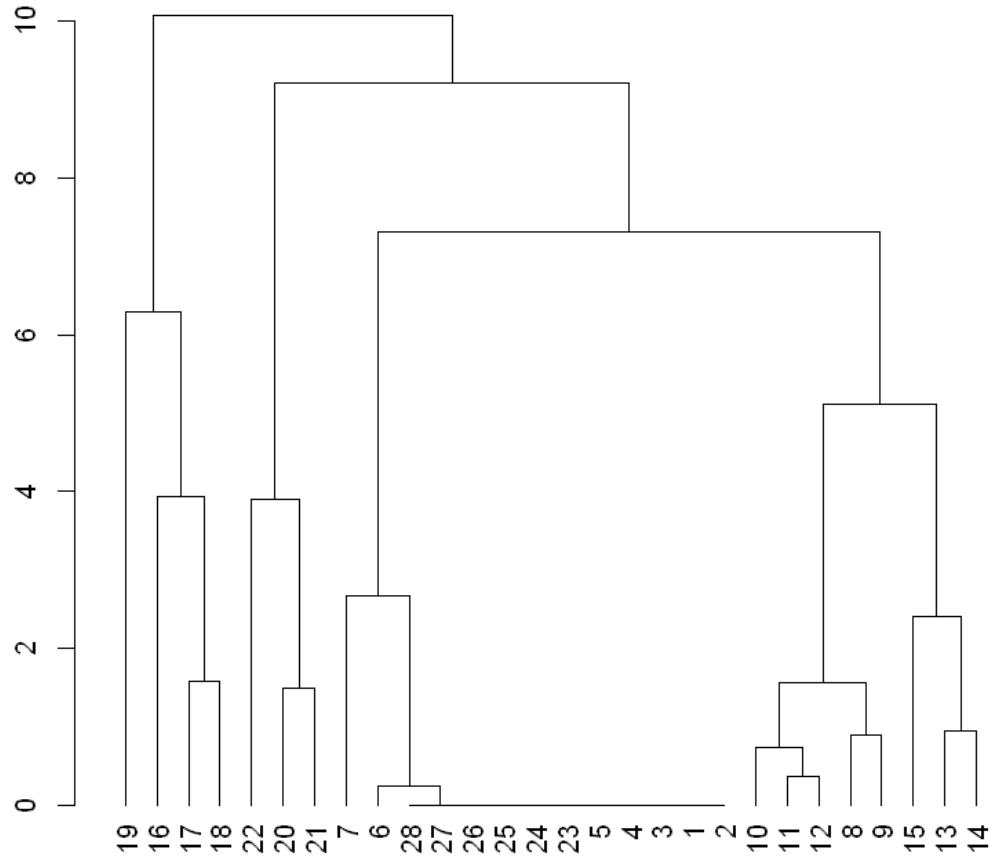
pc_hclust <- hclust(dist_recon, method='complete')

plot(as.dendrogram(pc_hclust))
# next plotting the hclust data as a dendrogram to visualize the relatic
# between each row.

# this clusters based on the rows
# what i can see is that in the center of the dendro gram we have the tc
# rows 1,2,3,4,5,6,7 being clustered together with the bottom rows 23,24
# 27,28 so this shows they have about the same distance values.
# next we see that the bottom middle rows are clustered together on the
```

```
# of the dendro gram which are rows 16,17,18,19,20,21,22.
# lastly, we see on the right of the dendrogram the top middle rows are
# together and the rows are 8,9,10,11,12,13,14,15
```

```
# NOTE: maybe i can try finding the distance of the columns and put that
# with the distance of the rows to get a better idea of each pixel someh
```



In [15]:

2066

In [12]:

```
# THOUGHT PROCESS GOING INTO MAKING THE NEXT CELL
# maybe we can use pca results instead
# we can take the average of each digit and find how many pc(principal c
# are needed to recreate 90% of the variance of the average
# then take one single digit and see if that single digit needs to use
# the same amount of pc to recreate 90% of the variance also
label_1[1,1]
```

1

In [10]:

```
# NUMBER 3 complete with written responce at the end of this cell

# this cell is from the midterm. we are using it to get the averages of

# this code it to just take the average of one single label number which
# label_0. we are doing this just to test on one label then will make a
# function in this cell to pass in all the labels_(0-9)

func_make_pc <- function (label){

  pixel_label_0 <- label[,-1]
  # taking all the data that has label hand writting as 0 and removing
  # the first column so we only have the pixels that represent 0
  # and we put that into a variable called pixel_label_0

  #length(pixel_label_0)
  c <- c(1:length(pixel_label_0))
  # now we want to make a variable that keeps track of every a number from
  # 1 to the max length of the columns this data consists

  ave_pix <- c()
  # we want to have an empty list ready for when we want to append the
  # average values of each column

  for (x in c){
    # we want to loop through my c variable with x, then we will make
    # x the column in the variable pixel_label_0

    #pixel_label_0[,x]
    ave_pix <- append(ave_pix,mean(pixel_label_0[,x]))
    # with each for loop we will look at the column and take the mean of
    # that entire column, then append the mean to the variable ave_pix
  }

  matrix_ave_pix <- matrix(rev(ave_pix), nrow=28, ncol=28)
  # here we take the list of means from ave_pix to make a matrix, where
  # each entry represents the mean number for each column in the
  # dataset labeled 0.

  matrix_ave_pix <- apply(matrix_ave_pix,2,rev)
  matrix_ave_pix

  # UNCOMMENT THE BELOW CODE TO SEE THE IMAGE
  #image(matrix_ave_pix, axes = FALSE, col = grey.colors(256, start=0, end=1))

  # here we plot the matrix using image function, and make the colors
  # shades of grey
  #pixel_label_0

#THIS NEXT PART MAKES THE AVERAGE PIX INTO A MATRIX FOR PCA
```

```

# reused code from making pca of label 3 to make one of label 0
# at the end i found a way to automate the process of finding the least
# of pc's when added up gives 90%

test <- data.frame(matrix_ave_pix)
#test
# note: i am using "my_label1" variable from the 3rd cell. this variable
# uses code that gets information from the first 10 rows in the pixel da
# the last pixel data ends up being the digit "3"

# so the digit we are looking to run pca on is 3

list_to_remove <- c()

for (x in c(1:length(test))){
  if (var(test[,x]) == 0) {
    list_to_remove <- append(list_to_remove, x)
  }
}

list_to_remove
# I used this for loop to find all the columns that has a variance of 0
# the column number into the list_to_remove variable. now i will edit th
# matirx test and remove those columns

test_update <- test[,-(list_to_remove)]
test_update

pca_label_0 <- prcomp(test_update[,c(2:length(test_update))], center = T)

# automate the below code to find how many pc equal 90%
standard_dev <- pca_label_0$sdev
# this is getting the standard deviation

var_label_0 <- standard_dev^2
# this is to get the variance of the pc
#var_label_1[1:10]

prop_var <- var_label_0/sum(var_label_0)
#prop_var[1:3]
# this shows the proportion of variance explained

# this while loop automates the proces of finding the least amount of
# pc's variances which add up to equal 90%
x <- 1
while (sum(prop_var[1:x])*100 < 90){
  x=x+1
}

# this keeps track of how many pc's it takes to create 90% of the variar
# for the average label data
#print(length(prop_var[1:x]))

```

```

print(sum(prop_var[1:x]))
assign(paste0("digit_",label[1,1]), (length(prop_var[1:x])))

#digit_1
# here i am assigning the first column and row as the variable number w/
# creating the variable. that way i can keep track of how many pc's are
# each variable passed through the function
#sum(prop_var[1:3])

# now that i know 4 pc's is the least amount when added gives 90% of the
# variance information for the label 0 when averaged
# i can now look at an individual digit to see whether it will also neec
# about 4 pc's to give its variance
# thus this will be my way of quickly classifying which digit i was give
# based on this quick calculation.

}

avg_digit_0_pc <- func_make_pc(label_0)
avg_digit_1_pc <- func_make_pc(label_1)
avg_digit_2_pc <- func_make_pc(label_2)
avg_digit_3_pc <- func_make_pc(label_3)
avg_digit_4_pc <- func_make_pc(label_4)
avg_digit_5_pc <- func_make_pc(label_5)
avg_digit_6_pc <- func_make_pc(label_6)
avg_digit_7_pc <- func_make_pc(label_7)
avg_digit_8_pc <- func_make_pc(label_8)
avg_digit_9_pc <- func_make_pc(label_9)

cat("the pc for 0 is :", avg_digit_0_pc, "\n")
cat("the pc for 1 is :", avg_digit_1_pc, "\n")
cat("the pc for 2 is :", avg_digit_2_pc, "\n")
cat("the pc for 3 is :", avg_digit_3_pc, "\n")
cat("the pc for 4 is :", avg_digit_4_pc, "\n")
cat("the pc for 5 is :", avg_digit_5_pc, "\n")
cat("the pc for 6 is :", avg_digit_6_pc, "\n")
cat("the pc for 7 is :", avg_digit_7_pc, "\n")
cat("the pc for 8 is :", avg_digit_8_pc, "\n")
cat("the pc for 9 is :", avg_digit_9_pc, "\n")

# this is how we will build our classifier. We know that the average of
# digit represents gives us a number of pc's that will add up to 90%
# variance
# so when we use the same code but only plug in one single digit we shou
# that that digit would need the same amount of pc's as its averaged cou
# to make 90% of the variances also.

# additional thing i noticed is that the odd numbers all needed 2 pc's w
# even numbers needed 3 pc's
# so we can definitely at least try to classify wheather the digit is ev
# or odd based on this information

```

```
# and the only digit we can classify is a 0, this is because its the only digit that requires above 3 pc's to get to 90% variance
```

```
[1] 0.9483771
[1] 0.9238257
[1] 0.9710878
[1] 0.9446342
[1] 0.9704346
[1] 0.9395174
[1] 0.967781
[1] 0.9023977
[1] 0.9554669
[1] 0.9493153
the pc for 0 is : 4
the pc for 1 is : 2
the pc for 2 is : 3
the pc for 3 is : 2
the pc for 4 is : 3
the pc for 5 is : 2
the pc for 6 is : 3
the pc for 7 is : 2
the pc for 8 is : 3
the pc for 9 is : 2
```

In [11]:

```
new_pixel <- data.frame(data)
#test_pixel <- new_pixel[9,]
#test_label <- new_pixel[9,1]
#cat("this is the unknown digit: ", test_label, "\n")

#unknown_digit_pc <- func_make_pc(test_pixel)
#cat("this is the pc for the unknown digit: ", unknown_digit_pc, "\n")

# the average pc of the same digit fed into the classifier should be roughly equal to the average pc we see in the cell above.

# so lets take 10 to 20 digits labeled 1 and see if its pc average ends up being 2, as seen as above.

# this code below looks through the entire original data set. then finds all of the row numbers that have the digit i will be testing
# next it takes only the first 10 of the rows numbers
new_test_df <- c()
for (x in c(1:length(new_pixel[,1]))){
  if (new_pixel[x,1] == 1){
    # here is where we choose which digit to use
    new_test_df <- append(new_test_df, x)

  }
}

#length(new_test_df)/2
# i tried to do half of the digit data as tests but it doesn't work
only_20_of_digit <- sample(new_test_df, 20)
#only_20_of_digit <- new_test_df
#only_20_of_digit
```

```
# now i want to loop through the row numbers with the function to get ea
# pc for the digit. note all the digits should be the same
# we will then take the average of all 20 pc and hope it equals the
# above average pc's for that number

take_avg <- c()
for (x in only_20_of_digit){
  new_pixel <- data.frame(data)
  test_pixel <- new_pixel[x,]
  test_label <- new_pixel[x,1]
  cat("this is the unknown digit: ", test_label, "\n")

  unknown_digit_pc <- func_make_pc(test_pixel)
  cat("this is the pc for the unknown digit: ", unknown_digit_pc, "\n\n")
  take_avg <- append(take_avg, unknown_digit_pc)
}

# as you can see the unknown digits are 1 like we want
# now to take the average of each pc which is,
new_list_avg <- c()
for (x in c(1:length(take_avg))){
  if (take_avg[x]< 5){
    new_list_avg <- append(new_list_avg,take_avg[x])
  }
}
new_list_avg
mean(new_list_avg)
# i wanted to remove any pc's that were above 4, because they would be c
# interferring with the classifier.
# none of the average pc's reached above 5.
```

```
this is the unknown digit:  1
[1] 0.90721
this is the pc for the unknown digit:  4

this is the unknown digit:  1
[1] 0.9417889
this is the pc for the unknown digit:  4

this is the unknown digit:  1
[1] 0.9328796
this is the pc for the unknown digit:  3

this is the unknown digit:  1
[1] 0.904581
this is the pc for the unknown digit:  5

this is the unknown digit:  1
[1] 0.903382
this is the pc for the unknown digit:  5

this is the unknown digit:  1
[1] 0.9897542
this is the pc for the unknown digit:  2

this is the unknown digit:  1
[1] 0.9494501
this is the pc for the unknown digit:  2
```

```
this is the unknown digit:  1  
[1] 0.9562665  
this is the pc for the unknown digit:  2  
  
this is the unknown digit:  1  
[1] 0.9641385  
this is the pc for the unknown digit:  2  
  
this is the unknown digit:  1  
[1] 0.9708957  
this is the pc for the unknown digit:  3  
  
this is the unknown digit:  1  
[1] 0.9395901  
this is the pc for the unknown digit:  3  
  
this is the unknown digit:  1  
[1] 0.9382433  
this is the pc for the unknown digit:  3  
  
this is the unknown digit:  1  
[1] 0.9534331  
this is the pc for the unknown digit:  4  
  
this is the unknown digit:  1  
[1] 0.9727557  
this is the pc for the unknown digit:  2  
  
this is the unknown digit:  1  
[1] 0.945469  
this is the pc for the unknown digit:  1  
  
this is the unknown digit:  1  
[1] 0.939379  
this is the pc for the unknown digit:  3  
  
this is the unknown digit:  1  
[1] 0.9328765  
this is the pc for the unknown digit:  2  
  
this is the unknown digit:  1  
[1] 0.9318862  
this is the pc for the unknown digit:  4  
  
this is the unknown digit:  1  
[1] 0.9265655  
this is the pc for the unknown digit:  4  
  
this is the unknown digit:  1  
[1] 0.9633652  
this is the pc for the unknown digit:  3
```

1. 4
2. 4
3. 3
4. 2
5. 2

6. 2
7. 2
8. 3
9. 3
10. 3
11. 4
12. 2
13. 1
14. 3
15. 2
16. 4
17. 4
18. 3

2.833333333333333

In []:

🔒 Tara91 / Final Private

<> Code ⚡ Issues 🛡 Pull requests ⏪ Actions 📁 Projects ! Security 🛍 Insights

⚡ main ▾

...

Final / Final.Rmd



Tara91 Add files via upload

🕒 History

👤 1 contributor

351 lines (240 sloc) | 10.6 KB

...

```
1 ---  
2 title: "Final"  
3 output: html_notebook  
4 ---  
5 Tarabeth Gerrity  
6 TG64  
7  
8 =====  
9 Project 1.  
10 =====  
11 This data is about basketball players from the year 2008 and is  
12 in the file ppg2008.csv. It has various statistics on players in the NBA.  
13 You might not know what each of the metrics means (I don't),  
14 but they are just different dimensions of data.  
15  
16 This is a visualization and data mining exercise.  
17  
18 What can you say about this dataset, use tools that you learned here.  
19 and make a report or a visual that highlights something interesting,  
20 maybe compare players, especially how they have performed since,  
21 based on the data in here. Many of these players have reached  
22 their peak recently and you will be able to find statistics about  
23 their performance in 2019.  
24  
25 Could you have predicted the successes and failures of some of the players, based  
26 on analyses of the data ?  
27 maybe you could be a talent scout for an NBA team ?  
28  
29
```

```
30 Think of it as your job, as a reporter for NY times, to make a single graphic
31 that highlights something about this data. Explain the analysis that went
32 into the graphic and present the code too. This should be done in a notebook so
33 it is easy to evaluate.
34
35
36
37 =====
38 Project 2.
39 =====
40
41 Dataset 2: train.csv.gz
42 This is hand-written digits (0-9) from many people.
43 It contains 785 columns, first column is digit and the 784 remaining columns
44 are pixels 0-783. These are essentially 28x28 squares, so you can map the 784
45 columns to entries in the squares. Each entry is 0-255, 0 is for black and 255
46 stands for white, numbers in between are shades of gray
47
48 ```{r}
49 train <- read.csv("/Users/morgenstern/Desktop/BioStats/Final/train.csv")
50
51 #column1, label- which digit is drawn
52 #columns 2 - 785- pixels (784 pixels total)
53 # digit is represented as pixels in a 28x28 square
54
55 for (i in 1:10){
56   m <- matrix(as.numeric((train[i,2:785])), nrow=28, ncol=28, byrow = TRUE)#digits drawn up
57   m <- m[nrow(m):1,]#flips the matrix to be upside up/flips across the horizontal
58   heatmap((m), Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
59 }
60 ##### 0 = white
61 ##### 255 = black
62
63 ```
64
65
66
67 1) Use PCA to reduce dimensions. How many components do you need to keep
68 to reproduce the digits reasonably well ? what is your final matrix ?
69 ```{r}
70 library(fpc)
71 library(resample)
72 set.seed(7)
73 ##test program on subset of train
74 #test <- (train[0:200,2:785])
75 ##test full train dataset
76 test <- train[,2:785]
77 ##remove all pixels with 0 variance--- all columns with only 0's
78 ##Since we're not scaling in PCA, nozero dataframes unneeded until pheatmap
```

```
79 i <- (colVars(test, na.rm=T) > 0.5) # T if col variance is not 0, F otherwise
80 nonzero <- test[i] # all the non-zero columns
81 nonzero##should have only pixels with variance
82 zerocols <- test[(colVars(test, na.rm=T) <= 0.5)] #Store the 0 variance columns to re-add
83 ##dataframe in a later step
84 colVars(nonzero[,1])
85
86 ### Do no include digit label in analysis!
87 #initialize the plot
88
89 kclusters <- kmeans(nonzero,1)
90 plot(1,kclusters$tot.withinss,xlim=c(0,10))
91
92 for (i in 2:10){
93   kclusters <- kmeans(nonzero,i)
94   points(i,kclusters$tot.withinss)
95 }
96
97 #Wait, shouldn't we just force 10 clusters, 1 for each digit? No breakpoint needed.
98
99 ebp <- kmeans(nonzero,10)
100 ebp$tot.withinss
101
102 ebp$cluster
103 ## Which cluster each sample falls under
104
105
106 ##plots clusters
107 aggregate(nonzero, by=list(cluster=ebp$cluster),mean)
108 plotcluster(nonzero,ebp$cluster)
109 #
110
111 #create the pca object
112 test.pca <- prcomp(nonzero,center = FALSE,scale. = FALSE)
113 print(test.pca)
114
115 plot(test.pca)
116 summary(test.pca)
117 library(ggbiplot)
118
119
120 ggbiplot(test.pca,obs.scale = 1, var.scale = 1, var.axes=FALSE, labels=(train$label[1:length(train$label)]))
121 #var.axes=FALSE removes the arrows and pixelnames, to see the grouping better
122
123 ### Can alter this to label points instead with the actual digit assignment.
124 ##(train[0:50,1])
125 ## Use this to give line # instead
126 #labels=row.names(test)
127
```

```

128
129   ``
130   ``{r}
131
132 nComp = 50 ## 200 reproduces original matrix. ~50 the 4 becomes unrecognizable
133 ##other numbers fare better at lower component numbers though
134 #mu = colMeans(test)
135 Xhat = test.pca$x[,1:nComp] %*% t(test.pca$rotation[,1:nComp])
136 #Xhat = scale(Xhat, center = -mu, scale = FALSE)
137
138 #loop through original data re-adding any of the missing columns
139 ## $ operator won't work inside function, use [[]] to call variable instead
140 Phat <- data.frame(Xhat)
141 Phat <- cbind(Phat, zerocols) ##add back in the zeroval columns
142
143 colnames(train)
144 colnames(Phat)
145
146 #install.packages("gtools")
147 library(gtools)
148 Phat <- Phat[, mixedsort(colnames(Phat))]
149 ##use this package to sort
150 ##sort orders columns pixel1, pixel11,pixel111 instead of pixel1, pixel2 etc
151 ##Graph the new matrix
152
153 ##Why does the resulting matrix contain negative values?
154 Phat[Phat < 1] <- 0
155
156
157 for (i in 1:10){
158   m <- matrix(as.numeric((Phat[i,1:784])),nrow=28, ncol=28, byrow = TRUE)#digits drawn up
159   m <- m[nrow(m):1,]#flips the matrix to be upside up/flips across the horizontal
160   heatmap((m),Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
161 }
162
163 ## Testing the matrices
164
165 #pm <- matrix(as.numeric((Phat[9,1:784])),nrow=28, ncol=28, byrow = TRUE)
166 #heatmap((pm), Rowv=NA, Colv = NA, col = gray.colors(256, start = 1, end=0))
167
168 #om <- matrix(as.numeric((train[9,2:785])),nrow=28, ncol=28, byrow = TRUE)
169 #heatmap((om), Rowv=NA, Colv = NA, col = gray.colors(256, start = 1, end=0))
170
171
172 ``
173
174
175 2) Draw a tree of the pixels, and see if you can explain the results based on
176 geometry of the pixels (how far apart are they in the 2-d space).

```

```
177 Try to Explain the PCA results in light of this.  
178  
179 ```{r}  
180 ##Note, 2D space. The closest 8 pixels will not be sequential  
181  
182 ##Use euclidean correlation?  
183  
184  
185 test  
186 dm <- dist(t(test), method = "euclidean")  
187 test.hclust <- hclust(dm, method = "complete")  
188 #install.packages("dendextend")  
189 library(dendextend)  
190 #par(mar=c(5,5,5,12))  
191 #nPar <- list(lab.cex = 0.6, pch = c(NA, 19),cex = 0.7, col = "blue")  
192 #ePar = list(col = 2:3, lwd = 2:1)  
193 plot(as.dendrogram(test.hclust),horiz=TRUE)  
194  
195 pdf("/Users/morgenstern/Desktop/test.pdf", width=500, height=100)  
196  
197 # Do some plotting  
198 plot(as.dendrogram(test.hclust),horiz=TRUE)  
199 # Close the PDF file's associated graphics device (necessary to finalize the output)  
200 dev.off()  
201  
202 ##A tree. Not very useful on its own. Can we do a heatmap for better visualization?  
203  
204 ...  
205 ```{r}  
206  
207 library(pheatmap)  
208  
209 pdf("/Users/morgenstern/Desktop/testheatmap2.pdf", width=300, height=100)  
210 ## Take the original train dataset  
211  
212 trainsort <- train[1:200, 1:785]  
213 trainsort <- trainsort[sort(trainsort$label),]  
214 #sorting the dataframe by label column groups each row by digits.  
215  
216  
217 ##This heat map groups samples into digits- pixel correlation patterns emerge  
218 pheatmap(trainsort,  
219     cellwidth = 20,  
220     cutree_rows = length(unique(trainsort$label)))  
221 dev.off()  
222  
223 ##doing the same but without the zero variance pixels  
224  
225 pdf("/Users/morgenstern/Desktop/nozeropheatmap.pdf", width=200, height=100)
```

```
226 trainsort <- train[1:200, 1:785]
227 trainsort <- trainsort[sort(trainsort$label),]
228 trainsort <- trainsort[, mixedsort(colnames(trainsort))]
229 i <- (colVars(trainsort, na.rm=T) > 0) # T if col variance is not 0, F otherwise
230 nonzerotrainsort <- trainsort[i] # all the non-zero columns
231 nonzerotrainsort##should have only pixels with variance
232
233 pheatmap(nonzerotrainsort,
234           cellwidth = 20)
235 dev.off()
236
237 ...
238
239
240
241 3) Can you use some of the tools you have learnt to build a classifier,
242 so if you get a new set of pixels you can predict what is in the
243 picture. This is the start of a real project, but you don't have all the tools
244 (such as neural networks) which might be more suited for this task.
245 Split your dataset into two (a training set and a test set),
246 build your classifier and figure out
247 how well it does with the test data in predicting the digits.
248 Define the sensitivity and specificity of your classifier.
249 How well does it recognize your own handwriting
250 (make sure your handwriting is not in the training set)
251
252 ````{r}
253
254 training <- data.frame(train[0:200,2:785])
255
256 testing <- data.frame(train[500,2:785])
257
258 myDigit <- data.frame()
259
260
261 testsubject <- data.frame() ## a line of pixel values
262
263 nComp = 50 ## Reuse value from previous step
264 PredictedDigit <- predict(test.pca, newdata = testing)
265
266 Prediction <- PredictedDigit[, 1:nComp]
267
268 ebp <- kmeans(training,10)
269 ebp$tot.withinss
270 ebp$cluster
271 ## Which cluster each sample falls under
272 ##plots clusters
273 aggregate(training, by=list(cluster=ebp$cluster),mean)
274 plotcluster(training,ebp$cluster)
```

```
275 #  
276  
277 #create the pca object  
278 training.pca <- prcomp(training,center = FALSE,scale. = FALSE)  
279  
280 summary(training.pca)  
281  
282 TrainingPlot <- ggbiplot(training.pca,obs.scale = 1, var.scale = 1, var.axes=FALSE, labels=TRUE)  
283  
284 TrainingPlot  
285  
286 PredictionPlot <- geom_point(aes(x=Prediction[1], y=Prediction[2]), colour="Red", size =4)  
287 print(TrainingPlot + PredictionPlot)  
288 print(train[500,1])  
289  
290 ...  
291  
292  
293  
294  
295  
296 plot_3 <- plot_2 + geom_point(aes(x=new_cus_group_PC1_PC2[1], y=new_cus_group_PC1_PC2[2]),  
297  
298 plot_3 <- plot_3 + labs(title="Plotting new observation against PC1 and PC2")  
299  
300 print(plot_3)  
301  
302  
303  
304 4) You can try simple things like take average of all data for each number and  
305 then take a "dot" product with your test set, and identify the pixels. This might  
306 work, maybe for some digits, and not others.  
307  
308  
309  
310  
311 =====  
312 Project 3.  
313 =====  
314  
315  
316 Dataset 3: a) Mnemiopsis_col_data.csv b) Mnemiopsis_count_data.csv  
317  
318 This is gene expression data, The columns represent samples, whose information is  
319 in the col_data file. The count_data file contains counts for each gene (rows).  
320 The file, info_gene.txt contains information about the organism and some links to  
321 look up gene functions.  
322 It will be a good experience to learn to use the genome resources,  
323 as this is the kind of struggles most researchers go through when
```

324 they start looking at genes.
325
326
327 1) Build hierarchical trees based on the columns and for the rows (exclude rows
328 that are "low" expression)
329
330 2) Draw a heat map of the expression data
331
332 3) Use DESeq2 to analyse this data
333 a) which are the most significantly changing genes in this dataset ?
334 b) which genes are most consistently highly expressed in these datasets
335 they are the "house-keeping" genes
336 c) How consistent are these results with the analysis you did
337 in the midterm project ?
338 d) What else can you say about the data in terms of consistency,
339 and the results that you find from your analyses. The question is open-ended,
340 think of this as your experiment, you need to write a paper based on this data
341 so you have to figure out what kind of "story" you can tell based on this.
342
343 e) what is the most interesting pathway or gene that is responding in this study?
344
345
346
347
348
349
350
351

caroleen21 / Biostatistics-Final Public

<> Code ⚡ Issues 🛡 Pull requests ⏪ Actions 📁 Projects 🛡 Security 🛍 Insights

godev main ▾

...

Biostatistics-Final / Carolee Nguyen Final Project Biostatistics - Part 2.Rmd



caroleen21 Add files via upload

🕒 History

1 contributor

277 lines (178 sloc) | 7.12 KB

...

```
1 ---  
2 title: "RNAseqHW"  
3 author: "Carolee Nguyen"  
4 output: html_document  
5 ---  
6  
7 ```{r setup, include=FALSE}  
8 knitr::opts_chunk$set(echo = TRUE, error=TRUE)  
9 ...  
10  
11 For dataset 2  
12 1) Build hierarchical trees based on the columns and for the rows (exclude rows  
13 that are "low" expression)  
14 2) Draw a heat map of the expression data  
15 3) Use DESeq2 to analyse this data  
16   a) which are the most significantly changing genes in this dataset ?  
17   b) which genes are most consistently highly expressed in these datasets  
18     they are the "house-keeping" genes  
19   c) How consistent are these results with the analysis you did  
20     in the midterm project ?  
21   d) What else can you say about the data in terms of consistency,  
22     and the results that you find from your analyses. The question is open-ended,  
23     think of this as your experiment, you need to write a paper based on this data  
24     so you have to figure out what kind of "story" you can tell based on this.  
25  
26   e) what is the most interesting pathway or gene that is responding in this study ?  
27  
28  
29 ```{r readdata}
```

```
30
31 countdata = read.csv(file = "/Users/carol/Downloads/Mnemiopsis_count_data.csv", header=TRUE)
32 countdata = as.data.frame(countdata) #Read data file as a data frame
33
34 print(grep("aboral", countdata)) #Use grep function to extract all cases of "aboral"
35 print(grep("oral", countdata)) #Use grep function to extract all cases of "oral"
36
37 dfaboral <- countdata[,c(2,3,4,5)] #Create a dataframe for aboral
38 dforal <- countdata[,c(6,7,8,9)] #Create a dataframe for oral
39
40
41
42 dfcounts <- countdata[,-1]
43 rownames(dfcounts) <- countdata[,1]
44
45
46 print(dfcounts)
47 ``
48
49
50 ### Step 2
51
52
53 ````{r createfactor}
54
55
56
57 expgroup <- data.frame(condition=1:8) #Create a dataframe with 8 values corresponding to t
58 expgroup$condition[c(1:4)] <- 'Aboral' #Assign the first 20 values as healthy using grep i
59 expgroup$condition[c(5:8)] <- 'Oral' #Assign the last 20 values as CF using grep informati
60
61
62 rownames(expgroup) <- colnames(countdata[,c(2:9)]) #Use the column names of read count as
63
64 print(expgroup) #Print the expgroup dataframe
65 ``
66
67 ### Step 3
68
69 ````{r runDESeq2}
70
71 library("DESeq2") #Load DeSeq2 library
72 cds <- DESeqDataSetFromMatrix(countData = dfcounts, #Create a count data set matrix named
73                                     colData = expgroup, #Set colData as expgroup
74                                     design = ~ condition) #Set design as condition
75 cds #Call the matrix
76
77 ``
78
```

```
79 ### Step 4
80
81
82 ````{r estimate}
83
84 #Utilize the previous count data set matrix in the functions below:
85 cds <- estimateSizeFactors(cds) #Function for estimating size factors
86 cds <- estimateDispersions(cds) #Function for estimating dispersion
87 plotDispEsts(cds) #Function for plotting dispersion
88
89
90
91
92 ````

93
94 ### Step 5
95
96 ````{r deseqres}
97
98 #Utilize the previous count data set matrix in the functions below:
99 cds <- DESeq(cds) #perform differential gene expression
100 res <- results(cds) #use results function to obtain differential gene expression results
101 res #Output DeSeq2 results
102
103
104 ````

105
106 ### Step 6
107
108
109
110 ````{r diffexp}
111 #Look for genes that have an adjust p-value <0.05 and greater than 1
112 resSigind = res[ which(res$padj < 0.05 & res$log2FoldChange > 1), ]
113 #Look for genes that have an adjust p-value <0.05 and less than 1
114 resSigrep = res[ which(res$padj < 0.05 & res$log2FoldChange < -1), ]
115 #Combine the two previous results together
116 diffexpgenes = rbind(resSigind, resSigrep)
117 #Print the row names, which is the list of differentially expressed genes
118 rownames(diffexpgenes)
119
120 nrow(diffexpgenes)
121
122
123
124 ### Step 7
125
126
127
```

```
128  ````{r normvalues}
129 #Utilize the previous count data set matrix in the functions below:
130 normvalues = counts(cds, normalized=TRUE)
131 print(normvalues) #print the normalized values
132
133 colnames(normvalues) #Show column names for matching in the step below
134 ```
135
136 ### Step 8
137
138
139 ````{r diffvalues}
140
141 ###match(rownames(diffexpgenes), rownames(normvalues))
142 diffexpvalues = normvalues[rownames(normvalues) %in% rownames(diffexpgenes), ]
143 #Print the dataframe
144 print(diffexpvalues)
145 #Print the dimension of the dataframe to see if it matches the list of differentially expr
146 print(dim(diffexpvalues))
147
148
149 aboral1 = sort(diffexpvalues[,1])
150 tail(aboral1)
151 #ML174735a ML034334a ML034337a ML034336a ML01482a ML46651a
152 #17157.06 21911.36 22424.14 23631.30 30581.63 118211.07
153
154 aboral2 = sort(diffexpvalues[,2])
155 tail(aboral2)
156 #ML174735a ML034337a ML034336a ML034334a ML01482a ML46651a
157 #40418.84 58540.29 65914.50 68219.38 80559.11 93870.29
158
159 aboral3 = sort(diffexpvalues[,3])
160 tail(aboral3)
161 #ML174735a ML034337a ML034336a ML034334a ML46651a ML01482a
162 #27488.06 47004.61 50998.50 55714.11 56425.56 71249.61
163
164 aboral4 = sort(diffexpvalues[,4])
165 tail(aboral4)
166 #ML174735a ML034337a ML034336a ML46651a ML034334a ML01482a
167 #37089.43 67521.80 82850.63 91834.25 92639.95 110042.52
168
169 oral1 = sort(diffexpvalues[,5])
170 tail(oral1)
171 #ML01433a ML087114a ML199832a ML004510a ML21583a ML306119a
172 #14912.24 17974.26 18039.35 22064.90 23649.09 31192.30
173
174 oral2 = sort(diffexpvalues[,6])
175 tail(oral2)
176 #ML11032a ML306119a ML34341a ML087114a ML199832a ML004510a
```

```
177 #16119.59 16538.89 18610.45 20852.11 21035.21 34514.43
178
179 oral3 = sort(diffexpvalues[,7])
180 tail(oral3)
181 #ML46651a ML43587a ML11032a ML087114a ML01482a ML004510a
182 #27853.26 29726.57 30124.16 42894.61 44780.12 66495.13
183
184 oral4 = sort(diffexpvalues[,8])
185 tail(oral4)
186 #ML01433a ML11032a ML09436a ML087114a ML46651a ML004510a
187 #26045.23 28540.28 33828.03 39617.47 64936.93 77464.55
188 ``
189
190 ### Step 9
191
192 ````{r diffgroups}
193
194
195 #Create a hierarchical cluster
196
197 hc <- hclust(dist(diffexpgenes), method = "complete")
198
199 plot = plot(hc, main = "Dendrogram", horiz = TRUE, cex = 0.5)
200
201 print(plot)
202
203
204
205
206
207 ``
208
209
210 ### Step 10
211
212 ````{r heatmap}
213
214 #Load the pheatmap package
215 library(pheatmap)
216
217 #Plot the heatmap using designated variables
218 pheatmap(diffexpvalues,
219           color = colorRampPalette(c("navy", "white", "firebrick3"))(50),
220           scale = "row",
221           cluster_rows = TRUE,
222           cellwidth = 5,
223           cellheight = 3,
224           fontsize = 5,
225           width=2000,
```

```
226     height=800  
227     )  
228  
229  
230 pheatmap(diffexpvalues)  
231 ````  
232  
233 ### Step 11  
234  
235  
236 ````{r eval=F}  
237 #Install all Bioconductor packages  
238 BiocManager::install("GOstats")  
239 BiocManager::install("GO.db")  
240 BiocManager::install("Category")  
241 BiocManager::install("org.Hs.eg.db")  
242  
243 ````  
244  
245  
246  
247 ````{r loadlibraries}  
248 #Load all libraries  
249  
250 library(GOstats)  
251 library(GO.db)  
252 library(Category)  
253 library(org.Hs.eg.db)  
254  
255 ````  
256  
257  
258  
259 ````{r gostats}  
260 #Create a GOHyperParams object using variables created from the previous steps  
261 params=new("GOHyperGParams",  
262   geneIds=rownames(diffexpgenes),  
263   universeGeneIds=rownames(readcount),  
264   annotation="org.Hs.eg",  
265   ontology="BP",  
266   pvalueCutoff=0.001,  
267   conditional=TRUE,  
268   testDirection="over")  
269  
270 #Run HyperFTest  
271 (overRepresented=hyperGTest(params))  
272 #Print out summary of desired columns  
273 summary(overRepresented)[,c(1,2,5,6,7)]  
274
```

[sarahdoon / Final](#) Public[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)[main](#) ▼

...

[Final](#) / Sarah_Feraidoon Final.R

sarahdoon Add files via upload

[History](#)[1 contributor](#)

201 lines (184 sloc) | 6.46 KB

...

```
1 # Dataset 1
2 # 1) Use PCA to reduce dimensions. How many components do you need to keep
3 # to reproduce the digits reasonably well? what is your final matrix ?
4 data <- read.csv("/Users/sarahferaidoon/Desktop/train.csv.gz")
5 data
6
7 # Number 2
8 n <- c(0:9)
9 # keep track of all digits corresponding to a handwriting from
10 # 0 to 9 in variable n
11 for (x in n){
12   # loop through each digit
13   # assign puts all data we want to grep in a variable
14   # paste0 creates variable names dynamically
15   # grep takes digit from first column following pattern
16   # put entire row matching digit into a variable
17   # loop moves onto next digit
18   assign(paste0("label_",x),data[grep(x,data[1]),])
19 }
20 label_0
21 label_1
22 label_2
23 label_3
24 label_4
25 label_5
26 label_6
27 label_7
28 label_8
29 label_9
30 data[grep("0",data[,1]),]
31
32
33 length(label_1)
34 # scale FALSE to take out columns with variance of 0
35 pca_label_1 <- prcomp(label_1[,c(2:784)],center=TRUE,scale=FALSE)
36 summary(pca_label_1)
37 # St dev goes below 0.05 after PC496. It is above 0.05 from PC1-PC496
38 plot(pca_label_1$x[,1],pca_label_1$x[,2])
39 biplot(pca_label_1, scale=0)
40 # using screeplot, find the pc cut off that contains 90% of the variance
```

```
41 # and convert it back to the data to see how the digits look
42 screeplot(pca_label_1,type="lines")
43 new_label_1 <- label_1[,c(2:496)]
44
45 # get standard deviation
46 standard_dev <- pca_label_1$sdev
47 # get variance of pc
48 var_label_1 <- standard_dev^2
49 var_label_1[1:10]
50
51 # explain proportion of variance
52 prop_var <- var_label_1/sum(var_label_1)
53 prop_var[1:35]
54 # the first 35 pc (principal components) consist of 90% of the variance
55 sum(prop_var[1:35])
56 #convert the first 35 pc back into data and see how digit looks
57 pc.use <- 35
58 ???reconstructed <- pca_label_1$x[,1:pc.use] %*% t(pca_label_1$rotation[,1:pc.use])
59 reconstructed
60 image(reconstructed, axes=FALSE, col=grey.colors(256, start=0, end=1))
61 pca_label_1$x[,1:pc.use]
62
63
64 # Dataset 2
65 # 1) Build hierarchical trees based on the columns and for
66 # the rows (exclude rows that are "low" expression)
67 # read data file as variable
68 countdata <- read.csv(file="/Users/sarahferaidoon/Desktop/Mnemiopsis_count_data.csv", header=TRUE)
69 # read data file as dataframe
70 countdata <- as.data.frame(countdata)
71 countdata
72 # use grep function to extract all cases of "aboral"
73 print(grep("aboral",countdata))
74 # use grep function to extract all cases of "oral"
75 print(grep("oral",countdata))
76 # create dataframe for aboral
77 dfaboral <- countdata[,c(2,3,4,5)]
78 # create dataframe for oral
79 dforal <- countdata[,c(6,7,8,9)]
80 dfcounts <- countdata[,-1]
81 rownames(dfcounts) <- countdata[,1]
82 print(dfcounts)
83 # Create a dataframe called expgroup, with 1 column labeled condition (assigning healthy column). column names of
84 # Create a df with 8 values (corresponding to the columns in readcount)
85 expgroup <- data.frame(condition=1:8)
86 # use grep info and assign the first 20 values as healthy
87 expgroup$condition[c(1:4)] <- "Aboral"
88 # use grep info and assign the last 20 values as CF
89 expgroup$condition[c(5:8)] <- "Oral"
90 # use column names of read count as row names of expgroup
91 rownames(expgroup) <- colnames(countdata[,c(2:9)])
92 # print the df
93 print(expgroup)
94
95 # Step 3
96 # Load SeSeq2 library
97 if (!require("BiocManager", quietly = TRUE))
98   install.packages("BiocManager")
99
100 BiocManager::install("DESeq2")
```

```
101 library("DESeq2")
102 cds <- DESeqDataSetFromMatrix(countData = dfcounts, colData=expgroup, design = ~condition)
103 cds
104
105 #Step 4
106 # Make sure DESeq2 can correct for library size and dispersion estimates
107 # Use functions estimateSizeFactors and estimateDispersions
108 # Plot the dispersion using plotDispEsts
109 # Analyze graph
110 # Function for estimating size factors
111 cds <- estimateSizeFactors(cds)
112 # Function for estimating dispersion
113 cds <- estimateDispersions((cds))
114 # Function for plotting dispersion
115 plotDispEsts(cds)
116
117 # Step 5
118 # Perform differential expression with DeSeq and results functions
119 # perform differential gene expression
120 cds <- DESeq(cds)
121 # Obtain differential gene expression results
122 res <- results(cds)
123 # Show output
124 res
125
126 # Step 6
127 # Find p-values less than 0.05 and log2FoldChange > 1 or less than -1
128 # How many? save in list: diffexpgenes
129 # Genes with adjust p-value <0.05 and greater than 1
130 ressigind <- res[which(res$padj < 0.05 & res$log2FoldChange > 1),]
131 # Genes with adjust p-value <0.05 and less than -1
132 ressigrep <- res[which(res$padj < 0.05 & res$log2FoldChange < -1),]
133 # Combine results
134 diffexpgenes <- rbind(ressigind, ressigrep)
135 # print list of differentially expressed genes
136 rownames(diffexpgenes)
137 nrow(diffexpgenes)
138 # ~ 1500 genes
139
140 # Step 7
141 # Normalize values of counts data in cds with counts() function and normalized=Tm call this normvalues
142 normvalues <- counts(cds, normalized=TRUE)
143 # print normalized values
144 print(normvalues)
145 #Show columns names for matching
146 colnames(normvalues)
147
148 # Step 8
149 # Create new matrix/dataframe containing expression values from normvalues for only diffexpgenes, call this diffexpgvalues
150 diffexpgvalues <- normvalues[rownames(normvalues) %in% rownames(diffexpgenes),]
151 # print the dataframe
152 print(diffexpgvalues)
153 # see if this matches the list of differentially expressed genes from last step
154 print(dim(diffexpgvalues))
155
156 # Step 9
157 # Create hierarchical cluster
158 hc <- hclust(dist(diffexpgenes), method="complete")
159 plot <- plot(hc, main="Dendrogram", horiz=TRUE, cex=0.5)
160 print(plot)
```

```
161  
162 # Step 10  
163 # Create a heat map using pheatmap package  
164 install.packages("pheatmap")  
165 library(pheatmap)  
166 # Plot the heatmap using designated variables  
167 pheatmap(diffexpvalues,  
168     color=colorRampPalette(c("navy","white","firebrick3"))(50),  
169     scale="row",  
170     cluster_rows=TRUE,  
171     cellwidth=5,  
172     cellheight=3,  
173     fontsize=5)  
174  
175 # Step 11  
176 # Use GOStats package to determine which GO-terms are enriched in diffexpgenes  
177 # install the following packages from Bioconductor:  
178 BiocManager::install("GOstats")  
179 BiocManager::install("GO.db")  
180 BiocManager::install("Category")  
181 BiocManager::install("org.Hs.eg.db")  
182 if (!require("BiocManager", quietly = TRUE))  
183   install.packages("Category")  
184 # Load libraries  
185 library(GOstats)  
186 library(GO.db)  
187 library(Category)  
188 library(org.Hs.eg.db)  
189 # Create a GOHyperParams object using variables created from steps  
190 params <- new("GOHyperGParams",  
191     geneIds=rownames(diffexpgenes),  
192     universeGeneIds=rownames(diffexpvalues),  
193     annotation="org.Hs.eg",  
194     ontology="BP",  
195     pvalueCutoff=0.001,  
196     conditional=TRUE,  
197     testDirection="over")  
198 # Run HyperFTest  
199 (overRepresented<-hyperGTest(params))  
200 # Print summary of desired columns  
201 summary(overRepresented)[,c(1,2,5,6,7)]
```

Ryan Johnson
Carolee Nguyen
Sarah Feraidoon
Tarbeth Gerrity

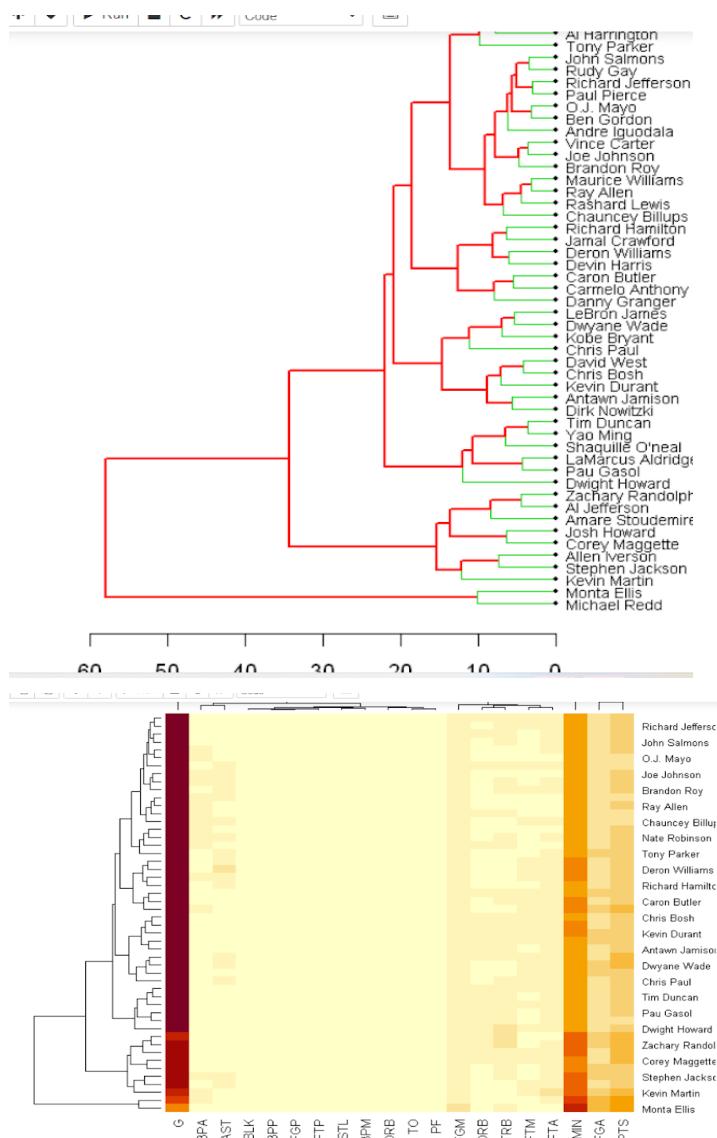
Applied Biostatistics Final:

Project 1.

=====

This data is about basketball players from the year 2008 and is in the file ppg2008.csv. It has various statistics on players in the NBA. You might not know what each of the metrics means (I don't), but they are just different dimensions of data.

This is a visualization and data mining exercise.



What can you say about this dataset, use tools that you learned here and make a report or a visual that highlights something interesting, maybe compare players, especially how they have performed since, based on the data in here. Many of these players have reached their peak recently and you will be able to find statistics about their performance in 2019.

Could you have predicted the successes and failures of some of the players, based on analyses of the data ? Maybe you could be a talent scout for an NBA team ?

It seems like the dimensions, G, MIN, and PTS are very closely related amongst the players when looking at the heatmap. G especially looks like many of the players score a similar number for this parameter, which stands for games played. (we looked it up). So it makes sense that many of the top named players have similar total games played and the outliers in a lighter color must be newer players.

Think of it as your job, as a reporter for NY times, to make a single graphic that highlights something about this data. Explain the analysis that went into the graphic and present the code too. This should be done in a notebook so it is easy to evaluate.

Code to make the heat map and dendrogram:

```
dm <- dist(df_players)
#dm
hclust_data <- hclust(dm,method='complete')
hclust_data
# creating a hclust matrix of the distance data from the players.

#library(dendextend)
npar <- list(lab.cex=0.7, pch=c(NA,20),cex=0.7)
par(mar=c(5,5,3,12), cex.axis=1)
epar = list(col=2:3,lwd=2:1)
plot(as.dendrogram(hclust_data),nodePar=npar, edgePar= epar, horiz=TRUE)
# above i plotted a dendrogram of the hclust data generated above to visualize
# the data in a tree and see the clusters of players

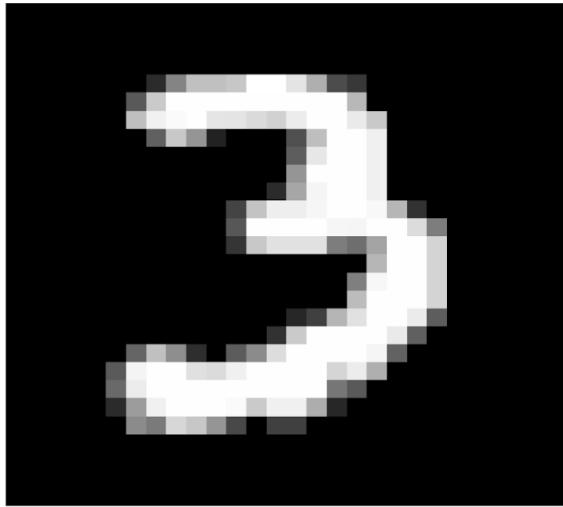
df_players_matrix <- as.matrix(df_players)
#df_players_matrix
# i created a matrix of the players stats then made a heatmap to visualize it
heatmap(df_players_matrix, margins=c(5,5), hclustfun=hclust)
```

Project 2.

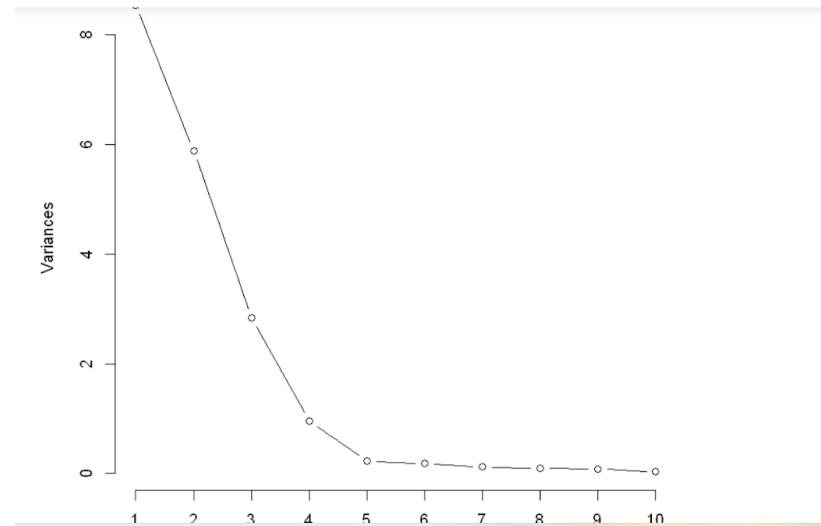
=====

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 ?

We used data from the digit 3. This is how it looks before pca

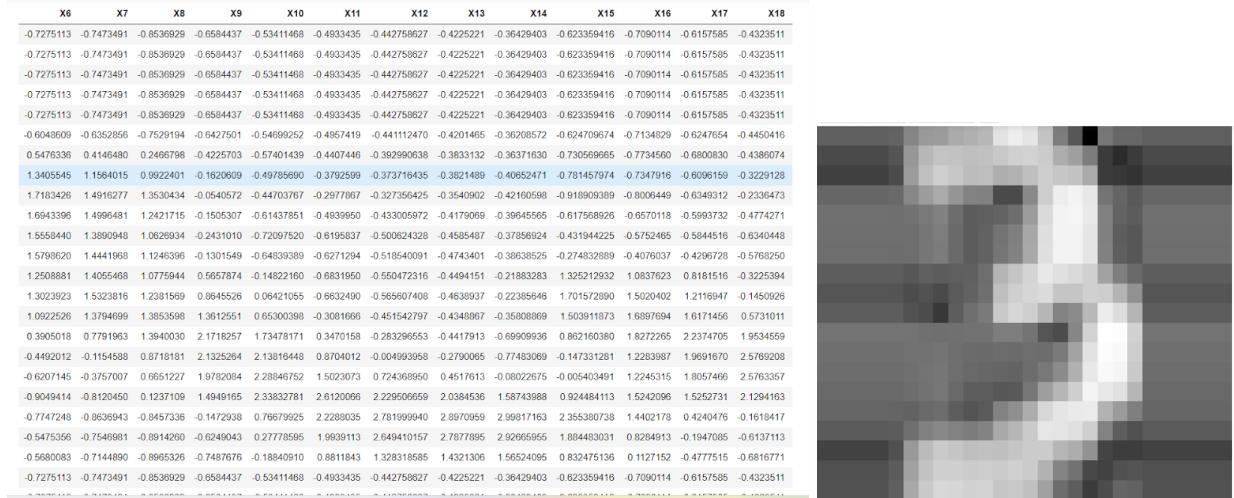


After looking at the screeplot we can see that only 3 principal components are needed to reproduce the digits reasonably well for this digit. Here is the screeplot:



After running a test we saw that only 3 pc's were needed to get 90% of the variances, however we decided to go with using 4 pc's to recreate the image because it ends up looking more distinguishable. Here is a small cutout of how the final matrix looks for digit 3 after using only the

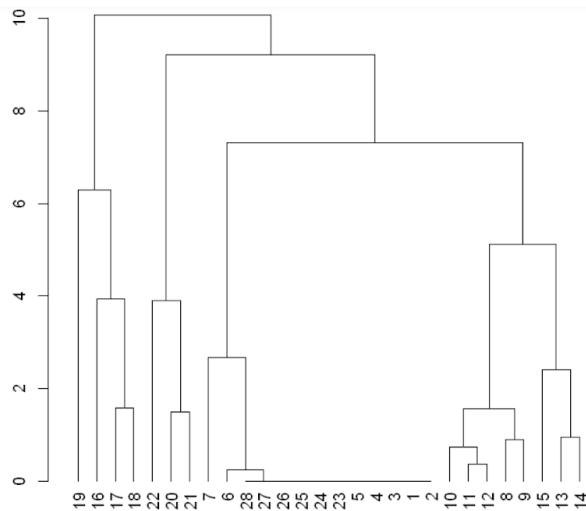
4 pc's and also how the image looks after reconstruction.



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.

Here is the tree of rows in the image and how they relate to each other.



This tree only represents the rows in image 3 reconstructed.

What I can see is that in the center of the dendrogram we have the top rows 1,2,3,4,5,6,7 being clustered together with the bottom rows 23,24,25,26, 27,28 so this shows they have about the same distance values.

Next we see that the bottom middle rows are clustered together on the left of the dendrogram which are rows 16,17,18,19,20,21,22.

Lastly, we see on the right of the dendrogram the top middle rows are clustered together and the rows are 8,9,10,11,12,13,14,15.

We can then repeat this process by transposing the matrix so the columns become the rows and the rows become the columns. This will then give us a tree of the columns. Combining the data from the rows and columns we would be able to understand how each pixel is clustered in relation to the others. Only based on the row data we can see that there are about 3 to 4 bands that are clustered together, so this represents the 4 pc's we used to reconstruct the image.

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 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)

To build a classifier we thought one way to do so would be using the pca result to see how many pc's are needed to reconstruct each digit. Then we would compare that number of pc's to the number of pc's it takes to reconstruct the average pixels of all of the single digits.

So first, we took all the pixel data for each digit and averaged the rows into one row.

Next, we made a matrix of each of the 10 digits and ran pca to find how many pc's made up 90% of the variance for each.

```
[1] 0.9483771
[1] 0.9238257
[1] 0.9710878
[1] 0.9446342
[1] 0.9704346
[1] 0.9395174
[1] 0.967781
[1] 0.9023977
[1] 0.9554669
[1] 0.9493153
the pc for 0 is : 4
the pc for 1 is : 2
the pc for 2 is : 3
the pc for 3 is : 2
the pc for 4 is : 3
the pc for 5 is : 2
the pc for 6 is : 3
the pc for 7 is : 2
the pc for 8 is : 3
the pc for 9 is : 2
```

The decimal numbers represent how much variance each average digit contains after pca and you can see the pc's required to recreate each digit. This is the training data

Next, we thought of finding how to get test data. So we will get a random digit and run pca on that digit, if the number of pc's required to recreate that number is similar to any of the pc's above we can assume the unknown digit corresponds to that digit.

This classifier does not perform well because the digit 0 is the only digit with a unique pc count. One thing that this classifier can do is at least tell us if we have an even or odd unknown number. The odd numbers seem to require 2 pc's while the even numbers require 3 pc's for reconstruction at 90% variance.

When looking at sensitivity vs specificity, we will have to adjust the sensitivity to require multiple unknown numbers, of the same digit, to be fed into the function so we can take an average of their pc's to compare it to the pc's in the training results.

The specificity would be choosing to not include any test digits that return a pc above 5.

```
this is the unknown digit: 0
[1] 0.9997239
this is the pc for the unknown digit: 5

this is the unknown digit: 0
[1] 0.9986482
this is the pc for the unknown digit: 3

this is the unknown digit: 0
[1] 0.9976168
this is the pc for the unknown digit: 3

4 5 5 4 5 4 4 4 5 4 5 4 4 4 4 4 3 2 5 5 5 4 5 4 4 5 4 3 4 4 4 4 4 4 4 4 3 3 5 4 5 4 5 5 4
4 3 5 3 3 3 4 4 5 4 4 5 5 4 4 5 4 5 5 5 3 4 3 4 5 5 5 5 4 4 4 5 4 4 4 4 3 4 4
4 4 4 5 4 4 3 4 5 3 3 5 4 5 3 3
```

4.16981132075472

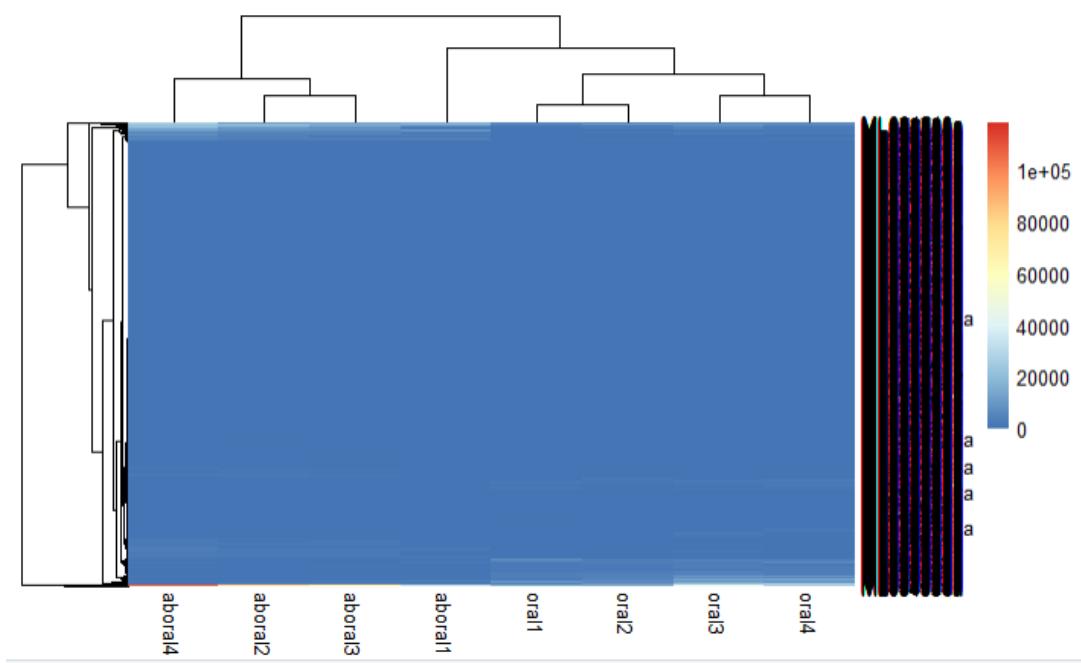
So here you can see we took 120 random samples of digit 0 and found the pc's for each. Next, we used a for loop to remove any pc's that were above 5 then took the average of the remaining pc's which turned out to be 4.1. This means our random number should be 0. This doesn't work too well with other digits, unless we change the sensitivity or specificity.

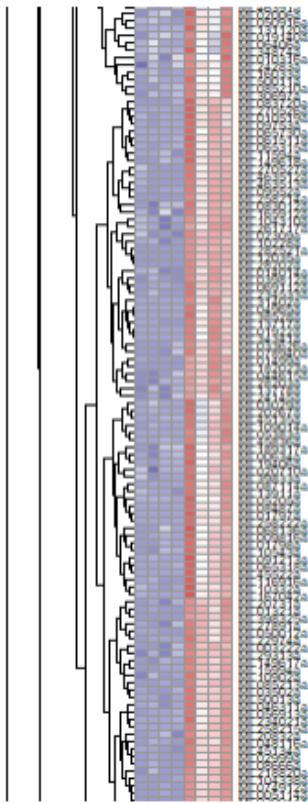
Project 3.

-
- 1) Build hierarchical trees based on the columns and for the rows (exclude rows that are "low" expression)



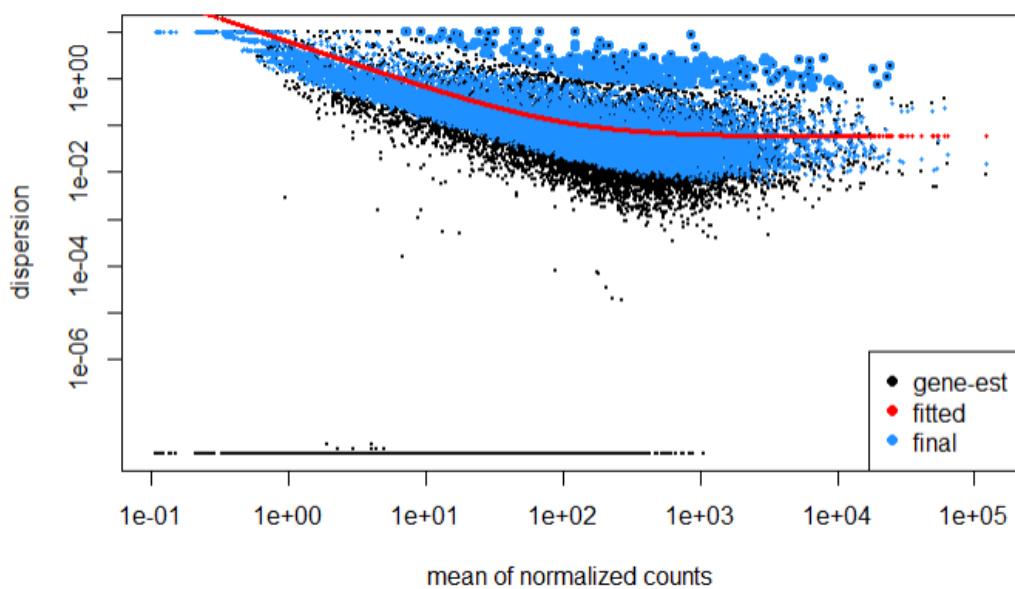
2) Draw a heat map of the expression data





3) Use DESeq2 to analyze this data

a) which are the most significantly changing genes in this dataset?



There are around 1574 genes that are differentially expressed in this dataset. This is determined using the criteria in which genes have an adjusted p-value of less than 0.05 and log2FoldChange value greater than 1 or less than -1 after performing differential expression using DeSeq. They are listed in the R script under Step 7. Several of them are listed here (first 100): "ML000125a" "ML000132a" "ML00016a" "ML000715a" "ML00083a" "ML00085a" "ML001116a" "ML001117a" "ML00114a" "ML001527a" "ML00154a" "ML00173a" "ML00191a" "ML00218a" "ML002221a" "ML002223a" "ML00229a" "ML00283a" "ML003015a" "ML003020a" "ML003022a" "ML003220a" "ML003222a" "ML003248a" "ML00342a" "ML00368a" "ML00372a" "ML00413a" "ML004415a" "ML004421a" "ML004510a" "ML004922a" "ML004923a" "ML00497a" "ML005030a" "ML00504a" "ML005114a" "ML005116a" "ML005119a" "ML00515a" "ML005211a" "ML005212a" "ML005214a" "ML005219a" "ML005324a" "ML005329a" "ML00576a" "ML006114a" "ML00613a" "ML00621a" "ML006313a" "ML00634a" "ML00653a" "ML00679a" "ML006910a" "ML006927a" "ML00692a" "ML006934a" "ML00694a" "ML00697a" "ML00699a" "ML00702a" "ML00704a" "ML00705a" "ML00732a" "ML007335a" "ML007414a" "ML007426a" "ML007430a" "ML00744a" "ML008015a" "ML008017a" "ML008029a" "ML008323a" "ML008325a" "ML008327a" "ML008715a" "ML00892a" "ML00906a" "ML009149a" "ML00918a" "ML009410a" "ML009411a" "ML00948a" "ML00949a" "ML009710a" "ML009811a" "ML009816a" "ML010112a" "ML010316a" "ML010519a" "ML01053a" "ML010623a" "ML01167a" "ML01248a" "ML013112a" "ML01321a" "ML01323a" "ML01348a" "ML01357a" "ML01358a" "ML01359a" "ML014014a" "ML01412a" "ML01433a" "ML01438a" "ML014423a" "ML01466a"

b) which genes are most consistently highly expressed in these datasets they are the "house-keeping" genes

These are the most consistently highly expressed genes in the dataset for each category and their respective expression values:

Aboral1:

```
#ML174735a ML034334a ML034337a ML034336a ML01482a ML46651a  
#17157.06 21911.36 22424.14 23631.30 30581.63 118211.07
```

Aboral2:

```
#ML174735a ML034337a ML034336a ML034334a ML01482a ML46651a  
#40418.84 58540.29 65914.50 68219.38 80559.11 93870.29
```

Aboral3:

```
#ML174735a ML034337a ML034336a ML034334a ML46651a ML01482a  
#27488.06 47004.61 50998.50 55714.11 56425.56 71249.61
```

Aboral4:

```
#ML174735a ML034337a ML034336a ML46651a ML034334a ML01482a
```

#37089.43 67521.80 82850.63 91834.25 92639.95 110042.52

Amongst the Aboral category, there are several genes that are consistently highly expressed. These genes include: ML174735a, ML034337a, ML034336a, ML034334a, ML46651a, ML01482a. This was performed by sorting the expression values for each category and extracting the top 5 genes for each category. It appears that each Aboral category (Aboral1-Aboral4) have the same 5 consistent highly expressed genes, but their orders are slightly different for each category.

Oral1:

#ML01433a ML087114a ML199832a ML004510a ML21583a ML306119a
#14912.24 17974.26 18039.35 22064.90 23649.09 31192.30

Oral2:

#ML11032a ML306119a ML34341a ML087114a ML199832a ML004510a
#16119.59 16538.89 18610.45 20852.11 21035.21 34514.43

Oral3:

#ML46651a ML43587a ML11032a ML087114a ML01482a ML004510a
#27853.26 29726.57 30124.16 42894.61 44780.12 66495.13

Oral4:

#ML01433a ML11032a ML09436a ML087114a ML46651a ML004510a
#26045.23 28540.28 33828.03 39617.47 64936.93 77464.55

Amongst the Oral category, there are several genes that are consistently highly expressed. These genes include: ML01433a, ML11032a, ML199832a, ML004510a, ML087114a, ML46651a, and ML306119a. The only genes that appear in every Oral category are genes ML087114a and ML004510a. Unlike the Aboral categories, in which the top five genes are the same for each Aboral category, there is more variety in the top highly expressed genes for the Oral categories.

c) How consistent are these results with the analysis you did in the midterm project ?

The top 5 genes identified in the midterm for the Aboral category using the p-value are ML01482a, ML03658a, ML034334a, ML034336a, ML46651a. On the other hand, the top five genes identified in the final project are: ML174735a, ML034337a, ML034336a, ML034334a, ML46651a, ML01482a. As we can see, the analysis performed does not appear to be consistent with previous results, due to the fact that the highly expressed genes in Aboral differ. The only gene that is consistent within both midterm and final projects is the gene ML034336a. This may be due to differences in the way that we calculated the adjusted p-value for each gene, compared to the method used by DeSeq.

Additionally, the top 5 genes identified for the oral category using the p-value are ML165814a, ML050913a, ML06904a, ML174758a, and ML210018a. On the other hand, the top five genes

identified in the final project are: ML01433a, ML11032a, ML199832a, ML004510a, ML087114a, ML46651a, and ML306119a. As we can see, the analysis performed does not appear to be consistent with previous results, due to the fact that the highly expressed genes in Oral differ. This may be due to differences in the way that we calculated the adjusted p-value for each gene, compared to the method used by DeSeq.

d) What else can you say about the data in terms of consistency, and the results that you find from your analyses. The question is open-ended, think of this as your experiment, you need to write a paper based on this data so you have to figure out what kind of "story" you can tell based on this.

In terms of consistency, we can see that the most highly expressed genes in each category remain relatively similar across the dataset in the final project. This is observed in the Aboral and Oral dataset, in which the most highly expressed genes are consistent throughout the entire dataset. However, when compared to the midterm results, the top genes do not appear to be consistent with one another. Both the midterm and final projects demonstrated consistency amongst the top five genes expressed in each Aboral and Oral category, but only within each project. In contrast, both the midterm and final results differ from one another. In order to determine the top genes, we calculated the p-value for each of the genes' expression value. However, in the midterm, the p-values were calculated by hand, whereas in the final, the p-values were calculated using DeSeq2. Therefore, to maintain consistent results, it is recommended that a consistent technique be used. We can tell that the genes remain relatively similar to one another, so when writing a paper, there is some form of consistency, we just need to determine which technique is the most accurate.

e) what is the most interesting pathway or gene that is responding in this study ?

ML46651a is the most interesting pathway responding in this study. This is due to the fact that the ML46651a gene is the most highly expressed in each Aboral and Oral category, therefore making it a consistently expressed gene throughout the entire dataset. This gene is a cellular component that is part of the membrane attack complex.