|  |  |
| --- | --- |
| Setup | Cross-Validation Accuracy |
| Unprocessed Data | 0.752273 |
| 0-Elements ignored | 0.715139 |

Calculation of Distribution parameters

#Takes in a column vector from a data frame and fits a normal curve to

#it by finding the maximum likelihood estimates of the parameters of

#a normal distribution (i.e. mean and sd), given the data in the

#column. If ignore\_zeros is true, it'll ignore the zeros in the

#column

fit\_normal <- function(col, ignore\_zeros) {

if (ignore\_zeros == F) {

return(list(mean(col), sd(col)))

} else {

zeros\_excluded <- col[which(col != 0)]

return(list(mean(zeros\_excluded), sd(zeros\_excluded)))

}

}

#Finds the parameters of the normal distribution fitted to the values

#of the rows of the training data frame that correspond to people

#with diabetes (i.e. class == 1)

param\_preg\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"preg"], F)

param\_plas\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"plas"], F)

param\_pres\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"pres"], F)

param\_skin\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"skin"], F)

param\_test\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"test"], F)

param\_mass\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"mass"], F)

param\_pedi\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"pedi"], F)

param\_age\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"age"], F)

#Finds the parameters of the normal distribution fitted to the values

#of the rows of the training data frame that correspond to people

#without diabetes (i.e. class == 0)

param\_preg\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"preg"], F)

param\_plas\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"plas"], F)

param\_pres\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"pres"], F)

param\_skin\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"skin"], F)

param\_test\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"test"], F)

param\_mass\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"mass"], F)

param\_pedi\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"pedi"], F)

param\_age\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"age"], F)

Calculating the Naïve Bayes’ Predictions

#Goes through the test data, and determines the posterior probability

#of the person having diabetes given the feature data

prob\_of\_diabetes\_yes <- length(which(training\_df$class == 1)) / nrow(training\_df)

posterior\_prob\_of\_diabetes\_yes <- log(dnorm(testing\_df[j,"preg"], param\_preg\_yes[[1]], param\_preg\_yes[[2]])) +

log(dnorm(testing\_df[j,"plas"], param\_plas\_yes[[1]], param\_plas\_yes[[2]])) +

log(dnorm(testing\_df[j,"pres"], param\_pres\_yes[[1]], param\_pres\_yes[[2]])) +

log(dnorm(testing\_df[j,"skin"], param\_skin\_yes[[1]], param\_skin\_yes[[2]])) +

log(dnorm(testing\_df[j,"test"], param\_test\_yes[[1]], param\_test\_yes[[2]])) +

log(dnorm(testing\_df[j,"mass"], param\_mass\_yes[[1]], param\_mass\_yes[[2]])) +

log(dnorm(testing\_df[j,"pedi"], param\_pedi\_yes[[1]], param\_pedi\_yes[[2]])) +

log(dnorm(testing\_df[j,"age"], param\_age\_yes[[1]], param\_age\_yes[[2]])) +

log(prob\_of\_diabetes\_yes)

#Goes through the test data, and determines the posterior probability

#of the person not having diabetes given the feature data

prob\_of\_diabetes\_no <- length(which(training\_df$class == 1)) / nrow(training\_df)

posterior\_prob\_of\_diabetes\_no <- log(dnorm(testing\_df[j,"preg"], param\_preg\_no[[1]], param\_preg\_no[[2]])) +

log(dnorm(testing\_df[j,"plas"], param\_plas\_no[[1]], param\_plas\_no[[2]])) +

log(dnorm(testing\_df[j,"pres"], param\_pres\_no[[1]], param\_pres\_no[[2]])) +

log(dnorm(testing\_df[j,"skin"], param\_skin\_no[[1]], param\_skin\_no[[2]])) +

log(dnorm(testing\_df[j,"test"], param\_test\_no[[1]], param\_test\_no[[2]])) +

log(dnorm(testing\_df[j,"mass"], param\_mass\_no[[1]], param\_mass\_no[[2]])) +

log(dnorm(testing\_df[j,"pedi"], param\_pedi\_no[[1]], param\_pedi\_no[[2]])) +

log(dnorm(testing\_df[j,"age"], param\_age\_no[[1]], param\_age\_no[[2]])) +

log(prob\_of\_diabetes\_no)

classified <- 0

if (posterior\_prob\_of\_diabetes\_yes >= posterior\_prob\_of\_diabetes\_no) {

classified <- 1

}

Test-train split code

#Given a number, gives ~80% of them a lable of 1 and 20% of them a

#label of 0. The number of rows in a data frame will be passed here

#to split them into test-train groups.

test\_train\_split <- function(num\_rows) {

newCol <- rbinom(n = num\_rows, size = 1, prob = 0.8)

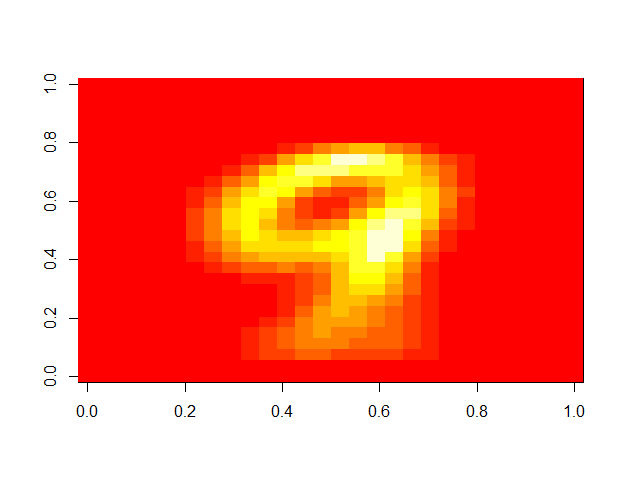
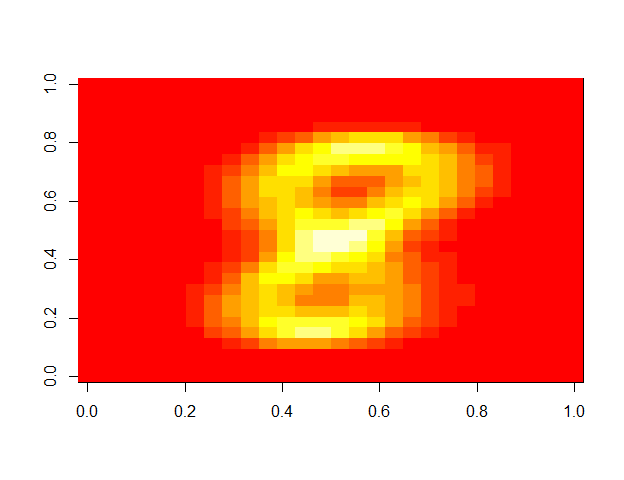
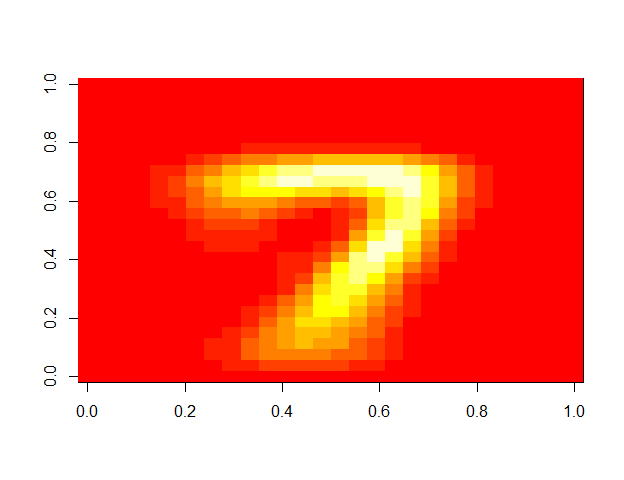
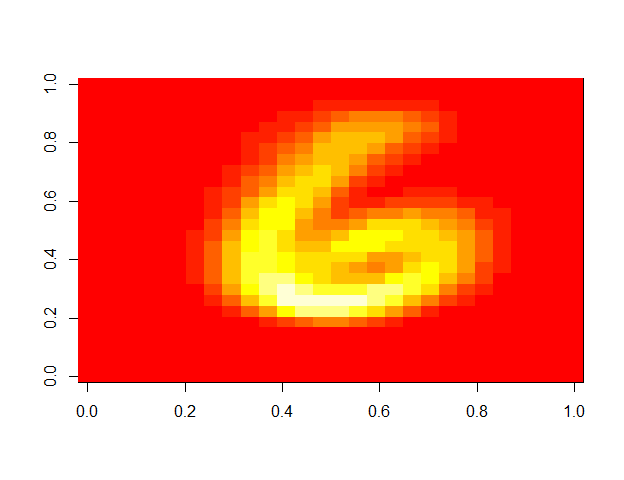
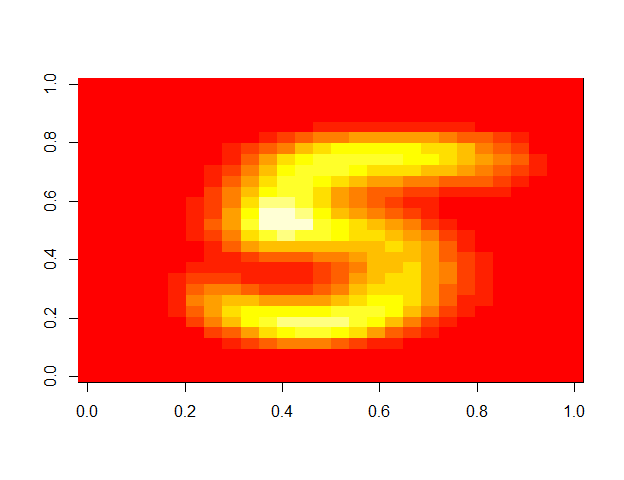
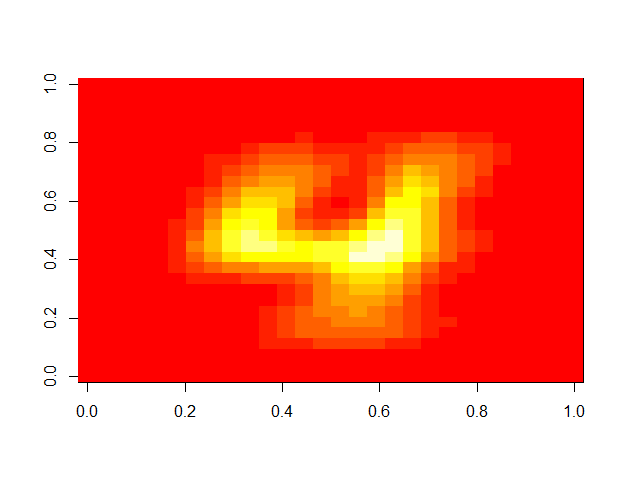
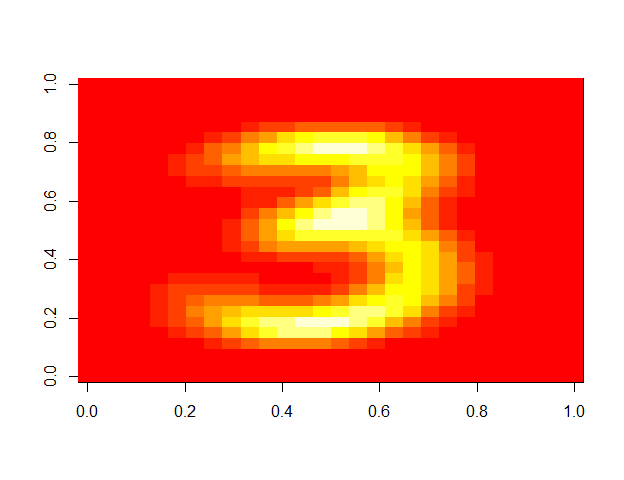
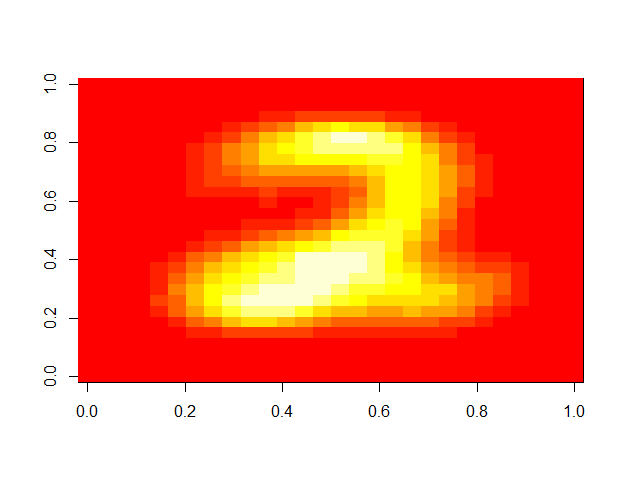
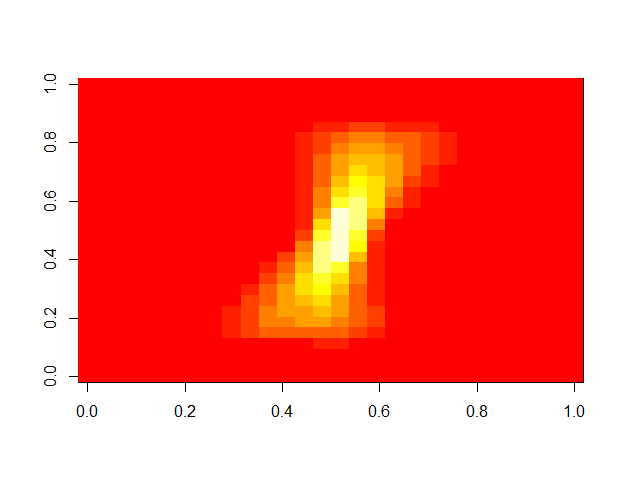
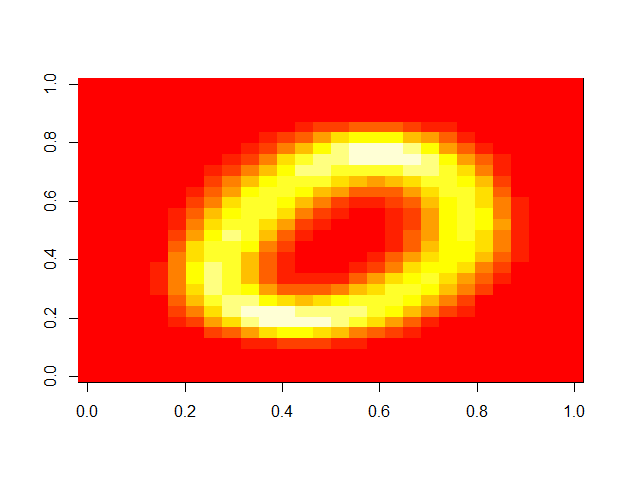
split <- list(which(newCol == 1), which(newCol == 0))

return(split)

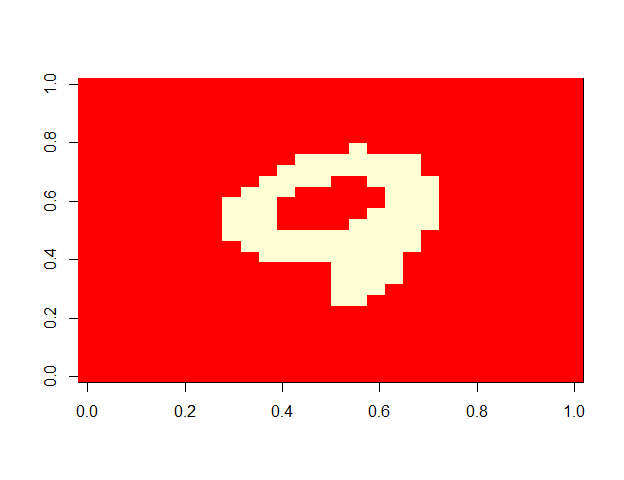
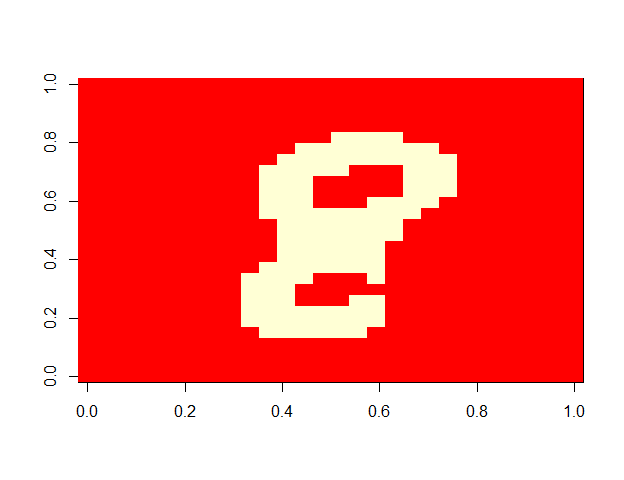
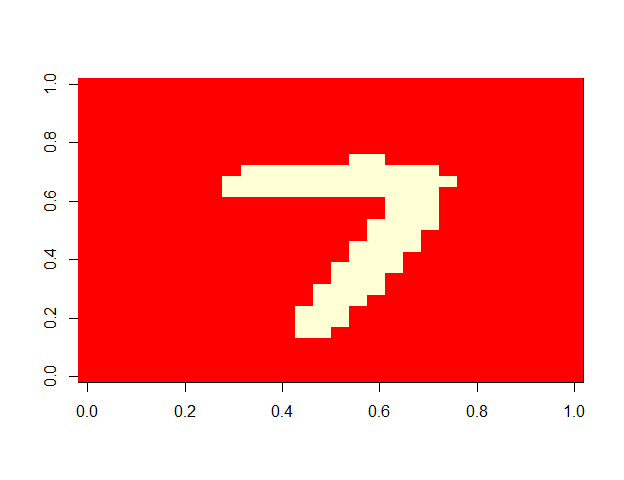
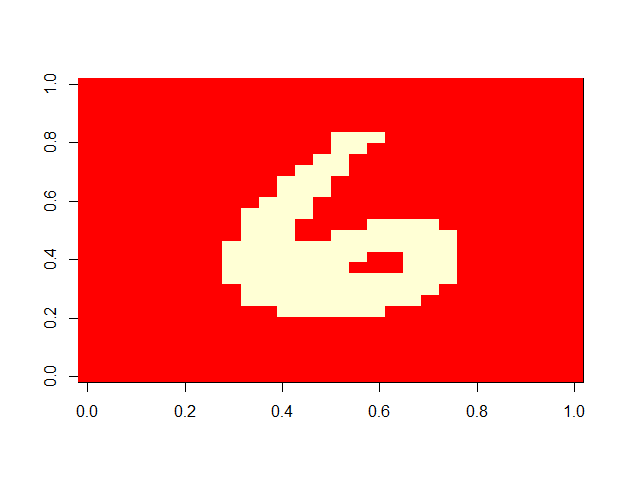
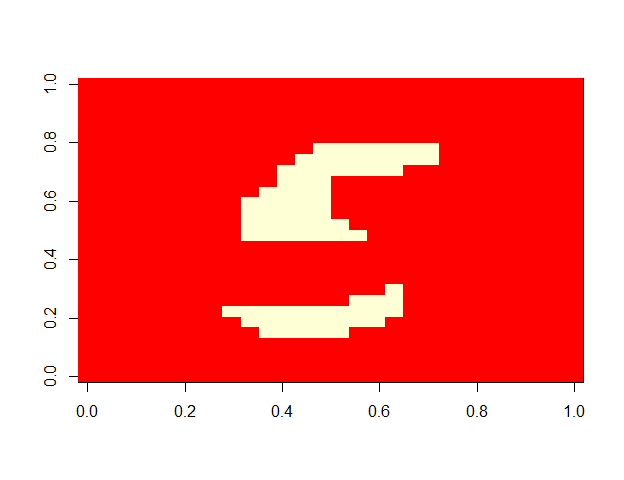
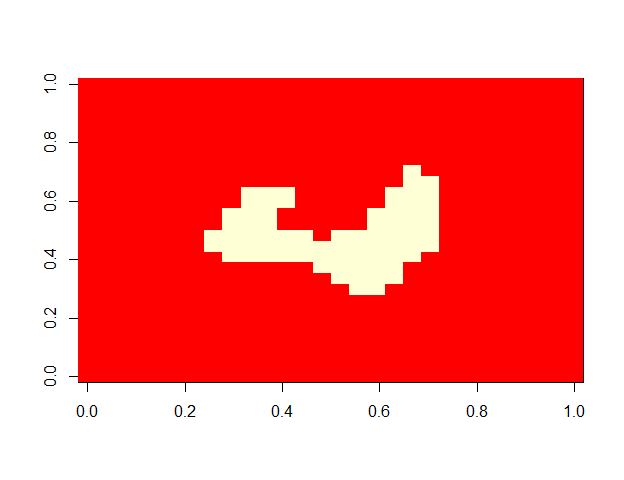
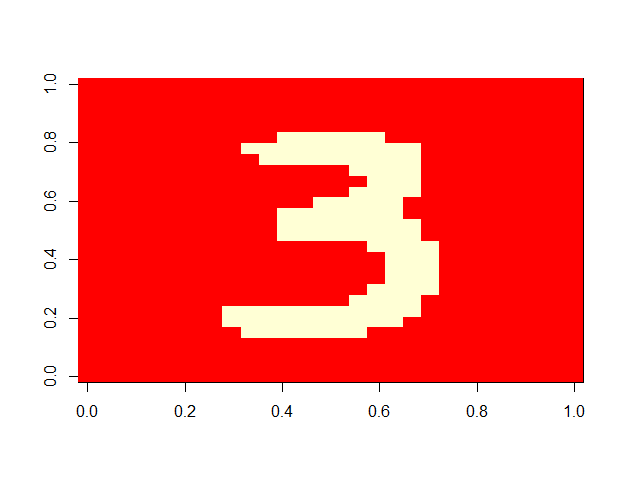
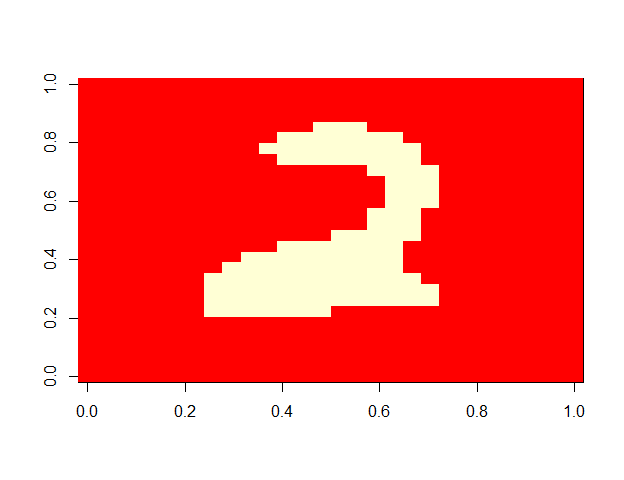
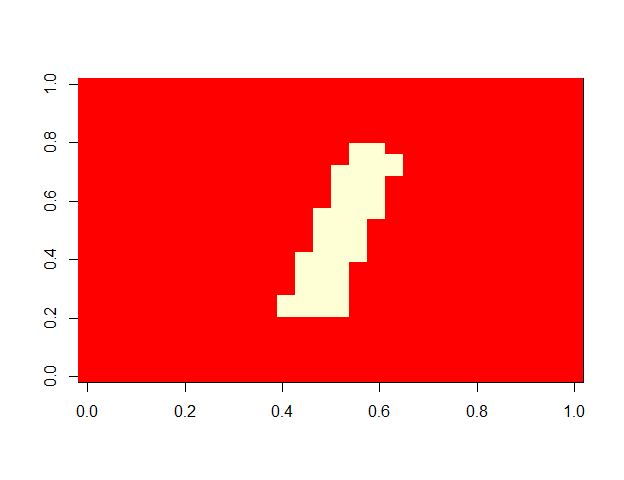
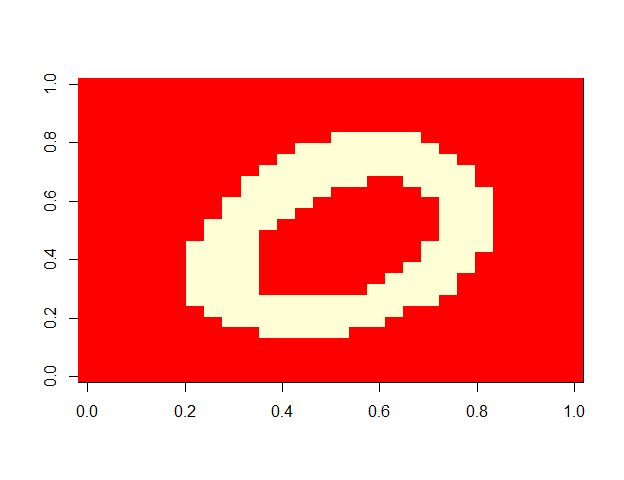
}

|  |  |  |
| --- | --- | --- |
| Method | Testing Set Accuracy | Training Set Accuracy |
| Gaussian + untouched | 0.7239 | 0.7082 |
| Gaussian + stretched | 0.8362 | 0.8255 |
| Bernoulli + untouched | 0.8442 | 0.8382 |
| Bernoulli + stretched | 0.8176 | 0.8035 |
| 10 trees + 4 depth + untouched | 0.8511 | 0.8447 |
| 10 trees + 4 depth + stretched | 0.8540 | 0.8506 |
| 10 trees + 16 depth + untouched | 0.9550 | 0.9981 |
| 10 trees + 16 depth + stretched | 0.9595 | 0.9990 |
| 30 trees + 4 depth + untouched | 0.8669 | 0.8626 |
| 30 trees + 4 depth + stretched | 0.8743 | 0.8726 |
| 30 trees + 16 depth + untouched | 0.9641 | 0.9989 |
| 30 trees + 16 depth + stretched | 0.9687 | 0.9998 |

Oritinal Mean Images



After applying threshold of 110



Normal Distribution Parameter Calculation

#Finds average training image for each digit

avg\_imgs <- list(

avg\_img\_0 <- find\_average\_image(training\_images[which(vec == 0)]),

avg\_img\_1 <- find\_average\_image(training\_images[which(vec == 1)]),

…

avg\_img\_9 <- find\_average\_image(training\_images[which(vec == 9)])

)

avg\_imgs <- rapply(avg\_imgs, function(x) ifelse(x > 110, 255, 0), how = "replace")

#Finds sd for each digit

sd\_imgs <- list(

sd\_img\_0 <- find\_sd\_image(training\_images[which(vec == 0)]),

sd\_img\_1 <- find\_sd\_image(training\_images[which(vec == 1)]),

…

sd\_img\_9 <- find\_sd\_image(training\_images[which(vec == 9)])

)

Bernoulli Distribution Parameters

bern\_avg\_imgs <- list(

bern\_avg\_img\_0 <- bern\_find\_average\_image(training\_images[which(vec == 0)]),

bern\_avg\_img\_1 <- bern\_find\_average\_image(training\_images[which(vec == 1)]),

…

bern\_avg\_img\_9 <- bern\_find\_average\_image(training\_images[which(vec == 9)])

)

Calculation of Naïve Bayes’ predictions. These function are for the calculation of the untouched image prediction. The function for the touched images are almost identical

#Given a matrix, finds the posterior distributions for each digit,

#and returns the most likely value

determine\_digit <- function(mat) {

lst\_of\_posterior\_probs <- c()

for (k in 0:9) {

avg\_img <- avg\_imgs[[k+1]]

sd\_img <- sd\_imgs[[k+1]]

accum <- 0.1 + sum(log(0.0000000001 + dnorm(mat, avg\_img, sd\_img + 10)))

lst\_of\_posterior\_probs <- c(lst\_of\_posterior\_probs, accum)

}

return(which(lst\_of\_posterior\_probs == max(lst\_of\_posterior\_probs))[1] - 1)

}

#Given a matrix, finds the posterior distributions for each digit,

#and returns the most likely value

bern\_determine\_digit <- function(mat) {

mat[which(mat != 0)] <- 1

lst\_of\_posterior\_probs <- c()

for (k in 0:9) {

avg\_img <- bern\_avg\_imgs[[k+1]]

accum <- 0.1 + sum(log(0.0000000001 + dbinom(mat, avg\_img, size = 1))) # for (i in 1:nrow(mat)) {

lst\_of\_posterior\_probs <- c(lst\_of\_posterior\_probs, accum)

}

return(which(lst\_of\_posterior\_probs == max(lst\_of\_posterior\_probs))[1] - 1)

}

gaussian\_untouched\_guesses <- lapply(testing\_images, determine\_digit)

gaussian\_untouched\_guesses\_training <- lapply(training\_images, determine\_digit)

bern\_untouched\_guesses <- lapply(testing\_images, bern\_determine\_digit)

bern\_untouched\_guesses\_training <- lapply(training\_images, bern\_determine\_digit)

Training of decision trees

#Creates the random forest models

model\_training\_untouched\_10\_4 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_untouched,

ntrees = 10, max\_depth = 4)

model\_training\_touched\_10\_4 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_touched,

ntrees = 10, max\_depth = 4)

...

model\_training\_touched\_30\_16 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_touched,

ntrees = 30, max\_depth = 16)

Calculation of decision tree prediction

#Predictions

testing\_untouched\_10\_4 <- predict(model\_training\_untouched\_10\_4, h2o\_df\_testing\_untouched)

length(which(as.vector(testing\_untouched\_10\_4[,1]) == vec2))

testing\_untouched\_10\_16 <- predict(model\_training\_untouched\_10\_16, h2o\_df\_testing\_untouched)

length(which(as.vector(testing\_untouched\_10\_16[,1]) == vec2))

...

testing\_touched\_30\_16 <- predict(model\_training\_touched\_30\_16, h2o\_df\_testing\_touched)

length(which(as.vector(testing\_touched\_30\_16[,1]) == vec2))

**Full Code**

**Question 1**

library(ggplot2)

data <- read.csv("pima-indians-diabetes.csv", header = T)

names(data) <- c("preg", "plas", "pres", "skin", "test", "mass", "pedi",

"age", "class")

#Given a number, gives ~80% of them a lable of 1 and 20% of them a

#label of 0. The number of rows in a data frame will be passed here

#to split them into test-train groups.

test\_train\_split <- function(num\_rows) {

newCol <- rbinom(n = num\_rows, size = 1, prob = 0.8)

split <- list(which(newCol == 1), which(newCol == 0))

return(split)

}

#Takes in a column vector from a data frame and fits a normal curve to

#it by finding the maximum likelihood estimates of the parameters of

#a normal distribution (i.e. mean and sd), given the data in the

#column. If ignore\_zeros is true, it'll ignore the zeros in the

#column

fit\_normal <- function(col, ignore\_zeros) {

if (ignore\_zeros == F) {

return(list(mean(col), sd(col)))

} else {

zeros\_excluded <- col[which(col != 0)]

return(list(mean(zeros\_excluded), sd(zeros\_excluded)))

}

}

#######PART A##########

accuracy <- c()

for (i in 1:10) {

#Gets a test-train split

lst <- test\_train\_split(nrow(data))

training\_df <- data[lst[[1]],]

testing\_df <- data[lst[[2]],]

#Finds the parameters of the normal distribution fitted to the values

#of the rows of the training data frame that correspond to people

#with diabetes (i.e. class == 1)

param\_preg\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"preg"], F)

param\_plas\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"plas"], F)

param\_pres\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"pres"], F)

param\_skin\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"skin"], F)

param\_test\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"test"], F)

param\_mass\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"mass"], F)

param\_pedi\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"pedi"], F)

param\_age\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"age"], F)

#Finds the parameters of the normal distribution fitted to the values

#of the rows of the training data frame that correspond to people

#without diabetes (i.e. class == 0)

param\_preg\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"preg"], F)

param\_plas\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"plas"], F)

param\_pres\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"pres"], F)

param\_skin\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"skin"], F)

param\_test\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"test"], F)

param\_mass\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"mass"], F)

param\_pedi\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"pedi"], F)

param\_age\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"age"], F)

num\_correct <- 0

for (j in 1:nrow(testing\_df)) {

#Goes through the test data, and determines the posterior probability

#of the person having diabetes given the feature data

prob\_of\_diabetes\_yes <- length(which(training\_df$class == 1)) / nrow(training\_df)

posterior\_prob\_of\_diabetes\_yes <- log(dnorm(testing\_df[j,"preg"], param\_preg\_yes[[1]], param\_preg\_yes[[2]])) +

log(dnorm(testing\_df[j,"plas"], param\_plas\_yes[[1]], param\_plas\_yes[[2]])) +

log(dnorm(testing\_df[j,"pres"], param\_pres\_yes[[1]], param\_pres\_yes[[2]])) +

log(dnorm(testing\_df[j,"skin"], param\_skin\_yes[[1]], param\_skin\_yes[[2]])) +

log(dnorm(testing\_df[j,"test"], param\_test\_yes[[1]], param\_test\_yes[[2]])) +

log(dnorm(testing\_df[j,"mass"], param\_mass\_yes[[1]], param\_mass\_yes[[2]])) +

log(dnorm(testing\_df[j,"pedi"], param\_pedi\_yes[[1]], param\_pedi\_yes[[2]])) +

log(dnorm(testing\_df[j,"age"], param\_age\_yes[[1]], param\_age\_yes[[2]])) +

log(prob\_of\_diabetes\_yes)

#Goes through the test data, and determines the posterior probability

#of the person not having diabetes given the feature data

prob\_of\_diabetes\_no <- length(which(training\_df$class == 1)) / nrow(training\_df)

posterior\_prob\_of\_diabetes\_no <- log(dnorm(testing\_df[j,"preg"], param\_preg\_no[[1]], param\_preg\_no[[2]])) +

log(dnorm(testing\_df[j,"plas"], param\_plas\_no[[1]], param\_plas\_no[[2]])) +

log(dnorm(testing\_df[j,"pres"], param\_pres\_no[[1]], param\_pres\_no[[2]])) +

log(dnorm(testing\_df[j,"skin"], param\_skin\_no[[1]], param\_skin\_no[[2]])) +

log(dnorm(testing\_df[j,"test"], param\_test\_no[[1]], param\_test\_no[[2]])) +

log(dnorm(testing\_df[j,"mass"], param\_mass\_no[[1]], param\_mass\_no[[2]])) +

log(dnorm(testing\_df[j,"pedi"], param\_pedi\_no[[1]], param\_pedi\_no[[2]])) +

log(dnorm(testing\_df[j,"age"], param\_age\_no[[1]], param\_age\_no[[2]])) +

log(prob\_of\_diabetes\_no)

classified <- 0

if (posterior\_prob\_of\_diabetes\_yes >= posterior\_prob\_of\_diabetes\_no) {

classified <- 1

}

if (classified == testing\_df[j, "class"]) {

num\_correct <- num\_correct + 1

}

}

accuracy <- c(accuracy, num\_correct / nrow(testing\_df))

}

average\_accuracy <- mean(accuracy)

#######PART B##########

#Almost the same as PART A. Teh only idfference is that in some of the

#columns, the zeros are being considered as missing values and excluded.

accuracy <- c()

for (i in 1:10) {

#Gets a test-train split

lst <- test\_train\_split(nrow(data))

training\_df <- data[lst[[1]],]

testing\_df <- data[lst[[2]],]

#Finds the parameters of the normal distribution fitted to the values

#of the rows of the training data frame that correspond to people

#with diabetes (i.e. class == 1)

param\_preg\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"preg"], F)

param\_plas\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"plas"], F)

param\_pres\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"pres"], T)

param\_skin\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"skin"], T)

param\_test\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"test"], F)

param\_mass\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"mass"], T)

param\_pedi\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"pedi"], F)

param\_age\_yes <- fit\_normal(training\_df[which(training\_df$class == 1),"age"], T)

#Finds the parameters of the normal distribution fitted to the values

#of the rows of the training data frame that correspond to people

#without diabetes (i.e. class == 0)

param\_preg\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"preg"], F)

param\_plas\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"plas"], F)

param\_pres\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"pres"], T)

param\_skin\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"skin"], T)

param\_test\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"test"], F)

param\_mass\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"mass"], T)

param\_pedi\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"pedi"], F)

param\_age\_no <- fit\_normal(training\_df[which(training\_df$class == 0),"age"], T)

num\_correct <- 0

for (j in 1:nrow(testing\_df)) {

#Goes through the test data, and determines the posterior probability

#of the person having diabetes given the feature data

prob\_of\_diabetes\_yes <- length(which(training\_df$class == 1)) / nrow(training\_df)

posterior\_prob\_of\_diabetes\_yes <- log(dnorm(testing\_df[j,"preg"], param\_preg\_yes[[1]], param\_preg\_yes[[2]])) +

log(dnorm(testing\_df[j,"plas"], param\_plas\_yes[[1]], param\_plas\_yes[[2]])) +

log(dnorm(testing\_df[j,"pres"], param\_pres\_yes[[1]], param\_pres\_yes[[2]])) +

log(dnorm(testing\_df[j,"skin"], param\_skin\_yes[[1]], param\_skin\_yes[[2]])) +

log(dnorm(testing\_df[j,"test"], param\_test\_yes[[1]], param\_test\_yes[[2]])) +

log(dnorm(testing\_df[j,"mass"], param\_mass\_yes[[1]], param\_mass\_yes[[2]])) +

log(dnorm(testing\_df[j,"pedi"], param\_pedi\_yes[[1]], param\_pedi\_yes[[2]])) +

log(dnorm(testing\_df[j,"age"], param\_age\_yes[[1]], param\_age\_yes[[2]])) +

log(prob\_of\_diabetes\_yes)

#Goes through the test data, and determines the posterior probability

#of the person not having diabetes given the feature data

prob\_of\_diabetes\_no <- length(which(training\_df$class == 1)) / nrow(training\_df)

posterior\_prob\_of\_diabetes\_no <- log(dnorm(testing\_df[j,"preg"], param\_preg\_no[[1]], param\_preg\_no[[2]])) +

log(dnorm(testing\_df[j,"plas"], param\_plas\_no[[1]], param\_plas\_no[[2]])) +

log(dnorm(testing\_df[j,"pres"], param\_pres\_no[[1]], param\_pres\_no[[2]])) +

log(dnorm(testing\_df[j,"skin"], param\_skin\_no[[1]], param\_skin\_no[[2]])) +

log(dnorm(testing\_df[j,"test"], param\_test\_no[[1]], param\_test\_no[[2]])) +

log(dnorm(testing\_df[j,"mass"], param\_mass\_no[[1]], param\_mass\_no[[2]])) +

log(dnorm(testing\_df[j,"pedi"], param\_pedi\_no[[1]], param\_pedi\_no[[2]])) +

log(dnorm(testing\_df[j,"age"], param\_age\_no[[1]], param\_age\_no[[2]])) +

log(prob\_of\_diabetes\_no)

classified <- 0

if (posterior\_prob\_of\_diabetes\_yes >= posterior\_prob\_of\_diabetes\_no) {

classified <- 1

}

if (classified == testing\_df[j, "class"]) {

num\_correct <- num\_correct + 1

}

}

accuracy <- c(accuracy, num\_correct / nrow(testing\_df))

}

average\_accuracy <- mean(accuracy)

**Question 2**

library(h2o)

#Stores training labels in a vector called vec

to.read.label = file("train-labels.idx1-ubyte", "rb")

readBin(to.read.label, integer(), n=2, size=4, endian="big")

vec <- readBin(to.read.label, integer(), n=60000, size=1, endian="big")

#Stores training images in a list called images

to.read.img = file("train-images.idx3-ubyte", "rb")

readBin(to.read.img, integer(), n=4, endian="big")

training\_images <- list()

training\_images\_size <- 1

for(i in 1:60000) {

m = matrix(readBin(to.read.img,integer(), size=1, n=28\*28, endian="big", signed = F),28,28)

m <- m[,28:1]

training\_images[[training\_images\_size]] <- m

training\_images\_size <- training\_images\_size + 1

}

training\_images <- rapply(training\_images, function(x) ifelse(x > 128, 255, 0), how = "replace")

#Testing Thresholding on a small list

# img\_lst <- list(training\_images[[3]], training\_images[[321]])

# img\_lst <- rapply(img\_lst, function(x) ifelse(x > 100, 255, 0), how = "replace")

#Stores training labels in a vector called vec

to.read.label = file("t10k-labels.idx1-ubyte", "rb")

readBin(to.read.label, integer(), n=2, size=4, endian="big")

vec2 <- readBin(to.read.label, integer(), n=10000, size=1, endian="big")

#Stores training images in a list called images

to.read.img = file("t10k-images.idx3-ubyte", "rb")

readBin(to.read.img, integer(), n=4, endian="big")

testing\_images <- list()

testing\_images\_size <- 1

for(i in 1:10000) {

m = matrix(readBin(to.read.img,integer(), size=1, n=28\*28, endian="big", signed = F),28,28)

m <- m[,28:1]

testing\_images[[testing\_images\_size]] <- m

testing\_images\_size <- testing\_images\_size + 1

}

testing\_images <- rapply(testing\_images, function(x) ifelse(x > 128, 255, 0), how = "replace")

#Finds the average value for each pixel and returns the resulting

#matrix

find\_average\_image <- function(lst\_of\_images) {

return(Reduce('+', lst\_of\_images)/length(lst\_of\_images))

}

bern\_find\_average\_image <- function(lst\_of\_images) {

lst <- lapply(lst\_of\_images, function(x) {x[which(x != 0)] <- 1; return(x)})

return(Reduce('+', lst)/length(lst))

}

#Finds the standard deviation for each pixel and returns the resulting

#matrix

#Code obtained from: https://stackoverflow.com/questions/39351013/standard-deviation-over-a-list-of-matrices-in-r

find\_sd\_image <- function(lst) {

n <- length(lst)

rc <- dim(lst[[1]])

ar1 <- array(unlist(lst), c(rc, n))

round(apply(ar1, c(1, 2), sd), 2)

}

#Finds average training image for each digit

avg\_imgs <- list(

avg\_img\_0 <- find\_average\_image(training\_images[which(vec == 0)]),

avg\_img\_1 <- find\_average\_image(training\_images[which(vec == 1)]),

avg\_img\_2 <- find\_average\_image(training\_images[which(vec == 2)]),

avg\_img\_3 <- find\_average\_image(training\_images[which(vec == 3)]),

avg\_img\_4 <- find\_average\_image(training\_images[which(vec == 4)]),

avg\_img\_5 <- find\_average\_image(training\_images[which(vec == 5)]),

avg\_img\_6 <- find\_average\_image(training\_images[which(vec == 6)]),

avg\_img\_7 <- find\_average\_image(training\_images[which(vec == 7)]),

avg\_img\_8 <- find\_average\_image(training\_images[which(vec == 8)]),

avg\_img\_9 <- find\_average\_image(training\_images[which(vec == 9)])

)

avg\_imgs <- rapply(avg\_imgs, function(x) ifelse(x > 110, 255, 0), how = "replace")

#Finds sd for each digit

sd\_imgs <- list(

sd\_img\_0 <- find\_sd\_image(training\_images[which(vec == 0)]),

sd\_img\_1 <- find\_sd\_image(training\_images[which(vec == 1)]),

sd\_img\_2 <- find\_sd\_image(training\_images[which(vec == 2)]),

sd\_img\_3 <- find\_sd\_image(training\_images[which(vec == 3)]),

sd\_img\_4 <- find\_sd\_image(training\_images[which(vec == 4)]),

sd\_img\_5 <- find\_sd\_image(training\_images[which(vec == 5)]),

sd\_img\_6 <- find\_sd\_image(training\_images[which(vec == 6)]),

sd\_img\_7 <- find\_sd\_image(training\_images[which(vec == 7)]),

sd\_img\_8 <- find\_sd\_image(training\_images[which(vec == 8)]),

sd\_img\_9 <- find\_sd\_image(training\_images[which(vec == 9)])

)

bern\_avg\_imgs <- list(

bern\_avg\_img\_0 <- bern\_find\_average\_image(training\_images[which(vec == 0)]),

bern\_avg\_img\_1 <- bern\_find\_average\_image(training\_images[which(vec == 1)]),

bern\_avg\_img\_2 <- bern\_find\_average\_image(training\_images[which(vec == 2)]),

bern\_avg\_img\_3 <- bern\_find\_average\_image(training\_images[which(vec == 3)]),

bern\_avg\_img\_4 <- bern\_find\_average\_image(training\_images[which(vec == 4)]),

bern\_avg\_img\_5 <- bern\_find\_average\_image(training\_images[which(vec == 5)]),

bern\_avg\_img\_6 <- bern\_find\_average\_image(training\_images[which(vec == 6)]),

bern\_avg\_img\_7 <- bern\_find\_average\_image(training\_images[which(vec == 7)]),

bern\_avg\_img\_8 <- bern\_find\_average\_image(training\_images[which(vec == 8)]),

bern\_avg\_img\_9 <- bern\_find\_average\_image(training\_images[which(vec == 9)])

)

#Given a matrix, finds the posterior distributions for each digit,

#and returns the most likely value

determine\_digit <- function(mat) {

lst\_of\_posterior\_probs <- c()

for (k in 0:9) {

avg\_img <- avg\_imgs[[k+1]]

sd\_img <- sd\_imgs[[k+1]]

accum <- 0.1 + sum(log(0.0000000001 + dnorm(mat, avg\_img, sd\_img + 10)))

lst\_of\_posterior\_probs <- c(lst\_of\_posterior\_probs, accum)

}

return(which(lst\_of\_posterior\_probs == max(lst\_of\_posterior\_probs))[1] - 1)

}

#Given a matrix, finds the posterior distributions for each digit,

#and returns the most likely value

bern\_determine\_digit <- function(mat) {

mat[which(mat != 0)] <- 1

lst\_of\_posterior\_probs <- c()

for (k in 0:9) {

avg\_img <- bern\_avg\_imgs[[k+1]]

accum <- 0.1 + sum(log(0.0000000001 + dbinom(mat, avg\_img, size = 1))) # for (i in 1:nrow(mat)) {

lst\_of\_posterior\_probs <- c(lst\_of\_posterior\_probs, accum)

}

return(which(lst\_of\_posterior\_probs == max(lst\_of\_posterior\_probs))[1] - 1)

}

gaussian\_untouched\_guesses <- lapply(testing\_images, determine\_digit)

gaussian\_untouched\_guesses\_training <- lapply(training\_images, determine\_digit)

bern\_untouched\_guesses <- lapply(testing\_images, bern\_determine\_digit)

bern\_untouched\_guesses\_training <- lapply(training\_images, bern\_determine\_digit)

length(which(gaussian\_untouched\_guesses == vec2))

length(which(bern\_untouched\_guesses == vec2))

length(which(gaussian\_untouched\_guesses\_training == vec))

length(which(bern\_untouched\_guesses\_training == vec))

find\_vertical\_ink\_range <- function(img) {

top <- min(apply(img, 2, function(x) return(min(which(x != 0)))))

bottom <- max(apply(img, 2, function(x) return(max(which(x != 0)))))

return(c(top, bottom))

}

find\_horizontal\_ink\_range <- function(img) {

left <- min(apply(img, 1, function(x) return(min(which(x != 0)))))

right <- max(apply(img, 1, function(x) return(max(which(x != 0)))))

return(c(left, right))

}

#Resizes an image to a given width and height. Returns the matrix

#Code obtained from https://stackoverflow.com/questions/35786744/resizing-image-in-r

resize\_image <- function(img, new\_width, new\_height) {

new\_img = apply(img, 2, function(y){return (spline(y, n = new\_height)$y)})

new\_img = t(apply(new\_img, 1, function(y){return (spline(y, n = new\_width)$y)}))

new\_img[new\_img < 0] = 0

new\_img = round(new\_img)

return (new\_img)

}

#A vector with the numbers in the list images, cropped by 4 pixels on

#each side.

bounded\_box\_images\_training <- lapply(training\_images, function(x) {

vert <- find\_vertical\_ink\_range(x);

hor <- find\_horizontal\_ink\_range(x);

return(x[vert[1]:vert[2], hor[1]:hor[2]])

})

#bounded\_box\_images\_training <- rapply(bounded\_box\_images\_training, function(x) ifelse(x > 220, 255, 0), how = "replace")

#A vector with the numbers in the list images, cropped by 4 pixels on

#each side.

bounded\_box\_images\_testing <- lapply(testing\_images, function(x) {

vert <- find\_vertical\_ink\_range(x);

hor <- find\_horizontal\_ink\_range(x);

return(x[vert[1]:vert[2], hor[1]:hor[2]])

})

#bounded\_box\_images\_testing <- rapply(bounded\_box\_images\_testing, function(x) ifelse(x > 220, 255, 0), how = "replace")

resized\_bounded\_box\_images\_training <- lapply(bounded\_box\_images\_training, resize\_image, 20, 20)

resized\_bounded\_box\_images\_testing <- lapply(bounded\_box\_images\_testing, resize\_image, 20, 20)

touched\_avg\_imgs <- list(

touched\_avg\_img\_0 <- find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 0)]),

touched\_avg\_img\_1 <- find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 1)]),

touched\_avg\_img\_2 <- find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 2)]),

touched\_avg\_img\_3 <- find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 3)]),

touched\_avg\_img\_4 <- find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 4)]),

touched\_avg\_img\_5 <- find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 5)]),

touched\_avg\_img\_6 <- find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 6)]),

touched\_avg\_img\_7 <- find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 7)]),

touched\_avg\_img\_8 <- find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 8)]),

touched\_avg\_img\_9 <- find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 9)])

)

touched\_avg\_imgs <- rapply(touched\_avg\_imgs, function(x) ifelse(x > 110, 255, 0), how = "replace")

touched\_sd\_imgs <- list(

touched\_sd\_img\_0 <- find\_sd\_image(resized\_bounded\_box\_images\_training[which(vec == 0)]),

touched\_sd\_img\_1 <- find\_sd\_image(resized\_bounded\_box\_images\_training[which(vec == 1)]),

touched\_sd\_img\_2 <- find\_sd\_image(resized\_bounded\_box\_images\_training[which(vec == 2)]),

touched\_sd\_img\_3 <- find\_sd\_image(resized\_bounded\_box\_images\_training[which(vec == 3)]),

touched\_sd\_img\_4 <- find\_sd\_image(resized\_bounded\_box\_images\_training[which(vec == 4)]),

touched\_sd\_img\_5 <- find\_sd\_image(resized\_bounded\_box\_images\_training[which(vec == 5)]),

touched\_sd\_img\_6 <- find\_sd\_image(resized\_bounded\_box\_images\_training[which(vec == 6)]),

touched\_sd\_img\_7 <- find\_sd\_image(resized\_bounded\_box\_images\_training[which(vec == 7)]),

touched\_sd\_img\_8 <- find\_sd\_image(resized\_bounded\_box\_images\_training[which(vec == 8)]),

touched\_sd\_img\_9 <- find\_sd\_image(resized\_bounded\_box\_images\_training[which(vec == 9)])

)

touched\_bern\_avg\_imgs <- list(

touched\_bern\_avg\_img\_0 <- bern\_find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 0)]),

touched\_bern\_avg\_img\_1 <- bern\_find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 1)]),

touched\_bern\_avg\_img\_2 <- bern\_find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 2)]),

touched\_bern\_avg\_img\_3 <- bern\_find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 3)]),

touched\_bern\_avg\_img\_4 <- bern\_find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 4)]),

touched\_bern\_avg\_img\_5 <- bern\_find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 5)]),

touched\_bern\_avg\_img\_6 <- bern\_find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 6)]),

touched\_bern\_avg\_img\_7 <- bern\_find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 7)]),

touched\_bern\_avg\_img\_8 <- bern\_find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 8)]),

touched\_bern\_avg\_img\_9 <- bern\_find\_average\_image(resized\_bounded\_box\_images\_training[which(vec == 9)])

)

gaussian\_touched\_guesses <- lapply(resized\_bounded\_box\_images\_testing, determine\_digit\_touched)

gaussian\_touched\_guesses\_training <- lapply(resized\_bounded\_box\_images\_training, determine\_digit\_touched)

bern\_touched\_guesses <- lapply(resized\_bounded\_box\_images\_testing, bern\_determine\_digit\_touched)

bern\_touched\_guesses\_training <- lapply(resized\_bounded\_box\_images\_training, bern\_determine\_digit\_touched)

length(which(gaussian\_touched\_guesses == vec2))

length(which(bern\_touched\_guesses == vec2))

length(which(gaussian\_touched\_guesses\_training == vec))

length(which(bern\_touched\_guesses\_training == vec))

#Given a matrix, finds the posterior distributions for each digit,

#and returns the most likely value

determine\_digit\_touched <- function(mat) {

lst\_of\_posterior\_probs <- c()

for (k in 0:9) {

avg\_img <- touched\_avg\_imgs[[k+1]]

sd\_img <- touched\_sd\_imgs[[k+1]]

accum <- 0.1 + sum(log(0.0000000001 + dnorm(mat, avg\_img, sd\_img + 10)))

lst\_of\_posterior\_probs <- c(lst\_of\_posterior\_probs, accum)

}

#print(lst\_of\_posterior\_probs)

return(which(lst\_of\_posterior\_probs == max(lst\_of\_posterior\_probs))[1] - 1)

}

#Given a matrix, finds the posterior distributions for each digit,

#and returns the most likely value

bern\_determine\_digit\_touched <- function(mat) {

mat[which(mat != 0)] <- 1

lst\_of\_posterior\_probs <- c()

for (k in 0:9) {

avg\_img <- touched\_bern\_avg\_imgs[[k+1]]

accum <- 0.1 + sum(log(0.0000000001 + dbinom(mat, avg\_img, size = 1))) # for (i in 1:nrow(mat)) {

lst\_of\_posterior\_probs <- c(lst\_of\_posterior\_probs, accum)

}

#print(lst\_of\_posterior\_probs)

return(which(lst\_of\_posterior\_probs == max(lst\_of\_posterior\_probs))[1] - 1)

}

####################

#This part of the code deals with the random forest genreation

####################

h2o.init()

#Takes a matrix, and converts it into a row vector

convert\_to\_row <- function(mat) {

return(c(mat))

}

#Converts training data into a data frame, where each row is

#a lable for the digi, followed by 784 pixel values

make\_df\_for\_h2o <- function(col\_labels, img\_list) {

lst <- lapply(img\_list, convert\_to\_row)

names(lst) <- 1:length(lst)

df <- data.frame(as.character(col\_labels), t(as.data.frame(lst)))

return(df)

}

#Creates the data frames for the four sets.

h2o\_training\_untouched <- make\_df\_for\_h2o(vec, training\_images)

h2o\_training\_touched <- make\_df\_for\_h2o(vec, resized\_bounded\_box\_images\_training)

h2o\_testing\_untouched <- make\_df\_for\_h2o(vec2, testing\_images)

h2o\_testing\_touched <- make\_df\_for\_h2o(vec2, resized\_bounded\_box\_images\_testing)

#Turns the above data frames into h2o frames

h2o\_df\_training\_untouched <- as.h2o(x = h2o\_training\_untouched)

h2o\_df\_training\_touched <- as.h2o(x = h2o\_training\_touched)

h2o\_df\_testing\_untouched <- as.h2o(x = h2o\_testing\_untouched[,-1])

h2o\_df\_testing\_touched <- as.h2o(x = h2o\_testing\_touched[,-1])

#Creates the random forest models

model\_training\_untouched\_10\_4 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_untouched,

ntrees = 10, max\_depth = 4)

model\_training\_touched\_10\_4 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_touched,

ntrees = 10, max\_depth = 4)

model\_training\_untouched\_10\_16 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_untouched,

ntrees = 10, max\_depth = 16)

model\_training\_touched\_10\_16 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_touched,

ntrees = 10, max\_depth = 16)

model\_training\_untouched\_30\_4 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_untouched,

ntrees = 30, max\_depth = 4)

model\_training\_touched\_30\_4 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_touched,

ntrees = 30, max\_depth = 4)

model\_training\_untouched\_30\_16 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_untouched,

ntrees = 30, max\_depth = 16)

model\_training\_touched\_30\_16 <- h2o.randomForest(y = 1,

training\_frame = h2o\_df\_training\_touched,

ntrees = 30, max\_depth = 16)

#Predictions

testing\_untouched\_10\_4 <- predict(model\_training\_untouched\_10\_4, h2o\_df\_testing\_untouched)

length(which(as.vector(testing\_untouched\_10\_4[,1]) == vec2))

testing\_untouched\_10\_16 <- predict(model\_training\_untouched\_10\_16, h2o\_df\_testing\_untouched)

length(which(as.vector(testing\_untouched\_10\_16[,1]) == vec2))

testing\_untouched\_30\_4 <- predict(model\_training\_untouched\_30\_4, h2o\_df\_testing\_untouched)

length(which(as.vector(testing\_untouched\_30\_4[,1]) == vec2))

testing\_untouched\_30\_16 <- predict(model\_training\_untouched\_30\_16, h2o\_df\_testing\_untouched)

length(which(as.vector(testing\_untouched\_30\_16[,1]) == vec2))

testing\_touched\_10\_4 <- predict(model\_training\_touched\_10\_4, h2o\_df\_testing\_touched)

length(which(as.vector(testing\_touched\_10\_4[,1]) == vec2))

testing\_touched\_10\_16 <- predict(model\_training\_touched\_10\_16, h2o\_df\_testing\_touched)

length(which(as.vector(testing\_touched\_10\_16[,1]) == vec2))

testing\_touched\_30\_4 <- predict(model\_training\_touched\_30\_4, h2o\_df\_testing\_touched)

length(which(as.vector(testing\_touched\_30\_4[,1]) == vec2))

testing\_touched\_30\_16 <- predict(model\_training\_touched\_30\_16, h2o\_df\_testing\_touched)

length(which(as.vector(testing\_touched\_30\_16[,1]) == vec2))

training\_untouched\_10\_4 <- predict(model\_training\_untouched\_10\_4, h2o\_df\_training\_untouched)

length(which(as.vector(training\_untouched\_10\_4[,1]) == vec))

training\_untouched\_10\_16 <- predict(model\_training\_untouched\_10\_16, h2o\_df\_training\_untouched)

length(which(as.vector(training\_untouched\_10\_16[,1]) == vec))

training\_untouched\_30\_4 <- predict(model\_training\_untouched\_30\_4, h2o\_df\_training\_untouched)

length(which(as.vector(training\_untouched\_30\_4[,1]) == vec))

training\_untouched\_30\_16 <- predict(model\_training\_untouched\_30\_16, h2o\_df\_training\_untouched)

length(which(as.vector(training\_untouched\_30\_16[,1]) == vec))

training\_touched\_10\_4 <- predict(model\_training\_touched\_10\_4, h2o\_df\_training\_touched)

length(which(as.vector(training\_touched\_10\_4[,1]) == vec))

training\_touched\_10\_16 <- predict(model\_training\_touched\_10\_16, h2o\_df\_training\_touched)

length(which(as.vector(training\_touched\_10\_16[,1]) == vec))

training\_touched\_30\_4 <- predict(model\_training\_touched\_30\_4, h2o\_df\_training\_touched)

length(which(as.vector(training\_touched\_30\_4[,1]) == vec))

training\_touched\_30\_16 <- predict(model\_training\_touched\_30\_16, h2o\_df\_training\_touched)

length(which(as.vector(training\_touched\_30\_16[,1]) == vec))

**References**

Obtained a code snippet from

<https://stackoverflow.com/questions/35786744/resizing-image-in-r>

Received help with function, from the following SO pages

<https://stackoverflow.com/questions/54448330/r-repeatedly-apply-function-to-corresponding-elements-of-matrices-and-collect/54448573?noredirect=1#comment95706806_54448573>

<https://stackoverflow.com/questions/54429714/r-converting-a-list-of-lists-into-a-data-frame/54429927#54429927>