CS 498 HW1, UIUC

Apoorva Srinivasa (apoorva6@illinois.edu)

Julia Tcholakova (dt2@illinois.edu)

Jan 28th, 2019

Part 1 Accuracies

Setup	CV Accuracy - train	CV Accuracy - test
Unprocessed data	0.7652529	0.7590909
0-value elements ignored	0.7517129	0.7448052

Part 1 Code Snippets

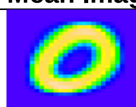
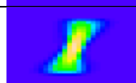
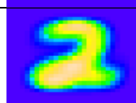
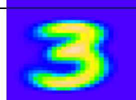
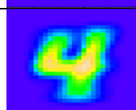
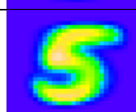
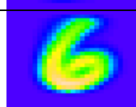
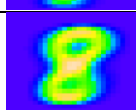
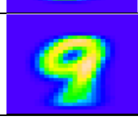
```
pimadata<-read.csv('pima-indians-diabetes.csv', header=TRUE)
trainaccuracy <- rep(0,10)
testaccuracy <- rep(0,10)
for (i in 1:10){
  # Split data into test and train
  index <- sample(seq_len(nrow(pimadata)), size = floor(0.8 * nrow(pimadata)))
  train <- pimadata[index, ]
  test <- pimadata[-index, ]
  #Extracting positive and negative labels for training dataset
  pos_labels <- subset(train,train[,c(9)] > 0)
  neg_labels <- subset(train,train[,c(9)] == 0)
  #Extracting the labels and data for test/train and storing them seperately
  y_test <- test[,c(9)]
  x_test <- test[,-c(9)]
  y_train <- train[,c(9)]
  x_train <- train[,-c(9)]
  x_train_pos <- pos_labels[,-c(9)]
  x_train_neg <- neg_labels[,-c(9)]
  #For positive label
  # P(positive_label)
  p_pos <- nrow(x_train_pos) / (nrow(x_train_pos)+nrow(x_train_neg))
  # Mean
  pos_mean <- sapply(x_train_pos, mean)
  # Std deviation
  pos_sd <- sapply(x_train_pos, sd)
  #For negative label
  # P(Negative_label)
  p_neg <- nrow(x_train_neg) / (nrow(x_train_pos)+nrow(x_train_neg))
  # Mean
  neg_mean <- sapply(x_train_neg, mean)
  # Std deviation
  neg_sd <- sapply(x_train_neg, sd)
  ## For training
  # Finding normal distribution cdf - log probabilities P(x|Y)
  pos_log_prob <- rowSums(log(mapply(dnorm,x_train,pos_mean,pos_sd))) + log(p_pos)
  neg_log_prob <- rowSums(log(mapply(dnorm,x_train,neg_mean,neg_sd))) + log(p_neg)
  #Checking prediction accuracy
  correct_pos_train <- ifelse(pos_log_prob>neg_log_prob,1,0)
  trainaccuracy[i] <- (sum(correct_pos_train == y_train))/length(y_train)
  ## For test data
  # Finding normal distribution cdf - log probabilities P(x|Y)
  pos_log_prob <- rowSums(log(mapply(dnorm,x_test,pos_mean,pos_sd))) + log(p_pos)
  neg_log_prob <- rowSums(log(mapply(dnorm,x_test,neg_mean,neg_sd))) + log(p_neg)
  #Checking prediction accuracy
  correct_pos_test <- ifelse(pos_log_prob>neg_log_prob,1,0)
  testaccuracy[i] <- (sum(correct_pos_test == y_test))/length(y_test)
}

train_accuracy_no_treatment <- mean(trainaccuracy)
test_accuracy_no_treatmnent <- mean(testaccuracy)
```

Part 2 MNIST Accuracies

Method	Training Set Accuracy	Test Set Accuracy
Gaussian + untouched	0.7826667	0.7903960
Gaussian + stretched	0.8209167	0.8299000
Bernoulli + untouched	0.6445000	0.6386000
Bernoulli + stretched	0.7010300	0.7115000
10 trees + 4 depth + untouched	0.8514333	0.8548000
10 trees + 4 depth + stretched	0.8122167	0.8204000
10 trees + 16 depth + untouched	0.9978000	0.9545000
10 trees + 16 depth + stretched	0.9983000	0.9580000
30 trees + 4 depth + untouched	0.8661000	0.8692000
30 trees + 4 depth + stretched	0.8262833	0.8350000
30 trees + 16 depth + untouched	0.9990667	0.9643000
30 trees + 16 depth + stretched	0.9992333	0.9671000

Part 2A Digit Images

Digit	Mean Image
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Part 2 Code: Please refer to page 6 for full code on this

```
#Gaussian
## run for untouched data
for (i in 0:9){
  train_sub <- subset(train_2, train_2$y == i)
 # y_train <- train_sub$y
  x_train <- train_sub[,-c(785)]
  test_sub <- subset(test_2, test_2$y == i)
 # y_test <- test_sub$y
  x_test <- test_sub[,-c(785)]
  #Epsilon is added to get rid of the sd=0 condition that generates "Infinity" values.
  eps <- 0.1
  prob_tr <- nrow(x_train) / nrow(train_2)
  mean_tr <- sapply(x_train, mean)
  sd_tr <- sapply(x_train, sd)+eps
   #use distro parm based on the training set
  log_prob_train[[i+1]] <- rowSums(log(mapply(dnorm,train_2x,mean_tr,sd_tr))) + log(prob_tr)
  log_prob_test[[i+1]] <-  rowSums(log(mapply(dnorm,test_2x,mean_tr,sd_tr))) + log(prob_tr)
}…(more code p.6)
acc_test_u = mean(test_2$y == test_2$y_hat)
acc_train_u = mean(train_2$y == train_2$y_hat)
#Bernoulli
 #calc the class j prob
  for(r in 1:nrow(train_bern_u)){
    log_pixel = rep(0,784)
    for(i in 1:784){
      n_ji = 0
      p_ji = 0
      n_ji = sum(x_train[,i])
      #Laplace smoothing
      p_ji = (n_ji + 1) / (n_j + 2)
      x_i = x_train[r,i]
      log_x = (x_i * log(p_ji) + (1-x_i) * log(1-p_ji))
      log_pixel[i] = log_x
    }
    prob_class_tr_vec[r] = log(prior_tr) + sum(log_pixel) }
#Random forest with h2o library (parts of code)
#random forest with untouched data
train_hex = as.h2o(train_2, destination_frame = "train.hex")
test_hex = as.h2o(test_2, destination_frame = "test.hex")
#model 5b
mod_5b = h2oModel(train_hex, 1:784, 785, 10, 4)
acc = calcAccuracy(mod_5b, train_hex, test_hex, box = FALSE)
accuracy_df[5, 3] = acc[1]
accuracy_df[5, 4] = acc[2] (more detailed code p.6)
#random forest with stretched data
train_hex_s = as.h2o(final_train, destination_frame = "train.hex.s")
test_hex_s = as.h2o(final_test, destination_frame = "test.hex.s")
#model 6b
mod_6b_s = h2oModel(train_hex_s, 1:400, 401, 10, 4)
acc = calcAccuracy(mod_6b_s, train_hex_s, test_hex_s, box = TRUE)
accuracy_df[6, 3] = acc[1] accuracy_df[6, 4] = acc[2]
```

Page 6 All Code

```
#HW1 Part 1 A
#Read csv data
pimadata<-read.csv('pima-indians-diabetes.csv', header=TRUE)
#Allocate empty vectors to store accuracies of each individual iterations
trainaccuracy <- rep(0,10)
testaccuracy <- rep(0,10)
# run training/testing 10 times
for (i in 1:10)
{
  # Split data into test and train
  index <- sample(seq_len(nrow(pimadata)), size = floor(0.8 * nrow(pimadata)))
  train <- pimadata[index, ]
  test <- pimadata[-index, ]
  #Extracting positive and negative labels for training dataset
  pos_labels <- subset(train,train[,c(9)] > 0)
  neg_labels <- subset(train,train[,c(9)] == 0)
  #Extracting the labels and data for test/train and storing them seperately
  y_test <- test[,c(9)]
  x_test <- test[,-c(9)]
  y_train <- train[,c(9)]
  x_train <- train[,-c(9)]
  x_train_pos <- pos_labels[,-c(9)]
  x_train_neg <- neg_labels[,-c(9)]
  #For positive label
  # P(positive_label)
  p_pos <- nrow(x_train_pos) / (nrow(x_train_pos)+nrow(x_train_neg))
  # Mean
  pos_mean <- sapply(x_train_pos, mean)
  # Std deviation
  pos_sd <- sapply(x_train_pos, sd)
  #For negative label
  # P(Negative_label)
  p_neg <- nrow(x_train_neg) / (nrow(x_train_pos)+nrow(x_train_neg))
  # Mean
  neg_mean <- sapply(x_train_neg, mean)
  # Std deviation
  neg_sd <- sapply(x_train_neg, sd)
  ## For training
  # Finding normal distribution cdf - log probabilities P(x|Y)
  pos_log_prob <- rowSums(log(mapply(dnorm,x_train,pos_mean,pos_sd))) + log(p_pos)
  neg_log_prob <- rowSums(log(mapply(dnorm,x_train,neg_mean,neg_sd))) + log(p_neg)
  #Checking prediction accuracy
  correct_pos_train <- ifelse(pos_log_prob>neg_log_prob,1,0)
  trainaccuracy[i] <- (sum(correct_pos_train == y_train))/length(y_train)
  ## For test data
  # Finding normal distribution cdf - log probabilities P(x|Y)
  pos_log_prob <- rowSums(log(mapply(dnorm,x_test,pos_mean,pos_sd))) + log(p_pos)
  neg_log_prob <- rowSums(log(mapply(dnorm,x_test,neg_mean,neg_sd))) + log(p_neg)
  #Checking prediction accuracy
  correct_pos_test <- ifelse(pos_log_prob>neg_log_prob,1,0)
```

```
  testaccuracy[i] <- (sum(correct_pos_test == y_test))/length(y_test)
}

train_accuracy_no_treatment <- mean(trainaccuracy)
test_accuracy_no_treatmnent <- mean(testaccuracy)


#Part 1B
#Treating the value 0 in columns 3,4,6 and 8 as NAs
for( i in c(3,4,6,8))
{
  treatzero <- pimadata[,i] == 0
  pimadata[treatzero,i] <- NA
}

#Allocate empty vectors to store accuracies of each individual iterations
trainaccuracy <- rep(0,10)
testaccuracy <- rep(0,10)

# run training/testing 10 times
for (i in 1:10)
{
  # Split data into test and train
  index <- sample(seq_len(nrow(pimadata)), size = floor(0.8 * nrow(pimadata)))
  train <- pimadata[index, ]
  test <- pimadata[-index, ]

  #Extracting positive and negative labels for training dataset
  pos_labels <- subset(train,train[, c(9)] > 0)
  neg_labels <- subset(train,train[, c(9)] == 0)

  #Extracting the labels and data for test/train and storing them separately
  y_test <- test[,c(9)]
  x_test <- test[,-c(9)]
  y_train <- train[,c(9)]
  x_train <- train[,-c(9)]


  x_train_pos <- pos_labels[,-c(9)]
  x_train_neg <- neg_labels[,-c(9)]

  #For positive label
  # P(positive_label)
  p_pos <- nrow(x_train_pos) / (nrow(x_train_pos)+nrow(x_train_neg))
  # Mean
  pos_mean <- sapply(x_train_pos, mean, na.rm=TRUE)
  # Std deviation
  pos_sd <- sapply(x_train_pos, sd,na.rm=TRUE)

  #For negative label
  # P(Negative_label)
  p_neg <- nrow(x_train_neg) / (nrow(x_train_pos)+nrow(x_train_neg))
```

```
  # Mean
  neg_mean <- sapply(x_train_neg, mean,na.rm=TRUE)
  # Std deviation
  neg_sd <- sapply(x_train_neg, sd,na.rm=TRUE)

  function(fun,mean,sd,p)
  {
    pos_log_prob <- rowSums(log(mapply(dnorm,x_train,pos_mean,pos_sd))) + log(p_pos)
  }
  ## For training
  # Finding normal distribution cdf - log probabilities P(x|Y)
  pos_log_prob <- rowSums(log(mapply(dnorm,x_train,pos_mean,pos_sd)), na.rm=TRUE) + log(p_pos)
  neg_log_prob <- rowSums(log(mapply(dnorm,x_train,neg_mean,neg_sd)), na.rm=TRUE) + log(p_neg)

  #Checking prediction accuracy
  correct_pos_train <- ifelse(pos_log_prob>neg_log_prob,1,0)
  trainaccuracy[i] <- (sum(correct_pos_train == y_train))/length(y_train)

  ## For test data
  # Finding normal distribution cdf - log probabilities P(x|Y)
  pos_log_prob <- rowSums(log(mapply(dnorm,x_test,pos_mean,pos_sd)), na.rm=TRUE) + log(p_pos)
  neg_log_prob <- rowSums(log(mapply(dnorm,x_test,neg_mean,neg_sd)), na.rm=TRUE) + log(p_neg)

  #Checking prediction accuracy
  correct_pos_test <- ifelse(pos_log_prob>neg_log_prob,1,0)
  testaccuracy[i] <- (sum(correct_pos_test == y_test))/length(y_test)
}


train_accuracy_with_treatment <- mean(trainaccuracy)
test_accuracy_with_treatmnent <- mean(testaccuracy)

#Homework 1 Part 2
#Part 2A
#final sets for stretched data
library(imager)
library(caret)
library(quanteda)


load_image_file <- function(filename) {
  ret = list()
  f = file(filename,'rb')
  readBin(f,'integer',n=1,size=4,endian='big')
  ret$n = readBin(f,'integer',n=1,size=4,endian='big')
  nrow = readBin(f,'integer',n=1,size=4,endian='big')
  ncol = readBin(f,'integer',n=1,size=4,endian='big')
  x = readBin(f,'integer',n=ret$n*nrow*ncol,size=1,signed=F)
  ret$x = matrix(x, ncol=nrow*ncol, byrow=T)
  close(f)
  ret
}
```

```
load_label_file <- function(filename) {
  f = file(filename,'rb')
  readBin(f,'integer',n=1,size=4,endian='big')
  n = readBin(f,'integer',n=1,size=4,endian='big')
  y = readBin(f,'integer',n=n,size=1,signed=F)
  close(f)
  y
}

train <- load_image_file('train-images.idx3-ubyte')
test <- load_image_file('t10k-images.idx3-ubyte')
train$y = load_label_file("train-labels.idx1-ubyte")
test$y  = load_label_file("t10k-labels.idx1-ubyte")

train_thresh <- apply(train$x, 1, threshold)
test_thresh <-  apply(test$x, 1, threshold)

#Stretch the image to 20*20
stretch <- function(image){
  return(as.matrix(resize(as.cimg(image), size_x = 20, size_y = 20)))
}

train_stretch <- lapply(train_thresh, stretch)
test_stretch <- lapply(test_thresh, stretch)


#Concatenate
train_concat <- lapply(train_stretch, function(x){c(x)})
test_concat <- lapply(test_stretch, function(x){c(x)})

# Some random trial and error to get a dataframe in the end
train_rbind <- do.call(rbind,train_concat)
test_rbind <- do.call(rbind,test_concat)

train_df <- as.data.frame(train_rbind)
test_df <- as.data.frame(test_rbind)

#Add y labels
final_train <- cbind(train_df, Label = as.factor(train$y))
final_test <- cbind(test_df, Label = as.factor(test$y))

show_digit_20(train_df[1,])

#Gaussian
#Gaussian untouched and stretched data
#Extarcting features and labels for test and train untouched
train_2x <- train_2[,-c(785)]
train_2y <- train_2[,c(785)]
test_2x <- test_2[,-c(785)]
test_2y <- test_2[,c(785)]

log_prob_train <- list()
```

```
log_prob_test  <- list()
log_prob_train_s <- list()
log_prob_test_s  <- list()

# run for untouched data
for (i in 0:9)
{
  train_sub <- subset(train_2, train_2$y == i)
 # y_train <- train_sub$y
  x_train <- train_sub[,-c(785)]
  test_sub <- subset(test_2, test_2$y == i)
 # y_test <- test_sub$y
  x_test <- test_sub[,-c(785)]
  #Epsilon is added to get rid of the sd=0 condition that generates "Infinity" values.
  eps <- 0.1
  prob_tr <- nrow(x_train) / nrow(train_2)
  mean_tr <- sapply(x_train, mean)
  sd_tr <- sapply(x_train, sd)+eps
  log_prob_train[[i+1]] <- rowSums(log(mapply(dnorm,train_2x,mean_tr,sd_tr))) + log(prob_tr)
  #changed prior probability for test to use the prior from training set, as more valid
  log_prob_test[[i+1]] <-  rowSums(log(mapply(dnorm,test_2x,mean_tr,sd_tr))) + log(prob_tr)
{
train_2$y_hat = rep(0, nrow(train_2))
test_2$y_hat = rep(0, nrow(test_2))

#recording the y_hat for train and test
for (i in 1:nrow(train_2)){
  image = sapply(log_prob_train, "[[", i)
  y_hat = which.max(image)-1
  train_2[i, 786] = y_hat

}
for (i in 1:nrow(test_2)){
  image = sapply(log_prob_test, "[[", i)
  y_hat = which.max(image)-1
  test_2[i, 786] = y_hat

}
acc_test_u = mean(test_2$y == test_2$y_hat)
acc_train_u = mean(train_2$y == train_2$y_hat)

#running for stretched data
train_2x_s <- final_train[,-c(401)]
train_2y_s <- final_train[,c(401)]
test_2x_s <- final_test[,-c(401)]
test_2y_s <- final_test[,c(401)]

for (i in 0:9)
{
  train_sub <- subset(final_train, final_train$Label == i)
  # y_train <- train_sub$y
  x_train <- train_sub[,-c(401)]
```

```
  test_sub <- subset(final_test, final_test$Label == i)
  # y_test <- test_sub$y
  x_test <- test_sub[,-c(401)]
  #Epsilon is added to get rid of the sd=0 condition that generates "Infinity" values.
  #eps <- 1e-6
  eps = 0.1
  prob_tr <- nrow(x_train) / nrow(final_train)
  mean_tr <- sapply(x_train, mean)
  sd_tr <- sapply(x_train, sd)+eps
  log_prob_train_s[[i+1]] <- rowSums(log(mapply(dnorm,train_2x_s,mean_tr,sd_tr))) + log(prob_tr)
 #prior probability for test to use the prior from training set
  log_prob_test_s[[i+1]] <-  rowSums(log(mapply(dnorm,test_2x_s,mean_tr,sd_tr))) + log(prob_tr)

}
#calc accuracies
final_train$y_hat = rep(0, nrow(final_train))
final_test$y_hat = rep(0, nrow(final_test))

#recording the y_hat for train and test
for (i in 1:nrow(final_train)){
  image = sapply(log_prob_train_s, "[[", i)
  y_hat = which.max(image)-1
  final_train[i, 402] = y_hat

}
for (i in 1:nrow(final_test)){
  image = sapply(log_prob_test_s, "[[", i)
  y_hat = which.max(image)-1
  final_test[i, 402] = y_hat

}
acc_test_s = mean(final_test$Label == final_test$y_hat)
acc_train_s = mean(final_train$Label == final_train$y_hat)

#Bernoulli
#Part 2 A Bernoulli
#install.packages("statip")
library('statip')
library("matrixStats")
#Extarcting features and labels for test and train untouched
#train_2 and test_2 are available already
train_2x <- train_2[,-c(785)]
train_2y <- train_2[,c(785)]
test_2x <- test_2[,-c(785)]
test_2y <- test_2[,c(785)]

#threshold for Bernoulli
do_thresh = function(x){
  if(x >= 127){
     x = 1
  }
  else if (x < 127){
```

```
    x = 0
  }
  else{
   x=0
  }

  return(x)
}

for(i in 1:784){
  train_2x[,i] = mapply(do_thresh, train_2x[,i])
  test_2x[,i] = mapply(do_thresh, test_2x[,i])
}

#untouched data
train_bern_u = cbind(train_2x,train_2y)
test_bern_u = cbind(test_2x,test_2y)
log_prob_train_bern <- list()
log_prob_test_bern  <- list()
#referencing lectures from Berkeley - http://www-inst.eecs.berkeley.edu/~cs70/sp15/notes/n21_slides.pdf
for (j in 0:9)
{
  prob_class_tr_vec = rep(0,nrow(train_bern_u))
  prob_class_test_vec = rep(0,nrow(test_bern_u))
  train_sub <- subset(train_bern_u, train_bern_u$train_2y == j)
  x_train <- train_sub[,-c(785)]
  test_sub <- subset(test_bern_u, test_bern_u$test_2y == j)
  x_test <- test_sub[,-c(785)]
  #prior prob for a class
  prior_tr <- nrow(x_train) / nrow(train_bern_u)
  #number of j class examples
  n_j = nrow(x_train)
  #calc the class j prob
  for(r in 1:nrow(train_bern_u)){
    log_pixel = rep(0,784)
    for(i in 1:784){
      n_ji = 0
      p_ji = 0
      n_ji = sum(x_train[,i])
      #Laplace smoothing
      p_ji = (n_ji + 1) / (n_j + 2)
      x_i = x_train[r,i]
      log_x = (x_i * log(p_ji) + (1-x_i) * log(1-p_ji))
      log_pixel[i] = log_x
    }
    prob_class_tr_vec[r] = log(prior_tr) + sum(log_pixel)

  }
  #t
  for(r in 1:nrow(test_bern_u)){
    log_pixel_test = rep(0,784)
    for(i in 1:784){
```

```
      n_ji = 0
      p_ji = 0
      n_ji = sum(x_test[,i])
      #Laplace smoothing
      p_ji = (n_ji + 1) / (n_j + 2)
      x_i_test = x_test[r,i]
      log_x_test = (x_i_test * log(p_ji) + (1-x_i_test) * log(1-p_ji))
      log_pixel_test[i] = log_x_test
    }
    prob_class_test_vec[r] = log(prior_tr) + sum(log_pixel_test)
  }
  log_prob_train_bern[[j+1]] = prob_class_tr_vec
  log_prob_test_bern[[j+1]] = prob_class_test_vec

}

#calc accuracies
train_bern_u$y_hat = rep(0, nrow(train_bern_u))
test_bern_u$y_hat = rep(0, nrow(test_bern_u))

#recording the y_hat for train and test
for (i in 1:nrow(train_bern_u)){
  image = sapply(log_prob_train_bern, "[[", i)
  y_hat = which.max(image)-1
  train_bern_u[i, 786] = y_hat
}

for (i in 1:nrow(test_bern_u)){
  image = sapply(log_prob_test_bern, "[[", i)
  y_hat = which.max(image)-1
  test_bern_u[i, 786] = y_hat

}
acc_train_u_b = mean(train_bern_u$train_2y == train_bern_u$y_hat)
acc_test_u_b = mean(test_bern_u$test_2y == test_bern_u$y_hat)


#stretched data, using the final_train and final_test data, already prepared
log_prob_train_bern <- list()
log_prob_test_bern  <- list()
#referencing lectures from Berkeley - http://www-inst.eecs.berkeley.edu/~cs70/sp15/notes/n21_slides.pdf
for (j in 0:9)
{
  prob_class_tr_vec = rep(0,nrow(final_train))
  prob_class_test_vec = rep(0,nrow(final_test))
  train_sub <- subset(final_train, final_train$Label == j)
  x_train <- train_sub[,-c(401)]
  test_sub <- subset(final_test, final_test$Label == j)
  x_test <- test_sub[,-c(401)]
  #prior prob for a class
  prior_tr <- nrow(x_train) / nrow(final_train)
  #number of j class examples
```

```
  n_j = nrow(x_train)
  #calc the class j prob
  for(r in 1:nrow(final_train)){
    log_pixel = rep(0,400)
    for(i in 1:400){
      n_ji = 0
      p_ji = 0
      n_ji = sum(x_train[,i])
      #Laplace smoothing
      p_ji = (n_ji + 1) / (n_j + 2)
      x_i = x_train[r,i]
      log_x = (x_i * log(p_ji) + (1-x_i) * log(1-p_ji))
      log_pixel[i] = log_x
    }
    prob_class_tr_vec[r] = log(prior_tr) + sum(log_pixel)

  }
  #t
  for(r in 1:nrow(final_test)){
    log_pixel_test = rep(0,400)
    for(i in 1:400){
      n_ji = 0
      p_ji = 0
      n_ji = sum(x_test[,i])
      #Laplace smoothing
      p_ji = (n_ji + 1) / (n_j + 2)
      x_i_test = x_test[r,i]
      log_x_test = (x_i_test * log(p_ji) + (1-x_i_test) * log(1-p_ji))
      log_pixel_test[i] = log_x_test
    }
    prob_class_test_vec[r] = log(prior_tr) + sum(log_pixel_test)
  }
  log_prob_train_bern[[j+1]] = prob_class_tr_vec
  log_prob_test_bern[[j+1]] = prob_class_test_vec

}

#calc accuracies
final_train$y_hat = rep(0, nrow(final_train))
final_test$y_hat = rep(0, nrow(final_test))

#recording the y_hat for train and test
for (i in 1:nrow(final_train)){
  image = sapply(log_prob_train_bern, "[[", i)
  y_hat = which.max(image)-1
  final_train[i, 402] = y_hat
}

for (i in 1:nrow(final_train)){
  image = sapply(log_prob_test_bern, "[[", i)
  y_hat = which.max(image)-1
  final_test[i, 402] = y_hat
```

```
}
acc_train_u_s = mean(final_train$Label == final_train$y_hat)
acc_test_u_s = mean(final_test$Label == final_test$y_hat)




#part 2A image means
#function to show the image
show_digit = function(arr784, col = topo.colors(100), ...) {
  image(matrix(as.matrix(arr784[-785]), nrow = 28)[, 28:1], col = col, ...)
}
#to store the mean pixels"
store_means = list()
#for (i in 0:9)
for(i in 0:9)
{
  #subsets by labels
  label = subset(train_2, train_2$y == i)
  #get only the pixels
  label_px = label[,-c(785)]

  image_mean = apply(label_px,2,mean)
  store_means[[i+1]] = image_mean

}

#Part 2B
### random forest classification part 2B
library(h2o)
#init the clusters
h2o.init(ip = 'localhost', port = 54321, max_mem_size = '2650m')
#funtion to create the model with different parameters
h2oModel = function(tframe,a,b,nTree,depth){
  mod = h2o.randomForest(training_frame = tframe,
                         x = a,
                         y = b,
                         ntrees = nTree,
                         max_depth = depth)
  return(mod)
}
#function to calc the accuracies
calcAccuracy = function(mod, train_data, test_data, box = FALSE){

  train_pred = h2o.predict(object = mod, newdata = train_data)
  test_pred = h2o.predict(object = mod, newdata = test_data)
  if(box == TRUE){
    acc_train = mean(train_pred[, 1] == train_data[,401])
    acc_test = mean(test_pred[, 1] == test_data[,401])
  }
  else{
    acc_train = mean(train_pred[, 1] == train_data[,785])
```

```
    acc_test = mean(test_pred[, 1] == test_data[,785])
  }

  acc = c(acc_train, acc_test)
  return(acc)
}
#dataframe to store the accuracy results
num = seq(1:12)
method =
  c("Gaussian + untouched", "Gaussian + stretched","Bernoulli + untouched", "Bernoulli + stretched",
    "10 trees + 4 depth + untouched", "10 trees + 4 depth + stretched", "10 trees + 16 depth + untouched",
    "10 trees + 16 depth + stretched", "30 trees + 4 depth + untouched", "30 trees + 4 depth + stretched",
    "30 trees + 16 depth + untouched", "30 trees + 16 depth + stretched")

training_accuracy = rep(0, 12)
test_accuracy = rep(0, 12)
accuracy_df = data.frame(num, method, training_accuracy, test_accuracy)

#random forest with untouched data
train_hex = as.h2o(train_2, destination_frame = "train.hex")
test_hex = as.h2o(test_2, destination_frame = "test.hex")
#create the models with untouched images
#model 5b
mod_5b = h2oModel(train_hex, 1:784, 785, 10, 4)
acc = calcAccuracy(mod_5b, train_hex, test_hex, box = FALSE)
accuracy_df[5, 3] = acc[1]
accuracy_df[5, 4] = acc[2]

#model 7b
mod_7b = h2oModel(train_hex, 1:784, 785, 10, 16)
acc = calcAccuracy(mod_7b, train_hex, test_hex, box = FALSE)
accuracy_df[7, 3] = acc[1]
accuracy_df[7, 4] = acc[2]

#model 9b
mod_9b = h2oModel(train_hex, 1:784, 785, 30, 4)
acc = calcAccuracy(mod_9b, train_hex, test_hex, box = FALSE)
accuracy_df[9, 3] = acc[1]
accuracy_df[9, 4] = acc[2]

#model 11b
mod_11b = h2oModel(train_hex, 1:784, 785, 30, 16)
acc = calcAccuracy(mod_11b, train_hex, test_hex, box = FALSE)
accuracy_df[11, 3] = acc[1]
accuracy_df[11, 4] = acc[2]

#random forest with stretched data
train_hex_s = as.h2o(final_train, destination_frame = "train.hex.s")
test_hex_s = as.h2o(final_test, destination_frame = "test.hex.s")

#create the models with stretched images
#model 6b
```

```
mod_6b_s = h2oModel(train_hex_s, 1:400, 401, 10, 4)
acc = calcAccuracy(mod_6b_s, train_hex_s, test_hex_s, box = TRUE)
accuracy_df[6, 3] = acc[1]
accuracy_df[6, 4] = acc[2]

#model 8b
mod_8b_s = h2oModel(train_hex_s, 1:400, 401, 10, 16)
acc = calcAccuracy(mod_8b_s, train_hex_s, test_hex_s, box = TRUE)
accuracy_df[8, 3] = acc[1]
accuracy_df[8, 4] = acc[2]

#model 10b
mod_10b_s = h2oModel(train_hex_s, 1:400, 401, 30, 4)
acc = calcAccuracy(mod_10b_s, train_hex_s, test_hex_s, box = TRUE)
accuracy_df[10, 3] = acc[1]
accuracy_df[10, 4] = acc[2]

#model 12b
mod_12b_s = h2oModel(train_hex_s, 1:400, 401, 30, 16)
acc = calcAccuracy(mod_12b_s, train_hex_s, test_hex_s, box = TRUE)
accuracy_df[12, 3] = acc[1]
accuracy_df[12, 4] = acc[2]

knitr::kable(accuracy_df)

h2o.removeAll()
```