Part1A: Average accuracy is 67.43%.

Part1B: Average accuracy is 74.19%.

Part1D: Average accuracy is 70.86%.

```r
library(caret)
data <- read.table("pima-indians-diabetes.csv",sep = ',')
colnames(data) <- c("feat_a","feat_b","feat_c","feat_d","feat_e","feat_f","feat_g","feat_h","res")
for (i in 1:nrow(data)){
  if (data[i,]$feat_c == 0){
    data[i,]$feat_c <- NA
  }
  if(data[i,]$feat_d == 0){
    data[i,]$feat_d <- NA
  }
  if(data[i,]$feat_f == 0){
    data[i,]$feat_f <- NA
  }
  if(data[i,]$feat_h == 0){
    data[i,]$feat_h <- NA
  }
}
start <- 1
end <- round(nrow(data)/10)
total_acc <- 0
for (i in 1:10){
  eva_data <- data[start:end,]
  ind <- createDataPartition(eva_data$res, times=1, p=0.8, list=F)
  train_data <- eva_data[ind,]
  test_data <- eva_data[-ind,]
  p_y <- sum(train_data$res=="TRUE")/nrow(train_data)
  p_n <- sum(train_data$res=="FALSE")/nrow(train_data)
  ind_y <- train_data$res == "TRUE"
  ind_n <- train_data$res == "FALSE"
  feat_a_y_mean <- mean(train_data[ind_y,]$feat_a)
  feat_a_n_mean <- mean(train_data[ind_n,]$feat_a)
  feat_a_y_sd <- sd(train_data[ind_y,]$feat_a)
  feat_a_n_sd <- sd(train_data[ind_n,]$feat_a)
  feat_b_y_mean <- mean(train_data[ind_y,]$feat_b)
  feat_b_n_mean <- mean(train_data[ind_n,]$feat_b)
  feat_b_y_sd <- sd(train_data[ind_y,]$feat_b)
  feat_b_n_sd <- sd(train_data[ind_n,]$feat_b)
  feat_c_y_mean <- mean(train_data[ind_y,]$feat_c, na.rm=TRUE)
  feat_c_n_mean <- mean(train_data[ind_n,]$feat_c, na.rm=TRUE)
  feat_c_y_sd <- sd(train_data[ind_y,]$feat_c, na.rm=TRUE)
  feat_c_n_sd <- sd(train_data[ind_n,]$feat_c, na.rm=TRUE)
  feat_d_y_mean <- mean(train_data[ind_y,]$feat_d, na.rm=TRUE)
```

```r
library(caret)
library(klaR)
data <- read.table("pima-indians-diabetes.csv",sep = ',')
colnames(data) <- c("feat_a","feat_b",
                    "feat_c","feat_d",
                    "feat_e","feat_f",
                    "feat_g","feat_h",
                    "res")

start <- 1
end <- round(nrow(data)/10)
total_acc <- 0
for (i in 1:10){
  eva_data = data[start:end,]
  ind <- createDataPartition(eva_data$res, times=1, p=0.8, list=F)
  train_data = eva_data[ind,]
  test_data = eva_data[-ind,]
  model <- svmlight(res~., data=train_data)
  pred <- predict(model, newdata=test_data)
  svm_table <- table(actual=test_data$res, predict=pred$class)
  ratio <- sum(diag(svm_table))/sum(svm_table)

  print(svm_table)
  print(ratio)

  total_acc <- total_acc + ratio

  start <- start + round(nrow(data)/10)
  end <- end + round(nrow(data)/10)
  if(end > nrow(data))
    end <- nrow(data)
}

print(total_acc/10)
```

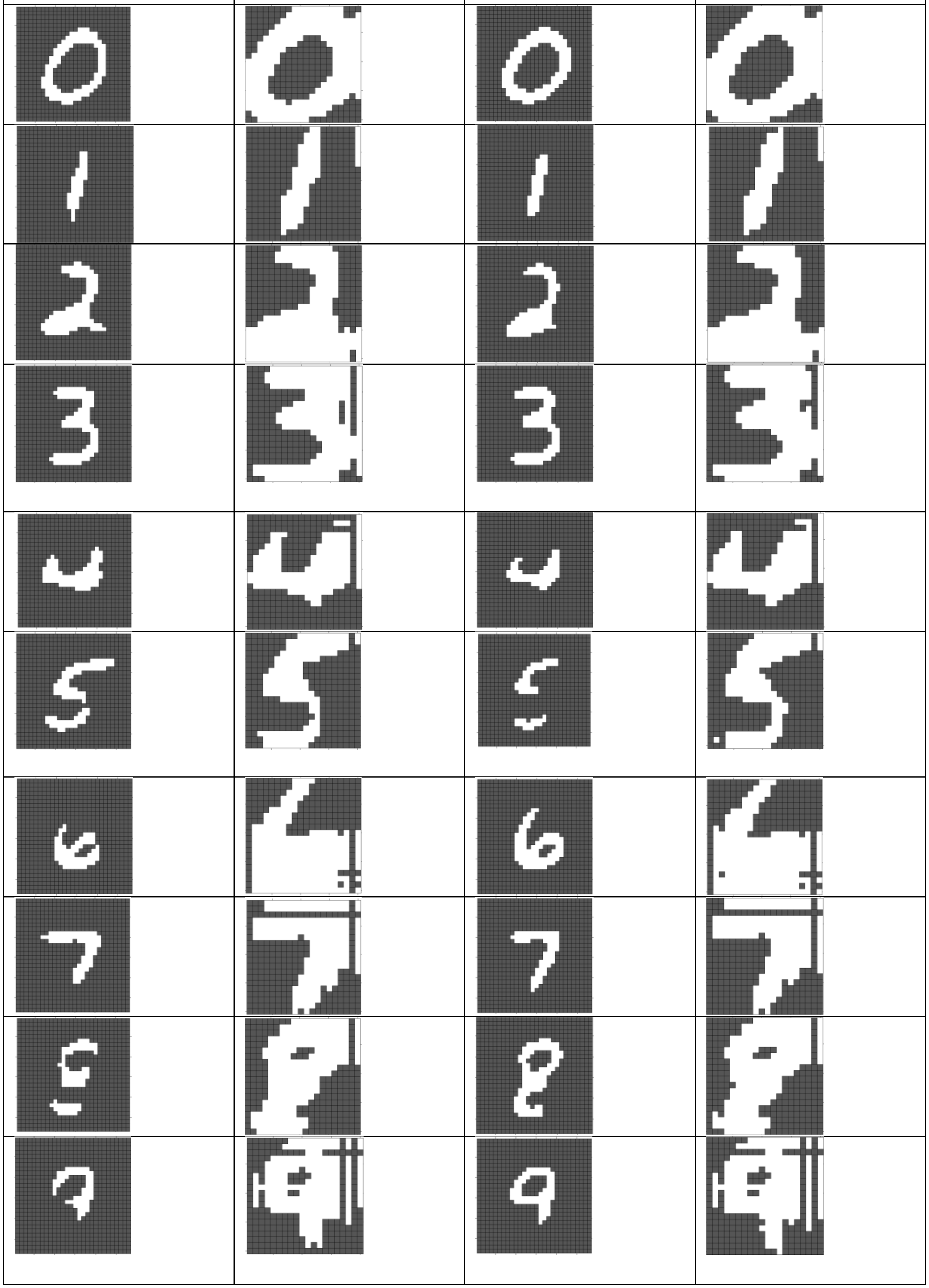
```r
  pred <- c()
  for (i in 1:nrow(test_data)){
    y <- dnorm(test_data[i,]$feat_a,feat_a_y_mean,feat_a_y_sd,log=T)+dnorm(test_data[i,]$feat_b,feat_b_y_mean,feat_b_y_sd,log=T)+
      dnorm(test_data[i,]$feat_g,feat_g_y_mean,feat_g_y_sd,log=T)+dnorm(test_data[i,]$feat_e,feat_e_y_mean,feat_e_y_sd,log=T)+
      log(p_y)
    if (!is.na(dnorm(test_data[i,]$feat_c,feat_c_y_mean,feat_c_y_sd,log=T)))
      y <- y + dnorm(test_data[i,]$feat_c,feat_c_y_mean,feat_c_y_sd,log=T)
    if (!is.na(dnorm(test_data[i,]$feat_d,feat_d_y_mean,feat_d_y_sd,log=T)))
      y <- y + dnorm(test_data[i,]$feat_d,feat_d_y_mean,feat_d_y_sd,log=T)
    if (!is.na(dnorm(test_data[i,]$feat_f,feat_f_y_mean,feat_f_y_sd,log=T)))
      y <- y + dnorm(test_data[i,]$feat_f,feat_f_y_mean,feat_f_y_sd,log=T)
    if (!is.na(dnorm(test_data[i,]$feat_h,feat_h_y_mean,feat_h_y_sd,log=T)))
      y <- y + dnorm(test_data[i,]$feat_h,feat_h_y_mean,feat_h_y_sd,log=T)
    n <- dnorm(test_data[i,]$feat_a,feat_a_n_mean,feat_a_n_sd,log=T)+dnorm(test_data[i,]$feat_b,feat_b_n_mean,feat_b_n_sd,log=T)+
      dnorm(test_data[i,]$feat_e,feat_e_n_mean,feat_e_n_sd,log=T)+dnorm(test_data[i,]$feat_g,feat_g_n_mean,feat_g_n_sd,log=T)+
      log(p_n)
    if (!is.na(dnorm(test_data[i,]$feat_c,feat_c_n_mean,feat_c_n_sd,log=T)))
      n <- n + dnorm(test_data[i,]$feat_c,feat_c_n_mean,feat_c_n_sd,log=T)
    if (!is.na(dnorm(test_data[i,]$feat_d,feat_d_n_mean,feat_d_n_sd,log=T)))
      n <- n + dnorm(test_data[i,]$feat_d,feat_d_n_mean,feat_d_n_sd,log=T)
    if (!is.na(dnorm(test_data[i,]$feat_f,feat_f_n_mean,feat_f_n_sd,log=T)))
      n <- n + dnorm(test_data[i,]$feat_f,feat_f_n_mean,feat_f_n_sd,log=T)
    if (!is.na(dnorm(test_data[i,]$feat_h,feat_h_n_mean,feat_h_n_sd,log=T)))
      n <- n + dnorm(test_data[i,]$feat_h,feat_h_n_mean,feat_h_n_sd,log=T)
    if(y>n){
      pred[i] <- c("TRUE")
    } else {
      pred[i] <- ("FALSE")
    }
  }
  nb_table <- table(actual=test_data$res, predict=pred)
  ratio <- sum(diag(nb_table))/sum(nb_table)
  print(nb_table)
  print(ratio)
  total_acc <- total_acc + ratio
  start <- start + round(nrow(data)/10)
  end <- end + round(nrow(data)/10)
  if(end > nrow(data))
    end <- nrow(data)
}
print(total_acc/10)
```

Method	Accuracy
Gaussian + untouched	61.83%
Gaussian + stretched	72.26%
Bernoulli + untouched	83.53%
Bernoulli + stretched	82.27%
10 trees + 4 depth + untouched	62.43%
10 trees + 4 depth + stretched	68.14%
10 trees + 16 depth + untouched	95.19%
10 trees + 16 depth + stretched	90.20%
30 trees + 4 depth + untouched	64.93%
30 trees + 4 depth + stretched	72.26%
30 trees + 16 depth + untouched	96.90%
30 trees + 16 depth + stretched	93.57%

Best Model: random forest model with most tree number and largest depth

Explanation: with large tree number and depth, classify can be more detailed.

Gaussian + untouched	Gaussian + stretched	Bernoulli + untouched	Bernoulli + stretched



```r
library(e1071)
library(sparklyr)
library(dplyr)
Sys.setenv(
  SPARK_HOME="C:\\spark\\spark-2.3.1-bin-hadoop2.7"
)
sc <- spark_connect(master = "local")

# Take in data
image_train_data <- read.table("train.csv", sep = ',')
# image_test_data <- read.table("val.csv", sep=",")

# Rename column of data frame
colnames(image_train_data)[1] <- "Number"
# colnames(image_test_data)[1] <- "Number"
colnames(image_train_data)[2:785] <- sprintf("pixel%d", 1:784)
# colnames(image_test_data)[2:785] <- sprintf("pixel%d", 1:784)
image_train_data$Number <- as.factor(image_train_data$Number)
# image_test_data$Number <- as.factor(image_test_data$Number)

# Set a threshold (127)
image_train_data[image_train_data <= 127] <- 0
image_train_data[image_train_data >= 128] <- 1
# image_test_data[image_test_data <= 127] <- 0
image_test_data[image_test_data >= 128] <- 1

# Resizing function that resize and strech the image
resizingFunc <- function(inputdata){…}


# Gaussian & untouched
image_res_data <- read.table("test.csv", sep=",")
colnames(image_res_data) <- sprintf("pixel%d", 1:784)
image_res_data[image_res_data <= 127] <- 0
image_res_data[image_res_data >= 128] <- 1
image_model <- naiveBayes(Number~., data=image_train_data)
image_pred <- predict(image_model, newdata=image_res_data)
# image_table <- table(actual=image_test_data$Number, predict=image_pred)
```

```r

# Gaussian & streched bounding (28*28 -> 20*20)

# resizing bounding first
image_train_data <- read.table("train_stretched_noheader.csv", sep=",")
colnames(image_train_data)[1] <- "Number"
colnames(image_train_data)[2:401] <- sprintf("pixel%d", 1:400)
image_train_data$Number <- as.factor(image_train_data$Number)
image_res_data <- read.table("test_stretched2.csv", sep=",")
colnames(image_res_data) <- sprintf("pixel%d", 1:400)

image_model <- naiveBayes(Number~., data=image_train_data)
image_pred <- predict(image_model, newdata=image_res_data)


# Bernoulli & untouched
#
image_res_data <- read.table("test.csv", sep=",")
colnames(image_res_data) <- sprintf("pixel%d", 1:784)
image_res_data[image_res_data <= 127] <- 0
image_res_data[image_res_data >= 128] <- 1

image_train_tbl <- copy_to(sc, image_train_data[1:30000,])
image_res_tbl <- copy_to(sc, image_res_data)
image_ber_model <- image_train_tbl %>%
  ml_naive_bayes(Number~., model_type="bernoulli")
image_ber_pred <- predict(image_ber_model, newdata=image_res_tbl)
# image_ber_table <- table(actual=image_test_data[,1], predict=image_ber_pred)
# ratio <- sum(diag(image_ber_table))/sum(image_ber_table)
#
# print(image_ber_table)
# print(ratio)

# Bernoulli & streched bounding (28*28 -> 20*20)
```

```r
resizingFunc <- function(inputdata){
  print("Start truncating vector!")
  # Resizing from 28*28 to 20*20
  col_to_delete <- vector()
  count <- 1
  for (i in 2:length(inputdata[1,])){
    row <- floor((i-2)/28)
    col <- (i-2) %% 28
    if ((row <= 3 || row >= 24) || (col <= 3 || col >= 24)){
      col_to_delete[count] <- i
      count <- count + 1
    }
  }
  boundingData <- inputdata[-col_to_delete]
  colnames(boundingData) <- c(0:400)
  colnames(boundingData)[1] <- "Number"
  print("Start stretching vector!")
  # Stretch the image
  max_col <- 1
  min_col <- 20
  max_row <- 1
  min_row <- 20
  for (k in 1:nrow(boundingData)){
    print(k)
    image_1 <- boundingData[k,][2:401]
    max_col <- 1
    min_col <- 20
    max_row <- 1
    min_row <- 20
    for (i in 1:20){
      for (j in 1:20){
        if(image_1[[(i-1)*20+j]] == 1){
          if (i < min_row)
            min_row <- i
          if (i > max_row)
            max_row <- i
          if (j < min_col)
            min_col <- j
          if (j > max_col)
            max_col <- j
        }
      }
    }
    row_a <- 20/(max_row-min_row)
    row_b <- 20*min_row/(min_row-max_row)
    col_a <- 20/(max_col-min_col)
    col_b <- 20*min_col/(min_col-max_col)
    for (i in 1:20){
      for (j in 1:20){
        if(image_1[[(i-1)*20+j]] == 1){
          end_y <- round(col_a*j+col_b)
          if (end_y < 1 || end_y > 20)
            end_y <- j
          image_1[[(i-1)*20+end_y]] <- 1
          end_x <- round(row_a*i+row_b)
          if (end_x < 1 || end_x > 20)
            end_x <- i
          image_1[[(end_x-1)*20+j]] <- 1
        }
      }
    }
    boundingData[k,][2:401] <- image_1
  }
  return(boundingData)
}
```

```r
library(randomForest)
# take in data from csv
image_train_data_untouched <- read.table("train_noheader.csv", sep = ',')
colnames(image_train_data_untouched)[1] <- "Number"
colnames(image_train_data_untouched)[2:785] <- sprintf("pixel%d", 1:784)
image_train_data_untouched$Number <- as.factor(image_train_data_untouched$Number)
image_train_data_untouched[image_train_data_untouched <= 127] <- 0
image_train_data_untouched[image_train_data_untouched >= 128] <- 1

# image_test_data_untouched <- read.table("val_noheader.csv", sep=',')
# colnames(image_test_data_untouched)[1] <- "Number"
# image_test_data_untouched$Number <- as.factor(image_test_data_untouched$Number)
# image_test_data_untouched[image_test_data_untouched <= 127] <- 0
# image_test_data_untouched[image_test_data_untouched >= 128] <- 1

image_res_data_untouched <- read.table("test.csv", sep=",")
colnames(image_res_data_untouched) <- sprintf("pixel%d", 1:784)
image_res_data_untouched[image_res_data_untouched <= 127] <- 0
image_res_data_untouched[image_res_data_untouched >= 128] <- 1

image_train_data_stretched <- read.table("train_stretched_noheader.csv", sep=",")
colnames(image_train_data_stretched)[1] <- "Number"
colnames(image_train_data_stretched)[2:401] <- sprintf("pixel%d", 1:400)
image_train_data_stretched$Number <- as.factor(image_train_data_stretched$Number)

# image_test_data_stretched <- read.table("val_stretched_noheader.csv", sep=',')
# colnames(image_test_data_stretched)[1] <- "Number"
# image_test_data_stretched$Number <- as.factor(image_test_data_stretched$Number)

image_res_data_stretched <- read.table("test_stretched2.csv", sep=",")
colnames(image_res_data_stretched) <- sprintf("pixel%d", 1:400)

# generate random forest
model_rf_untouched <- randomForest(Number~., data=image_train_data_untouched, maxnodes=15, ntree=10)
model_rf_stretched <- randomForest(Number~., data=image_train_data_stretched, maxnodes=15, ntree=10)

untouched_pred <- predict(model_rf_untouched, newdata=image_res_data_untouched)
stretched_pred <- predict(model_rf_stretched, newdata=image_res_data_stretched)
```