

Junhao Pan

CS 498 AM1

Homework 1

Part 1:

1A: 0.75817

1B: 0.74575

1D: 0.77386

```

9 # 1. Find p(y) aka prior
10 prior_0 <- 0
11 count_0 <- 0
12 for (i in 1:n_row) {
13   if (lbs[i] == 0)
14     count_0 <- count_0 + 1
15 }
16 prior_0 <- count_0/n_row
17 prior_1 <- 1 - prior_0
18
19 # 2. Find p(x|y) aka likelihood
20 normal_param_0 <- matrix(ncol = n_col - 1, nrow = 2)
21 normal_param_1 <- matrix(ncol = n_col - 1, nrow = 2)
22 for (i in 1:(n_col - 1)) {
23   feat_0 <- matrix(ncol = 1, nrow = count_0)
24   feat_1 <- matrix(ncol = 1, nrow = n_row - count_0)
25   n_0 <- 1
26   n_1 <- 1
27   for (j in 1:n_row) {
28     entry = d_train[j, i]
29     if (lbs[j] == 0) {
30       feat_0[n_0] <- entry
31       n_0 <- n_0 + 1
32     } else {
33       feat_1[n_1] <- entry
34       n_1 <- n_1 + 1
35     }
36   }
37   normal_param_0[i, 1] <- mean(feat_0)
38   normal_param_0[i, 2] <- sqrt(var(feat_0))
39   normal_param_1[i, 1] <- mean(feat_1)
40   normal_param_1[i, 2] <- sqrt(var(feat_1))
41 }
42
43 # 3. Test
44 n_row_test <- n_row_total - n_row
45 predicts <- matrix(ncol = 1, nrow = n_row_test)
46 for (i in 1:n_row_test) {
47   prob_0 <- log(prior_0)
48   prob_1 <- log(prior_1)
49   for (j in 1:(n_col - 1)) {
50     entry <- d_test[i, j]
51     prob_0 <- prob_0 + dnorm(entry, mean = normal_param_0[1, j], sd = normal_param_0[2, j], log = TRUE)
52     prob_1 <- prob_1 + dnorm(entry, mean = normal_param_1[1, j], sd = normal_param_1[2, j], log = TRUE)
53   }
54   if (prob_1 > prob_0) {
55     predicts[i, 1] <- 1
56   } else {
57     predicts[i, 1] <- 0
58   }
59 }

```

P1A

P1B is roughly the same with substitution of NA

```

1 library(caret)
2
3 d_train_all <- read.csv("pima-indians-diabetes.csv")
4 n_col_total <- ncol(d_train_all)
5 n_row_total <- nrow(d_train_all)
6
7 accuracy <- matrix(ncol = 1, nrow = 10)
8
9 for (t in 1:10) {
10
11   train_idx <- createDataPartition(d_train_all[,1], p = 0.8, list = FALSE, times = 1)
12
13   d_test <- d_train_all[-train_idx,]
14   d_train <- d_train_all[train_idx,]
15
16   n_col <- ncol(d_train)
17   n_row <- nrow(d_train)
18
19   lbs <- d_train[, 9]
20
21   svm <- svmlight(d_train[, 1:8], d_train[, 9], pathsvm = ".")
22   pred <- predict(svm, d_test[, 1:8])
23
24   correct_count <- 0
25   for (i in 1:n_row_test) {
26     if (pred[i] == d_test[i, 9])
27       correct_count <- correct_count + 1
28   }
29   accuracy[t, 1] <- correct_count/n_row_test
30 }
31
32 final_accuracy = mean(accuracy)








































```

P1D with SVM Light

Gaussian untouched	0.6746
Gaussian stretched	0.6576
Bernoulli untouched	0.6917
Bernoulli stretched	0.6346
10 trees 4 depth untouched	0.7284
10 trees 4 depth stretched	0.72515
10 trees 16 depth untouched	0.9575
10 trees 16 depth stretched	0.96415
32 trees 4 depth untouched	0.76645
32 trees 4 depth stretched	0.7558
32 trees 16 depth untouched	0.97055
32 trees 16 depth stretched	0.97455

It seems that the more trees in a forest, the more accurate the model is. More trees would enable better performance in making up the classifiers. Also, more depth the trees grow can fit the data better. In all, the 32 trees 16 depth scheme has the best performance.

Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions
Submission and Description							Public Score	Use for Final Score
jpan22_12.csv a few seconds ago by Junhao Pan add submission details							0.97455	<input type="checkbox"/>
jpan22_11.csv a few seconds ago by Junhao Pan add submission details							0.97055	<input type="checkbox"/>
jpan22_10.csv a few seconds ago by Junhao Pan add submission details							0.75580	<input type="checkbox"/>
jpan22_9.csv a few seconds ago by Junhao Pan add submission details							0.76645	<input type="checkbox"/>
jpan22_8.csv a minute ago by Junhao Pan add submission details							0.96415	<input type="checkbox"/>
jpan22_7.csv a minute ago by Junhao Pan add submission details							0.95750	<input type="checkbox"/>
jpan22_6.csv a minute ago by Junhao Pan add submission details							0.72515	<input type="checkbox"/>
jpan22_5.csv a minute ago by Junhao Pan add submission details							0.72840	<input type="checkbox"/>
jpan22_4.csv 2 minutes ago by Junhao Pan add submission details							0.63460	<input type="checkbox"/>
jpan22_3.csv 2 minutes ago by Junhao Pan add submission details							0.69170	<input type="checkbox"/>
jpan22_2.csv 2 minutes ago by Junhao Pan add submission details							0.65760	<input type="checkbox"/>
jpan22_1.csv 2 minutes ago by Junhao Pan							0.67460	<input type="checkbox"/>

	0	1	2	3	4	5	6	7	8	9
Gaussian untouched										
Gaussian stretched										
Bernoulli untouched										
Bernoulli stretched										

There must be a way to display in gray scale instead of either white or black. I failed to figure out how to do that...

```
library(caret)
library(EBImage)

thres <- 200

d_test_all <- read.csv("test.csv", header = FALSE)
d_train_all <- read.csv("train.csv")
d_val_all <- read.csv("val.csv")

source("util.R")

d_train_fit_all <- read.csv("d_train_fit.csv")
d_val_fit_all <- read.csv("d_val_fit.csv")
d_test_fit_all <- read.csv("d_test_fit.csv")

d_train_fit_all <- d_train_fit_all[, 2:ncol(d_train_fit_all)]
d_val_fit_all <- d_val_fit_all[, 2:ncol(d_val_fit_all)]
d_test_fit_all <- d_test_fit_all[, 2:ncol(d_test_fit_all)]

d_train <- d_train_all[, 2:ncol(d_train_all)]
d_val <- d_val_all[, ]
d_test <- d_test_all[, ]

d_train_fit <- d_train_fit_all[, ]
d_val_fit <- d_val_fit_all[, ]
d_test_fit <- d_test_fit_all[, ]
```

```
rb_train <- function(data_t, data_v, data_tt, n_tree, n_level, code) {

  x <- data_t[, 2:ncol(data_t)]
  y <- as.factor(data_t[, 1])
  rb <- Rborist(x, y, nTree = n_tree, nLevel = n_level)
  z <- data_v[, 2:ncol(data_v)]
  true <- as.factor(data_v[, 1])
  pred <- predict(rb, z)
  pred_tt <- predict(rb, data_tt)
  y_pred <- pred$yPred - 1
  y_pred_tt <- pred_tt$yPred - 1
  # y_pred_r <- round(y_pred, digits = 0)

  test_csv <- matrix(y_pred_tt, nrow = length(y_pred_tt), ncol = 1)
  file_str <- paste("jpan22_", code, ".csv", sep = "")
  write.csv(test_csv, file = file_str)

  correct <- 0
  confusion <- matrix(0L, nrow = 10, ncol = 10)
  for (i in 1:length(true)) {
    lb <- data_v[i, 1]
    pd <- y_pred[i]
    confusion[lb, pd] <- confusion[lb, pd] + 1
    if (pd == lb) {
      correct <- correct + 1
    }
  }
  accuracy <- correct/length(true)

  ret <- list(confusion, accuracy)
  return (ret)
}
```

```
pred <- function(data, priors, params, thres) {
  means <- matrix(unlist(params[1]), ncol = ncol(data), byrow = TRUE)
  stddv <- matrix(unlist(params[2]), ncol = ncol(data), byrow = TRUE)
  bern1 <- matrix(unlist(params[3]), ncol = ncol(data), byrow = TRUE)

  predicts_norm <- matrix(0L, nrow = nrow(data), ncol = 1)
  predicts_bern <- matrix(0L, nrow = nrow(data), ncol = 1)
  for (i in 1:nrow(data)) {
    probs_norm <- matrix(0L, nrow = 10, ncol = 1)
    probs_bern <- matrix(0L, nrow = 10, ncol = 1)
    for (j in 1:10) {
      probs_norm[j] <- log(priors[j])
      probs_bern[j] <- log(priors[j])
    }
    for (j in 3:ncol(data)) {
      entry <- data[i, j-1]
      for (k in 1:10) {
        if (means[k, j] != 0)
          probs_norm[k] <- probs_norm[k] + dnorm(entry, mean = means[k, j], sd = stddv[k, j], log = TRUE)
        if (entry > thres)
          probs_bern[k] <- probs_bern[k] + log(bern1[k, j])
      }
    }
    idx_norm <- arrayInd(which.max(probs_norm), dim(probs_norm))
    idx_bern <- arrayInd(which.max(probs_bern), dim(probs_bern))
    predicts_norm[i] <- idx_norm[, 1] - 1
    predicts_bern[i] <- idx_bern[, 1] - 1
  }
  ret = list(predicts_norm, predicts_bern)
  return (ret)
}
```

```
train <- function(data, counts, thres) {
  n_row <- nrow(data)
  n_col <- ncol(data)

  means <- matrix(0L, nrow = 10, ncol = n_col)
  stddv <- matrix(0L, nrow = 10, ncol = n_col)
  bern1 <- matrix(0L, nrow = 10, ncol = n_col)

  for (i in 3:n_col) {
    # First dimension for normal, second for bern
    feat_0 <- matrix(0L, nrow = counts[1], ncol = 2)
    feat_1 <- matrix(0L, nrow = counts[2], ncol = 2)
    feat_2 <- matrix(0L, nrow = counts[3], ncol = 2)
    feat_3 <- matrix(0L, nrow = counts[4], ncol = 2)
    feat_4 <- matrix(0L, nrow = counts[5], ncol = 2)
    feat_5 <- matrix(0L, nrow = counts[6], ncol = 2)
    feat_6 <- matrix(0L, nrow = counts[7], ncol = 2)
    feat_7 <- matrix(0L, nrow = counts[8], ncol = 2)
    feat_8 <- matrix(0L, nrow = counts[9], ncol = 2)
    feat_9 <- matrix(0L, nrow = counts[10], ncol = 2)
    nums <- matrix(0L, nrow = 10, ncol = 1)
    for (j in 1:n_row) {
      entry = data[j, i]
      # if (entry > thres)
      #   entry = entry + 500
      if (lbs[j] == 0) {
        nums[1] <- nums[1] + 1
        n <- nums[1]
        if (entry > thres) {
          feat_0[n, 1] <- entry
          feat_0[n, 2] <- 1
        }
      } else if (lbs[j] == 1) {
        nums[2] <- nums[2] + 1
        n <- nums[2]
        if (entry > thres) {
          feat_1[n, 1] <- entry
          feat_1[n, 2] <- 1
        }
      } else if (lbs[j] == 2) {
        nums[3] <- nums[3] + 1
        n <- nums[3]
        if (entry > thres) {
          feat_2[n, 1] <- entry
          feat_2[n, 2] <- 1
        }
      } else if (lbs[j] == 3) {
        nums[4] <- nums[4] + 1
        n <- nums[4]
        if (entry > thres) {
```