

Part1A: Average accuracy is 67.43%.

Part1B: Average accuracy is 74.19%.

Part1D: Average accuracy is 70.86%.

```

1 library(caret)
2 data <- read.table("pima-indians-diabetes.csv",sep = ',')
3 colnames(data) <- c("feat_a","feat_b","feat_c","feat_d","feat_e","feat_f","feat_g","feat_h","res")
4 for (i in 1:nrow(data)){
5   if (data[i,]$feat_c == 0){
6     data[i,]$feat_c <- NA
7   }
8   if(data[i,]$feat_d == 0){
9     data[i,]$feat_d <- NA
10  }
11  if(data[i,]$feat_f == 0){
12    data[i,]$feat_f <- NA
13  }
14  if(data[i,]$feat_h == 0){
15    data[i,]$feat_h <- NA
16  }
17 }
18 start <- 1
19 end <- round(nrow(data)/10)
20 total_acc <- 0
21 for (i in 1:10){
22   eva_data <- data[start:end,]
23   ind <- createDataPartition(eva_data$res, times=1, p=0.8, list=F)
24   train_data <- eva_data[ind,]
25   test_data <- eva_data[-ind,]
26   p_y <- sum(train_data$res=="TRUE")/nrow(train_data)
27   p_n <- sum(train_data$res=="FALSE")/nrow(train_data)
28   ind_y <- train_data$res == "TRUE"
29   ind_n <- train_data$res == "FALSE"
30   feat_a_y_mean <- mean(train_data[ind_y,]$feat_a)
31   feat_a_n_mean <- mean(train_data[ind_n,]$feat_a)
32   feat_a_y_sd <- sd(train_data[ind_y,]$feat_a)
33   feat_a_n_sd <- sd(train_data[ind_n,]$feat_a)
34   feat_b_y_mean <- mean(train_data[ind_y,]$feat_b)
35   feat_b_n_mean <- mean(train_data[ind_n,]$feat_b)
36   feat_b_y_sd <- sd(train_data[ind_y,]$feat_b)
37   feat_b_n_sd <- sd(train_data[ind_n,]$feat_b)
38   feat_c_y_mean <- mean(train_data[ind_y,]$feat_c, na.rm=TRUE)
39   feat_c_n_mean <- mean(train_data[ind_n,]$feat_c, na.rm=TRUE)
40   feat_c_y_sd <- sd(train_data[ind_y,]$feat_c, na.rm=TRUE)
41   feat_c_n_sd <- sd(train_data[ind_n,]$feat_c, na.rm=TRUE)
42   feat_d_y_mean <- mean(train_data[ind_y,]$feat_d, na.rm=TRUE)

```

```

1 library(caret)
2 library(klaR)
3 data <- read.table("pima-indians-diabetes.csv",sep = ',')
4 colnames(data) <- c("feat_a","feat_b",
5   "feat_c","feat_d",
6   "feat_e","feat_f",
7   "feat_g","feat_h",
8   "res")
9
10 start <- 1
11 end <- round(nrow(data)/10)
12 total_acc <- 0
13 for (i in 1:10){
14   eva_data = data[start:end,]
15   ind <- createDataPartition(eva_data$res, times=1, p=0.8, list=F)
16   train_data = eva_data[ind,]
17   test_data = eva_data[-ind,]
18   model <- svmlight(res~, data=train_data)
19   pred <- predict(model, newdata=test_data)
20   svm_table <- table(actual=test_data$res, predict=pred$class)
21   ratio <- sum(diag(svm_table))/sum(svm_table)
22
23   print(svm_table)
24   print(ratio)
25
26   total_acc <- total_acc + ratio
27
28   start <- start + round(nrow(data)/10)
29   end <- end + round(nrow(data)/10)
30   if(end > nrow(data))
31     end <- nrow(data)
32 }
33
34 print(total_acc/10)
35
36

```

```

63 for (i in 1:nrow(test_data)){
64   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)+
65     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)+
66     log(p_y)
67   if (!is.na(dnorm(test_data[i,]$feat_c,feat_c_y_mean,feat_c_y_sd,log=T)))
68     y <- y + dnorm(test_data[i,]$feat_c,feat_c_y_mean,feat_c_y_sd,log=T)
69   if (!is.na(dnorm(test_data[i,]$feat_d,feat_d_y_mean,feat_d_y_sd,log=T)))
70     y <- y + dnorm(test_data[i,]$feat_d,feat_d_y_mean,feat_d_y_sd,log=T)
71   if (!is.na(dnorm(test_data[i,]$feat_f,feat_f_y_mean,feat_f_y_sd,log=T)))
72     y <- y + dnorm(test_data[i,]$feat_f,feat_f_y_mean,feat_f_y_sd,log=T)
73   if (!is.na(dnorm(test_data[i,]$feat_h,feat_h_y_mean,feat_h_y_sd,log=T)))
74     y <- y + dnorm(test_data[i,]$feat_h,feat_h_y_mean,feat_h_y_sd,log=T)
75   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)+
76     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)+
77     log(p_n)
78   if (!is.na(dnorm(test_data[i,]$feat_c,feat_c_n_mean,feat_c_n_sd,log=T)))
79     n <- n + dnorm(test_data[i,]$feat_c,feat_c_n_mean,feat_c_n_sd,log=T)
80   if (!is.na(dnorm(test_data[i,]$feat_d,feat_d_n_mean,feat_d_n_sd,log=T)))
81     n <- n + dnorm(test_data[i,]$feat_d,feat_d_n_mean,feat_d_n_sd,log=T)
82   if (!is.na(dnorm(test_data[i,]$feat_f,feat_f_n_mean,feat_f_n_sd,log=T)))
83     n <- n + dnorm(test_data[i,]$feat_f,feat_f_n_mean,feat_f_n_sd,log=T)
84   if (!is.na(dnorm(test_data[i,]$feat_h,feat_h_n_mean,feat_h_n_sd,log=T)))
85     n <- n + dnorm(test_data[i,]$feat_h,feat_h_n_mean,feat_h_n_sd,log=T)
86   if(y>n){
87     pred[i] <- c("TRUE")
88   } else {
89     pred[i] <- ("FALSE")
90   }
91 }
92 nb_table <- table(actual=test_data$res, predict=pred)
93 ratio <- sum(diag(nb_table))/sum(nb_table)
94 print(nb_table)
95 print(ratio)
96 total_acc <- total_acc + ratio
97 start <- start + round(nrow(data)/10)
98 end <- end + round(nrow(data)/10)
99 if(end > nrow(data))
100   end <- nrow(data)
101 }
102 print(total_acc/10)
103

```

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%




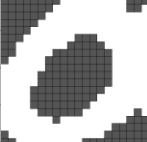
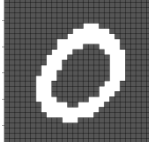
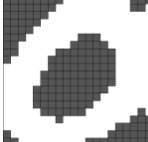
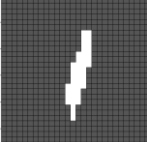
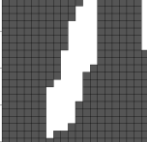
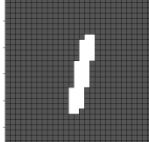
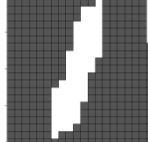

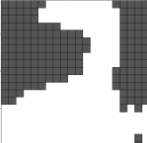
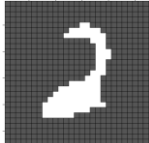
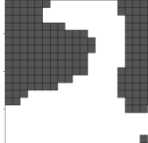
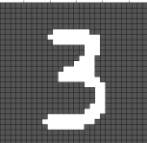
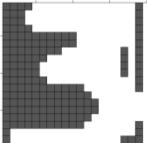
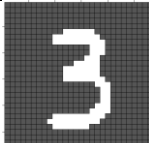
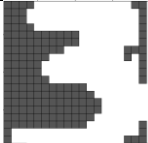

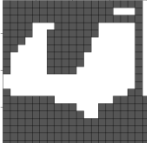

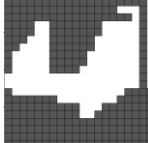
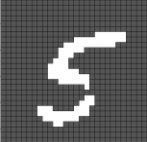
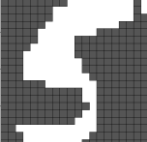
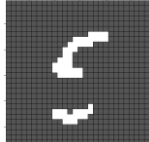
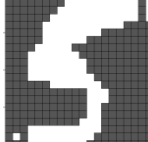

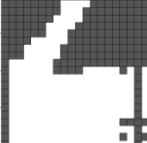
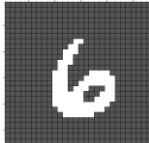
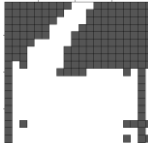
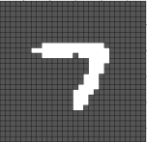
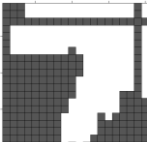
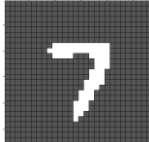
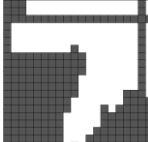
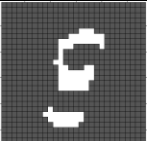
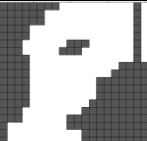
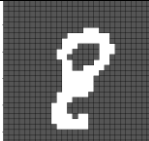
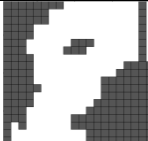
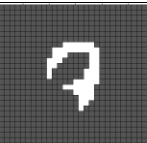
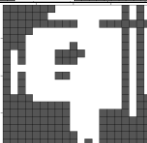

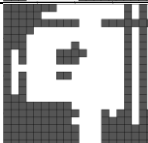
The image shows a terminal window at the top with the command: `kaggle competitions submit -c anl-hw1-part2 -f submission.csv -m "Message"`. Below the terminal is a screenshot of the Kaggle competition submission page for Hengzhe Ding. The page shows 6 submissions, all successful. The submissions are sorted by name. The table below represents the data shown in the screenshot.

Submission and Description	Public Score	Use for Final Score
<a href="#">hengzhe2_1.csv</a> 41 minutes ago by Hengzhe Ding <a href="#">add submission details</a>	0.61825	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_10.csv</a> an hour ago by Hengzhe Ding <a href="#">add submission details</a>	0.72260	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_11.csv</a> an hour ago by Hengzhe Ding <a href="#">add submission details</a>	0.96895	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_12.csv</a> an hour ago by Hengzhe Ding <a href="#">add submission details</a>	0.93570	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_2.csv</a> 31 minutes ago by Hengzhe Ding <a href="#">add submission details</a>	0.81550	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_3.csv</a> 23 minutes ago by Hengzhe Ding <a href="#">add submission details</a>	0.83525	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_4.csv</a> 6 minutes ago by Hengzhe Ding <a href="#">add submission details</a>	0.82660	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_5.csv</a> 3 hours ago by Hengzhe Ding <a href="#">add submission details</a>	0.62430	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_6.csv</a> 2 hours ago by Hengzhe Ding <a href="#">add submission details</a>	0.68135	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_7.csv</a> an hour ago by Hengzhe Ding <a href="#">add submission details</a>	0.95185	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_8.csv</a> an hour ago by Hengzhe Ding <a href="#">add submission details</a>	0.90195	<input checked="" type="checkbox"/>
<a href="#">hengzhe2_9.csv</a> an hour ago by Hengzhe Ding <a href="#">add submission details</a>	0.64925	<input checked="" type="checkbox"/>

No more submissions to show

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
			
			
			
			
			
			
			
			
			
			

```

1 library(e1071)
2 library(sparklyr)
3 library(dplyr)
4 Sys.setenv(
5   SPARK_HOME="C:\\spark\\spark-2.3.1-bin-hadoop2.7"
6 )
7 sc <- spark_connect(master = "local")
8
9 # Take in data
10 image_train_data <- read.table("train.csv", sep = ',')
11 # image_test_data <- read.table("val.csv", sep=',')
12
13 # Rename column of data frame
14 colnames(image_train_data)[1] <- "Number"
15 # colnames(image_test_data)[1] <- "Number"
16 colnames(image_train_data)[2:785] <- sprintf("pixel%d", 1:784)
17 # colnames(image_test_data)[2:785] <- sprintf("pixel%d", 1:784)
18 image_train_data$Number <- as.factor(image_train_data$Number)
19 # image_test_data$Number <- as.factor(image_test_data$Number)
20
21 # Set a threshold (127)
22 image_train_data[image_train_data <= 127] <- 0
23 image_train_data[image_train_data >= 128] <- 1
24 # image_test_data[image_test_data <= 127] <- 0
25 image_test_data[image_test_data >= 128] <- 1
26
27 # Resizing function that resize and stretch the image
28 resizingFunc <- function(inputdata){
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95 # Gaussian & untouched
96 image_res_data <- read.table("test.csv", sep=",")
97 colnames(image_res_data) <- sprintf("pixel%d", 1:784)
98 image_res_data[image_res_data <= 127] <- 0
99 image_res_data[image_res_data >= 128] <- 1
100 image_model <- naiveBayes(Number~., data=image_train_data)
101 image_pred <- predict(image_model, newdata=image_res_data)
102 # image table <- table(actual=image_test_data$Number, predict=image_pred)

```

```

1 library(randomForest)
2 # take in data from csv
3 image_train_data_untouched <- read.table("train_noheader.csv", sep = ',')
4 colnames(image_train_data_untouched)[1] <- "Number"
5 colnames(image_train_data_untouched)[2:785] <- sprintf("pixel%d", 1:784)
6 image_train_data_untouched$Number <- as.factor(image_train_data_untouched$Number)
7 image_train_data_untouched[image_train_data_untouched <= 127] <- 0
8 image_train_data_untouched[image_train_data_untouched >= 128] <- 1
9
10 # image_test_data_untouched <- read.table("val_noheader.csv", sep=',')
11 # colnames(image_test_data_untouched)[1] <- "Number"
12 # image_test_data_untouched$Number <- as.factor(image_test_data_untouched$Number)
13 # image_test_data_untouched[image_test_data_untouched <= 127] <- 0
14 # image_test_data_untouched[image_test_data_untouched >= 128] <- 1
15
16 image_res_data_untouched <- read.table("test.csv", sep=",")
17 colnames(image_res_data_untouched) <- sprintf("pixel%d", 1:784)
18 image_res_data_untouched[image_res_data_untouched <= 127] <- 0
19 image_res_data_untouched[image_res_data_untouched >= 128] <- 1
20
21 image_train_data_stretched <- read.table("train_stretched_noheader.csv", sep=",")
22 colnames(image_train_data_stretched)[1] <- "Number"
23 colnames(image_train_data_stretched)[2:401] <- sprintf("pixel%d", 1:400)
24 image_train_data_stretched$Number <- as.factor(image_train_data_stretched$Number)
25
26 # image_test_data_stretched <- read.table("val_stretched_noheader.csv", sep=',')
27 # colnames(image_test_data_stretched)[1] <- "Number"
28 # image_test_data_stretched$Number <- as.factor(image_test_data_stretched$Number)
29
30 image_res_data_stretched <- read.table("test_stretched2.csv", sep=",")
31 colnames(image_res_data_stretched) <- sprintf("pixel%d", 1:400)
32
33 # generate random forest
34 model_rf_untouched <- randomForest(Number~., data=image_train_data_untouched, maxnodes=15, ntree=10)
35 model_rf_stretched <- randomForest(Number~., data=image_train_data_stretched, maxnodes=15, ntree=10)
36
37 untouched_pred <- predict(model_rf_untouched, newdata=image_res_data_untouched)
38 stretched_pred <- predict(model_rf_stretched, newdata=image_res_data_stretched)

```

```

107
108 # Gaussian & stretched bounding (28*28 -> 20*20)
109
110 # resizing bounding first
111 image_train_data <- read.table("train_stretched_noheader.csv", sep=",")
112 colnames(image_train_data)[1] <- "Number"
113 colnames(image_train_data)[2:401] <- sprintf("pixel%d", 1:400)
114 image_train_data$Number <- as.factor(image_train_data$Number)
115 image_res_data <- read.table("test_stretched2.csv", sep=",")
116 colnames(image_res_data) <- sprintf("pixel%d", 1:400)
117
118 image_model <- naiveBayes(Number~., data=image_train_data)
119 image_pred <- predict(image_model, newdata=image_res_data)
120
121
122 # Bernoulli & untouched
123 #
124 image_res_data <- read.table("test.csv", sep=",")
125 colnames(image_res_data) <- sprintf("pixel%d", 1:784)
126 image_res_data[image_res_data <= 127] <- 0
127 image_res_data[image_res_data >= 128] <- 1
128
129 image_train_tbl <- copy_to(sc, image_train_data[1:30000,])
130 image_res_tbl <- copy_to(sc, image_res_data)
131 image_ber_model <- image_train_tbl %>%
132   ml_naive_bayes(Number~., model_type="bernoulli")
133 image_ber_pred <- predict(image_ber_model, newdata=image_res_tbl)
134 # image_ber_table <- table(actual=image_test_data[,1], predict=image_ber_pred)
135 # ratio <- sum(diag(image_ber_table))/sum(image_ber_table)
136 #
137 # print(image_ber_table)
138 # print(ratio)
139
140 # Bernoulli & stretched bounding (28*28 -> 20*20)

```

```

28 resizingFunc <- function(inputdata){
29   print("Start truncating vector!")
30   # Resizing from 28*28 to 20*20
31   col_to_delete <- vector()
32   count <- 1
33   for (i in 2:length(inputdata[,1])){
34     row <- floor((i-2)/28)
35     col <- (i-2) %% 28
36     if ((row <= 3 || row >= 24) || (col <= 3 || col >= 24)){
37       col_to_delete[count] <- i
38       count <- count + 1
39     }
40   }
41   boundingData <- inputdata[,col_to_delete]
42   colnames(boundingData) <- c(0:400)
43   colnames(boundingData)[1] <- "Number"
44   print("Start stretching vector!")
45   # Stretch the image
46   max_col <- 1
47   min_col <- 20
48   max_row <- 1
49   min_row <- 20
50   for (k in 1:nrow(boundingData)){
51     print(k)
52     image_1 <- boundingData[k,][2:401]
53     max_col <- 1
54     min_col <- 20
55     max_row <- 1
56     min_row <- 20
57     for (i in 1:20){
58       for (j in 1:20){
59         if(image_1[[(i-1)*20+j]] == 1){
60           if (i < min_row)
61             min_row <- i
62           if (i > max_row)
63             max_row <- i
64           if (j < min_col)
65             min_col <- j
66           if (j > max_col)
67             max_col <- j
68         }
69       }
70     }
71     row_a <- 20/(max_row-min_row)
72     row_b <- 20*min_row/(min_row-max_row)
73     col_a <- 20/(max_col-min_col)
74     col_b <- 20*min_col/(min_col-max_col)
75     for (i in 1:20){
76       for (j in 1:20){
77         if(image_1[[(i-1)*20+j]] == 1){
78           end_y <- round(col_a*j+col_b)
79           if (end_y < 1 || end_y > 20)
80             end_y <- j
81           image_1[[(i-1)*20+end_y]] <- 1
82           end_x <- round(row_a*i+row_b)
83           if (end_x < 1 || end_x > 20)
84             end_x <- i
85           image_1[[(end_x-1)*20+j]] <- 1
86         }
87       }
88     }
89     boundingData[k,][2:401] <- image_1
90   }
91   return(boundingData)
92 }

```