CS 498 AML HW1 Report

Author: Moyu Liu

Part1:

Dataset: https://www.kaggle.com/kumargh/pimaindiansdiabetescsv
Language: R-Studio
Reference:
SVM_Light : http://svmlight.joachims.org
Invoke SVN: https://www.java.com/en/download/help/path.xml
Package: SVM_Light, klaR, caret

Part2:

Dataset: https://www.kaggle.com/c/aml-hw1-part2
Language: R-Studio
Reference:
Rescale Image:
https://stackoverflow.com/questions/11123152/function-for-resizing-matrices-in-r
Plot Images:
Par and Cex:
http:/groups.linguistics.northwestern.eduspeech_comm_group/documents/Presentation_par_cex.pdf
Package: naivebayes, BBMISC, h2o, quanteda

Accuracies:

Part1A: Build a simple naive Bayes classifier to classify this data set.

Average Accuracy: 0.7746753

Part1B: Same as part A but take out '0' values in attributes 3, 4, 6, and 8.

Average Accuracy: 0.7461039

Part1D: Use SVM_Light to to train and evaluate an SVM to classify this data.

Average Accuracy: 0.7571429

Screenshot of Part1

This screenshot contains the test-train split.

```
#Generate 10 times cross validation
for(d in (1:10)) {

  correct = 0
  #Split data into train and test data
  training_index = sample(nrow(data_table), nrow(data_table)*0.8)
  testing_index = -training_index
  train_data = data_table[training_index, ]
  test_data = data_table[testing_index, ]

  #Calculate class percentage
  cat1 = 0
  for(num in train_data$class)
    if(num == 1)
      cat1 = cat1 + 1

  P1 = cat1/(0.8*nrow(data_table))
  P0 = 1 - P1

  mean_table <- array(0, c(2,8))
  std_table <- array(0, c(2,8))
```

This screenshot contains the probability calculation and evaluation.

```
#Calculate mean and std of each attribute
for(i in (1:2))
  for(j in (1:8)) {
    if(j==3|j==4|j==6|j==8) {
      attr_vals = train_data[train_data[['class']] == i-1, j]
      mean_table[i,j] = mean(attr_vals[attr_vals!=0])
      std_table[i,j] = sd(attr_vals[attr_vals!=0])
    }
    else {
      mean_table[i,j] = mean(train_data[train_data[['class']] == i-1, j])
      std_table[i,j] = sd(train_data[train_data[['class']] == i-1, j])
    }
  }

#Check accuracy on test data
for(i in (1:nrow(test_data))) {
  cat0_sum = log(P0)
  cat1_sum = log(P1)

  for(j in (1:8)) {
    cat0_sum = cat0_sum + dnorm(test_data[i,j], mean_table[1,j], std_table[1, j], log=TRUE)
    cat1_sum = cat1_sum + dnorm(test_data[i,j], mean_table[2,j], std_table[2, j], log=TRUE)
  }

  if(cat0_sum >= cat1_sum & test_data[i,'class'] == 0)
    correct = correct+1
  if(cat1_sum >= cat0_sum & test_data[i,'class'] == 1)
    correct = correct+1
}

avg_acc2 = avg_acc2 + correct/(nrow(test_data)*10)
```

Table of accuracies for Part2

Untouched		Stretched	
Gaussian	0.52354	Gaussian	0.82355
Bernoulli	0.83375	Bernoulli	0..79460
10 trees + 4 depth	0.84990	10 trees + 4 depth	0.85295
10 trees + 16 depth	0.96720	10 trees + 16 depth	0.96745
30 trees + 4 depth	0.86205	30 trees + 4 depth	0.86930
30 trees + 16 depth	0.97440	30 trees + 16 depth	0.97445

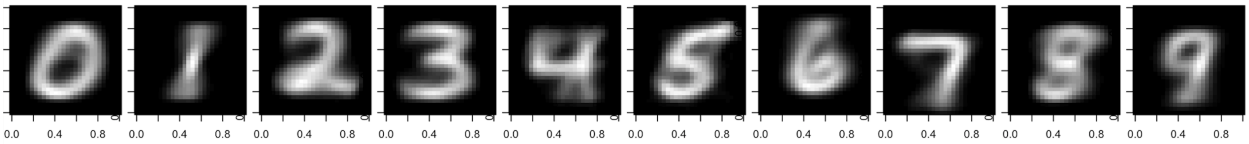
Explanation:

For Naive Bayes classifier, Bernoulli distribution is much better for untouched image. The reason is that Gaussian distribution using untouched image results in many 0 entries, which leads to low variances. Bernoulli distribution is better for data with 0 and 1 data. But when the image is stretched, Gaussian is good for dealing with image with useful information.

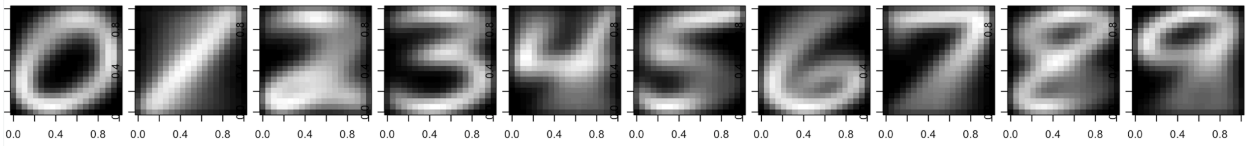
For Random forest classifier, more trees result in better accuracy because by merging the trees, it gives better result after merging. Also, the more depth we have, the higher accuracy since more depth means more branches to classify the data. As for untouched and stretched image, the higher portion of useful pixels in the image, the higher accuracy we shall get, even though the difference is really small from the table.

Screenshot for Generated Images

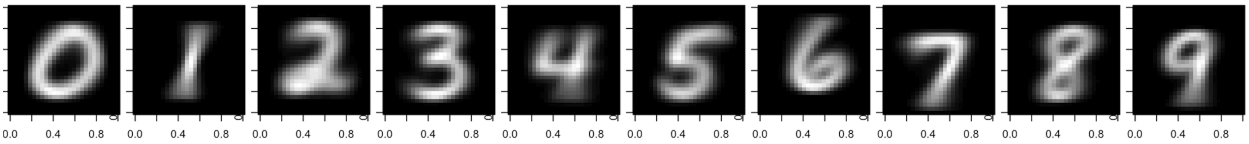
Gaussian + Untouched



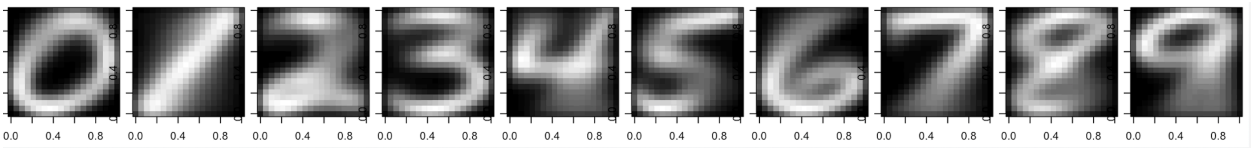
Gaussian + Stretched



Bernoulli + Untouched



Bernoulli + Stretched



Part2A: Rescale Image

```
#Resize the image
ResizeImage = function(before_resize){

  numrow = nrow(before_resize)
  after_resize = rep(0, nrow(before_resize) * 400)
  after_resize = matrix(after_resize, nrow = numrow, ncol = 400, byrow = TRUE)

  # Convert row to 28 x 28 matrix
  for(a in 1:nrow(before_resize)){

    orig_img = matrix(as.numeric(before_resize[a,1:ncol(before_resize)]), nrow = 28, ncol = 28, byrow = TRUE)

    left = 0
    right = 0
    top = 0
    bot = 0

    for(i in 1:28) {
      flag = 0
      for(j in 1:28) {
        if(orig_img[i,j]!=0) {
          top = i
          flag = 1
          break;
        }
      }
      if(flag == 1) {
        break;
      }
    }
```

```
# Rescale image functions
rescale <- function(x, newrange=range(x)){
  xrange <- range(x)
  mfac <- (newrange[2]-newrange[1])/(xrange[2]-xrange[1])
  newrange[1]+(x-xrange[1])*mfac
}

ResizeMat <- function(mat, ndim=dim(mat)){
  if(!require(fields)) stop("`fields` required.")

  # input object
  odim <- dim(mat)
  obj <- list(x= 1:odim[1], y=1:odim[2], z= mat)

  # output object
  ans <- matrix(NA, nrow=ndim[1], ncol=ndim[2])
  ndim <- dim(ans)

  # rescaling
  ncord <- as.matrix(expand.grid(seq_len(ndim[1]), seq_len(ndim[2])))
  loc <- ncord
  loc[,1] = rescale(ncord[,1], c(1,odim[1]))
  loc[,2] = rescale(ncord[,2], c(1,odim[2]))

  # interpolation
  ans[ncord] <- interp.surface(obj, loc)

  return(as.numeric(t(ans)))
}
```

Library & Evaluation

```
```{r}
library("quanteda")
library("e1071")

# Extract data
train_data = read.csv("dataset/train.csv")
ut_train = train_data[, 3:786]
st_train = as.data.frame(ResizeImage(ut_train))

val_data = read.csv("dataset/val.csv")
ut_val = val_data[, 2:785]
st_val = as.data.frame(ResizeImage(ut_val))

test_data = read.csv("dataset/test.csv", header = FALSE)
ut_test = test_data
colnames(ut_test) = colnames(ut_train)
st_test = as.data.frame(ResizeImage(test_data))
colnames(st_test) = colnames(st_train)
```
```

#For Gaussian Untouched
```
```{r}
gau_ut = naiveBayes(ut_train, as.factor(train_data[, 2]), laplace = 3)
predict1 = as.numeric(as.vector(predict(gau_ut, ut_test)))
csv1 = cbind((0:19999), data.frame(col1 = seq(0,19999), col2 = predict1))
colnames(csv1) = c('ImageId', 'Label')
```
```

#For Gausian Stressed
```
```{r}
gau_st = naiveBayes(st_train, as.factor(train_data[, 2]), laplace = 3)
predict2 = as.numeric(as.vector(predict(gau_st, st_test)))
csv2 = cbind((0:19999), data.frame(col1 = seq(0,19999), col2 = predict2))
colnames(csv2) = colnames(csv1)
```
```

#For Bernoulli Untouched
```
```{r}
ber_ut = textmodel_nb(as.dfm(ut_train), train_data[, 2], distribution = c("Bernoulli"))
predict3 = as.numeric(as.vector(predict(ber_ut, ut_test)))
csv3 = cbind((0:19999), data.frame(col1 = seq(0,19999), col2 = predict3))
colnames(csv3) = colnames(csv2)
```
```

#For Bernoulli Stressed
```
```{r}
ber_st = textmodel_nb(as.dfm(st_train), train_data[, 2], distribution = c("Bernoulli"))
predict4 = as.numeric(as.vector(predict(ber_st, st_test)))
csv4 = cbind((0:19999), data.frame(col1 = seq(0,19999), col2 = predict4))
colnames(csv4) = colnames(csv2)
```
```

Part2B:

```
```{r}
generate_forest <- function(train, test, numTrees, depth){
  trainHex<-as.h2o(train)
  rfHex <- h2o.randomForest(x=(colnames(train)[-1]), y=colnames(train)[1],
                            ntrees = numTrees, max_depth = depth,
                            training_frame=trainHex)
  testHex<-as.h2o(test)
  result = cbind((0:19999), as.data.frame(h2o.predict(rfHex,testHex)[,1]))
  colnames(result) <- c('ImageId','Label')
  predictions<-as.data.frame(h2o.predict(rfHex,testHex))
  print(sum(predictions[,1] == test_data[,1])/nrow(test_data))
  return(result)
}

colnames(test_data) <- colnames(train_data[3,786])
colnames(st_test) <- colnames(train_only)[2:401]

output5 = generate_forest(train_only, ut_test, 10, 4)
output6 = generate_forest(total_rescale_train, st_test, 10, 4)
output7 = generate_forest(train_only, ut_test, 10, 16)
output8 = generate_forest(total_rescale_train, st_test, 10, 16)
output9 = generate_forest(train_only, ut_test, 30, 4)
output10 = generate_forest(total_rescale_train, st_test, 30, 4)
output11 = generate_forest(train_only, ut_test, 30, 16)
output12 = generate_forest(total_rescale_train, st_test, 30, 16)

write.csv(output5, file = 'result/moyuliu2_05.csv', row.names=FALSE)
write.csv(output6, file = 'result/moyuliu2_06.csv', row.names=FALSE)
write.csv(output7, file = 'result/moyuliu2_07.csv', row.names=FALSE)
write.csv(output8, file = 'result/moyuliu2_08.csv', row.names=FALSE)
write.csv(output9, file = 'result/moyuliu2_09.csv', row.names=FALSE)
write.csv(output10, file = 'result/moyuliu2_10.csv', row.names=FALSE)
write.csv(output11, file = 'result/moyuliu2_11.csv', row.names=FALSE)
write.csv(output12, file = 'result/moyuliu2_12.csv', row.names=FALSE)
```
```