Homework 1

Problem 1：

1. **Three accuracies for 1A, 1B, 1D**
   a) **Part 1A:** 74.31373%
   b) **Part 1B:** 75.88235%
   c) **Part 1D:** 77.45098%

2. **Screenshot of code:**
   a) **Test-Train Split:**

```r
# extract train data by randomly choosing 80% from origin dataset
train_index <- createDataPartition(y=input_y, p=.8, list=FALSE)
```

b) Probability Calculations:

```r
# use a normal distribution to model distributions
# calculate mean and standard deviation
train_x_positive_mean <- sapply(train_x_positive, mean, na.rm=TRUE)
train_x_negative_mean <- sapply(train_x_negative, mean, na.rm=TRUE)

train_x_positive_sd <- sapply(train_x_positive, sd, na.rm=TRUE)
train_x_negative_sd <- sapply(train_x_negative, sd, na.rm=TRUE)

# calculate log probability
train_x_positive_offset <- t(t(train_x) - train_x_positive_mean)
train_x_positive_scale <- t(t(train_x_positive_offset) / train_x_positive_sd)
train_x_positive_square <- apply(train_x_positive_scale, c(1, 2), function(x)x^2)
train_x_positive_log_prob <- -(1/2)*rowSums(train_x_positive_square, na.rm=TRUE) - sum(log(train_x_positive_sd))

train_x_negative_offset <- t(t(train_x) - train_x_negative_mean)
train_x_negative_scale <- t(t(train_x_negative_offset) / train_x_negative_sd)
train_x_negative_square <- apply(train_x_negative_scale, c(1, 2), function(x)x^2)
train_x_negative_log_prob <- -(1/2)*rowSums(train_x_negative_square, na.rm=TRUE) - sum(log(train_x_negative_sd))
```

c) Evaluations:

```r
# calculate log probability
test_x_positive_offset <- t(t(test_x) - train_x_positive_mean)
test_x_positive_scale <- t(t(test_x_positive_offset) / train_x_positive_sd)
test_x_positive_square <- apply(test_x_positive_scale, c(1, 2), function(x)x^2)
test_x_positive_log_prob <- -(1/2)*rowSums(test_x_positive_square, na.rm=TRUE) - sum(log(train_x_positive_sd))

test_x_negative_offset <- t(t(test_x) - train_x_negative_mean)
test_x_negative_scale <- t(t(test_x_negative_offset) / train_x_negative_sd)
test_x_negative_square <- apply(test_x_negative_scale, c(1, 2), function(x)x^2)
test_x_negative_log_prob <- -(1/2)*rowSums(test_x_negative_square, na.rm=TRUE) - sum(log(train_x_negative_sd))

# predict labels
test_predict_y <- test_x_positive_log_prob > test_x_negative_log_prob

# calculate test accuracy
test_correct_account <- test_predict_y == test_y
test_accuracy[iter] <- sum(test_correct_account)/(sum(test_correct_account)+sum(!test_correct_account))
```

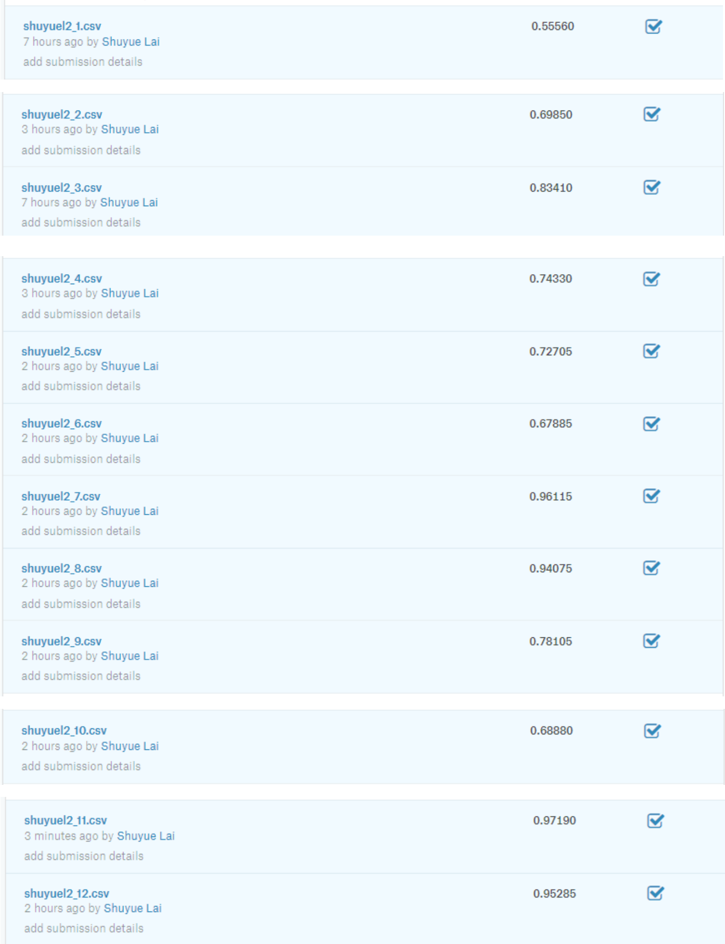
Problem 2：

3.

a) Table of accuracies for all 12 cases:

Case	1	2	3	4
Accuracy	55.560%	69.850%	83.410%	74.330%
Case	5	6	7	8
Accuracy	72.705%	67.885%	96.115%	94.075%
Case	9	10	11	12
Accuracy	78.105%	68.880%	97.190%	95.285%

b) Screenshot of Kaggle



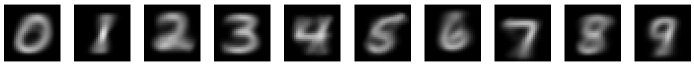
c) A brief explanation of which model is better and why:

i. According to the result, Bernoulli Naïve Bayes is better than Gaussian Naïve Bayes, because Bernoulli distribution works better in discrete cases which is MNIST acts.

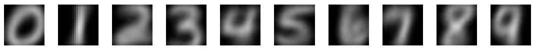
ii. According to the result, Random Forest Classifier works better with larger number of trees and larger maximum depth.

4. 40 mean images (4 * 10 of part 2A)

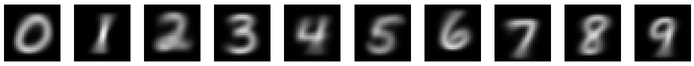
a) Case 1: Gaussian + Untouched



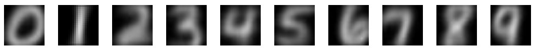
b) **Case 2: Gaussian + Stretched**



c) **Case 3: Bernoulli + Untouched**



d) **Case 4: Bernoulli + Stretched**



5. **Screenshot of code:**

a) **Library:**

```python
import csv
import numpy as np
import skimage.transform
from PIL import Image

from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.ensemble import RandomForestClassifier
```

b) Evaluations:

```python
# Gaussian Naive Bayes Classifier
def gaussianNaiveBayes(train_input_x, train_input_y, test_input_x):
    classifier = GaussianNB()
    classifier.fit(train_input_x, train_input_y.ravel())

    test_output_y = classifier.predict(test_input_x)
    return test_output_y

# Bernoulli Naive Bayes Classifier
def bernoulliNaiveBayes(train_input_x, train_input_y, test_input_x):
    classifier = BernoulliNB()
    classifier.fit(train_input_x, train_input_y.ravel())

    test_output_y = classifier.predict(test_input_x)
    return test_output_y

# Random Forest Classifier
def randomForest(num_tree, max_depth, train_input_x, train_input_y, test_input_x):
    classifier = RandomForestClassifier(n_estimators=num_tree, criterion='entropy', max_depth=max_depth)
    classifier.fit(train_input_x, train_input_y)

    test_output_y = classifier.predict(test_input_x)
    return test_output_y
```

```python
    ## EXECUTE
    # 1. GAUSSIAN + UNTOUCHED
    test_output_y = gaussianNaiveBayes(train_input_x, train_input_y, test_input_x)
    writeCsvFile("shuyuel2_1.csv", test_output_y)

    test_output_y = np.array(test_output_y).astype(float)
    meanImage(test_input_x, test_output_y, 28, "shuyuel2_1_")

    # 2. GAUSSIAN + STRETCHED
    test_output_y = gaussianNaiveBayes(train_input_x_stretched, train_input_y, test_input_x_stretched)
    writeCsvFile("shuyuel2_2.csv", test_output_y)

    test_output_y = np.array(test_output_y).astype(float)
    meanImage(test_input_x_stretched, test_output_y, 20, "shuyuel2_2_")

    # 3. BERNOULLI + UNTOUCHED
    test_output_y = bernoulliNaiveBayes(train_input_x, train_input_y, test_input_x)
    writeCsvFile("shuyuel2_3.csv", test_output_y)

    test_output_y = np.array(test_output_y).astype(float)
    meanImage(test_input_x, test_output_y, 28, "shuyuel2_3_")

    # 4. BERNOULLI + STRETCHED
    test_output_y = bernoulliNaiveBayes(train_input_x_stretched, train_input_y, test_input_x_stretched)
    writeCsvFile("shuyuel2_4.csv", test_output_y)

    test_output_y = np.array(test_output_y).astype(float)
    meanImage(test_input_x_stretched, test_output_y, 20, "shuyuel2_4_")

    # 5. 10 TREES + 4 DEPTH + UNTOUCHED
    test_output_y = randomForest(10, 4, train_input_x, train_input_y, test_input_x)
    writeCsvFile("shuyuel2_5.csv", test_output_y)

    # 6. 10 TREES + 4 DEPTH + STRETCHED
    test_output_y = randomForest(10, 4, train_input_x_stretched, train_input_y, test_input_x_stretched)
    writeCsvFile("shuyuel2_6.csv", test_output_y)
```