CS498 AMO Homework 1

Team :
Minyuan Gu (minyuan3@illinois.edu)
Yanislav Shterev (shterev2@illinois.edu)

Problem 1: Diabetes Classification

We have been using the Pima Indians dataset to train a Naïve Bayes classifier to predict whether given patient has diabetes or not. The dataset contains 8 feature columns containing only continuous values and 1 label column having as values 1 (having diabetes) and 0 (negative diabetes results). There are total of 767 data points. We have used Normal (Gaussian) distribution for estimating the parameters. The classifier's final accuracy has been gathered by the average of 10 random splits the data having 80% for training and 20% for testing.

As a second part to the problem we examined what the effect of clearing the missing data (omitting feature values having 0) will be. For 4 features from vectors (attribute 3 (Diastolic blood pressure), attribute 4 (Triceps skinfold thickness), attribute 6 (Body mass index), and attribute 8 (Age)) we replaced the zeros with NaN and in the process of training, parameter prediction and predicting (we only skipped these features/column values, but not the entire row, to retain largest possible training set size).

Part 1 Accuracies

The accuracies rounded up to the second decimal digit are as follow:

Setup	Cross-validation Accuracy
Unprocessed data	75.68%
0-value elements ignored	74.57%

Part 1 Code Snippets

1. Calculation of distribution parameters
2. Calculation of naive Bayes predictions
   Please find the above information from the below codes (for the NOT 0-ignoring version – function `train_and_predict`, please refer to full code on end pages):

```
def train_and_predict_ignore_o(train_data, test_data):
    # train/build the model based on training set data.
    # Noted: the last column of both train_data and test_data are the label!
    # Calculate p(Positive) & p(negative)
    positive = train_data[train_data[:, 8] == 1]
    negative = train_data[train_data[:, 8] == 0]

    p_positive = 1.0*positive.shape[0]/train_data.shape[0]
    p_negative = 1 - p_positive

    # adjust attribute 3,4,6,8 to ignore the zero values
    positive[(positive[:, 2] == 0), 2] = np.nan  # filter out attribute 3
    positive[(positive[:, 3] == 0), 3] = np.nan  # filter out attribute 4
    positive[(positive[:, 5] == 0), 5] = np.nan  # filter out attribute 6
    positive[(positive[:, 7] == 0), 7] = np.nan  # filter out attribute 8

    # Calculate mean and variance for all features for positive samples
    positive_mean = np.nanmean(positive[:,:-1], axis=0)
    positive_var = np.nanvar(positive[:,:-1], axis=0)

    # adjust attribute 3,4,6,8 to ignore the zero values
    negative[(negative[:, 2] == 0), 2] = np.nan  # filter out attribute 3
    negative[(negative[:, 3] == 0), 3] = np.nan  # filter out attribute 4
    negative[(negative[:, 5] == 0), 5] = np.nan  # filter out attribute 6
    negative[(negative[:, 7] == 0), 7] = np.nan  # filter out attribute 8

    # Calculate mean and variance for all features for negative samples
    negative_mean = np.nanmean(negative[:,:-1], axis=0)
    negative_var = np.nanvar(negative[:,:-1], axis=0)

    # to predict on the test data set.
    test_X = np.array(test_data)[:, :-1]
    test_Y = np.array(test_data)[:, -1]

    p_x_positive = np.log(norm.pdf(test_X, positive_mean, np.sqrt(positive_var)))
    p_x_negative = np.log(norm.pdf(test_X, negative_mean, np.sqrt(negative_var)))

    # ignore the features with zero values for both classes
    for index, item in enumerate(test_X):
        if item[2] == 0:
            p_x_positive[index][2] = 0
            p_x_negative[index][2] = 0
        if item[3] == 0:
            p_x_positive[index][3] = 0
            p_x_negative[index][3] = 0
        if item[5] == 0:
            p_x_positive[index][5] = 0
            p_x_negative[index][5] = 0
        if item[7] == 0:
            p_x_positive[index][7] = 0
            p_x_negative[index][7] = 0

    positive_class = np.sum(p_x_positive, axis=1) + np.log(p_positive)
    negative_class = np.sum(p_x_negative, axis=1) + np.log(p_negative)
    predicted = (positive_class > negative_class)*1
    accuracy = sum(predicted == test_Y)/test_Y.shape[0]
    return accuracy*100
```

3. Test-train split code

```
split_idx = int(data.shape[0]*0.2)
total_accuracy = 0.0
total_iteration = 10
for i in range(total_iteration):
    np.random.shuffle(data)
    test_set = data[:split_idx, :]
    train_set = data[split_idx:, :]
    total_accuracy = total_accuracy + train_and_predict_ignore_o(train_set, test_set)
```

```
average_accuracy = total_accuracy/total_iteration
print("Averaged accuracy of cross validation is : ", average_accuracy)
```

Part 2 MNIST Accuracies

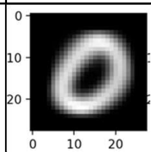
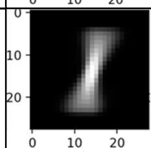
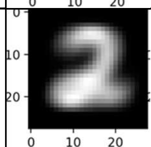
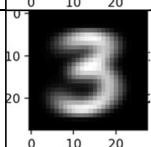
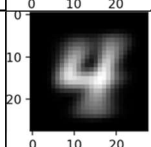
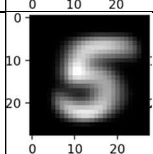
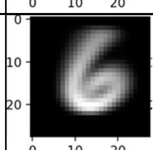
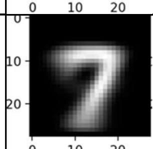
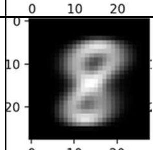
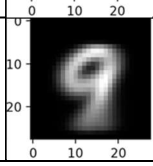
We used the MNIST dataset located at https://github.com/amplab/datascience-sp14/raw/master/lab7/mldata/mnist-original.mat to train Naïve Bayes classifier with Normal and Bernoulli distributions for estimating the parameters for the posterior probability. As comparison to this we also used the Random Forest classifier from the sklearn library to train and predict over the same dataset using different number of trees 10, 30 and different depth 4, 16 as additional parameters.

For each of the classifiers above we used untouched images (no cropping or resizing over the original image pixels) and bounded box stretched images (bounded and resized to 20x20 dimensions) as input. The results show that models using cleaned images have better accuracy for Gaussian NB and Random Forest (Bernoulli has similar results however). We also tested, if we disabled smoothing variable (epsilon=0 or <=1e-9) for Gaussian NB, the untouched images accuracy dropped significantly to around 59%, due to zero variance/mean (causing pdf of Normal Distribution to yield 0 values and log() yields –inf), but for bounded and stretched images accuracy is not affected. The following Gaussian NB accuracy was achieved by using smoothing val = 1e-1 (1e-1*<max variance> was introduced to avoid 0 variance pixels)

Results are as follows:

x	Method	Training Set Accuracy	Test Set Accuracy
1	Gaussian + untouched	79.47%	80.62%
2	Gaussian + stretched	83.16%	84.04%
3	Bernoulli + untouched	83.32%	84.27%
4	Bernoulli + stretched	81.55%	82.94%
5	10 trees + 4 depth + untouched	70.86%	69.7%
6	10 trees + 4 depth + stretched	71.06%	73.73%
7	10 trees + 16 depth + untouched	98.99%	93.7%
8	10 trees + 16 depth + stretched	99.52%	94.41%
9	30 trees + 4 depth + untouched	75.05%	72.72%
10	30 trees + 4 depth + stretched	74.38%	75.55%
11	30 trees + 16 depth + untouched	99.46%	95.52%
12	30 trees + 16 depth + stretched	99.74%	96.31%

Part 2A Digit Images

Digit	Mean Image
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Part 2 Code

- **Calculation of the Normal distribution parameters**
  The below function belongs to class of NaiveBayesNormalDistr. epsilon is like var_smoothing from the sklearn library GaussianNB. Please refer to the full code attached at the end.

```
def train(self, train_data, train_labels):
    self.digits = []
    self.p = []
    self.digits_mean = []
    self.digits_var = []
    for i in range(10):
        self.digits.append(train_data[train_labels[:] == i])
        self.p.append(1.0 * self.digits[i].shape[0] / train_data.shape[0])
        self.digits_mean.append(np.mean(self.digits[i], axis=0))
        self.digits_var.append(np.var(self.digits[i], axis=0))
    self.digits_var = np.array(self.digits_var)
    self.digits_var += self.epsilon * self.digits_var.max()
```

- **Calculation of the Bernoulli distribution parameters**
  The below function belongs to class of NaiveBayesBernoulli. We applied addictive smoothing (LAPLACE smoothing) with α = 1 (plus one). We mark ink pixels' value = 1, instead of 255 (refer to full code for threshold and assigning paper pixel =0 and ink pixel=1), so we can use sum to calculate p(ink|digit).

```
def train(self, train_data, train_labels):
    self.digits = []
    self.p = []
    self.digits_p_ink = []
    for i in range(10):
        self.digits.append(train_data[train_labels[:] == i])
        self.p.append(1.0 * self.digits[i].shape[0] / train_data.shape[0])
        p_ink = (np.sum(self.digits[i], axis=0) + 1) / (self.digits[i].shape[0] + train_data.shape[1])
        self.digits_p_ink.append(p_ink)
    self.digits_p_ink = np.array(self.digits_p_ink)
```

- **Calculation of the Naive Bayes predictions**

```
def predict(self, test_data):
    self.p_digit_class = []
    self.predicted = []
    if self.digits_mean == [] or self.digits_var ==[] or self.p == []:
        print("Fit your model to training data first")
        return []
    for i in range(10):
        normpdf = norm.pdf(test_data, self.digits_mean[i], np.sqrt(self.digits_var[i]))
        p_post = np.sum(np.log(normpdf), axis=1) + np.log(self.p[i])
        self.p_digit_class.append(p_post)
    self.p_digit_class = np.array(self.p_digit_class)
    self.predicted = np.argmax(self.p_digit_class, axis=0)
    return self.predicted
```

- **Training of a decision tree & Calculation of a decision tree predictions:**

```
def train_and_validate_randomforest(train_data, train_labels, test_data, test_label, n_trees, depth):
    clf = RandomForestClassifier(n_estimators=n_trees, max_depth=depth)
    clf.fit(train_data, train_labels)
    predicted = clf.predict(test_data)
    accuracy = sum(predicted == test_label)/test_label.shape[0]
    return accuracy
```

Please refer to the full code for details how to apply the above function on the training data and predict.

Libraries used & Reference:

David Forsyth's book - Probability and Statistics for Computer Science
David Forsyth's book - Applied Machine Learning
Trevor Walker's lecture and sample code – CS-498 Lecture videos and simple_nb.py
csv – for reading data from csv format: https://docs.python.org/3/library/csv.html
pima-indian-diabetes dataset - https://www.kaggle.com/kumargh/pimaindiansdiabetescsv
scipy.stats for normal and bernoulli's density and mass functions: https://docs.scipy.org/doc/scipy/reference/stats.html
Numpy - http://www.numpy.org/
matplotlib - to plot the MNIST images: https://matplotlib.org/
MNIST data set - http://yann.lecun.com/exdb/mnist/
mnist-python – to download and read the MNIST dataset: https://pypi.org/project/python-mnist/
opencv-python – to manipulate the MNIST images: https://pypi.org/project/opencv-python/
sklearn.ensemble – for random forest classifier : https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
Reference to smoothing variable of sklearn Naïve Bayes Gaussian - https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
& https://github.com/scikit-learn/scikit-learn/blob/7389dba/sklearn/naive_bayes.py
Reference to Addictive Smoothing (LAPLACE smoothing) - https://en.wikipedia.org/wiki/Additive_smoothing

Full Code of Problem 1

```
# -*- coding: utf-8 -*-
"""Apply Naive Bayes on Diabetes Classification

"""

import numpy as np
from scipy.stats import norm
import csv


# The following function will train & predict using Naive Bayes of Normal Distribution Model.
# This function will ignore the zero values on column 3,4,6,8 as missing data.
def train_and_predict_ignore_o(train_data, test_data):
    # train/build the model based on training set data.
    # Noted: the last column of both train_data and test_data are the label!

    # Calculate p(Positive) & p(negative)
    positive = train_data[train_data[:, 8] == 1]
    negative = train_data[train_data[:, 8] == 0]
```

```python
    p_positive = 1.0*positive.shape[0]/train_data.shape[0]
    p_negative = 1 - p_positive

    # adjust attribute 3,4,6,8 to ignore the zero values
    positive[(positive[:, 2] == 0), 2] = np.nan  # filter out attribute 3
    positive[(positive[:, 3] == 0), 3] = np.nan  # filter out attribute 4
    positive[(positive[:, 5] == 0), 5] = np.nan  # filter out attribute 6
    positive[(positive[:, 7] == 0), 7] = np.nan  # filter out attribute 8

    # Calculate mean and variance for all features for positive samples
    positive_mean = np.nanmean(positive[:,:-1], axis=0)
    positive_var = np.nanvar(positive[:,:-1], axis=0)

    # adjust attribute 3,4,6,8 to ignore the zero values
    negative[(negative[:, 2] == 0), 2] = np.nan  # filter out attribute 3
    negative[(negative[:, 3] == 0), 3] = np.nan  # filter out attribute 4
    negative[(negative[:, 5] == 0), 5] = np.nan  # filter out attribute 6
    negative[(negative[:, 7] == 0), 7] = np.nan  # filter out attribute 8

    # Calculate mean and variance for all features for negative samples
    negative_mean = np.nanmean(negative[:,:-1], axis=0)
    negative_var = np.nanvar(negative[:,:-1], axis=0)

    # to predict on the test data set.
    test_X = np.array(test_data)[:, :-1]
    test_Y = np.array(test_data)[:, -1]

    p_x_positive = np.log(norm.pdf(test_X, positive_mean, np.sqrt(positive_var)))
    p_x_negative = np.log(norm.pdf(test_X, negative_mean, np.sqrt(negative_var)))

    # ignore the features with zero values for both classes
    for index, item in enumerate(test_X):
        if item[2] == 0:
            p_x_positive[index][2] = 0
            p_x_negative[index][2] = 0
        if item[3] == 0:
            p_x_positive[index][3] = 0
            p_x_negative[index][3] = 0
        if item[5] == 0:
            p_x_positive[index][5] = 0
            p_x_negative[index][5] = 0
        if item[7] == 0:
            p_x_positive[index][7] = 0
            p_x_negative[index][7] = 0

    positive_class = np.sum(p_x_positive, axis=1) + np.log(p_positive)
    negative_class = np.sum(p_x_negative, axis=1) + np.log(p_negative)
    predicted = (positive_class > negative_class)*1
    accuracy = sum(predicted == test_Y)/test_Y.shape[0]
    return accuracy*100


# The following function will train & predict using Naive Bayes of Normal Distribution Model.
# This function will NOT ignore the zero values on column 3,4,6,8 as missing data.
def train_and_predict(train_data, test_data):
    # train/build the model based on training set data
    # Calculate p(Positive) & p(negative)
    positive = train_data[train_data[:, 8] == 1]
    negative = train_data[train_data[:, 8] == 0]

    p_positive = 1.0*positive.shape[0]/train_data.shape[0]
    p_negative = 1 - p_positive

    # Calculate mean and variance for all features for both positive and negative samples
    positive_mean = np.mean(positive, axis=0)[:-1]
    positive_var = np.var(positive, axis=0)[:-1]

    negative_mean = np.mean(negative, axis=0)[:-1]
    negative_var = np.var(negative, axis=0)[:-1]

    # to predict on the test data set.
    test_X = np.array(test_data)[:, :-1]
    test_Y = np.array(test_data)[:, -1]

    positive_class = np.sum(np.log(norm.pdf(test_X, positive_mean, np.sqrt(positive_var))), axis=1) +
np.log(p_positive)
    negative_class = np.sum(np.log(norm.pdf(test_X, negative_mean, np.sqrt(negative_var))), axis=1) +
np.log(p_negative)
    predicted = (positive_class > negative_class)*1
    accuracy = sum(predicted == test_Y)/test_Y.shape[0]
    return accuracy*100
```

```
data = []
X = []
Y = []
# import the data from the csv.
with open('pima-indians-diabetes.csv', newline='') as f:
    reader = csv.reader(f, delimiter=',', quoting=csv.QUOTE_NONNUMERIC)
    for row in reader:
        data.append(row)
    # remove the header from the data.
    data.pop(0)

data = np.array(data)
split_idx = int(data.shape[0]*0.2)

total_accuracy = 0.0
total_iteration = 10
for i in range(total_iteration):
    np.random.shuffle(data)
    test_set = data[:split_idx, :]
    train_set = data[split_idx:, :]
    total_accuracy = total_accuracy + train_and_predict_ignore_o(train_set, test_set)

average_accuracy = total_accuracy/total_iteration
print("Averaged accuracy of cross validation (ignoring zero values) is : ", average_accuracy, "%")

total_accuracy = 0.0
average_accuracy = 0.0
for i in range(total_iteration):
    np.random.shuffle(data)
    test_set = data[:split_idx, :]
    train_set = data[split_idx:, :]
    total_accuracy = total_accuracy + train_and_predict(train_set, test_set)

average_accuracy = total_accuracy/total_iteration
print("Averaged accuracy of cross validation is : ", average_accuracy, "%")
```

Full Code of Problem 2

```
# -*- coding: utf-8 -*-
"""Apply Naive Bayes (Gaussian & Bernoulli) and Random Forest on MNIST digits Classification

"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.stats import bernoulli
from sklearn.ensemble import RandomForestClassifier
from mnist import MNIST
import cv2

####################################################
#           Stretch images to 400 (20x20)          #
####################################################
def stretch_bounding_box(single_image_data):
    single_image_data = single_image_data.reshape((28, 28))
    vertical_min = np.nonzero(single_image_data)[0].min()
    vertical_max = np.nonzero(single_image_data)[0].max()
    horizon_min = np.nonzero(single_image_data)[1].min()
    horizon_max = np.nonzero(single_image_data)[1].max()
    return cv2.resize(single_image_data[vertical_min: vertical_max+1, horizon_min:horizon_max+1], (20,
20)).reshape(400)


####################################################
#          The Gaussian NB Classifier              #
```

```
#####################################################
class NaiveBayesNormalDistr:
    def __init__(self, epsilon=1e-9):
        self.epsilon = epsilon
        self.p_digit_class = []
        self.predicted = []
        self.digits = []
        self.digits_mean = []
        self.digits_var = []
        self.p = []

    def train(self, train_data, train_labels):
        self.digits = []
        self.p = []
        self.digits_mean = []
        self.digits_var = []
        # separate training data into different classes (digits)
        for i in range(10):
            self.digits.append(train_data[train_labels[:] == i])
            # Calculate p(digits 0,1,2,....9)
            self.p.append(1.0 * self.digits[i].shape[0] / train_data.shape[0])  # len(p) is 10
            # Calculate mean and variance for all features for all classes(digits)
            self.digits_mean.append(np.mean(self.digits[i], axis=0))  # each digits mean[i] shape is (784,1)
            self.digits_var.append(np.var(self.digits[i], axis=0))  # each digits_var[i] shape is (784,1)
        self.digits_var = np.array(self.digits_var)
        self.digits_var += self.epsilon * self.digits_var.max()

    def predict(self, test_data):
        self.p_digit_class = []
        self.predicted = []
        if self.digits_mean == [] or self.digits_var ==[] or self.p == []:
            print("Fit your model to training data first")
            return []

        for i in range(10):
            normpdf = norm.pdf(test_data, self.digits_mean[i], np.sqrt(self.digits_var[i]))
            p_post = np.sum(np.log(normpdf), axis=1) + np.log(self.p[i])
            self.p_digit_class.append(p_post)
        self.p_digit_class = np.array(self.p_digit_class)
        self.predicted = np.argmax(self.p_digit_class, axis=0)
        return self.predicted

    def get_accuracy(self, test_label):
        if len(self.predicted) == 0:
            print("Run predict() on your test data first")
            return 0
        elif len(self.predicted)!=len(test_label):
            print("Your test label shape mismatch the shape of your prediction data")
            return 0
        accuracy = sum(self.predicted == test_label) / len(test_label)
        return accuracy

    def plot_all_digits_mean(self):
        if len(self.digits_mean) > 0:
            # convert each digit mean from [10,784] to [10,28,28] (or [10,400] to [10,20,20])
            digit_mean_to_plot = np.array(self.digits_mean)
            image_size = int(np.sqrt(digit_mean_to_plot.shape[1]))
            digit_mean_to_plot =
digit_mean_to_plot.reshape((digit_mean_to_plot.shape[0],image_size,image_size))
            digit_mean_to_plot = (digit_mean_to_plot*255).astype(int)

            mainFigure = plt.figure(figsize=(10, 8))
            columns = 5
            rows = 2
            for i in range(1, columns * rows + 1):
                mainFigure.add_subplot(rows, columns, i)
                plt.imshow(digit_mean_to_plot[i-1], cmap='gray')
            plt.show()
        else:
            print("Train your model with data first.")


#####################################################
#          The Bernoulli NB Classifier              #
#####################################################
class NaiveBayesBernoulli:
    def __init__(self):
        self.predicted = []
        self.digits = []
        self.p = []
        self.p_digit_class = []
```

```python
        self.digits_p_ink = []

    def train(self, train_data, train_labels):
        self.digits = []
        self.p = []
        self.digits_p_ink = []
        # separate training data into different classes (digits)
        for i in range(10):
            self.digits.append(train_data[train_labels[:] == i])
            # Calculate p(digits 0,1,2,....9)
            self.p.append(1.0 * self.digits[i].shape[0] / train_data.shape[0])  # len(self.p) is 10
            # Count each ink pixels to calculate p(ink|C), using plus-one smoothing
            # Count of ink pixel +1 / Total images of such digit + number of ink pixels (dimension of row
sample)
            p_ink = (np.sum(self.digits[i], axis=0) + 1) / (self.digits[i].shape[0] + train_data.shape[1])
            self.digits_p_ink.append(p_ink)
        self.digits_p_ink = np.array(self.digits_p_ink)

    def predict(self, test_data):
        self.p_digit_class = []
        self.predicted = []
        if self.p == [] or self.digits_p_ink == []:
            print("Fit your model to training data first")
            return []

        for i in range(10):
            berpmf = bernoulli.pmf(test_data, self.digits_p_ink[i])
            p_post = np.sum(np.log(berpmf), axis=1) + np.log(self.p[i])
            self.p_digit_class.append(p_post)

        self.p_digit_class = np.array(self.p_digit_class)
        self.predicted = np.argmax(self.p_digit_class, axis=0)
        return self.predicted

    def get_accuracy(self, test_label):
        if len(self.predicted) == 0:
            print("Run predict() on your test data first")
            return 0
        elif len(self.predicted) != len(test_label):
            print("Your test label shape mismatch the shape of your prediction data")
            return 0
        accuracy = sum(self.predicted == test_label) / len(test_label)
        return accuracy


####################################################
#           The RandomForest Classifier            #
####################################################
def train_and_validate_randomforest(train_data, train_labels, test_data, test_label, n_trees, depth):
    clf = RandomForestClassifier(n_estimators=n_trees, max_depth=depth)
    clf.fit(train_data, train_labels)
    predicted = clf.predict(test_data)
    accuracy = sum(predicted == test_label)/test_label.shape[0]
    return accuracy


mndata = MNIST('./MNIST')
mndata.gz = True
images, labels = mndata.load_training()
test_images, test_labels = mndata.load_testing()

# filter out the mid grey pixels and convert it into binary picture
ink_threshold = 255*0.5
images = np.array(images, dtype='uint8')
images[images[:] < ink_threshold] = 0
images[images[:] >= ink_threshold] = 1  # mark it as ink pixel
labels = np.array(labels, dtype='uint8')

test_images = np.array(test_images, dtype='uint8')
test_images[test_images[:] < ink_threshold] = 0
test_images[test_images[:] >= ink_threshold] = 1  # mark it as ink pixel
test_labels = np.array(test_labels, dtype='uint8')


# produce the stretched images for train and test set
stretched_image_map = map(stretch_bounding_box, images)
stretched_image = np.array(list(stretched_image_map))

stretched_test_image_map = map(stretch_bounding_box, test_images)
stretched_test_image = np.array(list(stretched_test_image_map))
```

```
####################################################
#    The following predict over TEST data          #
####################################################
# use Naive Bayes Normal D to train and predict on untouched test images:
nb_normal = NaiveBayesNormalDistr(1e-1)
nb_normal.train(images, labels)
_ = nb_normal.predict(test_images)
print("Naive Bayes - normal distribution accuracy on untouched test data: ",
nb_normal.get_accuracy(test_labels))
# to plot the digits mean for all 10 digits.
nb_normal.plot_all_digits_mean()

# use Naive Bayes Normal D to train and predict on stretched test images:
nb_normal_stretched = NaiveBayesNormalDistr(1e-1)
nb_normal_stretched.train(stretched_image, labels)
_ = nb_normal_stretched.predict(stretched_test_image)
print("Naive Bayes - normal distribution accuracy on stretched test data: ",
nb_normal_stretched.get_accuracy(test_labels))

# use Naive Bayes Bernoulli to train and predict on untouched test images:
nb_bernoulli = NaiveBayesBernoulli()
nb_bernoulli.train(images, labels)
_ = nb_bernoulli.predict(test_images)
print("Naive Bayes - bernoulli accuracy on untouched test data: ", nb_bernoulli.get_accuracy(test_labels))
# use Naive Bayes Bernoulli to train and predict on stretched test images:
nb_bernoulli_stretched = NaiveBayesBernoulli()
nb_bernoulli_stretched.train(stretched_image, labels)
_ = nb_bernoulli_stretched.predict(stretched_test_image)
print("Naive Bayes - bernoulli accuracy on stretched test data: ",
nb_bernoulli_stretched.get_accuracy(test_labels))

# RANDOM FOREST - UNTOUCHED TEST DATA
# use Random forest with setting of trees = 10 and depth = 4
print("Untouched test data - Random Forest (10 Trees, 4 Depth)", train_and_validate_randomforest(images,
labels, test_images, test_labels, 10, 4))
# use Random forest with setting of trees = 10 and depth = 16
print("Untouched test data - Random Forest (10 Trees, 16 Depth)", train_and_validate_randomforest(images,
labels, test_images, test_labels, 10, 16))
# use Random forest with setting of trees = 30 and depth = 4
print("Untouched test data - Random Forest (30 Trees, 4 Depth)", train_and_validate_randomforest(images,
labels, test_images, test_labels, 30, 4))
# use Random forest with setting of trees = 30 and depth = 16
print("Untouched test data - Random Forest (30 Trees, 16 Depth)", train_and_validate_randomforest(images,
labels, test_images, test_labels, 30, 16))

# RANDOM FOREST - STRETCHED TEST DATA
# use Random forest with setting of trees = 10 and depth = 4
print("Stretched test data - Random Forest (10 Trees, 4 Depth)",
train_and_validate_randomforest(stretched_image, labels, stretched_test_image, test_labels, 10, 4))
# use Random forest with setting of trees = 10 and depth = 16
print("Stretched test data - Random Forest (10 Trees, 16 Depth)",
train_and_validate_randomforest(stretched_image, labels, stretched_test_image, test_labels, 10, 16))
# use Random forest with setting of trees = 30 and depth = 4
print("Stretched test data - Random Forest (30 Trees, 4 Depth)",
train_and_validate_randomforest(stretched_image, labels, stretched_test_image, test_labels, 30, 4))
# use Random forest with setting of trees = 30 and depth = 16
print("Stretched test data - Random Forest (30 Trees, 16 Depth)",
train_and_validate_randomforest(stretched_image, labels, stretched_test_image, test_labels, 30, 16))

####################################################
#    The following predict over TRAIN data         #
####################################################
# use Naive Bayes Normal D to train and predict on untouched test images:
nb_normal = NaiveBayesNormalDistr(1e-1)
nb_normal.train(images, labels)
_ = nb_normal.predict(images)
print("Naive Bayes - normal distribution accuracy on untouched training data: ",
nb_normal.get_accuracy(labels))
# to plot the digits mean for all 10 digits.
nb_normal.plot_all_digits_mean()

# use Naive Bayes Normal D to train and predict on stretched test images:
nb_normal_stretched = NaiveBayesNormalDistr(1e-1)
nb_normal_stretched.train(stretched_image, labels)
_ = nb_normal_stretched.predict(stretched_image)
print("Naive Bayes - normal distribution accuracy on stretched training data: ",
nb_normal_stretched.get_accuracy(labels))

# use Naive Bayes Bernoulli to train and predict on untouched test images:
nb_bernoulli = NaiveBayesBernoulli()
```

```
nb_bernoulli.train(images, labels)
_ = nb_bernoulli.predict(images)
print("Naive Bayes - bernoulli accuracy on untouched training data: ", nb_bernoulli.get_accuracy(labels))
# use Naive Bayes Bernoulli to train and predict on stretched test images:
nb_bernoulli_stretched = NaiveBayesBernoulli()
nb_bernoulli_stretched.train(stretched_image, labels)
_ = nb_bernoulli_stretched.predict(stretched_image)
print("Naive Bayes - bernoulli accuracy on stretched training data: ",
nb_bernoulli_stretched.get_accuracy(labels))

# RANDOM FOREST - UNTOUCHED TRAIN DATA
# use Random forest with setting of trees = 10 and depth = 4
print("Untouched Training data - Random Forest (10 Trees, 4 Depth)", train_and_validate_randomforest(images,
labels, images, labels, 10, 4))
# use Random forest with setting of trees = 10 and depth = 16
print("Untouched Training data - Random Forest (10 Trees, 16 Depth)", train_and_validate_randomforest(images,
labels, images, labels, 10, 16))
# use Random forest with setting of trees = 30 and depth = 4
print("Untouched Training data - Random Forest (30 Trees, 4 Depth)", train_and_validate_randomforest(images,
labels, images, labels, 30, 4))
# use Random forest with setting of trees = 30 and depth = 16
print("Untouched Training data - Random Forest (30 Trees, 16 Depth)", train_and_validate_randomforest(images,
labels, images, labels, 30, 16))

# RANDOM FOREST - STRETCHED TRAIN DATA
# use Random forest with setting of trees = 10 and depth = 4
print("Stretched Training data - Random Forest (10 Trees, 4 Depth)",
train_and_validate_randomforest(stretched_image, labels, stretched_image, labels, 10, 4))
# use Random forest with setting of trees = 10 and depth = 16
print("Stretched Training data - Random Forest (10 Trees, 16 Depth)",
train_and_validate_randomforest(stretched_image, labels, stretched_image, labels, 10, 16))
# use Random forest with setting of trees = 30 and depth = 4
print("Stretched Training data - Random Forest (30 Trees, 4 Depth)",
train_and_validate_randomforest(stretched_image, labels, stretched_image, labels, 30, 4))
# use Random forest with setting of trees = 30 and depth = 16
print("Stretched Training data - Random Forest (30 Trees, 16 Depth)",
train_and_validate_randomforest(stretched_image, labels, stretched_image, labels, 30, 16))
```