Part 1 Accuracies

Setup	Cross-validation
Unprocessed data	77.60%
o-value elements ignored	73.05%

Part 1 Code Snippets

1. Calculation of distribution parameters

```
def mean(numbers):
    return sum(filter(None, numbers)) / float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
    return math.sqrt(variance)

def getMeanandStddev(dataset):
    # Seperate the dataset
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    # Obtain the mean and standard deviation for each feature
    meanAndStddevGroup = {}
    for classValue, instances in separated.items():
        summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*instances)]
        del summaries[-1]
        meanAndStddevGroup[classValue] = summaries
    return meanAndStddevGroup
```

2. Calculation of Naive Bayes predictions

```
def calculateProbability(x, mean, stdev):
    # using Standard normal distribution
    exponent = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent

def calculateClassProbabilities(summaries, inputVector, classProb):
    probabilities = {}
    for classValue, classSummaries in summaries.items():
        probabilities[classValue] = math.log10(classProb[classValue])
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            if x is not None:
                probabilities[classValue] += math.log10(calculateProbability(x, mean, stdev))
    return probabilities

def getPredictions(summaries, testSet):
    # SGet probabilistic of each class in test data
    classProb = {}
    for i in range(len(testSet)):
        vector = testSet[i]
        if (vector[-1] not in classProb):
            classProb[vector[-1]] = 1
        else:
            classProb[vector[-1]] += 1
    classProb = {k: v / len(testSet) for k, v in classProb.items()}
    #
    predictions = []
    for i in range(len(testSet)):
        probabilities = calculateClassProbabilities(summaries, testSet[i], classProb)
        bestLabel, bestProb = None, -1
        for classValue, probability in probabilities.items():
            if bestLabel is None or probability > bestProb:
                bestProb = probability
                bestLabel = classValue
        predictions.append(bestLabel)
    return predictions
```

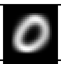
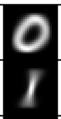
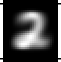
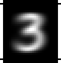
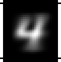
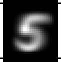
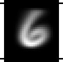
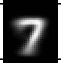
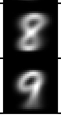
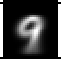
3. Test-train split code

```
def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    copy = list(dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy))
        trainSet.append(copy.pop(index))
    return [trainSet, copy]
```

Part 2 MNIST Accuracies

x	Method	Training Set Accuracy	Test Set Accuracy
1	Gaussian + untouched	80.10%	81.39%
2	Gaussian + stretched	82.74%	83.85%
3	Bernoulli + untouched	83.44%	84.47%
4	Bernoulli + stretched	46.71%	49.12%
5	10 trees + 4 depth + untouched	75.18%	75.66%
6	10 trees + 4 depth + stretched	71.38%	72.3%
7	10 trees + 16 depth + untouched	99.53%	94.44%
8	10 trees + 16 depth + stretched	99.51%	94.69%
9	30 trees + 4 depth + untouched	79.09%	79.73%
10	30 trees + 4 depth + stretched	75.15%	76.59%
11	30 trees + 16 depth + untouched	99.76%	96.09%
12	30 trees + 16 depth + stretched	99.76%	95.96%

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**

```
def fit(self, X, Y, smoothing=1e-2):
   self.gaussians = dict()
   self.priors = dict()
   labels = set(Y)
   for c in labels:
      current_x = X[Y == c]
      self.gaussians[c] = {
         'mean': current_x.mean(axis=0),
         'var': current_x.var(axis=0) + smoothing,
      }
      self.priors[c] = float(len(Y[Y == c])) / len(Y)
```

- **Calculation of the Bernoulli distribution parameters**

```
def fit(self, X, Y):
   self.bernoulli = dict()
   self.priors = dict()
   labels = set(Y)
   for c in labels:
      current_x = X[Y == c]
      self.bernoulli[c] = (np.array(current_x.sum(axis=0)) + self.alpha) / (current_x.shape[0] + 2 * self.alpha)
      self.priors[c] = float(len(Y[Y == c])) / len(Y)
   return self
```

- **Calculation of the Naïve Bayes prediction**

  For Normal distribution prediction:

```
def predict(self, X):
      N, D = X.shape
      K = len(self.gaussians)
      P = np.zeros((N, K))
      for c, g in iteritems(self.gaussians):
         mean, var = g['mean'], g['var']
         P[:,c] = mvn.logpdf(X, mean=mean, cov=var) + np.log(self.priors[c])
      return np.argmax(P, axis=1)
```

  For Bernoulli distribution prediction:

```
def predict(self, X):
   N, D = X.shape
   K = len(self.bernoulli)
   P = np.zeros((N, K))
   for c, g in iteritems(self.bernoulli):
      P[:, c] = np.sum(X * np.log(g) + np.abs(X - 1) * np.log(1 - g), axis=1) + np.log(self.priors[c])
   return np.argmax(P, axis=1)
```

- **Training of a decision tree**

```
random_forest_model = RandomForestClassifier(max_depth=d, random_state=0, n_estimators=n)
random_forest_model.fit(train_x1, train_y1)
```

- **Calculation of a decision tree prediction**

```
preiction = random_forest_model.predict(train_x1)
acc = np.mean(preiction == train_y1)
print("Accuracy achieved for Train data " + case + " is " + str(acc * 100) + "%")
preiction = random_forest_model.predict(test_x1)
acc = np.mean(preiction == test_y1)
print("Accuracy achieved for Test data " + case + " is " + str(acc * 100) + "%")
```

All codes

1. **Problem 1 codes**

```
import csv

import random

import math


def loadCsv(filename):

    lines = csv.reader(open(filename, "r"))

    dataset = list(lines)

    for i in range(len(dataset)):

        temp = [float(x) for x in dataset[i]]

        # values = [None if x == 0 else x for x in temp[:-1]]

        # values.append(temp[-1])

        # dataset[i] = values

        dataset[i] = temp

    return dataset


def splitDataset(dataset, splitRatio):

    trainSize = int(len(dataset) * splitRatio)

    trainSet = []

    copy = list(dataset)

    while len(trainSet) < trainSize:

        index = random.randrange(len(copy))

        trainSet.append(copy.pop(index))

    return [trainSet, copy]


def mean(numbers):

    numbersNotNone = numbers

    # numbersNotNone = [x for x in numbers if x is not None]

    # if len(numbersNotNone) == 0:

    #     return 0
    return sum(filter(None, numbersNotNone)) / float(len(numbersNotNone))


def stdev(numbers):
    numbersNotNone = numbers
    # numbersNotNone = [x for x in numbers if x is not None]
    # if len(numbersNotNone) == 0:
    #     return 0
    avg = mean(numbersNotNone)
    variance = sum([pow(x - avg, 2) for x in numbersNotNone]) / float(len(numbersNotNone) - 1)
    return math.sqrt(variance)


def getMeanandStddev(dataset):
    # Seperate the dataset
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    # Obtain the mean and standard deviation for each feature
    meanAndStddevGroup = {}
    for classValue, instances in separated.items():
        summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*instances)]
        del summaries[-1]
        meanAndStddevGroup[classValue] = summaries
    return meanAndStddevGroup


def calculateProbability(x, mean, stdev):
    # using Standard normal distribution
    exponent = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent
```

```
def calculateClassProbabilities(summaries, inputVector, classProb):
    probabilities = {}
    for classValue, classSummaries in summaries.items():
        probabilities[classValue] = math.log10(classProb[classValue])
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            if x is not None:
                probabilities[classValue] += math.log10(calculateProbability(x, mean, stdev))
    return probabilities


def getPredictions(summaries, testSet):
    # SGet probabilistic of each class in test data
    classProb = {}
    for i in range(len(testSet)):
        vector = testSet[i]
        if (vector[-1] not in classProb):
            classProb[vector[-1]] = 1
        else:
            classProb[vector[-1]] += 1
    classProb = {k: v / len(testSet) for k, v in classProb.items()}
    #
    predictions = []
    for i in range(len(testSet)):
        probabilities = calculateClassProbabilities(summaries, testSet[i], classProb)
        bestLabel, bestProb = None, -1
        for classValue, probability in probabilities.items():
            if bestLabel is None or probability > bestProb:
                bestProb = probability
                bestLabel = classValue
        predictions.append(bestLabel)
    return predictions


def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(testSet))) * 100.0


def main():
    filename = 'pima-indians-diabetes.csv'
    accuracy = 0
    for i in range(10):
        # Split dataset
        splitRatio = 0.8
        dataset = loadCsv(filename)
        trainingSet, testSet = splitDataset(dataset, splitRatio)
        # prepare model
        summaries = getMeanandStddev(trainingSet)
        # test model
        predictions = getPredictions(summaries, testSet)
        accuracy += getAccuracy(testSet, predictions)
    accuracy /= 10
    print("Accuracy: {}%".format(accuracy))
```

2. Problem 2 codes

```
from __future__ import print_function, division

from future.utils import iteritems

from builtins import range, input

import matplotlib.pyplot as plt

from datetime import datetime

import numpy as np

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data

import scipy

from scipy.stats import norm

from scipy.stats import multivariate_normal as mvn

from scipy.misc import imresize

from PIL import Image

import cv2

from sklearn.ensemble import RandomForestClassifier


def get_data():

    # Use the TensorFlow method to download and/or load the data.

    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

    X_train = mnist.train.images

    Y_train = np.argmax(mnist.train.labels,axis=1)

    X_test = mnist.test.images

    Y_test = np.argmax(mnist.test.labels,axis=1)

    return [X_train, Y_train, X_test, Y_test]


def strechData(Xtrain, Ytrain, Xtest, Ytest):

    train_modified = np.apply_along_axis(rescale_strech_image, axis=1, arr=Xtrain)

    test_modified = np.apply_along_axis(rescale_strech_image, axis=1, arr=Xtest)


    Xtrain_streched = np.reshape(train_modified, (train_modified.shape[0], 400))
    Xtest_streched = np.reshape(test_modified, (test_modified.shape[0], 400))


    Ytrain_final = Ytrain.astype(np.uint8)
    Ytest_final = Ytest.astype(np.uint8)


    return [Xtrain_streched, Ytrain_final, Xtest_streched, Ytest_final]


class NaiveBayes(object):
    def fit(self, X, Y, smoothing=1e-2):
        self.gaussians = dict()
        self.priors = dict()
        labels = set(Y)
        for c in labels:
            current_x = X[Y == c]
            self.gaussians[c] = {
                'mean': current_x.mean(axis=0),
                'var': current_x.var(axis=0) + smoothing,
            }
            self.priors[c] = float(len(Y[Y == c])) / len(Y)


    def plotMean(self):
        for c, g in iteritems(self.gaussians):
            mean = g['mean']
            pixels = np.array(mean * 255, dtype='uint8')
            pixels = np.reshape(pixels, (28, 28))
            plt.title('Digit is {}'.format(str(c)))
            # plt.imshow(pixels, cmap='gray')
            plt.imsave("Digit {}".format(str(c)), pixels, cmap='gray')


    def score(self, X, Y):
        P = self.predict(X)
        return np.mean(P == Y)


    def predict(self, X):
        N, D = X.shape
        K = len(self.gaussians)
        P = np.zeros((N, K))
        for c, g in iteritems(self.gaussians):
            mean, var = g['mean'], g['var']
            P[:,c] = mvn.logpdf(X, mean=mean, cov=var) + np.log(self.priors[c])
        return np.argmax(P, axis=1)


class Bernoulli(object):
    def __init__(self, alpha=1e-2):
        self.alpha = alpha


    def fit(self, X, Y):
        self.bernoulli = dict()
        self.priors = dict()
        labels = set(Y)
        for c in labels:
            current_x = X[Y == c]
            self.bernoulli[c] = (np.array(current_x.sum(axis=0)) + self.alpha) / (current_x.shape[0] + 2 * self.alpha)
            self.priors[c] = float(len(Y[Y == c])) / len(Y)


        return self


    def predict(self, X):
        N, D = X.shape
        K = len(self.bernoulli)
        P = np.zeros((N, K))
        for c, g in iteritems(self.bernoulli):
            P[:, c] = np.sum(X * np.log(g) + np.abs(X - 1) * np.log(1 - g), axis=1) + np.log(self.priors[c])
        return np.argmax(P, axis=1)

    def score(self, X, Y):
        P = self.predict(X)
        return np.mean(P == Y)

def train_test_accuracy(model, Xtrain, Ytrain, Xtest, Ytest, description):
    t0 = datetime.now()
    model.fit(Xtrain, Ytrain)
    print("{}, Training time: {}.".format(description, (datetime.now() - t0)))

    t0 = datetime.now()
    print("{}, Train accuracy: {}".format(description, model.score(Xtrain, Ytrain)))
    print("{}, Time to compute train accuracy: {}, Train size: {}".format(description, (datetime.now() - t0), len(Ytrain)))

    t0 = datetime.now()
    print("{}, Test accuracy: {}".format(description, model.score(Xtest, Ytest)))
    print("{}, Time to compute test accuracy: {}, Test size: {}.".format(description, (datetime.now() - t0), len(Ytest)))

    if (description == "Untouched_Gausian"):
        model.plotMean()

def rescale_strech_image(image):
    image = np.where(image * 255 > 128, 1, 0)
    img = np.reshape(image, (28, 28))
    row= np.unique(np.nonzero(img)[0])
    col = np.unique(np.nonzero(img)[1])
    image_data_new = img[min(row): max(row) + 1, min(col):max(col) + 1]
    image_data_new = image_data_new.astype('uint8')
    image_data_new = cv2.resize(image_data_new, (20, 20))
    return (np.array(image_data_new).astype(np.uint8))


def do_random_forest(d, n, train_x1, train_y1, test_x1, test_y1, case):
    random_forest_model = RandomForestClassifier(max_depth=d, random_state=0, n_estimators=n)
    random_forest_model.fit(train_x1, train_y1)
    preiction = random_forest_model.predict(train_x1)
    acc = np.mean(preiction == train_y1)
    print("Accuracy achieved for Train data " + case + " is " + str(acc * 100) + "%")
    preiction = random_forest_model.predict(test_x1)
    acc = np.mean(preiction == test_y1)
    print("Accuracy achieved for Test data " + case + " is " + str(acc * 100) + "%")


if __name__ == '__main__':
    Xtrain, Ytrain, Xtest, Ytest = get_data()
    Xtrain_streched, Ytrain_final, Xtest_streched, Ytest_final = strechData(Xtrain, Ytrain, Xtest, Ytest)

    # # Prob 2A
    # untouched NaiveBayes
    model = NaiveBayes()
    train_test_accuracy(model, Xtrain, Ytrain, Xtest, Ytest, "Untouched_Gausian")

    # streched NaiveBayes
    model = NaiveBayes()
    train_test_accuracy(model, Xtrain_streched, Ytrain_final, Xtest_streched, Ytest_final, "Streched_Gausian")

    # untouched Bernoulli
    model = Bernoulli()
    train_test_accuracy(model, Xtrain, Ytrain, Xtest, Ytest, "Untouched Bernoulli")
    # streched Bernoulli
    model = Bernoulli()
    train_test_accuracy(model, Xtrain_streched, Ytrain_final, Xtest_streched, Ytest_final, "Streched Bernoulli")

    # #Prob 2B
    do_random_forest(4, 10, Xtrain, Ytrain, Xtest, Ytest, "4/10 untouched")
    do_random_forest(4, 10, Xtrain_streched, Ytrain_final, Xtest_streched, Ytest_final, "4/10 bounded and stretched")

    do_random_forest(16, 10, Xtrain, Ytrain, Xtest, Ytest, "16/10 untouched")
    do_random_forest(16, 10, Xtrain_streched, Ytrain_final, Xtest_streched, Ytest_final, "16/10 bounded and stretched")

    do_random_forest(4, 30, Xtrain, Ytrain, Xtest, Ytest, "4/30 untouched")
    do_random_forest(4, 30, Xtrain_streched, Ytrain_final, Xtest_streched, Ytest_final, "4/30 bounded and stretched")

    do_random_forest(16, 30, Xtrain, Ytrain, Xtest, Ytest, "16/30 untouched")
    do_random_forest(16, 30, Xtrain_streched, Ytrain_final, Xtest_streched, Ytest_final, "16/30 bounded and stretched")
```