Part 1 Accuracies

SetUp	Cross-Validation Accuracy
Unprocessed Data	74.02597402597402%
0-value elements ignored	78.70370370370371%

```
Split 768 rows into train = 614 and test = 154 rows
Average Accuracy of Unprocessed data: 74.02597402597402%
Split 539 rows into train = 431 and test = 108 rows
Average Accuracy of processed data with 0 values ignored: 78.70370370370371%
```

Part 1 Code Snippets

1. Calculation of distribution Parameters

```
def mean(numbers):
  return sum(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 summarize(dataset):
  summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
  del summaries[-1]
  return summaries

def summarizeByClass(dataset):
  separated = separateByClass(dataset)
  summaries = {}
  for classValue, instances in separated.items():
    summaries[classValue] = summarize(instances)
  return summaries
```

2. Calculation of naive Bayes predictions

```
def calculateProbability(x, mean, stdev):
  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):
  probabilities = {}
  for classValue, classSummaries in summaries.items():
    probabilities[classValue] = 1
    for i in range(len(classSummaries)):
      mean, stdev = classSummaries[i]
      x = inputVector[i]
      probabilities[classValue] *= calculateProbability(x, mean, stdev)
  return probabilities

def predict(summaries, inputVector):
  probabilities = calculateClassProbabilities(summaries, inputVector)
  bestLabel, bestProb = None, -1
  for classValue, probability in probabilities.items():
    if bestLabel is None or probability > bestProb:
      bestProb = probability
      bestLabel = classValue
  return bestLabel

def getPredictions(summaries, testSet):
  predictions = []
  for i in range(len(testSet)):
    result = predict(summaries, testSet[i])
    predictions.append(result)
  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
```

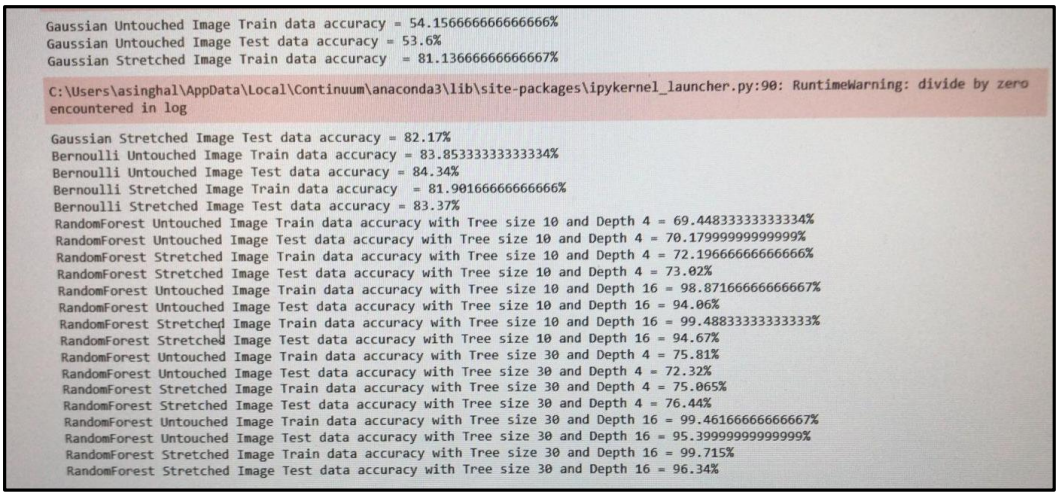
3. Test-train split code

```
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]
```

Entire Code for Part 1:

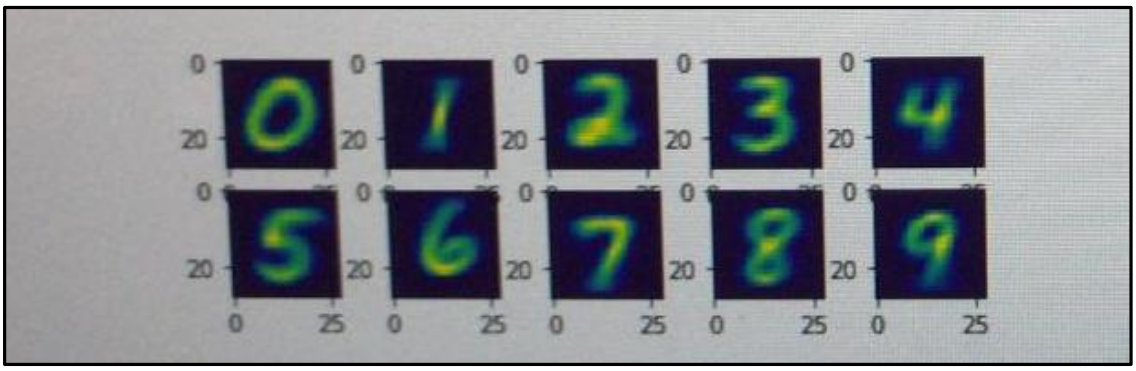
Part 2 MNIST Accuracies

X	Method	Training Set Accuracy	Test Set Accuracy
1	Gaussian + untouched	54.156666%	53.6%
2	Gaussian + stretched	81.13666%	82.136666%
3	Bernoulli + untouched	83.85333%	84.34%
4	Bernoulli + stretched	81.90166%	83.37%
5	10 trees + 4 depth + untouched	69.44833%	70.17999%
6	10 trees + 4 depth + stretched	72.1966%	73.02%
7	10 trees + 16 depth + untouched	98.87166%	94.06%
8	10 trees + 16 depth + stretched	99.48833%	94.67%
9	30 trees + 4 depth + untouched	75.81%	72.32%
10	30 trees + 4 depth + stretched	75.065%	76.44%
11	30 trees + 16 depth + untouched	99.46166%	95.399%
12	30 trees + 16 depth + stretched	99.715%	96.34%



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
- Calculation of the Bernoulli distribution parameters

```
def BernoulliNB(X_train, y_train, X_test, y_test, stretchedFlag):

  nclass = np.unique(y_train).shape[0]
  class_labels = np.unique(y_train)
  nfeature = X_train.shape[1]
  priors= []
  bpoint = np.zeros((nclass,nfeature))
  for i, val in enumerate(class_labels):
    sep = [y_train == val]#separated
    priors.append((np.sum(sep)) / len(y_train))
    bpoint[i]  = (np.mean(X_train[tuple(sep)], axis=0))/255

  if stretchedFlag == 1:
    PredictionsTrain = getBernoulliPredictions(X_train, y_train,bpoint,priors,nclass,class_labels)
    accuracyTrain = getAccuracy (y_train,PredictionsTrain)
    print("Bernoulli Stretched Image Train data accuracy  = {0}%".format(accuracyTrain))
    PredictionsTest = getBernoulliPredictions(X_test,y_test,bpoint,priors,nclass,class_labels)
    accuracyTest = getAccuracy (y_test,PredictionsTest)
    print("Bernoulli Stretched Image Test data accuracy = {0}%".format(accuracyTest))
  else:
    PredictionsTrain = getBernoulliPredictions(X_train, y_train,bpoint,priors,nclass,class_labels)
    accuracyTrain = getAccuracy (y_train,PredictionsTrain)
    print("Bernoulli Untouched Image Train data accuracy = {0}%".format(accuracyTrain))
    PredictionsTest = getBernoulliPredictions(X_test,y_test,bpoint,priors,nclass,class_labels)
    accuracyTest = getAccuracy (y_test,PredictionsTest)
    print("Bernoulli Untouched Image Test data accuracy = {0}%".format(accuracyTest))
```

- Calculation of the Naive Bayes predictions
- Training of a decision tree

```
def RandomForestNB(X_train, y_train, X_test, y_test, X_trainTouched, X_testTouched):

  RandomForest(X_train, y_train, X_train, y_train, 10, 4, train=1, stretchedFlag=0)
  RandomForest(X_train, y_train, X_test, y_test, 10, 4, train=0, stretchedFlag=0)
  RandomForest(X_trainTouched, y_train, X_trainTouched, y_train, 10, 4, train=1, stretchedFlag=1)
  RandomForest(X_trainTouched, y_train, X_testTouched, y_test, 10, 4, train=0, stretchedFlag=1)

  RandomForest(X_train, y_train, X_train, y_train, 10, 16, train=1, stretchedFlag=0)
  RandomForest(X_train, y_train, X_test, y_test, 10, 16, train=0, stretchedFlag=0)
  RandomForest(X_trainTouched, y_train, X_trainTouched, y_train, 10, 16, train=1, stretchedFlag=1)
  RandomForest(X_trainTouched, y_train, X_testTouched, y_test, 10, 16, train=0, stretchedFlag=1)

  RandomForest(X_train, y_train, X_train, y_train, 30, 4, train=1, stretchedFlag=0)
  RandomForest(X_train, y_train, X_test, y_test, 30, 4, train=0, stretchedFlag=0)
  RandomForest(X_trainTouched, y_train, X_trainTouched, y_train, 30, 4, train=1, stretchedFlag=1)
  RandomForest(X_trainTouched, y_train, X_testTouched, y_test, 30, 4, train=0, stretchedFlag=1)

  RandomForest(X_train, y_train, X_train, y_train, 30, 16, train=1, stretchedFlag=0)
  RandomForest(X_train, y_train, X_test, y_test, 30, 16, train=0, stretchedFlag=0)
  RandomForest(X_trainTouched, y_train, X_trainTouched, y_train, 30, 16, train=1, stretchedFlag=1)
  RandomForest(X_trainTouched, y_train, X_testTouched, y_test, 30, 16, train=0, stretchedFlag=1)
```

- Calculation of a decision tree predictions

```
def RandomForest(X_train, y_train, X_test, y_test, estimators, depth, train, stretchedFlag):

  model = RandomForestClassifier(n_estimators=estimators, max_depth=depth)
  model.fit(X_train, y_train)
  Predictions = model.predict(X_test)
  accuracy = getAccuracy (y_test,Predictions)
  if train ==1:
    if stretchedFlag ==0:
      print("RandomForest Untouched Image Train data accuracy with Tree size {0} and Depth {1} = {2}%".format(estimators,depth,accuracy))
    else:
      print("RandomForest Stretched Image Train data accuracy with Tree size {0} and Depth {1} = {2}%".format(estimators,depth,accuracy))
  else:
    if stretchedFlag ==0:
      print("RandomForest Untouched Image Test data accuracy with Tree size {0} and Depth {1} = {2}%".format(estimators,depth,accuracy))
    else:
      print("RandomForest Stretched Image Test data accuracy with Tree size {0} and Depth {1} = {2}%".format(estimators,depth,accuracy))
```

Entire Part 2 Code

FULL CODE PART 1

```
# Example of Naive Bayes implemented from Scratch in Python
import csv
import random
import math

def loadCsv1(filename):
	lines = csv.reader(open(filename, "r"))
	dataset = list(lines)
	for i in range(len(dataset)):
		dataset[i] = [float(x) for x in dataset[i]]
	return dataset

def loadCsv2(filename):
	lines = csv.reader(open(filename, "r"))
	dataset1 = list(lines)
	dataset=[]
	length=len(dataset1)
	for row in dataset1:
		if "0" not in [row[2], row[3], row[5], row[7]]:
			dataset.append([float(x) for x in row])
	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 separateByClass(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)
	return separated

def mean(numbers):
	return sum(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 summarize(dataset):
	summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
	del summaries[-1]
	return summaries

def summarizeByClass(dataset):
	separated = separateByClass(dataset)
	summaries = {}
	for classValue, instances in separated.items():
		summaries[classValue] = summarize(instances)
	return summaries

def calculateProbability(x, mean, stdev):
```

```
    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):
    probabilities = {}
    for classValue, classSummaries in summaries.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            probabilities[classValue] *= calculateProbability(x, mean, stdev)
    return probabilities


def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel


def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    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'
        splitRatio = 0.80

        #unprocessed dataset load
        dataset = loadCsv1(filename)
        #print(dataset)
        accuracySum = 0
        for i in range(10):
                trainingSet, testSet = splitDataset(dataset, splitRatio)
                #print('Split {0} rows into train = {1} and test = {2}
rows'.format(len(dataset),len(trainingSet),len(testSet)))
                # prepare model
                summaries = summarizeByClass(trainingSet)
                # test model
                predictions = getPredictions(summaries, testSet)
                accuracy = getAccuracy(testSet, predictions)
                #print('Accuracy: {0}%'.format(accuracy))
                accuracySum = accuracySum + accuracy
        accuracySum = accuracySum/10
        print('Split {0} rows into train = {1} and test = {2}
rows'.format(len(dataset),len(trainingSet),len(testSet)))
        print('Average Accuracy of Unprocessed data: {0}%'.format(accuracy))

        #processed dataset load
        dataset = loadCsv2(filename)
        #print(dataset)
```

```
        accuracySum = 0
        for i in range(10):
                trainingSet, testSet = splitDataset(dataset, splitRatio)
                #print('Split {0} rows into train = {1} and test = {2}
rows'.format(len(dataset),len(trainingSet),len(testSet)))
                # prepare model
                summaries = summarizeByClass(trainingSet)
                # test model
                predictions = getPredictions(summaries, testSet)
                accuracy = getAccuracy(testSet, predictions)
                #print('Accuracy: {0}%'.format(accuracy))
                accuracySum = accuracySum + accuracy
        accuracySum = accuracySum/10
        print('Split {0} rows into train = {1} and test = {2}
rows'.format(len(dataset),len(trainingSet),len(testSet)))
        print('Average Accuracy of processed data with 0 values ignored: {0}%'.format(accuracy))


main()
```

FULL CODE PART 2

```
import random
import math
import pandas as pd
import numpy as np
from mnist import MNIST
import cv2
from scipy.stats import bernoulli
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

def GaussianNB(X_train, y_train, X_test, y_test, stretchedFlag):

	nclass = np.unique(y_train).shape[0]
	class_labels = np.unique(y_train)
	mean, std, priors= [], [], []
	for i, val in enumerate(class_labels):
		sep = [y_train == val]#separated
		priors.append((np.sum(sep)) / len(y_train))
		mean.append(np.mean(X_train[tuple(sep)], axis=0))
		std.append(np.std(X_train[tuple(sep)], axis=0))

	if stretchedFlag == 1:
		PredictionsTrain = getGaussianPredictions(X_train,
y_train,class_labels,mean,std,priors)
		accuracyTrain = getAccuracy (y_train,PredictionsTrain)
		print("Gaussian Stretched Image Train data accuracy  = {0}%".format(accuracyTrain))
		PredictionsTest = getGaussianPredictions(X_test,
y_test,class_labels,mean,std,priors)
		accuracyTest = getAccuracy (y_test,PredictionsTest)
		print("Gaussian Stretched Image Test data accuracy = {0}%".format(accuracyTest))
	else:
		PredictionsTrain = getGaussianPredictions(X_train,
y_train,class_labels,mean,std,priors)
		accuracyTrain = getAccuracy (y_train,PredictionsTrain)
		print("Gaussian Untouched Image Train data accuracy =
{0}%".format(accuracyTrain))
		PredictionsTest = getGaussianPredictions(X_test,
y_test,class_labels,mean,std,priors)
		accuracyTest = getAccuracy (y_test,PredictionsTest)
		print("Gaussian Untouched Image Test data accuracy = {0}%".format(accuracyTest))
		#Plot the mean pixel values calculated for the Normal distribution of the untouched
images
		for i in range(10):
			plt.subplot(4, 5, i+1)
			plt.imshow(mean[i].reshape(28,28))

def getGaussianPredictions(X_test, y_test,class_labels,mean,std,priors):

	pred = []
	smoothing = 0.00001
```

```
        for n in range(len(y_test)):
                classifier = []
                sample = X_test[n] #test sample
                for i, val in enumerate(class_labels):
                        means = mean[i]
                        var = np.square(std[i]) + smoothing
                        prob = 1 / np.sqrt(2 * np.pi * var) * np.exp(-np.square(sample - means)/(2 *
var))
                        result = np.sum(np.log(prob)) + np.log(priors[i]) #, np.log(self.priors[i])) #not
needed, we assume equal prior
                        classifier.append(result)
                pred.append(np.argmax(classifier))
        return pred

def BernoulliNB(X_train, y_train, X_test, y_test, stretchedFlag):

        nclass = np.unique(y_train).shape[0]
        class_labels = np.unique(y_train)
        nfeature = X_train.shape[1]
        priors= []
        bpoint = np.zeros((nclass,nfeature))
        for i, val in enumerate(class_labels):
                sep = [y_train == val]#separated
                priors.append((np.sum(sep)) / len(y_train))
                bpoint[i]  = (np.mean(X_train[tuple(sep)], axis=0))/255

        if stretchedFlag == 1:
                PredictionsTrain = getBernoulliPredictions(X_train,
y_train,bpoint,priors,nclass,class_labels)
                accuracyTrain = getAccuracy (y_train,PredictionsTrain)
                print("Bernoulli Stretched Image Train data accuracy  = {0}%".format(accuracyTrain))
                PredictionsTest =
getBernoulliPredictions(X_test,y_test,bpoint,priors,nclass,class_labels)
                accuracyTest = getAccuracy (y_test,PredictionsTest)
                print("Bernoulli Stretched Image Test data accuracy = {0}%".format(accuracyTest))
        else:
                PredictionsTrain = getBernoulliPredictions(X_train,
y_train,bpoint,priors,nclass,class_labels)
                accuracyTrain = getAccuracy (y_train,PredictionsTrain)
                print("Bernoulli Untouched Image Train data accuracy =
{0}%".format(accuracyTrain))
                PredictionsTest =
getBernoulliPredictions(X_test,y_test,bpoint,priors,nclass,class_labels)
                accuracyTest = getAccuracy (y_test,PredictionsTest)
                print("Bernoulli Untouched Image Test data accuracy = {0}%".format(accuracyTest))

def getBernoulliPredictions(X_test,y_test,bpoint,priors,nclass,class_labels):

        samples = X_test.shape[0]
        log_py_on_x = np.zeros((samples,nclass))
        for i in range(nclass):
```

```
                log_py_on_x[:,i] = np.log(priors[i]) +
np.sum(np.log(bernoulli.pmf(X_test/255,bpoint[i])),axis=1)
        label = class_labels[np.argmax(log_py_on_x, axis=1)]
        return label


def RandomForestNB(X_train, y_train, X_test, y_test, X_trainTouched, X_testTouched):

        RandomForest(X_train, y_train, X_train, y_train, 10, 4, train=1, stretchedFlag=0)
        RandomForest(X_train, y_train, X_test, y_test, 10, 4, train=0, stretchedFlag=0)
        RandomForest(X_trainTouched, y_train, X_trainTouched, y_train, 10, 4, train=1,
stretchedFlag=1)
        RandomForest(X_trainTouched, y_train, X_testTouched, y_test, 10, 4, train=0,
stretchedFlag=1)

        RandomForest(X_train, y_train, X_train, y_train, 10, 16, train=1, stretchedFlag=0)
        RandomForest(X_train, y_train, X_test, y_test, 10, 16, train=0, stretchedFlag=0)
        RandomForest(X_trainTouched, y_train, X_trainTouched, y_train, 10, 16, train=1,
stretchedFlag=1)
        RandomForest(X_trainTouched, y_train, X_testTouched, y_test, 10, 16, train=0,
stretchedFlag=1)

        RandomForest(X_train, y_train, X_train, y_train, 30, 4, train=1, stretchedFlag=0)
        RandomForest(X_train, y_train, X_test, y_test, 30, 4, train=0, stretchedFlag=0)
        RandomForest(X_trainTouched, y_train, X_trainTouched, y_train, 30, 4, train=1,
stretchedFlag=1)
        RandomForest(X_trainTouched, y_train, X_testTouched, y_test, 30, 4, train=0,
stretchedFlag=1)

        RandomForest(X_train, y_train, X_train, y_train, 30, 16, train=1, stretchedFlag=0)
        RandomForest(X_train, y_train, X_test, y_test, 30, 16, train=0, stretchedFlag=0)
        RandomForest(X_trainTouched, y_train, X_trainTouched, y_train, 30, 16, train=1,
stretchedFlag=1)
        RandomForest(X_trainTouched, y_train, X_testTouched, y_test, 30, 16, train=0,
stretchedFlag=1)

def RandomForest(X_train, y_train, X_test, y_test, estimators, depth, train, stretchedFlag):

        model = RandomForestClassifier(n_estimators=estimators, max_depth=depth)
        model.fit(X_train, y_train)
        Predictions = model.predict(X_test)
        accuracy = getAccuracy (y_test,Predictions)
        if train ==1:
                if stretchedFlag ==0:
                        print("RandomForest Untouched Image Train data accuracy with Tree size
{0} and Depth {1} = {2}%".format(estimators,depth,accuracy))
                else:
                        print("RandomForest Stretched Image Train data accuracy with Tree size {0}
and Depth {1} = {2}%".format(estimators,depth,accuracy))
        else:
                if stretchedFlag ==0:
```

```
                print("RandomForest Untouched Image Test data accuracy with Tree size {0}
and Depth {1} = {2}%".format(estimators,depth,accuracy))
            else:
                print("RandomForest Stretched Image Test data accuracy with Tree size {0}
and Depth {1} = {2}%".format(estimators,depth,accuracy))

def processimg(X,stretchedFlag):

    thresholding = 127

    if stretchedFlag == 1:
        n = X.shape[0]
        X_reshape = X.reshape(n,28,28)
        X_reshape = X_reshape.astype('float')
        X_stretched_boundingbox = np.zeros((n,20,20))
        # iterating over images: create bounding box, stretch images to 20 x 20, and store
them in a list
        for i in range(n):
            lower_bound =  np.where(np.any(X_reshape[i],axis=1))[0][0]
            upper_bound =  np.where(np.any(X_reshape[i],axis=1))[0][-1]
            left_bound =  np.where(np.any(X_reshape[i],axis=0))[0][0]
            right_bound =  np.where(np.any(X_reshape[i],axis=0))[0][-1]
            X_stretched_boundingbox[i]=
cv2.resize(X_reshape[i][lower_bound:upper_bound,left_bound:right_bound], (20,20))
        X_stretched_boundingbox = X_stretched_boundingbox.reshape(n,20*20)
        X_thresholding = (X_stretched_boundingbox > thresholding).astype(float)*255
    else:
        X_thresholding = (X >thresholding).astype(float)*255

    return X_thresholding

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

def main():
    mndata = MNIST(r'C:\Users\asinghal\Desktop\MCS-DS\AML\HW1\part2')
    mndata.gz = True
    X_train, y_train = mndata.load_training()
    X_test, y_test = mndata.load_testing()
    X_train = np.array(X_train)
    X_test = np.array(X_test)
    X_train = processimg(X_train,stretchedFlag=0)
    X_test = processimg(X_test,stretchedFlag=0)
    X_trainTouched = processimg(X_train,stretchedFlag=1)
    X_testTouched = processimg(X_test,stretchedFlag=1)

    GaussianNB(X_train, y_train, X_test, y_test, stretchedFlag=0)
```

```
        GaussianNB(X_trainTouched, y_train, X_testTouched, y_test, stretchedFlag=1)
        BernoulliNB(X_train, y_train, X_test, y_test, stretchedFlag=0)
        BernoulliNB(X_trainTouched, y_train, X_testTouched, y_test, stretchedFlag=1)
        RandomForestNB(X_train, y_train, X_test, y_test, X_trainTouched, X_testTouched)

main()
```