CS 498 AMO (MCS-DS online) – Homework 1

Submitted by: Pushpit Saxena (netid: pushpit2)

Part 1 Accuracies

Setup	**Cross-validation Accuracy**
Unprocessed data	75.620915%*
0-value elements ignored	75.032680%*

* The accuracy mentioned here is from one of the 10 test-train split run. On average I see the accuracy for either of the case hovering in the range of (73.5% - 77%) for different runs.

Part 1 Code Snippets

Calculation of distribution parameters (Page 6 - Page 7)

```
def fit(self, X, Y):
        self.normDF = {}
        self.priors = {}
        categories = set(Y)
        if self.ignoreMissingVal:
            X[X == 0] = np.nan
        for c in categories:
            XForC = X[Y == c]
            self.normDF[c] = {
                'mean' : np.nanmean(XForC, axis=0),
                'var': np.nanvar(XForC, axis=0)
            }
            self.priors[c] = 1.0 * len(Y[Y == c])/ len(Y)
```

Calculation of naive Bayes predictions (Page 7 - Page 8)

```
def predict(self, X):
        P = {}
        for c, g in self.normDF.items():
            mean, var = g['mean'], g['var']
            classConditionalProb = 0
            # Please see the detailed method implementation in the full code attached at the end.
            classConditionalProb = np.sum(self.__calculateLogNormPdf(X, mean, np.sqrt(var)))
            P[c] = classConditionalProb + np.log(self.priors[c])
            bestCategory, bestProb = None, float("-inf")
        for category, probability in P.items():
            if bestCategory is None or probability > bestProb:
                bestProb = probability
                bestCategory = category
        return bestCategory
```

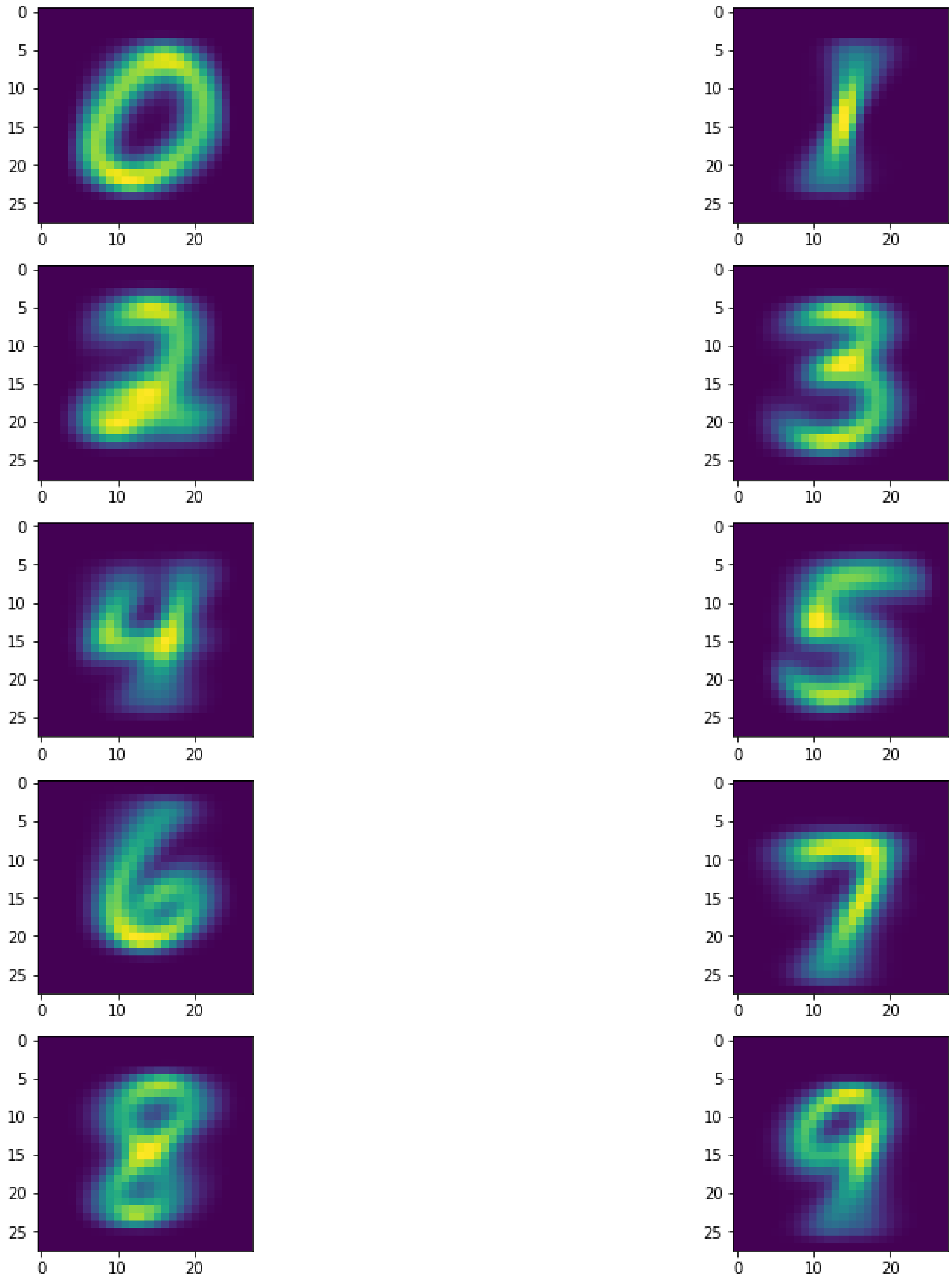
Test-train split code (Page 6)

```
def testTrainSplit(self, data, ratio):
    localCopy = list(data)
    testSize = int(len(data) * ratio)
    testData = []
    while len(testData) < testSize:
        testData.append(localCopy.pop(random.randrange(len(localCopy))))
    testNPArr  = np.array(testData)
    trainNPArr = np.array(localCopy)
    return trainNPArr[:, :8], trainNPArr[:, 8].astype(int), testNPArr[:, :8], testNPArr[:, 8].astype(int)
```

Part 2 MNIST Accuracies

x	Method	Training Set Accuracy	Test Set Accuracy
1	Gaussian + untouched	79.041667	79.860000
2	Gaussian + stretched	81.656667	82.450000
3	Bernoulli + untouched	83.576667	84.270000
4	Bernoulli + stretched	82.521667	83.690000
5	10 trees + 4 depth + untouched	69.641667	70.580000
6	10 trees + 4 depth + stretched	71.623333	73.170000
7	10 trees + 16 depth + untouched	99.071667	93.560000
8	10 trees + 16 depth + stretched	99.533333	94.430000
9	30 trees + 4 depth + untouched	74.141667	74.790000
10	30 trees + 4 depth + stretched	77.023333	78.310000
11	30 trees + 16 depth + untouched	99.405000	95.550000
12	30 trees + 16 depth + stretched	99.695000	96.410000

Mean images – Code on Page 11 - 12



Part 2 Code

Calculation of the Normal distribution parameters (Page 10 – 11)

```
for c in categories:
    XForC = X[Y == c]
    self.normDF[c] = {
         'mean' : np.nanmean(XForC, axis=0),
         'var': np.nanvar(XForC, axis=0)
     }
     self.priors[c] = 1.0 * len(Y[Y == c])/ len(Y)
```

Calculation of the Bernoulli distribution parameters (Page 12 – 13)

```
self.priors[d] = log(1.0 * N/ len(Y))
countOf1 = np.count_nonzero(XForD, axis=0)
countOf1 = (countOf1 + 1.) /(N + 2.)  #Laplace smoothing
self.cond_probs[d] = countOf1
```

Calculation of the Naive Bayes predictions

For Gaussian (Page 11):

```
 for c, g in self.normDF.items():
    mean, var = g['mean'], g['var']
    var = var + smoothing
    normPdf = norm.pdf(X, mean, np.sqrt(var))
    normPdf[normPdf == 0] = np.nan
    classConditionalProb = np.nansum(np.log(normPdf))
    P[c] = classConditionalProb + np.log(self.priors[c])
```

For Bernoulli (Page 13):

```
pred_class = None
max_ = float("-inf")
for d in self.priors:
    log_sum = self.priors[d]
    log_sum += np.sum(np.log(self.cond_probs[d][X == 1]))
    log_sum += np.sum(np.log(1 - self.cond_probs[d][X == 0]))
    if log_sum > max_:
        max_ = log_sum
        pred_class = d
```

Training of a decision tree (Page 14)

```
clf = RandomForestClassifier(n_estimators=numOfTrees, max_depth=maxDepth)
clf.fit(x_train, y_train)
```

Calculation of a decision tree predictions (Page 14)

```
y_pred_test = clf.predict(x_test)
correct_test = np.sum(y_pred_test == y_test)
accuracy_test = (correct_test/float(len(y_test)) * 100.0)
```

Part A – complete code

```
import pandas as pd
import numpy as np
from scipy.stats import norm
import random
import math
raw_data = pd.read_csv('/Users/psaxena21/Documents/Mine/AML/pima-indians-diabetes.csv', header=None)
d = raw_data.values


class NaiveBayes:
    def __init__(self, ignoreMissingVal):
        self.ignoreMissingVal = ignoreMissingVal

    def testTrainSplit(self, data, ratio):
        localCopy = list(data)
        testSize = int(len(data) * ratio)
        testData = []
        while len(testData) < testSize:
            testData.append(localCopy.pop(random.randrange(len(localCopy))))
        testNPArr  = np.array(testData)
        trainNPArr = np.array(localCopy)
        return trainNPArr[:, :8], trainNPArr[:, 8].astype(int), testNPArr[:, :8], testNPArr[:, 8].astype(int)

    def fit(self, X, Y):
        self.normDF = {}
        self.priors = {}
        categories = set(Y)
        if self.ignoreMissingVal:
```

```
        X[X == 0] = np.nan
    for c in categories:
        XForC = X[Y == c]
        self.normDF[c] = {
            'mean' : np.nanmean(XForC, axis=0),
            'var': np.nanvar(XForC, axis=0)
        }
        self.priors[c] = 1.0 * len(Y[Y == c])/ len(Y)

  def __calculateProbability(self, x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1.0 / (math.sqrt(2*math.pi) * stdev)) * exponent

  def __calculateLogNormPdf(self, X, mean, stddev):
    local_X = X[X != 0] if self.ignoreMissingVal else X
    local_mean = mean[X != 0] if self.ignoreMissingVal else mean
    local_stddev = stddev[X != 0] if self.ignoreMissingVal else stddev
    return np.log(np.array([self.__calculateProbability(local_X[i], local_mean[i], local_stddev[i]) for i in
range(len(local_X))]))

  def predict(self, X):
    P = {}
    for c, g in self.normDF.items():
        # print "c:", c
        mean, var = g['mean'], g['var']
        classConditionalProb = 0
#           classConditionalProb = np.nansum(np.log(norm.pdf(X[X, mean, np.sqrt(var))))
        classConditionalProb = np.sum(self.__calculateLogNormPdf(X, mean, np.sqrt(var)))
        P[c] = classConditionalProb + np.log(self.priors[c])

        bestCategory, bestProb = None, float("-inf")
    for category, probability in P.items():
```

```
            if bestCategory is None or probability > bestProb:
                bestProb = probability
                bestCategory = category
        return bestCategory


def runClassifier(data, ignoreMissingVal=False):
    accuracy = 0
    for i in range(10):
        nb = NaiveBayes(ignoreMissingVal)
        np.random.shuffle(data)
        X_Train, Y_Train, X_Test, Y_Test = nb.testTrainSplit(data, 0.20)
        nb.fit(X_Train, Y_Train)
        correct = 0
        for i in range(len(Y_Test)):
            Y_Pred = nb.predict(X_Test[i])
            if Y_Test[i] == Y_Pred:
                correct += 1
        accuracy += (correct/float(len(Y_Test)) * 100.0)
    return accuracy/10

acc = runClassifier(d)
print("Average accuracy over 10 test-train splits and without ignoring missing values is %f" % (acc))


acc = runClassifier(d, True)
print("Average accuracy over 10 test-train splits and ignoring missing values is %f" % (acc))
```

Part B – Complete code

```
%matplotlib inline
from matplotlib import pyplot as plt
import tensorflow as tf
from scipy.stats import norm
```

```python
import numpy as np
import cv2
from math import log
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
ret, x_train_thresh = cv2.threshold(x_train, 127, 1, cv2.THRESH_BINARY)
ret, x_test_thresh = cv2.threshold(x_test, 127, 1, cv2.THRESH_BINARY)

def getStretchedImage(img):
    rows = np.any(img, axis=1)
    cols = np.any(img, axis=0)
    rmin, rmax = np.where(rows)[0][[0, -1]]
    cmin, cmax = np.where(cols)[0][[0, -1]]
    cropped = img[rmin:rmax, cmin:cmax]
    return cv2.resize(cropped, (20,20), interpolation=cv2.INTER_NEAREST)

def plotSampleFig(x_train_thresh, x_train_thresh_stretch):
    pixels = x_train_thresh[11232]
    img2 = cv2.resize(pixels, (20, 20), interpolation = cv2.INTER_NEAREST)
    img3 = getStretchedImage(pixels)
    f, axarr = plt.subplots(2,2)
    axarr[0,0].imshow(pixels, cmap=plt.cm.binary)
    axarr[0,1].imshow(img2, cmap=plt.cm.binary)
    axarr[1,0].imshow(img3, cmap=plt.cm.binary)
    axarr[1,1].imshow(x_train_thresh_stretch[11232], cmap=plt.cm.binary)
    plt.show()


x_train_thresh_stretch = np.array([getStretchedImage(x_train_thresh[i]) for i in range(len(x_train_thresh))])
x_test_thresh_stretch = np.array([getStretchedImage(x_test_thresh[i]) for i in range(len(x_test_thresh))])
plotSampleFig(x_train_thresh, x_train_thresh_stretch)
```

```
print("Shape of stretched training set " + repr(x_train_thresh_stretch.shape[0]))
x_train_flat_thresh = x_train_thresh.reshape(60000, 784)
x_test_flat_thresh = x_test_thresh.reshape(10000, 784)
x_train_flat_thresh_stretch = x_train_thresh_stretch.reshape(60000, 400)
x_test_flat_thresh_stretch = x_test_thresh_stretch.reshape(10000, 400)

def calculateAccuracy(nb, x_test, y_test, dataType="Untouched", setType="Test", modelType="Gaussian"):
    y_pred = np.apply_along_axis(nb.predict, 1, x_test)
    correct = np.sum(y_pred == y_test)
    accuracy = (correct/float(len(y_test)) * 100.0)
    print("Set-type: %s, DataType: %s, ModelType: %s, Accuracy: %f"%(setType, dataType, modelType, accuracy))

# Gaussian NB implementation
class NaiveBayesGaussian:

    def fit(self, X, Y):
        self.normDF = {}
        self.priors = {}
        categories = set(Y)
        for c in categories:
            XForC = X[Y == c]
            self.normDF[c] = {
                'mean' : np.nanmean(XForC, axis=0),
                'var': np.nanvar(XForC, axis=0)
            }
            self.priors[c] = 1.0 * len(Y[Y == c])/ len(Y)

    def predict(self, X, smoothing=.01):
        P = {}
        for c, g in self.normDF.items():
            mean, var = g['mean'], g['var']
            var = var + smoothing
```

```
            normPdf = norm.pdf(X, mean, np.sqrt(var))

            normPdf[normPdf == 0] = np.nan

            classConditionalProb = np.nansum(np.log(normPdf))

            P[c] = classConditionalProb + np.log(self.priors[c])

            bestCategory, bestProb = None, float("-inf")

        for category, probability in P.items():

            if bestCategory is None or probability > bestProb:

                bestProb = probability

                bestCategory = category

        return bestCategory



nb = NaiveBayesGaussian()

nb.fit(x_train_flat_thresh, y_train)

print("Done training NB Gaussian untouched")

calculateAccuracy(nb, x_test_flat_thresh, y_test, "Untouched", "Test", "Gaussian")

calculateAccuracy(nb, x_train_flat_thresh, y_train, "Untouched", "Train", "Gaussian")


              # Implementation for mean image plot

f, axarr = plt.subplots(5,2 , figsize=(15,15))

k = 0

mean_img_arr = [g['mean'] for c, g in nb.normDF.items()]

for i in range(5):

    for j in range(2):

        axarr[i, j].imshow(mean_img_arr[k].reshape((28,28)))

        k+= 1

plt.show()


nb = NaiveBayesGaussian()

nb.fit(x_train_flat_thresh_stretch, y_train)

print("Done training NB Gaussian stretched")
```

```
calculateAccuracy(nb, x_test_flat_thresh_stretch, y_test, "Stretched", "Test", "Gaussian")

calculateAccuracy(nb, x_train_flat_thresh_stretch, y_train, "Stretched", "Train", "Gaussian")


#Bernoulli NB implementation

class BernoulliNBClassifier(object):


    def __init__(self):

        self.priors = {}

        self.cond_probs = {}


    def fit(self, X, Y):

        digits = set(Y)

        for d in digits:

            XForD = X[Y == d]

            N = len(Y[Y == d])

            self.priors[d] = log(1.0 * N/ len(Y))


            """Compute log( P(X|Y) )

               Use Laplace smoothing

               n1 + 1 / (n1 + n2 + 2)

            """

            countOf1 = np.count_nonzero(XForD, axis=0)

            countOf1 = (countOf1 + 1.) /(N + 2.)

            self.cond_probs[d] = countOf1



    def predict(self, X):

        """Make a prediction from text

        """


        pred_class = None

        max_ = float("-inf")
```

```
        # Perform MAP estimation
        for d in self.priors:
            log_sum = self.priors[d]
            log_sum += np.sum(np.log(self.cond_probs[d][X == 1]))
            log_sum += np.sum(np.log(1 - self.cond_probs[d][X == 0]))
            if log_sum > max_:
                max_ = log_sum
                pred_class = d

        return pred_class

nb = BernoulliNBClassifier()
nb.fit(x_train_flat_thresh, y_train)
print("Done training NB Bernoulli untouched")
calculateAccuracy(nb, x_test_flat_thresh, y_test, "Untouched", "Test", "Bernoulli")
calculateAccuracy(nb, x_train_flat_thresh, y_train, "Untouched", "Train", "Bernoulli")


nb = BernoulliNBClassifier()
nb.fit(x_train_flat_thresh_stretch, y_train)
print("Done training NB Bernoulli stretched")
calculateAccuracy(nb, x_test_flat_thresh_stretch, y_test, "Stretched", "Test", "Bernoulli")
calculateAccuracy(nb, x_train_flat_thresh_stretch, y_train, "Stretched", "Train", "Bernoulli")

#DecisionTrees
from sklearn.ensemble import RandomForestClassifier

def runDecisionForestClassifier(x_train, y_train, x_test, y_test, numOfTrees, maxDepth, dataType="Untouched"):
    clf = RandomForestClassifier(n_estimators=numOfTrees, max_depth=maxDepth)
    clf.fit(x_train, y_train)
    y_pred_test = clf.predict(x_test)
```

```
    y_pred_train = clf.predict(x_train)


    correct_test = np.sum(y_pred_test == y_test)
    correct_train = np.sum(y_pred_train == y_train)
    accuracy_test = (correct_test/float(len(y_test)) * 100.0)
    accuracy_train = (correct_train/float(len(y_train)) * 100.0)
    print("%s image, %d trees, %d depth, test-set-accuracy: %f, train-set-accuracy: %f" %(dataType, numOfTrees,
maxDepth, accuracy_test, accuracy_train ))



runDecisionForestClassifier(x_train_flat_thresh, y_train, x_test_flat_thresh, y_test, 10, 4)
runDecisionForestClassifier(x_train_flat_thresh, y_train, x_test_flat_thresh, y_test, 30, 4)
runDecisionForestClassifier(x_train_flat_thresh, y_train, x_test_flat_thresh, y_test, 10, 16)
runDecisionForestClassifier(x_train_flat_thresh, y_train, x_test_flat_thresh, y_test, 30, 16)
strch = "Stretched"
runDecisionForestClassifier(x_train_flat_thresh_stretch, y_train, x_test_flat_thresh_stretch, y_test, 10, 4, strch)
runDecisionForestClassifier(x_train_flat_thresh_stretch, y_train, x_test_flat_thresh_stretch, y_test, 30, 4, strch)
runDecisionForestClassifier(x_train_flat_thresh_stretch, y_train, x_test_flat_thresh_stretch, y_test, 10, 16, strch)
runDecisionForestClassifier(x_train_flat_thresh_stretch, y_train, x_test_flat_thresh_stretch, y_test, 30, 16, strch)
```

References for code:

https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

https://github.com/GPSingularity/Machine-Learning-in-Python/blob/master/nbsingularity.py

https://mattshomepage.com/articles/2016/Jun/07/bernoulli_nb/

https://nlp.stanford.edu/IR-book/html/htmledition/the-bernoulli-model-1.html