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)

1. **def** fit(self, X, Y):
2. self.normDF = {}
3. self.priors = {}
4. categories = set(Y)
5. **if** self.ignoreMissingVal:
6. X[X == 0] = np.nan
7. **for** c **in** categories:
8. XForC = X[Y == c]
9. self.normDF[c] = {
10. 'mean' : np.nanmean(XForC, axis=0),
11. 'var': np.nanvar(XForC, axis=0)
12. }
13. self.priors[c] = 1.0 \* len(Y[Y == c])/ len(Y)

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

1. **def** predict(self, X):
2. P = {}
3. **for** c, g **in** self.normDF.items():
4. mean, var = g['mean'], g['var']
5. classConditionalProb = 0
6. # Please see the detailed method implementation in the full code attached at the end.
7. classConditionalProb = np.sum(self.\_\_calculateLogNormPdf(X, mean, np.sqrt(var)))
8. P[c] = classConditionalProb + np.log(self.priors[c])
9. bestCategory, bestProb = None, float("-inf")
10. **for** category, probability **in** P.items():
11. **if** bestCategory **is** None **or** probability > bestProb:
12. bestProb = probability
13. bestCategory = category
14. **return** bestCategory

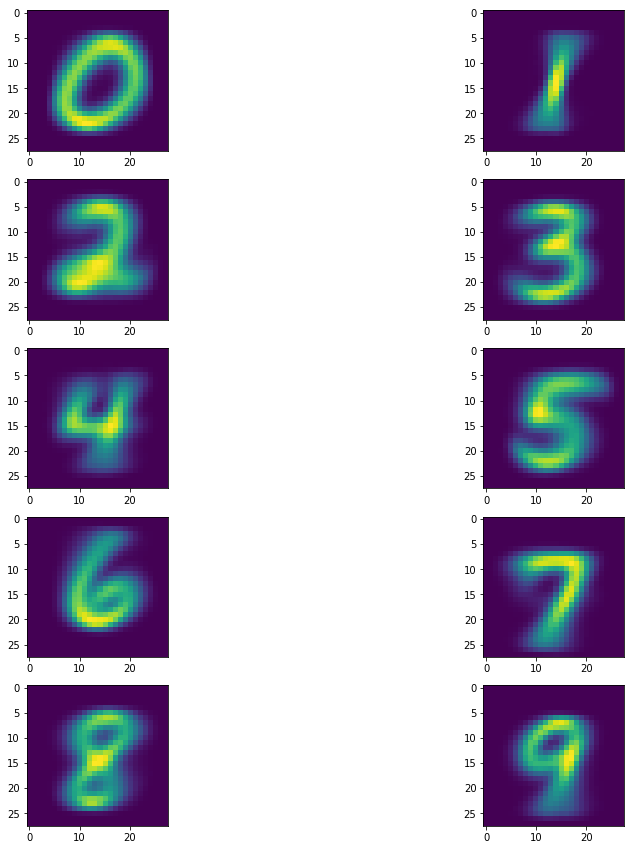
#### Test-train split code (Page 6)

1. **def** testTrainSplit(self, data, ratio):
2. localCopy = list(data)
3. testSize = int(len(data) \* ratio)
4. testData = []
5. **while** len(testData) < testSize:
6. testData.append(localCopy.pop(random.randrange(len(localCopy))))
7. testNPArr  = np.array(testData)
8. trainNPArr = np.array(localCopy)
9. **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)

1. **for** c **in** categories:
2. XForC = X[Y == c]
3. self.normDF[c] = {
4. 'mean' : np.nanmean(XForC, axis=0),
5. 'var': np.nanvar(XForC, axis=0)
6. }
7. self.priors[c] = 1.0 \* len(Y[Y == c])/ len(Y)

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

1. self.priors[d] = log(1.0 \* N/ len(Y))
2. countOf1 = np.count\_nonzero(XForD, axis=0)
3. countOf1 = (countOf1 + 1.) /(N + 2.)  #Laplace smoothing
4. self.cond\_probs[d] = countOf1

#### Calculation of the Naive Bayes predictions

#### For Gaussian (Page 11):

1. **for** c, g **in** self.normDF.items():
2. mean, var = g['mean'], g['var']
3. var = var + smoothing
4. normPdf = norm.pdf(X, mean, np.sqrt(var))
5. normPdf[normPdf == 0] = np.nan
6. classConditionalProb = np.nansum(np.log(normPdf))
7. P[c] = classConditionalProb + np.log(self.priors[c])

#### For Bernoulli (Page 13):

1. pred\_class = None
2. max\_ = float("-inf")
3. **for** d **in** self.priors:
4. log\_sum = self.priors[d]
5. log\_sum += np.sum(np.log(self.cond\_probs[d][X == 1]))
6. log\_sum += np.sum(np.log(1 - self.cond\_probs[d][X == 0]))
7. **if** log\_sum > max\_:
8. max\_ = log\_sum
9. pred\_class = d

#### Training of a decision tree (Page 14)

1. clf = RandomForestClassifier(n\_estimators=numOfTrees, max\_depth=maxDepth)
2. clf.fit(x\_train, y\_train)

#### Calculation of a decision tree predictions (Page 14)

1. y\_pred\_test = clf.predict(x\_test)
2. correct\_test = np.sum(y\_pred\_test == y\_test)
3. 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

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>