Assignment # 2
Morgan Benavidez
Z23589091
https://colab.research.google.com/drive/19ihokHezJHFtyzJBuCwyjYTVCTiDJP3x?usp=sharing

---

Assignment2

Tuesday, February 7, 2023 6:30 AM

Test = Gold Standard Labels

Predicted

0 = non-fraudulent emails

1 = Fraudulent emails

	F	NF	Total
F	22	4	26
NF	4	10	14

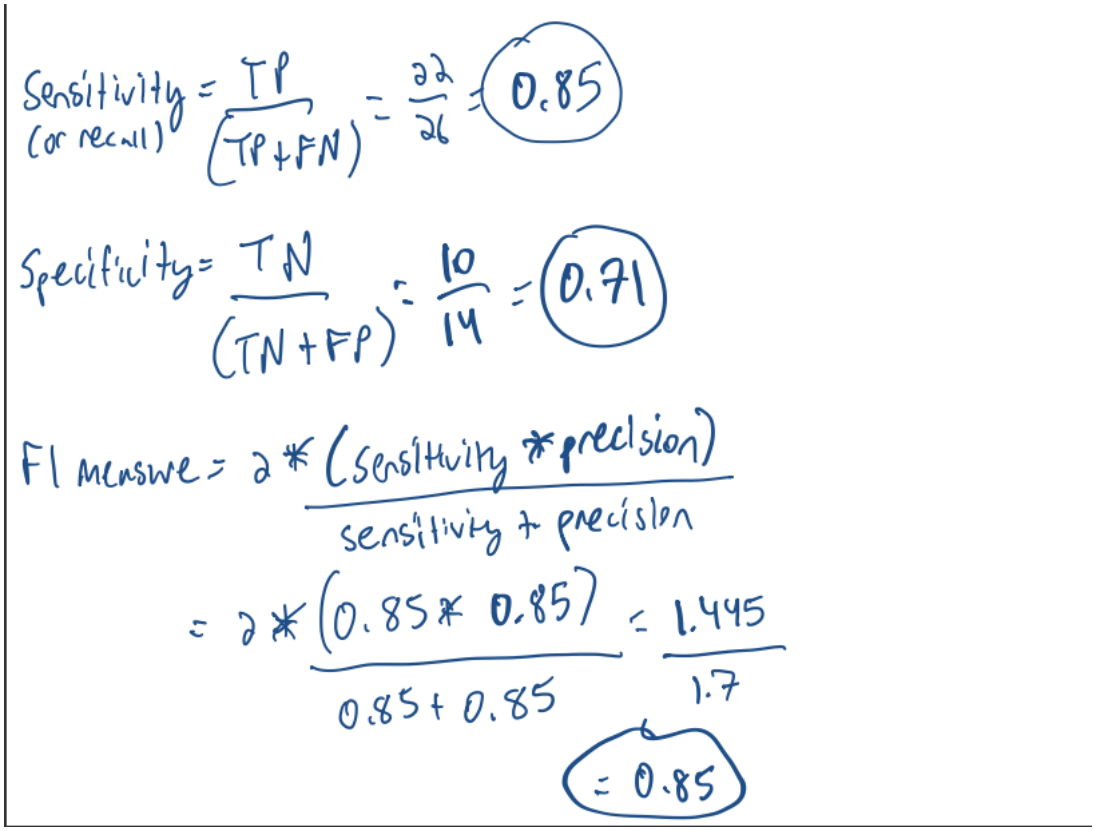
"important"

	F	NF
F	true +	False −
NF	false +	true −

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + FN + TN)}$$

$$= \frac{(22 + 10)}{40} = \frac{32}{40} = 0.80$$

$$\text{Precision} = \frac{TP}{(TP + FP)} = \frac{22}{26} = 0.85$$




```
1 from keras.datasets import mnist
```

```
1 import numpy as np
2 import matplotlib.pyplot as plt
```

```
1 import matplotlib.image as mpimg
2 import random
```

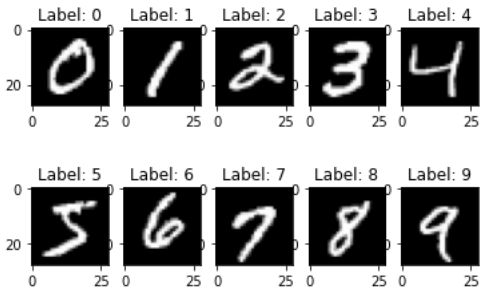
```
1 # a)
2
3 #Load mnist data set and split into training and testing sets
4 (x_train, y_train), (x_test, y_test) = mnist.load_data()
5
6 # Print number of images in each training and testing set
7 # and the image width and height
8 print(x_train.shape, y_train.shape)
9 height = x_train.shape[1]
10 width = x_train.shape[2]
11 print("Image height =", height, "pixels")
12 print("Image width =", width, "pixels")
13
14 print(x_test.shape, y_test.shape)
15 height2 = x_test.shape[1]
16 width2 = x_test.shape[2]
17 print("Image height =", height, "pixels")
18 print("Image width =", width, "pixels")
```

```
(60000, 28, 28) (60000,)
Image height = 28 pixels
Image width = 28 pixels
(10000, 28, 28) (10000,)
Image height = 28 pixels
Image width = 28 pixels
```

```
1 # b) - First edition: contains the for loop inside of the function
2
3 # Function takes a set of images and its labels as inputs and plots a figure
4 # with 10 subplots for each digit 0-9. Each subplot is labeled with digit in title
5 def plot_digits(images, labels):
6   total = 0
7   digits = [0,1,2,3,4,5,6,7,8,9]
```

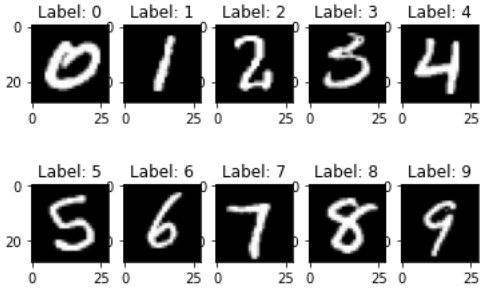
```
 8
 9  for digit in digits:
10    x_train_digit = images[labels==digit,:,:] # can use 0, 27 instead of :,:
11    total += x_train_digit.shape[0]
12
13    x_train_single = x_train_digit[0,:,:]
14    plt.subplot(2, 5, digit+1)
15    plt.imshow(x_train_single, cmap='gray')
16    plt.title('Label: ' + str(digit))
17
18  # Verifies that every digit in the set has been accounted for
19  print(total)
20
21 plot_digits(x_train, y_train)
```

```
60000
```




```
 1 # b) - Second Edition: takes a single image and its label to create the subplot
 2
 3 # Function takes a set of images and its labels as inputs and plots a figure
 4 # with 10 subplots for each digit 0-9. Each subplot is labeled with digit in title
 5 def plot_digits2(image, label):
 6     global counter
 7     plt.subplot(2, 5, counter)
 8     plt.imshow(image, cmap='gray')
 9     plt.title('Label: ' + str(label))
10     counter +=1
11
```

```
 1 # c) - For loop calls the function in b Second Edition to create the figure with subplots
 2
 3 counter = 1
 4
 5 for i in range(0,10):
 6
 7   # Each one of these is the group of each individual digit
 8   x_train_digit = x_train[y_train==i,:,:]
 9
10   # I used random.randint to choose a random index in each group of digits
11   # to select different images to use in the subplot
12   # Every time you run this, the subplots will be built of different images
13   x_train_single = x_train_digit[random.randint(0, 1000),:,:]
14
15   # Call Function to build figure
16   plot_digits2(x_train_single, i)
```




```
 1 # d) - Select the 0 and 8 digits from the training and testing sets and rename
 2
 3
 4 # Extracting 0s and 8s from training set and concatenating
 5
```

```
zeros = x_train[y_train==0,:,:]
#zeros_labels = np.full(shape=len(zeros), fill_value=0)
zeros_labels = np.zeros(len(zeros)) # Does the same as above line

eights = x_train[y_train==8,:,:]
#eights_labels = np.full(shape=len(eights), fill_value=8)
eights_labels = np.ones(len(eights))*8 # Does the same as above line

x_train_01 = np.concatenate((zeros, eights), axis=0)
y_train_01 = np.concatenate((zeros_labels, eights_labels), axis=0)

print(x_train_01.shape)
print(y_train_01.shape)


# Extracting 0s and 8s from testing set and concatenating

zeros2 = x_test[y_test==0,:,:]
#zeros_labels = np.full(shape=len(zeros), fill_value=0)
zeros_labels2 = np.zeros(len(zeros2)) # Does the same as above line

eights2 = x_test[y_test==8,:,:]
#eights_labels = np.full(shape=len(eights), fill_value=8)
eights_labels2 = np.ones(len(eights2))*8 # Does the same as above line

x_test_01 = np.concatenate((zeros2, eights2), axis=0)
y_test_01 = np.concatenate((zeros_labels2, eights_labels2), axis=0)

print(x_test_01.shape)
print(y_test_01.shape)

```

```
(11774, 28, 28)
(11774,)
(1954, 28, 28)
(1954,)
```

```
# e)

# Random indices to create validation set
indices = random.sample(range(len(x_train_01)), 500)

# Creation of validation set from training sets
x_valid_01 = x_train_01[indices]
y_valid_01 = y_train_01[indices]
# Show the length of these validation sets
print("Length of x_valid_01 =", len(x_valid_01))
print("Length of y_valid_01 =", len(y_valid_01))

# Show the removal of items in validation sets from training sets
print("Original size of x_train_01 =", len(x_train_01))
x_train_01 = np.delete(x_train_01, indices, 0)
print("Size of x_train_01 after 500 random removed =", len(x_train_01))

print("Original size of y_train_01 =", len(y_train_01))
y_train_01 = np.delete(y_train_01, indices, 0)
print("Size of y_train_01 after 500 random removed =", len(y_train_01))

```

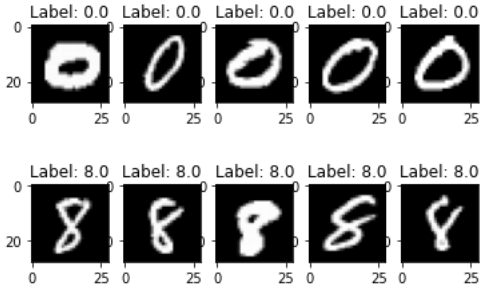
```
Length of x_valid_01 = 500
Length of y_valid_01 = 500
Original size of x_train_01 = 11774
Size of x_train_01 after 500 random removed = 11274
Original size of y_train_01 = 11774
Size of y_train_01 after 500 random removed = 11274
```

```
# f)

# Show the shapes of the validation sets and the training sets
print("Validation Sets =", x_valid_01.shape, y_valid_01.shape)
print("Number of images in Validation Set =", x_valid_01.shape[0], "\n")
print("Training Sets =", x_train_01.shape, y_train_01.shape)
print("Number of images in Training Set =", x_train_01.shape[0], "\n")
print("Testing Sets =", x_test_01.shape, y_test_01.shape)
print("Number of images in Testing Set =", x_test_01.shape[0], "\n")
```

```
Validation Sets = (500, 28, 28) (500,)
Number of images in Validation Set = 500

Training Sets = (11274, 28, 28) (11274,)
Number of images in Training Set = 11274

Testing Sets = (1954, 28, 28) (1954,)
Number of images in Testing Set = 1954
```

```
# g)

counter = 1

x = random.randint(0, 490)

for i in range(x, x+10):
#for i in range(0, 10):

  plot_digits2(x_valid_01[i], y_valid_01[i])


  v = x_valid_01[i]
  y = y_valid_01[i]
  valid = np.sum(v[12:16, 12:16], axis=1)
  valid = np.sum(valid, axis=0)/4
  print(valid)
```

```
350.5
59.75
100.75
0.0
16.0
745.5
713.5
732.5
598.25
363.5
```

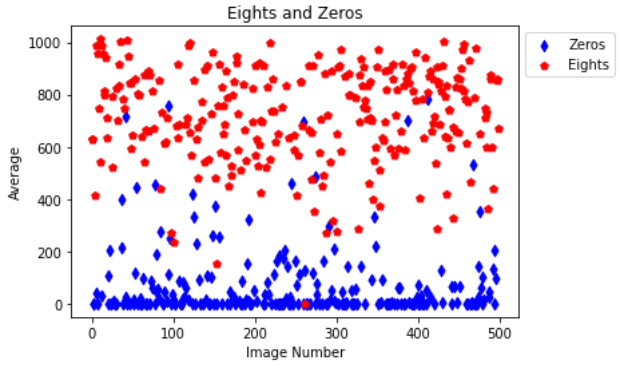



```
# h)

# Convert each image to an average of 4x4 center pixels for training data
train_features = np.sum(x_train_01[:,12:16, 12:16], axis=2)
train_features = np.sum(train_features, axis=1)/4
print(train_features.shape)

# Convert each image to an average of 4x4 center pixels for validation data
validate_features =np.sum(x_valid_01[:,12:16, 12:16], axis=2)
validate_features = np.sum(validate_features, axis=1)/4
print(validate_features.shape)

# Convert each image to an average of 4x4 center pixels for testing data
testing_features = np.sum(x_test_01[:,12:16, 12:16], axis=2)
testing_features = np.sum(testing_features, axis=1)/4
print(testing_features.shape)
```

```
(11274,)
(500,)
(1954,)
```

```
# i)

```

```
def plot_eights_and_zeros(y_valid_01, validate_features):

  Zeros = []
  ZeroImageNumber = []
  Eights = []
  EightImageNumber = []

  for i in range(len(validate_features)):

    if (y_valid_01[i] == 0.0):
      Zeros.append(validate_features[i])
      ZeroImageNumber.append(i)
      #plt.scatter(i, validate_features[i], color='blue', marker='d')
      pass
    elif (y_valid_01[i] == 8.0):
      Eights.append(validate_features[i])
      EightImageNumber.append(i)
      #plt.scatter(i, validate_features[i], color='red', marker='p')
      pass

  plt.scatter(ZeroImageNumber, Zeros, color='blue', marker='d')
  plt.scatter(EightImageNumber, Eights, color='red', marker='p')
  plt.ylabel('Average')
  plt.xlabel('Image Number')
  plt.title('Eights and Zeros')
  #plt.axis([-1, 5, -1, 5])
  plt.legend(['Zeros', 'Eights', 'threshold_y', 'threshold_x'], bbox_to_anchor=(1, 1), loc='upper left')
  #plt.figure(figsize=(2,2))
  plt.figure(figsize=(4,4))
  plt.show()

plot_eights_and_zeros(y_valid_01, validate_features)
```




```
<Figure size 288x288 with 0 Axes>
```

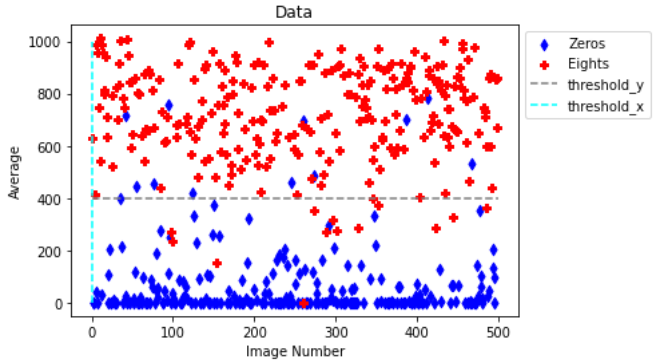
```
# j)

# My guess is that choosing a threshold of "y=400" will yield the greatest accuracy.
```

```
# k)

import numpy as np
import matplotlib.pyplot as plt
import time

def classification_accuracy(threshold_x, threshold_y, C1, C2):

  correct = 0
  incorrect = 0
  total = len(C1) + len(C2)
  print(total)

  for i in range(0, len(C1)):

    # Zeros
    y = C1[i][1]

    if (y >= threshold_y):
      incorrect += 1
    elif (y < threshold_y):
      correct += 1
```

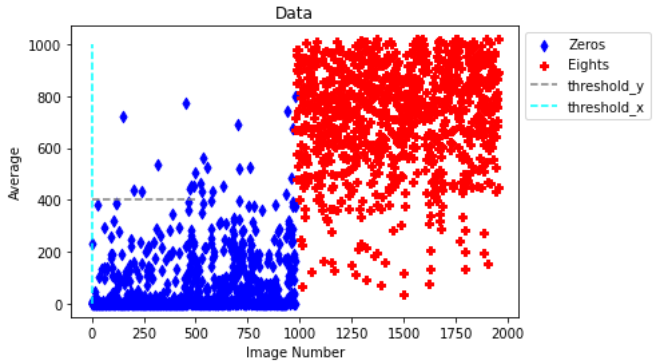
```
23
24   for i in range(0, len(C2)):
25
26     # Eights
27     y2 = C2[i][1]
28
29     if (y2 >= threshold_y):
30       correct += 1
31     elif (y2 < threshold_y):
32       incorrect += 1
33
34
35   return correct, incorrect, total
36
37 def obtain_thresholds():
38
39   while (True):
40     try:
41       var = input("Threshold x must be a number, please enter an integer: ")
42       if (var == 'x'):
43         testing = False
44         threshold_x = 'x'
45         break
46       threshold_x = int(var)
47     except ValueError:
48       print("Threshold must be an integer, please try again.")
49       continue
50     else:
51       break
52
53   while (True):
54     try:
55       var2 = input("Threshold y must be a number, please enter an integer: ")
56       if (var2 == 'x'):
57         testing = False
58         threshold_y = 'x'
59         break
60       threshold_y = int(var2)
61     except ValueError:
62       print("Threshold must be an integer, please try again.")
63       continue
64     else:
65       break
66
67   return threshold_x, threshold_y
68
69 def print_accuracy_results(results):
70
71   print('Correct: ' + str(results[0]))
72   print('Incorrect: ' + str(results[1]))
73   print('Total: ' + str(results[2]))
74   print('Classification Accuracy: ' + str(results[0]/results[2]))
75
76
77 def create_plot(C1, C2, threshold_x, threshold_y):
78
79   zero = []
80   zerolabel = []
81   eight = []
82   eightlabel = []
83
84   for i in range(0, len(C1)):
85
86     zero.append(C1[i][1])
87     zerolabel.append(C1[i][0])
88
89   for i in range(0, len(C2)):
90
91     eight.append(C2[i][1])
92     eightlabel.append(C2[i][0])
93
94
95   plt.scatter(zerolabel, zero, color='blue', marker='d')
96   plt.scatter(eightlabel, eight, color='red', marker='P')
97   plt.hlines(y=threshold_y, xmin=0, xmax=500, linestyle='dashed', color='gray')
98   plt.vlines(x=threshold_x, ymin=0, ymax=1000, linestyle='dashed', color='cyan')
99
```

```
  plt.ylabel('Average')
  plt.xlabel('Image Number')
  plt.title('Data')
  #plt.axis([-25, 10000, -25, 1200])
  plt.legend(['Zeros', 'Eights', 'threshold_y', 'threshold_x'], bbox_to_anchor=(1, 1), loc='upper left')
  plt.show()
  print('\n')

def main(C1, C2):

  thresholds = obtain_thresholds()
  threshold_x = thresholds[0]
  threshold_y = thresholds[1]

  if (threshold_x == 'x' or threshold_y == 'x'):
    return 'x'
  else:
    results = classification_accuracy(threshold_x, threshold_y, C1, C2)
    print_accuracy_results(results)
    create_plot(C1, C2, threshold_x, threshold_y)
    return 'Please enter another set of thresholds, or x to quit.'


def build_tuples(y_valid_01, validate_features):

  Zeros = []
  Eights = []

  for i in range(len(validate_features)):

    if (y_valid_01[i] == 0.0):
      Zeros.append((i, validate_features[i]))
      #plt.scatter(i, validate_features[i], color='blue', marker='d')
      pass
    elif (y_valid_01[i] == 8.0):
      Eights.append((i, validate_features[i]))
      #plt.scatter(i, validate_features[i], color='red', marker='p')
      pass

  return Zeros, Eights

do_all = []

all_sets = [(validate_features, y_valid_01), (testing_features, y_test_01), (train_features, y_train_01)]
for a_set in all_sets:

  C1, C2 = build_tuples(a_set[1], a_set[0])

  do_all.append((C1, C2))


for i in range(0, len(do_all)):

  testing = True
  while (testing == True):
    # Validate
    if (i == 0):
      print("\n", "Validation Set")
    elif (i == 1):
      print("\n", "Testing Set")
    elif (i == 2):
      print("\n", "Training Set")

    message = main(do_all[i][0], do_all[i][1])
    if (message == 'x'):
      testing = False
      print('User Terminated Program')
    else:
      print(message)

```

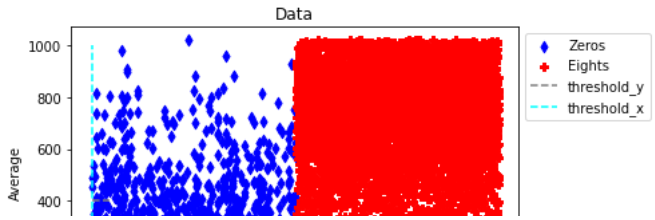
```
 Validation Set
Threshold x must be a number, please enter an integer: 0
Threshold y must be a number, please enter an integer: 400
500
Correct: 474
Incorrect: 26
Total: 500
Classification Accuracy: 0.948
```

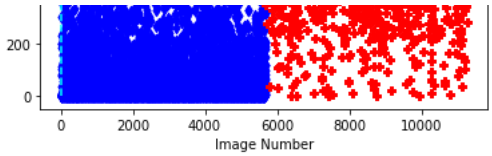



```
Please enter another set of thresholds, or x to quit.

 Validation Set
Threshold x must be a number, please enter an integer: x
Threshold y must be a number, please enter an integer: x
User Terminated Program

 Testing Set
Threshold x must be a number, please enter an integer: 0
Threshold y must be a number, please enter an integer: 400
1954
Correct: 1869
Incorrect: 85
Total: 1954
Classification Accuracy: 0.9564994882292733
```




```
Please enter another set of thresholds, or x to quit.

 Testing Set
Threshold x must be a number, please enter an integer: x
Threshold y must be a number, please enter an integer: x
User Terminated Program

 Training Set
Threshold x must be a number, please enter an integer: 0
Threshold y must be a number, please enter an integer: 400
11274
Correct: 10564
Incorrect: 710
Total: 11274
Classification Accuracy: 0.9370232393116906
```






```
Please enter another set of thresholds, or x to quit.

 Training Set
Threshold x must be a number, please enter an integer: x
Threshold y must be a number, please enter an integer: x
User Terminated Program
```

✓ 0s completed at 8:00 PM