

Assignment 1: Raw Data to Feature Space

Priyanka Budavi
University of North Carolina Greenboro
Email: p_budavi@uncg.edu

1

1 Build your programming environment!

1.1 Install Anaconda 3

Installed Anaconda 3 as the software is easy to use and also it comes with an inbuilt Python, Jupyter Notebook and Spyder packages which is required as a part of our assignment.

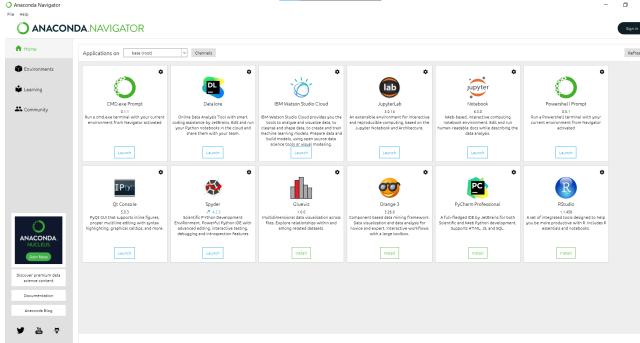


Figure 1: Installed Anaconda 3

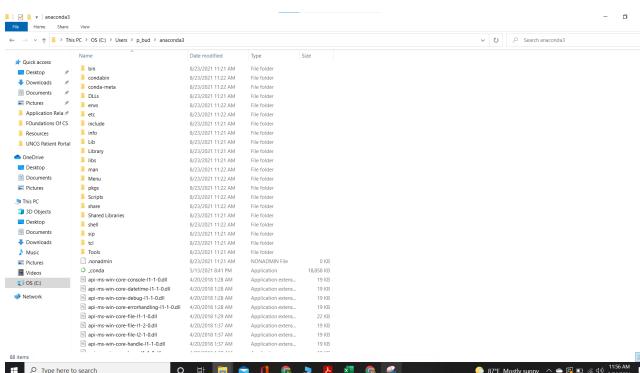


Figure 2: Workspace for the Anaconda 3

1.2 Install the Spyder IDE

The Spyder package was already present in Anaconda 3. We can access this by clicking on the launch button present on anaconda home page.

1.3 Install Jupyter Notebook

Anaconda 3 has Jupyter Notebook package and hence we can access it by clicking on windows and launching the Jupyter Notebook.

1.4 Download OpenCV

I downloaded openCV via Anaconda prompt by using the command `conda install -c conda-forge opencv`. I followed the steps from <https://www.javatpoint.com/opencv-installation-website>.

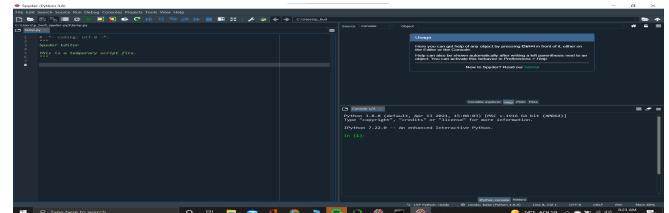


Figure 3: Installed Spyder IDE

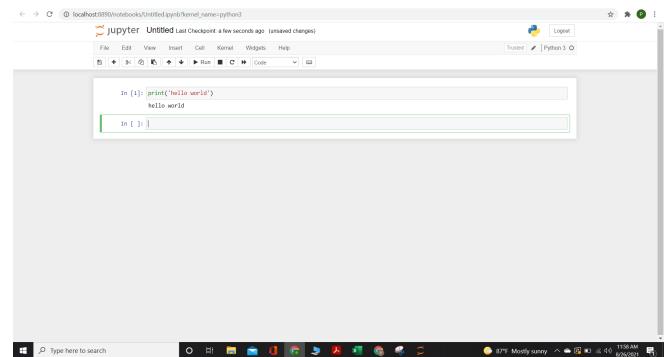


Figure 4: Sample program in Jupyter Notebook

2 Download or generate a fruits and vegetables image dataset!

2.1 Download Tropical Fruits DataSet

Downloaded the Tropical fruits from <http://www.ic.unicamp.br/~rocha/pub/downloads/tropical-fruits-DB-1024x768.tar.gz>. Figure 6 shows the data set stored in my workspace.

2.2 Download the FIDS30 DataSet

Downloaded the dataset from <http://data.vicos.si/datasets/FIDS30/FIDS30.zip>. Figure 7 shows the data set stored in my workspace.

2.3 Choose three fruits

The fruits I am choosing are Banana , Apple and Guava. I chose banana and apple just as single image to see how the data set would be generated because of the background noise. Guava has less background noise so I chose that image to compare all of them when combined together. I have assigned banana as Image0 , apple as Image1 and Guava as Image2. Figures 8-10 show the images of the fruits that I will be working on for the assignment.

Figure 5: Installed OpenCV

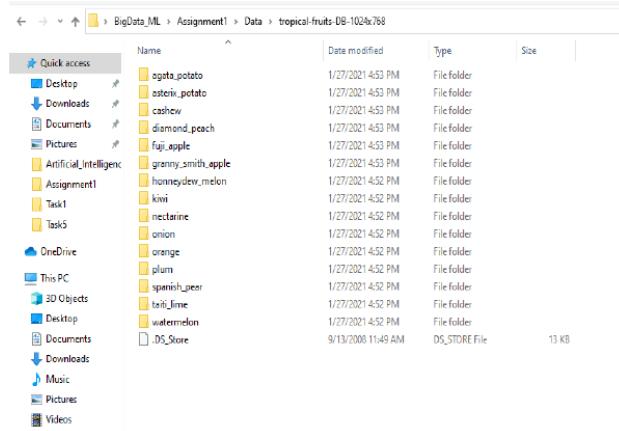


Figure 6: Tropical Fruits DataSet

3 Read your selected images and display them on the environment

3.1 Read RBG values of the selected Images

I wrote a python code using openCV libraries to read the images and display the R, G, B values for Image0 , Image1, Image2. I used CV2 library to read and get the dimensions of the images and matplotlib.pyplot to display the images. Figure 11 shows the code for the same.

3.2 Convert the color images to grayscale and display them while printing their dimensions

I wrote a python code using CV2 library. Firstly, the original images were converted to grayscale by using the function `cvtColor`. Then I derived the dimensions of the converted gray image by using the function `shape`. Figure 12 shows the code for all the fruits. Figure 13 - 15 are the gray images of the chosen fruits.

4 Resize the images to reduce their dimensions!

4.1 Create a function so that the dimensions are divisible by 8

I created a function which takes dimension as a parameter. The code checks if the passed argument is divisible by 8. I wrote an if condition to check if the remainder is equal to zero. If the remainder is zero that means it is divisible.

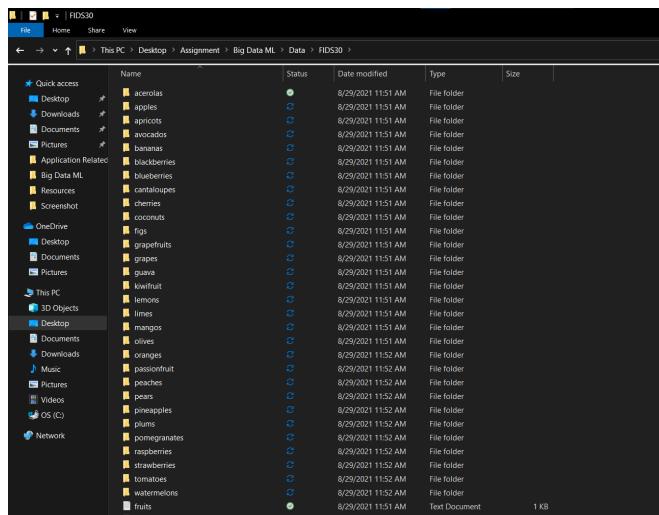


Figure 7: FIDS30 DataSet



wiseGEEK

Figure 8: Image0

by 8. If the dimension is not divisible by 8, I adjust the value such that its a multiple of 8. Check the code for logic. Figure 16 shows the python code.

4.2 Convert the image to standard height 256

Figure 17 shows the code to convert the image to a standard height of 256. I set the standard height to 256 and used a function to get the new width without changing the aspect ratio of the dimension. The new width is rounded off to the nearest value divisible by 8. I performed this task for Image0 , Image1 and Image2 and saved the new dimension images in a different file with Image0 as 256 * 256 , Image1 as 256 * 256 , Image2 as 256 * 256. Refer Figure 17 for the code.

5 Generate block-feature vectors!

5.1 Create non-overlapping feature vector of 8*8 size

I reused the code of Task 4 to convert the image to gray-scale and make its width divisible by 8 with a standard height of 256. Figure 18 shows the non-overlapping code



Figure 9: Image1



Figure 10: Image2

to generate feature vectors of 8×8 size. I created a CSV file to store the pixel values and labeling each Image as 0 for Image0 , 1 for Image1 and 2 for Image2. Image 0 has 64 feature and each feature is a pixel of size 8×8 . Image 0 has 1024 sub-blocks of size 8×8 and same are for Image 1 and Image 2. Figure 18 Shows the code for non-overlapping feature extraction. Figure 19-21 show the CSV files generated.

6 Generate sliding block-feature vectors!

In generating sliding block feature, an image is converted to a gray scale of height 256 and resized width. Image0 is labeled as 0 , Image1 as 1 and Image2 as 2. Each Image CSV files are generated by the code in Figure 22. Image0 has 79930 sub-blocks of size 8*8 and 64 features. Image1 has 620002 sub blocks of size 8*8 and 64 features. Image2 has 62002 subblocks of size 8*8.

Figure 11: Read the images and display their R, G, B values

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
apple_image = cv2.imread('apple.jpg')
orange_image = cv2.imread('orange.jpg')

# Convert to grayscale
apple_gray = cv2.cvtColor(apple_image, cv2.COLOR_BGR2GRAY)
orange_gray = cv2.cvtColor(orange_image, cv2.COLOR_BGR2GRAY)

# Print the dimensions
print("apple dimensions : ", apple_gray.shape)
height = apple_gray.shape[0]
width = apple_gray.shape[1]
channels = apple_gray.shape[2]

# Printing the dimensions
print("apple dimensions : ", dimensions)
print("Image Resolution : ", dimensions)
print("Image Width : ", width)
print("Image Height : ", height)
print("Number of Channels : ", channels)

plt.imshow(apple_gray, cmap='gray')
plt.show()

# Convert to grayscale
orange_gray = cv2.cvtColor(orange_image, cv2.COLOR_BGR2GRAY)
apple_gray = cv2.cvtColor(apple_image, cv2.COLOR_BGR2GRAY)

# Print the dimensions
print("apple dimensions : ", dimensions)
print("Image Resolution : ", dimensions)
print("Image Width : ", width)
print("Image Height : ", height)
print("Number of Channels : ", channels)
```

Figure 12: Grayscale image dimensions

7 Derive statistical descriptors

7.1 Number of Observations

Image0 has 1024 observation and 64 features, Image1 has 1024 observation and 64 features and Image2 has 1024 observation and 64 features where each feature in all images is of size 8 * 8. The number of observation in Image0 is more than Image1 and Image2 while observing the overlapping dataset.

7.2 Mean of the Images and the their plot

To answer the statistical questions, I took the mean of each image in the CSV file. Using the Matplotlib.pyplot I plotted the values of each image on the graph by using red color to banana , blue to apple and green to guava. Figure 23 shows the code for generating the plot. The mean value for banana falls between 217-219 , for apple 138- 139 and for guava its between 121-124. Figure 24 shows the mean plot for overlapping data.

7.3 Standard Deviation of the Images and their plot

I took the standard deviation of the images and plotted a graph against them. Image0 ranges between 38-40 , Image1 ranges between 50-55 and Image2 between 89 -90. These are the values for non overlapping. Figure 26-28 show the required plot and code. Thus I conclude saying that there is an increasing trend between the data sets. Also the co variance between the features is positive that's why is trending up.

7.4 Scatter Plot of the Images

I plotted scatter plot for Image0 by assigning red as the color of the data sets. I selected Feature 20 and Feature



Figure 13: Image 0



Figure 14: Image 1

40 of Image0 to plot the graph. For Image1 I chose green color and selected Feature 10 and Feature 40 from the Image1.csv file. Similarly for Image2 I chose blue and selected Feature 15 and Feature 40 from the Image2.csv file. Figure 29 - 35 show the overlapping and non-overlapping graph of the images.

7.5 Histogram of the images

I took histogram plot for Image0 , Image1 , Image2. For Image0 all the features are present towards the right end and rest other features are skewed for both overlapping and non overlapping. In Image1 the feature are distributed everywhere but its skewed in the middle for both overlapping and non overlapping. Image2 the features are distributed more towards the centre than at the edges.

7.6 Is the dataset imbalanced, inaccurate or incomplete ?

The no of observations in Image0 , Image1 and Image2 for non-overlapping data set are the same. Thus the data set is balanced. But for overlapping data set , the number of observations in Image0 is more than Image1 and Image2 so data is imbalanced . There was no missing data in the



Figure 15: Image 2

```

1 # -*- coding: utf-8 -*-
2
3 # Created on Mon Aug 30 10:23:23 2021
4
5 # Author: Priyanka Budoli
6
7
8 import cv2
9 import numpy as np
10
11 from __future__ import division
12
13 original_image = cv2.imread('Grey_Img0.jpg')
14
15 print("Original Dimensions : ",original_image.shape )
16
17 height = (grayBananas.shape[0])/2
18 width = (grayBananas.shape[1])/2
19
20 print("Halved height : ", height)
21 print("Halved width : ", width)
22
23 #Function for Dividing by 8
24 def dimension_divide(dimension_value):
25     remainder = dimension_value % 8
26     if remainder == 0:
27         dimension_value = dimension_value
28         print("Value divisible by 8 : ", dimension_value)
29     else:
30         dimension_value = dimension_value - (remainder)
31         print("Value not divisible by 8 : ", dimension_value)
32
33     return dimension_value
34
35
36 #Values when both need to be divisible by 8
37 new_width = round(dimension_divide(width))
38 new_height = round(dimension_divide(height))
39
40 #Values when both the height and width is to be divisible by 8
41 print("New Dimension divisible by 8 : ", new_height, new_width )
42
43 image = cv2.resize(original_image, (new_height , new_width))
44
45 cv2.imwrite('Banana_imagediv8.jpg', image)

```

Figure 16: Code for dimensions divisible by 8

CSV file and so it wasn't incomplete. The labels in the csv file were generated correctly hence the data is accurate.

7.7 Is it a trivial data or possibly a big data ?

It is trivial data. The total images that we are considering for analysis is just three. So we have only 3 labels or classes. The number of observation generated from the images is more than the number of features so it is a low dimensional data set. For the data to be a big data it needs to exponentially increase time

7.8 Does it have scalability problem? Are they high dimensional?

It is a low dimensional data. Each feature is of size 8*8. So if the features increase then the scalability also increases. But I don't see high dimensionality and scalability problem with the data set generated from three fruit images.

7.9 Do you need to standardize? Do you need to normalize?

Yes it need to be standardized and normalized. For example, A variable that ranges between 0 and 500 will out-

```
Gray = cv2.imread('gray_fruit.jpg')
print("The original dimensions: ", Gray.shape)
print("The original dimensions: ", Gray.shape[0])
print("The original width : ", Gray.shape[1])
height = Gray.shape[0]
width = Gray.shape[1]
new_fruit_width = 256
new_fruit_height = 256
aspect_ratio = round((height/width))
print("Aspect Ratio : ", aspect_ratio)
new_fruit_width * aspect_ratio = 256
print("Width : ", new_fruit_width)
print("Height : ", new_fruit_height)
return round(new_fruit_width)

def resize(dimensions, value):
    dimension = dimensions[0]
    remainder = dimension % value
    if remainder == 0:
        dimensions[0] = dimension // value
    else:
        dimensions[0] = dimension // value + 1
    return dimensions

def divisible_by_8(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_32(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_256(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_512(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1024(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2048(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_4096(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_8192(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_16384(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_32768(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_65536(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_131072(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_262144(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_524288(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1048576(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2097152(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_4194304(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_8388608(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_16777216(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_33554432(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_67108864(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_134217728(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_268435456(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_536870912(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_107374184(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_214748368(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_429496736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_858993472(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1717986944(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_3435973888(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_6871947776(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_13743895552(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_27487791104(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_54975582208(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_109951164416(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_219902328832(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_439804657664(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_879609315328(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1759218630656(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_3518437261312(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_7036874522624(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_14073749045248(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_28147498090496(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_56294996180992(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_112589992361984(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_225179984723968(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_450359969447936(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_900719938895872(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1801439877791744(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_3602879755583488(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_7205759511166976(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_14411519022333952(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_28823038044667904(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_57646076089335808(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_115292152178671616(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_230584304357343232(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_461168608714686464(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_922337217429372928(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1844674434858755856(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_3689348869717511712(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_7378697739435023424(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1475739547887004688(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2951479095774009376(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_5902958191548018752(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1180591638309603756(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2361183276619207512(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_4722366553238415024(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_9444733106476830048(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_18889466212953660096(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_37778932425907320192(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_75557864851814640384(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_15111572902362928168(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_30223145804725856336(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_60446291609451712672(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_12089258321890342536(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_24178516643780685072(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_48357033287561370144(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_96714066575122740288(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_19342813315024548056(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_38685626630049096112(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_77371253260098192224(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_154742506520196384448(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_309485013040392768896(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_618970026080785537792(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_123794005216157107584(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_247588010432314215168(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_495176020864628430336(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_990352041729256860672(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1980704083458533721344(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_3961408166917067442688(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_7922816333834134885376(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1584563266766826977152(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_3169126533533653954304(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_6338253067067307908608(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1267650613413461581216(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2535301226826923162432(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_5070602453653846324864(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1014120490730769264972(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2028240981461538529944(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_4056481962923077059888(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_8112963925846154119776(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1622592785169230823952(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_3245185570338461647904(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_6490371140676923295808(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_12980742281353846591616(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_25961484562707693183232(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_51922969125415386366464(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_103845938250830772732928(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_207691876501661545465856(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_415383753003323090931712(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_830767506006646181863424(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_166153501201329236372688(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_332307002402658472745376(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_664614004805316945490752(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1329228009610633890981504(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2658456019221267781963008(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_5316912038442535563926016(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_10633824076885071127852032(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_21267648153770142255704064(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_42535296307540284501408128(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_85070592615080569002816256(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_17014118523016113800563256(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_34028237046032227601126512(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_68056474092064455202253024(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_13611294818412891040450648(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_27222589636825782080901296(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_54445179273651564161802592(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_10889035854722732832360584(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_21778071709445465664721168(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_43556143418890931329442336(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_87112286837781862658884672(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_17422457367556372517776944(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_34844914735112745035553888(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_69689829470225490071107776(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_13937965894045098014221552(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_27875931788090196028443104(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_55751863576180392056886208(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_11150372715236078011373416(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_22300745430472156022746832(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_44601490860944312045493664(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_89202981721888624090987328(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_17840596344377248180195656(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_35681192688754496360391312(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_71362385377508992720782624(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_14272477075501798544156528(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_28544954151003597088313056(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_57089908302007194176626112(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_11417981660401438835325224(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_22835963320802877670650448(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_45671926641605755341300896(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_91343853283211510682601792(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_18268770656642302136520384(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_36537541313284604273040768(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_73075082626569208546081536(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_14615016525138401709216312(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_29230033050276803418432624(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_58460066100553606836865248(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_11692013200110721367372596(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_23384026400221442734745192(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_46768052800442885469490384(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_93536105600885770938980768(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_187072211201771541877961536(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_374144422403543083755923072(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_748288844807086167511846144(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_149657768961417233502369288(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_299315537922834467004738576(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_598631075845668934009477152(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_119726215689133786801895304(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_239452431378267573603790608(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_478904862756535147207581216(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_957809725513070294415162432(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_191561945102614058883032464(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_383123890205228117766064928(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_766247780410456235532129856(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_153249556082091247106425772(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_306499112164182494212851544(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_612998224328364988425703088(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1225996448656729976851406176(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2451992897313459953702812352(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_4903985794626919907405624704(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_9807971589253839814811249408(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1961594317850767962962249816(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_3923188635701535925924499632(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_7846377271403071851848999264(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_15692754542806143703697998528(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_31385509085612287407395997056(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_62771018171224574814791994112(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_125542036342449149629583988224(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_251084072684898299259167976448(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_502168145369796598518335952896(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_100433629073959319703667905576(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_200867258147918639407335811152(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_401734516295837278814671622304(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_803469032591674557629343244608(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_160693806518335011525868648916(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_321387613036670023051737297832(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_642775226073340046103474595664(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1285550452146680092206949191328(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2571100904293360184413898382656(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_5142201808586720368827796765312(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_10284403617173440737655993530624(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2056880723434688147531198706128(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_4113761446869376295062397412256(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_8227522893738752590124794824512(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_16455045787477505802495897649024(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_32910091574955011604981795298048(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_65820183149850023209963590596096(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_131640366299700464019927811192192(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_263280732599400928039855622384384(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_526561465198801856079711244768768(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_1053122930397603712159424895337368(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2106245860795207424318849790674736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_42124917215904148486376995813494736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_842498344318082969727539916269894736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_16849966886361659394550783245397894736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_336999337727233187891015664907957894736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_67399867545446637578203132981591577894736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_13479973509089327515640626596318315577894736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_2695994701817865503128125319263663115577894736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_539198940363573100625625063852732623115577894736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_107839788072714620125125012770546524623115577894736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_21567957614542924025025002554109304924623115577894736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_4313591522908584805005000510821860984923115577894736(dimensions, value):
    if dimensions[0] % value == 0:
        return True
    else:
        return False

def divisible_by_862718304581716960100100052164372196989
```

Figure 17: Code to set standard height to 256

```
function for dividing by 4
def divide_dimensions(dimensions,value):
    remainder = dimensions % 4
    if remainder == 0:
        dimensions_value = 0
        dimensions_value += value % 4
    else:
        dimensions_value = dimensions % 4
        dimensions_value += value % 4 - remainder
    return dimensions_value

#values who both need to be divisible by 8
new_fruit_size = width //divisible_by_8
new_height_size = height //divisible_by_8

#values who both the height and width is to be divisible by 8
print("New Dimension divisible by 8 : ", new_height_size, new_fruit_size)

image_resized = cv2.resize(image_gray, (new_height_size , new_fruit_size))
cv2.imwrite('fruit_resized.jpg', image_resized)

#Code for generating non overlapping blocks
dd = round((new_height_size) / (new_fruit_size)/8)
print(dd)
for i in range(0, dd):
    for j in range(0, dd):
        tmp = image_resized[ :i*8, :j*8]
        print(tmp)
        flat_image[8*i:8*(i+1), 8*j:8*(j+1)] = tmp.flatten()
        k = k + 1

feature_space = pd.DataFrame(flat_image)
feature_space.to_csv('image1.csv', index=False)
```

Figure 18: Non-Overlapping feature block

weigh a variable that ranges between 0 and 1. Using these variables without standardization/normalization will give the variable with the larger range weight of 500 in the analysis. Thus its necessary the variables fall in the same normal distribution curve to receive good accuracy.

7.10 How do they affect the data characteristics?

The data characteristics which is volume , velocity and variety seems to have no effect on the dataset.

8 Construct a feature space!

I merged the Image0 and Image1 to produce a concatenated file i.e Image01.csv. Figure 42 shows the image of the code which merges and uses random function to distribute the data in the excel randomly. The data is randomized row wise to get jumble the data. Similary the Image012.csv was obtained by concatenating Image01.csv with Image2. Did the same for overlapping dataset. Figure 43-44 show the merged data set.

9 Display sub spaces

9.1 Plot two dimension feature space

I assigned Image0 , Image1 and Image2 with red , green and blue color to plot the scatter. Then I chose two features from each image to obtain a 2D feature space. I chose feature20 and feature 40 from Image0 and Image1 .feature

Figure 19: Non-Overlapping Banana with label 0

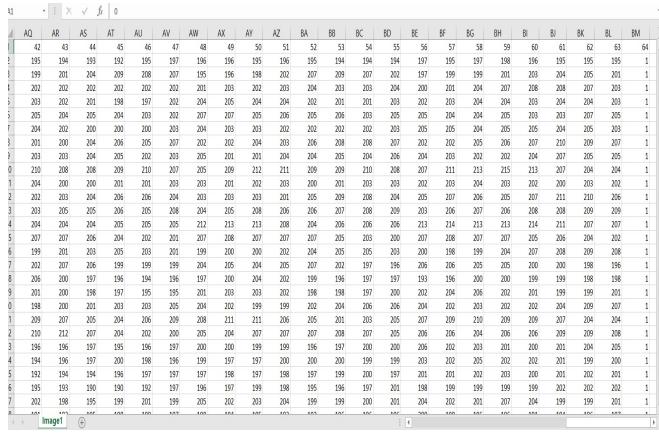


Figure 20: Non-Overlapping apple with label 1

10 and feature 40 of Image0 and Image2 and feature 25
 30 from Image1 and Image2. I followed the same steps to
 obtain the Overlapping feature space scatter plot. Figure
 45 - 51 show the images of the 2D feature space of all
 Images and the respective code.

9.2 Plot three dimensional feature space

I selected feature 10 , feature 20 , feature 40 from Image0 , Image1 and Image2 to plot a three dimensional scatter plot. Figure 52-53 shows the plot and code for the same. The points are overlapping on each other and thus make it difficult to classify them. Thus it is inseparable classes.

9.3 Discuss the observation from the figures

We observe that the classes are overlapping on each other this makes the class not separable easily. Even though towards the left we see two classes which looks easy to distinguish but that alone will not help. So the complexity of classifying them is high.

10 Read multiple files from a folder

I reused the code from the task 6 and wrote a function which will read all the images and create their respective CSV files. Figure 54 -55 show the code and the CSV files generated by reading all of them.

Figure 21: Non-Overlapping Guava with label 2

```

Created on Fri Sep 10 22:46:22 2021
@author: p_bud

import cv2
import numpy as np
import pandas as pd
import os
import random

read_fruit = cv2.imread('Image0.jpg')
new_fruit_width = 256
fruit_gray = cv2.cvtColor(read_fruit, cv2.COLOR_BGR2GRAY)
height, width = fruit_gray.shape
height -= 100

def width_resize(height, width):
    print("Original Height : ", height)
    print("Original Width : ", width)
    aspect_ratio = width / height
    new_height = int(aspect_ratio * 256)
    print("My width = ", new_fruit_width)
    return new_height, new_fruit_width

def dimension_divs(height, width):
    new_fruit_width = width_resize(height, width)
    remainder = height % new_fruit_width
    if remainder == 0:
        new_fruit_width = new_fruit_width - remainder
    else:
        new_fruit_width = new_fruit_width - remainder

    return new_fruit_width

#Values when both need to be divisible by 8
new_fruit_width = dimension_divs(height, width)
print("Standard Height 256 division, std_height, new_fruit_width")
resized_fruit = cv2.resize(fruit_gray, dsize=(new_fruit_width, std_height))
heightd, widthd = resized_fruit.shape

dd = round((heightd - 7) / (widthd - 7))
flat_image = np.full((dd, dd), 0)

for i in range(0, heightd - 7, 1):
    for j in range(0, widthd - 7, 1):
        tmp = resized_fruit[i:i + 7, j:j + 7]
        tmp = tmp[0:dd, 0:dd] - tmp.flatten()
        k = k + 1
        flat_image[k // dd, k % dd] = tmp

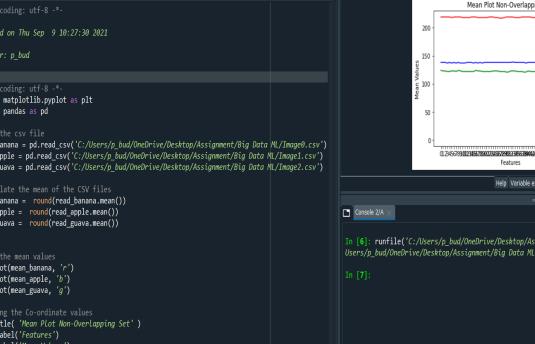
feature_space = pd.DataFrame(flat_image)
feature_space.to_csv('Image0_Overlap.csv', index=False)

```

Figure 22: Overlapping feature block

11 Describe the effects of block size on the dimensionality of the feature space and the number of vectors in the domain

The images that we generated are of size 8*8 each pixel. Thus the feature vector is of size 64. Suppose we decrease the size by 4*4 then the number of observations will increase and if we increase by 16 * 16 the features will increase and this leads to the reduction in the number of observation. Hence I conclude that this would lead to high dimension or low dimension issue which doesn't give accurate results.



The screenshot shows a Jupyter Notebook interface with several code cells and a figure. The code cells are numbered 1 through 13, containing Python code for reading CSV files and plotting mean values. The figure, titled 'Mean Plot Non-Overlapping Set', displays four horizontal lines representing mean values for different categories over 100 features. The y-axis ranges from 0 to 200, and the x-axis ranges from 0.256200 to 0.256250.

```
#C:\Users\p_bud\OneDrive\Desktop\Assignment\Big Data ML\Mean_Plot.py
# General_FeatureBook_Vectors_OverDeveloping.py | utils.py* | General_FeatureBook_Vectors_OverDeveloping.py | Mean_Plot.py
1 #-*- coding: utf-8 -*-
2 #
3 # Created on Thu Sep 9 10:27:30 2021
4 #
5 #Author: p_bud
6 #
7 #
8 #-*- coding: utf-8 -*-
9 import matplotlib.pyplot as plt
10 import pandas as pd
11
12 #Read the csv file
13 read_banana = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Image0.csv')
14 read_apple = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Image1.csv')
15 read_guava = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Image2.csv')
16
17 #Calculate the mean of all the CSV files
18 mean_banana = round(read_banana.mean())
19 mean_apple = round(read_apple.mean())
20 mean_guava = round(read_guava.mean())
21
22
23 #Plot the mean values
24 plt.plot(mean_banana, 'r')
25 plt.plot(mean_apple, 'b')
26 plt.plot(mean_guava, 'g')
27
28 #Setting the co-ordinate values
29 plt.title('Mean Plot Non-Overlapping Set')
30 plt.xlabel('Features')
31 plt.ylabel('Mean Values')
32
33
```

In [6]: runfile('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/')

In [7]:

Figure 23: Mean Plot

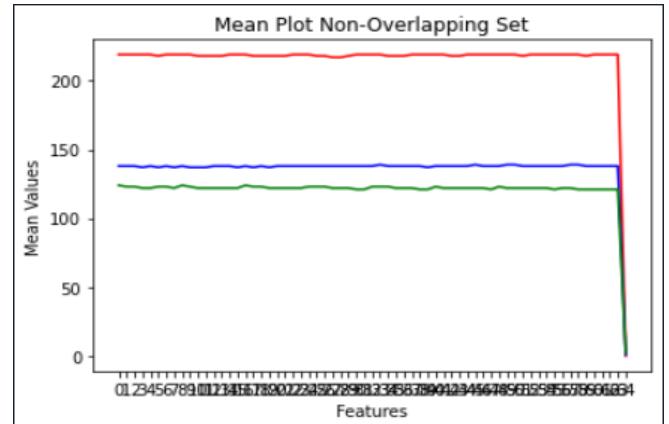


Figure 24: Mean Plot Non-Overlapping

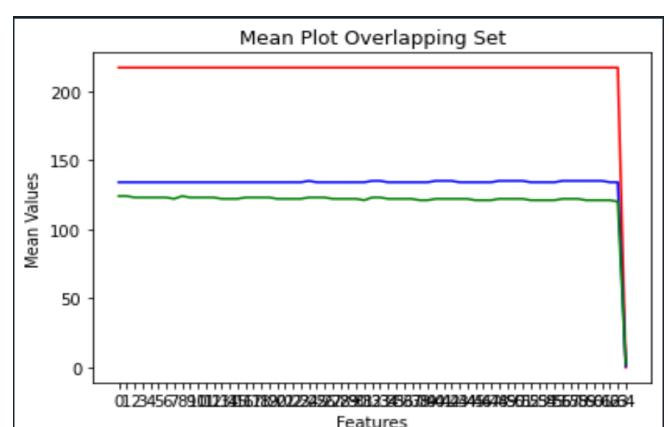


Figure 25: Mean Plot Overlapping

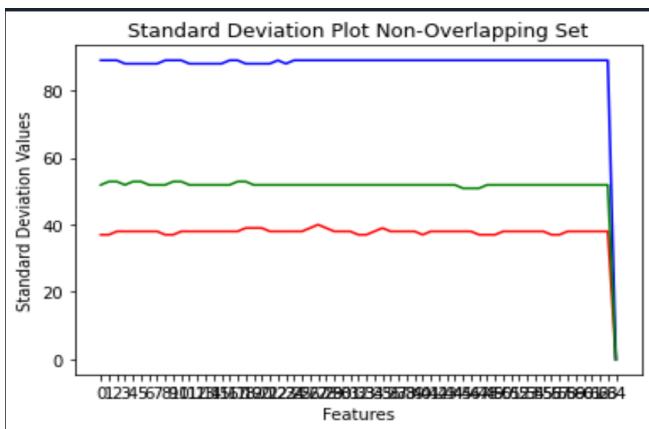


Figure 26: Standard Deviation Plot Non Overlapping

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Sep 13 12:00:09 2021
4 @author: p_bud
5 """
6
7 import matplotlib.pyplot as plt
8 import pandas as pd
9
10 #read the csv file
11 read_fruit = pd.read_csv('Image0.csv')
12
13 f20 = read_fruit['20']
14 f40 = read_fruit['40']
15
16
17
18 plt.scatter(f20, f40, color ='red', s=1)
19 plt.show()

```

Figure 29: Scatter Plot Code

```

# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import pandas as pd

#read the csv file
read_banana = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Image0.csv')
read_apple = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Image1.csv')
read_guava = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Image2.csv')

#calculate the mean of the CSV files
std_banana = round(read_banana.std())
std_apple = round(read_apple.std())
std_guava = round(read_guava.std())

print(" Banana ", std_banana)
print(" Apple ", std_apple)
print(" Guava ", std_guava)

#Plot the mean values
plt.plot(std_banana, 'r')
plt.plot(std_apple, 'b')
plt.plot(std_guava, 'g')

#Setting the Co-ordinate values
plt.title('Standard Deviation Plot Non-Overlapping Set')
plt.xlabel('Features')
plt.ylabel('Standard Deviation Values')

```

Figure 27: Standard Deviation Non Overlapping Code

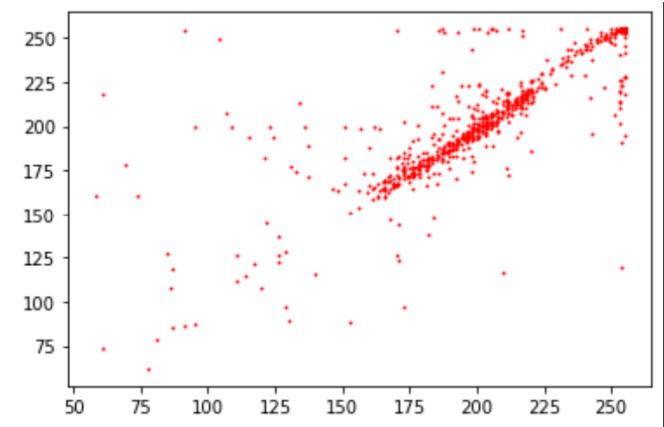


Figure 30: Non-Overlapping Scatter Plot Banana Feature 20 and Feature 40

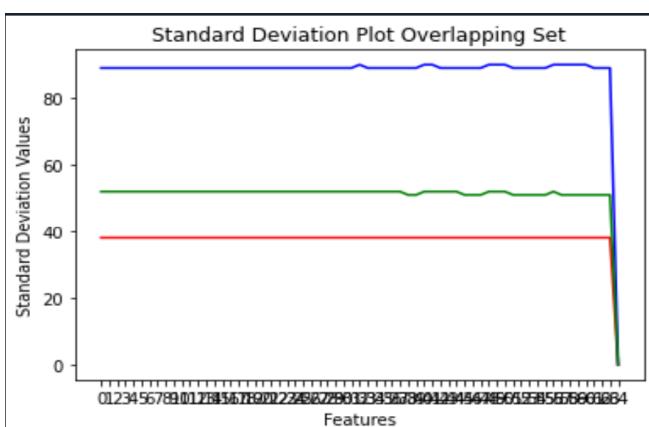


Figure 28: Standard Deviation Overlapping Plot

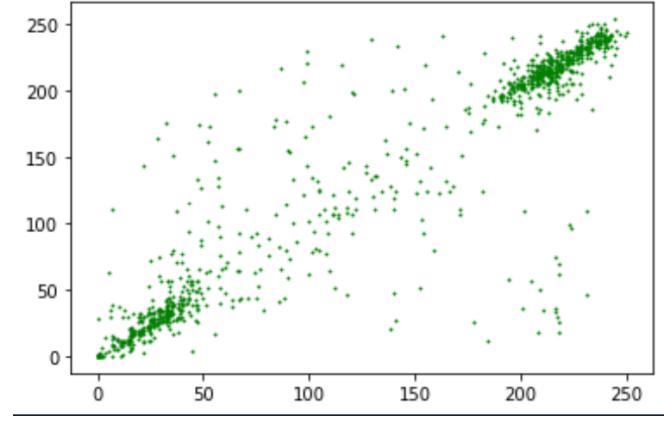


Figure 31: Non-Overlapping Scatter Plot apple Feature 10 and Feature 40

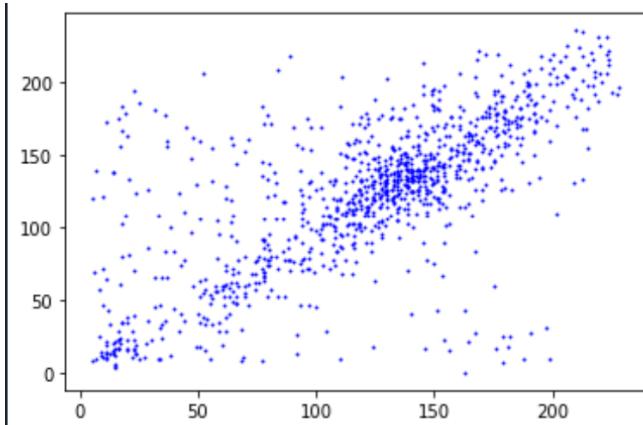


Figure 32: Non -Overlapping Scatter Plot Guava Feature 15 and Feature 40

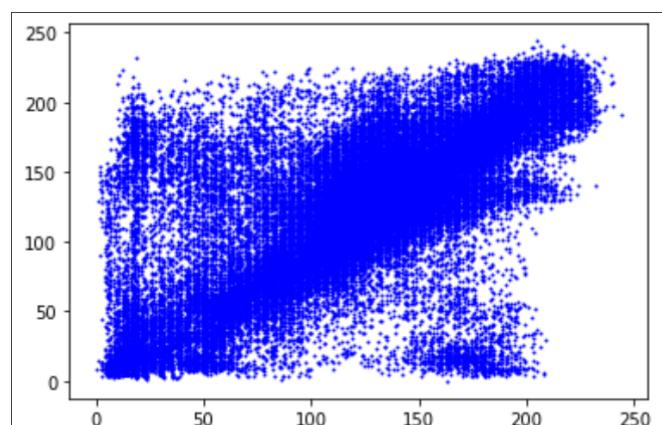


Figure 35: Overlapping Scatter Plot Guava Feature 15 and Feature 40

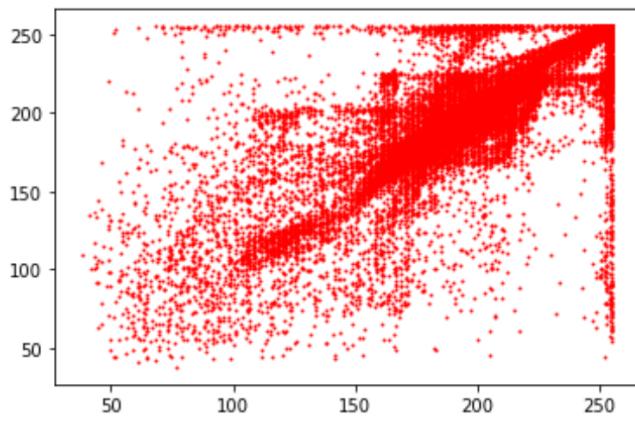


Figure 33: Overlapping Scatter Plot Banana Feature 20 and Feature 40

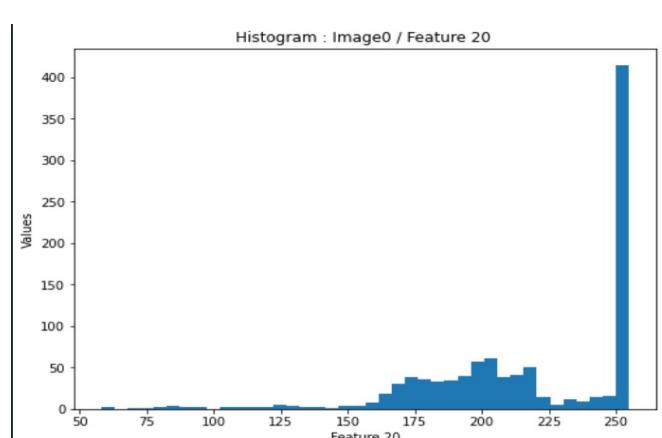


Figure 36: Histogram of Feature 20 for Image0

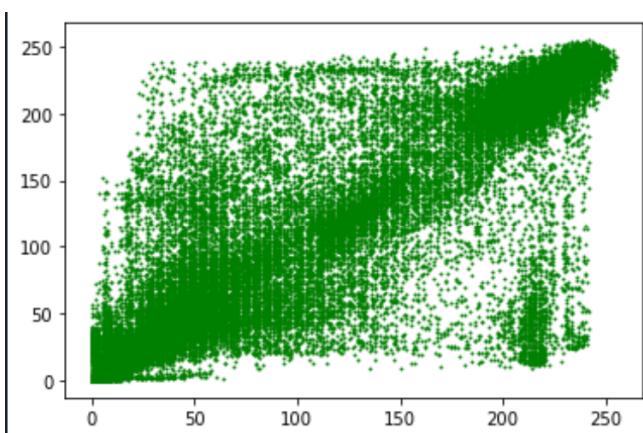


Figure 34: Overlapping Scatter Plot apple Feature 10 and Feature 40

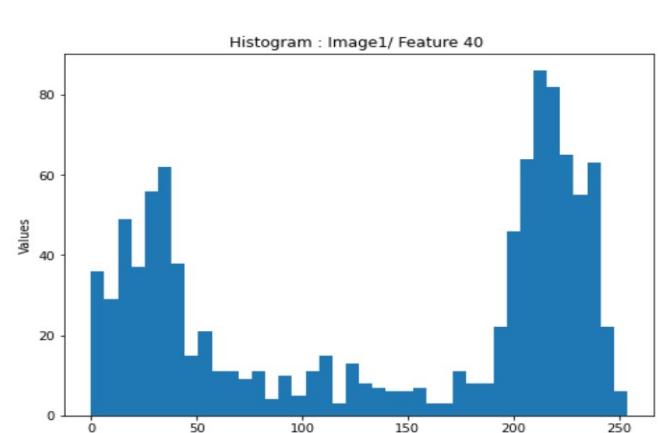


Figure 37: Histogram of feature 40 for Image1

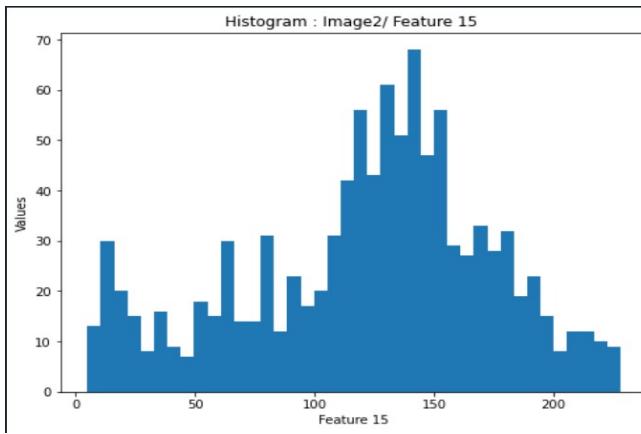


Figure 38: Histogram of Feature 15 for Image2

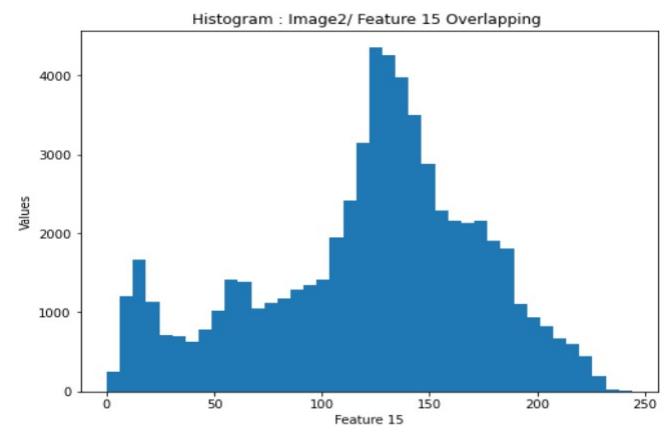


Figure 41: Histogram of feature 15 for Image2: Overlap

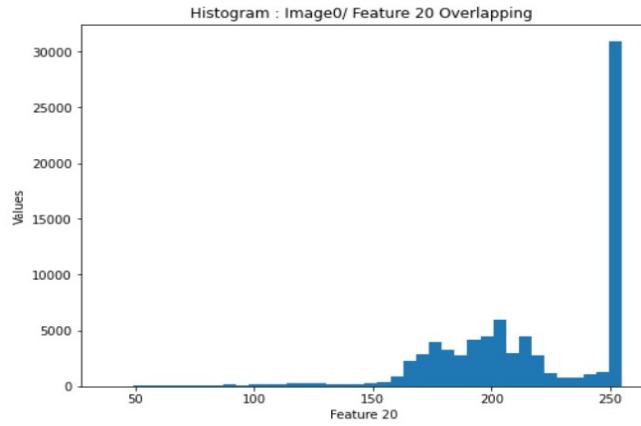


Figure 39: Histogram of feature 20 for Image0: Overlap



Figure 42: Merge and Randomize

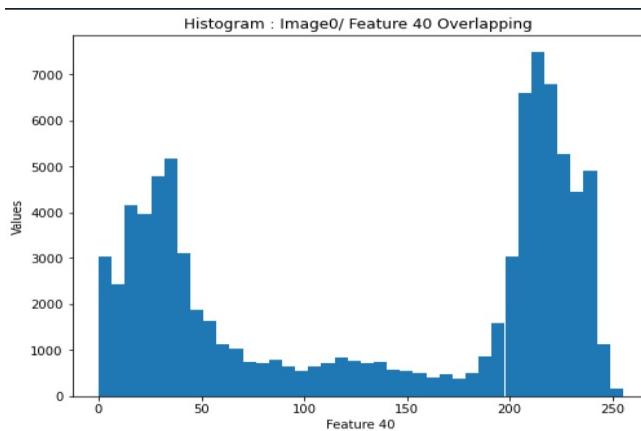


Figure 40: Histogram of feature 40 for Image1: Overlap

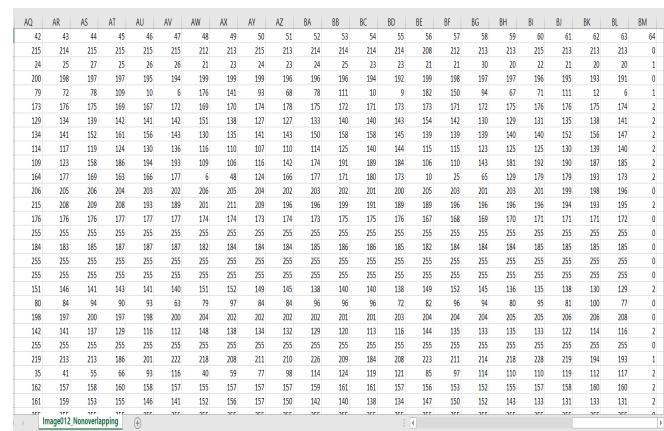


Figure 43: Feature Space of Non-Overlapping DataSet

AQ	AR	AS	AT	AV	AW	AX	AY	AZ	BA	BB	BC	BD	BE	BF	BG	BH	BI	BJ	BK	BL	BM		
42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	
215	214	215	215	215	215	212	213	215	215	214	214	214	214	208	212	213	213	215	213	213	213	0	
24	25	27	25	26	26	21	23	24	23	24	25	23	23	21	21	30	20	22	21	20	20	1	
200	198	197	197	195	194	194	199	199	196	196	196	194	192	199	198	197	197	196	195	193	191	0	
79	72	78	109	10	6	176	141	93	68	78	111	10	9	9	182	150	94	67	71	111	12	6	1
173	176	175	169	167	172	169	170	174	176	175	172	171	173	173	171	172	175	176	176	175	174	2	
129	134	139	142	141	142	151	138	127	127	133	140	140	143	154	142	130	129	131	135	138	141	2	
134	141	152	161	156	143	130	135	141	143	150	158	158	145	138	139	140	140	152	156	147	1		
114	117	119	124	130	136	116	110	107	110	114	125	140	144	115	115	123	125	125	130	139	140	1	
109	123	158	186	194	194	169	109	106	116	142	174	191	189	184	106	110	143	181	192	190	187	185	
164	177	169	163	166	177	6	48	124	166	177	171	180	173	10	25	65	129	179	179	193	173	1	
206	205	206	204	203	202	206	205	204	202	202	203	202	201	200	205	203	201	201	199	198	196	0	
215	208	209	208	193	189	201	211	209	196	196	199	191	189	189	196	196	196	196	194	193	195	2	
176	176	177	177	177	174	174	173	175	175	176	167	168	169	170	171	171	171	172	172	172	0		
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0	
184	183	185	187	187	187	182	184	184	184	185	186	186	185	182	184	184	184	185	185	185	185	0	
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0	
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0	
151	146	141	143	141	140	151	152	149	145	145	140	140	138	149	152	145	145	136	135	138	130	129	
80	84	94	90	93	63	79	97	84	84	96	96	72	82	96	94	80	95	81	100	77	0		
198	197	200	197	198	198	200	204	202	202	202	201	201	203	204	204	204	205	205	206	206	208	0	
142	141	137	129	116	112	148	138	134	132	129	120	113	116	144	135	133	135	133	122	114	116	2	
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0	
219	213	213	216	201	222	218	208	211	210	216	209	204	208	223	211	214	218	218	219	194	193	1	
35	41	55	66	93	116	40	59	77	98	114	124	119	121	85	97	114	110	110	119	112	117	2	
162	157	158	160	158	157	155	157	157	159	161	161	157	156	153	152	155	157	158	160	160	160	2	
161	159	153	146	141	152	150	142	140	138	134	147	150	152	143	143	131	133	131	131	133	131	2	

Figure 44: Feature Space of Overlapping Data Set

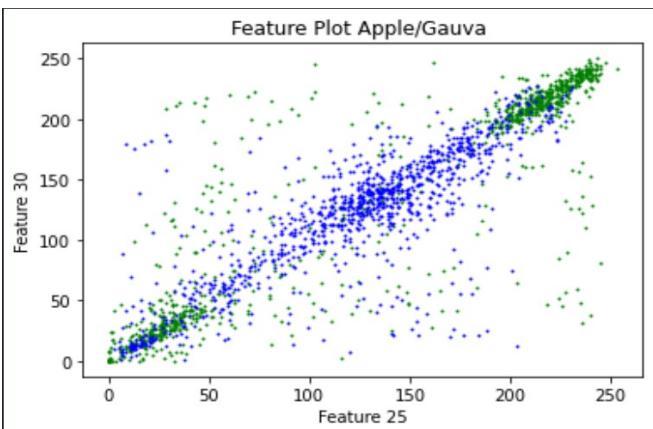


Figure 47: Feature Space of Image1 and Image2

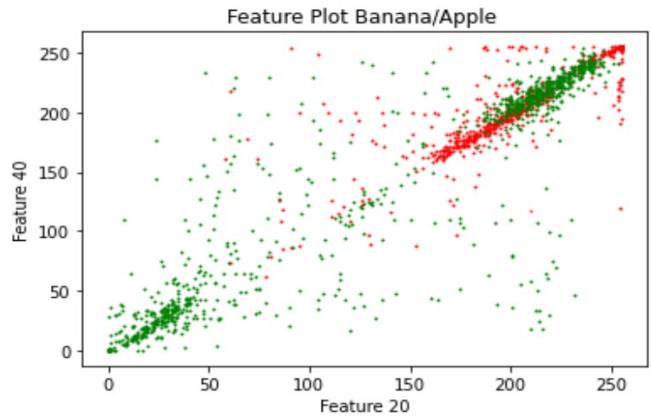


Figure 45: Feature Space of Image0 and Image1

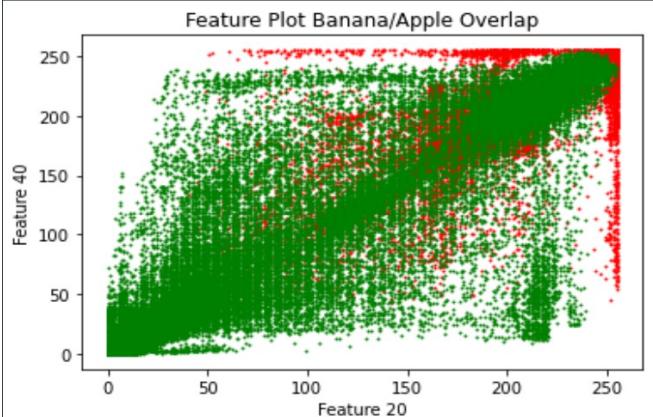


Figure 48: Overlapping Feature Space of Image0 and Image1

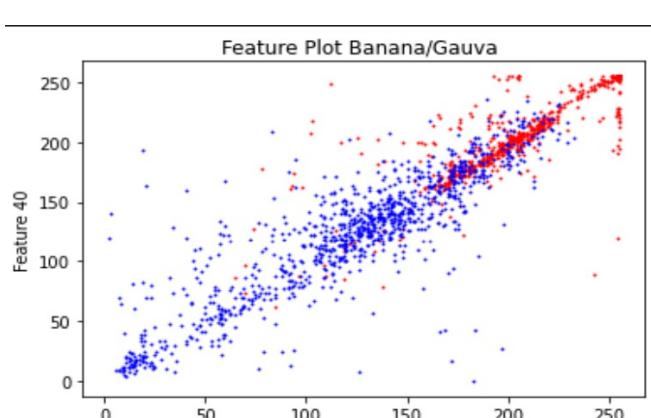


Figure 46: Feature Space of Image0 and Image2

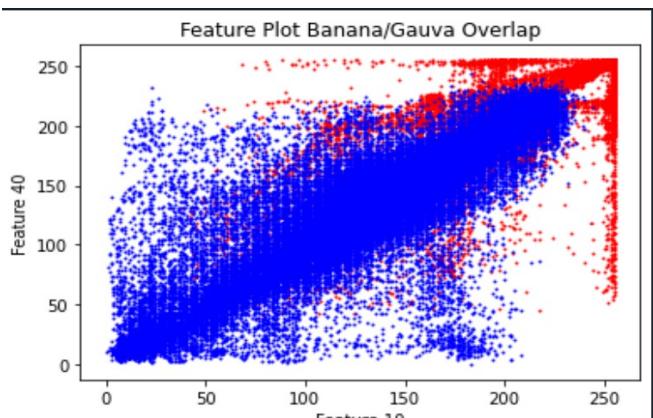


Figure 49: Overlapping Feature Space of Image0 and Image2

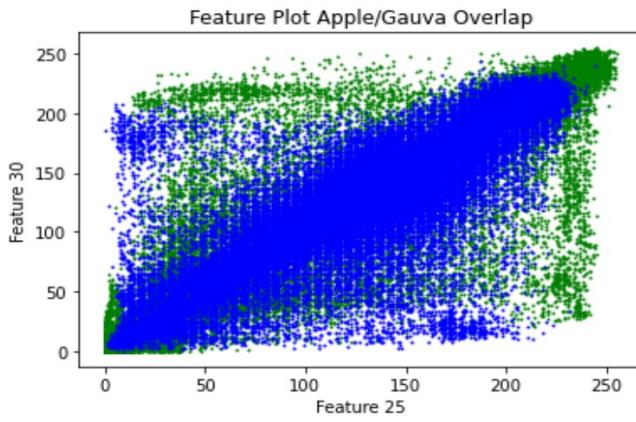


Figure 50: Overlapping Feature Space of Image1 and Image2

```

3   Created on Wed Sep 15 14:22:14 2021
4   @author: p_bud
5   """
6
7
8   import matplotlib.pyplot as plt
9   import pandas as pd
10
11
12 | read_banana = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/CSV files/Image0.csv')
13 | read_apple = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/CSV files/Image1.csv')
14 | read_gauava = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/CSV files/Image2.csv')
15 |
16 | f10 = read_banana['10']
17 | f20 = read_banana['20']
18 | f40 = read_banana['40']
19 |
20 | A10 = read_apple['10']
21 | A20 = read_apple['20']
22 | A40 = read_apple['40']
23 |
24 | G10 = read_gauava['10']
25 | G20 = read_gauava['20']
26 | G40 = read_gauava['40']
27 |
28 | # Creating figure
29 | fig = plt.figure(figsize = (40, 7))
30 | ax = plt.axes(projection = "3d")
31 |
32 | # Creating plot
33 | ax.scatter3D(f10, f20, f40, color = "red")
34 | ax.scatter3D(A10, A20, A40, color = "green")
35 | ax.scatter3D(G10, G20, G40, color = "blue")
36 |
37 | plt.title("3D Scatter Plot of Banana, Apple, Guava")
38 | plt.xlabel('Feature 10')
39 | plt.xlabel('Feature 20')
40 | plt.xlabel('Feature 40')
41 | ax.set_zlabel('Feature 40')
42 | plt.show()

```

Figure 53: Code for 3D scatter plot

```

1  # -*- coding: utf-8 -*-
2  """
3   Created on Tue Sep 14 10:24:31 2021
4   @author: p_bud
5   """
6
7
8   import matplotlib.pyplot as plt
9   import pandas as pd
10
11
12 #read the csv file
13 read_banana = pd.read_csv('Image0.csv')
14 read_apple = pd.read_csv('Image1.csv')
15 read_gauava = pd.read_csv('Image2.csv')
16
17
18 f20 = read_banana['20']
19 f40 = read_banana['40']
20
21 G20 = read_apple['20']
22 G40 = read_apple['40']
23
24 plt.scatter(f20, f40, color =['red'], s=1)
25 plt.scatter(G20, G40, color =['green'], s=1)
26
27 plt.title("Feature Plot Banana/Apple ")
28 plt.xlabel('Feature 20')
29 plt.ylabel('Feature 40')
30 plt.show()
31
32

```

Figure 51: Code for 2D scatter plot

```

def read_multiple_files( image , filename):
    fruit_gray = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    height , width = fruit_gray.shape
    print("Dimension of Gray image : ", height , width)
    new_fruit_width = width //height, width)

    #values when both the height and width is to be divisible by 8
    print("New Dimension divisible by 8 : " , std_height , new_fruit_width )
    image_resized = cv2.resize(fruit_gray , dsize = (std_height , new_fruit_width))
    heightD , widthD = image_resized.shape
    dd = heightD * (heightD) // 64
    print(dd)
    flat_image = np.full((dd , 65) , 1)
    k=0
    for i in range(0 , heightD , 8):
        for j in range(0 , widthD , 8):
            tmp = image_resized[i : i+8 , j:j+8]
            print(tmp)
            flat_image[k:64] = tmp.flatten()
            k = k + 1

    feature_space = pd.DataFrame(flat_image)
    feature_space.to_csv(filename, index=False)
    return feature_space

folder_path = 'C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Priyanka_Budovi_Submission_ML/Data/FIDS30/apples'
images = []
for imagename in os.listdir(folder_path):
    img = cv2.imread(os.path.join(folder_path,imagename))
    if img is not None:
        images.append(img)
        splitted_filename = imagename.split('.')
        rename_csvfile = 'AppleDataset'+splitted.filename[0]+'.csv'
        read_multiple_files(img,rename_csvfile)

```

Figure 54: Code for reading multiple files

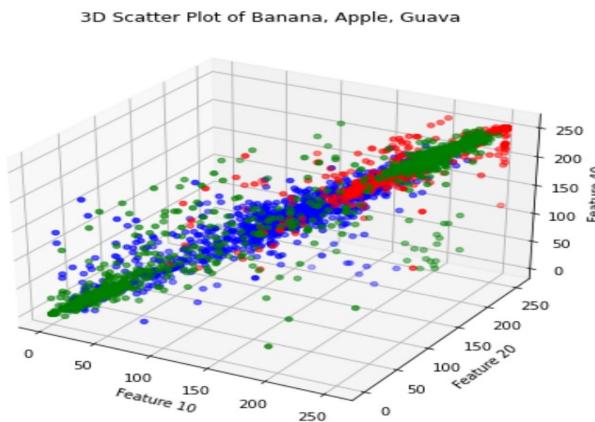


Figure 52: Three Dimensional feature space

Name	Status	Date modified	Type	Size
AppleDataset0	●	9/18/2021 5:17 PM	Microsoft Excel Co...	135 KB
AppleDataset1	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset2	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset3	●	9/18/2021 5:17 PM	Microsoft Excel Co...	135 KB
AppleDataset4	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset7	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset8	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset9	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset11	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset12	●	9/18/2021 5:17 PM	Microsoft Excel Co...	135 KB
AppleDataset13	●	9/18/2021 5:17 PM	Microsoft Excel Co...	135 KB
AppleDataset14	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset15	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset16	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset17	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset19	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset20	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset22	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset23	●	9/18/2021 5:17 PM	Microsoft Excel Co...	271 KB
AppleDataset24	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset25	●	9/18/2021 5:17 PM	Microsoft Excel Co...	134 KB
AppleDataset26	●	9/18/2021 5:17 PM	Microsoft Excel Co...	135 KB
AppleDataset28	●	9/18/2021 5:17 PM	Microsoft Excel Co...	134 KB
AppleDataset32	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset34	●	9/18/2021 5:17 PM	Microsoft Excel Co...	136 KB
AppleDataset35	●	9/18/2021 5:17 PM	Microsoft Excel Co...	271 KB
AppleDataset38	●	9/18/2021 5:17 PM	Microsoft Excel Co...	271 KB

Figure 55: Multiple CSV files are created

12 ASSIGNMENT 2 : FEATURE SPACE TO A CLASSIFIER

12.1 Split the data into 80:20 and save the file

I used the dataset created in the previous assignment and split them into 80:20 ratio using the code and later saved the files into train dataset and test data set with respective file names. Figure 56 show the code for splitting the data into training and testing dataset.

```

1  # -*- coding: utf-8 -*-
2
3  Created on Wed Oct 20 14:42:25 2021
4
5  @author: p_bud
6
7
8  import pandas as pd
9  import numpy as np
10
11  # Read a Feature space
12  input_data = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Prityanka_Budov1_Submission_ML/Data/CSV files/Image01.csv', header=None)
13
14  #Store the value of the CSC into X
15  X = input_data
16  row, col = X.shape
17
18  #Split the data into 80% - Training Dataset
19  I1 = round(row*0.8)
20  X1 = np.array(X)
21
22  #remaining in the Testing Dataset
23  I1 = row-I1
24
25  #Training and Testing Dataset
26  X_train = X1[0:I1,:]
27  X_test = X1[I1:row,:]
28
29  #NonOverlapping dataset
30  data_train = pd.DataFrame(X_train)
31  data_train.to_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Prityanka_Budov1_Submission_ML/Assignment2/Data/Train_Image01_NonOverlap.csv')
32
33  #Test NonOverlapping dataset
34  data_test = pd.DataFrame(X_test)
35  data_test.to_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Prityanka_Budov1_Submission_ML/Assignment2/Data/Test_Image01_NonOverlap.csv', index=False)
36
37

```

Figure 56: Code to split the data into 80:20 training and testing dataset

12.2 Select two features in each category of training and testing datasets and plot their histograms to see if they follow the same distribution

I selected two features from each category of the dataset and tried to plot histogram for each feature. If you observe the train and test dataset, the two features follow the same gaussian curve. Then I tried to calculate the mean of the two features to verify the same. Figure 57-61 shows the images of the histogram representing different category of the dataset and how they are distributed. Figure 62-65 show the mean values of different category. I chose the features such that they dont differ much in the values.

```

Created on Wed Oct 20 22:28:29 2021
@authors: p_bud
**_
import matplotlib.pyplot as plt
import pandas as pd

#read the file
input_data = pd.read_csv("C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Prityanka_Budov1_Submission_ML/Assignment2/Data/Training Dataset/Train_I
f61 = input_data['61']
f62 = input_data['62']

nbins = 40
plt.title('Histogram of Non Overlapping Feature 61 Train data set/image01')
plt.hist(f61, nbins, color='g', edgecolor='k')
plt.text(f61.mean(), 250, 'mean')
plt.title('Histogram of Non Overlapping Feature 62 Train data set/image01')
plt.hist(f62, nbins, color='g', edgecolor='k')
plt.text(f62.mean(), 250, 'mean')
plt.show()

plt.title('Histogram of Non Overlapping Feature 61 Test data set/image01')
plt.hist(f61, nbins, color='g', edgecolor='k')
plt.text(f61.mean(), 250, 'mean')
plt.title('Histogram of Non Overlapping Feature 62 Test data set/image01')
plt.hist(f62, nbins, color='g', edgecolor='k')
plt.text(f62.mean(), 250, 'mean')
plt.show()

```

Figure 57: Code for histogram

Non Overlapping Image01 61/62 Feature Train and Test Dataset

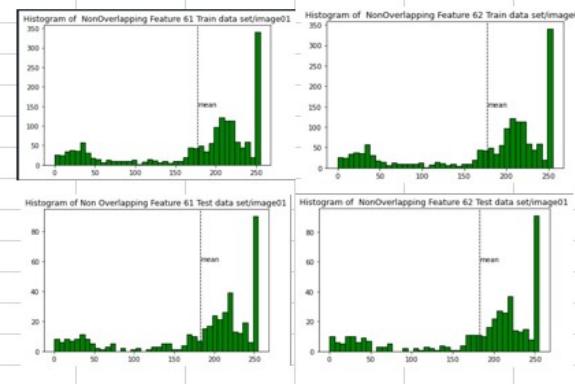


Figure 58: Combined Non Overlapping Train/Test dataset of Image01

Overlapping Image01 61/62 Feature Train and Test Dataset

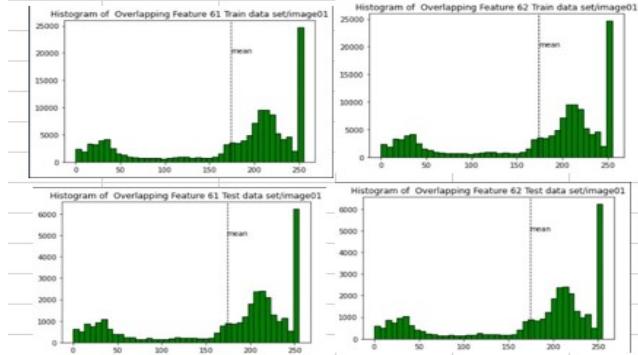


Figure 59: Combined Overlapping Train/Test dataset of Image01

12.3 Use the same two features to generate scatter plots for each category of training and testing datasets

Figure 66 shows the code to generate the scatter plots for feature 61 and feature 62 of Image01 and Image012 for both overlapping and non overlapping datasets. Figure 67-74 shows the scatter plots of all category of dataset.

13 Implement a regression-based model

13.1 Implement and train lasso regression using the training sets of the overlapping and non-overlapping feature vectors (feature spaces)

I used the formula from the text book to calculate the co-efficients to implement the lasso regression. I used the training data-set to train the model and calculated the co-efficients. The value of coefficients is then multiplied to the training dataset to get the predicted values. The values obtained are in the range from 0 to 1 (Two class classifier). Now use the testing data-set and multiply the data with the co-efficients obtained previously and predict the values. Also, I tried to alter the lambda value till the accuracy is better. Lambda for non overlapping data set ranges from 0.1 to 150 and the accuracy is almost same. But after 150

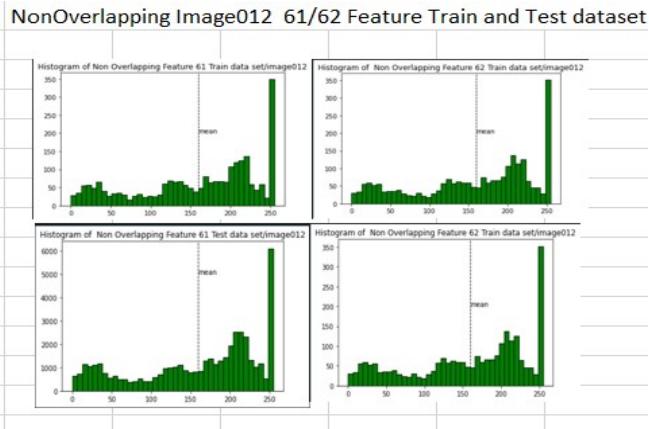


Figure 60: Combined Non Overlapping Train/Test dataset of Image012



Figure 61: Combined Overlapping Train/Test dataset of Image012

the accuracy decreases. I did the same for both overlapping and non overlapping data-set. The lambda value used while getting the co-efficient is 0.1. Figure 75-77 shows the code for implementing lasso regression and also predicted the labels. The highlighted are the actual values and predicted values.

13.2 Adding the predicted values to the excel

The code in figure 75 saves the predicted values of the dataset and the predicted value is added to the 66th column of the dataset. Figure 76-77 show the saved columns.

13.3 Construct confusion matrices using the actual and predicted labels

I constructed the confusion matrix for the actual and the predicted values and can be seen in the figure 78 . The class zero is considered as a positive class. Figure 79-80 show the confusion matrix for overlapping and non overlapping dataset.

13.4 Comparing performance of Data sets

Calculated the measures from the overlapping and non-overlapping dataset for Image01. I selected accuracy and precision as the two measures. The performance of non

Mean of NonOverlapping Dataset Image01 Feature 61/62

```
In [316]: f61.mean()
Out[316]: 177.1996336996337
In [317]: f62.mean()
Out[317]: 177.30769230769232
In [318]:
In [319]: f61.mean()
Out[319]: 182.99024390243903
In [320]: f62.mean()
Out[320]: 182.65121951219513
In [321]: |
```

Figure 62: Combined mean values of Non Overlapping Train/Test dataset of Image01

Mean of Overlapping Dataset Image01 Feature 61/62

```
In [322]: f61.mean()
Out[322]: 174.15474835490363
In [323]: f62.mean()
Out[323]: 174.14466295310817
In [324]:
In [325]: f61.mean()
Out[325]: 173.95821147526155
In [326]: f62.mean()
Out[326]: 173.92335755419543
In [327]: |
```

Figure 63: Combined mean values of Overlapping Train/Test dataset of Image01

overlapping dataset is better. For overlapping dataset the accuracy is 29% and the precision score is 36%. In the non overlapping dataset, accuracy is 40% and precision score is 42%. Figure 81 - 82 shows the measures for both the datasets.

14 Implement random forest or deep learning

14.1 Training Random Forest for Two Class Classifier

I have used RandomClassifier from sklearn.ensemble library to perform two class classification on merged dataset of image0 and image1. Figure 83 shows the implementation of the RandomForest. I used the two class data set to train the model i.e x_train stores my training data. I drop the label column from the dataset and store in X1. Now the data in X1 is my dataset that will be fed to the RandomForest object to fit the model. I then used the testing data set x_test to test the trained model. Using the predict() I feed only the x_test data without the labels to predict my y_hat values (Predicted Values/labels).I saved the obtained values in a separate files. I followed the same process for both overlapping and non-overlapping data set. I also generated the random forest tree to visualize how the data set

Mean of NonOverlapping Dataset Image012 Feature 61/62

```
In [337]: f61.mean()
Out[337]: 158.10081300813007

In [338]: f62.mean()
Out[338]: 158.5918699186992

In [339]:
```

```
In [319]: f61.mean()
Out[319]: 182.99024390243903

In [320]: f62.mean()
Out[320]: 182.65121951219513

In [321]:
```

Figure 64: Combined mean values of Non Overlapping Train/Test dataset of Image012

Mean of Overlapping Dataset Image012 Feature 61/62

```
In [328]: f61.mean()
Out[328]: 159.62626118539083

In [329]: f62.mean()
Out[329]: 159.5541315308692

In [330]:
```

```
In [331]: f61.mean()
Out[331]: 159.70472052142998

In [332]: f62.mean()
Out[332]: 159.64436982984294

In [333]:
```

Figure 65: Combined mean values of Overlapping Train/Test dataset of Image012

is being split and values are predicted.

14.2 Training Random Forest for Three Class Classifier

For a three class classifier I used the merged data-set of Image0 , Image1 and Image2. The merged data set is then split into 80:20 ratio to train and test the model. I followed the same process as described for the two class classifier and saved the predicted values in a separate file. I also generated the Random Forest Trees and saved it as a pdf file. Figure 85 - 86 show the graphical visualization of the trees.

14.3 Construct confusion matrices using the actual and predicted label

After testing the model, the predicted results in 66th column and original values in 65th column are fed to Confusion matrix. I am taking class 0 as the true positive class and the evaluation is done based on that. Figure 84, 87 and 88 shows two class Confusion matrices respectively. Same is followed for a three class dataset.

14.4 Performance evaluation of the random forest model

I calculated the measures using the formula as well as the inbuilt functions. Figure 90 shows the measures using the

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Oct 25 13:59:51 2021
4
5 @author: p_bud
6 """
7
8
9 import seaborn as sns
10 import matplotlib.pyplot as plt
11 import pandas as pd
12 df = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Priyanka_Budavi_Submission_NL'
13
14 f61 = df['61']
15 f62 = df['62']
16 label = df['64']
17
18 fig = plt.figure(figsize = (20 , 20))
19 ax1 = fig.add_subplot(211)
20 ax1.set_title("Image012 Non overlapping Train dataset")
21 ax1.set_xlabel("Feature 61")
22 ax1.set_ylabel("Feature 62")
23 sns.scatterplot(x = f61 , y= f62 , hue= label , palette= "tab10")
24
25 plt.show()
26
27
28
```

Figure 66: Code for creating Scatter plot of Non Overlapping and Overlapping Dataset

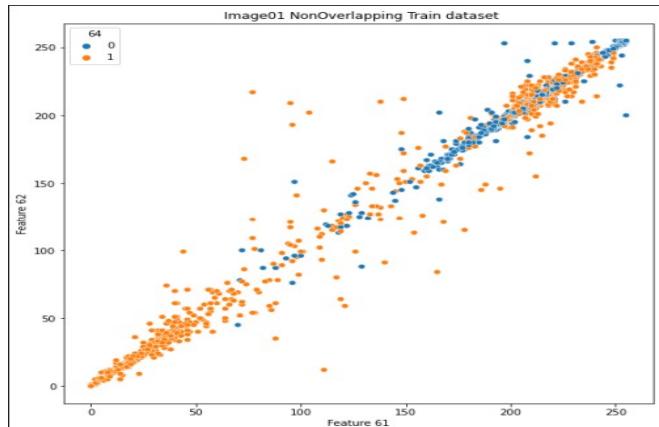


Figure 67: Scatter plot for Feature 61 and 62 of NonOverlapping Image01 Train Dataset

formula. The class 0 is considered as a positive class while calculating using the formula. While using the inbuilt function which class is positive is not known. I choose accuracy and precision to determine the performance. In two class classifier the performance of overlapping data set is good as it is 98%. In three class the overlapping data set performed better with an accuracy of 99%. Also the precision for two class and three class overlapping data set performed better i.e 99%. Refer figure 89

15 Evaluation of the learning models

15.1 Evaluating of models using built-in measures

For this task, I am using inbuilt library accuracy_score, precision_score, recall_score from sklearn.metrics to evaluate the performance of all four types of data sets. Figure 90 show the execution of Lasso regression for in-built performance evaluation. All other data sets are executed in same way to get the results. Refer figure 91- 92 for other models.

15.2 Comparing the performance of models using inbuilt accuracy measures and confusion-based

The chart shows the performance of Lasso Regression and Random forest using Confusion matrix and in-built library

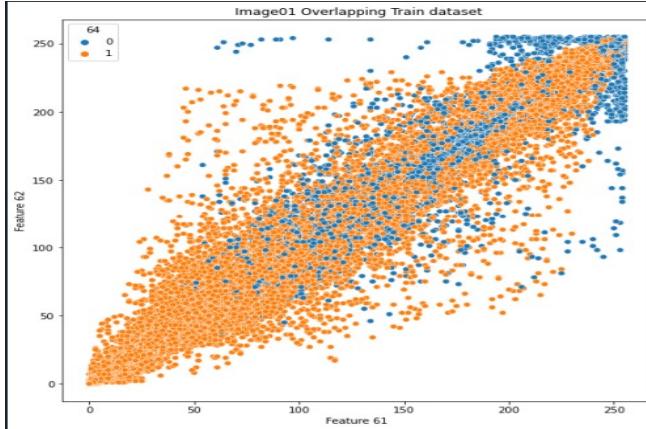


Figure 68: Scatter plot for Feature 61 and 62 of Overlapping Image01 Train Dataset

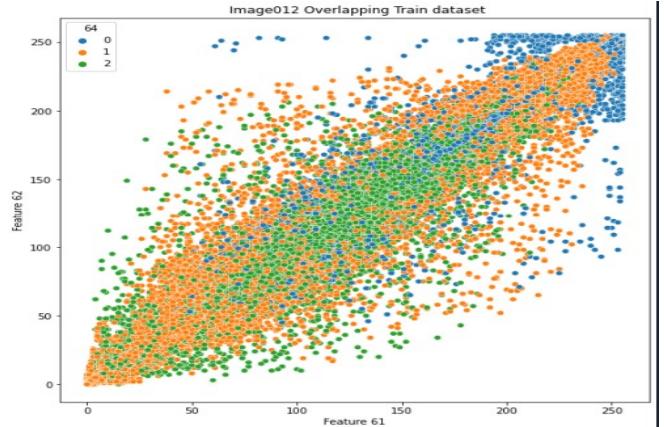


Figure 70: Scatter plot for Feature 61 and 62 of Overlapping Image012 Train Dataset

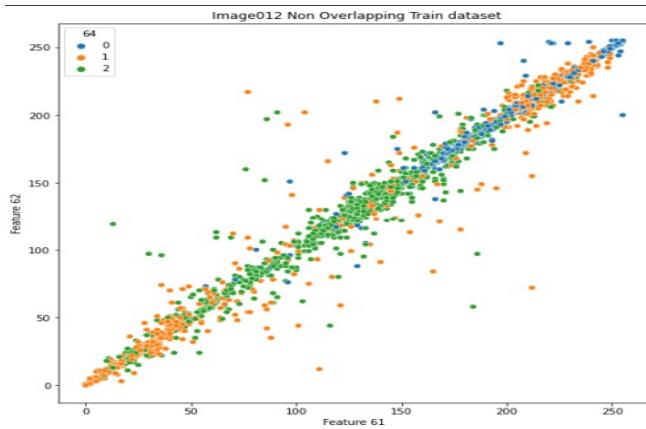


Figure 69: Scatter plot for Feature 61 and 62 of Non Overlapping Image012 Train Dataset

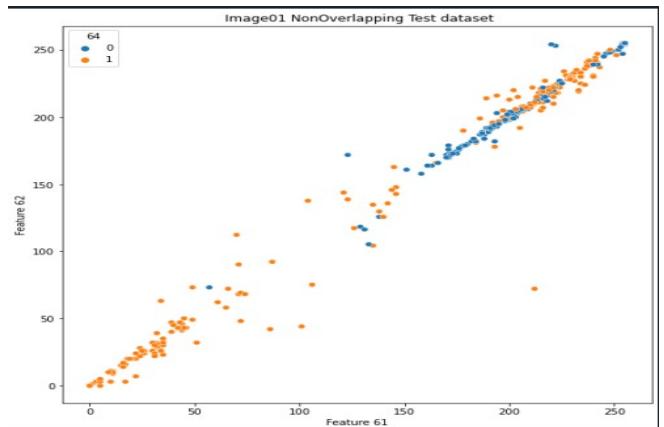


Figure 71: Scatter plot for Feature 61 and 62 of Non Overlapping Image01 Test Dataset

as shown in Figure 93. Looking at the results in figure 93, there is only slight difference in values generated by confusion based matrix and in-built metrics. Confusion based metrics has given better accuracy levels compared to in-built metrics. Also, in in-built metrics the positive class is unknown it always considered the default where as in confusion based we explicitly mention the positive class.

15.3 Comparing the model to find the best one

From Figure 93, we can see that random forest model is performing better than Lasso regression model. There is only slight difference in the calculations of in-built Confusion based metrics and imported libraries. In most cases, both of them gave the same results. Random forest gave best results for overlapping data set compared to non-overlapping data set. In overlapping data set we get many observation which results in better classification when random forest is considered. The accuracy is 98% for overlapping two class classifier and 99% for overlapping three class classifier.

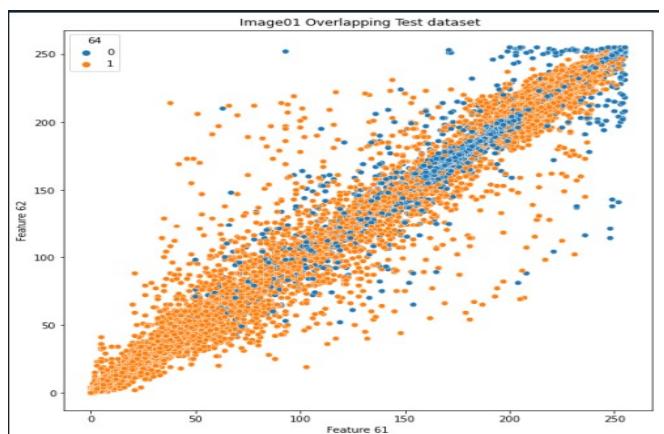


Figure 72: Scatter plot for Feature 61 and 62 of Overlapping Image01 Test Dataset

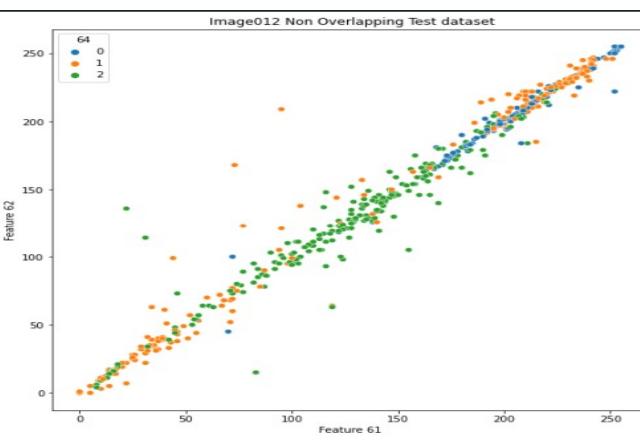


Figure 73: Scatter plot for Feature 61 and 62 of Non Overlapping Image012 Test Dataset

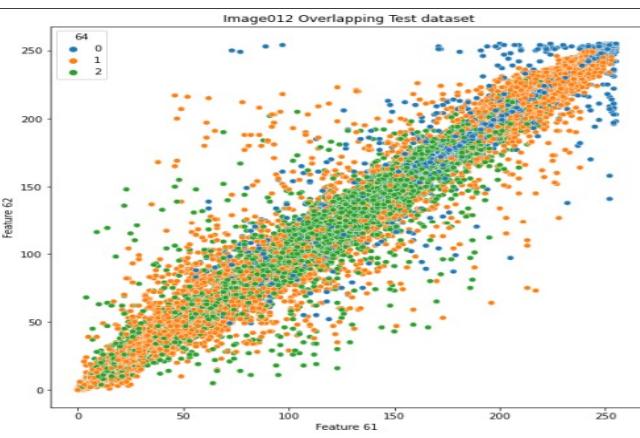


Figure 74: Scatter plot for Feature 61 and 62 of Overlapping Image012 Test Dataset

```
y = pd.read_csv('C:/Users/ylu/Downloads/Desktop/Assignment/Big Data ML/PyTorch/Book/Submission_ML/Assignment2/Data/Training Dataset.csv')
y = pd.read_csv('C:/Users/ylu/Downloads/Desktop/Assignment/Big Data ML/PyTorch/Book/Submission_ML/Assignment2/Data/Testing Dataset.csv')

# load the lib and store in y
y = y['target']
y = np.array(y)

# drop the label column
X = train.drop(['target'], axis=1)
X = np.array(X)

X1 = X[:, 0]
X2 = X[:, 1]
X3 = X[:, 2]
X4 = X[:, 3]
X5 = X[:, 4]
X6 = X[:, 5]
X7 = X[:, 6]
X8 = X[:, 7]
X9 = X[:, 8]
X10 = X[:, 9]
X11 = X[:, 10]
X12 = X[:, 11]
X13 = X[:, 12]
X14 = X[:, 13]
X15 = X[:, 14]
X16 = X[:, 15]
X17 = X[:, 16]
X18 = X[:, 17]
X19 = X[:, 18]
X20 = X[:, 19]
X21 = X[:, 20]
X22 = X[:, 21]
X23 = X[:, 22]
X24 = X[:, 23]
X25 = X[:, 24]
X26 = X[:, 25]
X27 = X[:, 26]
X28 = X[:, 27]
X29 = X[:, 28]
X30 = X[:, 29]
X31 = X[:, 30]
X32 = X[:, 31]
X33 = X[:, 32]
X34 = X[:, 33]
X35 = X[:, 34]
X36 = X[:, 35]
X37 = X[:, 36]
X38 = X[:, 37]
X39 = X[:, 38]
X40 = X[:, 39]
X41 = X[:, 40]
X42 = X[:, 41]
X43 = X[:, 42]
X44 = X[:, 43]
X45 = X[:, 44]
X46 = X[:, 45]
X47 = X[:, 46]
X48 = X[:, 47]
X49 = X[:, 48]
X50 = X[:, 49]
X51 = X[:, 50]
X52 = X[:, 51]
X53 = X[:, 52]
X54 = X[:, 53]
X55 = X[:, 54]
X56 = X[:, 55]
X57 = X[:, 56]
X58 = X[:, 57]
X59 = X[:, 58]
X60 = X[:, 59]
X61 = X[:, 60]
X62 = X[:, 61]
X63 = X[:, 62]
X64 = X[:, 63]
X65 = X[:, 64]
X66 = X[:, 65]
X67 = X[:, 66]
X68 = X[:, 67]
X69 = X[:, 68]
X70 = X[:, 69]
X71 = X[:, 70]
X72 = X[:, 71]
X73 = X[:, 72]
X74 = X[:, 73]
X75 = X[:, 74]
X76 = X[:, 75]
X77 = X[:, 76]
X78 = X[:, 77]
X79 = X[:, 78]
X80 = X[:, 79]
X81 = X[:, 80]
X82 = X[:, 81]
X83 = X[:, 82]
X84 = X[:, 83]
X85 = X[:, 84]
X86 = X[:, 85]
X87 = X[:, 86]
X88 = X[:, 87]
X89 = X[:, 88]
X90 = X[:, 89]
X91 = X[:, 90]
X92 = X[:, 91]
X93 = X[:, 92]
X94 = X[:, 93]
X95 = X[:, 94]
X96 = X[:, 95]
X97 = X[:, 96]
X98 = X[:, 97]
X99 = X[:, 98]
X100 = X[:, 99]

# scale value - smallest parameter
X = np.subtract(X, X.min())
X = np.divide(X, X.max())

# y_hat = y * beta_0 + beta_1 * X1 + ... + beta_99 * X99
y_hat = y * beta_0 + beta_1 * X1 + beta_2 * X2 + beta_3 * X3 + beta_4 * X4 + beta_5 * X5 + beta_6 * X6 + beta_7 * X7 + beta_8 * X8 + beta_9 * X9 + beta_10 * X10 + beta_11 * X11 + beta_12 * X12 + beta_13 * X13 + beta_14 * X14 + beta_15 * X15 + beta_16 * X16 + beta_17 * X17 + beta_18 * X18 + beta_19 * X19 + beta_20 * X20 + beta_21 * X21 + beta_22 * X22 + beta_23 * X23 + beta_24 * X24 + beta_25 * X25 + beta_26 * X26 + beta_27 * X27 + beta_28 * X28 + beta_29 * X29 + beta_30 * X30 + beta_31 * X31 + beta_32 * X32 + beta_33 * X33 + beta_34 * X34 + beta_35 * X35 + beta_36 * X36 + beta_37 * X37 + beta_38 * X38 + beta_39 * X39 + beta_40 * X40 + beta_41 * X41 + beta_42 * X42 + beta_43 * X43 + beta_44 * X44 + beta_45 * X45 + beta_46 * X46 + beta_47 * X47 + beta_48 * X48 + beta_49 * X49 + beta_50 * X50 + beta_51 * X51 + beta_52 * X52 + beta_53 * X53 + beta_54 * X54 + beta_55 * X55 + beta_56 * X56 + beta_57 * X57 + beta_58 * X58 + beta_59 * X59 + beta_60 * X60 + beta_61 * X61 + beta_62 * X62 + beta_63 * X63 + beta_64 * X64 + beta_65 * X65 + beta_66 * X66 + beta_67 * X67 + beta_68 * X68 + beta_69 * X69 + beta_70 * X70 + beta_71 * X71 + beta_72 * X72 + beta_73 * X73 + beta_74 * X74 + beta_75 * X75 + beta_76 * X76 + beta_77 * X77 + beta_78 * X78 + beta_79 * X79 + beta_80 * X80 + beta_81 * X81 + beta_82 * X82 + beta_83 * X83 + beta_84 * X84 + beta_85 * X85 + beta_86 * X86 + beta_87 * X87 + beta_88 * X88 + beta_89 * X89 + beta_90 * X90 + beta_91 * X91 + beta_92 * X92 + beta_93 * X93 + beta_94 * X94 + beta_95 * X95 + beta_96 * X96 + beta_97 * X97 + beta_98 * X98 + beta_99 * X99

# Train data accuracy
y_hatTrain_saved = pd.DataFrame(y_hatTrain)
y_hatTest_saved = pd.DataFrame(y_hatTest)

# confusion matrix, y_hatTrain
c = confusion_matrix(y_hatTrain, y_hatTrain)
```

Figure 75: Implementation of Lasso Regression

44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65
234	232	236	235	237	237	236	239	237	230	234	234	234	235	236	238	235	231	233	232	1	0
201	202	202	199	202	201	199	200	201	199	198	198	198	200	200	199	199	198	198	198	198	0
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0
204	206	202	203	205	201	201	204	204	206	204	206	204	203	202	202	204	207	206	205	205	1
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0
174	174	173	172	180	171	175	175	175	173	172	172	183	180	179	176	176	175	173	173	173	0
4	4	4	4	36	34	31	19	3	4	4	3	38	35	33	24	4	3	3	1	1	0
255	255	255	254	255	255	255	255	255	255	255	254	255	255	255	255	254	255	254	255	250	0
187	188	188	188	189	189	187	187	186	187	186	186	186	188	188	188	188	187	187	185	185	0
176	177	177	176	174	175	175	176	176	175	175	175	176	170	171	171	170	170	171	171	172	0
212	215	213	212	216	216	217	216	216	215	215	215	218	217	211	211	211	219	217	217	219	1
205	206	209	207	203	203	204	208	208	209	206	208	203	201	203	205	209	206	205	205	205	0
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0
20	101	26	25	22	25	27	23	22	27	29	26	20	24	23	24	22	30	26	30	1	0
10	6	1	1	27	21	18	16	10	6	1	0	26	22	17	14	9	5	0	1	1	0
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0
171	175	173	171	212	211	216	187	184	173	174	171	178	210	216	195	185	184	181	174	174	0
238	245	242	249	251	150	144	212	218	214	224	229	242	251	253	247	211	229	241	241	241	1
160	147	133	124	188	183	173	163	150	134	126	115	182	174	166	151	138	128	118	118	107	0
219	217	217	218	214	221	218	219	217	218	217	211	221	212	218	218	217	219	218	218	219	1
195	196	196	194	194	194	194	194	194	193	194	194	197	195	195	192	193	193	195	194	194	0
216	222	252	254	211	211	213	213	216	222	252	254	212	214	214	215	217	222	253	254	0	1
255	255	255	254	255	255	255	255	255	255	255	254	255	255	255	255	255	255	254	254	254	0
182	180	179	179	209	209	165	194	184	186	182	180	179	233	234	210	210	196	193	182	180	0
226	227	227	227	222	223	223	224	224	223	223	223	222	222	222	222	221	221	221	221	221	0
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0
237	235	233	233	244	244	242	246	248	238	231	231	238	240	237	244	252	251	246	242	0	1

Figure 76: The Actual values and Predicted values for Non-Overlapping Dataset

44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68
173	174	173	174	178	170	178	177	176	175	176	175	175	183	181	179	177	177	176	176	175	175	175	175	175
213	213	213	214	210	211	212	214	214	214	214	215	215	209	210	211	212	213	213	213	215	211	0	0	0
201	201	205	203	200	200	202	203	201	204	204	203	202	203	203	203	203	203	204	204	205	204	205	204	0
254	254	253	253	249	243	251	252	252	252	253	244	251	251	251	251	252	252	253	254	251	252	252	253	254
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
22	22	26	16	6	10	13	17	21	23	25	25	5	5	9	12	15	24	21	23	26	1	0	0	
209	204	205	204	205	208	209	210	208	208	206	204	205	208	205	208	208	210	210	208	207	206	206	206	206
203	214	208	202	215	211	210	213	207	214	209	213	214	210	214	216	210	216	213	209	216	216	216	216	216
34	32	33	35	28	26	30	29	38	33	31	34	29	29	27	27	27	30	31	31	32	31	32	31	32
8	12	14	14	0	1	2	5	8	11	13	0	1	3	5	9	10	12	12	11	11	1	1	1	
223	218	217	216	216	217	219	220	224	224	212	209	217	220	232	220	225	228	220	220	220	1	0	0	
191	191	190	192	191	191	191	191	191	192	189	189	191	191	191	191	191	191	191	191	189	191	191	191	
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
178	170	151	79	178	184	182	180	175	165	163	85	185	181	179	178	170	161	151	83	99	0	1	1	
215	214	213	209	207	214	215	212	215	215	210	205	211	217	217	209	209	209	209	209	209	209	209	209	209
90	122	157	214	50	48	58	58	105	118	145	193	52	46	46	57	91	90	62	133	183	1	0	0	
191	191	192	191	190	190	190	191	191	191	191	191	191	191	191	191	191	191	191	191	191	191	191	191	
204	204	204	203	205	206	204	204	204	204	203	202	203	204	204	204	204	204	204	204	205	204	205	204	
122	106	33	63	165	158	147	140	142	119	110	81	174	165	166	174	155	139	111	98	1	1	1	1	
188	189	188	188	188	188	188	188	188	188	188	187	187	187	187	187	187	187	187	187	187	187	187	187	
265	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
26	26	33	27	23	21	22	22	30	26	25	31	22	22	21	23	26	26	26	26	26	26	26	26	26
32	30	33	30	34	35	34	36	36	34	34	32	38	36	38	36	33	34	33	35	35	35	35	35	35
151	155	154	156	156	158	149	142	148	151	149	149	160	167	155	149	153	151	149	149	147	147	147	147	147
228	225	224	224	214	224	225	225	228	228	227	225	224	224	225	227	228	227	228	225	225	225	225	225	225
157	155	167	166	144	145	148	152	168	196	201	179	145	167	167	195	197	198	186	186	1	1	1	1	

Figure 77: The Actual values and Predicted values for Overlapping Dataset

```
62
63 #Confusion matrix
64 CC = confusion_matrix(Y, yhatTrain)
65 TN = CC[1,1]
66 FP = CC[1,0]
67 FN = CC[0,1]
68 TP = CC[0,0]
69 FPFN = FP+FN
70 TPTN = TP+TN
71
72
73 Accuracy = 1/(1+(FPFN/TPTN))
74 print("Train_Accuracy_Score:",Accuracy)
75 Precision = 1/(1+(FP/TP))
76 print("Train_Precision_Score:",Precision)
77 Sensitivity = 1/(1+(FN/TP))
78 print("Train_Sensitivity_Score:",Sensitivity)
79 Specificity = 1/(1+(FP/TN))
80 print("Train_Specificity_Score:",Specificity)
81 print('-----')
```

Figure 78: Confusion Matrix code

```

In [120]:
    ...: print(" Confusion Matrix Train Dataset")
Confusion Matrix Train Dataset

In [121]: CC
Out[121]:
array([[186, 633],
       [421, 398]], dtype=int64)

In [122]:
    ...: print(" Confusion Matrix Test Dataset")
Confusion Matrix Test Dataset

In [123]: CC_test
Out[123]:
array([[ 44, 161],
       [ 97, 108]], dtype=int64)

```

Figure 79: Confusion Matrix for Non overlapping Dataset

```

Train_Accuracy_Score: 0.38705738705738707
Train_Precision_Score: 0.35569422776911075
Train_Sensitivity_Score: 0.2783882783882784
Train_Specificity_Score: 0.49572649572649574
-----
Test_Accuracy_Score: 0.4097560975609756
Test_Precision_Score: 0.4280155642023346
Test_Sensitivity_Score: 0.5365853658536585
Test_Specificity_Score: 0.2829268292682927
-----
```

Figure 82: Quantitative Measure of Non-Overlapping dataset

```

    ...: print(" Confusion Matrix Train Dataset")
Confusion Matrix Train Dataset

In [127]: CC
Out[127]:
array([[ 7215, 56702],
       [36338, 32412]], dtype=int64)

In [128]:
    ...: print(" Confusion Matrix Test Dataset")
Confusion Matrix Test Dataset

In [129]: CC_test
Out[129]:
array([[ 1784, 14228],
       [ 9160,  7995]], dtype=int64)

In [130]:
```

Figure 80: Confusion Matrix for Overlapping Dataset

```

10 x_train = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Priyanka_Budavi_Submission_ML/Assignment2/Data/CC_train.csv')
11 x_test = pd.read_csv('C:/Users/p_bud/OneDrive/Desktop/Assignment/Big Data ML/Priyanka_Budavi_Submission_ML/Assignment2/Data/CC_test.csv')
12
13 y = x_train[64]
14 Y = np.array(y)
15 x_train.drop(64, axis=1, inplace=True)
16 X = x_train
17 X1 = np.array(X)
18
19 # Train the model
20 randomClassifier = RandomForestClassifier(random_state=0, n_estimators=50, n_jobs=1)
21 randomClassifier = randomClassifier.fit(X1, Y)
22
23 #Testing the model using Trained results
24 test_y = x_test[64]
25 Y_test = np.array(test_y)
26 x_test.drop(64, axis=1, inplace=True)
27 X_test = np.array(x_test)
28 y_pred = randomClassifier.predict(X_test)
29 yhat_saved = pd.DataFrame(y_pred)
30 yhat_saved.to_csv('Predicted_01_NO.csv', index=False)
31
32
33 CC_test = confusion_matrix(test_y, y_pred)
34 # 0 as my positive class
35 TN = CC_test[1,1]
36 FP = CC_test[1,0]
37 FN = CC_test[0,1]
38 TP = CC_test[0,0]
39
40 FPFN = FP+FN
41 TPTN = TP+TN
42
43
44 #Qualitative measures of performance calculation using Confusion matrix
45 Accuracy = 1/(FPFN/TPTN)
46 print("Test_Accuracy_Score:",Accuracy)
47 Precision = 1/(FP/TP)
48 print("Test_Precision_Score:",Precision)
49 Sensitivity = 1/(FN/TP)
50 print("Test_Sensitivity_Score:",Sensitivity)
51 Specificity = 1/(FP/TN)
52 print("Test_Specificity_Score:",Specificity)
53
54 print("-----")
55 print("Measures using Library")
56
57 print("-----")
58 #Qualitative measures of performance calculation using sklearn metrics
59 print('sklearn.metrics Accuracy',accuracy_score(test_y, y_pred))
60 print('sklearn.metrics precision',precision_score(test_y, y_pred))
61 print('sklearn.metrics sensitivity',recall_score(test_y, y_pred))
```

Figure 83: Code for Random Forest

```

Train_Accuracy_Score: 0.29904949987562846
Train_Precision_Score: 0.1659466911764706
Train_Sensitivity_Score: 0.11299028427491903
Train_Specificity_Score: 0.47202909090909095
-----
Test_Accuracy_Score: 0.2949618596798022
Test_Precision_Score: 0.3598650168728909
Test_Sensitivity_Score: 0.4662197610026232
Test_Specificity_Score: 0.11147889083187609
-----
```

Figure 81: Quantitative Measure of Overlapping dataset

```

CC_test = confusion_matrix(test_y, y_pred)
# 0 as my positive class
TN = CC_test[1,1]
FP = CC_test[1,0]
FN = CC_test[0,1]
TP = CC_test[0,0]

FPFN = FP+FN
TPTN = TP+TN

#Qualitative measures of performance calculation using Confusion matrix
Accuracy = 1/(FPFN/TPTN)
print("Test_Accuracy_Score:",Accuracy)
Precision = 1/(FP/TP)
print("Test_Precision_Score:",Precision)
Sensitivity = 1/(FN/TP)
print("Test_Sensitivity_Score:",Sensitivity)
Specificity = 1/(FP/TN)
print("Test_Specificity_Score:",Specificity)

print("-----")
print("Measures using Library")

print("-----")
#Qualitative measures of performance calculation using sklearn metrics
print('sklearn.metrics Accuracy',accuracy_score(test_y, y_pred))
print('sklearn.metrics precision',precision_score(test_y, y_pred))
print('sklearn.metrics sensitivity',recall_score(test_y, y_pred))
```

Figure 84: Confusion Matrix for Random Forest

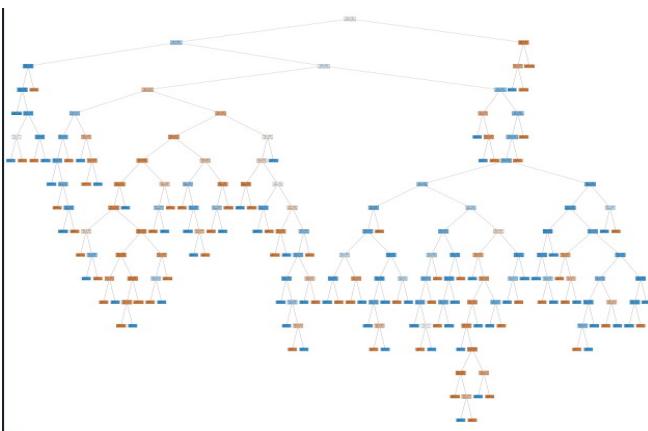


Figure 85: Random Forest Tree for Image01 Non Overlapping Dataset

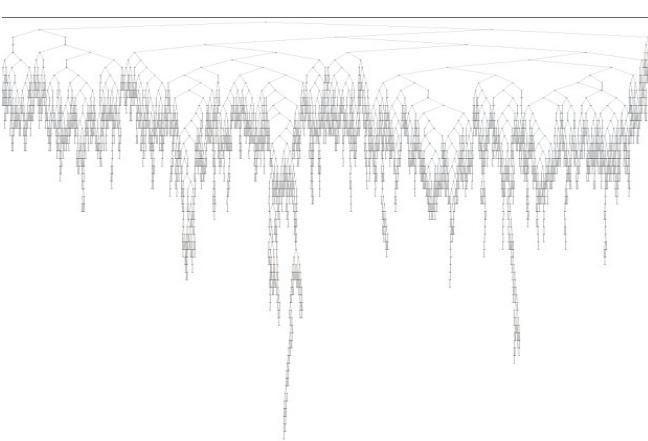


Figure 86: Random Forest Tree for Image01 Overlapping Dataset

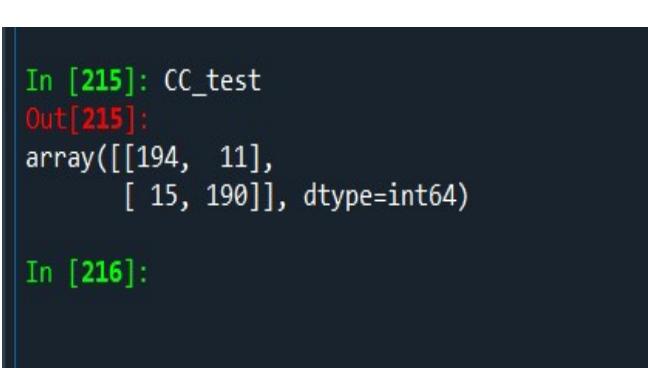


Figure 87: Confusion Matrix for two class NonOverlapping Dataset

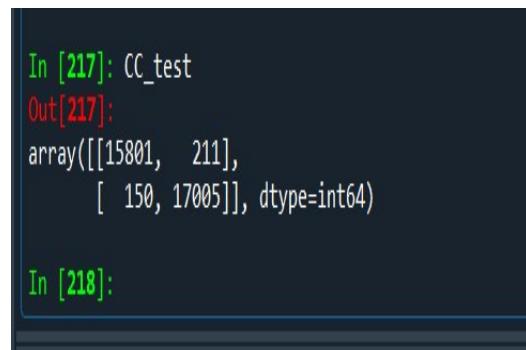


Figure 88: Confusion Matrix for two class Overlapping Dataset

TWO CLASS OVERLAPPING				
N_ESTIMATORS	Accuracy	Precision	Sensitivity	Specificity
50	0.98	0.99	0.98	0.99
100	0.98	0.99	0.98	0.99
TWO CLASS NON OVERLAPPING				
N_ESTIMATORS	Accuracy	Precision	Sensitivity	Specificity
50	0.936	0.928	0.946	0.926
100	0.936	0.928	0.946	0.926
THREE CLASS OVERLAPPING				
N_ESTIMATORS	Accuracy	Precision	Sensitivity	Specificity
50	0.991	0.992	0.991	0.992
100	0.93	0.938	0.938	0.938
THREE CLASS NON OVERLAPPING				
N_ESTIMATORS	Accuracy	Precision	Sensitivity	Specificity
50	0.924	0.918	0.933	0.9166
100	0.93	0.938	0.938	0.938

Figure 89: Performance Measures of Two Class and Three class dataset

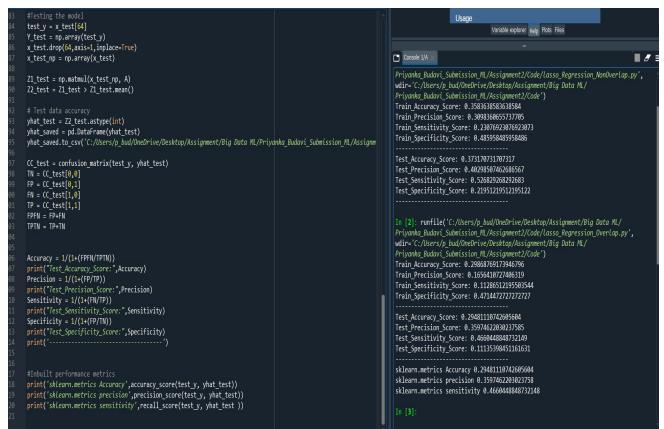


Figure 90: Lasso Regression Overlapping and Non Overlapping measures

```
#Qualitative measures of performance calculation using Confusion matrix
Accuracy = 1/(1+(TP/(TP+FP)))
print("Test_Accuracy Score :",Accuracy)
Precision = TP/(TP+FP)
print("Test_Precision_Score :",Precision)
Sensitivity = 1/(1+(TP/TP))
print("Test_Sensitivity_Score :",Sensitivity)
Specificity = 1/(1+(TP/FP))
print("Test_Specificity_Score :",Specificity)

print("-----")
print("Measures using library")

print("-----")
#Qualitative measures of performance calculation using sklearn metrics
print("sklearn.metrics accuracy :",accuracy_score(test_y, y_pred))
print("sklearn.metrics precision :",precision_score(test_y, y_pred))
print("sklearn.metrics sensitivity :",recall_score(test_y, y_pred))

#Generate the random forest tree
plt.figure(figsize=(10,10))
tree_clf = tree.DecisionTreeClassifier(max_depth=3, filled=True)
gini_importance_df = pd.DataFrame(importances, columns=[["Feature"]])
gini_importance_df["Importance"] = importances
```

Figure 91: Random Forest two class measures

```
Usage
Variable explorer Help File

Grade EA

In [8]: runfile('C:/Users/Jayesh/Desktop/KaggleProjects/Big Data/Ml/Projects/Adult_Salaries/Assignment_Ml/Assignment/Code/Random_Forest_ThreeClassifier.py', wdir='C:/Users/Jayesh/Desktop/KaggleProjects/Big Data/Ml/Projects/Adult_Salaries/Assignment_Ml/Assignment/Code')
Test_Accuracy_Score : 0.89842468035196
Test_Precision_Score : 0.9384540835196
Test_Sensitivity_Score : 0.89547480435196
Test_Specificity_Score : 0.89547480535196

Measures using library

sklearn.metrics.accuracy_score(test_y, y_pred)
sklearn.metrics.precision_score(test_y, y_pred)
sklearn.metrics.sensitivity_score(test_y, y_pred)

Measures using sklearn

Accuracy measure
Import math
((Y12)*math.log(Y12, 2) + (S12)*math.log(S12, 2))

Entropy measure
Import math
tree.plot_tree(RandomForestClassifier.estimators_[2], filled=True)
tree.export_graphviz(RandomForestClassifier.estimators_[2], feature_names=)

#Extract the random forest tree
plt.figure(figsize=(80,100))
tree.plot_tree(RandomForestClassifier.estimators_[2], filled=True)
tree.export_graphviz(RandomForestClassifier.estimators_[2], feature_names=)
```

Figure 92: Random Forest three class measures

		Confusion Based						
	Accuracy	Accuracy	Precision	Precision	Specificity	Specificity	Sensitivity	Sensitivity
RF Two class Nonoverlap[01]	0.936	0.936	0.945	0.928	0.926	0.946	0.9268	0.926
RF Two class Overlap[01]	0.989	0.989	0.987	0.99	0.991	0.986	0.991	0.99
RF Two class Nonoverlap[012]	0.804	0.924	0.805	0.918	0.803	0.933	0.807	0.91
RF Two class Overlap[012]	0.95	0.991	0.958	0.992	0.956	0.992	0.57	0.99
Lasso Two Class Nonoverlap[01]	0.3731	0.373	0.59	0.4	0.373	0.826	0.373	0.48
Lasso Two Class overlap[01]	0.294	0.294	0.359	0.359	0.466	0.466	0.466	0.46
Lasso Two Class Nonoverlap[012]	0.195	0.29	0.119	0.375	0.119	0.36	0.192	0.50
Lasso Two Class overlap[012]	0.2013	0.276	0.119	0.355	0.179	0.491	0.19	0.4

Figure 93: Built-in measures and confusion based measure

16 ASSIGNMENT 3: A CLASSIFIER ON A BIG DATA SYSTEM

17 Completed tasks from Assignment 1 and Assignment 2

All the below tasks related to assignment 1 and assignment 2 are completed.

1. In assignment 1, all four feature vectors/ data frames of two class non-overlapping, two class overlapping, three class non-overlapping and three class overlapping are generated and stored in respective csv files.

2. In assignment 2, implementation of Lasso regression as two class classifier and random forest as two and three class classifier is performed in local environment and the best model was selected.

18 Understanding and using the Data-bricks

18.1 Signing up Data-bricks community version

I created an account to sign in for community version of data-bricks. I used my email ID too create the same.

18.2 Exploring Quick-start tutorial and setting up Big data environment in Data-bricks community

I created my community Data-bricks account which gives around 15GB of free memory. Then, created cluster CSC610 using "Create Cluster" option. I chose 7.3 ML version Apache spark 3.0.1, Scala 2.12 as there was no 7.2 version present in the list. I referred the video uploaded on data-bricks and created the tables as shown in the video. I created 4 data-set that is overlapping and non-overlapping data-set for two class and three class.

18.3 Cluster creation

I created my cluster and named it as CSC610 and it took around 5 min to create one. Figure 94 shows the creation of a cluster.

18.4 Table creation

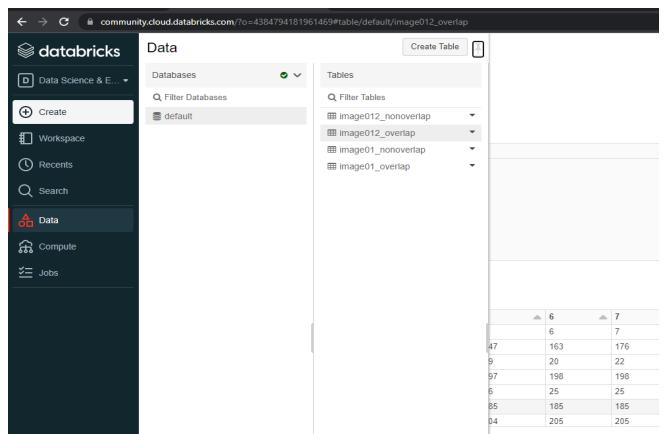
Four data frames are created by uploading two class overlapping and non-overlapping, three class overlap and non-overlap feature vectors into big data system. These are the same feature vectors used in previous assignments. The data for image01_nonoverlap , image012_nonoverlap , image01_overlap , image012_overlap was created as shown in the figure 95. The data was set huge for overlapping and also I was getting error, so I reduced the data and created them using the UI way.

19 Converting the code to run your data sets

For this task, I am using the random forest model which I implemented in assignment 2 . Previously I calculated the

Name	State	Nodes	Runtime
CSC610	Running	1 (0 spot)	7.3 LTS ML (includes Apache Spark 3.0.1, Scala 2.12)

Figure 94: Cluster Creation



Tables		
Q Filter Tables		
image012_nonoverlap		
image012_overlap		
image01_nonoverlap		
image01_overlap		

Figure 95: Data-frame creation

accuracy by changing the n_estimators and then chose the best model amongst all of them. So I will use the same data in data-bricks environment and compute the accuracy. Below are the general steps followed to calculate the same. I showed the execution for one data-set Same process is followed for the other three data-sets.

19.1 Pyspark

Imported the pyspark as shown in the figure 96 for computational purpose and fetching the data from pyspark.sql.dataframe.DataFrame which was uploaded earlier in Task2 using the select query as shown in the video.

19.2 Splitting the data in train and test

After importing the input data we should check if there are any null values present and this can be done by using isna() function. Then, split the data into 80% Training data-set and 20% Testing data-set. Splitting is done using train_test_split() function imported from sklearn.model_selection module. I split the data into 80:20 by setting the value of test_size = 0.2 Figure 97 shows the code for splitting the data.

19.3 Histogram and Box-Plot for the data-set

Figure 98 and 99 shows the histogram and box-plots respectively for Feature 30 using sea-born package

```

Cmd 1
1 import pyspark

Command took 0.05 seconds -- by budavipriyanka@gmail.com at 11/17/2021, 3:05:27 PM on Assignment3

Cmd 2
1 df = sqlContext.sql("Select * FROM image01_nonooverlap ")

df: pyspark.sql.dataframe.DataFrame = [0: integer, 1: integer ... 63 more fields]

Command took 0.06 seconds -- by budavipriyanka@gmail.com at 11/17/2021, 3:05:28 PM on Assignment3

Cmd 3
1 input_data = df.select("*").toPandas()

(1) Spark Jobs

Command took 0.59 seconds -- by budavipriyanka@gmail.com at 11/17/2021, 3:05:29 PM on Assignment3

Shift+Enter to run

```

Figure 96: Import Pyspark

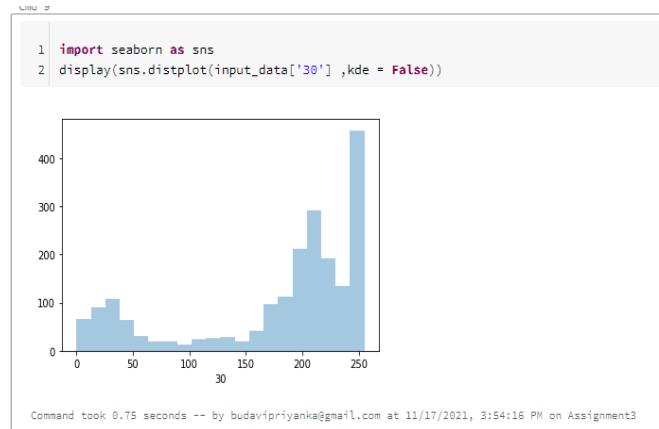


Figure 98: Histogram for feature 30

```

Cmd 6
1 from sklearn.model_selection import train_test_split
2
3 train, test = train_test_split(input_data, test_size=0.2, random_state=123)
4 X_train = train.drop(["64"], axis=1)
5 X_test = test.drop(["64"], axis=1)
6 y_train = train['64']
7 y_test = test['64']

Command took 0.02 seconds -- by budavipriyanka@gmail.com at 11/17/2021, 3:55:05 PM on Assignment3

```

Figure 97: Code for splitting the data

19.4 Random-Forest Classifier : Training Model

RandomForestClassifier is trained using 80% Train data set for 50 estimators(as this was the best predicted model in this scenario). The time take to train the model took 0.38 sec on data-bricks. Using x_test we predict the labels for testing data-set. This command took 0.12 seconds to predict the values. Figure 100 show the code for Random-Forest Classifier.

19.5 Generating ROC Curve

ROC curve is drawn using roc_auc_score by passing test and predicted outputs. Also, predict_probabilities with multiple threshold values among tpr along yaxis and fpr along xaxis. Figure 101 and 102 shows the generated ROC curve.

19.6 Feature Importance

Feature importance refers to a class of techniques for assigning scores to input features to a predictive model that indicates the relative importance of each feature when making a prediction. Importance of every feature is calculated to find the feature that is maximum impacting in classification. Figure 103 shows feature 63 seems to have highest importance compared to other features. All the features are sorted to know the importance of the highest feature.

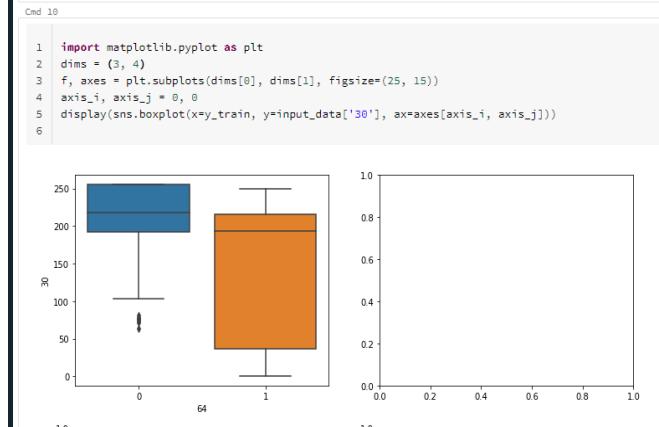


Figure 99: Box-plot for feature 30

19.7 Calculating Accuracy scores, AUC scores and Confusion matrix

Accuracy of the model is calculated using sklearn.metrics accuracy_score, AUC score is calculated and confusion matrix is printed. Figure 104 shows the execution of accuracy and confusion matrix.

20 COMPARING THE RESULTS ON A LOCAL SYSTEM VERSUS A BIG DATA SYSTEM

20.1 Local System Results

Two class Nonoverlap data set executed the model in just 0.16 seconds. Three class Nonoverlap data set just took 6.62 seconds to execute or train the model. Two class overlap dataset took 25.32 seconds and three class overlap data set took 7.42 minutes. The overlapping dataset took longer time to train the model.

20.2 Big Data System Results

Two class Nonoverlap data set executed the model in just 0.44 seconds. Three class Nonoverlap data set took 11.7 seconds to execute or train the model. Two class overlap dataset took 37.71 seconds and three class overlap data set took 8.89 minutes. The overlapping dataset took longer

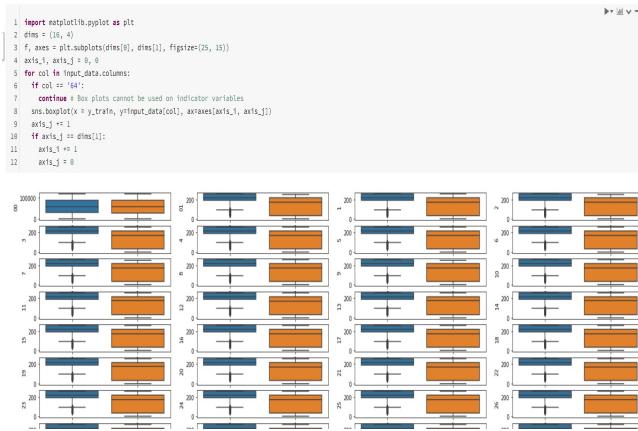


Figure 100: Box-plot for Multiple features

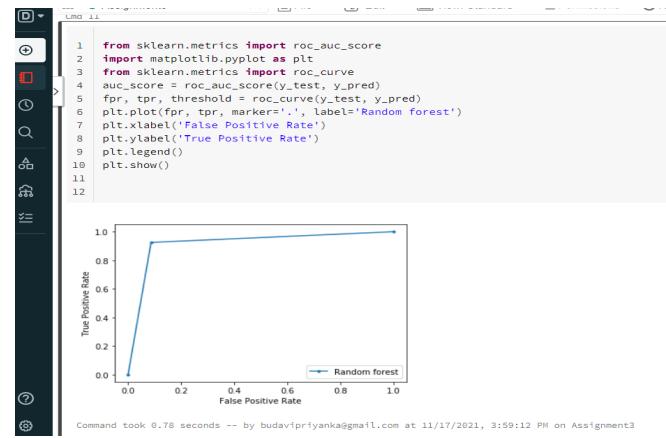


Figure 102: ROC curve

```

1 from sklearn.ensemble import RandomForestClassifier
2 randomFClassifier = RandomForestClassifier(random_state=0,n_estimators=50,oob_score=True, n_jobs=-1)
3 randomFClassifier = randomFClassifier.fit(X_train, y_train)

Command took 0.38 seconds -- by budavipriyanka@gmail.com at 11/17/2021, 3:57:42 PM on Assignment3

Cmd 8

1 y_pred = randomFClassifier.predict(X_test)
2 y_pred

```

Figure 101: Using the RandomForest Model

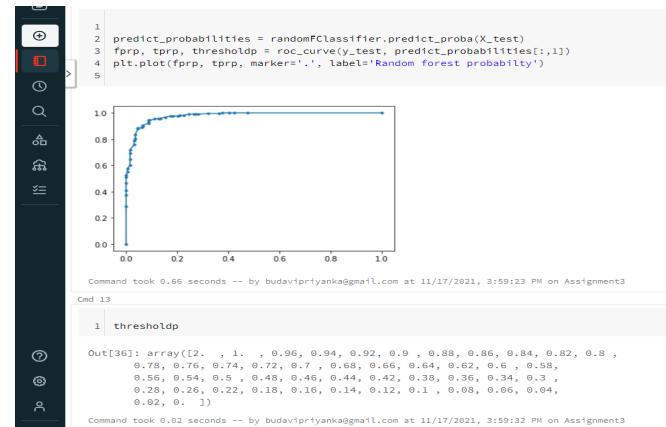


Figure 103: ROC curve with multiple threshold values

time to train the model. Figure 105 shows the observations for local system and big data system. Executed all the datasets and noted the time taken to train the model. Figure 106 shows the code implementation of Random Forest and displaying the time taken to train the model.

20.3 Comparing the Local system and Big data system

From the table i.e Figure 105 we observe that the time taken to train the model for two class overlap and non overlapping data-set, the local system performed better. This means that for small data-set big data system took longer time to train the model. The data-set for overlapping three class classifier, the number of observations are many and hence the accuracy will be great and also it will take more time to compute compared to other data-sets. To check this I chose three class overlapping data-set and increased the n_estimators to 1500. Till now estimator with 1000 the local system performed better. As the value of n_estimators increased to 1500 the local system took more time to compute i.e it took 12.87 minutes. While the big data system took 12.71 minutes. Thus by this we can conclude that as the number of data increases the big data performs better than the local system. Figure 107 and 108 shows the computation time with 1500 n_estimators. Figure 110 shows the table of computed time to run on local and big-data. With 1GB data we observe that the system failed and

showed error when trying to deal with large data-set. During the creation of data-set I was getting error uploading the UI way but I was able to upload the data via notebook and created the table for the same. I used multiple three class overlapping data-set to create 1GB data. I used this data-set because this would be the best fit to test the time taken to train model on Big data and Local System as it was taking more time to train as showed in the Figure 106. After running the model I conclude that, as the number of observations increases the system will perform low and further fail due to large data-set and in such situation big data performs better.

References

- [1] Š. Marko, “Automatic fruit recognition using computer vision,” Mentor: Matej Kristan), Fakulteta za racunalništvo in informatiko, Univerza v Ljubljani, 2013.

```

1 import pandas as pd
2 feature_importances = pd.DataFrame(randomForestClassifier.feature_importances_, index=x_train.columns.tolist(), columns=['importance'])
3 feature_importances.sort_values('importance', ascending=False)
4
5 Out[57]:
6
7 importance
83 0.036992
82 0.035737
80 0.034631
15 0.034505
14 0.030717
...
36 0.04639
4 0.004780
29 0.04606
20 0.003778
37 0.005613

```

64 rows × 1 columns

Figure 104: ROC curve with multiple threshold values

```

1 from sklearn.ensemble import RandomForestClassifier
2
3 import time
4
5 t1 = time.time()
6
7 x_train = pd.read_csv('C:/Users/91/Downloads/Assignment0/bigData/H_Pyjama_Budget_Subscription_ML/Assignment3/Assignment3.csv')
8 x_test = pd.read_csv('C:/Users/91/Downloads/Assignment0/bigData/H_Pyjama_Budget_Subscription_ML/Assignment3/Assignment3.csv')
9
10 y = x_train[6]
11 Y = y.array()
12 x_train.drop(6, axis=1, inplace=True)
13 X = x_train[1:6]
14 X1 = np.array(X)
15
16 #Fitting the model
17 t2 = time.time()
18 randomClassifier = RandomForestClassifier(random_state=0, n_estimators=1000, oob_score=True, n_jobs=-1)
19 randomClassifier.fit(X1, Y)
20
21 t3 = time.time()
22 print("Time elapsed: ", t3-t1)
23
24 importance = randomClassifier.feature_importances_
25 indices = importance.argsort()[-1:0:-1]
26
27 oob_error = 1 - randomClassifier.oob_score_
28
29 #Predicting the results using Trained result
30 t4 = time.time()
31 Y_pred = x_test[6]
32 Y_test = y.array(y_test)
33 x_test.drop(6, axis=1, inplace=True)
34 X2 = x_test[1:6]
35 X3 = np.array(X2)
36
37 y_pred = randomClassifier.predict(X3)
38 y_pred = pd.DataFrame(y_pred)
39 y_pred.to_csv('C:/Users/91/Downloads/Assignment0/bigData/H_Pyjama_Budget_Subscription_ML/Assignment3/Assignment3/overlap01/overlap01/predicted1.csv', index=False)
40
41 CC_test = confusion_matrix(y_test, y_pred)
42
43 CM = pd.read_csv('C:/Users/91/Downloads/Assignment0/bigData/H_Pyjama_Budget_Subscription_ML/Assignment3/Assignment3/overlap01/overlap01/cm1.csv')
44
45 FP = CM[CM['TN'] == 0]
46 FN = CM[CM['TP'] == 0]
47 TP = CM[CM['FN'] == 0]
48
49 FP_N = len(FP)
50 FN_N = len(FN)

```

Time elapsed: 445.3893158814575

Figure 107: Random Forest code to display the time take to train the model

```

1 Time elapsed: 772.2303709983826
2 Test_Accuracy_Score: 0.9897657213316893
3 Test_Precision_Score: 0.9904663746457099
4 Test_Sensitivity_Score: 0.9881748071979434
5 Test_Specificity_Score: 0.9912322274881517
6 sklearn.metrics Accuracy 0.9476825842696629
7 Traceback (most recent call last):

```

Figure 108: Local Computation Time

```

1 from sklearn.metrics import accuracy_score
2 accuracy = accuracy_score(y_test, y_pred)
3 print(accuracy)

0.9195121951219513
Command took 0.02 seconds -- by budavipriyanka@gmail.com at 11/17/2021, 3:59:42 PM on Assignment3

Cmd 16

1 from sklearn.metrics import confusion_matrix
2 CC_test = confusion_matrix(y_test, y_pred)
3 print(CC_test)

[[179 17]
 [ 16 198]]
Command took 0.03 seconds -- by budavipriyanka@gmail.com at 11/17/2021, 3:59:48 PM on Assignment3

Cmd 17

1 from sklearn.metrics import roc_auc_score
2 auc_score = roc_auc_score(y_test, y_pred)
3 print(auc_score)
4
5

0.919249475491131
Command took 0.03 seconds -- by budavipriyanka@gmail.com at 11/17/2021, 3:59:51 PM on Assignment3

```

Figure 105: Accuracy , Confusion Matrix , AUC score

```

1 from sklearn.ensemble import RandomForestClassifier
2
3 randomClassifier = RandomForestClassifier(random_state=0, n_estimators=1500, oob_score=True, n_jobs=1)
4
5 randomClassifier = randomClassifier.fit(x_train, y_train)

```

Command took 12.71 minutes -- by budavipriyanka@gmail.com at 11/19/2021, 1:54:08 PM on Assignment3

Figure 109: Big data computation Time

Local and Big Data execution Random Forest				
	NonOverlap01	NonOverlap02	Overlap01	Overlap02
Estimators	50	1000	100	1000
Big Data Environment	0.44	11.7	37.71	8.89 minutes
Local	0.16	6.62	25.32	7.42 minutes
Accuracy	0.91	0.8276	0.98	0.9517
Values are in seconds				

Figure 106: Comparing local and Big-data environment

Overlap three class (huge Dataset)				
Estimator value	Big Data Time (min)	Local Time (min)	Local Accuracy	Big data Accuracy
500	40.8	Error	NA	89.5
250	24.86	32.85	88.853	87
150	18.38	20.56	88.93	88.98
100	9.74	10.35	89.3	88.87

Figure 110: System Error with huge Dataset(1GB)