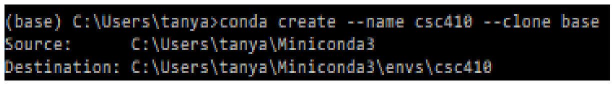
EXTRACTING FEATURES FROM THE RAW DATA FOUND IN A SET OF FRUIT IMAGES AND CONVERTING THEM INTO A FEATURE SPACE

Eric Cortes-Aguilera
University of North Carolina at Greensboro
Greensboro, NC 27402, USA
e$_cortes@uncg.edu$

***Abstract*—Big Data and machine learning are two interesting facets of the Computer Science field. These two facets are constantly innovating and improving allowing many to develop more and more advanced programs of different types. As part of the Big Data and Machine Learning course CSC 410 taught at the University of North Carolina at Greensboro by Dr. Shan Suthaharan, I have been given the opportunity to explore these two aspects of the Computer science field through a semester long assignment. This assignment involves using three sets of different fruit images and various machine learning algorithms to help create a program that can distinguish between all three.This article explains the process of accomplishing this objective through step by step tasks.**

I. Build the Programming Environment

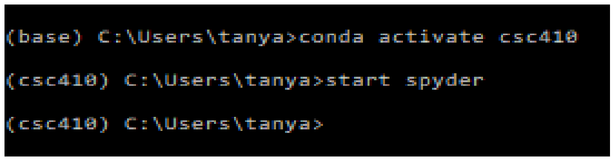
The programming environment chosen to accomplish the objective of this assignment was python version 3.7.2 via the software Anaconda. Through the anaconda software a virtual environment was created called csc410 which was a clone of the previously existing base environment as can be seen in Fig 1.



```
(base) C:\Users\tanya>conda create --name csc410 --clone base
Source:      C:\Users\tanya\Miniconda3
Destination: C:\Users\tanya\Miniconda3\envs\csc410
```

Fig. 1. Creating an environment by cloning the base.

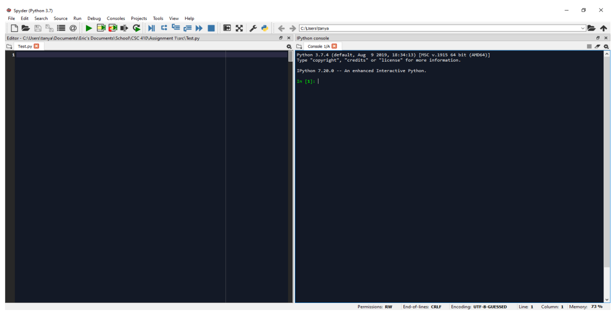
All the programming for this assignment took place on this virtual environment as can be seen in Fig 2. Since this virtual environment was a clone of the base environment and the base environment was previously used for other classes, all of the needed packages were already installed. These packages include: os, opencv, numpy, pandas, matplotlib, mpltoolkits, and spyder.



```
(base) C:\Users\tanya>conda activate csc410

(csc410) C:\Users\tanya>start spyder

(csc410) C:\Users\tanya>
```

Fig. 2. Starting the environment.

The chosen IDE to accomplish the objective of this assignment was the Sypder IDE as can be seen in Fig 3. Jupyter notebook could have also been used and was previously installed but for this assignment it was not used.

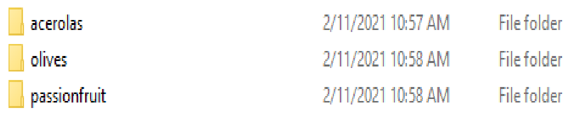


```
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.20.0 -- An enhanced Interactive Python.

In [1]:
```

Fig. 3. Programming IDE.

II. Download Dataset

The fruit data sets used were obtained from an online website that had a folder called Fruit Image Data Set (FIDS30) which contained approximately 30 different sets of fruit images [1]. From this folder three different types of fruits were chosen to accomplish the objective of this assignment. Those as seen in Fig 4 were the acerolas, the olives, and the passion fruits.



acerolas	2/11/2021 10:57 AM	File folder
olives	2/11/2021 10:58 AM	File folder
passionfruit	2/11/2021 10:58 AM	File folder

Fig. 4. Folder containing files of images.

The acerolas (Fig 5) and olives (Fig 6) both have a strong similarity in shape in size and to the naked eye they also appear to have the same number of features. This is why they were chosen, since it would pose itself as a challenge to the machine learning program. This would be the harder test for the machine, being able to distinguish between both of these fruits. The passion fruits (Fig 7) were chosen as to be the easier test for the machine learning program. This fruits are different to the acerolas and olives in shape, size, and appearance. Since the passion fruits contain that very distinguishable center it should be an easier task for the program to distinguish between this fruit and the rest. The passion fruit also appears to contain more features which would thus help the program even more. The distinction between the passion fruits and the other fruits would be the easier task while distinguishing between the acerolas and olives would be the harder task.

Fig. 5. Images of acerolas.

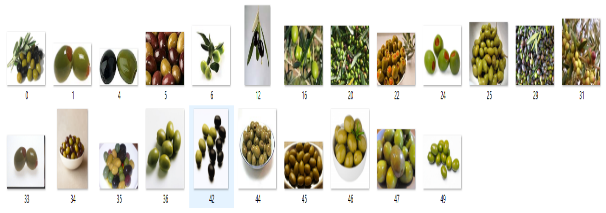


Fig. 6. Images of olives.

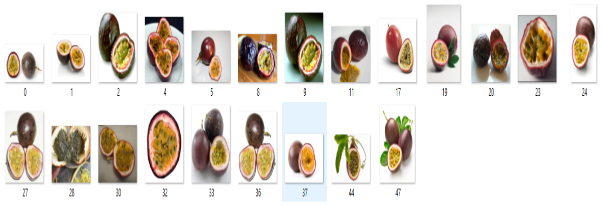
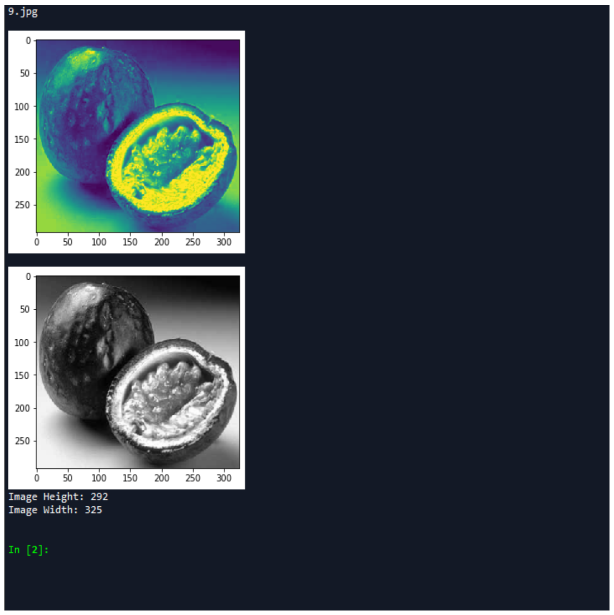


Fig. 7. Images of passion fruits.

These three fruits were also chosen out of the rest of the fruits do to their low image number count. Do to the lack of processing power and ram available on the machine in which this program is being built. The low number of images would also facilitate the process of achieving the objective. This might hinder the effectiveness of the program however since this is a learning assignment it shouldn't be such a big issue. Also there were not many options due to the lack of processing power and ram of the available computer this program is being built on.

III. Read and Display Images in the Environment

To read and display the selected images from the online fruit image data set a function was created called displayImages that handles this task. This function takes in an directory as a parameter then loops through all the files (images) in that directory using the package os. The address of each image is obtained by combining the directory and filename together. Using the opevcv package the image is the read from the computer. After that the filename is printed along with the GRB channel images using myplotlib. The image is converted to grey scale using opencv and then displayed again using matplotlib. The width and height of the grey image is printed and then the image is saved to a folder. Fig 8 shows the what the function does to an image. The directory is passed into the function, the image is read under that directory, the RGB are displayed, then converted to grey scale, displayed in grey scale, then the dimensions are displayed, and finally the image is saved. This is repeated for each image in the directory.



```
9.jpg
```

```
Image Height: 292
Image Width: 325


In [2]:
```

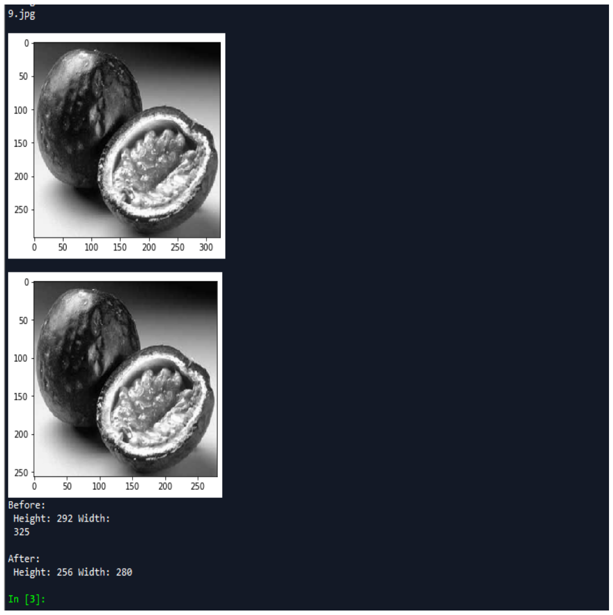
Fig. 8. Results for displaying images.

IV. Resizing the Dimensions of the Images

To resize the images another function was created similarly to the function that reads and displays the images. However this function first prints the filename then reads and displays the original image. Following that then the size of the new image is calculating the ratio from the original height to the new height of 256. That ratio is then multiplied by the width to get the new width however since the new width needs to be divisible by 8, the modular function is used to see how many pixels need to be removed form the width. After removing those pixels from the width the image is resized using opencv. The image is then displayed with matplotlib and its before as well as after dimensions are also printed. An example of this is seen in Fig 9 however this is repeated automatically for each image of each kind.

V. Generate Block-Feature Vectors

After resizing the images to be of height of 256 and the width being divisible by 8 now block feature vectors can be extracted to be one step closer to the final objective. To do this another function was created. Like the previous functions this function also obtains an address and reads each image in that address one by one. The function gets the image address, reads the image, and converts the image into 2D. This process

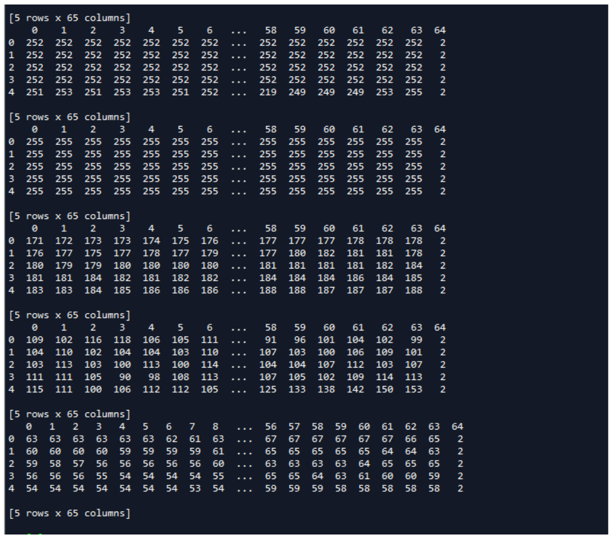


```
9.jpg
```

```
Before:
 Height: 292 Width:
 325

After:
 Height: 256 Width: 280

In [3]:
```

Fig. 9. Results for resizing images.

was already done when the images were converted to gray scale however when saving the images to a folder they regain their 3D shape. This is why they need to be converted once again in this function into 2D with the help of opencv. The height and width of the image is then obtained and two for loops that loop through the image in blocks of 8x8 pixels extracting features vectors. The feature vectors are stored in a data frame using pandas as rows on a spreadsheet with the last column being the result of the feature. 0 for the first fruit, 1 for the second fruit, and 2 for the third fruit. The first 5 rows of the spreadsheet are printed. Then the spreadsheet is then saved onto the computer in a new address. An example of this process taking place can be seen in Fig 10.



```
[5 rows x 65 columns]
     0    1    2    3    4    5    6  ...   58   59   60   61   62   63  64
0  252  252  252  252  252  252  252  ...  252  252  252  252  252  252   2
1  252  252  252  252  252  252  252  ...  252  252  252  252  252  252   2
2  252  252  252  252  252  252  252  ...  252  252  252  252  252  252   2
3  252  252  252  252  252  252  252  ...  252  252  252  252  252  252   2
4  251  253  251  253  253  251  252  ...  219  249  249  249  253  255   2

[5 rows x 65 columns]
     0    1    2    3    4    5    6  ...   58   59   60   61   62   63  64
0  255  255  255  255  255  255  255  ...  255  255  255  255  255  255   2
1  255  255  255  255  255  255  255  ...  255  255  255  255  255  255   2
2  255  255  255  255  255  255  255  ...  255  255  255  255  255  255   2
3  255  255  255  255  255  255  255  ...  255  255  255  255  255  255   2
4  255  255  255  255  255  255  255  ...  255  255  255  255  255  255   2

[5 rows x 65 columns]
     0    1    2    3    4    5    6  ...   58   59   60   61   62   63  64
0  171  172  173  173  174  175  176  ...  177  177  177  178  178  178   2
1  176  177  175  177  178  177  179  ...  177  180  182  181  181  178   2
2  180  179  179  180  180  180  180  ...  181  181  181  181  182  184   2
3  181  181  184  182  181  182  182  ...  184  184  184  186  184  185   2
4  183  183  184  185  186  186  186  ...  188  188  187  187  187  188   2

[5 rows x 65 columns]
     0    1    2    3    4    5    6  ...   58   59   60   61   62   63  64
0  109  102  116  118  106  105  111  ...   91   96  101  104  102   99   2
1  104  110  102  104  104  103  110  ...  107  103  100  106  109  101   2
2  103  113  103  100  113  100  114  ...  104  104  107  112  103  107   2
3  111  111  105   90   98  108  113  ...  107  105  102  109  114  113   2
4  115  111  100  106  112  112  105  ...  125  133  138  142  150  153   2

[5 rows x 65 columns]
    0   1   2   3   4   5   6   7   8  ...  56  57  58  59  60  61  62  63  64
0  63  63  63  63  63  63  62  61  63  ...  67  67  67  67  67  67  66  65   2
1  60  60  60  60  59  59  59  59  61  ...  65  65  65  65  65  64  64  63   2
2  59  58  57  56  56  56  56  56  60  ...  63  63  63  63  64  65  65  65   2
3  56  56  56  55  54  54  54  54  55  ...  65  65  64  63  61  60  60  59   2
4  54  54  54  54  54  54  54  53  54  ...  59  59  59  58  58  58  58  58   2

[5 rows x 65 columns]
```

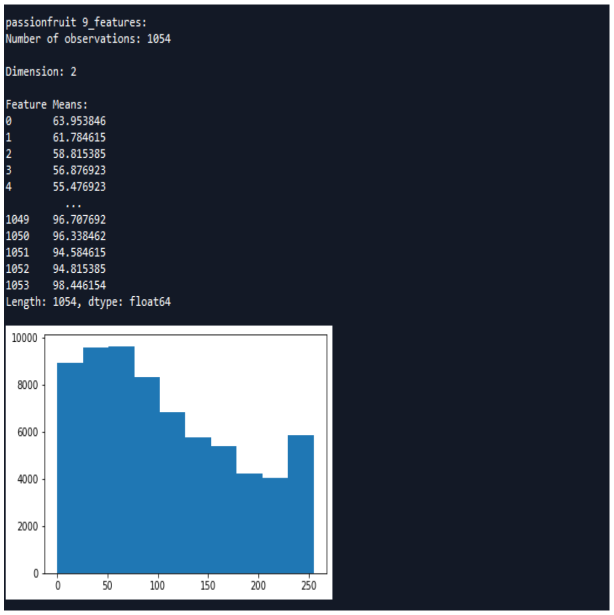
Fig. 10. Results for generating block-feature vectors.

VI. Generate Sliding Block-Feature Vectors

Due to the time constraint on this assignment this particular task was unable to be accomplished. The processing power of the computer of which this program was developed on probably would not be able to support such computationally intensive requirement. This task may be attempted later and if able to get a hold of a machine capable of running such process.

VII. Derive Statistical Descriptors

Now that the block-feature vectors were extracted some statistical information can be derived from them. A function was created to accomplish this task. The function gets the address of each spreadsheet, prints the fruit name and number then converts it into a data frame using pandas. The number of observations (rows) are printed along with the dimension of the data, and the means of each feature vector. Lastly a histogram is made with the data from the spreadsheet. This is done for each spreadsheet. Fig 11 is just one iteration of the execution of the function. This represents only one spreadsheet of feature vectors that was created. Due to having reduced space in this document the rest of the iterations of the execution of this function were not able to be included.



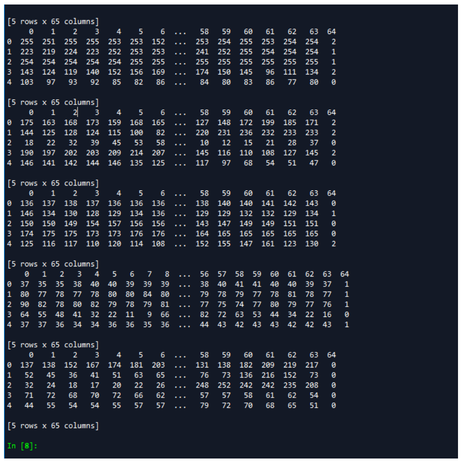
```
passionfruit 9_features:
Number of observations: 1054

Dimension: 2

Feature Means:
0       63.953846
1       61.784615
2       58.815385
3       56.876923
4       55.476923
          ...
1049    96.707692
1050    96.338462
1051    94.584615
1052    94.815385
1053    98.446154
Length: 1054, dtype: float64
```

Fig. 11. Results for displaying statistical descriptions.

Even though not all iterations of the execution of this function could be seen if the python code which is included with this document is ran then you can observe the other iterations as well. Some conclusions can be derived from the statistical data extracted from the execution of this function. When the function is executed one can observe that the number of observations differ in each spreadsheet ranging from 600 observations up to 2000 observations per spreadsheet. This means that the data set is imbalanced due to the variety of the observations. The data however is not inaccurate or incomplete as can be seen through the averages of each spreadsheet displayed. No data or feature result is missing which is why we can derive this conclusion. When executing the function one can observe that there is however one common thing between all the spreadsheets and that is their dimension of 2. Due to this we cannot classify the data as big data due to the low dimension of the data. The data can be most likely classified as large data due to the large number of observations. Since the features are not constantly changing or uncontrollably/continuously growing then the data has no scalability issue. Since the feature space hasn't been established yet and all we have is a histogram to observe then we cannot tell if the data needs to be standardized or normalized. However observing task 9 with the display of the sub spaces of the feature spaces we can clearly observe that standardization and normalization is needed.

VIII. Construct a Feature Space

To construct a feature space of the feature vector spreadsheets for all three images a function was created to do this task. The first thing this function does is obtains a list of the directories of the folders containing the spreadsheets for each fruit. Then using the package os the addresses for each spreadsheet as saved and stored into 3 different lists for each fruit. Then the feature space for the first two fruits the acerolas and olives are made. this is done by first determining which fruit has less spreadsheets and stores the number of spreadsheets that fruit has. This is the number for the iterations of the loop. This is to prevent any issues of trying to create a feature space with one spreadsheet non existent. The loop then reads both files of each fruit and stores then as data frames. Those data frames are stored as a list then using pandas they are concatenated into into one data frame. A numpy array is then made of the length of the newly concatenated data frames which will serve as the new indexes of the final data frame. The indexes are randomized and then the merged data frame containing both fruit spreadsheets is added to the mixed indexes to create a randomized feature space. The head of the feature space is printed for visual representation and lastly it is saved. The same process is repeated for creating the feature space of all three fruits the acerolas, olives, and passion fruits. The only distinction in this loop is that now you read three different data frames and merge those three different data frames. The number for the iterations of the loop also changes since now it has to find the mini mun number of files between all three fruits. Other then that the same process holds but in its very own loop. The results of this function can be seen in Fig 12. Here one can observe the different feature spaces, one with two fruits (0, 1) and one with all three fruits (0, 1, 2). If you observe the results of the feature spaces (last column) you can see one has two different numbers and one has all three different number. You can also observe the randomization of them as well.

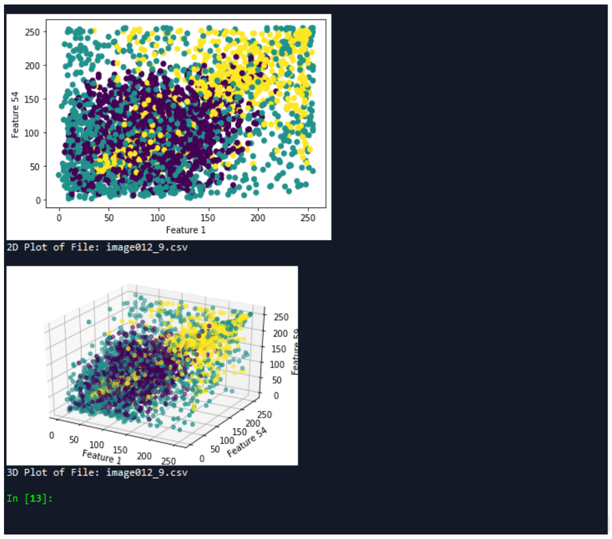


```
[5 rows x 65 columns]
     0    1    2    3    4    5    6  ...   58   59   60   61   62   63  64
0  255  251  255  255  253  253  152  ...  253  254  255  253  254  254   2
1  223  219  224  223  252  253  253  ...  241  252  255  254  254  254   1
2  254  254  254  254  254  255  255  ...  255  255  255  255  255  255   1
3  143  124  119  140  152  156  169  ...  174  150  145   96  111  134   2
4  103   97   93   92   85   82   86  ...   84   80   83   86   77   80   0

[5 rows x 65 columns]
     0    1    2    3    4    5    6  ...   58   59   60   61   62   63  64
0  175  163  168  173  159  168  165  ...  127  148  172  199  185  171   2
1  144  125  128  124  115  100   82  ...  220  231  236  232  233  233   2
2   18   22   32   39   45   53   58  ...   10   12   15   21   28   37   0
3  190  197  202  203  209  214  207  ...  145  116  110  108  127  145   2
4  146  141  142  144  146  135  125  ...  117   97   68   54   51   47   0

[5 rows x 65 columns]
     0    1    2    3    4    5    6  ...   58   59   60   61   62   63  64
0  136  137  138  137  136  136  136  ...  138  140  140  141  142  143   0
1  146  134  130  128  129  134  136  ...  129  129  132  132  129  134   1
2  150  150  149  154  157  156  156  ...  143  147  149  149  151  151   0
3  174  175  175  173  173  176  176  ...  164  165  165  165  165  165   0
4  125  116  117  110  120  114  108  ...  152  155  147  161  123  130   2

[5 rows x 65 columns]
    0   1   2   3   4   5   6   7   8  ...  56  57  58  59  60  61  62  63  64
0  37  35  35  38  40  40  39  39  39  ...  38  40  41  41  40  40  39  37   1
1  80  77  78  77  78  80  80  84  80  ...  79  78  79  77  78  81  78  77   1
2  90  82  78  80  82  79  78  79  81  ...  77  75  74  77  80  79  77  76   1
3  64  55  48  41  32  22  11   9  66  ...  82  72  63  53  44  34  22  16   0
4  37  37  36  34  34  36  36  35  36  ...  44  43  42  43  43  42  42  43   1

[5 rows x 65 columns]
     0    1    2    3    4    5    6  ...   58   59   60   61   62   63  64
0  137  138  152  167  174  181  203  ...  131  138  182  209  219  217   0
1   52   45   36   41   51   63   65  ...   76   73  136  216  152   73   0
2   32   24   18   17   20   22   26  ...  248  252  242  242  235  208   0
3   71   72   68   70   72   66   62  ...   57   57   58   61   62   54   0
4   44   55   54   54   55   57   57  ...   79   72   70   68   65   51   0

[5 rows x 65 columns]

In [8]:
```

Fig. 12. Results for constructing feature spaces.

IX. Display sub spaces

To display the sub spaces of the feature spaces created in the previous task the function first reads the address of the folder containing all the spreadsheets. Then using os it loops through all the spreadsheets. Then the features randomly selected are stored into their respective variables. Feature 1 is stores into the x variable, feature 54 is stored into the y variable, feature 59 is stored into variable z and then the column of the results of the feature space are stored into r. To construct the 2D plot matplotlib is used to create a scatter plot of feature 1 (x variable), feature 54 (y variable), and r is also passed to it to distinguish the observations and color code them. The graph is then plotted. The same process is then repeated for the 3D graph. However this time mpl_toolkits is used and all three variables are passed into the plot along with the results. The scatter plot is then plotted. An example of its execution can be seen in Fig 13.

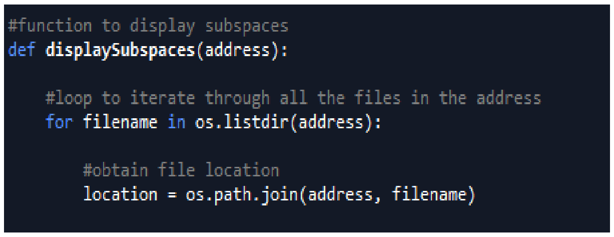


```
2D Plot of File: image012_9.csv

3D Plot of File: image012_9.csv

In [13]:
```

Fig. 13. Results for displaying subspaces.

X. Automation of Python Code

The automation of the source code for each task previously talked about is accomplished in the function for each task. The functions for each task read in a directory and using os a loop is made to loop through every file under that directory and the address of each file is computed by combining the directory with the filename. That allows for the function to run for every file and read every file under that address. All that needs to be done is pass the directory of the folder containing the files needed to the function and calling the function. this is repeated for all the folders that need that function. This is seen in Fig 14.



```
#function to display subspaces
def displaySubspaces(address):

    #loop to iterate through all the files in the address
    for filename in os.listdir(address):

        #obtain file location
        location = os.path.join(address, filename)
```

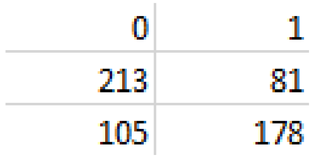
Fig. 14. Source code for automating the source code.

XI. Discussion of Block Size Effects

In this assignment feature vectors of size 64 were extracted from the raw data of 2D gray scaled images. This was achieved through the extraction of block vectors of 8 x 8 size from said images. However, the block size itself plays a huge part. If the block size was increased then the dimensionality of the feature space would increase as well and thus the size of the feature vectors would increase accordingly. However, if the same images were used with the increased block size then the number of vectors or observations or rows of the spreadsheets would decrease. The increase of the dimension of the feature space would help the efficiency of the classifier that divides the domain only if the number of vectors also increases. In this case the number of vectors would not increase which means that the efficiency of the classifier would decrease. In conclusion if the block size increases so does the dimensionality but the number of vectors decreases. If nothing is done to increase the vector count the classifier's efficiency decreases.

XII. Implement a Regression-Based Model

The regression-based model chosen for this part was the lasso regression model on the two-class data sets. The creation of this model did not use any already existing library found in the python repertoire of available libraries. The implementation of this regression model was executed by directing programming a function that executed the lasso regression equation that is wildly available to all. This required execution step by step all the parts of the equation to develop all the coefficients needed for the model. The model was trained on 80 percent of the actual data. Once the model was trained a

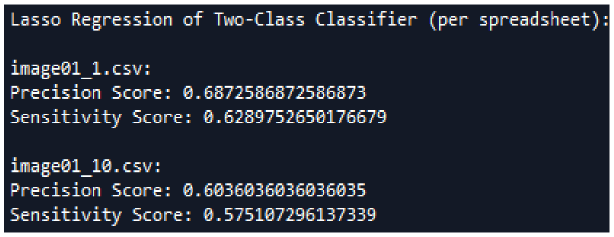


0	1
213	81
105	178

Fig. 15. Confusion matrix (two-class).

confusion matrix was created using the testing data set aside previously as seen in figure 15.

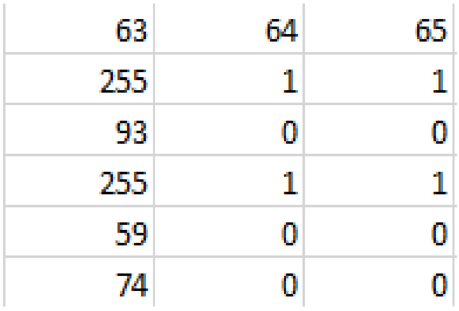
Then with the results of the confusion matrix the precision and sensitivity were calculated. These two metrics were chosen due to the low-dimensional aspect of the data as well as the its aspect of being unbalanced. Accuracy was not able to be used due to the fact that the data is unbalanced. These two metrics were also chosen as to match with the same metrics used later on with a python library. The precision and sensitivity of each of the models created was then displayed as seen in Fig 16.



```
Lasso Regression of Two-Class Classifier (per spreadsheet):

image01_1.csv:
Precision Score: 0.6872586872586873
Sensitivity Score: 0.6289752650176679

image01_10.csv:
Precision Score: 0.6036036036036035
Sensitivity Score: 0.575107296137339
```

Fig. 16. Execution.

Only a few model metric results are showed due to the reduced space in this document. Was was able to be observed in all the data sets was a similarity in precision and sensitivity. All of the data sets ranged between .5 to .9 in the precision category and the sensitivity category was similar as well except that in each case it was a few decimal spaces lower then the precision metric Then the same model was used to predict the labels of all of the data and then that was saved as an extra column (65) of the original spreadsheet as seen in Fig 17.

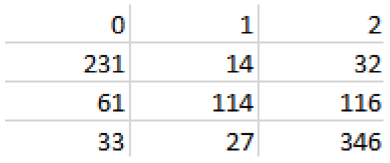


63	64	65
255	1	1
93	0	0
255	1	1
59	0	0
74	0	0

Fig. 17. Feature vectors (two-class) including predicted labels.

XIII. Implement a Random Forest Model

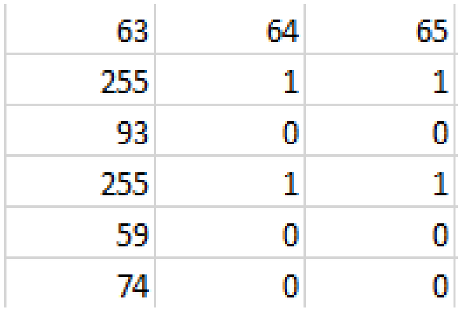
To implement a random forest model the scikit learn library was used. A similar process of execution as of the lasso regression model was implemented. First, the data was separated into 80 percent training and 20 percent testing. Then a random forest model was created using 300 as the number of estimators. Then the model was trained with the training data. Once again the model was then tested using a confusion matrix of the testing data and the precision as well as sensitivity metrics were used again to evaluate the models. This matrix can be seen in Fig 18.



0	1	2
231	14	32
61	114	116
33	27	346

Fig. 18. Confusion matrix (three-class).

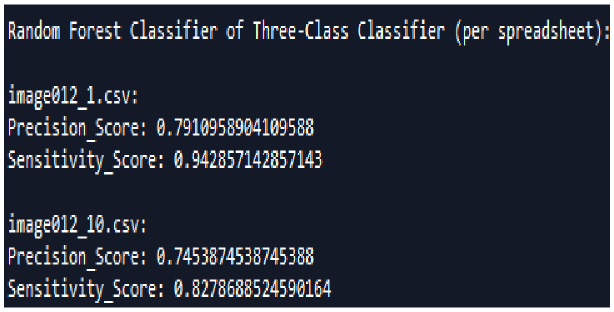
Then the model was used to predict the labels of the entire data spreadsheet and those predicted labels were then once again added to the original spreadsheet under the column number 65 as seen in Fig 19.



63	64	65
255	1	1
93	0	0
255	1	1
59	0	0
74	0	0

Fig. 19. Feature vectors (three-class) including predicted labels.

The precision and sensitivity of each model was also displayed as seen in Fig 20 however these results proved to be higher on average then those of the lasso regression model. This is most likely due to the fact that this model implements three class classifiers while the other model did not.

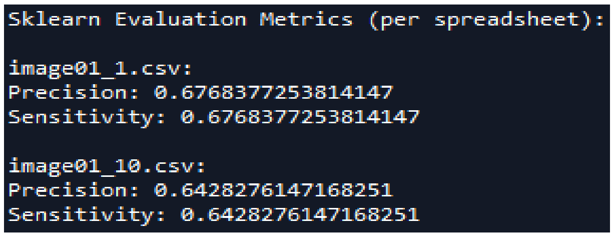


```
Random Forest Classifier of Three-Class Classifier (per spreadsheet):

image012_1.csv:
Precision_Score: 0.7910958904109588
Sensitivity_Score: 0.942857142857143

image012_10.csv:
Precision_Score: 0.7453874538745388
Sensitivity_Score: 0.8278688524590164
```

Fig. 20. Execution.

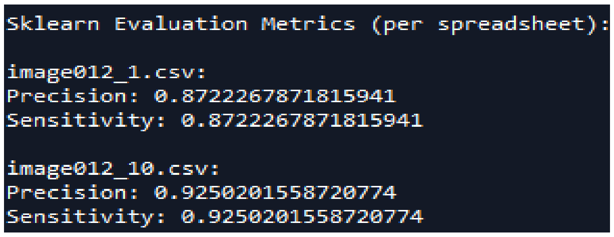
XIV. Evaluation of Learning Models

After creating the lasso regression model as well as the random forest model and using the confusion matrix as a form to derive evaluation metrics, built in measures from python libraries were then used as a form of comparing evaluation metrics. The same metrics precision as well as sensitivity were used as to match with the measures used previously in the confusion matrices. The same scikit learn library was used to obtain the built in metrics used in this part. The execution of these built in evaluation metrics can be seen in Fig 21 and Fig 22.



```
Sklearn Evaluation Metrics (per spreadsheet):

image01_1.csv:
Precision: 0.6768377253814147
Sensitivity: 0.6768377253814147

image01_10.csv:
Precision: 0.6428276147168251
Sensitivity: 0.6428276147168251
```

Fig. 21. Execution of evaluation (two-class).



```
Sklearn Evaluation Metrics (per spreadsheet):

image012_1.csv:
Precision: 0.8722267871815941
Sensitivity: 0.8722267871815941

image012_10.csv:
Precision: 0.9250201558720774
Sensitivity: 0.9250201558720774
```

Fig. 22. Execution of evaluation (three-class).

Fig 21 corresponds to the two class metrics while Fig 22 corresponds to three class metrics. Observing the results of both built in metrics as well as confusion matrix metrics we can observe that with the three class classifiers the built in metrics did indeed prove to be higher then the confusion matrix metrics. This can be because the confusion matrices were made only using new unseen testing data while the built in metrics used the entire seen data. However for the two class models it appears to be that the metrics remained almost the same between the built in and confusion matrix metrics. Looking at the results obtained in this process it can be concluded that the three class classifier with the random forest regression proved to be more effective with both the confusion matrix metrics and even more with the random forest metrics.

XV. SETTING UP DATABRICKS (BIG DATA SYSTEM)

Databricks is a software company founded in 2013, which offers a platform for many people to work on data science, data engineering, data analysis, machine learning and artificial intelligence projects. Databricks provides a big data system that allow users the ability to run programs that normal personal computers may struggle to execute or may require a longer execution time. For this assignment, the databricks software will be used to run a machine learning classifier on the feature spaces that were previously generated in this assignment. Then its performance will then be compared to the performance of a personal computer running the same machine learning classifier.

To use this software an account needed to be made on the databricks website. A community edition account was made since it did not require any subscription to be made to use. This allowed for the use of one database along with 15GB of memory available. After completing the sign up process of linking an email account, specifying a name along with a company name, the tutorial ”Get started as a Databricks Workspace user” which was previously named ”Explore the Quickstart Tuturial” was followed to set up a big data environment. This tutorial can be found under the documentation section on the databricks website.

The first task on the tutorial was to orient oneself with the layout of the platform which was pretty simple due to the layout of the platform. Through this step I was able to learn the layout of the platform which includes a navigation bar on the left hand side of the screen with accessible buttons to all the functions of the system including the data, cluster, and workspace sections. The second thing that was learned from this section was the ability to create a cluster which is the first step needed to run a big data program on the platform. To do this you go to the cluster section of the platform which can be found on the initial page under the common tasks heading called ”create cluster” or through the respective button on the navigation bar. Once on this page the button ”create cluster” was clicked and the option to create a new cluster opened. On here the option was given to chose the version of runtime. My instructor suggested to use the version 7.2 ML (Scala 2.12, Spark 3.0.0) however this option was no longer available so the version 7.4 LTS (Scala 2.12, Spark 3.0.1) was chosen as suggested by the tutorial. After naming the cluster ”csc410” and waiting for the cluster to activate, the feature spaces were then added to the system. The tutorial taught us to do this by going to the data page which can be found the same was as the cluster page. on this page the ”create new table” button was clicked to add the respective spreadsheets. The appropriate feature space was uploaded then the table was created using the create table with UI option which allowed the selection of a cluster which in this case was csc410 the active cluster. For each table the options of using the first row as headers was selected along with the option to infer the datatype. Then one by one each feature space was added to the system each receiving the name of table1, table2, table3... respectively. Lastly the tutorial taught how to create a python notebook which to access this option was similar to finding the create cluster option. Once on the respective site one goes to users then selects the current user and by clicking on the dropdown arrow next to the username an option to create a notebook is given. After creating a python notebook and giving it a name the process of creating a machine learning program was then able to begin.

To begin creating a machine learning program the fist step is to verify that the feature spaces that were uploaded appropriately. To verify this a loop was created to iterate through all the tables created. For each table a sql query was made to obtain the table information as a dataframe. After doing this then pandas was used to create the obtained dataframe into a pandas dataframe. Then from this point a histogram as well as box plot were made of the 23 feature of the feature space. Fig 23 as well as Fig 23 as the outputted results of these plots for the first 4 tables. As can be seen, the data appears to have successfully uploaded when compared to previous plots seen in in this article previously.

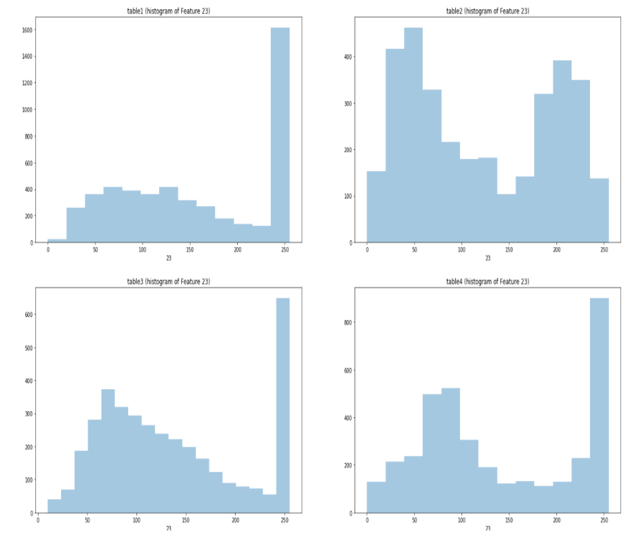


Fig. 23. Histograms of feature spaces.

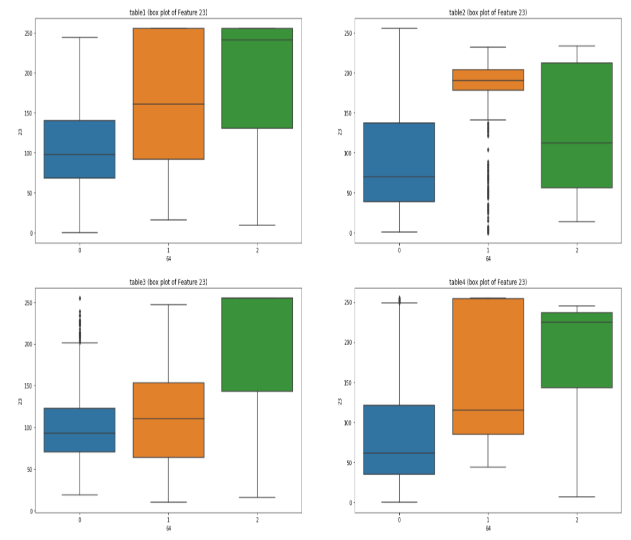
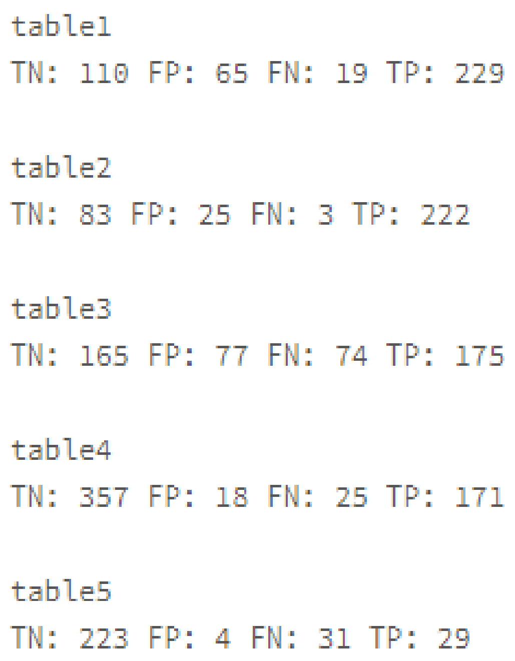


Fig. 24. Box plots of feature spaces.

XVI. IMPLEMENTING RANDOM FOREST ON DATABRICKS

The machine learning algorithm chosen to be programmed on the databricks big data system was the random forest classifier. This was so that its performance can be easily compared to the performance of the random forest classifier which was previously made on the local computer previously in this project. The process of implementing this random forest classifier was similar to the process which was explained previously. First the table was obtained by making a sql query. Then the data was converted to a pandas dataframe to facilitates is use. After this was done then the last column of the dataframes was extracted which was the label column and it was stored as its separate variable. The same was done to the features. Then both feature and label variables were converted to numpy arrays which facilitated the process of obtaining the training data as well as the testing data for both features (x) and labels (y). Once this was done the number of rows of the dataframe was obtained to calculate the row that corresponded to the corresponding train/test split of the data. The train/test data was split into 80 percent training and 20 percent testing. Once the dividing row was obtained the data was split into its respective categories. Once this data was split then a random forest classifier was made using the built in sklearn library in python. The `n_estimators` selected was 300 which was the same number for the classifier used in the local machine to reduce user error when comparing the systems. The classifier was then trained with the training data then tested with the testing data to obtain the predicted y values or in other words the predicted labels. Using the actual labels and the predicted labels from the classifier a confusion matrix was made. This is seen in Fig 25.



```
table1
TN: 110 FP: 65 FN: 19 TP: 229

table2
TN: 83 FP: 25 FN: 3 TP: 222

table3
TN: 165 FP: 77 FN: 74 TP: 175

table4
TN: 357 FP: 18 FN: 25 TP: 171

table5
TN: 223 FP: 4 FN: 31 TP: 29
```

Fig. 25. Confusion matrices of feature spaces.

After the confusion matrix was made its values were used to calculate the precision as well as sensitivity of the model. Which can be seen in Fig 29. Lastly the feature importance was obtained and displayed on a plot as seen in Fig 26. The ROC curve of the model was also obtained and displayed as seen in Fig 27.

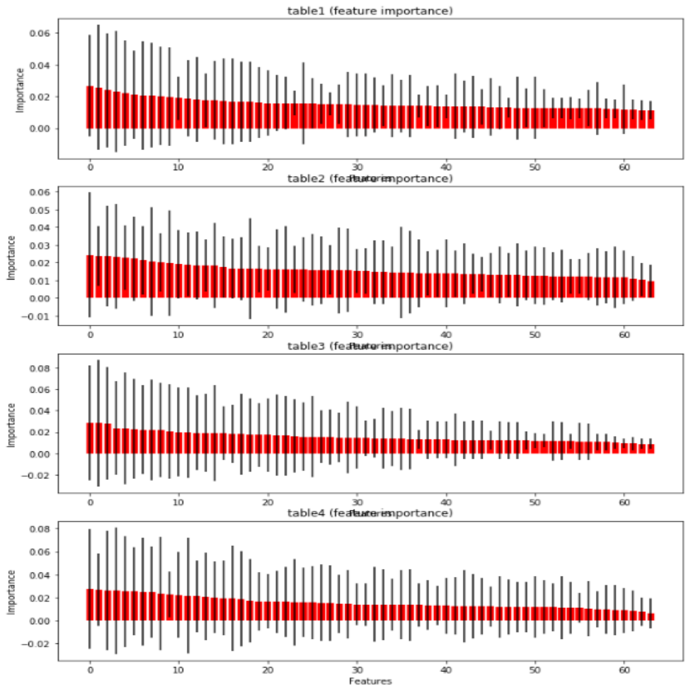


Fig. 26. Feature importance of feature spaces.

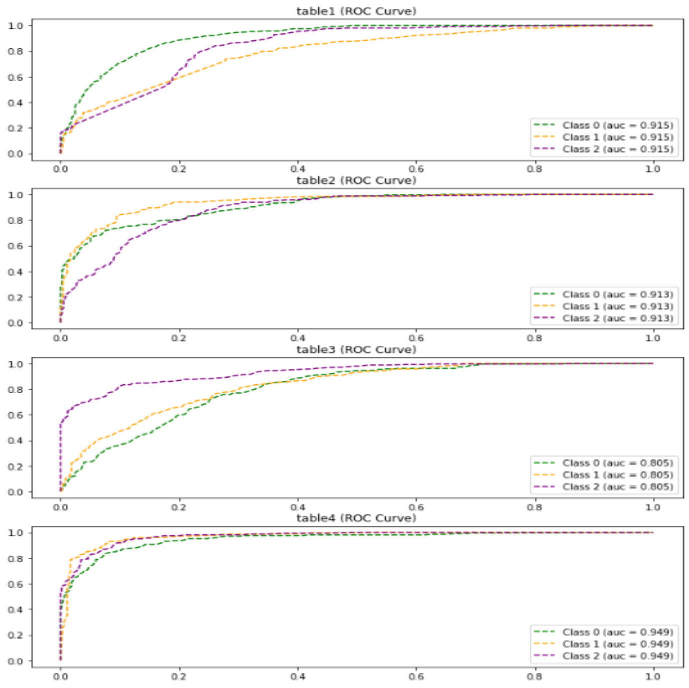


Fig. 27. ROC curves of feature spaces.

XVII. Comparing Local Results vs. Databricks Results (Big Data System)

Looking at the results from the ROC curve plot as well as the feature importance plot we can observe close similarities between the local machine running the random forest classifier and the databricks big data system running the random forest classifier. They appear to very similar if not the same and this can be proven using the results obtained from the confusion matrices. Fig 28 displays the results of the measures obtained from the local machine and Fig 29 displays the results from the databricks big data system. These results are very similar and almost exact in some cases. The difference is not significant. For example looking at the results produced by the databricks system in comparison to the local machine, table one had a better precision score however had a lower sensitivity score in comparison to the local machine. Table two had a lower precision score and sensitivity score. Table three had a higher precision score but lower sensitivity score. Lastly, table four had a higher precision score but lower sensitivity score. However all of these differences were very insignificant to the point that no real distinction could be made between the local and the big data. However one significant difference was the execution time both systems took. The databricks system had a faster execution time then the local machine did. The databricks system took 2.02 seconds to execute the program while the local machine took more then 5 minutes to execute. The local machine used was a machine that had an 2009 Intel Pentium with 4GB of ram which has only 2GB available. This was due to the lack of resources to obtain a better local machine. This big data system demonstrated to have a higher capacity to run more efficiently. Although the results did not point to any clear distinction if a more complex machine learning algorithm would be used like a deep learning algorithm the distinction could be more clear.

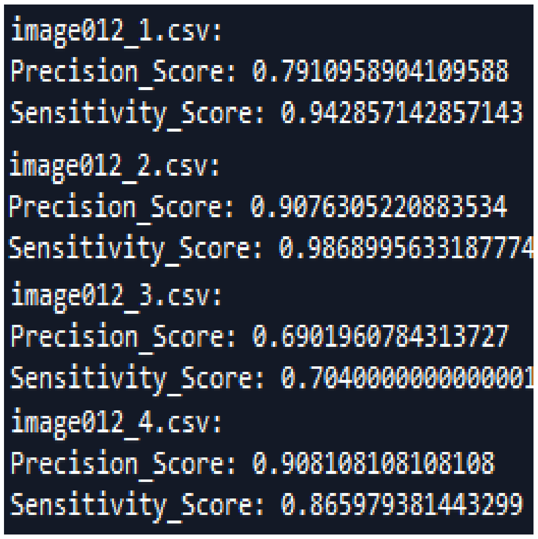

```
image012_1.csv:
Precision_Score: 0.7910958904109588
Sensitivity_Score: 0.942857142857143
image012_2.csv:
Precision_Score: 0.9076305220883534
Sensitivity_Score: 0.9868995633187774
image012_3.csv:
Precision_Score: 0.6901960784313727
Sensitivity_Score: 0.7040000000000001
image012_4.csv:
Precision_Score: 0.908108108108108
Sensitivity_Score: 0.865979381443299
```

Fig. 28. Measures for local system running random forest models.

XVIII. Conclusion

Through this assignment features were extracted from pictures of fruits.Then many machine learning algorithms including those of lasso regression and random forest both locally as well as on a big data system were used to distinguish between different fruit types. All of these algorithms had great results proving to be efficient at classifying the fruits. This assignment

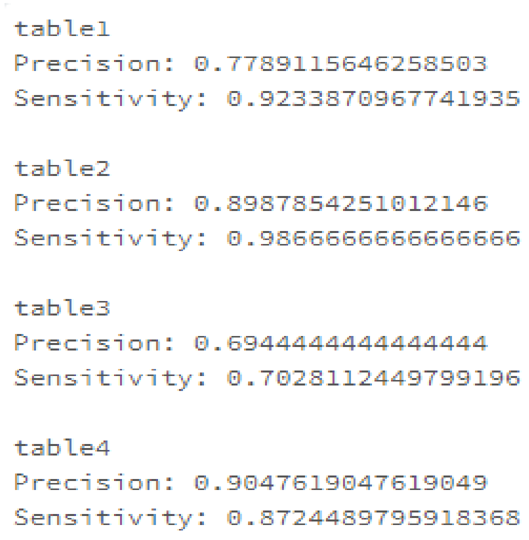

```
table1
Precision: 0.7789115646258503
Sensitivity: 0.9233870967741935

table2
Precision: 0.8987854251012146
Sensitivity: 0.9866666666666666

table3
Precision: 0.6944444444444444
Sensitivity: 0.7028112449799196

table4
Precision: 0.9047619047619049
Sensitivity: 0.8724489795918368
```

Fig. 29. Measures for databricks system running random forest models.

as part of the Big Data and Machine Learning course (CSC 410) taught at the University of North Carolina at Greensboro by Dr. Shan Suthaharan, expanded my knowledge of Big Data and Machine learning. This assignment was very beneficial educationally and much was learned. This was a very neat, fun and educational assignment all thanks to the University of North Carolina at Greensboro as well as thanks to Dr. Shan Suthaharan.

References

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