UWE Bristol University of the West of England

UWEcyber

Gold Award

in association with National Cyber Security Centre

Department for Digital, Culture, Media & Sport

Academic Centre of Excellence in **Cyber Security Education**

UFCFEL-15-3 Security Data Analytics and Visualisation

Portfolio Assignment 2: Machine Learning for Malware Analysis (2022)

The completion of this worksheet is worth a **maximum of 35 marks** towards your portfolio assignment for the UFCFEL-15-3 Security Data Analytics and Visualisation (SDAV) module.

Brief

In this task, you have been given a large sample of derived malware features that describe 14 different malware variants (2000 samples of each). The purpose of this task is to understand the underlying concepts of classification, and **your task will be to develop two classifiers that can classify malware varients**. The first part will focus on a small hand-made classifier using only 3 malware classes, to understand the principles of search space and minimisation of a function. The second part will focus on using off-the-shelf libraries to scale up the classification to all 14 classes of malware present in the dataset.

Assessment and Marking

For each question you will see the maximum number of marks you may be awarded for a complete answer in brackets.

Part 1: Developing a Classifier "by hand" - (Total Marks: 20)

- **Task 1:** Find the Centroid point of each of the three groups (3)
- **Task 2:** Plot the centroids on a Scatter Plot against the train data colour-coded by group (3)

- **Task 3:** For each item in test_data, measure the distance to each centroid point, assign membership to the group of minimum distance, and compare with the expected test data label to obtain a score of successful classifications (12)
- **Task 4:** Provide a final accuracy score for the performance of your "by hand" classifier (2)

Part 2: Developing a large-scale ML classifier - (Total Marks: 15)

- **Task 5:** Scale the Input Features for further processing using the StandardScaler function (1)
- **Task 6:** Obtain numerical labels for each class using the LabelEncoder function (1)
- **(Advanced) Task 7:** Prepare the dataset for ML testing, using the Train-Test-Split function of sklearn (2)
- **(Advanced) Task 8:** Use a Multi-Layer Perceptron (MLP) classifier to train a machine learning model, and obtain the accuracy score against your test data. (4)
- **(Advanced) Task 9:** Use a Random Forest (RF) classifier to train a machine learning model, and obtain the accuracy score against your test data. (4)
- **(Advanced) Task 10:** Show how ML parameters can improve the models to achieve a high accuracy score of over 80% (3)

This assignment should be submitted as PDF to your Blackboard portfolio submission as per the instructions in the assignment specification available on Blackboard. A copy of your work should also be provided via a UWE Gitlab repository, with an accessible link provided with your portfolio.

Contact

Questions about this assignment should be directed to your module leader (Phil.Legg@uwe.ac.uk). You can use the Blackboard Q&A feature to ask questions related to this module and this assignment, as well as the on-site teaching sessions.

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [3]:

```
features = pd.read_csv('./T2_data/malware_data.csv', header=None)
features
```

Out [3]:

In [4]:

```
labels = pd.read_csv('./T2_data/malware_label.csv', header=None)
labels = labels.drop(0, axis=1)
labels = labels.rename(columns = {1:'label'})
labels
```

Out [4]:

In the cells above, we have created two DataFrames: ***features*** and ***labels***.

Features: This table contains 28000 instances of malware, where each instance of malware is characterised by 256 distinct features relating to how it performs and its impact on the associated systems.

Labels: This table contains 28000 rows, where each row is the label of the malware class, related to the features table. There are 2000 samples of each malware varient, and 14 varients in total.

Part 1: Developing a Classifier "by hand"

In [5]:

```
# DO NOT MODIFY THIS CELL - this cell is splitting the data to provide a
suitable subset of data to work with for this task.
# If you change this cell your output will differ from that expected and
could impact your mark.


mal1_index = 17000
mal2_index = 21000
mal3_index = 12000
mal_range = 50
mal_test_range = 30

train_data = np.vstack([
features[mal1_index:mal1_index+mal_range][[0,1]].values,
features[mal2_index:mal2_index+mal_range][[0,1]].values,
features[mal3_index:mal3_index+mal_range][[0,1]].values ])
train_data = pd.DataFrame(train_data)
train_labels = np.vstack([ labels[mal1_index:mal1_index+mal_range].values,
labels[mal2_index:mal2_index+mal_range].values,
labels[mal3_index:mal3_index+mal_range].values ])
train_labels = pd.DataFrame(train_labels)
train_data['labels'] = train_labels
train_data = train_data.rename(columns={0:'x', 1:'y'})

test_data = np.vstack([
features[mal1_index+mal_range:mal1_index+mal_range+mal_test_range][[0,1]].val
ues,
features[mal2_index+mal_range:mal2_index+mal_range+mal_test_range][[0,1]].val
ues,
features[mal3_index+mal_range:mal3_index+mal_range+mal_test_range][[0,1]].val
ues ])
test_data = pd.DataFrame(test_data)
test_labels = np.vstack([
labels[mal1_index+mal_range:mal1_index+mal_range+mal_test_range].values,
labels[mal2_index+mal_range:mal2_index+mal_range+mal_test_range].values,
labels[mal3_index+mal_range:mal3_index+mal_range+mal_test_range].values ])
test_labels = pd.DataFrame(test_labels)
test_data['labels'] = test_labels
test_data = test_data.rename(columns={0:'x', 1:'y'})
```

```
train_data
```

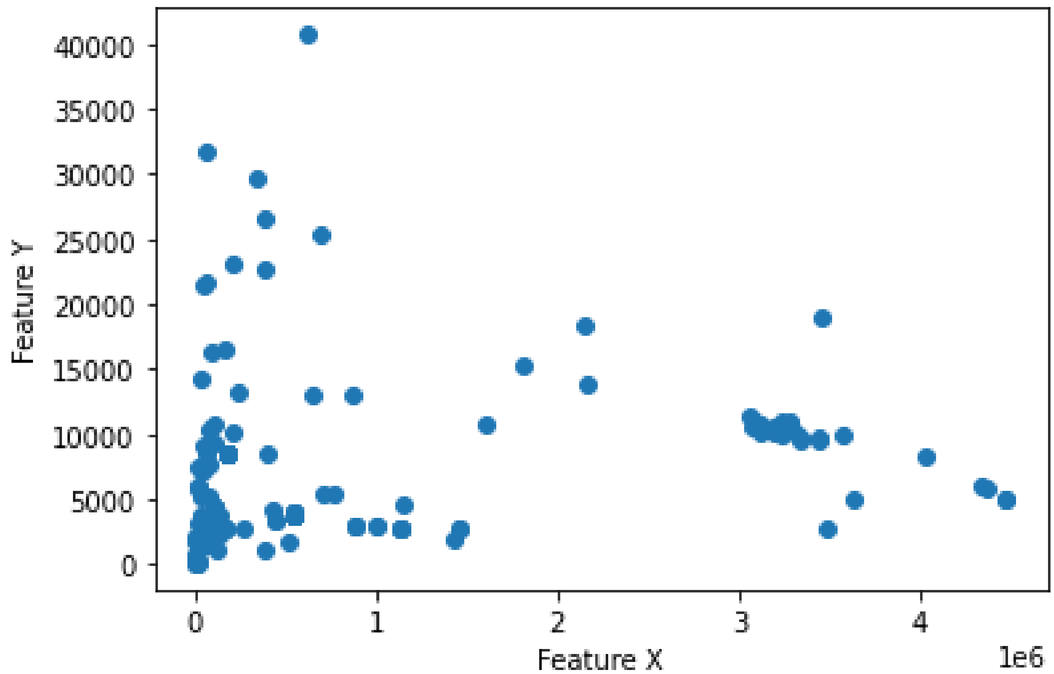
Out [5]:

In [7]:

```
plt.scatter(train_data['x'], train_data['y'])
plt.xlabel('Feature X')
plt.ylabel('Feature Y')
```

Out [7]:

```
Text(0, 0.5, 'Feature Y')
```



Task 1: Find the Centroid point of each of the three groups (3)

```
def get_centroid(mal_index,mal_range):
    points=features[mal_index:mal_index+mal_range][[0,1]].values
    x =  [p[0] for p in points]
    y =  [p[1] for p in points]
    centroid = (sum(x) / len(points), sum(y) / len(points))
    return centroid

centroids=[]
for i in range(3):
     centroids.append(get_centroid(indices[i], mal_range))
centroids=np.array(centroids)

print("Centroid Point for Dataset 1 is: ("+str(centroids[0,0])+" , 
"+str(centroids[0,1])+")")
print("Centroid Point for Dataset 2 is: ("+str(centroids[1,0])+" , 
"+str(centroids[1,1])+")")
```

```
print("Centroid Point for Dataset 3 is: ("+str(centroids[2,0])+" ,
"+str(centroids[2,1])+")")
```

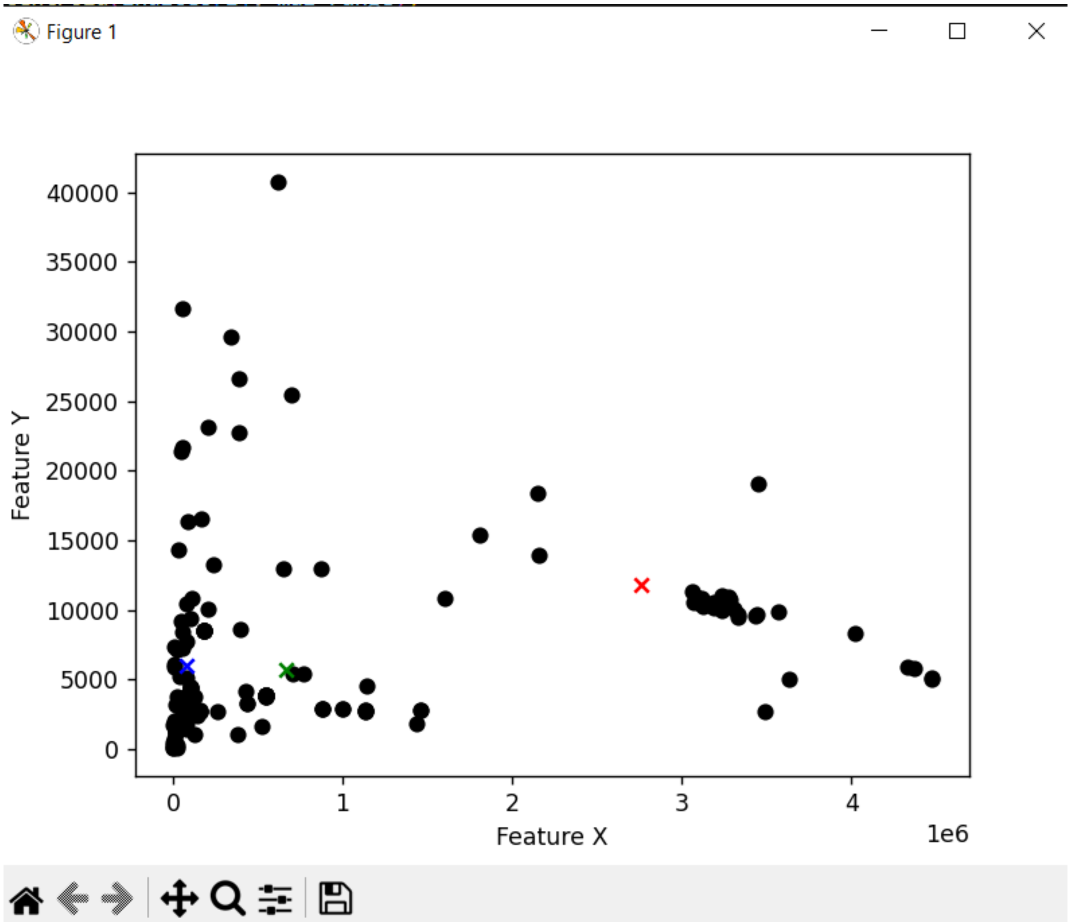
Output:

```
[150 rows x 3 columns]
Centroid Point for Dataset 1 is: (2181660.66 , 11087.1)
Centroid Point for Dataset 2 is: (478778.12 , 3754.04)
Centroid Point for Dataset 3 is: (100505.22 , 6158.28)
```

Task 2: Plot the centroids on a Scatter Plot against the train data colour-coded by group (3)

```
plt.scatter(centroids[0,0],centroids[0,1],marker='x',color='r')
plt.scatter(centroids[1,0],centroids[1,1],marker='x',color='g')
plt.scatter(centroids[2,0],centroids[2,1],marker='x',color='b')

plt.show()
```



Task 3: For each item in test_data, measure the distance to each centroid point, assign membership to the group of minimum distance, and compare with the expected test data label to obtain a score of successful classifications (12)

Hint: You may find the clustering activity worksheet helpful for how to approach this task

```
def find_groups(centroids,all_data,train_data):
    group1=[]
    group2=[]
    group3=[]
    groups=[group1,group2,group3]
    score=0
    for i in range(all_data.shape[0]):
        distance1=np.sqrt(np.abs(all_data[i , 0] - centroids[0,0]) ** 2 +
np.abs(all_data[i,1] - centroids[0,1]) ** 2)
        distance2=np.sqrt(np.abs(all_data[i , 0] - centroids[1,0]) ** 2 +
np.abs(all_data[i,1] - centroids[1,1]) ** 2)
        distance3=np.sqrt(np.abs(all_data[i , 0] - centroids[2,0]) ** 2 +
np.abs(all_data[i,1] - centroids[2,1]) ** 2)
        distances=[distance1, distance2, distance3]
```

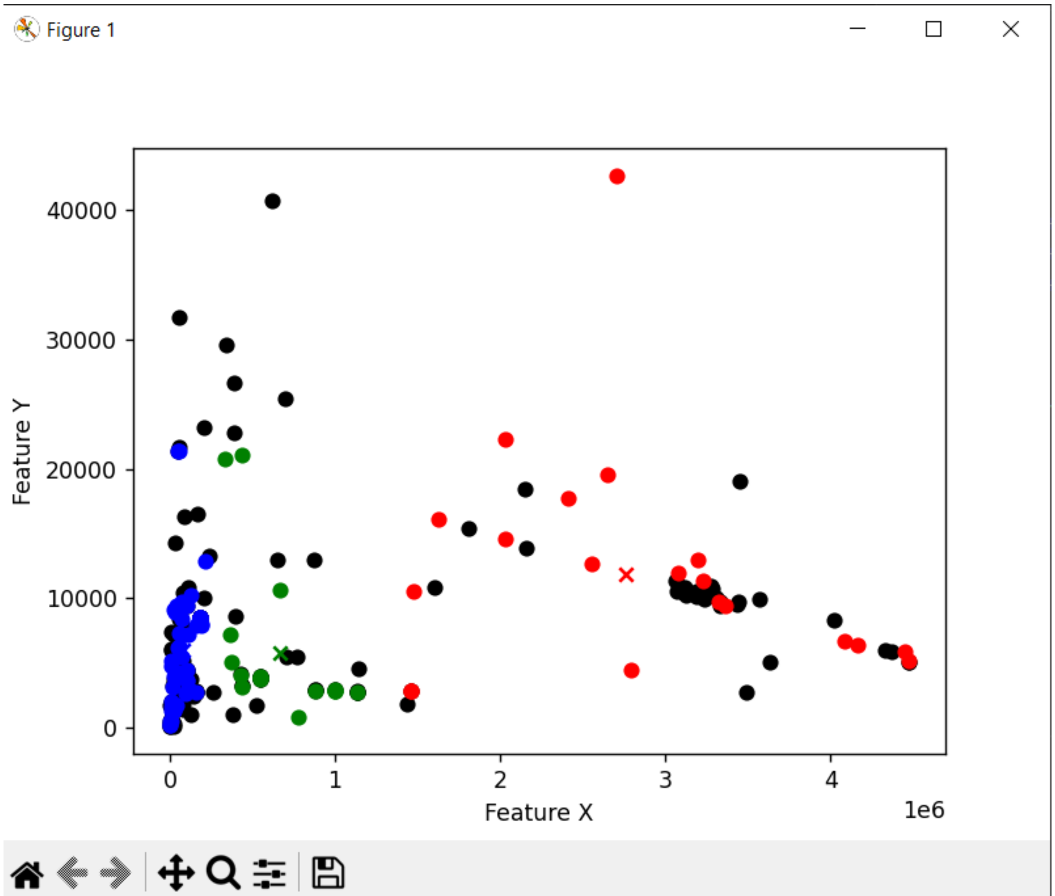
```python
        index=np.argmin(distances)
        label_test=all_data[i , 2]
        label_centroid=train_data[index * mal_range,2]
        if(label_test == label_centroid):
            score+=1
        groups[index].append([all_data[i,0] , all_data[i,1]])
    group1=np.array(group1)
    group2=np.array(group2)
    group3=np.array(group3)
    return group1,group2,group3,score

train_data2=np.array(train_data)
test_data2=np.array(test_data)

group1,group2,group3,score=find_groups(centroids,test_data2,train_data2)
print("Score of Success of Classifier is: ("+str(score)+" )")
centroids=np.array([np.mean(group1,axis=0), np.mean(group2,axis=0),
np.mean(group3,axis=0)]) #update centroid

plt.scatter(group1[:,0], group1[:,1], color='r')
plt.scatter(group2[:,0], group2[:,1], color='g')
plt.scatter(group3[:,0], group3[:,1], color='b')

plt.show()
```




```
Score of Success of Classifier is: (63 )
```

Task 4: Provide a final accuracy score for the performance of your "by hand" classifier (2)

```
score=score/len(test_data2)
print("Accuracy of Classifier is : ("+str(score)+" )") #(Total No. of correct prediction/Total No. of Prediction )
```

```
Accuracy of Classifier is : (0.7 )
```

Part 2: Developing a large-scale ML classifier

We will now extend the earlier principles for the full dataset. Essentially the task is the same, we want to find the parameters that allow us to clearly separate groups for classification.

Task 5: Scale the Input Features for further processing using the StandardScaler function (1)

```
from sklearn.preprocessing import StandardScaler
def standarize(full_data):
    # the scaler object (model)
    scaler = StandardScaler()
    # fit and transform the data
    scaled_data = scaler.fit_transform(full_data)

full_data=np.array(train_data)
full_data=np.concatenate((full_data,np.array(test_data)))

######### standarize the full data ##################
full_data=standarize(full_data)
print("Scaled Data:")
print("------------")
print(full_data)
##################################################
```

Task 6: Obtain numerical labels for each class using the LabelEncoder function (1)

```
from sklearn.preprocessing import LabelEncoder
def labelEncode(df):
    #create instance of label encoder
    lab = LabelEncoder()
    #perform label encoding on 'team' column
    df['labels'] = lab.fit_transform(df['labels'])
############ Label Encode ###################
frames = [train_data, test_data]
full_data_with_label = pd.concat(frames)
labelEncode(full_data_with_label)
print("Encoded Labels Data:")
print("--------------------")
```

```
print(full_data_with_label)
#####################################################
```

```
                x        y    labels
0   3114896.0  10815.0  wannacry
1   3436940.0   9551.0  wannacry
2   1812649.0  15343.0  wannacry
3   3067845.0  10541.0  wannacry
4     51591.0  21367.0  wannacry
..        ...      ...       ...
85    69418.0   9673.0      razy
86     8060.0   1580.0      razy
87     4394.0    540.0      razy
88   183380.0   8477.0      razy
89    42945.0   9401.0      razy

[240 rows x 3 columns]
```

```
                x        y  labels
0   3114896.0  10815.0       2
1   3436940.0   9551.0       2
2   1812649.0  15343.0       2
3   3067845.0  10541.0       2
4     51591.0  21367.0       2
..        ...      ...     ...
85    69418.0   9673.0       0
86     8060.0   1580.0       0
87     4394.0    540.0       0
88   183380.0   8477.0       0
89    42945.0   9401.0       0

[240 rows x 3 columns]
```

(Advanced) Task 7: Prepare the dataset for ML testing, using the Train-Test-Split function of sklearn (2)

```python
from sklearn.model_selection import train_test_split
def split_data(full_data,labels):
    X_train, X_test, y_train, y_test =
train_test_split(full_data,labels,random_state=104,
                                   test_size=0.25,
                                   shuffle=True)
###### Split the data ############
X_train, X_test, y_train,
y_test=split_data(full_data_with_label,full_data_with_label['labels'])
print("Splitted Data:")
print("---------------")
print ("X_train: ")
print(X_train)
print ("y_train:")
print(y_train)
print("X_test: ")
print(X_test)
print ("y_test: ")
print(y_test)
###################
```

```
X_train:
              x        y  labels
1     3436940.0   9551.0       2
106     83885.0   1834.0       0
81      21787.0   3933.0       0
52     551089.0   3893.0       1
87       4394.0    540.0       0
..          ...      ...     ...
16    4168106.0   6402.0       2
142    524381.0   1671.0       0
67      27037.0   1335.0       0
43     162017.0   2672.0       1
69     142290.0   2843.0       1
```

```
y_train:
1      2
106    0
81     0
52     1
87     0
      ..
16     2
142    0
67     0
43     1
69     1
Name: labels, Length: 180,
```

```
x_test:
              x         y  labels
64     101076.0    4401.0       1
148    183376.0    8477.0       0
28    4454087.0    5911.0       2
56     550822.0    3765.0       1
110     26992.0    7180.0       0
80     157062.0    7883.0       0
49    3332423.0    9712.0       2
117    378794.0    1009.0       0
66     550997.0    3755.0       1
138     49515.0    3187.0       0
95     162045.0    2787.0       1
52     373008.0    5058.0       1
58     112838.0    2961.0       1
99     551014.0    3862.0       1
13      55485.0    8426.0       2
73     162127.0    2704.0       1
135    183376.0    8477.0       0
18     189265.0    7895.0       2
115     14284.0    1217.0       0
28    1602025.0   10858.0       2
12    2029047.0   14535.0       2
45      48068.0    6195.0       1
134    128890.0    3777.0       0
34    3060364.0   11336.0       2
97     162036.0    2716.0       1
17     668082.0   10655.0       2
5     3235262.0   11015.0       2
41    3571123.0    9886.0       2
```

```
y_test:
64      1
148     0
28      2
56      1
110     0
80      0
49      2
117     0
66      1
138     0
95      1
52      1
58      1
99      1
13      2
73      1
135     0
18      2
115     0
28      2
12      2
45      1
134     0
34      2
97      1
17      2
5       2
41      2
```

(Advanced) Task 8: Use a Multi-Layer Perceptron (MLP) classifier to train a machine learning model, and obtain the accuracy score against your test data. (4)

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

def mpl_classifier(x_train,y_train,x_test):
    m1 = MLPClassifier(hidden_layer_sizes=(12, 13, 14), activation='relu',
solver='adam', max_iter=2500)
    m1.fit(x_train, y_train.values.ravel())
    predicted_values = m1.predict(x_test)
    return predicted_values
```

```
  X_train, X_test, y_train,
y_test=split_data(full_data_with_label,full_data_with_label['labels'])

###### MPL Classifiier ############
predicted_values=mpl_classifier(X_train,y_train,X_test)
print("MPL Classifier Data:")
print("--------------------")
print("MLP Confusion Matrix:")
print(confusion_matrix(y_test,predicted_values))
print("MLP Classification Report:")
print(classification_report(y_test,predicted_values))
##################################
```

```
MLP Confusuion Matrix:
[[17  0  0]
 [24  0  0]
 [19  0  0]]
MLP Classification Report:
              precision    recall  f1-score   support

           0       0.28      1.00      0.44        17
           1       0.00      0.00      0.00        24
           2       0.00      0.00      0.00        19

    accuracy                           0.28        60
   macro avg       0.09      0.33      0.15        60
weighted avg       0.08      0.28      0.13        60
```

(Advanced) Task 9: Use a Random Forest (RF) classifier to train a machine learning model, and obtain the accuracy score against your test data. (4)

```
def random_forest(x_train,y_train,x_test):
    #Create a Gaussian Classifier
    clf=RandomForestClassifier(n_estimators=100)
    #Train the model using the training sets y_pred=clf.predict(X_test)
    clf.fit(x_train,y_train)
    y_pred=clf.predict(x_test)
    return y_pred

###### Random Forest Classifiier ############
rf_prediction=random_forest(X_train,y_train,X_test)
print("Random Forest Classifier:")
```

```
print("-------------------------")
print("RF Confusuion Matrix:")
print(confusion_matrix(y_test,rf_prediction))
print("RF Classification Report:")
print(classification_report(y_test,rf_prediction))
print("Accuracy:",metrics.accuracy_score(y_test, rf_prediction))
##################################
```

```
Random Forest Classifier:
-------------------------
RF Confusuion Matrix:
[[17  0  0]
 [ 0 24  0]
 [ 0  0 19]]
RF Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       1.00      1.00      1.00        24
           2       1.00      1.00      1.00        19

    accuracy                           1.00        60
   macro avg       1.00      1.00      1.00        60
weighted avg       1.00      1.00      1.00        60

Accuracy: 1.0
```

(Advanced) Task 10: Show how ML parameters can improve the models to achieve a high accuracy score of over 80% (3)

Marks wil be awarded for how your tuning improves accuracy beyond 80%.

```
def set_data(index1,index2):
    train_data = np.vstack([
features[mal1_index:mal1_index+mal_range][[index1,index2]].values,
features[mal2_index:mal2_index+mal_range][[index1,index2]].values,
features[mal3_index:mal3_index+mal_range][[index1,index2]].values ])
    train_data = pd.DataFrame(train_data)
    full_data=np.array(train_data)
    train_labels = np.vstack([ labels[mal1_index:mal1_index+mal_range].values,
labels[mal2_index:mal2_index+mal_range].values,
labels[mal3_index:mal3_index+mal_range].values ])
    train_labels = pd.DataFrame(train_labels)
```

```
    train_data['labels'] = train_labels
    train_data = train_data.rename(columns={0:'x', 1:'y'})

    test_data = np.vstack([ features[mal1_index +
mal_range:mal1_index+mal_range+mal_test_range][[index1,index2]].values,
features[mal2_index+mal_range:mal2_index+mal_range+mal_test_range][[index1,index2
]].values,
features[mal3_index+mal_range:mal3_index+mal_range+mal_test_range][[index1,index2
]].values ])
    test_data = pd.DataFrame(test_data)
    full_data=np.concatenate((full_data,np.array(test_data)))

    test_labels = np.vstack([
labels[mal1_index+mal_range:mal1_index+mal_range+mal_test_range].values,
labels[mal2_index+mal_range:mal2_index+mal_range+mal_test_range].values,
labels[mal3_index+mal_range:mal3_index+mal_range+mal_test_range].values ])
    test_labels = pd.DataFrame(test_labels)
    test_data['labels'] = test_labels
    test_data = test_data.rename(columns={0:'x', 1:'y'})

    return train_data,test_data,full_data

def get_best_accuracy():
    accuracy=0
    for i in range(255):
        if(accuracy < 0.80):
            j=i+1
            train_data,test_data,full_data=set_data(i,j)
            frames = [train_data, test_data]
            full_data_with_label = pd.concat(frames)
            labelEncode(full_data_with_label)
            X_train, X_test, y_train,
y_test=split_data(full_data_with_label,full_data_with_label['labels'])
            ###### MPL Classifiier ############
            predicted_values=mpl_classifier(X_train,y_train,X_test)
            accuracy=metrics.accuracy_score(y_test, predicted_values)
            ##################################
        else:
            print("Best Accuracy For ML is : ",accuracy)
            print("Used Feature Columns: "+str(i+1)+" and "+str(j+1))
            return
    print("Can't achieve 80% ")

############ Best Accuracy ############
print("Best Accuracy for Classifier:")
```

```
print("-----------------------------")
get_best_accuracy()
#######################################
```

```
Best Accuracy For ML is :  0.8333333333333334
Used Feature Columns: 47 and 47
```