v1

UFCFEL Security Data Analytics and Visualisation

Portfolio Assignment: Part 2

Academic year: 2023-24

Conduct a security investigation to examine and classify different malware characteristics

For this task, the company **UMBUCH** have entrusted your support as a security data analyst once again. They believe they have found malware on some of their workstations; however they do not know what malware it is. They also believe that some machines may be infected with different kinds of malware. They have asked you to identify what malware is running on each of the contaminated machines, so that they may understand more about how this incident has occurred on their corporate network.

Dataset: You will be issued a dataset for this task like so: GitHub URL: https://github.com/ac-ai-bar-ley/ac-ads-2023

This exercise carries a weight of 30% towards your overall portfolio submission

Submission Documents

For Part 2 of your portfolio, your complete output file should be saved as:

- **STUDENT_ID-PART2.ipynb**

This should then be included in a ZIP file along with your other two portfolio documents.

The deadline for your portfolio submission is **THURSDAY 11th JANUARY @ 14:00**.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

features = pd.read_csv('./part2/malware_data.csv', header=None)
features
```

Out[1]:

	0	1	2	3	4	5	6	7	8	...	246	247	248	249	250	251	252	2
0	2134812	2134812	1336612	798712	1315612	1315612	868112	798612	1497612	868612	1494612	1497612	1798612	1494612	1497612	1494612	1798612	868
1	2134812	2137212	2037612	2038612	1977612	1402212	1939612	1402212	2137212	1938612	1938612	1937212	1738612	1938612	1937212	1938612	1738612	1938
2	2134812	1398612	798612	1838612	798612	868612	4974612	3337612	798612	1798612	1798612	3798612	3038612	3038612	3038612	3038612	3038612	4798
3	1732612	4812612	4812612	3712612	3812612	3812612	3812612	2812612	2412612	1812612	1812612	1812612	2812612	2812612	2812612	2812612	2812612	3812
4	3337612	1338612	3138612	3038612	3038612	3038612	4974612	3337612	3338612	3338612	3338612	3338612	3338612	3338612	3338612	3338612	3338612	3338
...																		
2500	2134812	1338612	1338612	1338612	1338612	1338612	1338612	1338612	1338612	1338612	1338612	1338612	1338612	1338612	1338612	1338612	1338612	1338
2501	2134812	4812612	4812612	3712612	3812612	3812612	3812612	2812612	2412612	1812612	1812612	1812612	2812612	2812612	2812612	2812612	2812612	3812
2502	2134812	1398612	798612	1838612	798612	868612	4974612	3337612	798612	1798612	1798612	3798612	3038612	3038612	3038612	3038612	3038612	4798
2503	2134812	4812612	4812612	3712612	3812612	3812612	3812612	2812612	2412612	1812612	1812612	1812612	2812612	2812612	2812612	2812612	2812612	3812
2504	2134812	1398612	798612	1838612	798612	868612	4974612	3337612	798612	1798612	1798612	3798612	3038612	3038612	3038612	3038612	3038612	4798

2505 rows × 256 columns

In [2]:
```
labels = pd.read_csv('./part2/malware_labels.csv', header=None)
labels = labels.drop(0, axis=1)
labels.rename(columns={1: 'labels'})
labels.value_counts()
```

Out[2]:
```
labels
blackenindi    2000
dropper        2000
fakeav         2000
filetour       2000
floytoolbx     2000
freezer        2000
gandcrab       2000
high           2000
lazy           2000
startsurf      2000
upatre         2000
vaccine        2000
wannacry       2000
zbot           2000
Name: count, dtype: int64
```

In this dataset, we have collected data from 26500 systems (rows), where each system reports 256 parameters (columns).

We will first look at developing a small classifier "by hand", to help you understand how the classifier works, and then we will look at using the full dataset on more powerful classifiers to see whether we can classify the different malware variants on the set of workstations.

Part 1: Developing a small classifier "by hand"

Do not modify this cell

In [3]:
```
# DO NOT MODIFY THIS CELL
max1_index = 17600
max2_index = 21800
max3_index = 23000
max1_range = 50
max2_test_range = 30

train_data = np.vstack(( features[max1_index:max1_index+max1_range][0,1],features[max2_index:max2_index+max
train_data = pd.DataFrame(train_data)
train_labels = np.vstack(( labels[max1_index:max1_index+max1_range].values, labels[max2_index:max2_index+max
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[max1_index:max1_index+max1_range][0,1],features[max2_index:max2_index+max
test_data = pd.DataFrame(test_data)
test_labels = np.vstack(( labels[max1_index:max1_index+max1_range+max2_test_range].values, labels[max2_index
test_labels = pd.DataFrame(test_labels)
test_data['labels'] = test_labels
test_data = test_data.rename(columns={0:'x', 1:'y'})

train_data
```

Out[3]:

	x	y	labels
0	2134812.0	1581312.	wannacry
1	3438438.0	893.0	wannacry
2	1813938.0	1534612.	wannacry
3	3863634.0	1384.0	wannacry
4	1135812.0	2134812.	wannacry
...			
146	1565812.	1734.0	lazy
147	864634.0	5114.0	lazy
148	1863634.0	863.0	lazy
149	1863634.0	8477.0	lazy
150	1135812.	8812.0	lazy

150 rows × 3 columns

We now have a smaller sample of data to work with called train_data, that has three malware families included, and we have reduced the number of features to make it easier to understand the process of how the classifier operates.

Question 1: Find the Centroid point of each of the three groups

(5 marks)

Hint: We have filtered the sample to a new variable called **train_data**. *Also, think about how to find the mid-point of a set of co-ordinate points a and y.*

In [4]:
```
# ANSWER
wannacry_centroid = []
lazy_centroid = []
startsurf_centroid = []

######### ADD YOUR CODE HERE ##########

# group data by 'labels' column
groups = train_data.groupby('labels')

# get centroids for each group
centroids = groups.mean()

#add values of each centroid in an array
wannacry_centroid = np.array(centroids[centroids.index == 'wannacry'])
lazy_centroid = np.array(centroids[centroids.index == 'lazy'])
startsurf_centroid = np.array(centroids[centroids.index == 'startsurf'])

#print the results
print("Wannacry:", wannacry_centroid)
print("Lazy: ", lazy_centroid)
print("Startsurf: ", startsurf_centroid)

Wannacry: [12181048.66  13801.1 ]
Lazy:  [1565812.    1734.    ]
Startsurf: [1878278.12  3754.40]
```

Question 2: Create a Scatter Plot that shows both the train data and the corresponding centroids. Colour-code your data by group, and use a different visual marker for the centroid.

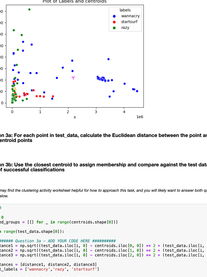
(5 marks)

In [5]:
```
# ANSWER
######### ADD YOUR CODE HERE ##########
# get x and y vectors
x = train_data['x']
y = np.array(train_data['y'])

# scatter plot for the data points with color-coded groups
sns.scatterplot(x=x, y=y, data=train_data, hue='labels', palette={'wannacry': 'blue', 'lazy': 'green','startsurf

plt.scatter(wannacry_centroid[0,0], wannacry_centroid[0,1], marker='1',s=100, color='magenta')
plt.scatter(lazy_centroid[0,0], lazy_centroid[0,1], marker='2',s=100, color='orange')
plt.scatter(startsurf_centroid[0,0], startsurf_centroid[0,1], marker='3',s=100, color='purple')

plt.title('Plot of Labels and centroids')
plt.show()
```



Question 3a: For each point in test_data, calculate the Euclidean distance between the point and each of the three centroid points

(5 marks)

Question 3b: Use the closest centroid to assign membership and compare against the test data label to obtain a score of successful classifications

(5 marks)

Hint: You may find the clustering activity worksheet helpful for how to approach this task, and you will likely want to answer both questions within the same for-loop below.

In [6]:
```
# ANSWER

score = 0
clusterdf_groups = [[] for _ in range(centroids.shape[0])]

for i in range(test_data.shape[0]):

    ######### Question 3a - ADD YOUR CODE HERE ##########
    distance0 = np.sqrt((test_data.iloc[i, 0] - centroids.iloc[0, 0])** 2 + (test_data.iloc[i, 1] - centroids.iloc
    distance1 = np.sqrt((test_data.iloc[i, 0] - centroids.iloc[1, 0])** 2 + (test_data.iloc[i, 1] - centroids.iloc
    distance2 = np.sqrt((test_data.iloc[i, 0] - centroids.iloc[2, 0])** 2 + (test_data.iloc[i, 1] - centroids.iloc

    distances = [distance0, distance1, distance2]
    dist_labels = ['wannacry','lazy', 'startsurf']

    # find the index of the minimum distance
    min_index = np.argmin(distances)

    # check if the predicted label matches the actual label
    if dist_labels[min_index] == test_data.iloc[i, 2]:
        score += 1

    ######### Question 3b - ADD YOUR CODE HERE ##########

    # get Euclidean distances for each centroid
    distances_b = [np.sqrt((test_data.iloc[i, 0] - centroids.iloc[j, 0])** 2 + (test_data.iloc[i, 1] - centroids.i

    # find the index of the minimum distance
    min_index_b = np.argmin(distances_b)

    # append data point to the corresponding clustered group (assign membership)
    clusterdf_groups[min_index_b].append(test_data.iloc[i, 2])

print ("Score:", score)

# plot the results
for idx, group_name in enumerate(clusterdf_groups):
    cluster_df = pd.DataFrame(group_data)
    group_name = dist_labels[idx]
    plt.scatter(group_data.iloc[:, 0], group_data.iloc[:, 1], label=f'{group_name} Group')

#add color legend
centroid_colors = ['blue', 'green', 'green']
for idx in range(centroids.shape[0]):
    centroid_name = dist_labels[idx]
    plt.scatter(centroids.iloc[idx, 0], centroids.iloc[idx, 1], color=centroid_colors[idx], marker='x', label=f'{cen

plt.xlabel('Feature X')
plt.ylabel('Feature Y')
plt.legend()
plt.title('Clustering Results')

plt.show()
```

```
Score: 11
```

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

(5 marks)

In [7]:
```
# ANSWER

######### ADD YOUR CODE HERE ##########

#get the accuracy score
accuracy = (score / test_data.shape[0]) * 100

print("Final accuracy: {accuracy:.2f}%")

final accuracy: 12.22%
```

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. You will want to consult the scikit-learn documentation to answer the following questions.

Question 5: Scale the Features DataFrame using the StandardScaler function

(5 marks)

Hint: The variable **features** *contains all of the numerical parameters that we will use to train our ML model, which we need to normalise using this function.*

In [8]:
```
# ANSWER
######### ADD YOUR CODE HERE ##########
from sklearn.preprocessing import StandardScaler

#create a StandardScaler instance
scaler = StandardScaler()

# fit the scaler to the data and transform the features dataframe
scaled_features = scaler.fit_transform(features)

# create a new dataframe with the scaled features
scaled_features_df = pd.DataFrame(scaled_features, columns=features.columns)

#print the scaled features dataframe
scaled_features_df
```

Out[8]:

	0	1	2	3	4	5	6	7	8	...	246	247	248	2
0	-0.267992	-0.267216	-0.608603	-0.300661	-0.609405	-0.609405	-0.765002	-0.801117	-1.000643	-0.832208	-0.836713	-0.836717	-0.836208	-0.83
1	-0.267992	-0.267197	-0.274349	-0.254503	-0.323019	-0.554459	-0.178094	-0.554459	-0.267197	-0.254503	-0.192924	-0.192108	-0.192594	-0.19
2	-0.267994	-0.270944	0.243108	0.270960	0.737969	0.402608	0.200412	0.204406	0.202703	1.367200	0.543903	0.550904	0.402966	0.479
3	-0.273671	-0.268006	-0.618019	-0.615310	-0.624007	-0.400185	-0.364600	-0.523376	-0.538809	-0.618373	-0.617219	-0.620202	-0.624174	-0.62
4	-0.270960	-0.267179	-0.608398	-0.300658	-0.609404	-0.609404	-0.537538	-0.537969	-0.538260	-0.400767	-0.836693	-0.837988	-0.254445	-0.3704
...														
2500	-0.267991	-0.268006	-0.533244	-0.191181	-0.300316	-0.310013	-0.535062	-0.310013	-0.535062	-0.310013	-0.535062	-0.309470	-0.310035	-0.3098
2501	-0.270960	-0.268006	-0.618019	-0.615310	-0.624007	-0.400185	-0.364600	-0.523376	-0.538809	-0.618373	-0.617219	-0.620202	-0.624174	-0.62
2502	-0.267991	-0.270951	-0.608376	-0.160026	-0.300316	-0.310013	-0.535062	-0.310013	-0.535062	-0.417708	-0.836457	-0.837100	-0.310035	-0.3098
2503	-0.270960	-0.268006	-0.618019	-0.615310	-0.624007	-0.400185	-0.364600	-0.523376	-0.538809	-0.618373	-0.617219	-0.620202	-0.624174	-0.62
2504	-0.267991	-0.270951	-0.608376	-0.160026	-0.300316	-0.310013	-0.535062	-0.310013	-0.535062	-0.417708	-0.836457	-0.837100	-0.310035	-0.3098

2500 rows × 256 columns

Question 6: Use the LabelEncoder function on the Labels variable to obtain numerical label data

(5 marks)

In [9]:
```
# ANSWER
######### ADD YOUR CODE HERE ##########
from sklearn.preprocessing import LabelEncoder

# get a LabelEncoder
label_encoder = LabelEncoder()

#transform the labels

num_labels = label_encoder.fit_transform(labels['label'])

num_labels
```

Out[9]:
```
array([ 7,  7,  7, ..., 13, 13, 13])
```

Question 7: Prepare the dataset for ML testing using the Train-Test-Split function

(5 marks)

In [10]:
```
# ANSWER
######### ADD YOUR CODE HERE ##########
from sklearn.model_selection import train_test_split

# split the dataset into training and testing
x_train, x_test, y_train, y_test = train_test_split(scaled_features_df, num_labels, test_size=0.3, random_state=1
```

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

(5 marks)

In [11]:
```
# ANSWER
######### ADD YOUR CODE HERE ##########
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score


# create an MLP Classifier
mlp_classifier = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500, random_state=1)

# fit the classifier to the training data
mlp_classifier.fit(x_train, y_train)

#predict on the test data
y_pred = mlp_classifier.predict(x_test)

#get accuracy score
accuracy = accuracy_score(y_test, y_pred)*100

# print the accuracy score
print("MLP Accuracy: {accuracy:.2f}%")

MLP Accuracy: 83.18%
```

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

(5 marks)

In [12]:
```
# ANSWER

######### ADD YOUR CODE HERE ##########
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


# create Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# fit the classifier to the training data
rf_classifier.fit(x_train, y_train)

#predict on the test data
y_pred = rf_classifier.predict(x_test)

# get accuracy score
accuracy = accuracy_score(y_test, y_pred)*100

print("RF Accuracy: {accuracy:.2f}%")

RF Accuracy: 81.13%
```

Question 10: Describe what you believe are the key findings of this task, comparing the by-hand classifier with the MLP classifier. As part of your discussion, consider the volume of data in both approaches and the issues of data separability. Give justifications for your answers. Limit your response to a maximum of 300 words.

(10 marks).

Notable insights are revealed when comparing the MLP classifier and the by-hand classifier. The MLP classifier demonstrated a noteworthy enhancement in accuracy (83.18%) when juxtaposed with the by-hand classifier (12.22%). This improvement highlights the MLP model's ability to manage higher data volumes effectively, which is important as datasets become more complex. Effective data separation was a challenge for the by-hand classifier, especially considering the complex and non-linear nature of malware characteristics. On the other hand, because it is a neural network, the MLP classifier was better at handling separability problems because it was better at identifying intricate patterns and dependencies in the data. Because the MLP model learns complex patterns and adapts to various data distributions, it performs more robustly with larger datasets and more complex classification tasks compared to conventional methods. This underscores why the by-hand method might find it difficult to scale to handle malware cases, particularly given how malware types and their levels of complexity change over time. All things considered, these results demonstrate the benefits of using machine learning models—MLP in particular—in the field of security data analytics. In the context of large-scale datasets, these models not only exhibit improved generalization capabilities and better scalability and adaptability to complex patterns, but they also improve malware classification accuracy and efficiency.

In [13]: