Riga Technical University

Telecommunications Software (RAE411).

Sixth Practical Exercise.

9th of May 2023

Zeyad Mohamed Nashaat Abdelghany.

230AMB013.

Spring 2023.

I. Introduction:

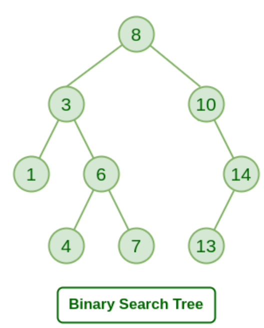
This report is the documentation for the sixth practical exercise. This exercise is divided into two sections: Binary search tree and SDN traffic classification with DT. Python is used to implement both sections using Anaconda.

II. Binary Search Tree:

Binary search tree is a node-based sorted data structure at which:

1. The left subtree of a node contains only nodes with lower keys' values than the parents' values.
2. The right subtree of a node contains only nodes with higher keys' values than the parents' values.
3. Each node has only a maximum value of two children.
4. There must be no duplicates for values in a tree.

Binary search trees are implemented to be used in the sorting algorithms. It is the basic data structure used in Microsoft Excel and Spreadsheets. Binary search trees have three main functions which are: searching, insertion, and deletion. There are also three ways to traverse through a binary tree which are: in-order, pre-order, and post-order traversing.



A. Node creation:

Nodes are the main building unit for a binary tree. A node has three properties: its value, its right child, and its left child. A class in python is done by the name of Node to create a node whenever it is needed in a tree. This class has a constructor which holds the data for the node, its left and right children.

```
class Node:
    def __init__(self, data):    #constructor to initialize the class whenever an object is created from it
        self.data=data           #self is the object created, data is the value for the node.
        self.lchild=None         #self is automatically passed to the function without writing it in the function init
        self.rchild=None
```

B. Tree creation:

Binary tree is mainly multiple nodes connected to each other. A class is created in python to create a tree from a given list. The constructor of this class mainly takes the first element in the tree as the root node and insert the other elements in the with the insertion function following the rules of the binary tree data structure.

```
class BinarySearchTree:
    def __init__(self, node_list):
        self.root=Node(node_list[0])  #starting with the first value in the list as the root
        for data in node_list[1:]:    #insertion of all of the other elements except the root which is already inserted
            self.insert(data)
```

C. Search function:

This function enables the user to insert a value for a node in the tree and return whether this value is found or not. It has 3 inputs: the tree root, the parent, and the value needed to be found. It is a recursive function that check whether the value is found in the tree or not.

```
    def search(self, node, parent, data):
        if node is None:                                    #if node, not found
            return False, node, parent
        if node.data == data:                               #if value of root is equal to data, the search value is the root
            return True, node, parent
        if node.data> data:                                 #if value of root is greater than data, child must be on left
            return self.search(node.lchild, node, data)
        else:                                               #if value of root is smaller than data, child must be on right
            return self.search(node.rchild, node, data)
```

D. Insertion function:

This function starts by making sure that the value inserted is not already in the tree to avoid duplicates. It is a recursive function that operates on the idea of creating a node to the value inserted. Then, the value of this node is then compared to its root. If the value is bigger than the root's value, then the function recurs in the right direction. If it is smaller, then it recurs in the left direction.

```
def insert(self, data):
    flag, n, p = self.search(self.root, self.root, data)    #making sure value is in the tree
    if not flag:                                            #if node not found, create a new node with value given
        new_node = Node(data)
        if data>p.data:                                     #if node value bigger than parent
            p.rchild = new_node                             #put it on right
        else:
            p.lchild = new_node                             #if smaller, put child on left of parent
```

E. Deletion function:

Deletion function has several scenarios and it depends whether the deleted node is a parent for one child or for two children. It makes sure at first that the needed node is present in the tree.

```
def delete(self, root, data):
    flag, n, p = self.search(root, root, data)
    if flag is False:
        print("No key value found")
    else:
        if n.lchild is None:                               #parent for one child only
            if n==p.lchild:
                p.lchild=n.rchild
            else:
                p.rchild=n.rchild
            del p
        elif n.rchild is None:
            if n==p.lchild:
                p.lchild=n.lchild
            else:
                p.rchild=n.lchild
            del p
        else:                                              #parent of two children
            pre=n.rchild
            if pre.lchild is None:
                n.data=pre.data
                n.rchild=pre.rchild
                del pre
            else:
                next=pre.lchild
                while next.lchild is not None:
                    pre=next
                    next=next.lchild
                n.data=next.data
                pre.lchild=next.rchild
                del n
```

F. Traversing functions:

Traversing functions are used with the search trees to have an algorithm on how we extract the whole nodes in a tree. It is the process of visiting each node in the tree exactly once.

a. Preorder traversing:

This algorithm works by printing: the root, then printing the whole left subtree until reaching to the leaves, then traversing in the right subtree.

```
def preorder(self, node):
    if node is not None:
        print(node.data),
        self.preorder(node.lchild)
        self.preorder(node.rchild)
```

b. Inorder traversing:

This algorithm works by printing: the left node first, then the parent, then the right node.

```
def inorder(self, node):
    if node is not None:
        self.inorder(node.lchild)
        print(node.data),
        self.inorder(node.rchild)
```

c. Postorder traversing:

This algorithm works by printing: the left node, then the right node, then the parent.

```
def postorder(self, node):
     if node is not None:
        self.preorder(node.lchild)
        self.preorder(node.rchild)
        print(node.data),
```

G. Implementation on given lists:

In this report, implementation of all of the three traversal method, deletion of element and searching for them is applied for each list.

a. List a:

List a is given as: [49, 38, 65, 97, 60, 76, 13, 27, 5, 1]

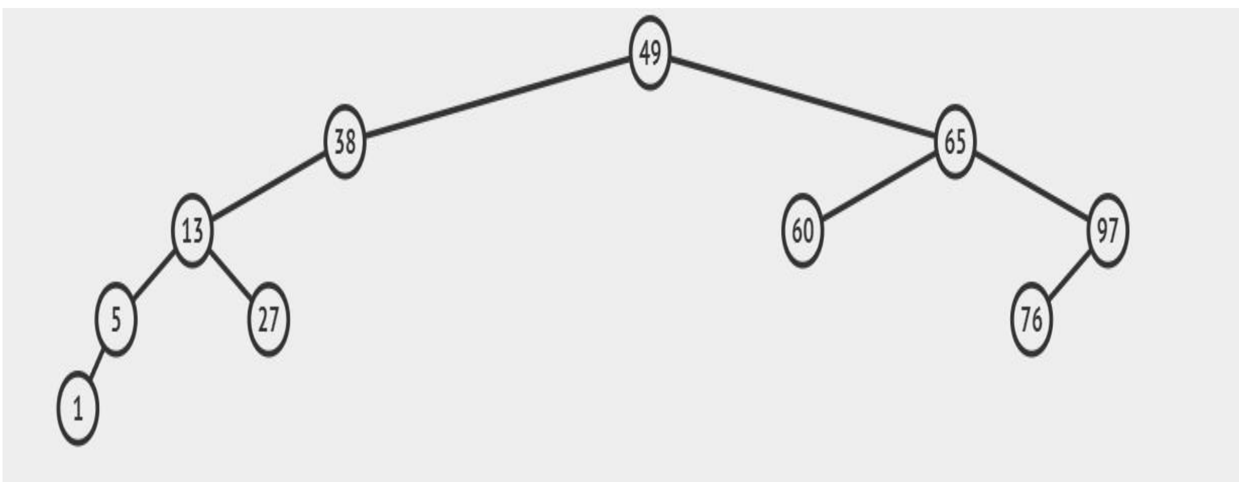


Figure 1 Original Tree for List a

```
a = [49, 38, 65, 97, 60, 100, 13, 27, 5, 1]
tree=BinarySearchTree(a)
print('preorder:')
tree.preorder(tree.root)
print()
print(tree.search(tree.root, tree.root, 13))
print('\ninorder:')
tree.inorder(tree.root)
print('\npostorder:')
tree.postorder(tree.root)
tree.delete(tree.root,65)
print('\npreorder after element deletion:')
tree.preorder(tree.root)
print()
print(tree.search(tree.root, tree.root, 65))
```

Figure 2 Code for operations on List a

```
preorder:
49
38
13
5
1
27
65
60
97
100

(True, <__main__.Node object at 0x000001E4D03FFFA0>, <__main__.Node object at 0x000001E4D03FF6A0>)

inorder:
1
5
13
27
38
49
60
65
97
100

postorder:
38
13
5
1
27
65
60
97
100
49

preorder after element deletion:
49
38
13
5
1
27
97
60
100

(False, None, <__main__.Node object at 0x000001E4D03FF1C0>)
```

Figure 3 Output for list a

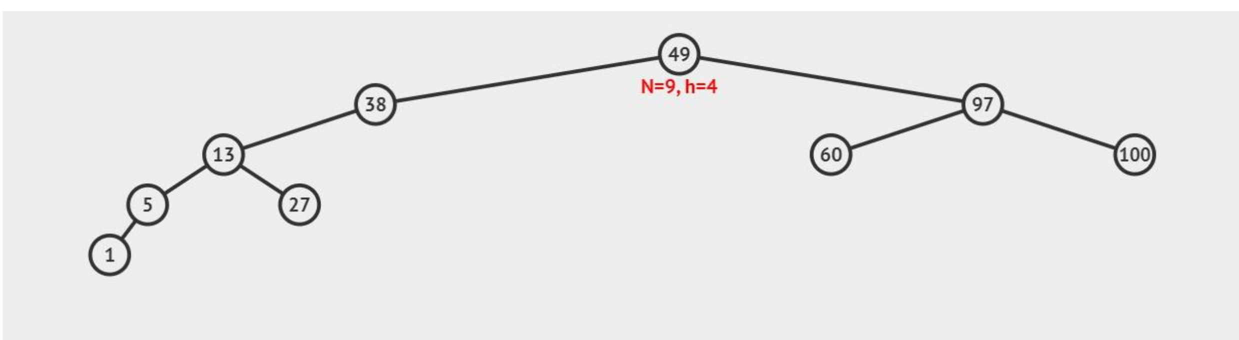


Figure 4 Tree for list a after element deletion

b. List b:

According to the given list: [149, 38, 65, 197, 60, 176, 13, 217, 5, 11].

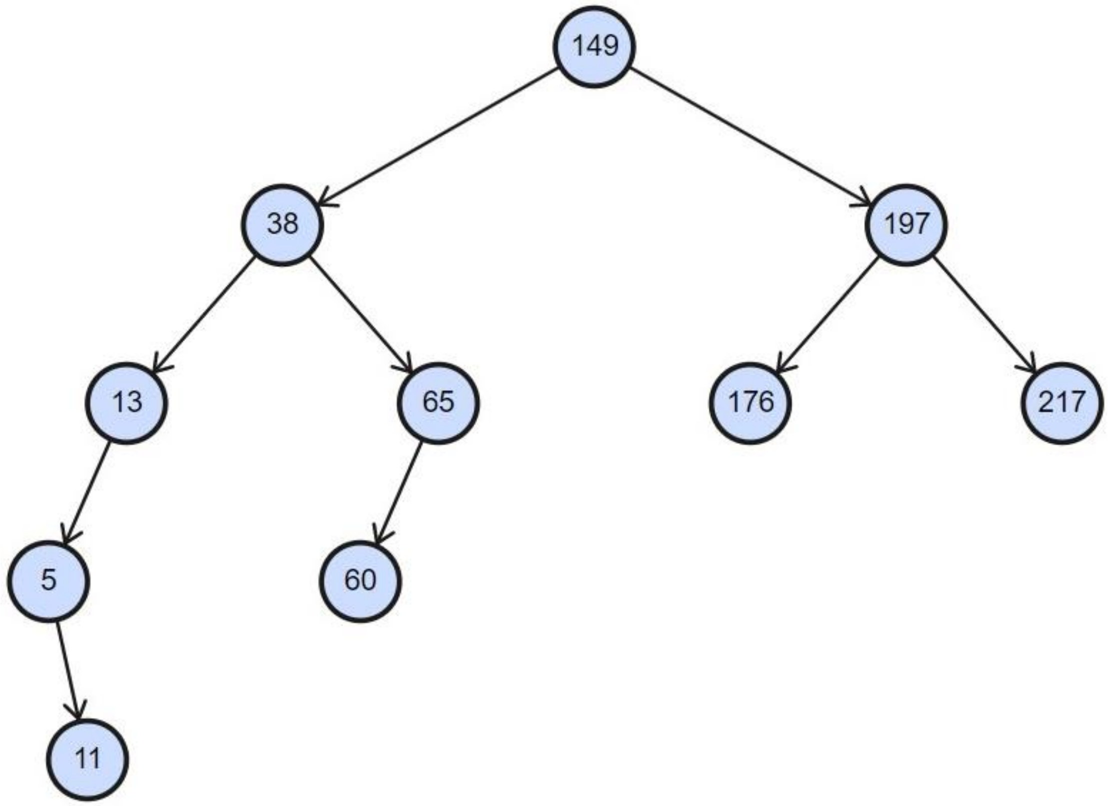


Figure 5 Original Tree for List b

```
b = [149, 38, 65, 197, 60, 176, 13, 217, 5, 11]
tree=BinarySearchTree(b)
print('preorder:')
tree.preorder(tree.root)
print()
print(tree.search(tree.root, tree.root, 217))
print('\ninorder:')
tree.inorder(tree.root)
print('\npostorder:')
tree.postorder(tree.root)
tree.delete(tree.root,197)
print('\npreorder after element deletion:')
tree.preorder(tree.root)
print()
print(tree.search(tree.root, tree.root, 197))
```

Figure 6 Code for operations on List b

```
preorder:
149
38
13
5
11
65
60
197
176
217

(True, <__main__.Node object at 0x000001E4D03F2220>, <__main__.Node object at 0x000001E4D03F21F0>)

inorder:
5
11
13
38
60
65
149
176
197
217

postorder:
38
13
5
11
65
60
197
176
217
149

preorder after element deletion:
149
38
13
5
11
65
60
217
176

(False, None, <__main__.Node object at 0x000001E4D03F2820>)
```

Figure 7 Output for list b

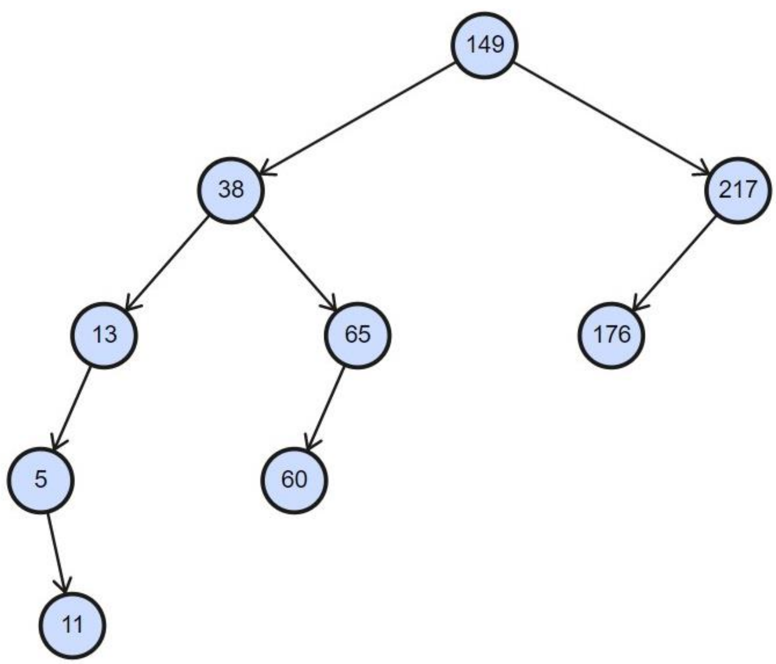


Figure 8 Tree for list b after element deletion

c. **List c:**

According to the given list: [49, 38, 65, 97, 64, 76, 13, 77, 5, 1, 55, 50].

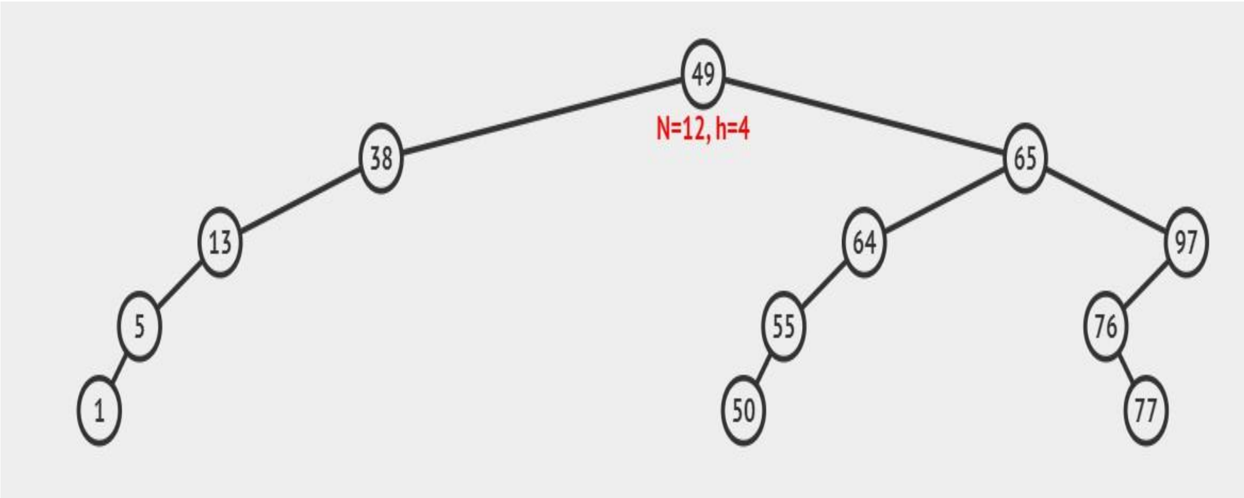


Figure 9 Original Tree for List c

```
c = [49, 38, 65, 97, 64, 76, 13, 77, 5, 1, 55, 50]
tree=BinarySearchTree(c)
print('preorder:')
tree.preorder(tree.root)
print()
print(tree.search(tree.root, tree.root, 77))
print('\ninorder:')
tree.inorder(tree.root)
print('\npostorder:')
tree.postorder(tree.root)
tree.delete(tree.root,38)
print('\npreorder after element deletion:')
tree.preorder(tree.root)
print()
print(tree.search(tree.root, tree.root, 38))
```

Figure 10 Code for operations on List c

```
preorder:
49
38
13
5
1
65
64
55
50
97
76
77

(True, <__main__.Node object at 0x000001E4D03FF970>, <__main__.Node object at 0x000001E4D03FF0D0>)

inorder:
1
5
13
38
49
50
55
64
65
76
77
97
postorder:
38
13
5
1
65
64
55
50
97
76
77
49

preorder after element deletion:
49
13
5
1
65
64
55
50
97
76
77

(False, None, <__main__.Node object at 0x000001E4D03FF730>)
```

Figure 11 Output for list c

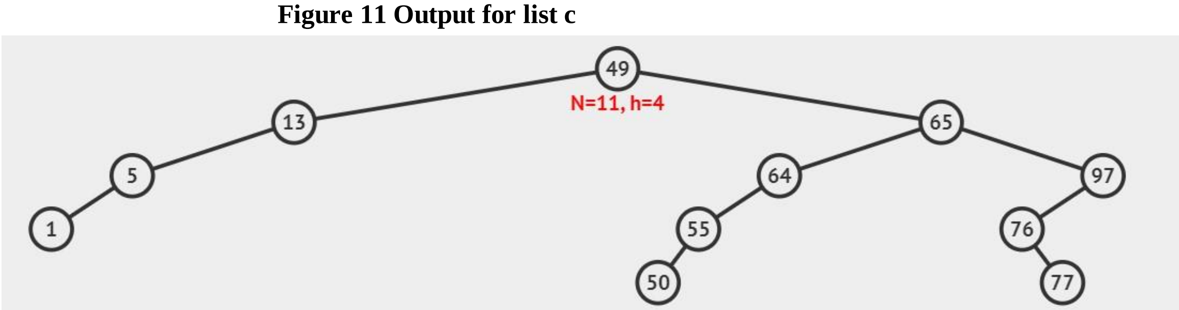


Figure 12 Tree for list c after element deletion

III. SDN classification with decision tree:

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

A. Program Code:

This code is used to perform some data analysis and classification tasks using a decision tree classifier on a dataset stored in a CSV file.

```python
import pandas as pd
import numpy as np
import scipy.stats as stats
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, recall_score, precision_score, f1_score, classification_report
from sklearn.model_selection import cross_val_score, KFold
import matplotlib.pyplot as plt
import seaborn as sn


# importing csv file from location
dataset = pd.read_csv("C:/Users/Zeyad Mohamed/Downloads/assignments-main (1)/assignments-main/SDN_traffic.csv")

print(dataset.head())
print(dataset.info())
print(dataset.describe())
print(dataset.duplicated())

#dataset needed for the analysis in CSV file
x = dataset[['forward_bps_var',
            "tp.sre", "tp.dst", "nw_proto",
            "forward pe", "forward_bc", "forward_pl",
            "forward piat", "forward_pps", "forward_bps", "forward_pl_mean",
            "forward piat mean", "forward_pps_mean", "forward_bps_mean", "forward_pl_var", "forward_piat_var",
            "forward.pps var", "forward_pl_q1", "forward_pl_q3",
            "forward piat.g1", "forward_piat_q3", "forward_pl_max", "forward_pl_min",
            "forward plat.max", "forward.piat_win", "forward_pps_max", "forward_pps_min",
            "forward bps_max", "forward_bps_min", "forward_duration", "forward_size_packets",
            "forward size bytes", "reverse_pc", "reverse_bc", "reverse.pl", "reverse piat", "reverse PRS",
            "reverse_bps", "reverse_pl_mean", "reverse_piat.mean", "reverse_pps_nean", "reverse bas mean", "reverse_pl_var",
            "reverse plin", "reverse_pl", "reverse_piat", "reverse_piat_var", "reverse_pps_var", "reverse_bps_var",
            "reverse_piat q1", "reverse_pl_q3", "reverse_piat_max", "reverse_piat_min", "reverse_pps_max", "reverse_pps_min",
            "reverse_piat_q3", "reverse_pl_max", "reverse_bps_max", "reverse_bps_min", "reverse_duration",
             "reverse size packets", "reverse size bytes"]]
```

```python
x.loc[1877, 'forward_bps_var'] = float(11968865203349)
x.loc[9131, 'forward_bps_var'] = float(12880593884833)
x.loc[2381, 'forward_bps_var'] = float(39987497172945)
x.loc[2562, 'forward_bps_var'] = float(663388742992)
x.loc[1931, 'forward_bps_var'] = float(37770223877794)
x.loc[2078, 'forward_bps_var'] = float(9822747730895)
x.loc[2567, 'forward_bps_var'] = float(37778223877794)
x.loc[2586, 'forward_bps_var'] = float(97227875883751)
x.loc[2754, 'forward_bps_var'] = float(18789751483737)
x.loc[2765, 'forward_bps_var'] = float(33969277035759)
x.loc[2984, 'forward_bps_var'] = float(39284786962856)
x.loc[3844, 'forward_bps_var'] = float(9169996863653)
x.loc[3349, 'forward_bps_var'] = float(37123283690575)
x.loc[3507, 'forward_bps_var'] = float(61019864598464)
x.loc[3610, 'forward_bps_var'] = float(46849628984872)
x.loc[3717, 'forward_bps_var'] = float(97158873841506)
x.loc[3845, 'forward_bps_var'] = float(11968865203349)
x.loc[3868, 'forward_bps_var'] = float(85874278395372)

X = pd.DataFrame(x)
X["forward_bps_var"] = pd.to_numeric(X["forward_bps_var"])
print(X.info())

Y = dataset[["category"]]
Y = Y.to_numpy()
Y = Y.ravel()
Labels, uniques = pd.factorize(Y)
Y = Labels
Y = Y.ravel()

X = stats.zscore(X)
X = np.nan_to_num(X)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=0, test_size=0.3)
```

```
clf = DecisionTreeClassifier(random_state=0, max_depth=2)
clf.fit(X_train, Y_train)

cv = KFold(n_splits=10, random_state=0, shuffle=True)
accuracy = clf.score(X_test, Y_test)
KFold10_accuracy = cross_val_score(clf, X_train, Y_train, scoring='accuracy', cv=cv, n_jobs=-1)
print(KFold10_accuracy.mean())

predict = clf.predict(X_test)
cm = confusion_matrix(Y_test, predict)
precision = precision_score(Y_test, predict, average='weighted', labels=np.unique(predict))
recall = recall_score(Y_test, predict, average='weighted', labels=np.unique(predict))
fiscoreMacro = f1_score(Y_test, predict, average='macro', labels=np.unique(predict))
print(classification_report(Y_test, predict, target_names=uniques))

importance = clf.feature_importances_
important_features_dict = {}
for idx, val in enumerate(importance):
    important_features_dict[idx] = val
important_features_list = sorted(important_features_dict,
                                 key=important_features_dict.get,
                                 reverse=True)
print(f'10 most important features: {important_features_list[:10]}')
```

```
fn = ['forward_bps_var',
      "tp.src", "tp.dst", "nw_proto",
      "forward_pe", "forward_bc", "forward_pl",
      "forward_piat", "forward_pps", "forward_bps", "forward_pl_mean",
      "forward_piat_mean", "forward_pps_mean", "forward_bps_mean", "forward_pl_var", "forward_piat_var",
      "forward_pps_var", "forward_pl_q1", "forward_pl_q3",
      "forward_piat_q1", "forward_piat_q3", "forward_pl_max", "forward_pl_min",
      "forward_piat_max", "forward_piat_win", "forward_pps_max", "forward_pps_min",
      "forward_bps_max", "forward_bps_min", "forward_duration", "forward_size_packets",
      "forward_size_bytes", "reverse_pc", "reverse_bc", "reverse_pl", "reverse_piat", "reverse_PRS",
      "reverse_bps", "reverse_pl_mean", "reverse_piat_mean", "reverse_pps_mean", "reverse_bps_mean", "reverse_pl_var",
      "reverse_plin", "reverse_pl", "reverse_piat", "reverse_piat_var", "reverse_pps_var", "reverse_bps_var",
      "reverse_piat_q1", "reverse_pl_q3", "reverse_piat_max", "reverse_piat_min", "reverse_pps_max", "reverse_pps_min",
      "reverse_piat_q3", "reverse_pl_max", "reverse_bps_max", "reverse_bps_min", "reverse_duration", "reverse_size_packets",
      "reverse_size_bytes"]

la = ['WWW', 'DNS', 'FTP', 'ICMP', 'P2P', 'VOIP']

plt.figure(1, dpi=300)
fig = tree.plot_tree(clf, filled=True, feature_names=fn, class_names=la)
plt.title("Decision tree trained on all the features")
plt.show()

import seaborn as sn
import matplotlib.pyplot as plt
```

```
labels = uniques
plt.figure(2, figsize=(5, 2))
plt.title("Confusion Matrix", fontsize=10)
cm_new = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
sn.heatmap(cm_new, annot=True, cmap="YlGnBu", fmt=".2f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.show()
```

B. Code Output:

```
                            id_flow        nw_src  tp_src         nw_dst  \
0  b2bb77a570fcfa9325eb9e51b6116d2a  172.16.25.104   41402  34.107.221.82
1  f07977b0d1d6645c4fe1e9efea080ff3  172.16.25.104   41406  34.107.221.82
2  e4026ba9b6c1957516e92bdd0d04878f  172.16.25.104   38232    52.84.77.43
3  e2d747932e41500b1463fe8ae4299ecb  172.16.25.104   38234    52.84.77.43
4  56325703391225ad65e013e7a2b02fac  172.16.25.104   60166    52.32.34.32

   tp_dst  nw_proto  forward_pc  forward_bc  forward_pl  forward_piat  ...  \
0      80         6           5         300       60.00           6.0  ...
1      80         6           5         300       60.00           6.0  ...
2     443         6           3         198       66.00          10.0  ...
3     443         6           3         198       66.00          10.0  ...
4     443         6           4         265       66.25           7.5  ...

   reverse_piat_max  reverse_piat_min  reverse_pps_max  reverse_pps_min  \
0         10.333333              6.00         0.166667         0.096774
1         10.000000              6.20         0.161290         0.100000
2         10.333333             10.00         0.100000         0.096774
3         10.333333             10.00         0.100000         0.096774
4          7.750000              7.75         0.129032         0.129032

   reverse_bps_max  reverse_bps_min  reverse_duration  reverse_size_packets  \
0        15.133333         5.806452               121                    15
1        15.133333         6.000000               121                    15
2         6.000000         5.806452                91                     9
3         6.000000         5.806452                91                     9
4         8.548387         8.548387                31                     4

   reverse_size_bytes category
0                1114      WWW
1                1114      WWW
2                 540      WWW
3                 540      WWW
4                 265      WWW
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id_flow               4234 non-null   object
 1   nw_src                4234 non-null   object
 2   tp_src                4234 non-null   int64
 3   nw_dst                4234 non-null   object
 4   tp_dst                4234 non-null   int64
 5   nw_proto              4234 non-null   int64
 6   forward_pc            4234 non-null   int64
 7   forward_bc            4234 non-null   int64
 8   forward_pl            4234 non-null   float64
 9   forward_piat          4234 non-null   float64
 10  forward_pps           4234 non-null   float64
 11  forward_bps           4234 non-null   float64
 12  forward_pl_mean       4234 non-null   float64
 13  forward_piat_mean     4234 non-null   float64
 14  forward_pps_mean      4234 non-null   float64
 15  forward_bps_mean      4234 non-null   float64
 16  forward_pl_var        4234 non-null   float64
 17  forward_piat_var      4234 non-null   float64
 18  forward_pps_var       4234 non-null   float64
 19  forward_bps_var       4234 non-null   object
 20  forward_pl_q1         4234 non-null   float64
 21  forward_pl_q3         4234 non-null   float64
 22  forward_piat_q1       4234 non-null   float64
 23  forward_piat_q3       4234 non-null   float64
 24  forward_pl_max        4234 non-null   float64
 25  forward_pl_min        4234 non-null   float64
 26  forward_piat_max      4234 non-null   float64
 27  forward_piat_min      4234 non-null   float64
 28  forward_pps_max       4234 non-null   float64
 29  forward_pps_min       4234 non-null   float64
 30  forward_bps_max       4234 non-null   float64
 31  forward_bps_min       4234 non-null   float64
 32  forward_duration      4234 non-null   int64
 33  forward size packets  4234 non-null   int64
```

```
 34  forward_size_bytes    4234 non-null   int64
 35  reverse_pc            4234 non-null   int64
 36  reverse_bc            4234 non-null   float64
 37  reverse_pl            4234 non-null   float64
 38  reverse_piat          4234 non-null   float64
 39  reverse_pps           4234 non-null   float64
 40  reverse_bps           4234 non-null   float64
 41  reverse_pl_mean       4234 non-null   float64
 42  reverse_piat_mean     4234 non-null   float64
 43  reverse_pps_mean      4234 non-null   float64
 44  reverse_bps_mean      4234 non-null   float64
 45  reverse_pl_var        4234 non-null   float64
 46  reverse_piat_var      4234 non-null   float64
 47  reverse_pps_var       4234 non-null   float64
 48  reverse_bps_var       4234 non-null   float64
 49  reverse_pl_q1         4234 non-null   float64
 50  reverse_pl_q3         4234 non-null   float64
 51  reverse_piat_q1       4234 non-null   float64
 52  reverse_piat_q3       4234 non-null   float64
 53  reverse_pl_max        4234 non-null   float64
 54  reverse_pl_min        4234 non-null   float64
 55  reverse_piat_max      4234 non-null   float64
 56  reverse_piat_min      4234 non-null   float64
 57  reverse_pps_max       4234 non-null   float64
 58  reverse_pps_min       4234 non-null   float64
 59  reverse_bps_max       4234 non-null   float64
 60  reverse_bps_min       4234 non-null   float64
 61  reverse_duration      4234 non-null   int64
 62  reverse_size_packets  4234 non-null   int64
 63  reverse_size_bytes    4234 non-null   int64
 64  category              4234 non-null   object
dtypes: float64(48), int64(12), object(5)
memory usage: 2.1+ MB
None
```

```
             tp_src        tp_dst     nw_proto     forward_pc    forward_bc  \
count   4234.000000   4234.000000  4234.000000    4234.000000  4.234000e+03
mean   39994.956542   8540.046528     6.660132    3835.848370  7.356521e+06
std    17331.881734  17575.486397     3.815368   18375.794566  3.585172e+07
min        0.000000      0.000000     1.000000       0.000000  0.000000e+00
25%    35248.500000     80.000000     6.000000       2.000000  1.200000e+02
50%    44009.000000    443.000000     6.000000       3.000000  1.980000e+02
75%    52130.250000    443.000000     6.000000       6.000000  3.850000e+02
max    65534.000000  60949.000000    17.000000  181104.000000  3.558093e+08

          forward_pl  forward_piat   forward_pps   forward_bps  \
count    4234.000000   4234.000000  4.234000e+03  4.234000e+03
mean      316.336560     15.261581  4.788105e+02  2.576202e+06
std      3732.045349    182.065520  2.021312e+04  1.200390e+08
min         0.000000      0.000000  0.000000e+00  0.000000e+00
25%        60.000000      0.048051  6.451613e-02  4.000000e+00
50%        66.000000      3.500000  1.666667e-01  1.260000e+01
75%        79.811688      7.500000  5.161290e-01  4.600000e+01
max    154375.000000   4125.000000  1.303625e+06  7.422774e+09
```

```
       forward_pl_mean  ...  reverse_pl_min  reverse_piat_max  \
count      4234.000000  ...     4234.000000       4234.000000
mean       1582.814224  ...       54.418871         23.652912
std        9644.341190  ...      269.495303        229.416470
min           0.000000  ...        0.000000          0.000000
25%          43.000000  ...        0.000938          0.000433
50%          61.250000  ...       15.500000          7.500000
75%          98.000000  ...       60.000000         15.500000
max      162975.000000  ...     5573.208202       4125.000000

       reverse_piat_min  reverse_pps_max  reverse_pps_min  reverse_bps_max  \
count      4.234000e+03     4.234000e+03     4.234000e+03     4.234000e+03
mean       5.189081e+02     1.263424e+03     6.683260e+04     1.270755e+05
std        2.792340e+04     4.689801e+04     2.674774e+06     4.139731e+06
min        0.000000e+00     0.000000e+00     0.000000e+00     0.000000e+00
25%        3.225807e-02     3.030303e-02     3.125000e-02     1.935484e+00
50%        6.559140e-01     9.677419e-02     1.000000e-01     6.026316e+00
75%        8.500000e+00     2.903226e-01     2.325000e+01     4.167742e+01
max        1.816375e+06     2.316875e+06     1.556534e+08     1.707531e+08
```

```
       reverse_bps_min  reverse_duration  reverse_size_packets  \
count     4.234000e+03       4234.000000          4.234000e+03
mean      6.747949e+04       3224.000000          2.750047e+05
std       3.034402e+06      20429.627234          1.519335e+06
min       0.000000e+00          0.000000          0.000000e+00
25%       2.242424e+00          5.000000          2.000000e+00
50%       9.258333e+00         30.000000          1.800000e+01
75%       4.900000e+01         60.000000          6.860000e+02
max       1.488506e+08     232137.000000          1.717689e+07

       reverse_size_bytes
count        4.234000e+03
mean         2.592156e+05
std          2.875554e+06
min          0.000000e+00
25%          0.000000e+00
50%          0.000000e+00
75%          2.460000e+02
max          1.214242e+08

[8 rows x 60 columns]
0       False
1       False
2       False
3       False
4       False
        ...
4229    False
4230    False
4231    False
4232    False
4233    False
Length: 4234, dtype: bool
```

IV. GitHub Link for the whole project:

For the link:

https://github.com/ZeyadNashaat/Telecommunications-Software-RAE411-/tree/main/Sixth%20practical%20exercise