RAE411

Telecommunications Software

Name:	Farjana Rahman
Student ID:	221AEM003
Session:	Spring 2023
Practical Exercise No:	6
Practical Exercise Title:	**Binary Search Tree and SDN Traffic classification with DT**
Course Instructor:	Tianhua Chen

In this experiment, we have learned to implement binary search tree in python including search, insert, delete and traversal functions. We have also experimented SDN traffic classification with DT.

Task-1:

There are three lists to create the tree and combine the functions given below. They are-

a = [49, 38, 65, 97, 60, 76, 13, 27, 5, 1]

b = [149, 38, 65, 197, 60, 176, 13, 217, 5, 11]

c = [49, 38, 65, 97, 64, 76, 13, 77, 5, 1, 55, 50]

The codes for the first list is given below-

In [78]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None


class BST:
    def __init__(self):
        self.root = None

    def insert(self, data):
        if self.root is None:
            self.root = Node(data)
        else:
            self._insert_recursive(self.root, data)

    def _insert_recursive(self, node, data):
        if data < node.data:
            if node.left is None:
                node.left = Node(data)
            else:
                self._insert_recursive(node.left, data)
        elif data > node.data:
            if node.right is None:
                node.right = Node(data)
            else:
                self._insert_recursive(node.right, data)

    def search(self, data):
        return self._search_recursive(self.root, data)
```

```
    def _search_recursive(self, node, data):
        if node is None or node.data == data:
            return node
        if data < node.data:
            return self._search_recursive(node.left, data)
        return self._search_recursive(node.right, data)

    def delete(self, data):
        self.root = self._delete_recursive(self.root, data)

    def _delete_recursive(self, node, data):
        if node is None:
            return node
        if data < node.data:
            node.left = self._delete_recursive(node.left, data)
        elif data > node.data:
            node.right = self._delete_recursive(node.right, data)
        else:
            if node.left is None:
                return node.right
            elif node.right is None:
                return node.left
            temp = self._get_min_value_node(node.right)
            node.data = temp.data
            node.right = self._delete_recursive(node.right, temp.data)
        return node

    def _get_min_value_node(self, node):
        current = node
        while current.left is not None:
            current = current.left
```

```
    def preorder_traversal(self):
        self._preorder_recursive(self.root)

    def _preorder_recursive(self, node):
        if node is not None:
            print(node.data, end=" ")
            self._preorder_recursive(node.left)
            self._preorder_recursive(node.right)

    def inorder_traversal(self):
        self._inorder_recursive(self.root)

    def _inorder_recursive(self, node):
        if node is not None:
            self._inorder_recursive(node.left)
            print(node.data, end=" ")
            self._inorder_recursive(node.right)

    def postorder_traversal(self):
        self._postorder_recursive(self.root)

    def _postorder_recursive(self, node):
        if node is not None:
            self._postorder_recursive(node.left)
            self._postorder_recursive(node.right)
            print(node.data, end=" ")
```

Now, we will combine all the functions together and create a binary search tree for

a = [49, 38, 65, 97, 60, 76, 13, 27, 5, 1]

```
# Create an instance of the Binary Search Tree
bst = BST()

# Create the binary search tree using the given list
a = [49, 38, 65, 97, 60, 76, 13, 27, 5, 1]
for num in a:
    bst.insert(num)

# Print the initial tree (preorder traversal)
print("Initial tree (preorder traversal):")
bst.preorder_traversal()
print()

# Print the initial tree (inorder traversal)
print("Initial tree (inorder traversal):")
bst.inorder_traversal()
print()

# Print the initial tree (postorder traversal)
print("Initial tree (postorder traversal):")
```

```
# Search for a value in the tree
search_value = 27
result = bst.search(search_value)
if result:
    print(f"{search_value} found in the tree.")
else:
    print(f"{search_value} not found in the tree.")

# Delete a value from the tree
delete_value = 13
bst.delete(delete_value)
print(f"\nAfter deleting {delete_value}:")
bst.inorder_traversal()
```

After running the code, we have found the answer which is shown below-

```
Initial tree (preorder traversal):
49 38 13 5 1 27 65 60 97 76
Initial tree (inorder traversal):
1 5 13 27 38 49 60 65 76 97
Initial tree (postorder traversal):
27 found in the tree.

After deleting 13:
1 5 27 38 49 60 65 76 97
```

Now for the second one, we will create another search tree.

b = [149, 38, 65, 197, 60, 176, 13, 217, 5, 11]

The coding is given below-

```
# Create an instance of the Binary Search Tree
bst = BST()

# Create the binary search tree using the given list
b = [149, 38, 65, 197, 60, 176, 13, 217, 5, 11]
for num in b:
    bst.insert(num)

# Print the initial tree (preorder traversal)
print("Initial tree (preorder traversal):")
bst.preorder_traversal()
print()

# Print the initial tree (inorder traversal)
print("Initial tree (inorder traversal):")
bst.inorder_traversal()
print()

# Print the initial tree (postorder traversal)
print("Initial tree (postorder traversal):")
```

```
# Search for a value in the tree
search_value = 194
result = bst.search(search_value)
if result:
    print(f"{search_value} found in the tree.")
else:
    print(f"{search_value} not found in the tree.")

# Delete a value from the tree
delete_value = 11
bst.delete(delete_value)
print(f"\nAfter deleting {delete_value}:")
bst.postorder_traversal()
```

After running the code, we have found the answer which shown below-

```
Initial tree (preorder traversal):
149 38 13 5 11 65 60 197 176 217
Initial tree (inorder traversal):
5 11 13 38 60 65 149 176 197 217
Initial tree (postorder traversal):
194 not found in the tree.

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

Lastly, for the third one, we will create another search tree.

c = [49, 38, 65, 97, 64, 76, 13, 77, 5, 1, 55, 50]

The code is shown below-

```
# Create an instance of the Binary Search Tree
bst = BST()

# Create the binary search tree using the given list
c = [49, 38, 65, 97, 64, 76, 13, 77, 5, 1, 55, 50]
for num in b:
    bst.insert(num)

# Print the initial tree (preorder traversal)
print("Initial tree (preorder traversal):")
bst.preorder_traversal()
print()

# Print the initial tree (inorder traversal)
print("Initial tree (inorder traversal):")
bst.inorder_traversal()
print()

# Print the initial tree (postorder traversal)
print("Initial tree (postorder traversal):")
```

```
# Search for a value in the tree
search_value = 77
result = bst.search(search_value)
if result:
    print(f"{search_value} found in the tree.")
else:
    print(f"{search_value} not found in the tree.")

# Delete a value from the tree
delete_value = 50
bst.delete(delete_value)
print(f"\nAfter deleting {delete_value}:")
bst.preorder_traversal()
```

After running the code, we have found the answer which shown below-

```
Initial tree (preorder traversal):
149 38 13 5 11 65 60 197 176 217
Initial tree (inorder traversal):
5 11 13 38 60 65 149 176 197 217
Initial tree (postorder traversal):
77 not found in the tree.

After deleting 50:
149 38 13 5 11 65 60 197 176 217
```

Task-2:

In this experiment, we have worked with a csv file called "SDN_traffic.csv". We have used this csv file as dataset and made traffic category classification with decision tree classifiers obtain classification metrics, and compare ID3 and CART are two algorithms's performance differences.

For this experiment, we have tested accuracy for 'category' column in both CART and ID3 algorithm. First, we have used CART algorithm. CART is a popular decision tree algorithm that can be used for both classification and regression tasks. To use the CART algorithm with a big CSV file in Python, we can follow these steps:

Import the necessary libraries-

```
In [32]: import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.preprocessing import LabelEncoder
```

Load the CSV file into a pandas DataFrame-

```
In [33]: data = pd.read_csv('SDN_traffic.csv')
```

As there are some non-numeric values in this dataset, we have to handle that before applying the CART algorithm-

```
In [34]: data =data.fillna(0)

In [35]: non_numeric_columns = data.select_dtypes(exclude=['float','int']).columns

In [36]: for column in non_numeric_columns:
             label_encoder = LabelEncoder()
             data[column] =label_encoder.fit_transform(data[column])
```

Then we have preprocessed the data. First, we split data into features (X) and target variable (y)-

```
In [37]: X = data.drop('category', axis=1)
         y = data['category']
```

Then we have splitted the data into training and testing sets-

```
In [38]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

After that we have created an instance of the DecisionTreeClassifier-

```
In [39]: clf = DecisionTreeClassifier(criterion='gini')
```

Then we have trained the classifier on the training data-

```
In [40]: clf.fit(X_train, y_train)
```

Then we have made the predictions on the testing data

```
In [41]: y_pred = clf.predict(X_test)
```

Finally we have evaluated the performance of the classifier

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7709563164108618
```

Now we are going to test the accuracy for ID3 algorithm. Following steps are given below-

- First, we have preprocess the CSV data. This step involves handling missing values, encoding categorical variables, and any other necessary preprocessing steps to prepare the data for training ID3 algorithm.
- Then we have separated the data into a feature matrix (X) and the corresponding target variable (y) that we want to predict using the ID3 algorithm.
- Finally we can either implement the ID3 algorithm from scratch or use an existing implementation from a library.

The following steps are shown below-

```
In [49]: import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.preprocessing import LabelEncoder
```

```
In [50]: data = pd.read_csv('SDN_traffic.csv')

In [51]: data =data.fillna(0)

In [52]: non_numeric_columns = data.select_dtypes(exclude=['float','int']).columns

In [53]: for column in non_numeric_columns:
             label_encoder = LabelEncoder()
             data[column] =label_encoder.fit_transform(data[column])
```

```
In [54]: X = data.drop('category', axis=1)
         y = data['category']

In [55]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [56]: clf = DecisionTreeClassifier(criterion='entropy')

In [57]: clf.fit(X_train, y_train)

Out[57]: DecisionTreeClassifier
         DecisionTreeClassifier(criterion='entropy')

In [58]: y_pred = clf.predict(X_test)
```

```
y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

Accuracy: 0.768595041322314
```

From the above experiments, we can see that, for both of the algorithm, we have applied test_size=0.2. According to that, we get accuracy when we applied CART algorithm is 0.77 and for ID3, it is 0.76

In this experiment, we have used numpy to find out the accuracy of Decision Classifiers .We can apply them on different types of data sets having different types of values and properties and can attain a best result by knowing that which algorithm will give the best result on a specific type of data set. This research work compares the performance of ID3 and CART algorithms. The experimentation result shows that the CART has the best classification accuracy when compared to ID3.