Implementation of Classification and Regression (CART) Decision Tree Algorithm

Project link: https://github.com/patsongco/32005-Machine-Learning-Assessment-2/blob/main/31005_machine_learning_assessment_2.ipynb

Introduction

Decision trees are a non-parametric supervised learning method used for classification and regression. A decision tree consists of a root node that branches into "child" nodes. Each branch of the tree ends in a terminal node. If the data attributes consist of $X_1, X_2, \dots, X_p$, the decision tree, f_T, maps an input data sample of p observed attribute values to the target of prediction, f_T: $[X_1, X_2, \dots, X_p] \rightarrow Y$. The tree f_T performs the mapping by greedily choosing a collection of rules based on the attribute values, $X_1, X_2, \dots, X_p$, that provide the best split to differentiate the target predictions, *Y*. Once a rule is selected, the node is split into two, and the same process is applied to the each of the two "child" nodes. This process continues reclusively until no further gain can be made, or some stopping criteria is reached.

This report presents the implementation of the CART decision tree algorithm on the IRIS dataset. CART constructs binary trees using the attribute and threshold that yield the largest information gain at each node. This algorithm was chosen over other decision tree algorithms, such as the ID3 algorithm, because it supports both categorical and numerical target variables. Implementation details will be described in detail.

Classification and Regression Tree (CART) Algorithm

The algorithm consists of one Class but contains some key functions and important split criteria calculations. Namely, the impurity criteria calculation and the information gain calculation. The greedy algorithm identifies a split on an attribute, such as "petal_length", that results in the largest information gain (IG) for a given impurity criterion (such as Entropy or Gini). Once a split is determined, the tree recursively adds children nodes until no further information gain can be made. A method for pruning the tree is included in the implementation to limit the complexity of the decision tree and reduce overfitting. A print method is also included to aid in viewing the output predictions. To fit the data, the algorithm input requires a Dataframe that contains *n* observed data samples of *p* attributes and a Series of *n* target values. The output is a Tree Object that fits the training data. For prediction, input is also a Dataframe, but the output is a List of predicted values. Technical details of the implementation as follows:

Impurity Criterion

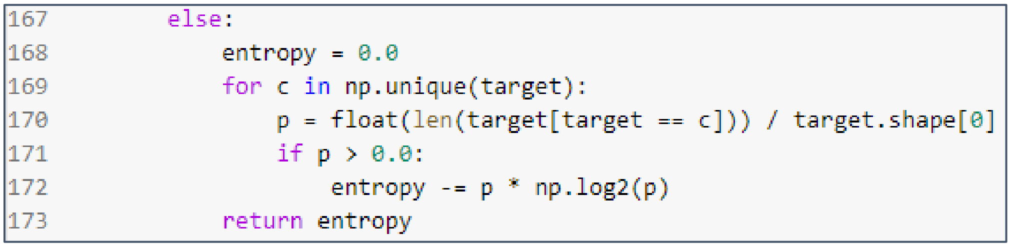
The impurity criterion is a measure of the homogeneity of the target labels at the current node. The CART algorithm implementation utilizes Entropy, Gini and Variance impurity criterions:

Entropy:

$$I_e = -\sum_{j=1}^{c} p_j \log_2(p_j) \quad \textbf{(1)}$$

p_j is the proportion of the samples that belong to class c for a particular node
*this is the definition of entropy for all non-empty classes ($p \neq 0$). The entropy is 0 if all samples at a node belong to the same class

The entropy calculation, like all impurity calculations are nested in the *_calc_impurity*() function (*line 152-173*) of the algorithm (Figure 1 below). The entropy is lower if more samples at a node belong to the same class. Stepping through the implementation, the for-loop (*line 169*, Figure 1) loops over all unique target values calculating the proportion of samples, p_j, that belong to that class, *c*. *Line 172* computes the total entropy by adding each instance of $p_j * \log_2(p_j)$ and taking the negative. The entropy is returned to be later used in the Information Gain calculation.



```
167         else:
168             entropy = 0.0
169             for c in np.unique(target):
170                 p = float(len(target[target == c])) / target.shape[0]
171                 if p > 0.0:
172                     entropy -= p * np.log2(p)
173             return entropy
```

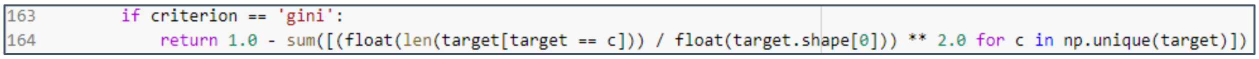
Figure 1: Entropy calculation code snippet

Gini Index:

$$I_G = 1 - \sum_{j=1}^{c} p_j^2 \qquad \textbf{(2)}$$

p_j is the proportion of the samples that belong to class c for a particular node

The Gini Index calculation is nested in the *_calc_impurity*() function (*line 152-173*) of the algorithm (Figure 2 below). The Gini Index is lower if more samples at a node belong to the same class and 0.5 (for a 2-class problem) if the classes are very heterogeneous. The equation (*line 164*, Figure 2) takes the square of the proportion of samples, p_j, that belong to a class, *c*, and adds these values for all unique classes. This total is subtracted from 1 and returned to be later used in the Information Gain calculation.



```
163         if criterion == 'gini':
164             return 1.0 - sum([(float(len(target[target == c])) / float(target.shape[0])) ** 2.0 for c in np.unique(target)])
```

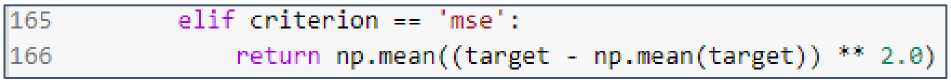
Figure 2: Gini Index calculation code snippet

Variance:

$$I_V = \frac{1}{N} \sum_{i=1}^{N} (y_i - \mu)^2 \qquad \textbf{(3)}$$

y_i is the label for an instance
N is the number of instances
μ is the mean given by $\frac{1}{N}\sum_{i=1}^{N} y_i$

The variance or mean squared error is used for instances where the target variable is continuous (regression). It is nested in the *_calc_impurity*() function (*line 152-173*) of the algorithm (Figure 3 below). The equation averages the square distance between the instance value and the mean. Implementation (*line 166*) makes use of the NumPy element wise mean function to return the variance that is used in the Information Gain calculation.



```
        elif criterion == 'mse':
            return np.mean((target - np.mean(target)) ** 2.0)
```

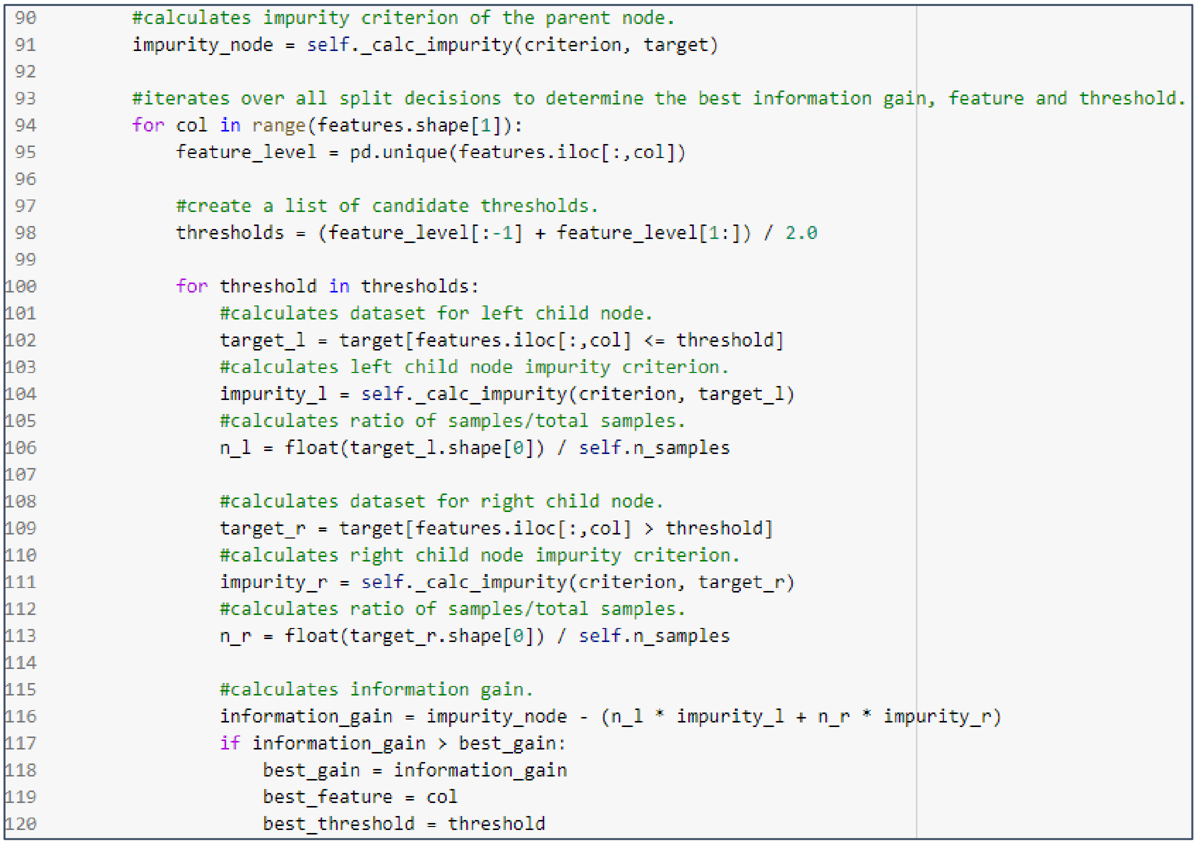
Figure 3: Variance or Mean Square Error calculation code snippet

Information Gain

$$IG(D,s) = Impurity(D) - \frac{N_{left}}{N} Impurity\left(D_{left}\right) - \frac{N_{right}}{N} Impurity\left(D_{right}\right) \tag{4}$$

D : Dataset of the parent node
D_{left} : Dataset of the left child node
D_{right} : Dataset of the right child node
s : Partition split of data on feature
N : Total number of samples
N_{left} : Number of samples at left child node
N_{right} : Number of samples at right child node
$Impurity$: Impurity criterion function (Entropy, Gini or Variance)

The information gain is the difference between the parent node impurity and the weighted sum of the two child nodes impurities. The chosen split at each tree node is chosen from the set $argmax\, IG(D,s)$ in order to maximize the information gain at each tree node. In the CART algorithm implementation, thresholds (splits) and their respective information gain value are computed, and the largest value stored.



```
        #calculates impurity criterion of the parent node.
        impurity_node = self._calc_impurity(criterion, target)

        #iterates over all split decisions to determine the best information gain, feature and threshold.
        for col in range(features.shape[1]):
            feature_level = pd.unique(features.iloc[:,col])

            #create a list of candidate thresholds.
            thresholds = (feature_level[:-1] + feature_level[1:]) / 2.0

            for threshold in thresholds:
                #calculates dataset for left child node.
                target_l = target[features.iloc[:,col] <= threshold]
                #calculates left child node impurity criterion.
                impurity_l = self._calc_impurity(criterion, target_l)
                #calculates ratio of samples/total samples.
                n_l = float(target_l.shape[0]) / self.n_samples

                #calculates dataset for right child node.
                target_r = target[features.iloc[:,col] > threshold]
                #calculates right child node impurity criterion.
                impurity_r = self._calc_impurity(criterion, target_r)
                #calculates ratio of samples/total samples.
                n_r = float(target_r.shape[0]) / self.n_samples

                #calculates information gain.
                information_gain = impurity_node - (n_l * impurity_l + n_r * impurity_r)
                if information_gain > best_gain:
                    best_gain = information_gain
                    best_feature = col
                    best_threshold = threshold
```

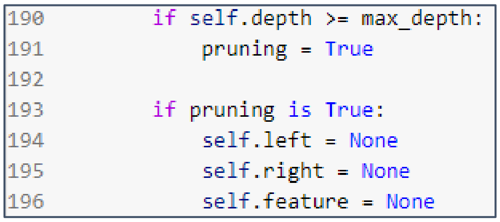
Figure 4: Information Gain calculation code snippet

The Information Gain calculation is nested in the *_grow_tree*() function (*line 61-127*) of the algorithm (Figure 4 above). Stepping through algorithm **(4)**:

- $Impurity(D)$, or the impurity criterion of the parent node is calculated at *line 91*
- The two for-loops loop over the unique classes and determines different threshold values that are evaluated for the highest information gain (*line 94* and *100*)
- $Impurity(D_{left})$, or the impurity of the left child node is calculated on *line 104*
- $Impurity(D_{right})$, or the impurity of the right child node is calculated on *line 111*
- $\frac{N_{left}}{N}$ and $\frac{N_{right}}{N}$ are calculated on *lines 106* and *113*, respectively
- Finally, putting all these together, *line 116* calculates the Information Gain for that threshold or split condition. This Information Gain is stored if it is the largest.

Pruning

The CART algorithm will repeatedly partition data into smaller and smaller subsets until those final subsets are homogeneous in terms of the target variable. The _prune() method is invoked at the end of building or fitting the tree (*line 38*) and involves cutting the leaf nodes of a tree after it has been built to reduce overfitting of the dataset. The main algorithm details are in the *lines 190-196* (Figure 5 below). *Line 190* checks if the current node depth is greater than or equal to the max_depth (which is set by the user). If so, the child nodes are deleted.

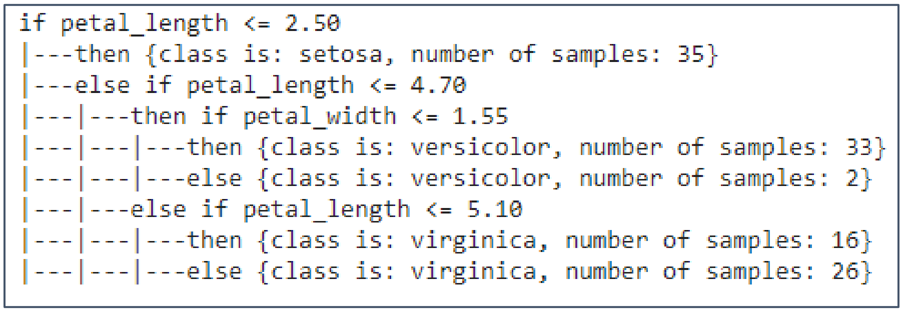


```
190         if self.depth >= max_depth:
191             pruning = True
192
193         if pruning is True:
194             self.left = None
195             self.right = None
196             self.feature = None
```

Figure 5: Pruning code snippet

Printing

A simple print method is added to the Tree Object Class so that the user can view the threshold and attribute split conditions. (Figure 6 below).



```
if petal_length <= 2.50
|---then {class is: setosa, number of samples: 35}
|---else if petal_length <= 4.70
|---|---then if petal_width <= 1.55
|---|---|---then {class is: versicolor, number of samples: 33}
|---|---|---else {class is: versicolor, number of samples: 2}
|---|---else if petal_length <= 5.10
|---|---|---then {class is: virginica, number of samples: 16}
|---|---|---else {class is: virginica, number of samples: 26}
```

Figure 6: Example print out of tree

Model Evaluation

The CART algorithm is evaluated on the IRIS Flower Dataset. The dataset is loaded from the seaborn library. It contains 4 attribute columns (*sepal_length, sepal_width, petal_length, petal_width*) and one target column (*species*). The target column contains 3 classes: *Sentosa, Versicolor and Virginica* (Figure 7). The IRIS dataset contains 150 entries or rows, with all values consisting of the data type *float64* (Float), except for the target class, which is of the data type *object* (String).

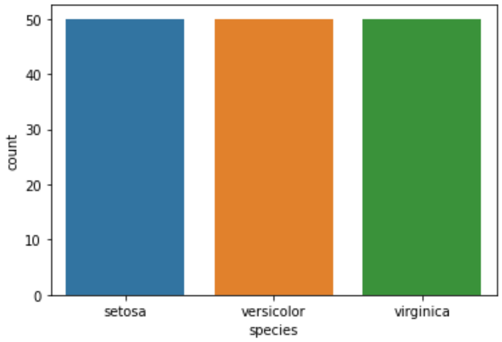
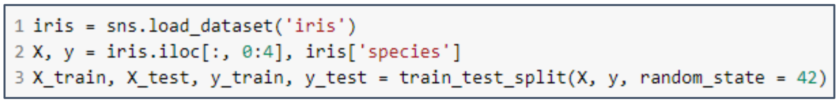


Figure 7: Distribution of target variable

The model will be evaluated by building a tree to fit the IRIS dataset and then testing the model accuracy on an unseen subset of the IRIS dataset. Further evaluation of the model will be completed by comparing the accuracy of the CART model to the accuracy of the Sklearn Decision Tree Classifier.

Data Preparation

Data preparation involved splitting the Dataframe into training and test sets. First, the attribute columns are separated from the target column (*line 2*, Figure 8). Then, using the train_test_split() function from the Sklearn library, we split the dataset into training and test sets (*line 3*, Figure 8)



```
1 iris = sns.load_dataset('iris')
2 X, y = iris.iloc[:, 0:4], iris['species']
3 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

Figure 8: Data preparation

Experiment Design and Evaluation

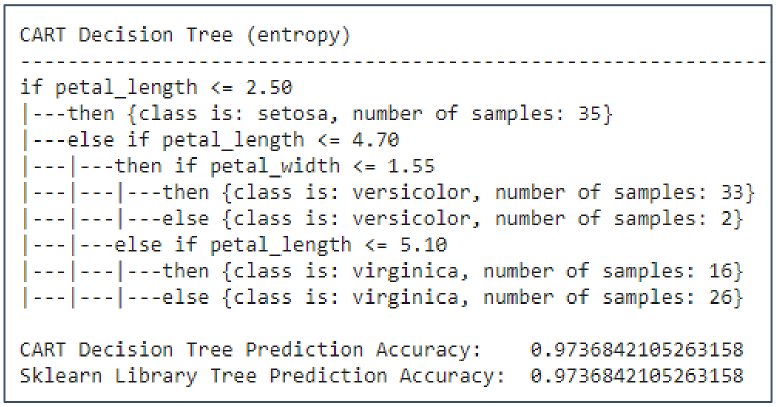
The first round of tests invokes the CART decision tree algorithm with the hyperparameter configuration: *tree* = “cls”, *criterion* = “entropy”, *max_depth* = 3. These hyperparameters configure the tree algorithm to use classification, entropy criterion and a maximum tree depth of 3. The model is constructed and fit on the training dataset (X_train, y_train). The trained model is then used to predict the target values of the test set (X_test). The accuracy is evaluated by comparing the prediction output to the test set target values (y_test). The accuracy of the model is then compared to the accuracy of the Sklearn Decision Tree Classifier.

The second round of tests uses the exact same experiment design as the first round, although, the *criterion* hyperparameter is changed to “gini”.

The third round of tests uses the exact same experiment design as the first round, although, the *max_depth* hyperparameter is changed to “5”.

Evaluation Results

First Round: Figure 9 below displays the attribute and threshold splits of the decision tree; how many samples were classified to each class and accuracy of the model.

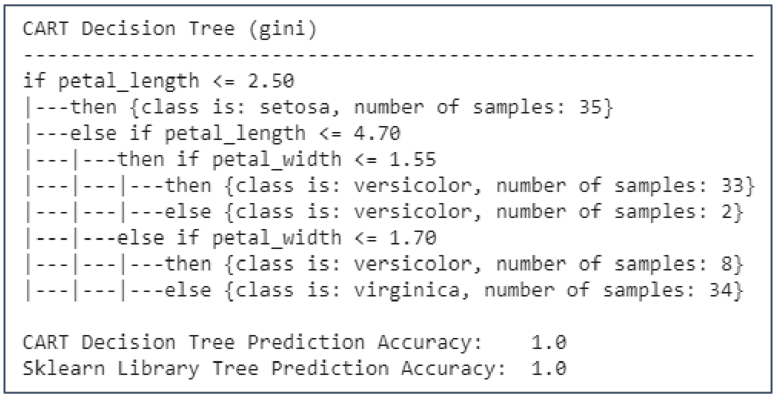


```
CART Decision Tree (entropy)
---------------------------------------------------------------
if petal_length <= 2.50
|---then {class is: setosa, number of samples: 35}
|---else if petal_length <= 4.70
|---|---then if petal_width <= 1.55
|---|---|---then {class is: versicolor, number of samples: 33}
|---|---|---else {class is: versicolor, number of samples: 2}
|---|---else if petal_length <= 5.10
|---|---|---then {class is: virginica, number of samples: 16}
|---|---|---else {class is: virginica, number of samples: 26}

CART Decision Tree Prediction Accuracy:    0.9736842105263158
Sklearn Library Tree Prediction Accuracy:  0.9736842105263158
```

Figure 9: Output of the first round

Second Round: Figure 10 below displays the attribute and threshold splits of the decision tree; how many samples were classified to each class and accuracy of the model.

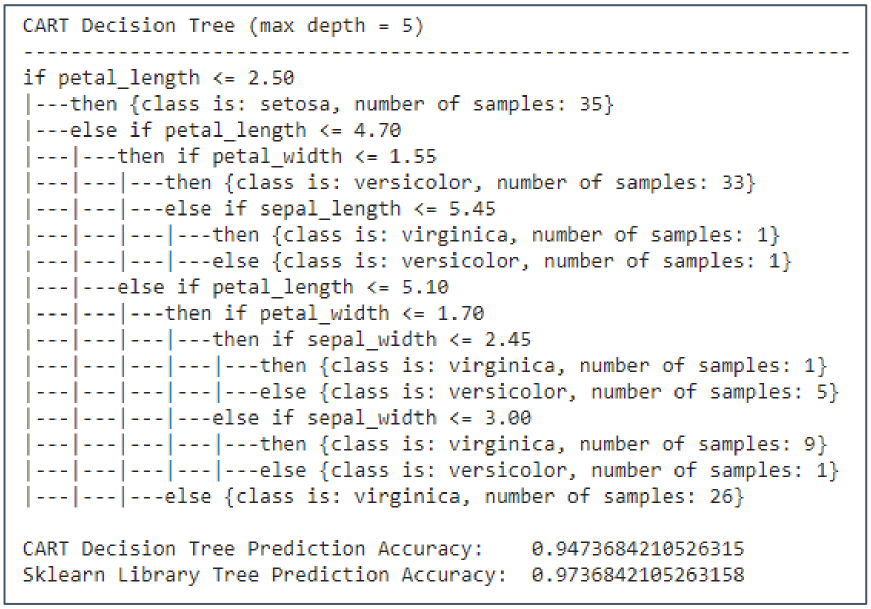


```
CART Decision Tree (gini)
--------------------------------------------------------------
if petal_length <= 2.50
|---then {class is: setosa, number of samples: 35}
|---else if petal_length <= 4.70
|---|---then if petal_width <= 1.55
|---|---|---then {class is: versicolor, number of samples: 33}
|---|---|---else {class is: versicolor, number of samples: 2}
|---|---else if petal_width <= 1.70
|---|---|---then {class is: versicolor, number of samples: 8}
|---|---|---else {class is: virginica, number of samples: 34}

CART Decision Tree Prediction Accuracy:    1.0
Sklearn Library Tree Prediction Accuracy:  1.0
```

Figure 10: Output of the second round

Third Round: Figure 11 below displays the attribute and threshold splits of the decision tree; how many samples were classified to each class and accuracy of the model.



```
CART Decision Tree (max depth = 5)
----------------------------------------------------------------------
if petal_length <= 2.50
|---then {class is: setosa, number of samples: 35}
|---else if petal_length <= 4.70
|---|---then if petal_width <= 1.55
|---|---|---then {class is: versicolor, number of samples: 33}
|---|---|---else if sepal_length <= 5.45
|---|---|---|---then {class is: virginica, number of samples: 1}
|---|---|---|---else {class is: versicolor, number of samples: 1}
|---|---else if petal_length <= 5.10
|---|---|---then if petal_width <= 1.70
|---|---|---|---then if sepal_width <= 2.45
|---|---|---|---|---then {class is: virginica, number of samples: 1}
|---|---|---|---|---else {class is: versicolor, number of samples: 5}
|---|---|---|---else if sepal_width <= 3.00
|---|---|---|---|---then {class is: virginica, number of samples: 9}
|---|---|---|---|---else {class is: versicolor, number of samples: 1}
|---|---|---else {class is: virginica, number of samples: 26}

CART Decision Tree Prediction Accuracy:    0.9473684210526315
Sklearn Library Tree Prediction Accuracy:  0.9736842105263158
```

Figure 11: Output of the third round

Conclusion

Evaluating the CART decision tree model using the IRIS dataset (Figure 9, 10, 11), we obtain an accuracy of over 94% for all instances of the model. The highest prediction accuracy on the test set was obtained using the Gini Impurity criterion over the Entropy Impurity criterion. We also see a decline in accuracy when the depth of the tree is increased to 5, this shows that the model tends to overfit the training data as the tree depth increases. This also confirms that pruning the tree has a positive outcome and suggests that more sophisticated pruning techniques can be implemented in the future.

Printing the tree gives us some intuition on which attributes and thresholds the split decisions are made. This shows us which decisions provide the most information gain at each level or node of the tree.

Lastly, the accuracy of our CART decision tree model is comparable to the out-of-the-box Sklearn Decision Tree Classifier, which implies that our implementation is correct.