PROJECT JOURNAL – DECISION TREE CLASSIFICATION

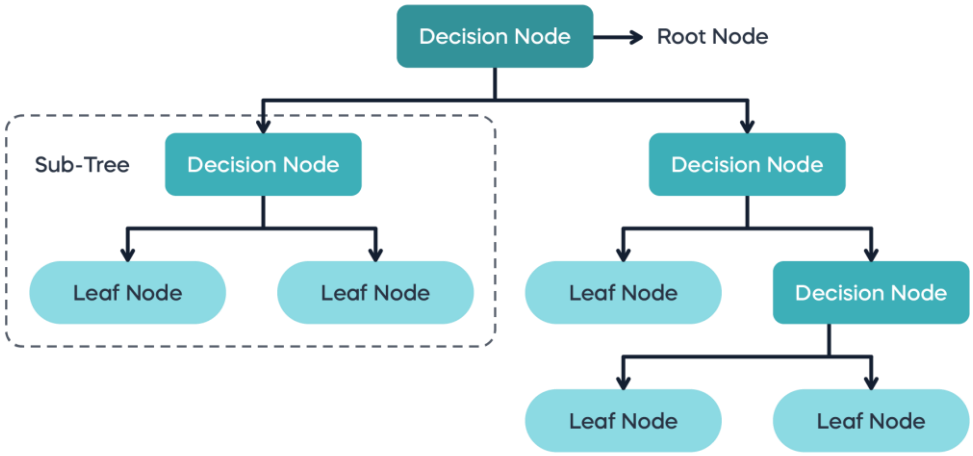
INTRODUCTION

This journal will be focusing on the journey of building a decision tree (DT) model from scratch by understanding the technical details and functions of DT and how the algorithm performs classifications and make predictions. My interest on studying such model came from the fact that DT is a commonly used algorithm that can predict both continuous and discrete values which means, they work well with regression and classification tasks, and it is one of the simple algorithms that can be easily explored and understood by.

With the power of artificial intelligence (AI) in this modern world, I will be utilising ChatGPT to help strength my understandings of the technical details of DT and assist with building the DT from scratch using Python. To test on the accuracy and reliability of the DT built from scratch, I will be using the Iris dataset to predict the species name based on the features of the Iris flowers that includes sepal length, sepal width, petal length and petal width.

THEORY OF DECISION TREE (DT)

DT is a supervised learning technique that can be used for both classification and regression problems, but mostly it is preferred for solving Classification problems. The DT is composed of two nodes: decision nodes and leaf nodes. Decision nodes represent the test on a feature and the leaf nodes represents the possible outcomes/values for that feature that doesn't contain further branches (further outcomes). This can be illustrated with the graphic below.



To select the best attributes for the root node and sub nodes, attribute selection measure (ASM) is used. There are 2 popular techniques for ASM, which are: Gini index and Entropy. These two measures of impurity of nodes can be used to get the information gain which determines which feature to be used for the next split.

Entropy

A measure of disorder or impurity in a node. Entropy has a range of 0 to 1 where higher value of entropy indicates higher level of impurity in a node. The higher the impurity, the harder it is to draw any conclusion. When a node consists of multiple classes, it is impure and especially with equal number of values between the classes, the entropy will be at its highest (=1).

$$E = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Information Gain

A measure of expected reduction in entropy (impurity in the data). This change in entropy represents how much information a feature provides for the target variable which is therefore used to decide which feature to split on at each step in building the tree. As the creation of sub nodes increases homogeneity (that is decreases the entropy of these nodes), more variances will be decreased after each split. Thus, information gain is the variance reduction and can calculate by how much the variance decreases after each split.

Information gain of a parent node can be calculated as the entropy of the parent node subtracted entropy of the weighted average of the child node. By calculating all features within the dataset, feature having maximum information gain will be the most important feature which will be the root node for the decision tree.

Gini Index/Impurity

Measure the probability for a random instance being misclassified when chosen randomly. The lower the Gini index, the better the lower the likelihood of misclassification. It is very simple to implement, and it only performs binary split. For categorical variables, it gives the results in terms of "success" or "failure". The Gini index has a minimum (highest level of purity) of 0 and a maximum value of 0.5. It can be calculated by below formula where c = number of classes and pi = probability associated with ith class.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Process of splitting the trees

1. Determining the root node → this is done by calculating the information gain of all features in respective to the output/outcome ➔ each child node split from each feature will have its own information gain calculated, then the weighted/average information gain of both determines the total information gain of that feature ➔ feature with the total highest information gain will be selected as the root node
2. After the root node split into 2 child nodes with 2 different threshold value, each feature from both child will be tested again in respective to the outcome. Each child node's IG for each feature will again be calculated and the total IG from both child of that feature will be calculated → highest total IG will be selected for next split ➔ If a node became pureed (leaf node), there will be no further splits and the value of that leaf node will be the final value.
3. For decision node as of the child node from the root node, if a node is impure, it will split further and this split will again involve with splitting criteria, choosing the feature that gives the highest information gain.
4. This process will be repeated until the maximum depth is reached or when all nodes are pureed. If maximum depth is reached and not all nodes became pure, it will take the majority class value as of its final value.

IMPLEMENTATION OF THE ALGORITHM

Google Collaboration Link:
https://colab.research.google.com/drive/1pjMK0IPRpioktQF7naC8OXLKDuzTSYOB?usp=sharing

1. Node class:

The node class provides constructors with different attributes which will be used later when building the tree. The decision node evaluates a condition which is defined by feature index, threshold, left child, right child, information gain. The value node represents the value of leaf node.

- ⇨ Feature index = feature attribute (e.g. sepal length)
- ⇨ Threshold = the condition that differentiates and distributes data points into left and right child
- ⇨ Info gain = the information gain of each feature after each split, determines the next feature for split
- ⇨ Value = leaf value when there is no further split (pure node) or when maximum depth have reached, taking the majority class of the node

Node Class

```
[6]  class Node():
         def __init__(self, feature_index=None, threshold=None, left=None, right=None, info_gain=None, value=None):
             ''' constructor '''

             # for decision node
             self.feature_index = feature_index
             self.threshold = threshold
             self.left = left # access left child/node
             self.right = right #access right child/node
             self.info_gain = info_gain

             # for leaf node
             self.value = value #majority class of the node
```

1. Tree class:

Constructor that initialises the value for minimum split and maximum depth in order to reduce overfitting

```
class DecisionTreeClassifier():
    def __init__(self, min_samples_split=4, max_depth=4):
        ''' constructor '''

        # initialize the root of the tree
        self.root = None

        # stopping criteria
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
```

1.1. Building Tree Method

The build tree method is a recursive method to build the binary tree. it is just splitting the features (‘X) from the target variable (‘Y’).

Then, it extracts the number of samples and the number of features by using the np.shape function. In the ‘if statement’ it is checking if the number of samples and depth are met or not. If these two conditions are not met, we can't split the tree further.

if information gain is positive, it will split the node into left and right child irrespectively and updates its decision node. If condition are met, it will compute the leaf node and return its value.

```
def build_tree(self, dataset, curr_depth=0):
        ''' recursive function to build the tree '''

        X, Y = dataset[:,:-1], dataset[:,-1]
        num_samples, num_features = np.shape(X)

        # split until stopping conditions are met
        if num_samples>=self.min_samples_split and curr_depth<=self.max_depth:
                # find the best split
                best_split = self.get_best_split(dataset, num_samples, num_features)
                # check if information gain is positive
                if best_split["info_gain"]>0:
                        # recur left
                        left_subtree = self.build_tree(best_split["dataset_left"], curr_depth+1)
                        # recur right
                        right_subtree = self.build_tree(best_split["dataset_right"], curr_depth+1)
                        # return decision node
                        return Node(best_split["feature_index"], best_split["threshold"],
                                    left_subtree, right_subtree, best_split["info_gain"])

        # compute leaf node
        leaf_value = self.calculate_leaf_value(Y)
        # return leaf node
        return Node(value=leaf_value)
```

1.2. Get Best Split Method

```
def get_best_split(self, dataset, num_samples, num_features):
        ''' function to find the best split '''

        # dictionary to store the best split
        best_split = {}
        max_info_gain = -float("inf")

        # loop over all the features
        for feature_index in range(num_features): #explore through every possible values of a feature in the dataset
                feature_values = dataset[:, feature_index]
                possible_thresholds = np.unique(feature_values)
                # loop over all the feature values present in the data
                for threshold in possible_thresholds:
                        # get the current split
                        dataset_left, dataset_right = self.split(dataset, feature_index, threshold)
                        # check if childs are not null
                        if len(dataset_left)>0 and len(dataset_right)>0:
                                y, left_y, right_y = dataset[:, -1], dataset_left[:, -1], dataset_right[:, -1] #extracting the target value "y"
                                # compute information gain
                                curr_info_gain = self.information_gain(y, left_y, right_y, "gini")
                                # update the best split if needed
                                if curr_info_gain>max_info_gain:
                                        best_split["feature_index"] = feature_index
                                        best_split["threshold"] = threshold
                                        best_split["dataset_left"] = dataset_left
                                        best_split["dataset_right"] = dataset_right
                                        best_split["info_gain"] = curr_info_gain
                                        max_info_gain = curr_info_gain

        # return best split
        return best_split
```

I have initialized a maximum IG as negative infinity. Because we want to maximize the IG and to find that we must use a number that is less than any other number. Now first, it loops through all the features and inside this loop it traverses through all the possible threshold values to find the best feature.

It then checks to ensure that the child is not empty, extracts the target y from original dataset and extracts the target/final value of the left and right child. It then calculates the information gain and if the current information gain is greater than maximum information gain that have found, it will update it along with its attribute within the best split dictionary.

After exploring all possible splits, it will return the best split dictionary with information about the best split (the split with highest gain, returned from the information gain function).

1.3. Split method

```
def split(self, dataset, feature_index, threshold):
        ''' function to split the data '''

        dataset_left = np.array([row for row in dataset if row[feature_index]<=threshold])
        dataset_right = np.array([row for row in dataset if row[feature_index]>threshold])
        return dataset_left, dataset_right
```

Checks each row in dataset and split the dataset into left and child node according to the threshold criteria and finally, return these 2 dataset/child.

1.4. Information Gain Function

```
def information_gain(self, parent, l_child, r_child, mode="entropy"):
        ''' function to compute information gain '''

        weight_l = len(l_child) / len(parent)
        weight_r = len(r_child) / len(parent)
        if mode=="gini":
                gain = self.gini_index(parent) - (weight_l*self.gini_index(l_child) + weight_r*self.gini_index(r_child))
        else:
                gain = self.entropy(parent) - (weight_l*self.entropy(l_child) + weight_r*self.entropy(r_child))
        return gain
```

It calculates the weights for left and right child in respective to the decision/parent node (the Gini impurity/entropy for each child node). Then the total Gini impurity/entropy is calculated for each feature split.

1.5. Entropy Function

```
def entropy(self, y):
        ''' function to compute entropy '''

        class_labels = np.unique(y)
        entropy = 0
        for cls in class_labels:
                p_cls = len(y[y == cls]) / len(y)
                entropy += -p_cls * np.log2(p_cls)
        return entropy
```

This function simply calculates the entropy for a given array of target variables

1.6. Gini index Function

```
def gini_index(self, y):
        ''' function to compute gini index '''

        class_labels = np.unique(y)
        gini = 0
        for cls in class_labels:
                p_cls = len(y[y == cls]) / len(y)
                gini += p_cls**2
        return 1 - gini
```

Same as Entropy function, calculates the Gini index for a given array of target variables.

1.7. Print Tree Method

```
def print_tree(self, tree=None, indent=" "):
        ''' function to print the tree '''

        if not tree:
                tree = self.root

        if tree.value is not None:
                print(tree.value)

        else:
                print("X_"+str(tree.feature_index), "<=", tree.threshold, "?", tree.info_gain)
                print("%sleft:" % (indent), end="")
                self.print_tree(tree.left, indent + indent)
                print("%sright:" % (indent), end="")
                self.print_tree(tree.right, indent + indent)
```

Simply builds the tree where if no tree is defined, it will set as root and if the tree value is not none (i.e. a leaf node), it will return its current value.

1.8. Fit Method

```
def fit(self, X, Y):
        ''' function to train the tree '''

        dataset = np.concatenate((X, Y), axis=1) #used X and Y to create the dataset
        self.root = self.build_tree(dataset)
```

Combining the X (features) and Y (target variable) into a single array. Then the build_tree function is called where the root node will be returned by the build_tree function and it will store that into the self.root.

Predict Method

```
def predict(self, X):
        ''' function to predict new dataset ''' # using new features

        preditions = [self.make_prediction(x, self.root) for x in X]
        return preditions
```

It takes a test dataset make predictions. To do this, it uses a function called make_prediction. We are passing the root node as a parameter to the make_prediction function

1.9. Make Prediction Method

```
def make_prediction(self, x, tree):
        ''' function to predict a single data point '''

        if tree.value!=None: return tree.value      #if a leaf node, return its value
        feature_val = x[tree.feature_index]
        if feature_val<=tree.threshold:
                return self.make_prediction(x, tree.left)
        else:
                return self.make_prediction(x, tree.right)
```

The function takes a node as a parameter. If node is a leaf node, then it just returns the value otherwise it extracts the feature value of our new data point at the given feature index. Then it checks if the feature value is less than or equal to the threshold. If it's true, then it recurses through the left subtree else we recurse through the right subtree

MODEL EVALUATION/RESULTS

Data Preparation:

the iris dataset is used as a case study for this analysis. The iris dataset has four predictor attributes namely ***sepal_length, sepal_width, petal_length, petal_width***. The last attribute is species having three outcomes (Setosa, Virginia, and Versicolor). This dataset set has 150 samples with 50 samples for each species. I have downloaded the iris dataset and placed this in the local drive of my laptop. Then, I have loaded this dataset into google collaboration through:

```
from google.colab import files
uploaded = files.upload()
iris_data = pd.read_csv("iris.csv")
iris_data.head()
```

Train The Test-Split:

```
from sklearn.model_selection import train_test_split
X = iris_data.iloc[:, :-1].values
Y = iris_data.iloc[:, -1].values.reshape(-1,1)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.2, random_state=41)
```

This divided the dataset into training and testing with a ratio of 80:20.

Fit The Model:

```
[69]  classifier = DecisionTreeClassifier(min_samples_split=11, max_depth=4)
      classifier.fit(X_train,Y_train)
      classifier.print_tree()
      Y_pred = classifier.predict(X_test)
```

Defined the minimum sample split to 11, maximum depth to 4 as suggested by AI assistance and used to fit function to train the model and used to print tree function to print the tree.

Results:

```
[75]  Y_pred = classifier.predict(X_test)
      from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
[71]  print("Accuracy Score =", accuracy_score(Y_test, Y_pred))
```

Y_pred represents predicted species of the iris flowers by only providing the predictors (X_test) to the trained classifier. The result shows an accuracy of 0.9667, approximately 96.67% success rates.

Below shows the classification report consisting precision, recall and f1-score for the model.

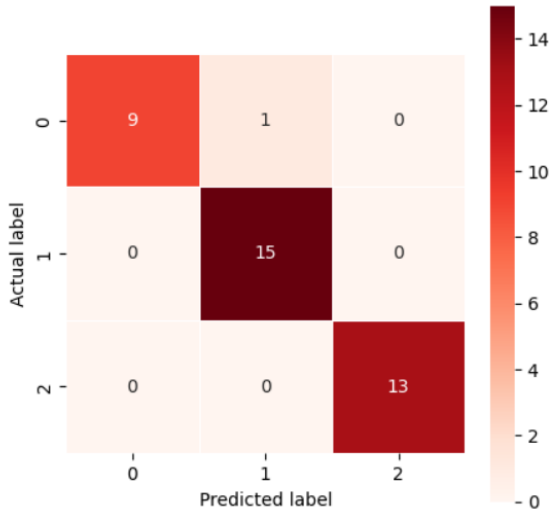
```
print("Classification Report - \n\n", classification_report(Y_test, Y_pred))
```

```
Classification Report -

              precision    recall  f1-score   support

      setosa       1.00      0.90      0.95        10
  versicolor       0.94      1.00      0.97        15
   virginica       1.00      1.00      1.00        13

    accuracy                           0.97        38
   macro avg       0.98      0.97      0.97        38
weighted avg       0.98      0.97      0.97        38
```

Below shows the confusion matrix of the model, the correlation between the predicted label and actual label.

```
[11] import matplotlib.pyplot as plt
     import seaborn as sns
     %matplotlib inline
     # for encoding
     cm = confusion_matrix(Y_test, Y_pred)
     cm
     plt.figure(figsize=(5,5))
     sns.heatmap(data=cm, linewidths=.5, annot=True, square = True, cmap = 'Reds')
     plt.ylabel('Actual label')
     plt.xlabel('Predicted label')
```

```
Text(0.5, 63.222222222222186, 'Predicted label')
```



LINKING TO THE LEARNING FRAMEWORK

Data

In decision tree, the data component represents the independent and dependent variables of the dataset. In the case of Iris dataset, the independent variable is sepal length, sepal width, petal length and petal length will be used to train and predict a dependent variable/outcome of the species name amongst setosa, virginica and versicolor.

Hypothesis Family

In decision tree, the hypothesis family could be considered as the set of all possible trees that could be generated using different combinations of features and thresholds. Each decision tree in this family represents a different hypothesis about how the data is structured. In the context of Iris dataset, this is to build a tree with the optimal splitting rules applied that selects the best possible features and threshold for the next splits. By following such splitting rule, it will ultimately result with an optimum tree which has the highest accuracy when predicting new variables.

Algorithm

In decision tree, the algorithm plays an important role during the training process of building the tree. This is the specific set of rules and steps that the DT follows to make decisions. It involves selecting the best features to split on, determining the thresholds for those features, and creating branches based on the conditions.

Loss

In decision tree, although it doesn't have a specific loss function to evaluate the model performance, however, the splitting criterion/functions and the use of maximum depth and minimum split act similarly to the nature of the loss function. The splitting functions when building a tree ensures that misclassification error is reduced with an optimal feature and threshold selection. The use of different maximum depth may result in different loss result of the model where higher maximum depth may lead to overfitting (over-training the training dataset where noise is considered) or lower maximum depth may lead to underfitting (failed to learn the patterns in the data, resulting in poor performance of the training and unseen data).

AI ASSISTANCE

By exploring through the technical details and functions of the decision tree, I came across these terms: Entropy, Gini index and Information Gain. As reading off from the websites, I still couldn't get a good understanding of these terms and its differences. So I asked ChatGPT to assist my understanding by explaining each term in simple terms.

In simpler terms:

- Imagine you have a big box of mixed fruits (apples, oranges, and bananas). Entropy is a measure of how mixed up they are. If all the fruits are the same, the entropy is low (like having only apples in the box). If there's a good mix of different fruits, the entropy is high.
- Gini index is similar but it's like asking, "If I randomly pick two fruits from the box, what's the chance they will be different?" If the box is pure (only one type of fruit), the chance is zero (lowest Gini index). If the box is a mix of fruits, the chance is higher (higher Gini index).
- Information gain is like asking, "If I separate the fruits based on their color, how much more organized will the box be?" We want to choose the separation (feature) that gives us the most organized subsets.

In summary, decision trees use entropy and Gini index to measure disorder in data, and information gain helps them decide which feature to split on to reduce that disorder and make more accurate predictions.

Throughout the study of the decision tree model, by exploring through its limitations, one most important limitation is overfitting. Overfitting occurs when the model learns too much from the training set where noise is consisted within the learning process. Therefore, by solving this challenge, I have researched further where setting minimum samples split, and maximum depth can solve overfitting in an efficient way. So I asked ChatGPT with its preferred value for each one and I have incorporated the median number of the suggested value as ChatGPT have given a range.

For the Iris dataset, which is a relatively small dataset with 150 samples and 4 features, you can start with the following values for `min_samples_split` and `max_depth`:

1. `min_samples_split`:
   - Since the Iris dataset has 150 samples, a reasonable starting value for `min_samples_split` might be around 5% to 10% of the total samples. This translates to roughly 7 to 15 samples.
2. `max_depth`:
   - For a relatively simple dataset like Iris, you can start with a small value for `max_depth`, such as 3 to 5. This should be sufficient to capture the underlying patterns in the data.

Keep in mind that these are just starting points. You may need to further fine-tune these hyperparameters through experimentation and potentially using techniques like cross-validation.

Regenerate

REFERENCES

javaTpoint. (n.d.). *Decision tree algorithm in Machine Learning - Javatpoint*. www.javatpoint.com. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm#:~:text=In%20a%20Decision%20tree%2C%20there,not%20contain%20any%20further%20branches.

Normalized Nerd. (2021). Decision Tree Classification Clearly Explained!. *YouTube*. https://www.youtube.com/watch?v=ZVR2Way4nwQ

Normalized Nerd. (2021). Decision Tree Classification in Python (from scratch!). *YouTube.* https://www.youtube.com/watch?v=sgQAhG5Q7iY

Sharma, A. (2023, September 18). *4 simple ways to split a decision tree in Machine Learning (updated 2023)*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/06/4-ways-split-decision-tree/

StatQuest with Josh Starmer. (2021). Decision and Classification Trees, Clearly Explained!!!. *YouTube.* https://www.youtube.com/watch?v=_L39rN6gz7Y