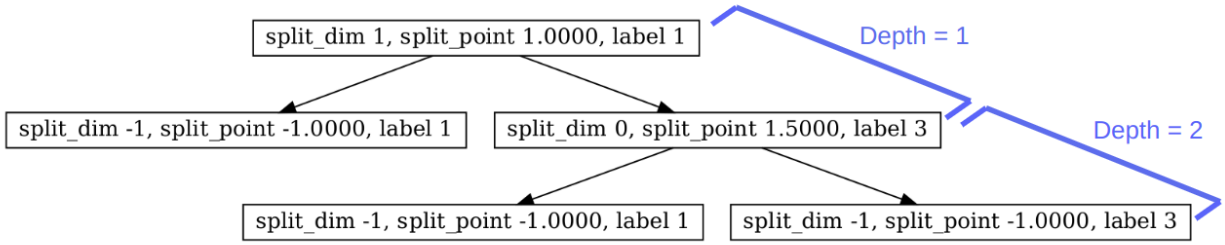
Programming Assignment - Decision Tree

Problem Description

In this programming assignment, our overall task is to create a decision tree with a maximum depth of 2 and traverse this decision tree to predict labels for the test data.

To complete this task, the objective is to complete three helper functions. The Autograder will test your work using these functions. If you need to, feel free to create extra functions to help you finish the main three helper functions.



Please complete the submission template in `template/submission.py` . Please ensure that the file you submit is named **submission.py**. Please ensure that you aren't printing anything to stdout in your submission.

Please pay attention to the type annotations to determine the data types of the data being fed to the functions and the return types expected. **Please do not round any values.** The answers in the provided sample test cases are only rounded for clarity.

Allowed Programming Languages and Libraries

This programming assignment is required to be implemented in python. Only standard built-in libraries would be usable. For instance, you would NOT be able to use libraries like NumPy or Scikit-Learn.

Useful References

Week 12 Lesson 1: Decision Tree Introduction

TA-led discussion on decision trees - https://www.coursera.org/learn/cs-412/lecture/RiJ67/the-decision-tree-algorithm

Note about the ‘Node’ Class

Students are required to construct their decision trees using the Node class. The **fit** method initializes **self.root** to a new object of the node class. Please build the tree recursively using this root node. The Autograder will use the root attribute of the Solution object in order to run various tests.

Method 1: *fit*

This method takes the train data and labels as input, and **does not return anything.** It initializes the root attribute of a Solution object to a new instance of the Node class. This root attribute will be used to check if your tree structure is correct.

We will be using **Information Gain** rather than Gain Ratio or Gini Index to construct our tree.

The tree is constructed recursively. Feel free to use other methods to implement the recursive procedure. As explained in the lectures, we iterate through all our attributes (also called features/dimensions) in our dataset. For each feature, we determine the split points. For each split point, we calculate the information gain on splitting the

dataset at that attribute and split point. Finally, we split the dataset on the best split point and repeat the same procedure on the two splits of the data.

Students are encouraged to make use of the **split_info** method for information gain calculations.

Determining the split points

Please use **Method 2** from *Decision Tree Introduction* slide 9 from the course resources to calculate candidate split points. for each attribute.

For instance, if the attribute values were **7.0**, **1.0, 2.0, 1.0, 2.0, 3.0**

We would first sort the attribute values

[1.0, 1.0, 2.0, 2.0, 3.0, 7.0]

And then calculate the midpoint between adjacent attribute values

[1.0, 1.5, 2.0, 2.5, 5.0]

The Split Condition

Please always use a binary split and a split point to split data. That is, each decision node has the form:

> If data[i][split_dim] ≤ split_point then *left node*, else *right node*

Please note the lesser than or equal to (≤) condition used to split the data.

Tie Breaking

We have to eliminate randomness and generate deterministic results. We, therefore, enforce the following rule in this assignment: In the event of ties, always choose the attribute or label with the smallest value. Namely,

- When training a DT, if splitting on either attribute X_1 or X_2 gives us the best information gain, choose the smaller of X_1 and X_2.
    - If a data point is **[1.55, 2.67, 8.95], 1.55** is the 0th attribute, **2.67** is the 1st attribute, **8.95** is the 2nd attribute (We use 0-based numbering)

- In prediction, if both labels L_1 and L_2 have the same number of training instances at a DT leaf node, predict the smaller of L_1 and L_2.

For all test cases in this assignment, we guarantee results are deterministic as long as the requirements above are satisfied.

Assigning the *node.label* Attribute

The attributes **(split_dim, split_point, label, left, right)** of every node in the decision tree are assigned during the fitting stage.

For each node in the decision tree, the label for that node is the majority label of the train datapoints of that node. The ***node.label*** attribute is assigned even for non-leaf nodes (even though only the label attributes of leaf nodes are used at prediction time).

How the method is evaluated

The Autograder will perform an inorder and preorder traversal of the constructed decision tree using the root node (**root** attribute of a Solution class instance. This is set in the first line of the fit() method) and compare the traversals against the same traversals of correctly constructed trees.

Method 2: *split_info*

This method is given datapoints, corresponding labels, and an arbitrary split dimension and split point.

The method must return the Information needed to classify the data if it were split at the provided split point and dimension.

The formula to calculate this is exactly the same as $Info_A(D)$ from *Decision Tree Introduction* slide 6 from the course resources. Please refer to the corresponding lecture video (Decision Tree Induction 1.1) for the details.

The method returns a single floating point number corresponding to the information needed.

We encourage students to call this method from the *fit()* method to make information gain calculations easier.

Method 3: *classify*

This method takes the train datapoints, train labels, and test datapoints as input. It is required to return a list of integers corresponding to the predicted classes for each test datapoint.

We encourage students first to construct the decision tree using the *fit* method (Which needs to be implemented). Then, traverse through the decision tree to predict the labels for each test data point.

Sample Test Cases

We provide a few sample test cases and their corresponding solutions under `sample_test_cases` to assist in debugging.

`sample_test_cases/classification`

Each input dataset contains training instances followed by test instances. Each line has the following space-separated format:

- [label] [attribute 0]:[value 0] [attribute 1]:[value 1]...

The name of each attribute, e.g., [attribute 2], is a non-negative integer. The value of an attribute, e.g., [value 2], is a float number. A line stands for **a test instance if [label] is -1** and a training instance otherwise. The label of a training instance can be any positive integer.

`sample_test_cases/tree_structure`

The input files have training data in the same format described above.

The output files show the preorder traversal of the resulting tree followed by the inorder traversal (https://en.wikipedia.org/wiki/Tree_traversal). Please note that the Autograder performs these travels using the root node. You are not required to implement these traversals.

`sample_test_cases/split_info`

The first line of the input is the split dimension, and the second line is the split point. Followed by the training data.

The output files have the floating point value corresponding to the information needed. These are rounded to four places for clarity. Please do not round your values.

Important Points to Note

1. Please do not print anything to stdout in your final submission.
2. The maximum depth of the decision tree is 2
3. How you calculate candidate split points is important.
4. For leaf nodes the node.split_dim = -1, node.split_point = -1.0, node.left=None and node.right=None (All attributes other than ‘label’ are unassigned).