**Decision Trees from Scratch**

A **Decision Tree** is a tree-structured classifier that uses a set of rules to predict a target variable. The tree consists of nodes where decisions are made based on feature values, leading to predictions at the leaf nodes.

**Process:**

* Splitting
* Selecting the best Split (Gini index - impurity)
* Pure node, Max depth

**Implementation:**

1. Implement a basic **Decision Tree Classifier** from scratch.
2. Use **Entropy (disorder)** and **Information Gain (variance)** to determine the best feature to split on.

Class A: 30

Class B: 20

Total: 50

PA = 30/50

PB = 20/50

Gini Index: G =

Entropy: E = -

Information Gain: IG =

import math

# Helper functions

def calculate\_entropy(data):

"""

Calculate the entropy of a dataset.

data: List of target labels

"""

total = len(data)

if total == 0:

return 0

counts = {}

for label in data:

counts[label] = counts.get(label, 0) + 1

entropy = 0

for count in counts.values():

prob = count / total

entropy -= prob \* math.log2(prob)

return entropy

def split\_data(dataset, feature\_index):

"""

Split the dataset based on a feature.

dataset: List of lists where each inner list is a data point

feature\_index: Index of the feature to split on

"""

splits = {}

for row in dataset:

key = row[feature\_index]

if key not in splits:

splits[key] = []

splits[key].append(row)

return splits

def calculate\_information\_gain(dataset, feature\_index, target\_index):

"""

Calculate the Information Gain for splitting on a specific feature.

dataset: List of lists where each inner list is a data point

feature\_index: Index of the feature to split on

target\_index: Index of the target variable

"""

total\_entropy = calculate\_entropy([row[target\_index] for row in dataset])

splits = split\_data(dataset, feature\_index)

total\_samples = len(dataset)

weighted\_entropy = 0

for subset in splits.values():

prob = len(subset) / total\_samples

subset\_entropy = calculate\_entropy([row[target\_index] for row in subset])

weighted\_entropy += prob \* subset\_entropy

information\_gain = total\_entropy - weighted\_entropy

return information\_gain

# Decision Tree Classifier

class DecisionTree:

def \_\_init\_\_(self, max\_depth=None):

self.max\_depth = max\_depth

self.tree = None

def fit(self, dataset, features, target\_index):

"""

Build the decision tree.

dataset: List of lists (rows of data)

features: List of feature names

target\_index: Index of the target variable

"""

self.tree = self.\_build\_tree(dataset, features, target\_index, depth=0)

def \_build\_tree(self, dataset, features, target\_index, depth):

# Check stopping criteria

target\_values = [row[target\_index] for row in dataset]

if len(set(target\_values)) == 1: # Pure node

return target\_values[0]

if not features or (self.max\_depth is not None and depth >= self.max\_depth): # No features or max depth

return max(set(target\_values), key=target\_values.count)

# Find the best feature to split

best\_feature\_index = -1

best\_gain = -float('inf')

for i in range(len(features)):

gain = calculate\_information\_gain(dataset, i, target\_index)

if gain > best\_gain:

best\_gain = gain

best\_feature\_index = i

if best\_gain == 0: # No further splits

return max(set(target\_values), key=target\_values.count)

# Split dataset

best\_feature = features[best\_feature\_index]

splits = split\_data(dataset, best\_feature\_index)

subtree = {}

remaining\_features = features[:best\_feature\_index] + features[best\_feature\_index + 1:]

for value, subset in splits.items():

subtree[value] = self.\_build\_tree(subset, remaining\_features, target\_index, depth + 1)

return {best\_feature: subtree}

def predict(self, row):

"""

Predict the class label for a single data point.

row: List of feature values

"""

node = self.tree

while isinstance(node, dict):

feature = list(node.keys())[0]

value = row[feature]

node = node[feature].get(value, None)

if node is None:

return None

return node

# Example Usage

dataset = [

['Sunny', 'Hot', 'High', 'No'],

['Sunny', 'Hot', 'High', 'No'],

['Overcast', 'Hot', 'High', 'Yes'],

['Rainy', 'Mild', 'High', 'Yes'],

['Rainy', 'Cool', 'Normal', 'Yes'],

['Rainy', 'Cool', 'Normal', 'No'],

['Overcast', 'Cool', 'Normal', 'Yes'],

['Sunny', 'Mild', 'High', 'No'],

['Sunny', 'Cool', 'Normal', 'Yes'],

['Rainy', 'Mild', 'Normal', 'Yes']

]

features = ['Outlook', 'Temperature', 'Humidity']

target\_index = 3

tree = DecisionTree(max\_depth=3)

tree.fit(dataset, features, target\_index)

print(tree.tree)