**Problem Statement:**

Apply CART on buy computer dataset.

**Dataset Description:**

The dataset contains 14 records and is used to predict whether a person will buy a computer based on various features:

Columns:

id: Unique identifier.

age: Age group (youth, middle\_age, senior).

income: Income level (high, medium, low).

student: Whether the person is a student (yes/no).

credit\_rating: Credit rating (fair, excellent).

Buy\_Computer: Target variable (yes/no), indicating if the person buys a computer.

Objective: Predict whether an individual will buy a computer based on age, income, student status, and credit rating.

Target: Buy\_Computer (yes/no).

**Procedure:**

Step 1: Select the best feature and threshold to split the data at each node based on Gini impurity (for classification).

Step 2: Split the data into two subsets based on the chosen feature and threshold.

Step 3: Repeat the splitting process recursively for each subset until one of the stopping conditions is met.

Step 4: Assign class labels or target values to leaf nodes.

Step 5: Prune the tree to avoid overfitting, if necessary.

Step 6: Repeat the process recursively for each subset.

Step 7: Stop when one of the stopping conditions is met (pure subsets or no attributes left to split).

**Source code:**

import numpy as np

import pandas as pd

import math

from collections import Counter

import graphviz

# Load dataset

df = pd.read\_csv('Buy\_Computer.csv')

df\_data = df.iloc[:, 1:]

# Handle missing values by filling with the mode of each column

df\_data.fillna(df\_data.mode().iloc[0], inplace=True)

# Remove duplicate rows

df\_data.drop\_duplicates(inplace=True)

# Convert categorical data into numerical format

df = df\_data.apply(lambda x: x.astype('category').cat.codes if x.dtype == 'object' else x)

# Entropy Calculation

def entropy(y):

counter = Counter(y)

total = len(y)

return -sum((count/total) \* math.log2(count/total) for count in counter.values())

# Information Gain Calculation

def information\_gain(df, feature, target):

total\_entropy = entropy(df[target])

values = df[feature].unique()

weighted\_entropy = sum((len(df[df[feature] == v]) / len(df)) \* entropy(df[df[feature] == v][target]) for v in values)

return total\_entropy - weighted\_entropy

# ID3 Algorithm - Recursively Build the Decision Tree

def id3(df, features, target, tree=None):

if len(df[target].unique()) == 1:

return df[target].iloc[0]

if len(features) == 0:

return df[target].mode()[0]

best\_feature = max(features, key=lambda f: information\_gain(df, f, target))

tree = {best\_feature: {}}

for value in df[best\_feature].unique():

subset = df[df[best\_feature] == value]

tree[best\_feature][value] = id3(subset, [f for f in features if f != best\_feature], target)

return tree

# Build the decision tree

features = df.columns[:-1]

target = 'Buy\_Computer'

decision\_tree = id3(df, features, target)

# Print Decision Tree

import pprint

pprint.pprint(decision\_tree)

# Function to visualize the tree

def visualize\_tree(tree, parent=None, graph=None):

if graph is None:

graph = graphviz.Digraph(format="png")

if isinstance(tree, dict):

for node, sub\_tree in tree.items():

if parent is None:

graph.node(node, label=node, shape="diamond")

else:

graph.node(node, label=node, shape="diamond")

graph.edge(parent, node)

for value, branch in sub\_tree.items():

child\_name = f"{node}\_{value}"

graph.node(child\_name, label=str(value))

graph.edge(node, child\_name)

visualize\_tree(branch, parent=child\_name, graph=graph)

else:

graph.node(str(tree), label=str(tree), shape="box")

graph.edge(parent, str(tree))

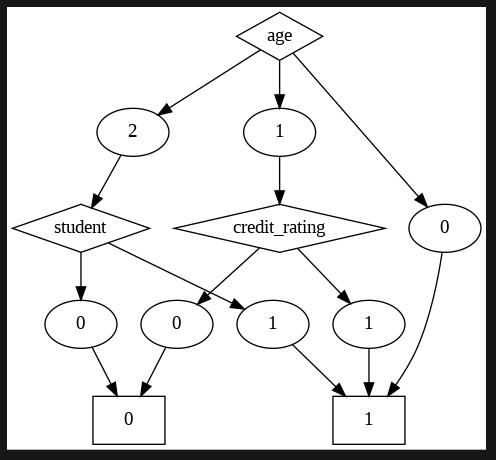
return graph

# Draw the decision tree

graph = visualize\_tree(decision\_tree)

graph.render("decision\_tree", view=True)

**Output:**



**Discussion:**

This process involves efficient preprocessing, where missing values are filled with the mode, duplicate rows are removed, and categorical features are encoded using LabelEncoder. The decision tree is both **visualized using graphviz** for a graphical representation and printed as a **text-based tree** for easier interpretation. This workflow ensures that the data is clean, the model is simple yet effective, and both graphical and textual outputs are provided for better transparency.