**Hunt's Algorithm**

One of the Decision Tree-based Methods in classification.

**Example**

|  |
| --- |
| Problem: Classification of whether a loan applicant should be **approved** or **denied** based on their **income** and **credit score**. |

**Data**



**点击图片可查看完整电子表格**

**Steps**

1. **Start at the root node with all the training data**

We have 6 applicants with attributes: Income and Credit Score, and the target variable (Decision).

2. **Check if all records belong to the same class**

If all records have the same decision (e.g., all "Approved" or all "Denied"), the current node is a leaf node, and the decision can be assigned. In this case, they are mixed (Approved and Denied), so proceed to the next step.

3. **Select the best attribute for splitting**

* Use a splitting criterion like information gain (based on entropy) or Gini index to select the best attribute.
* For simplicity, let's assume **Income** is chosen as the best attribute for splitting.

4. **Split the data based on the chosen attribute**

We now split the data into three groups: **High**, **Medium**, and **Low** based on Income.

* Group 1 (Income = High): [Applicant A, Applicant D]
* Group 2 (Income = Medium): [Applicant C, Applicant F]
* Group 3 (Income = Low): [Applicant B, Applicant E]

5. **Check if the split data is pure**

* Group 1 (Income = High): [Approved, Approved] → **Pure**, so this becomes a leaf node with the decision "Approved".
* Group 2 (Income = Medium): [Denied, Approved] → **Mixed**, so we need to split further.
* Group 3 (Income = Low): [Denied, Denied] → **Pure**, so this becomes a leaf node with the decision "Denied".

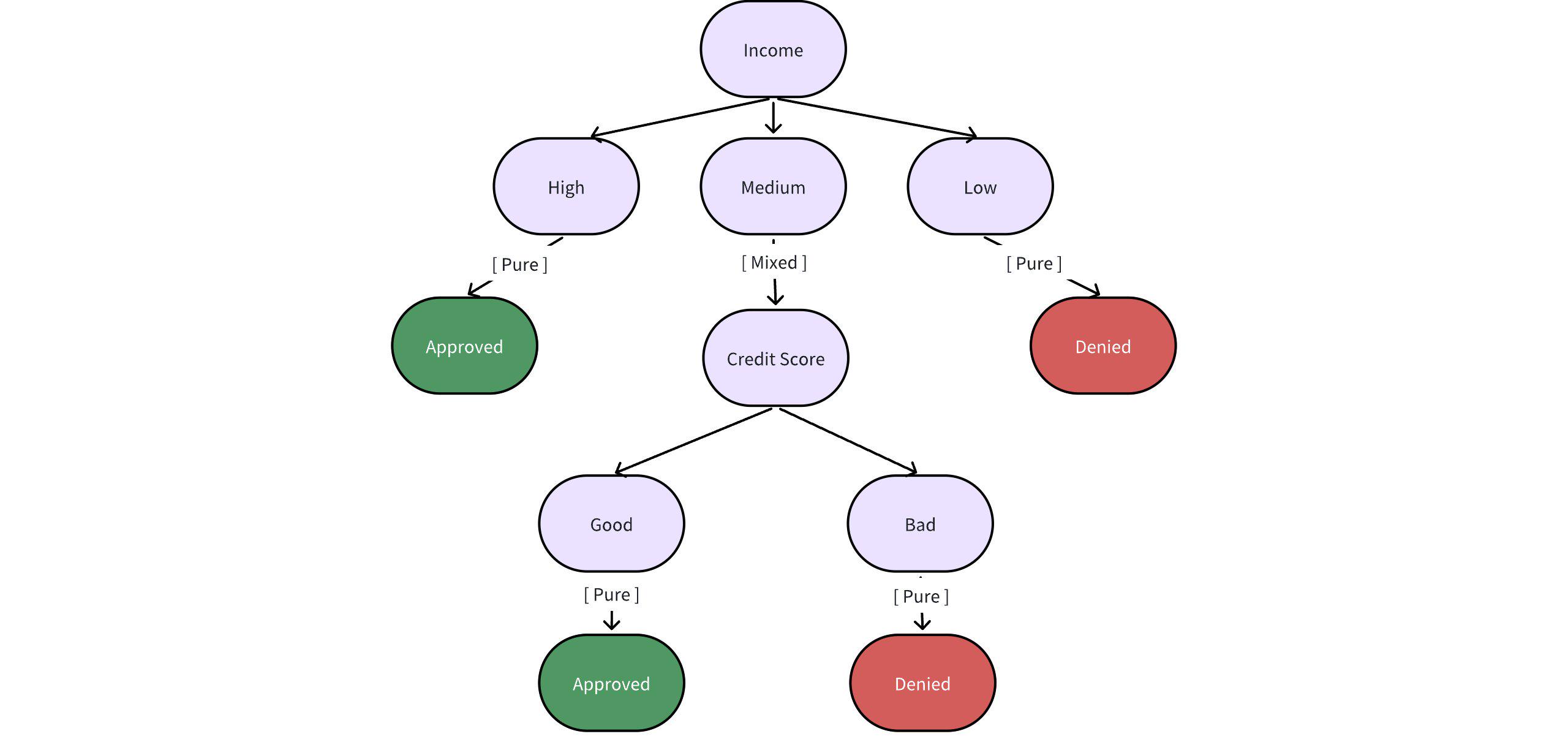
6. **Further split Group 2 (Income = Medium) using another attribute (Credit Score)**

Now we split Group 2 based on the **Credit Score** attribute.

Both subgroups are **pure**, so we can create two leaf nodes:

* **Credit Score = Good**: Decision = Approved.
* **Credit Score = Bad**: Decision = Denied.

**Result**



**Python Code**

Entropy of the data:

|  |
| --- |
| Python import math  # Helper function to calculate entropy def entropy(data):  total\_count = len(data)  if total\_count == 0:  return 0  pos\_count = len([d for d in data if d['Decision'] == 'Approved'])  neg\_count = len([d for d in data if d['Decision'] == 'Denied'])    pos\_prob = pos\_count / total\_count if pos\_count != 0 else 0  neg\_prob = neg\_count / total\_count if neg\_count != 0 else 0   if pos\_prob == 0 or neg\_prob == 0:  return 0   return -pos\_prob \* math.log2(pos\_prob) - neg\_prob \* math.log2(neg\_prob)  # Helper function to split data based on an attribute def split\_data(data, attribute):  subsets = {}  for row in data:  key = row[attribute]  if key not in subsets:  subsets[key] = []  subsets[key].append(row)  return subsets  # Helper function to calculate information gain def information\_gain(data, attribute):  total\_entropy = entropy(data)  subsets = split\_data(data, attribute)  weighted\_entropy = sum((len(subset) / len(data)) \* entropy(subset) for subset in subsets.values())  return total\_entropy - weighted\_entropy  # Recursive function to build the decision tree def build\_tree(data, attributes):  decisions = [row['Decision'] for row in data]   # Base case: If all decisions are the same, return the decision  if decisions.count(decisions[0]) == len(decisions):  return decisions[0]   # Base case: If no attributes left to split, return majority decision  if not attributes:  return max(set(decisions), key=decisions.count)   # Find the best attribute to split  gains = {attribute: information\_gain(data, attribute) for attribute in attributes}  best\_attr = max(gains, key=gains.get)   # Split data by the best attribute  subsets = split\_data(data, best\_attr)    # Create a subtree for each subset  tree = {best\_attr: {}}  remaining\_attributes = [attr for attr in attributes if attr != best\_attr]    for attr\_value, subset in subsets.items():  tree[best\_attr][attr\_value] = build\_tree(subset, remaining\_attributes)   return tree  # Example dataset data = [  {'Applicant': 'A', 'Income': 'High', 'Credit Score': 'Good', 'Decision': 'Approved'},  {'Applicant': 'B', 'Income': 'Low', 'Credit Score': 'Good', 'Decision': 'Denied'},  {'Applicant': 'C', 'Income': 'Medium', 'Credit Score': 'Bad', 'Decision': 'Denied'},  {'Applicant': 'D', 'Income': 'High', 'Credit Score': 'Bad', 'Decision': 'Approved'},  {'Applicant': 'E', 'Income': 'Low', 'Credit Score': 'Good', 'Decision': 'Denied'},  {'Applicant': 'F', 'Income': 'Medium', 'Credit Score': 'Good', 'Decision': 'Approved'} ]  # Attributes available for splitting attributes = ['Income', 'Credit Score']  # Build and print the decision tree decision\_tree = build\_tree(data, attributes) print(decision\_tree) |

**Extension**

|  |
| --- |
| How to determine the best split? |

Greedy approach: Nodes with homogeneous class distribution are preferred. (homogeneous adj. ) ---- A node with lower entropy is more **pure**, meaning the data points in that node belong mostly (or entirely) to one class.

Need a measure of node impurity. ( impurity n. 不纯度 )

A **better node** has lower entropy because it indicates that the data points are more homogeneous. In decision tree construction, we aim to reduce the impurity of nodes as much as possible by choosing attributes that provide the best splits (i.e., those that maximize information gain, which is a reduction in entropy).

Example:

Scenario 1: Splitting by **Income**

|  |
| --- |
| Python Data split by Income:  Income = High → [Approved, Approved] (Pure) Income = Medium → [Approved, Denied] (Impure) Income = Low → [Denied, Denied] (Pure) |

Scenario 2: Splitting by **Credit Score**

|  |
| --- |
| Python Data split by Credit Score:  Credit Score = Good → [Approved, Denied, Approved, Denied] (Impure) Credit Score = Bad → [Approved, Denied] (Impure) |

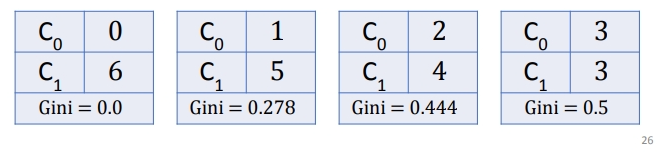
**Conclusion**

* **Scenario 1 (split by Income)** is better because it results in **two pure nodes** and **one impure node**.
* **Scenario 2 (split by Credit Score)** results in two **impure nodes**, so it’s a worse split compared to Scenario 1.

When building a decision tree, we aim to select attributes that result in the greatest reduction of impurity (or entropy), leading to more homogeneous (pure) nodes, which makes the tree more accurate in predicting the class labels.

**Measures of Node Impurity**

* Gini index
* Max value: when records in the node are equally distributed among all classes
* Min value: 0 when all records belong to one class

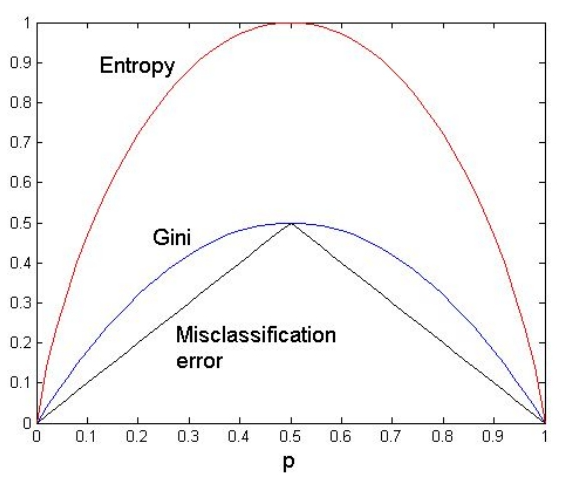


Example

1. ...

* Entropy [ Used in the above codes]
* Max value: when records are equally distributed among all classes
* Min value: 0 when all records belong to one class
* Misclassification error

Where is the relative frequency of class i at node t.



**Stop Criteria**

Stop expanding a node when all the records belong to the same class.

[ 这个条件检查数据集中所有的决策（'Approved' 或 'Denied'）是否都相同。如果是，说明所有的样本都具有相同的决策，因此我们可以直接返回这个决策值作为树的叶子节点。这是因为如果数据集中所有的决策都一致，就没有必要继续分裂数据了，直接用这个决策就可以了 ]

|  |
| --- |
| SQL if decisions.count(decisions[0]) == len(decisions):  return decisions[0] |

Stop when the rest of attributions set is empty. No attribution can be used to split further.

|  |
| --- |
| SQL if not attributes:  return max(set(decisions), key=decisions.count) |

P35