Day56_Decision_Tree_Classifier

August 2, 2025

1 Decision Tree Theory & Example

Decision Tree is a supervised machine learning algorithm used for both classification and regression. It works by splitting the dataset into smaller subsets based on the most significant attributes (features), forming a tree structure.

Each internal node tests a feature, each branch is the result of a test, and each leaf node is a class label (output).

Why Decision Tree?

- Easy to understand and visualize.
- Works for both numerical and categorical data.

Key Concepts:

Entropy:

Entropy is a measure of impurity or randomness in data. It answers: "How mixed are the values in our target column?"

- If all values are same entropy = 0 (pure)
- If values are mixed equally entropy = 1 (maximum impurity)

Formula:

$$\text{Entropy}(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$$

Where:

- p+ = proportion of positive class (e.g., 'up' in below example)
- p- = proportion of negative class (e.g., 'down')

Information Gain (IG):

Information Gain is the reduction in entropy after splitting on a feature.

$$\text{IG} = \text{Entropy(Parent)} - \text{Weighted Avg. Entropy of Children}$$

The feature with the highest IG becomes the root node.

Pruning:

Once a tree is built, **pruning** removes overfitted branches to improve generalization and avoid noise.

Gini Index:

Alternative to Entropy. It is faster to compute. Formula:

$$\text{Gini} = 1 - \sum p_j^2$$

Sklearn uses **Gini** by default because it is faster (no log function).

In ML, we build a tree using data and math (Entropy, Information Gain, Gini).

Important Question : How do we decide what becomes the root node?

Steps:

1. Identify the Dependent Variable (Target) — the output you want to predict (e.g., Profit).
2. Compute Entropy and Information Gain (IG) for all Independent Variables.
3. The feature with the highest Information Gain becomes the root node.

Root node = Feature with highest IG

Pruning: A technique used to reduce overfitting by cutting unnecessary branches. Similar to feature elimination.

Gini vs Entropy:

- Gini is default in sklearn because it's faster (no log).
- Entropy gives more information but is computationally heavier.

2 Manual Calculation

We are using a small dataset to manually understand how entropy and IG work. This dataset has four columns:

- Age: old, mid, new
- Competition: yes, no
- Type: software, hardware
- Profit (Target): up or down

Goal:

1. Calculate total entropy of the target column (Profit)
2. Calculate entropy for each feature (Age, Competition, Type)
3. Calculate Information Gain (IG) for each feature
4. Select the feature with highest IG → it becomes the root node

[1]:
```
import pandas as pd
import numpy as np
import math

# Manual dataset from notes
data = pd.DataFrame({
    'Age': ['old', 'old', 'old', 'mid', 'mid', 'mid', 'mid', 'mid', 'new',␣
 ↪'new'],
```

```
    'Competition': ['yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes', 'no',␣
  ↪'no'],
    'Type': ['s_w', 's_w', 'h_w', 's_w', 'h_w', 'h_w', 's_w', 's_w', 'h_w',␣
  ↪'s_w'],
    'Profit': ['down', 'down', 'down', 'down', 'down', 'up', 'up', 'up', 'up',␣
  ↪'up']
})

print("\n  Dataset:")
print(data)
```

```
  Dataset:
   Age Competition Type Profit
0  old         yes  s_w   down
1  old          no  s_w   down
2  old          no  h_w   down
3  mid         yes  s_w   down
4  mid         yes  h_w   down
5  mid          no  h_w     up
6  mid          no  s_w     up
7  mid         yes  s_w     up
8  new          no  h_w     up
9  new          no  s_w     up
```

2.1 Total Entropy of 'Profit'

Count how many 'up' and 'down' values in the target column (Profit)

[2]:
```
up = len(data[data['Profit'] == 'up'])    # 5
down = len(data[data['Profit'] == 'down'])  # 5
total = up + down
p_up = up / total
p_down = down / total
entropy_total = - (p_up * math.log2(p_up) + p_down * math.log2(p_down))
print("\n Entropy(Profit):", round(entropy_total, 3))
```

```
 Entropy(Profit): 1.0
```

2.2 Entropy(Age)

- Age=old → 3 down, 0 up Entropy = 0
- Age=mid → 2 down, 2 up Entropy = 1
- Age=new → 0 down, 2 up Entropy = 0

[3]:
```
entropy_age = (3/10)*0 + (4/10)*1 + (2/10)*0
IG_age = entropy_total - entropy_age
print("Entropy(Age):", round(entropy_age, 3), "→ IG(Age):", round(IG_age, 3))
```

```
Entropy(Age): 0.4 → IG(Age): 0.6
```

2.3 Entropy(Competition)

Competition=yes → 4 samples → 3 down, 1 up

[4]:
```
py_down = 3/4
py_up = 1/4
entropy_yes = - (py_down * math.log2(py_down) + py_up * math.log2(py_up))
```

Competition=no → 6 samples → 2 down, 4 up

[5]:
```
pn_down = 2/6
pn_up = 4/6
entropy_no = - (pn_down * math.log2(pn_down) + pn_up * math.log2(pn_up))
```

[6]:
```
entropy_comp = (4/10)*entropy_yes + (6/10)*entropy_no
IG_comp = entropy_total - entropy_comp
print("Entropy(Competition):", round(entropy_comp, 3), "→ IG(Competition):",␣
↪round(IG_comp, 3))
```

```
Entropy(Competition): 0.875 → IG(Competition): 0.125
```

3 Entropy(Type)

- s_w → 6 → 3 up, 3 down → entropy = 1
- h_w → 4 → 2 up, 2 down → entropy = 1

[7]:
```
entropy_type = (6/10)*1 + (4/10)*1
IG_type = entropy_total - entropy_type
print("Entropy(Type):", round(entropy_type, 3), "→ IG(Type):", round(IG_type,␣
↪3))
```

```
Entropy(Type): 1.0 → IG(Type): 0.0
```

3.1 Summary

[8]:
```
print("\n Information Gain Summary:")
print("IG(Age):", round(IG_age, 3))
print("IG(Competition):", round(IG_comp, 3))
print("IG(Type):", round(IG_type, 3))
```

```

 Information Gain Summary:
IG(Age): 0.6
IG(Competition): 0.125
IG(Type): 0.0
```

3.2 Visual Tree Explanation

[9]:
```
print("""
 Final Tree Split:
Root Node: Age (Highest IG = 0.6)

Age = old → Profit = down (pure)
Age = new → Profit = up (pure)
Age = mid → Need further split (best next: Competition)
""")
```

```
 Final Tree Split:
Root Node: Age (Highest IG = 0.6)

Age = old → Profit = down (pure)
Age = new → Profit = up (pure)
Age = mid → Need further split (best next: Competition)
```

```
Tree:

        Age
     /   |   \
   Old  Mid  New
   ↓     ↓     ↓
 Down  Comp    Up
        / \
      Y    N
     ↓     ↓
   Down    Up

Gain(Age) = 0.6
Gain(Competition) = 0.124
Gain(Type) = 0
→ Age becomes the root node
```

3.3 Gini vs Entropy Markdown

from IPython.display import Markdown as md

Gini Index vs Entropy

Criteria	Formula	Explanation
Entropy	$(-\sum p_j \log_2(p_j))$	Measures disorder, uses log
Gini Index	$(1 - \sum p_j^2)$	Measures impurity, faster

- Sklearn uses **Gini** by default.

- Use `criterion='entropy'` for entropy-based trees.

4 Practical Code

4.1 Imported Libraries

[10]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix
import warnings
warnings.filterwarnings('ignore')
```

4.2 Load Dataset

[11]:
```
print("\n Loaded Dataset:")
dataset = pd.read_csv(r"C:\Users\Lenovo\Downloads\logit classification.csv")
print(dataset.head())
```

```
 Loaded Dataset:
    User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0
```

4.3 Select Features and Target

[12]:
```
X = dataset[["Age", "EstimatedSalary"]].values
y = dataset["Purchased"].values
```

4.4 Train-Test Split

[13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
  ↪random_state=0)
```

4.5 Without Scaling

[14]:
```
tree_model_plain = DecisionTreeClassifier()
tree_model_plain.fit(X_train, y_train)
```

[14]: DecisionTreeClassifier()

[15]:
```
y_pred_plain = tree_model_plain.predict(X_test)
print("\n Decision Tree WITHOUT SCALING")
print("Accuracy:", accuracy_score(y_test, y_pred_plain))
```

```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_plain))
```

```
 Decision Tree WITHOUT SCALING
Accuracy: 0.91
Confusion Matrix:
 [[62  6]
 [ 3 29]]
```

4.6 With Scaling

[16]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[17]:
```
tree_model_scaled = DecisionTreeClassifier()
tree_model_scaled.fit(X_train_scaled, y_train)
```

[17]:
```
DecisionTreeClassifier()
```

[18]:
```
y_pred_scaled = tree_model_scaled.predict(X_test_scaled)
print("\n Decision Tree WITH SCALING")
print("Accuracy:", accuracy_score(y_test, y_pred_scaled))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_scaled))
```

```
 Decision Tree WITH SCALING
Accuracy: 0.9
Confusion Matrix:
 [[62  6]
 [ 4 28]]
```

4.7 With max_depth (1, 2, 3) Without Scaling

[19]:
```
for depth in [1, 2, 3]:
    model = DecisionTreeClassifier(max_depth=depth)
    model.fit(X_train, y_train)
```

[20]:
```
y_pred = model.predict(X_test)
print(f"\n Decision Tree (max_depth={depth}) WITHOUT SCALING")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
 Decision Tree (max_depth=3) WITHOUT SCALING
Accuracy: 0.94
Confusion Matrix:
 [[64  4]
 [ 2 30]]
```

4.8 With max_depth (1, 2, 3) With Scaling

[21]:
```
for depth in [1, 2, 3]:
    model = DecisionTreeClassifier(max_depth=depth)
    model.fit(X_train_scaled, y_train)
```

[22]:
```
y_pred = model.predict(X_test_scaled)
print(f"\n Decision Tree (max_depth={depth}) WITH SCALING")
print("Accuracy:", accuracy_score(y_test, y_pred_scaled))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_scaled))
```

```
 Decision Tree (max_depth=3) WITH SCALING
Accuracy: 0.9
Confusion Matrix:
 [[62  6]
 [ 4 28]]
```

5 Functional Version (LOOPED) Advanced

[23]:
```
def train_and_evaluate_decision_tree(X_train, X_test, y_train, y_test,␣
 ↪scaled=False):
    results = []
    for depth in [0, 1, 2, 3]:
        if depth == 0:
            clf = DecisionTreeClassifier()
        else:
            clf = DecisionTreeClassifier(max_depth=depth)
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        cm = confusion_matrix(y_test, y_pred)
        results.append({
            'Scaled': scaled,
            'max_depth': depth,
            'Accuracy': acc,
            'Confusion_Matrix': cm
        })
    return results
```

5.1 Run both scaled and unscaled

[24]:
```
results_unscaled = train_and_evaluate_decision_tree(X_train, X_test, y_train,␣
 ↪y_test, scaled=False)
results_scaled = train_and_evaluate_decision_tree(X_train_scaled,␣
 ↪X_test_scaled, y_train, y_test, scaled=True)
```

5.2 Combine and Display Results

[25]:
```
final_results = pd.DataFrame(results_unscaled + results_scaled)
print("\n Final Comparison Table:")
print(final_results)
```

```
 Final Comparison Table:
   Scaled  max_depth  Accuracy    Confusion_Matrix
0   False          0      0.91  [[62, 6], [3, 29]]
1   False          1      0.89  [[66, 2], [9, 23]]
2   False          2      0.94  [[64, 4], [2, 30]]
3   False          3      0.94  [[64, 4], [2, 30]]
4    True          0      0.90  [[62, 6], [4, 28]]
5    True          1      0.89  [[66, 2], [9, 23]]
6    True          2      0.94  [[64, 4], [2, 30]]
7    True          3      0.94  [[64, 4], [2, 30]]
```

5.3 Save to CSV

[26]:
```
final_results.to_csv("decision_tree_comparison.csv", index=False)
print("\n Results saved to 'decision_tree_comparison.csv'")
```

```
 Results saved to 'decision_tree_comparison.csv'
```

Analysis:

- All models with max_depth=2 or 3 gave the highest accuracy: **94%**.
- Models with depth=0 (no depth limit) performed similarly to max_depth=2 and 3.
- Scaling had **no impact** on decision tree accuracy here — performance was consistent with and without scaling.
- Models with max_depth=1 slightly underperformed (accuracy 89%), suggesting insufficient depth to capture splits.

Best Model:

- Any model with `max_depth=2 or 3`, whether scaled or not.
- Confusion Matrix: [[64, 4], [2, 30]] → Very few misclassifications.
- Balanced performance across both classes (Purchased = 0 or 1).