Technical Report: Decision Trees and Their Main Algorithms

1. Introduction

Decision Trees are one of the most popular and interpretable machine learning algorithms. They work by splitting the dataset into branches based on feature values until a decision or prediction is made. Trees are widely used in both classification and regression problems due to their simplicity, transparency, and effectiveness.

---

2. Main Types of Decision Trees

Decision Trees can be broadly categorized depending on the type of target variable:

2.1 Classification Trees

- Target variable: **categorical** (e.g., Yes/No, Class A/Class B).
- Splitting criteria: **Gini Index**, **Entropy & Information Gain**.
- Example: Predicting whether a customer will buy a product.

2.2 Regression Trees

- Target variable: **continuous** (numeric values).
- Splitting criteria: **Variance Reduction**, **Mean Squared Error (MSE)**.
- Example: Predicting the price of a house based on size and location.

---

3. Key Concepts in Decision Trees

- **Entropy**: Measure of impurity or disorder in a dataset.
- **Information Gain**: Reduction in entropy after a split.
- **Gini Index**: Alternative impurity measure, faster to compute.
- **Pruning**: Removing unnecessary branches to avoid overfitting.
- **Overfitting**: When the tree memorizes the training data but performs poorly on unseen data.

---

4. Main Decision Tree Algorithms

4.1 ID3 (Iterative Dichotomiser 3)

- Introduced by Ross Quinlan.
- Uses **Entropy** and **Information Gain** to select the best split.
- Works best with categorical features.
- Does not support pruning → trees can grow too large.

4.2 C4.5 (and C5.0)

- Successor to ID3 (also by Quinlan).
- Supports both **categorical and continuous** features.
- Uses **Gain Ratio** instead of pure Information Gain.
- Includes **pruning** to reduce overfitting.
- C5.0 is an improved, more efficient version.

4.3 CART (Classification and Regression Trees)

- Developed by Breiman et al.
- Can handle **classification and regression** tasks.
- Uses **Gini Index** for classification and **MSE** for regression.
- Always creates **binary splits**.
- Basis for the implementation in **scikit-learn**.

5. Applications of Decision Trees

- Credit risk assessment (loan approval).
- Medical diagnosis.
- Fraud detection.
- Customer churn prediction.
- Price prediction (regression).

Algorithm	Splitting Measure	Features Supported	Output Type	Pruning	Notes
ID3	Information Gain (Entropy)	Categorical only	Classification	No	Early algorithm, tends to overfit
C4.5	Gain Ratio	Categorical + Continuous	Classification	Yes	More robust than ID3
CART	Gini Index (classification), MSE (regression)	Both	Classification & Regression	Yes	Default in most libraries

6.1 ID3 (Iterative Dichotomiser 3)

Entropy & Information Gain:

Definition

- **Entropy measures the degree of uncertainty or impurity in a dataset.**
- **If all samples belong to the same class → Entropy = 0 (pure set).**
- **If the samples are equally distributed across classes → Entropy = 1 (maximum disorder).**

Formula:

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

Where:

- **pi = probability of class i in the dataset**
- **c = number of classes**

example:

1) PlayTennis (with Outlook, Temperature, Humidity, Wind)

#	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Totals: Yes = 9, No = 5

2) Entropy of the whole set S

$$Entropy(S) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} \approx 0.9403$$

(This matches the usual PlayTennis example result ≈ 0.94.)

3) Information Gain calculations (summary)

For each attribute we compute:

- **entropies of each attribute value group,**
- **weighted average entropy after split,**
- **Information Gain = Entropy(S) − weighted_entropy.**

I computed precise values (numbers rounded for clarity):

A. Outlook

Groups:

- **Sunny: 5 samples → [No, No, No, Yes, Yes] → entropy ≈ 0.9710**
- **Overcast: 4 samples → [Yes, Yes, Yes, Yes] → entropy = 0.0**
- **Rain: 5 samples → [Yes, Yes, No, Yes, No] → entropy ≈ 0.97095**

Weighted entropy after split:

$$\frac{5}{14} \cdot 0.9710 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.9710 \approx 0.6935$$

Information Gain:

$$IG(\text{Outlook}) = 0.9403 - 0.6935 \approx \mathbf{0.2467}$$

B. Humidity

Groups:

- **High: 7 samples → [No, No, Yes, Yes, No, Yes, No] → entropy ≈ 0.9852**
- **Normal: 7 samples → [Yes, No, Yes, Yes, Yes, Yes, Yes] → entropy ≈ 0.5917**

Weighted entropy:

$$\frac{7}{14} \cdot 0.9852 + \frac{7}{14} \cdot 0.5917 \approx 0.7884$$

Information Gain:

$$IG(\text{Humidity}) = 0.9403 - 0.7884 \approx \mathbf{0.1518}$$

C. Wind

Groups:

- **Weak: 8 samples → entropy ≈ 0.8113**
- **Strong: 6 samples → entropy ≈ 0.9183**

Weighted entropy:

$$\frac{8}{14} \cdot 0.8113 + \frac{6}{14} \cdot 0.9183 \approx 0.8922$$

Information Gain:

$$IG(\text{Wind}) = 0.9403 - 0.8922 \approx \mathbf{0.0481}$$

D. Temperature

Groups:

- **Hot: 4 samples → entropy ≈ 1.0**
- **Mild: 6 samples → entropy ≈ 0.9183**
- **Cool: 4 samples → entropy ≈ 0.8113**

Weighted entropy:

$$\frac{4}{14} \cdot 1 + \frac{6}{14} \cdot 0.9183 + \frac{4}{14} \cdot 0.8113 \approx 0.9111$$

Information Gain:

$$IG(\text{Temperature}) = 0.9403 - 0.9111 \approx \mathbf{0.0292}$$

4) Interpretation & recommendation

- **Outlook has the highest Information Gain ≈ 0.2467 → best first split (consistent with ID3 textbook).**
- **Humidity is second best (≈ 0.1518).**
- **Wind and Temperature give smaller gains (≈ 0.0481 and 0.0292).**
- **So an ID3-style algorithm would pick Outlook as the root attribute.**

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
# Build dataset
data = {
    "Outlook": ["Sunny","Sunny","Overcast","Rain","Rain","Rain","Overcast",
                "Sunny","Sunny","Rain","Sunny","Overcast","Overcast","Rain"],
    "Temperature": ["Hot","Hot","Hot","Mild","Cool","Cool","Cool",
                    "Mild","Cool","Mild","Mild","Mild","Hot","Mild"],
    "Humidity": ["High","High","High","High","Normal","Normal","Normal",
                 "High","Normal","Normal","Normal","High","Normal","High"],
    "Wind": ["Weak","Strong","Weak","Weak","Weak","Strong","Strong",
             "Weak","Weak","Weak","Strong","Strong","Weak","Strong"],
    "PlayTennis": ["No","No","Yes","Yes","Yes","No","Yes",
                   "No","Yes","Yes","Yes","Yes","Yes","No"]
}
df = pd.DataFrame(data)
```

✓ 15.1s

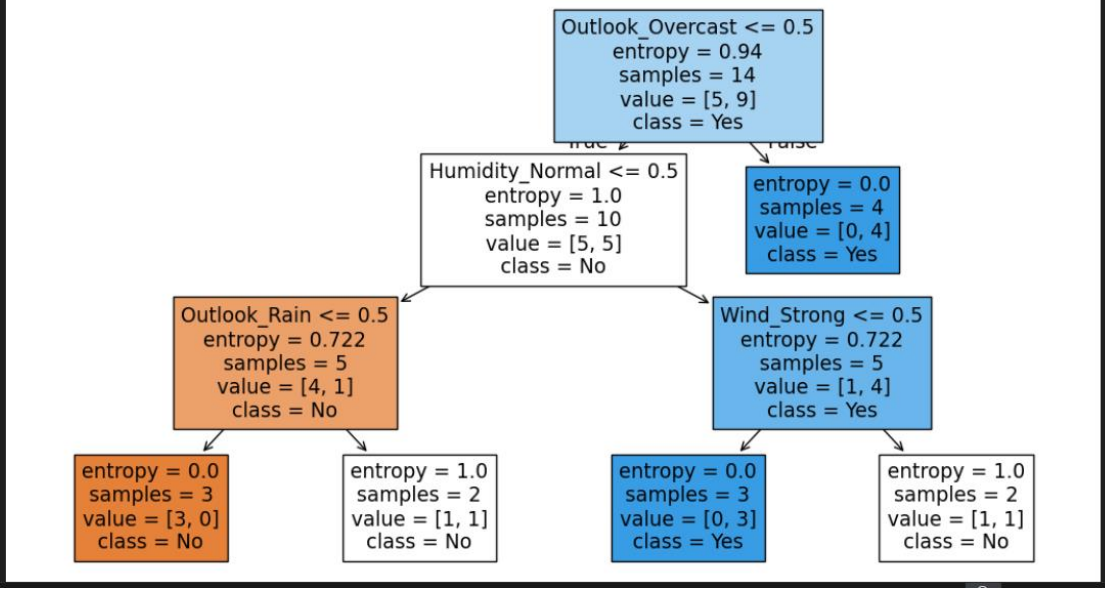
```python
df.head(14)
```

✓ 0.0s

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes

```python
# Features & target
X = df.drop("PlayTennis", axis=1)
y = df["PlayTennis"]
# Convert categorical features to numeric
X = pd.get_dummies(X)
# Train Decision Tree (use entropy like ID3)
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3, random_state=42)
clf.fit(X, y)
# Plot the decision tree
plt.figure(figsize=(13,7))
plot_tree(clf, feature_names=X.columns, class_names=clf.classes_, filled=True)
plt.show()
```

✓ 0.2s



Q) is it a must to convert it first to numerical form ??

Most machine learning algorithms work only with numerical data.
That means whenever we have categorical (non-numerical) features, we must encode them into numbers first.

Why?

- **Algorithms like Decision Trees, Random Forests, Logistic Regression, SVM, Neural Networks, etc. rely on mathematical calculations (entropy, gini, distances, dot products, gradients...).**
- **These calculations only make sense on numbers, not strings like "Sunny" or "Rain".**

Encoding Options

1. **Ordinal Encoding → assign integers (0,1,2…)**
    - **Best when categories have a natural order (e.g., Low < Medium < High).**
    - **Risky if there's no real order, since the model may think "Rain" > "Sunny".**
2. **One-Hot Encoding → binary columns for each category**
    - **Best for unordered categories.**
    - **Example: "Red, Green, Blue" → [1,0,0], [0,1,0], [0,0,1].**
3. **Label Encoding (like Ordinal but usually for target variable y)**
    - **For example: Yes=1, No=0.**
4. **Other advanced methods (for large categories, e.g., text)**
    - **Target encoding, Hash encoding, Embeddings (in Deep Learning).**

Special Case: Decision Trees

- **Decision Trees can technically handle categories directly (some implementations like CART in R do).**
- **But in scikit-learn (Python), we *must* convert them to numbers first → hence we use OrdinalEncoder or OneHotEncoder.**

✅ **So yes, for all datasets, categorical → numerical is required.**
But the method of encoding depends on the algorithm and data type.

(what I used in the last code)

1. One-Hot Encoding

- **Creates a new column for each category.**
- **Example: Outlook = Sunny / Overcast / Rain becomes 3 columns:**
- **Each row has a 1 in the column of its value, 0 otherwise.**
  Best when categories are nominal (no order).

2. Ordinal Encoding

- **Assigns an integer value to each category.**
- **Example:**

  Outlook: Sunny=0, Overcast=1, Rain=2

  Temperature: Hot=0, Mild=1, Cool=2

- **Simpler, keeps features in a single column.**
  => Downside: the model may interpret the numbers as having order/scale, which might not make sense (e.g., Rain > Sunny).

```
from sklearn.preprocessing import LabelEncoder , OneHotEncoder , OrdinalEncoder

outlook_encoder = OrdinalEncoder()
data['Outlook'] = outlook_encoder.fit_transform(data[['Outlook']])

tempreture_encoder = OrdinalEncoder()
data['Temperature'] = tempreture_encoder.fit_transform(data[['Temperature']])

humidity_encoder = OrdinalEncoder()
data['Humidity'] = humidity_encoder.fit_transform(data[['Humidity']])

windy_encoder = OrdinalEncoder()
data['Wind'] = windy_encoder.fit_transform(data[['Wind']])

playing_encoder = OrdinalEncoder()
data['Play Tennis'] = playing_encoder.fit_transform(data[['Play Tennis']])
```

```
data.head()
```

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	2.0	1.0	0.0	1.0	0.0
1	2.0	1.0	0.0	0.0	0.0
2	0.0	1.0	0.0	1.0	1.0
3	1.0	2.0	0.0	1.0	1.0
4	1.0	0.0	1.0	1.0	1.0

6.2) CART (Classification and Regression Trees)

What is CART?

- **Proposed by Breiman et al. (1986).**
- **A popular Decision Tree algorithm.**
- **Used for both:**
    - **Classification (Yes/No, Spam/Not Spam, etc.)**
    - **Regression (predicting a continuous value, e.g., house price).**

How CART works

1. **Splitting Criterion**
    - **For Classification Trees → uses Gini Index (not entropy).**
    - **For Regression Trees → uses Mean Squared Error (MSE).**
2. **Binary Splits Only**
    - **Unlike ID3/C4.5, which can split into multiple branches (e.g., Outlook → Sunny, Rain, Overcast),**
    - **CART always makes binary splits:**
        - **Example: Humidity <= 75 → left branch, right branch.**
3. **Tree Growth**
    - **Keep splitting until stopping conditions (min samples, max depth, or pure leaf).**
4. **Pruning**
    - **To avoid overfitting, CART can prune branches (cost complexity pruning).**

What is the Gini Index?

The Gini Index (or Gini Impurity) is a measure of how often a randomly chosen element from the dataset would be incorrectly labeled if it was randomly labeled according to the distribution of labels in a node.

- **It is used by the CART algorithm to decide the best split at each node.**
- **Unlike entropy (used in ID3/C4.5), Gini is simpler and faster to compute.**

  Formula:

$$Gini(D) = 1 - \sum_{i=1}^{k} p_i^2$$

Where:

- p_i = proportion (probability) of class i in the dataset D.
- k = number of classes.

Intuition

- If all samples in a node belong to one class → $\mathrm{pi}=1 p_i=1$, all others = 0 → **Gini = 0** (pure node).
- If classes are equally mixed → impurity is higher (closer to 0.5 or more).

Example: PlayTennis (Outlook=Sunny)

Suppose in our dataset, when Outlook = Sunny:

- 2 samples → PlayTennis = Yes
- 3 samples → PlayTennis = No

So:

$$p(Yes) = \frac{2}{5}, \quad p(No) = \frac{3}{5}$$

Gini:

$$Gini = 1 - \left(\frac{2}{5}\right)^2 - \left(\frac{3}{5}\right)^2$$

$$= 1 - \frac{4}{25} - \frac{9}{25}$$

$$= 1 - \frac{13}{25} = \frac{12}{25} = 0.48$$

So impurity = **0.48**.

How CART Uses Gini

1. At each split, CART calculates the **weighted Gini Index** of possible splits.

$$Gini_{split} = \sum_{j=1}^{m} \frac{|D_j|}{|D|} \cdot Gini(D_j)$$

   - D_j: subset after split
   - $|D_j|$: size of subset
2. The split with the **lowest Gini Index** (highest purity) is chosen.

Feature: Outlook

- Sunny (5: 2 Yes, 3 No) → Gini = 0.48
- Overcast (4: 4 Yes, 0 No) → Gini = 0
- Rain (5: 3 Yes, 2 No) → Gini = 0.48

Weighted:

$$(5/14)(0.48) + (4/14)(0) + (5/14)(0.48) = 0.343$$

Gain = 0.459 – 0.343 = **0.116**

Feature: Temperature

- Hot (4: 2 Yes, 2 No) → Gini = 0.5
- Mild (6: 4 Yes, 2 No) → Gini = 0.444
- Cool (4: 3 Yes, 1 No) → Gini = 0.375

Weighted:

$$(4/14)(0.5) + (6/14)(0.444) + (4/14)(0.375) = 0.440$$

Gain = 0.459 – 0.440 = **0.019**

Feature: Humidity

- High (7: 3 Yes, 4 No) → Gini = 0.489
- Normal (7: 6 Yes, 1 No) → Gini = 0.245

Weighted:

$$(7/14)(0.489) + (7/14)(0.245) = 0.367$$

Gain = 0.459 – 0.367 = **0.092**

Feature: Wind

- Weak (8: 6 Yes, 2 No) → Gini = 0.375
- Strong (6: 3 Yes, 3 No) → Gini = 0.5

Weighted:

$$(8/14)(0.375) + (6/14)(0.5) = 0.429$$

Gain = 0.459 – 0.429 = **0.030**

FEATURE	WEIGHTED GINI	GAIN
OUTLOOK	0.343	**0.116**
TEMPERATURE	0.440	0.019
HUMIDITY	0.367	0.092
WIND	0.429	0.030

->The best feature to split first is Outlook (highest gain = 0.116).

->That’s why in both ID3 (Entropy) and CART (Gini), the root node is Outlook in the PlayTennis dataset.

1) Root (we already chose)

Root = `Outlook` because it gave the largest Gini gain among all features.

Root Gini:

$$\text{Gini(root)} = 1 - (9/14)2 - (5/14)2 \approx 0.459 \quad Gini(root) = 1 - (9/14)_2 - (5/14)_2 \approx 0.459$$

Splitting by `Outlook` produced these child groups:

- **Sunny** (5 samples): Yes=2, No=3 → Gini ≈ 0.48
- **Overcast** (4 samples): Yes=4, No=0 → Gini = 0 (pure)
- **Rain** (5 samples): Yes=3, No=2 → Gini ≈ 0.48

`Overcast` is already pure → leaf **PlayTennis = Yes**.
We must further split **Sunny** and **Rain**.

---

2) Complete the Sunny branch (5 samples)

Sunny rows (Play):

- (Sunny, Hot, High, Weak) → No
- (Sunny, Hot, High, Strong) → No
- (Sunny, Mild, High, Weak) → No
- (Sunny, Cool, Normal, Weak) → Yes
- (Sunny, Mild, Normal, Strong)→ Yes

Counts: Yes = 2, No = 3.

We test remaining features (`Temperature`, `Humidity`, `Wind`) for best split (compute weighted Gini for each):

A. Split by Humidity

- `High` subset: 3 samples → No, No, No → Yes=0, No=3 → Gini = 0 (pure)
- `Normal` subset: 2 samples → Yes, Yes → Yes=2, No=0 → Gini = 0 (pure)

Weighted Gini = (3/5)*0 + (2/5)*0 = **0.0**

Perfect split — **Humidity** is ideal for Sunny.

So Sunny → split on **Humidity**:

- `Humidity = High` → leaf **PlayTennis = No**
- `Humidity = Normal` → leaf **PlayTennis = Yes**

(No need to check other splits once we have a perfect split.)

---

3) Complete the Rain branch (5 samples)

Rain rows (Play):

- (Rain, Mild, High, Weak) → Yes
- (Rain, Cool, Normal, Weak) → Yes
- (Rain, Cool, Normal, Strong)→ No
- (Rain, Mild, Normal, Weak) → Yes
- (Rain, Mild, High, Strong) → No

Counts: Yes = 3, No = 2.

Test splits:

A. Split by Wind

- `Weak` subset: 3 samples → Yes, Yes, Yes → Gini = 0 (pure)
- `Strong` subset: 2 samples → No, No → Gini = 0 (pure)

Weighted Gini = (3/5)*0 + (2/5)*0 = **0.0**

Perfect split — **Wind** is ideal for Rain.

So Rain → split on **Wind**:

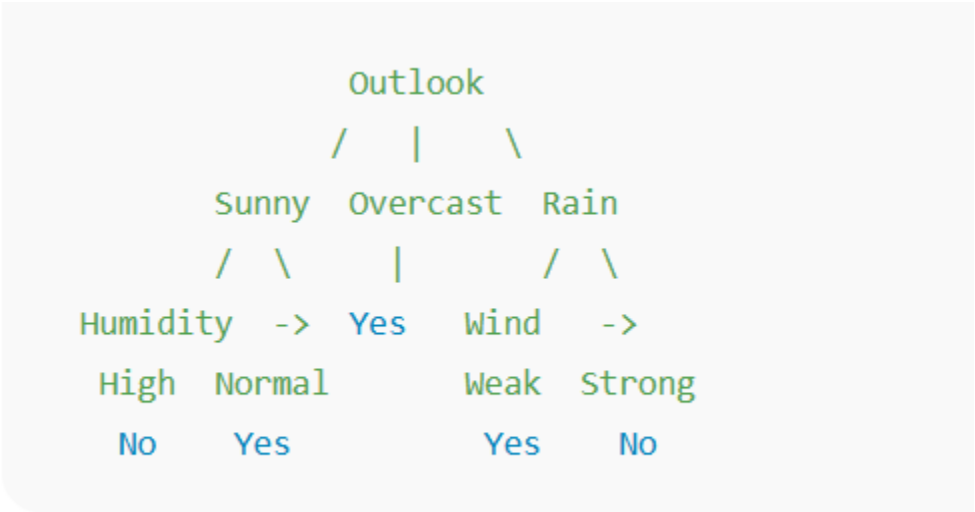
- `Wind = Weak` → leaf **PlayTennis = Yes**
- `Wind = Strong` → leaf **PlayTennis = No**

---

4) Final tree (text + ASCII)

Root: `Outlook`

- `Outlook = Overcast` → **Yes** (leaf)
- `Outlook = Sunny` → check `Humidity`
  - `Humidity = High` → **No** (leaf)
  - `Humidity = Normal` → **Yes** (leaf)
- `Outlook = Rain` → check `Wind`
  - `Wind = Weak` → **Yes** (leaf)
  - `Wind = Strong` → **No** (leaf)



5) Why stop here?

- Each leaf reached is **pure** (all samples in the leaf have the same class).
- Stopping criteria satisfied: pure leaf (no further split needed).
- This is the classic PlayTennis decision tree (same final structure produced by ID3/CART).

```python
X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']

encoder = OrdinalEncoder()
X_encoded = encoder.fit_transform(X)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
# ----------------------------
# 3. Build CART model (Gini)
# ----------------------------
clf = DecisionTreeClassifier(criterion="gini", random_state
clf.fit(X_encoded, y_encoded)
# ----------------------------
# 4. Visualize the tree
# ----------------------------
plt.figure(figsize=(12,6))
plot_tree(clf, feature_names=X.columns,
          class_names=label_encoder.classes_,
          filled=True, rounded=True)
plt.show()
```

[1] ✓ 3.9s Python