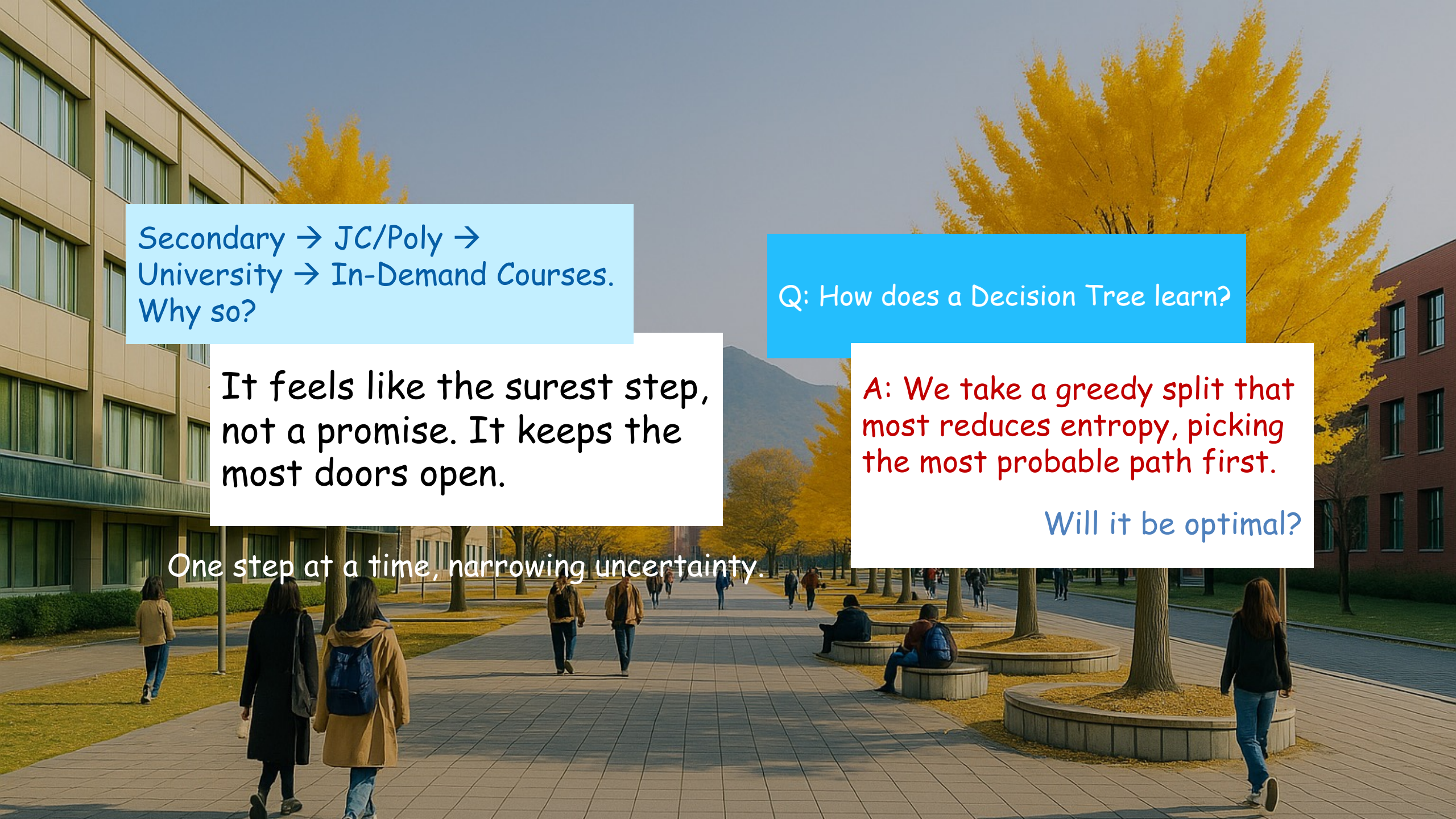
# Machine Learning

## Decision Tree Classification

Tarapong Sreenuch

8 February 2024

克明峻德，格物致知


Secondary → JC/Poly → University → In-Demand Courses. Why so?
It feels like the surest step, not a promise. It keeps the most doors open.
One step at a time, narrowing uncertainty.
Q: How does a Decision Tree learn?
A: We take a greedy split that most reduces entropy, picking the most probable path first.
Will it be optimal?

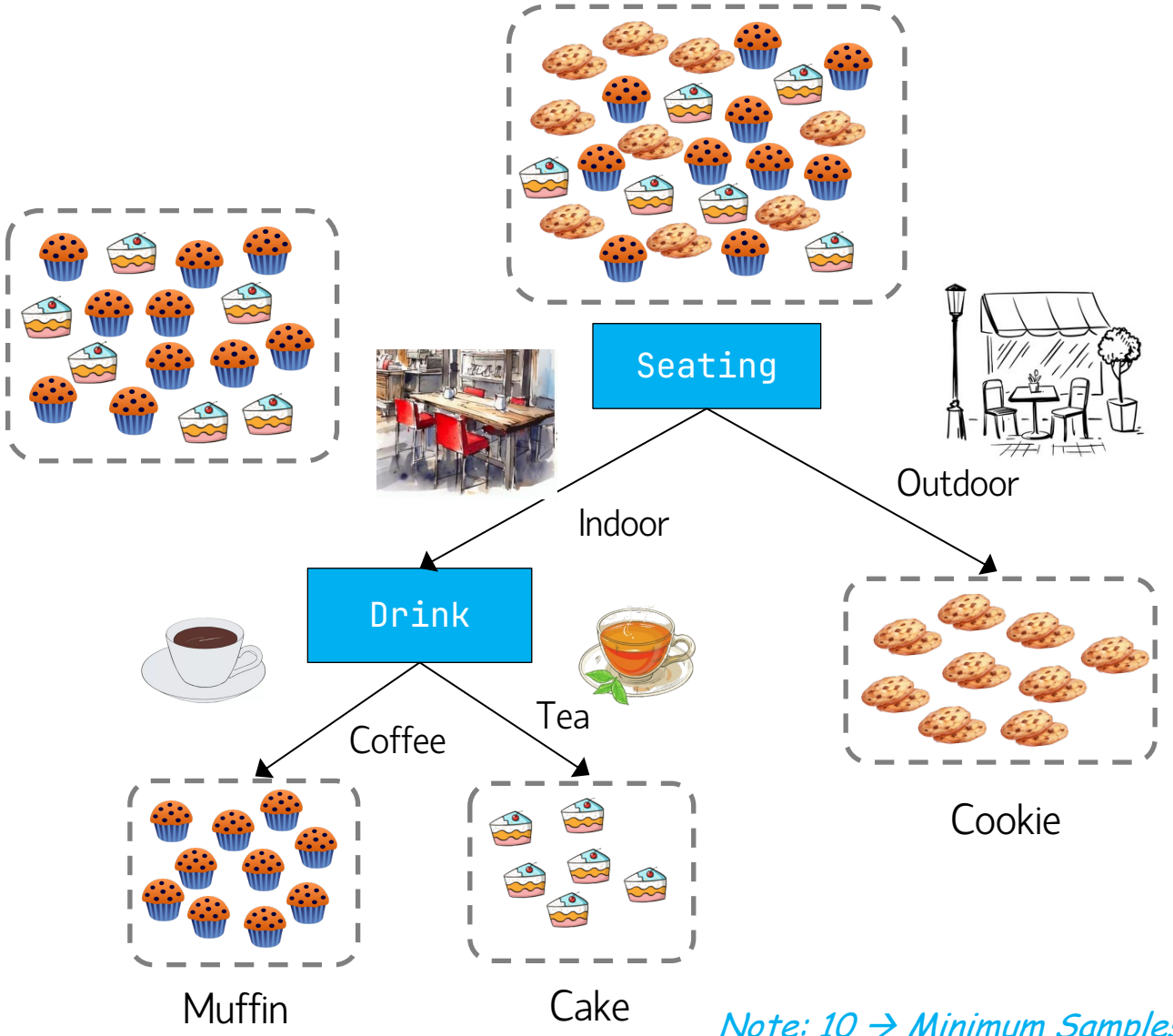
# Café Example Dataset

| Seating | Drink | Dessert |
|---|---|---|
| Outdoor | Coffee | Cookie |
| Outdoor | Coffee | Cookie |
| Outdoor | Coffee | Cookie |
| Outdoor | Coffee | Cookie |
| Outdoor | Coffee | Cookie |
| Outdoor | Coffee | Cookie |
| Outdoor | Tea | Cookie |
| Outdoor | Tea | Cookie |
| Outdoor | Tea | Cookie |
| Indoor | Coffee | Muffin |
| Indoor | Coffee | Muffin |
| Indoor | Coffee | Muffin |
| Indoor | Coffee | Muffin |

| Seating | Drink | Dessert |
|---|---|---|
| Indoor | Coffee | Muffin |
| Indoor | Coffee | Muffin |
| Indoor | Coffee | Muffin |
| Indoor | Coffee | Muffin |
| Indoor | Coffee | Muffin |
| Indoor | Coffee | Muffin |
| Indoor | Tea | Cake |
| Indoor | Tea | Cake |
| Indoor | Tea | Cake |
| Indoor | Tea | Cake |
| Indoor | Tea | Cake |
| Indoor | Tea | Cake |
| Indoor | Tea | Cake |

*Cross-Selling: Identify natural add-ons and present them at the right moment.*

# Decision Tree



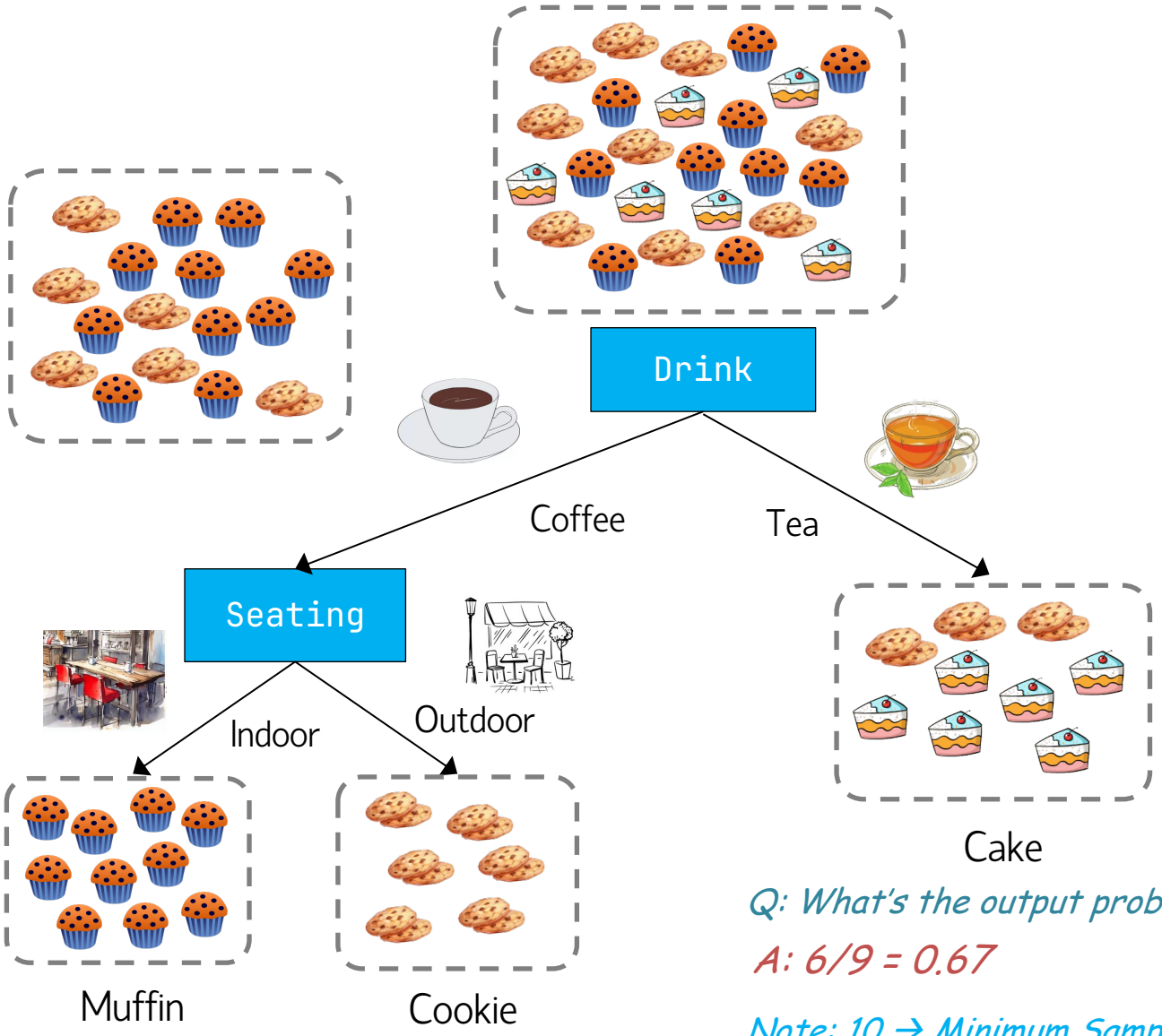

| Seating | Drink | Dessert |
|---|---|---|
| Outdoor | Tea | Cookie |
| Outdoor | Tea | Cookie |
| Outdoor | Tea | Cookie |
| Outdoor | Tea | Cookie |
| Outdoor | Coffee | Cookie |
| Indoor | Coffee | Muffin |
| Indoor | Coffee | Muffin |
| Indoor | Tea | Cake |
| Indoor | Tea | Cake |
| Indoor | Tea | Cake |

*Q: How many did we classify it correctly?*

*A:* $$\text{Accuracy} = \frac{2+3+5}{10} = 1.0$$

*Note: 10 → Minimum Samples Split*

# Decision Tree (cont.)




| Seating | Drink | Dessert |
|---|---|---|
| Outdoor | Tea | Cookie |
| Outdoor | Tea | Cookie |
| Outdoor | Tea | Cookie |
| Outdoor | Tea | Cookie |
| Outdoor | Coffee | Cookie |
| Indoor | Coffee | Muffin |
| Indoor | Coffee | Muffin |
| Indoor | Tea | Cake |
| Indoor | Tea | Cake |
| Indoor | Tea | Cake |

*Q: What's the output probability?*

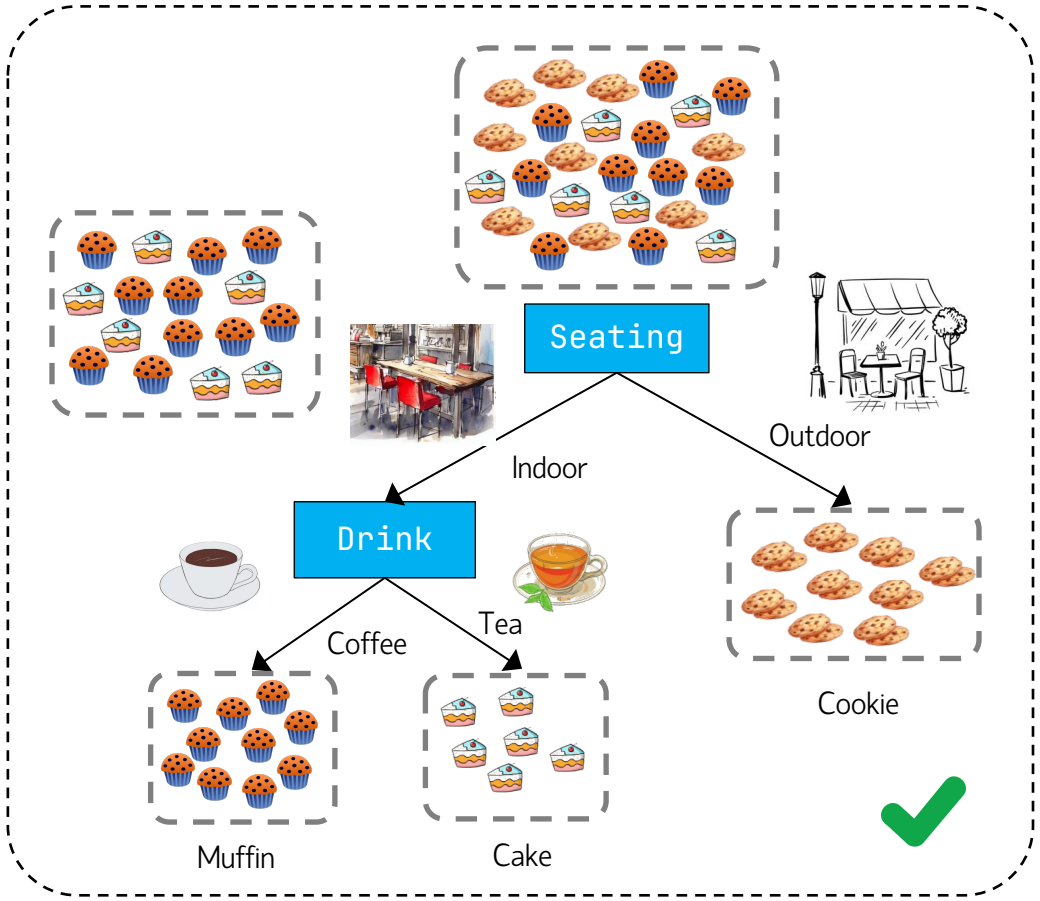
*A: 6/9 = 0.67*

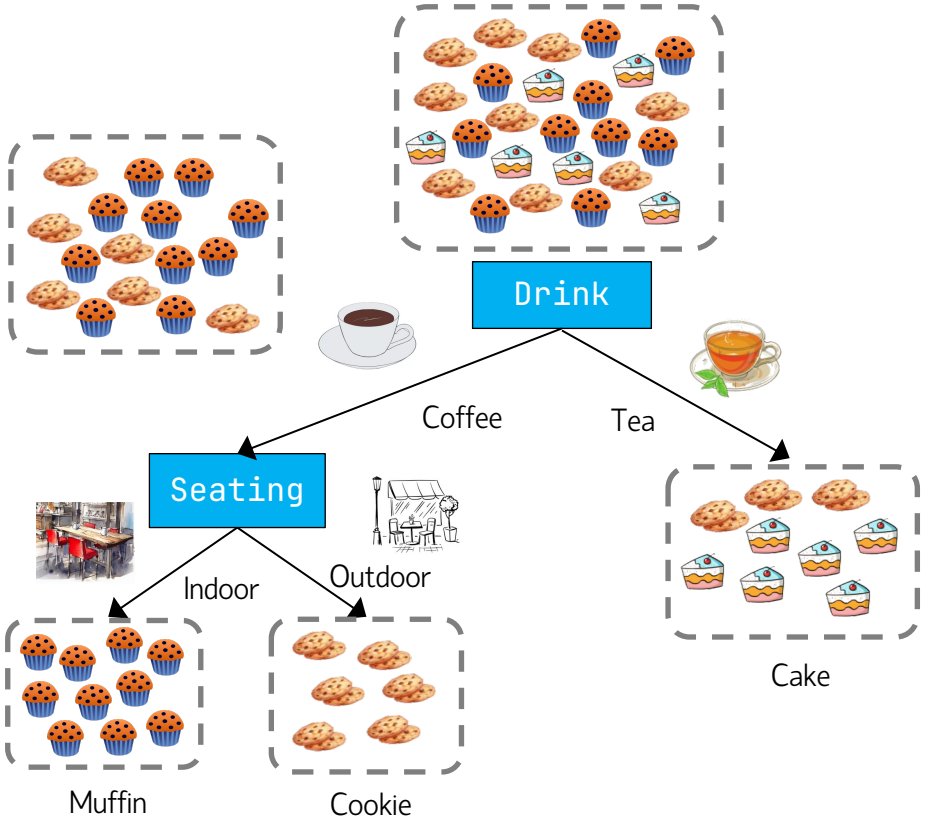
*Note: 10 → Minimum Samples Split*

*Q: How many did we classify it correctly?*

$$\text{Accuracy} = \frac{2+1+3}{10} = 0.6$$

# Which Tree Is Better?



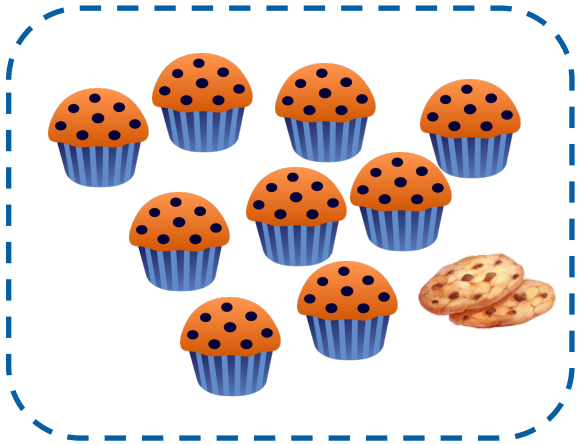

*Note: 10 → Minimum Samples Split*

*Q: We want certainty. How do we construct one?*

*A: Purer Leaves.*

# Classification Errors

*Low Entropy*

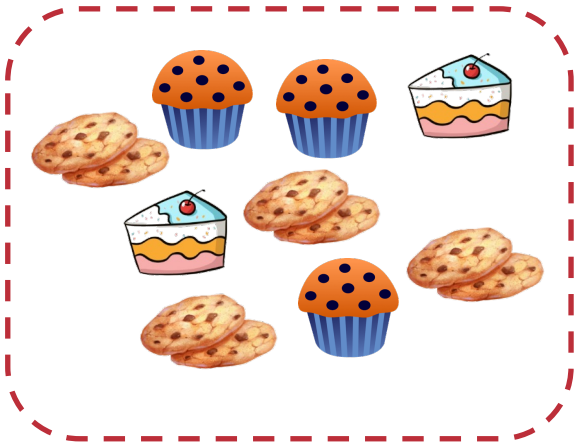


*10 customers who ordered Latte*

We are quite certain that customers who order Latte will also order Muffins.

Muffins. 9 out of 10 customers did that.

*High Entropy*



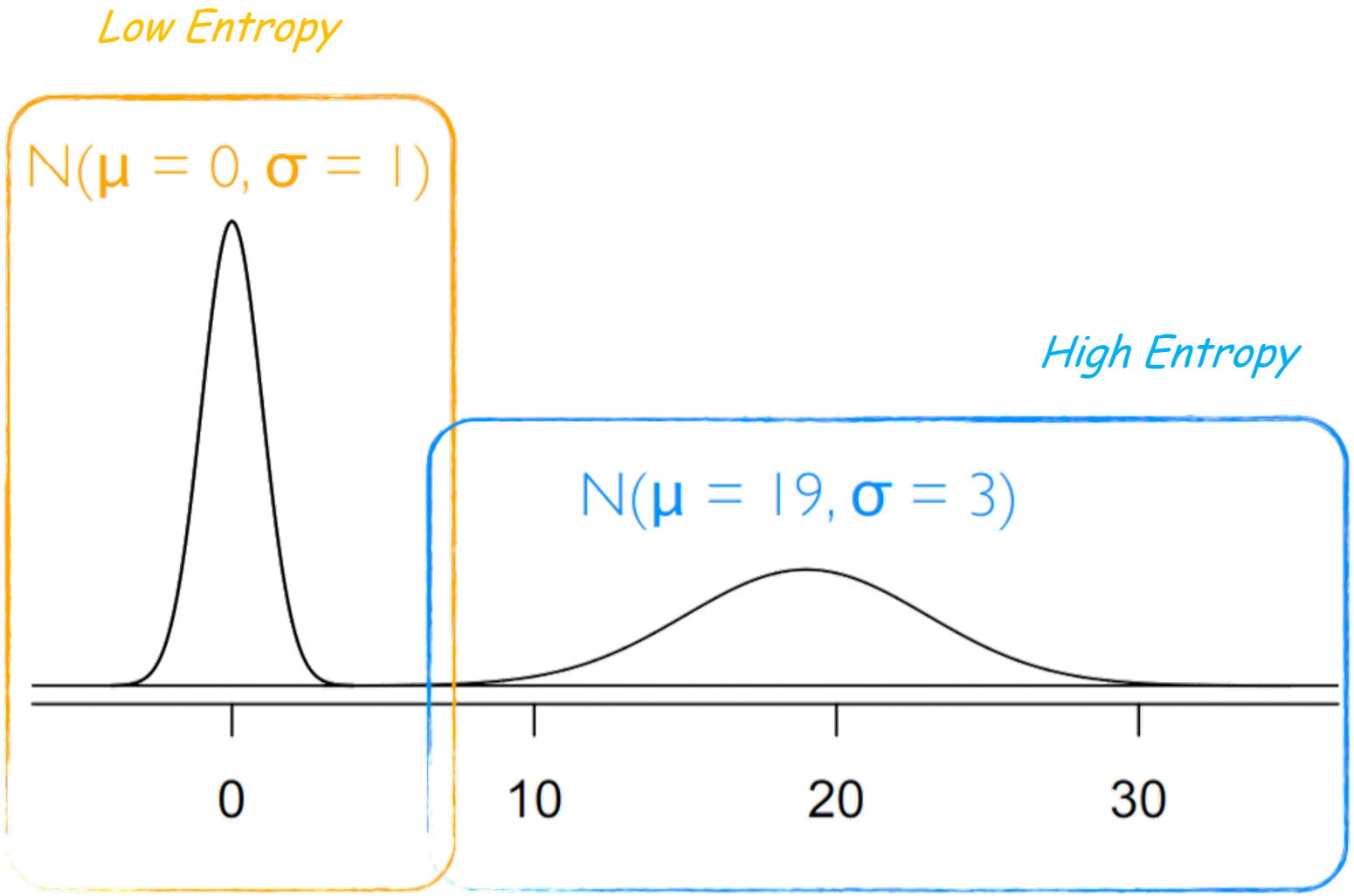
*9 customers who ordered Mocha*

We are less certain that customers who order Mocha will also order Cookies.

Only 4 out of 9 customers did that.

***Ordering Cookiers or not, we are going to make more classification error with this group of customers.***

# Normal Distribution

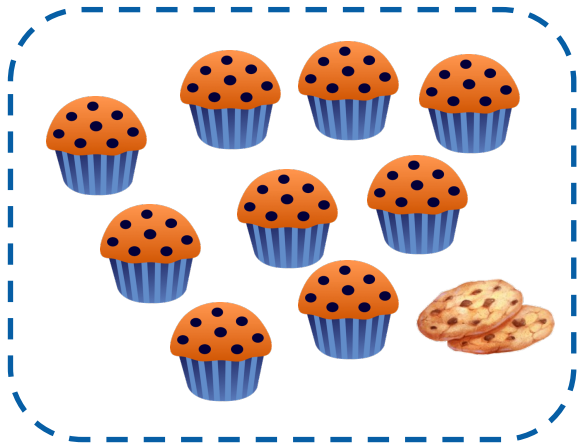


# Entropy

Entropy is a measure of disorder or impurity in the given dataset.

$$E = -\sum_{i=1}^{N} p_i \log_2(p_i)$$

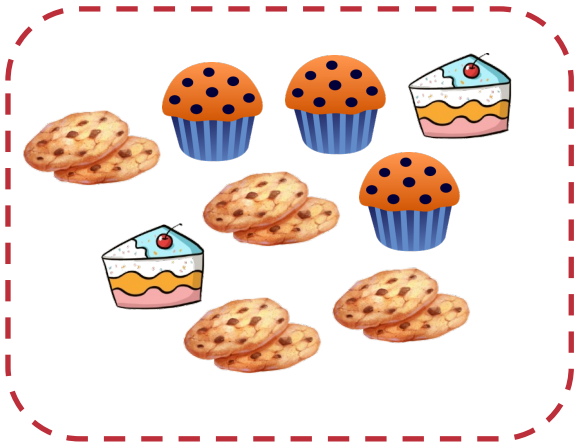
*Low Impurity → Low Entropy*



$$E = -\left(\frac{9}{10}\log_2\left(\frac{9}{10}\right) + \frac{1}{10}\log_2\left(\frac{1}{10}\right)\right)$$

$$= -(-0.14 - 0.33) = -(-0.47) = 0.47$$

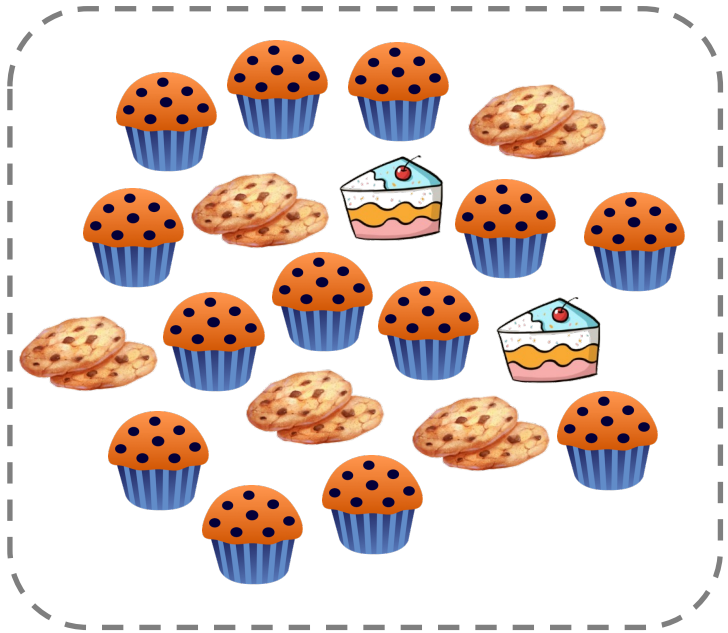
*High Impurity → High Entropy*



$$E = -\left(\frac{3}{9}\log_2\left(\frac{3}{9}\right) + \frac{4}{9}\log_2\left(\frac{4}{9}\right) + \frac{2}{9}\log_2\left(\frac{2}{9}\right)\right)$$

$$= -(-0.53 - 0.53 - 0.46) = -(-1.52) = 1.52$$

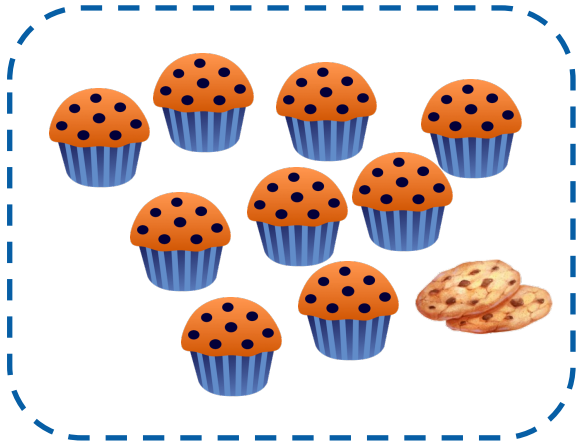
# Information Gain



$E = 1.23$

0.23

$E = 0.47$

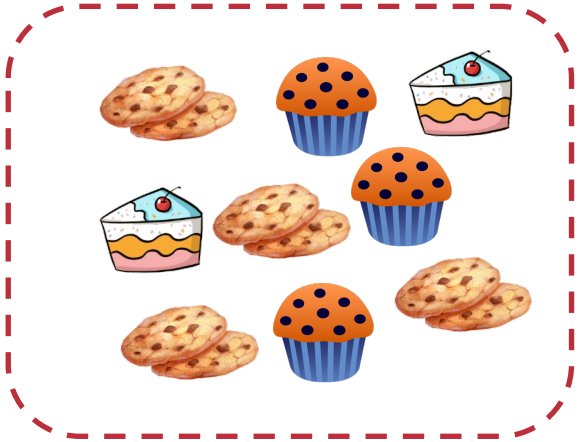


Latte

$$E = -\left(\frac{12}{19}\log_2\left(\frac{12}{19}\right) + \frac{5}{19}\log_2\left(\frac{5}{19}\right) + \frac{2}{19}\log_2\left(\frac{2}{19}\right)\right)$$

$$= -(-0.40 - 0.50 - 0.33) = 1.23$$

$E = 1.52$



Mocha

Information Gain $IG = 1.23 - 1.00$

$= 0.23$

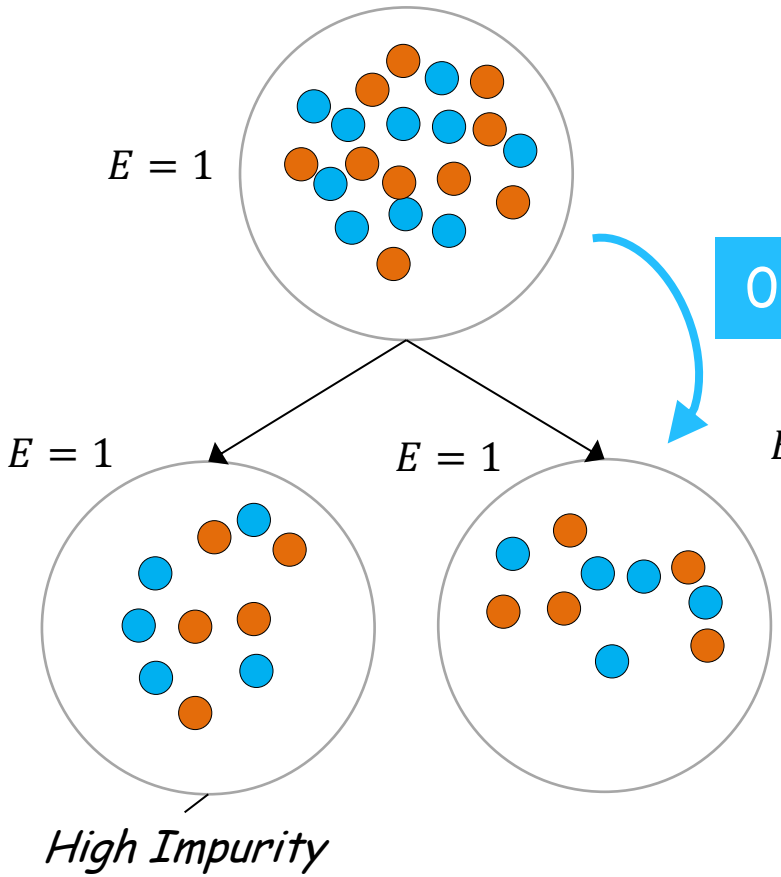
$$E = \frac{10}{19}0.47 + \frac{9}{19}1.52$$

(Weighted Sum)

$= 0.24 + 0.76 = 1.00$

# Information Gain (cont.)

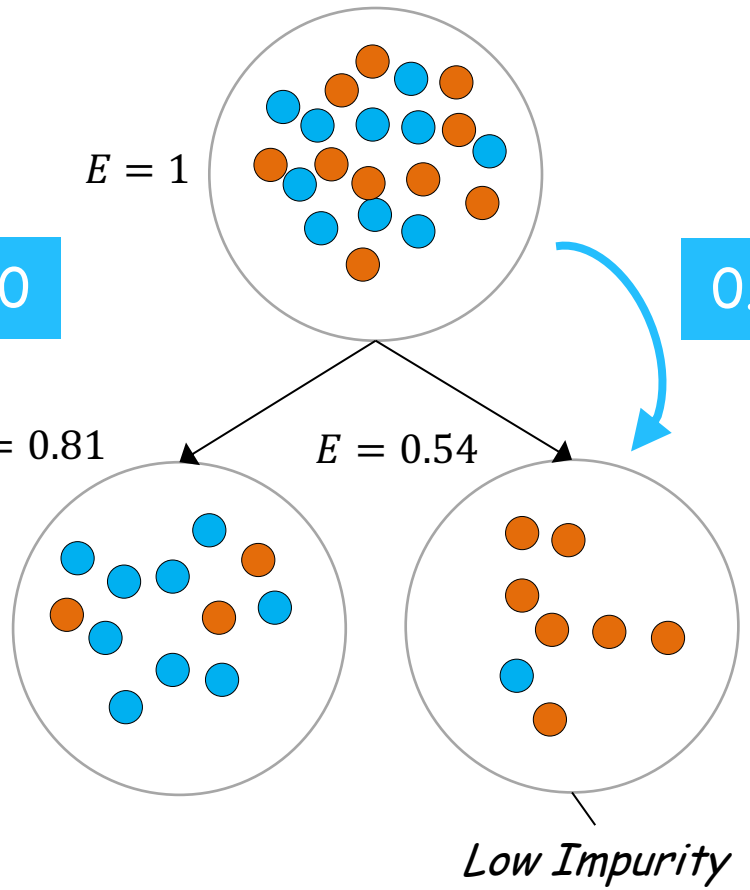
Entropy: $E = -\sum_{i=1}^{N} p_i \log_2(p_i)$




0.00

*High Impurity*

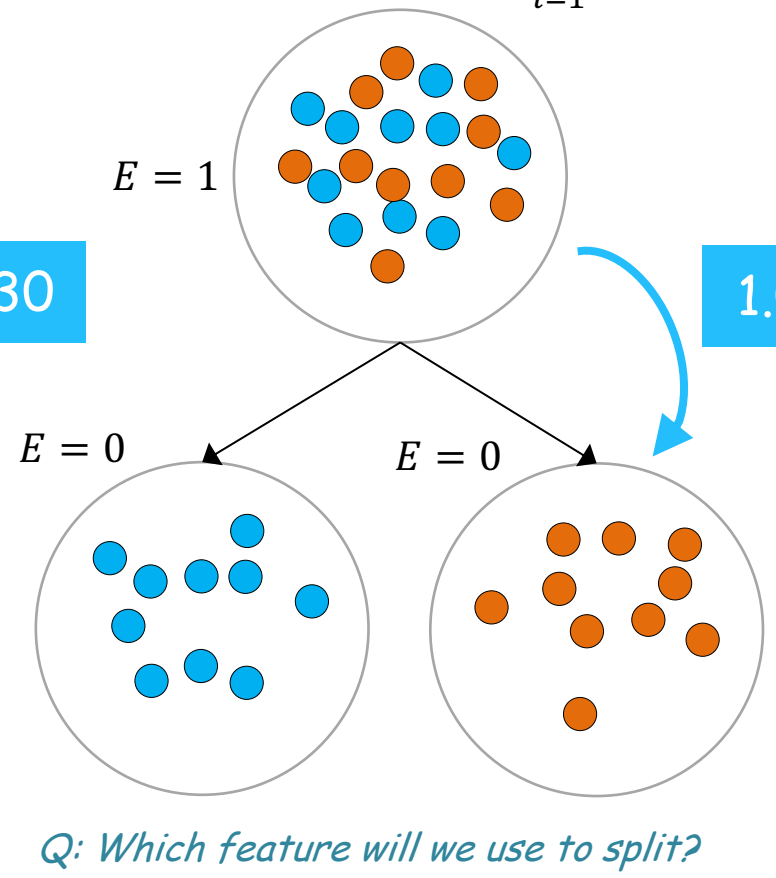
$$E = \frac{10}{20}1.00 + \frac{10}{20}1.00 = 1.00$$




0.30

*Low Impurity*

$$E = \frac{12}{20}0.81 + \frac{8}{20}0.54 = 0.70$$




1.00

*Q: Which feature will we use to split?*

*A: The one that maximises the information gain.*

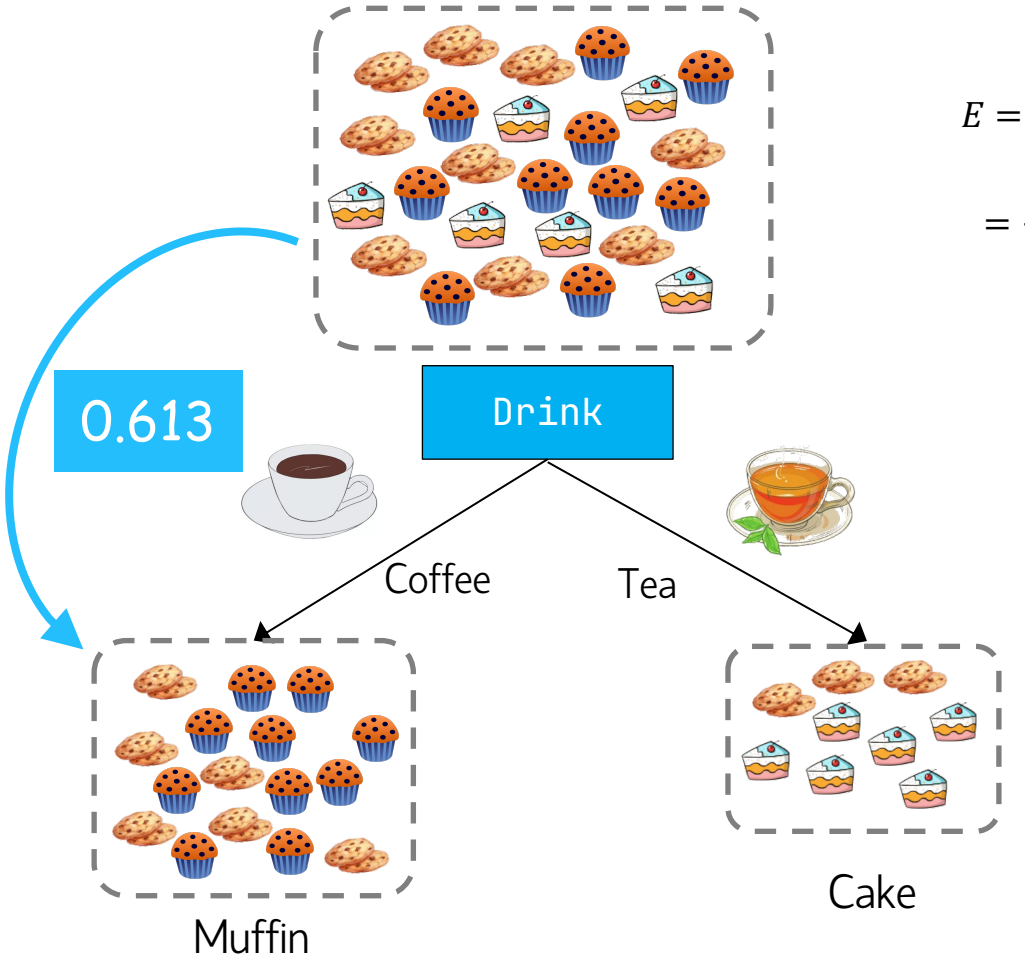
# Café Example Dataset

| Seating | Drink | TimeOfDay | Dessert |
|---|---|---|---|
| Indoor | Coffee | Morning | Muffin |
| Indoor | Coffee | Morning | Muffin |
| Indoor | Coffee | Morning | Muffin |
| Indoor | Coffee | Morning | Muffin |
| Indoor | Coffee | Morning | Muffin |
| Indoor | Coffee | Afternoon | Muffin |
| Indoor | Coffee | Afternoon | Muffin |
| Indoor | Coffee | Afternoon | Muffin |
| Indoor | Coffee | Evening | Muffin |
| Indoor | Coffee | Evening | Muffin |
| Indoor | Tea | Morning | Cake |
| Indoor | Tea | Afternoon | Cake |
| Indoor | Tea | Afternoon | Cake |

| Seating | Drink | TimeOfDay | Dessert |
|---|---|---|---|
| Indoor | Tea | Evening | Cake |
| Indoor | Tea | Evening | Cake |
| Indoor | Tea | Evening | Cake |
| Outdoor | Coffee | Morning | Cookie |
| Outdoor | Coffee | Morning | Cookie |
| Outdoor | Coffee | Afternoon | Cookie |
| Outdoor | Coffee | Afternoon | Cookie |
| Outdoor | Coffee | Evening | Cookie |
| Outdoor | Coffee | Evening | Cookie |
| Outdoor | Tea | Afternoon | Cookie |
| Outdoor | Tea | Evening | Cookie |
| Outdoor | Tea | Evening | Cookie |

*Cross-Selling: Identify natural add-ons and present them at the right moment.*

# Decision Tree Learning (1 of 8)




$$E = -\left(\frac{10}{25}\log_2\left(\frac{10}{25}\right) + \frac{9}{25}\log_2\left(\frac{9}{25}\right) + \frac{6}{25}\log_2\left(\frac{6}{25}\right)\right)$$

$$= -(-0.529 - 0.531 - 0.494) = 1.554$$

$$E = -\left(\frac{6}{9}\log_2\left(\frac{6}{9}\right) + \frac{3}{9}\log_2\left(\frac{3}{9}\right)\right)$$

$$= -(-0.390 - 0.528) = 0.918$$

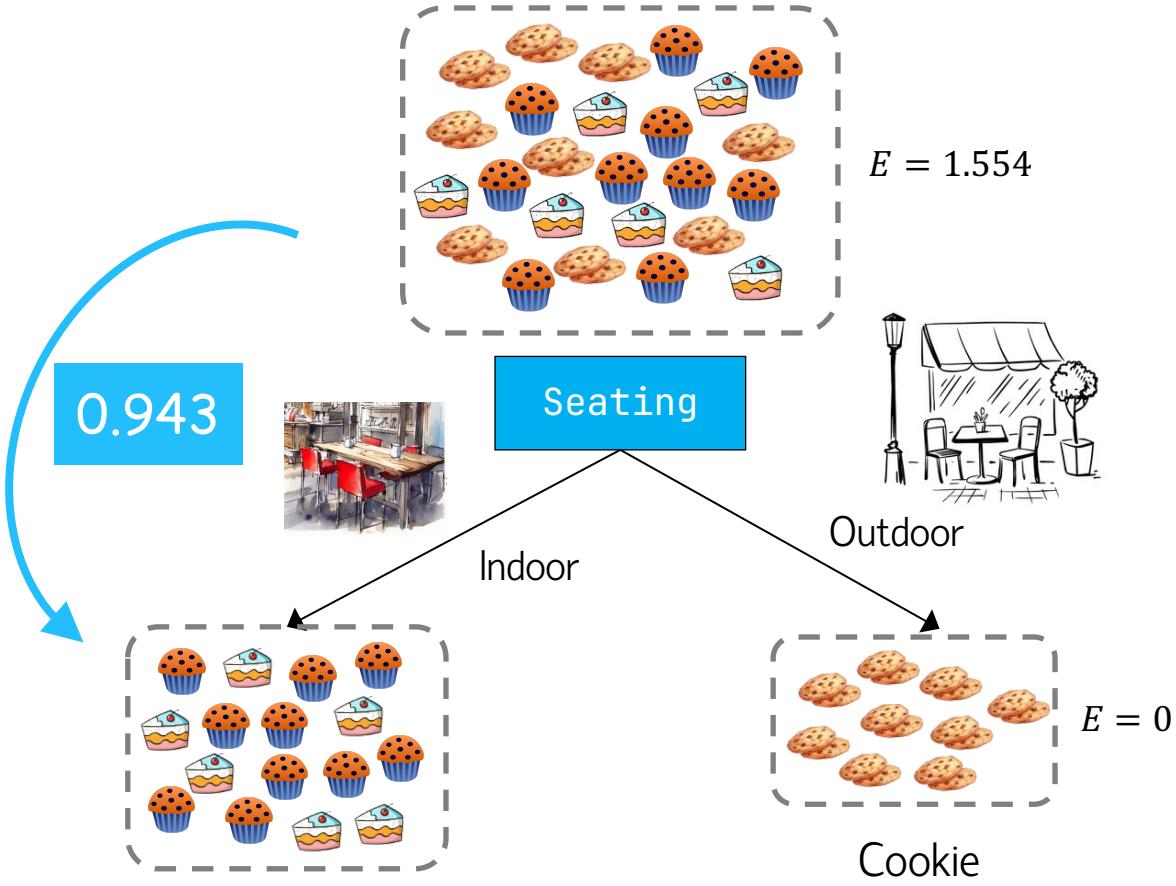
$$E = -\left(\frac{10}{16}\log_2\left(\frac{10}{16}\right) + \frac{6}{16}\log_2\left(\frac{6}{16}\right)\right)$$

$$= -(-0.424 - 0.531) = 0.955$$

$$IG = 1.554 - \left(\frac{16}{25}0.955 + \frac{9}{25}0.918\right)$$

$$= 1.554 - (0.611 + 0.330) = 0.613$$

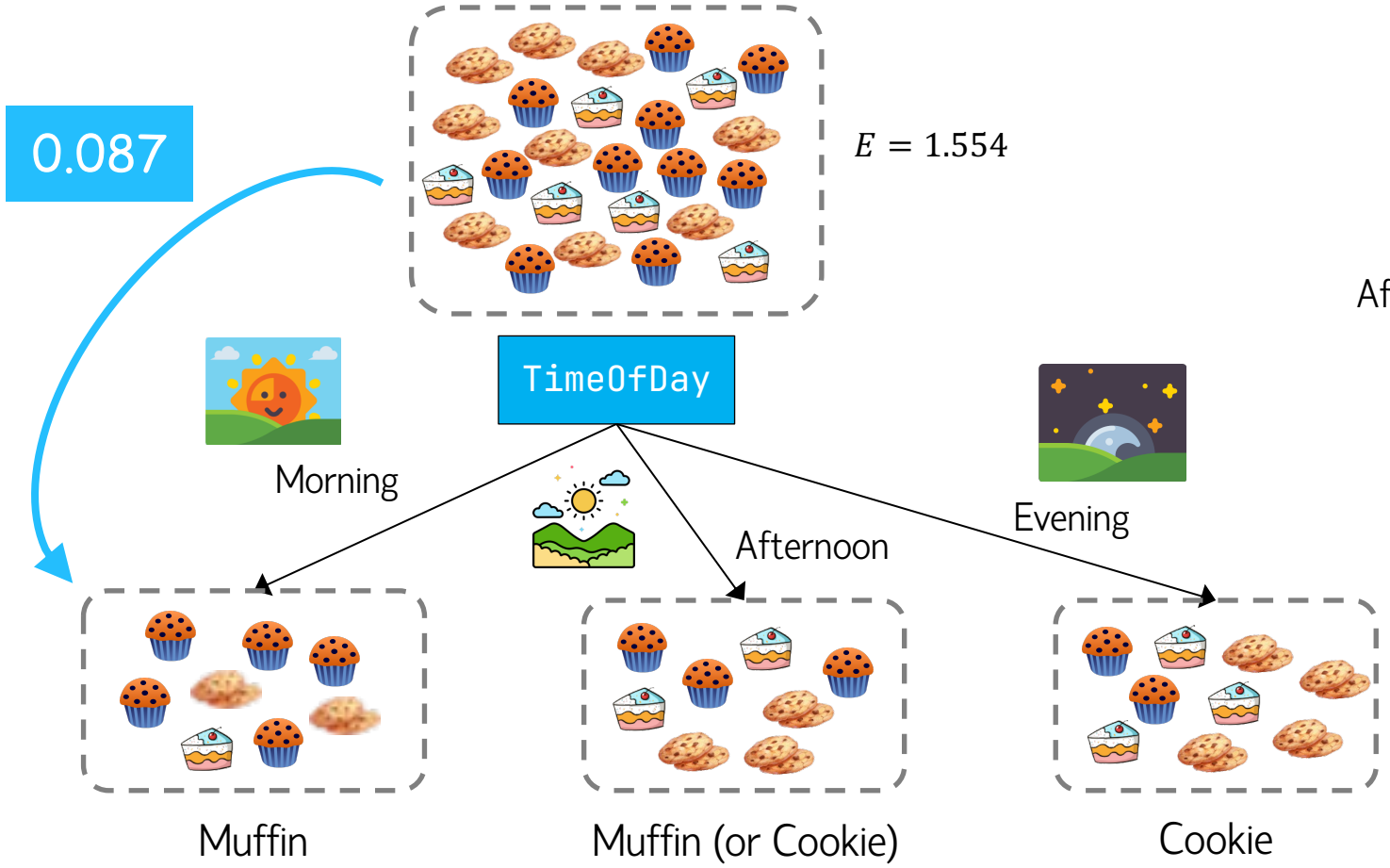
# Decision Tree Learning (2 of 8)




$$E = -\left(\frac{10}{16}\log_2\left(\frac{10}{16}\right) + \frac{6}{16}\log_2\left(\frac{6}{16}\right)\right)$$
$$= -(-0.424 - 0.531) = 0.955$$

$$IG = 1.554 - \left(\frac{16}{25}0.955 + \frac{9}{25}0.0\right)$$
$$= 1.554 - (0.611 + 0.0) = 0.943$$

# Decision Tree Learning (3 of 8)




Morning:

$$E = -\left(\frac{5}{8}\log_2\left(\frac{5}{8}\right) + \frac{2}{8}\log_2\left(\frac{2}{8}\right) + \frac{1}{8}\log_2\left(\frac{1}{8}\right)\right)$$

$$= -(-0.424 - 0.5 - 0.375) = 1.299$$

Afternoon:

$$E = -\left(\frac{3}{8}\log_2\left(\frac{3}{8}\right) + \frac{3}{8}\log_2\left(\frac{3}{8}\right) + \frac{2}{8}\log_2\left(\frac{2}{8}\right)\right)$$

$$= -(-0.531 - 0.531 - 0.5) = 1.562$$

Evening:

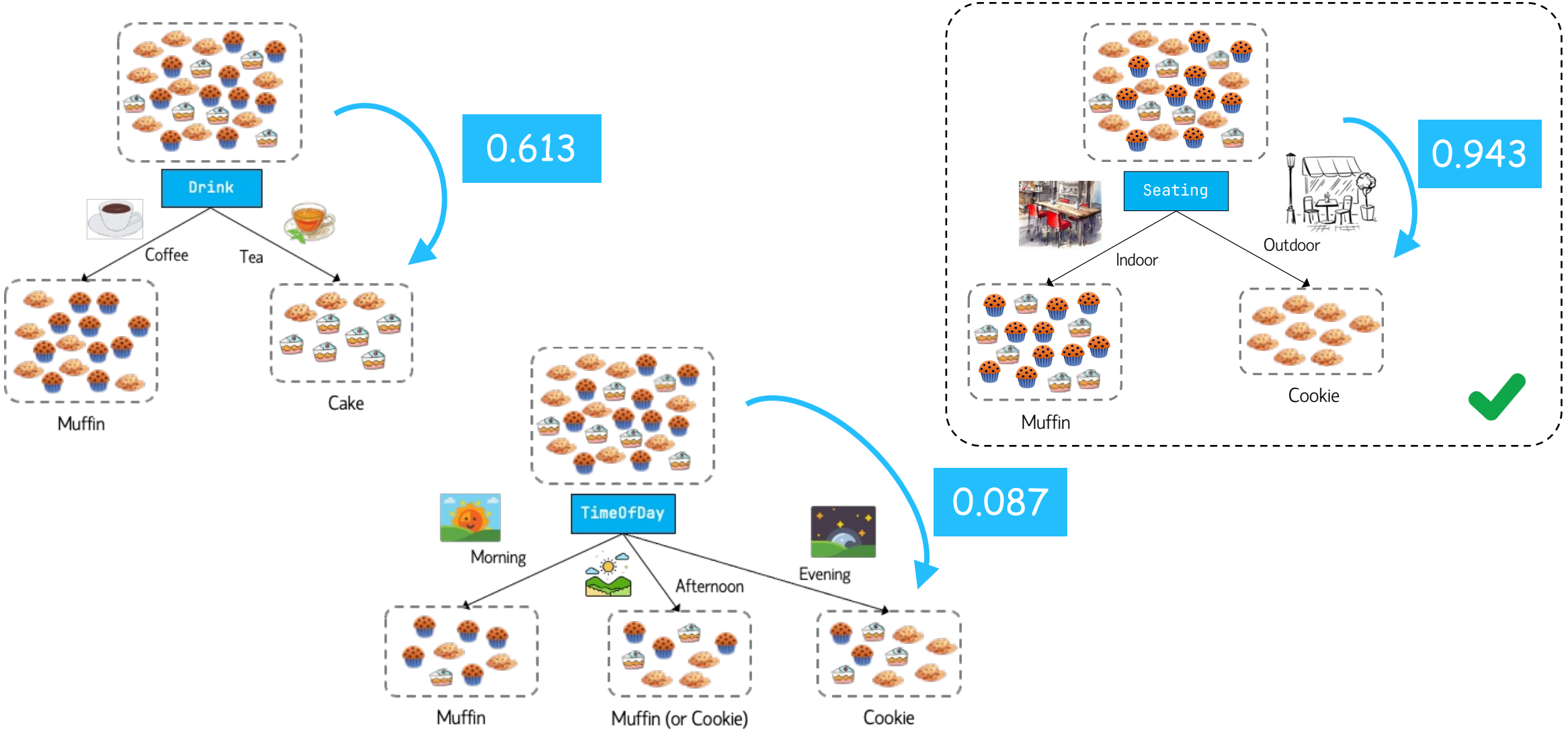
$$E = -\left(\frac{2}{9}\log_2\left(\frac{2}{9}\right) + \frac{4}{9}\log_2\left(\frac{4}{9}\right) + \frac{3}{9}\log_2\left(\frac{3}{9}\right)\right)$$

$$= -(-0.482 - 0.520 - 0.528) = 1.530$$

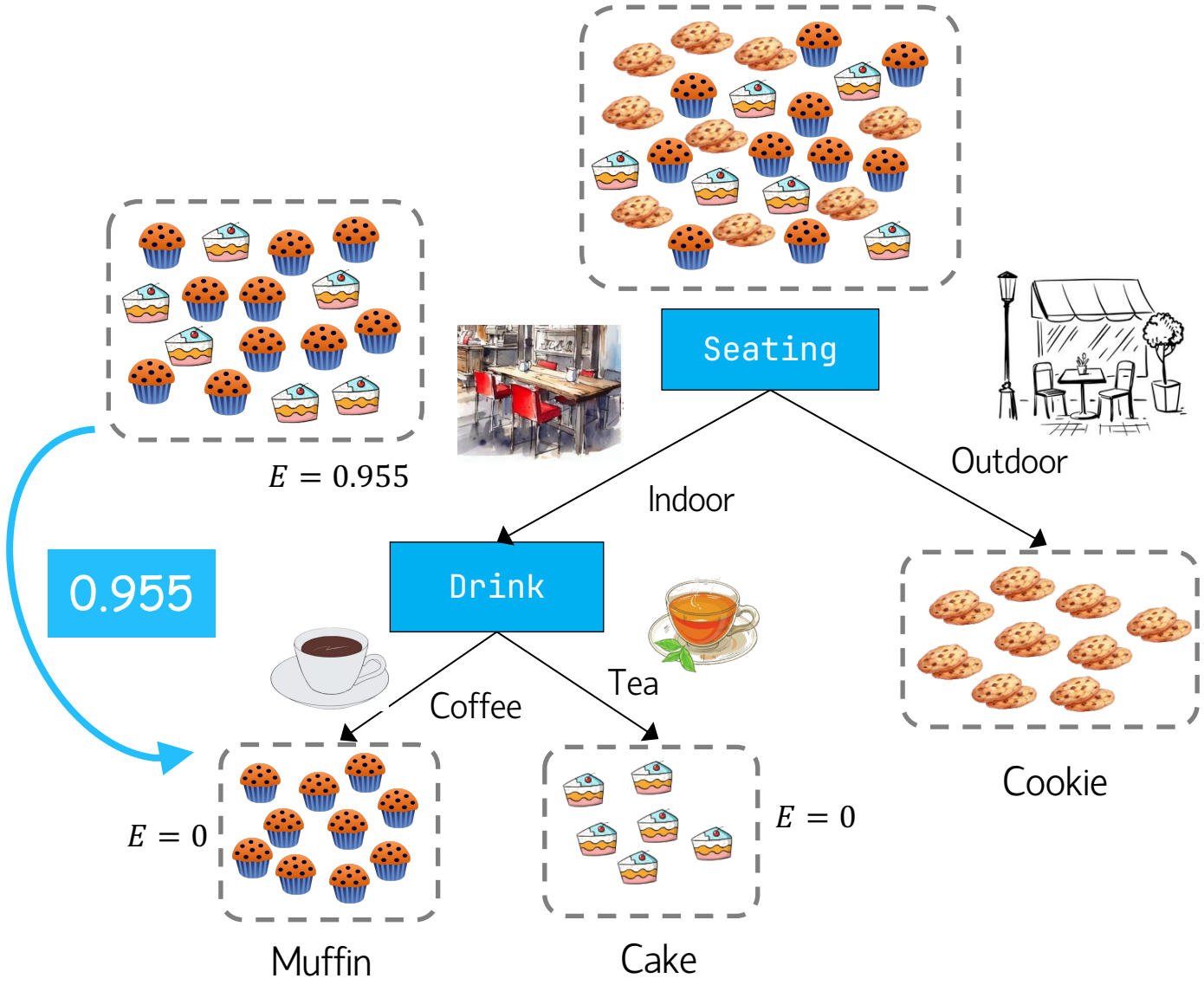
$$IG = 1.554 - \left(\frac{8}{25}1.299 + \frac{8}{25}1.562 + \frac{9}{25}1.530\right)$$

$$= 1.554 - (0.416 + 0.50 + 0.551) = 0.087$$

# Decision Tree Learning (4 of 8)



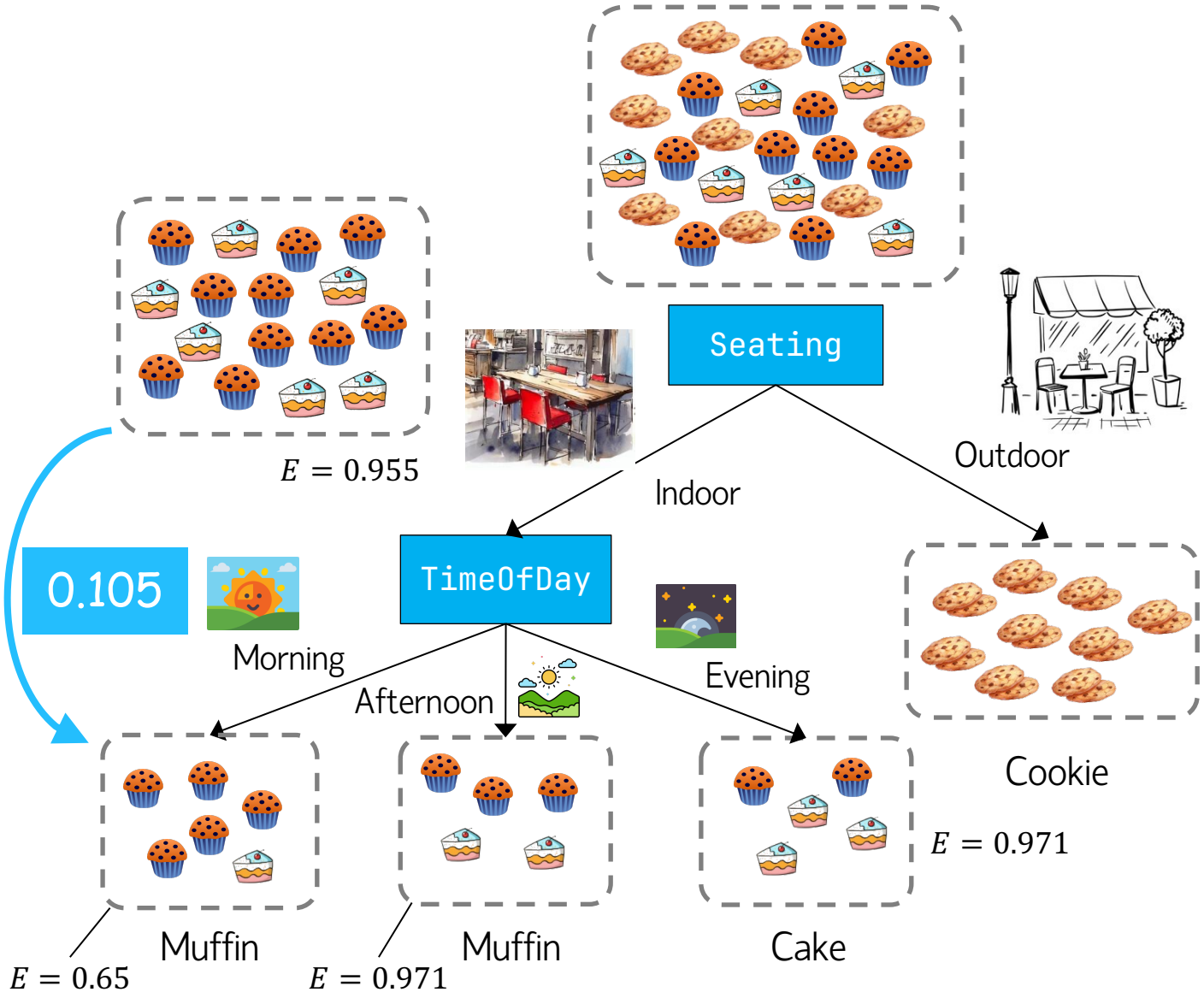
# Decision Tree Learning (5 of 8)




$$IG = 0.955 - \left(\frac{10}{16}0.0 + \frac{6}{16}0.0\right)$$

$$= 1.554 - (0.0 + 0.0) = 0.955$$

# Decision Tree Learning (6 of 8)




Morning: $E = -\left(\frac{5}{6}\log_2\left(\frac{5}{6}\right) + \frac{1}{6}\log_2\left(\frac{1}{6}\right)\right)$

$= -(-0.219 - 0.431) = 0.650$

Afternoon: $E = -\left(\frac{3}{5}\log_2\left(\frac{3}{5}\right) + \frac{2}{5}\log_2\left(\frac{2}{5}\right)\right)$

$= -(-0.442 - 0.529) = 0.971$

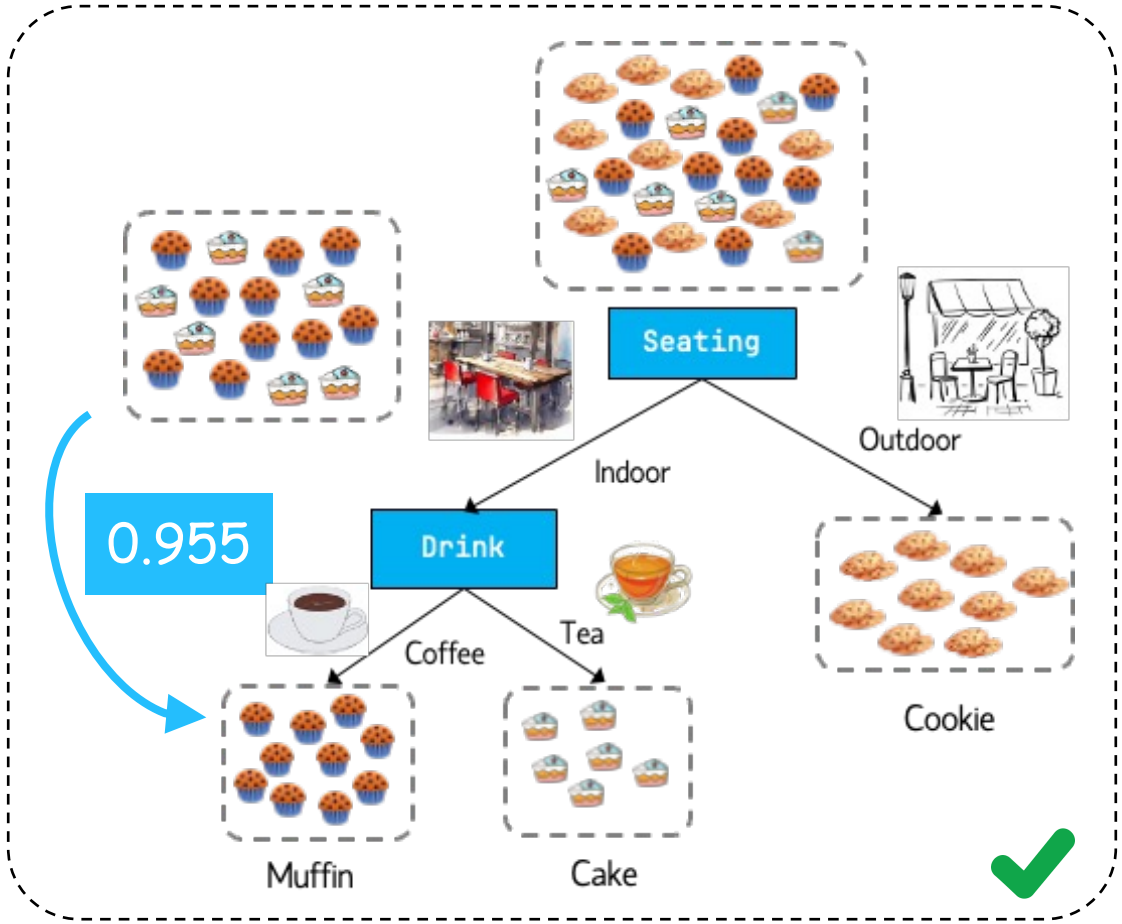
Evening: $E = -\left(\frac{2}{5}\log_2\left(\frac{2}{5}\right) + \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right)$

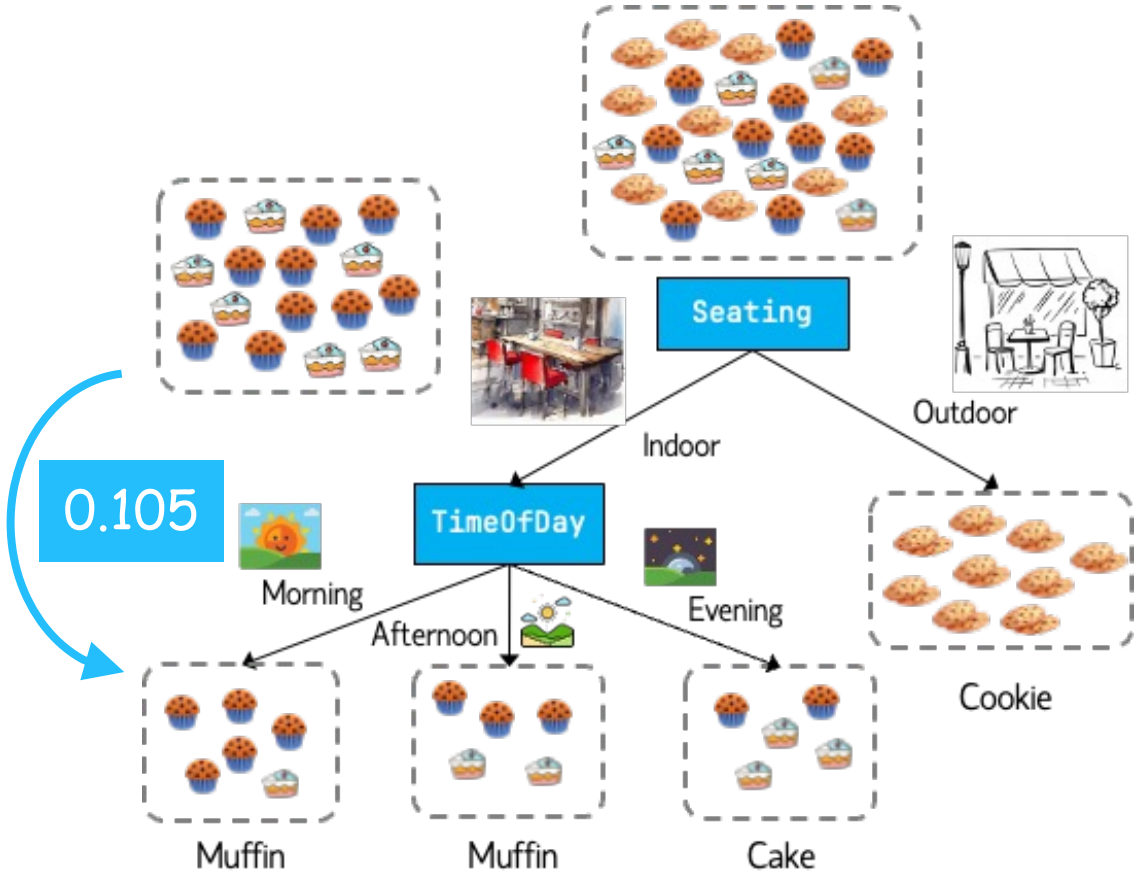
$= -(-0.529 - 0.442) = 0.971$

$$IG = 0.955 - \left(\frac{6}{16}0.650 + \frac{5}{16}0.971 + \frac{5}{16}0.971\right)$$

$$= 0.955 - (0.244 + 0.303 + 303) = 0.105$$

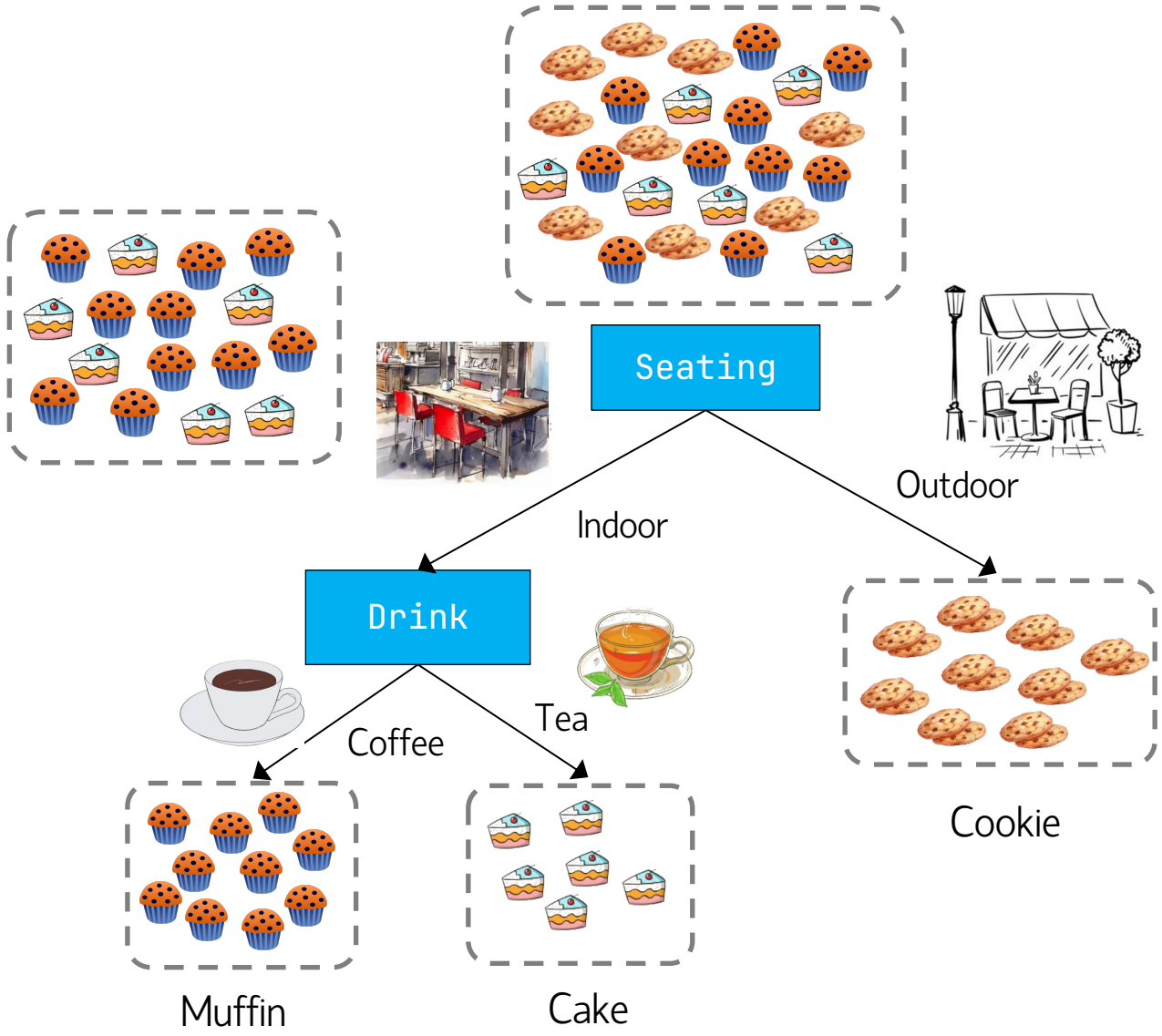
# Decision Tree Learning (7 of 8)




*Note: 10 → Minimum Samples Split*

# Decision Tree Learning (8 of 8)

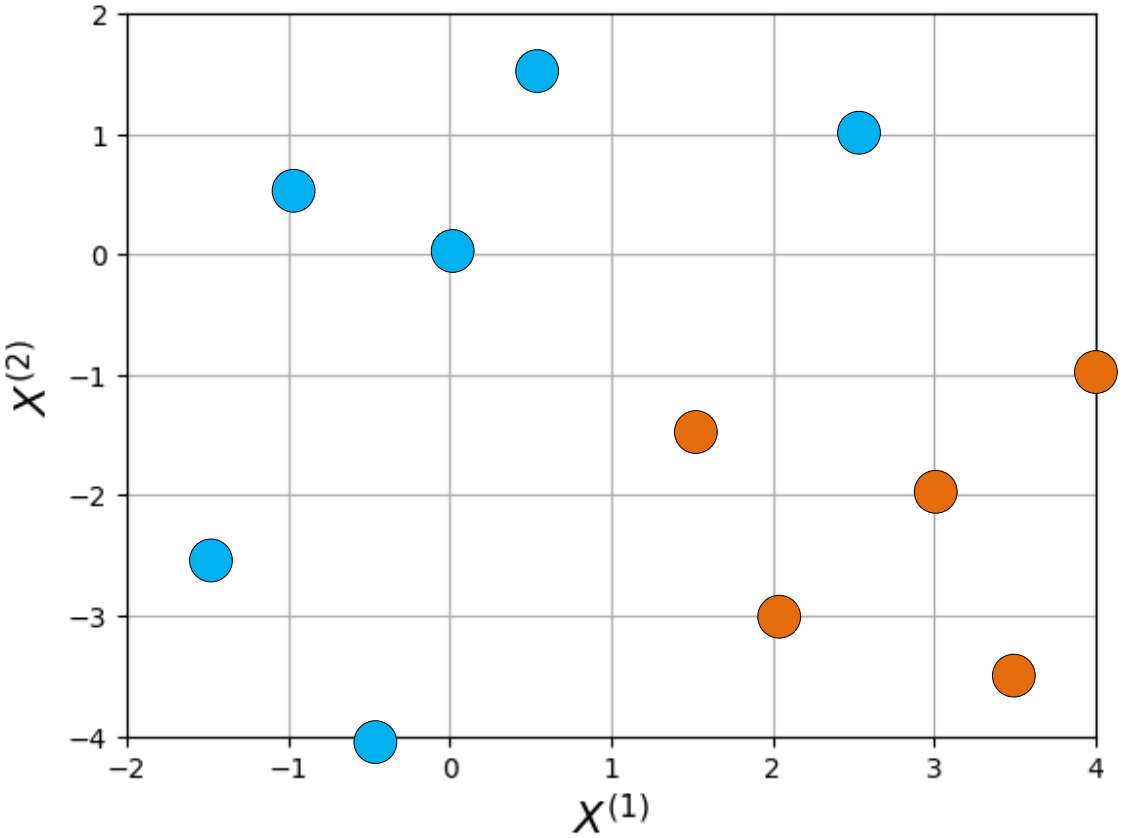



```
IF Seating = Outdoor
    → predict Cookie
ELSE  # Indoor
    IF Drink = Coffee
        → predict Muffin
    ELSE
        → predict Cake
```

# Pseudocode for Decision Tree Classifier

```
Function DecisionTree_Classify(new_point, data, features, labels)
Begin
    Step 1: Stopping Criteria
        1.1 If all labels in `labels` are the same OR `features` is empty Then
            Return majority_label(labels)

    Step 2: Compute Base Entropy
        base_entropy ← Entropy(labels)

    Step 3: Find Best Feature by Information Gain
        best_gain   ← -∞
        best_feature ← None

        For each feature F in features Do
            3.1 Compute weighted child entropy for F
                weighted_entropy ← 0
                For each value v in unique_values(data[F]) Do
                    subset_labels ← labels of rows in `data` where data[F] = v
                    weight ← |subset_labels| ÷ |labels|
                    weighted_entropy ← weighted_entropy + weight × Entropy(subset_labels)

            3.2 Compute information gain
                info_gain ← base_entropy - weighted_entropy
                If info_gain > best_gain Then
                    best_gain    ← info_gain
                    best_feature ← F

    Step 4: Split on best_feature and Recurse
        branch_value  ← new_point[best_feature]
        subset_data   ← rows in `data` where best_feature = branch_value
        subset_labels ← corresponding labels

        If subset_data is empty Then
            Return majority_label(labels)   // no examples: fallback
        Else
            remaining_features ← features minus {best_feature}
            Return DecisionTree_Classify(
                new_point,
                subset_data,
                remaining_features,
                subset_labels
            )
End
```

# Decision Tree Learning for Numerical Features (1 of 7)




$x^{(1)}$ and $x^{(2)}$ are numerical features.

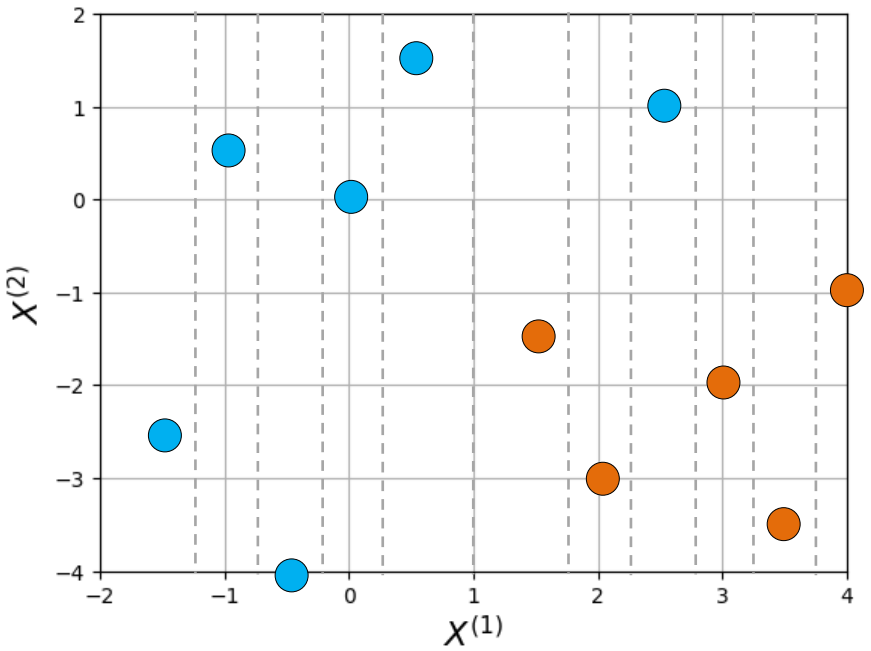
$$E = -\left(\frac{6}{11}\log_2\left(\frac{6}{11}\right) + \frac{5}{11}\log_2\left(\frac{5}{11}\right)\right)$$

$$= -(-0.477 - 0.517) = 0.994$$

$$= 0.994$$

*Q: How do we decide where to cut the axes?*

# Decision Tree Learning for Numerical Features (2 of 7)




Sort the unique values of $X$:

$$v_1 \leq v_2 \leq \cdots \leq v_m$$

Consider mid-points between each pair of adjacent values:

$$t_k = \frac{v_k + v_{k+1}}{2} \qquad k = 1, 2, \ldots, m-1$$

We now have up to $m - 1$ candidate splits.

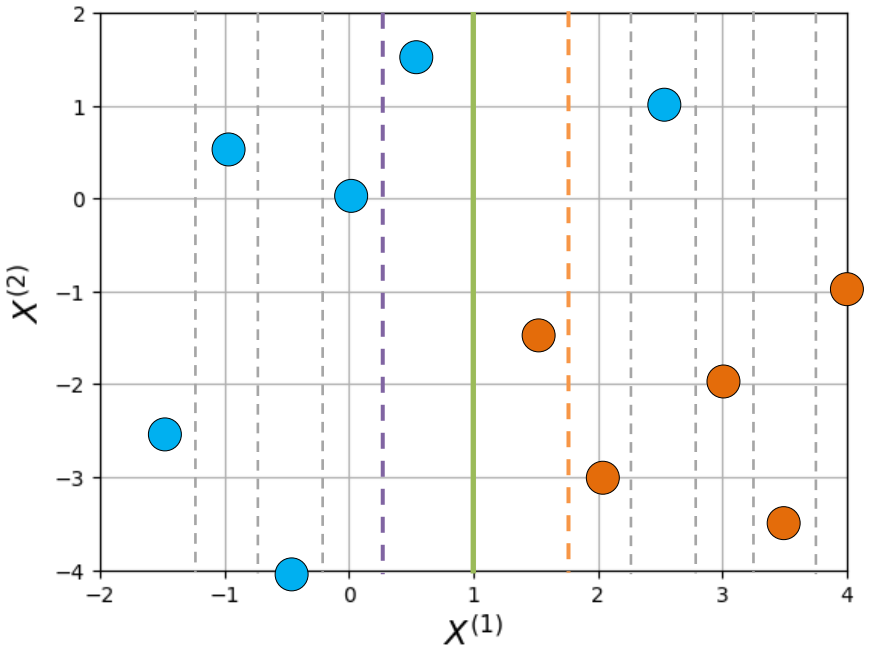
For each threshold $t_k$, compute entropy for each child and then their weighted average to find information gain from the split.

Pick the best threshold.

Unique Values of $x^{(1)}$ : `[-1.5, -1.0, -0.5, 0.0, 0.5, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]`

Candidate Splits : `[-1.25, -0.75, -0.25, 0.25, 1.0, 1.75, 2.25, 2.75, 3.25, 3.75]`

# Decision Tree Learning for Numerical Features (3 of 7)




$t_5 = 1.0$ : $IG = 0.994 - 0.355 = 0.629$

$t_4 = 0.25$ :

$$E = \frac{4}{11} \times -\left(\frac{4}{4}\log_2\frac{4}{4}\right) + \frac{7}{11} \times -\left(\frac{5}{7}\log_2\frac{5}{7} + \frac{2}{7}\log_2\frac{2}{7}\right)$$
$$= \frac{4}{11} \times 0.00 + \frac{7}{11} \times 0.863$$
$$= 0.549$$

$t_5 = 1.0$ :

$$E = \frac{5}{11} \times -\left(\frac{5}{5}\log_2\frac{5}{5}\right) + \frac{6}{11} \times -\left(\frac{5}{6}\log_2\frac{5}{6} + \frac{1}{6}\log_2\frac{1}{6}\right)$$
$$= \frac{5}{11} \times 0.00 + \frac{6}{11} \times 0.650$$
$$= 0.355$$

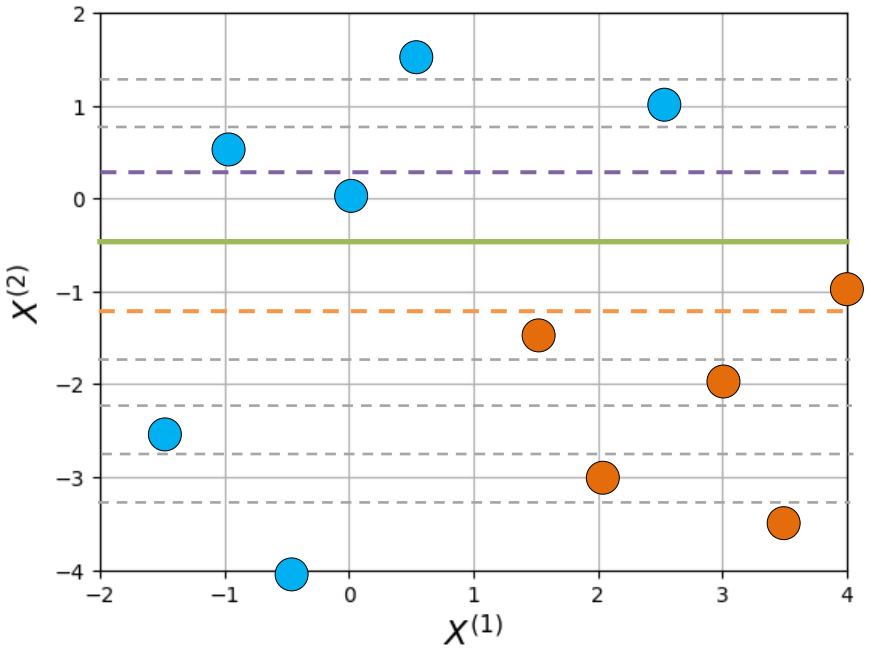
$t_6 = 1.75$ :

$$E = \frac{6}{11} \times -\left(\frac{5}{6}\log_2\frac{5}{6} + \frac{1}{6}\log_2\frac{1}{6}\right) + \frac{5}{11} \times -\left(\frac{4}{5}\log_2\frac{4}{5} + \frac{1}{5}\log_2\frac{1}{5}\right)$$
$$= \frac{6}{11} \times 0.650 + \frac{5}{11} \times 0.722$$
$$= 0.683$$

Unique Values of $x^{(1)}$ : `[-1.5, -1.0, -0.5, 0.0, 0.5, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]`

Candidate Splits : `[-1.25, -0.75, -0.25, 0.25, 1.0, 1.75, 2.25, 2.75, 3.25, 3.75]`

# Decision Tree Learning for Numerical Features (4 of 7)




$t_6 = -0.5$: $IG = 0.994 - 0.549 = 0.445$

$t_5 = -1.25$ : $E = \frac{6}{11} \times -\left(\frac{4}{6}\log_2\frac{4}{6} + \frac{2}{6}\log_2\frac{2}{6}\right) + \frac{5}{11} \times -\left(\frac{1}{5}\log_2\frac{1}{5} + \frac{4}{5}\log_2\frac{4}{5}\right)$

$= \frac{6}{11} \times 0.918 + \frac{5}{11} \times 0.722$

$= 0.829$

$t_6 = -0.5$ : $E = \frac{7}{11} \times -\left(\frac{5}{7}\log_2\frac{5}{7} + \frac{2}{7}\log_2\frac{2}{7}\right) + \frac{4}{11} \times -\left(\frac{4}{4}\log_2\frac{4}{4}\right)$

$= \frac{7}{11} \times 0.863 + \frac{4}{11} \times 0.00$

$= 0.549$

$t_7 = 0.25$ : $E = \frac{8}{11} \times -\left(\frac{5}{8}\log_2\frac{5}{8} + \frac{3}{8}\log_2\frac{3}{8}\right) + \frac{3}{11} \times -\left(\frac{3}{3}\log_2\frac{3}{3}\right)$

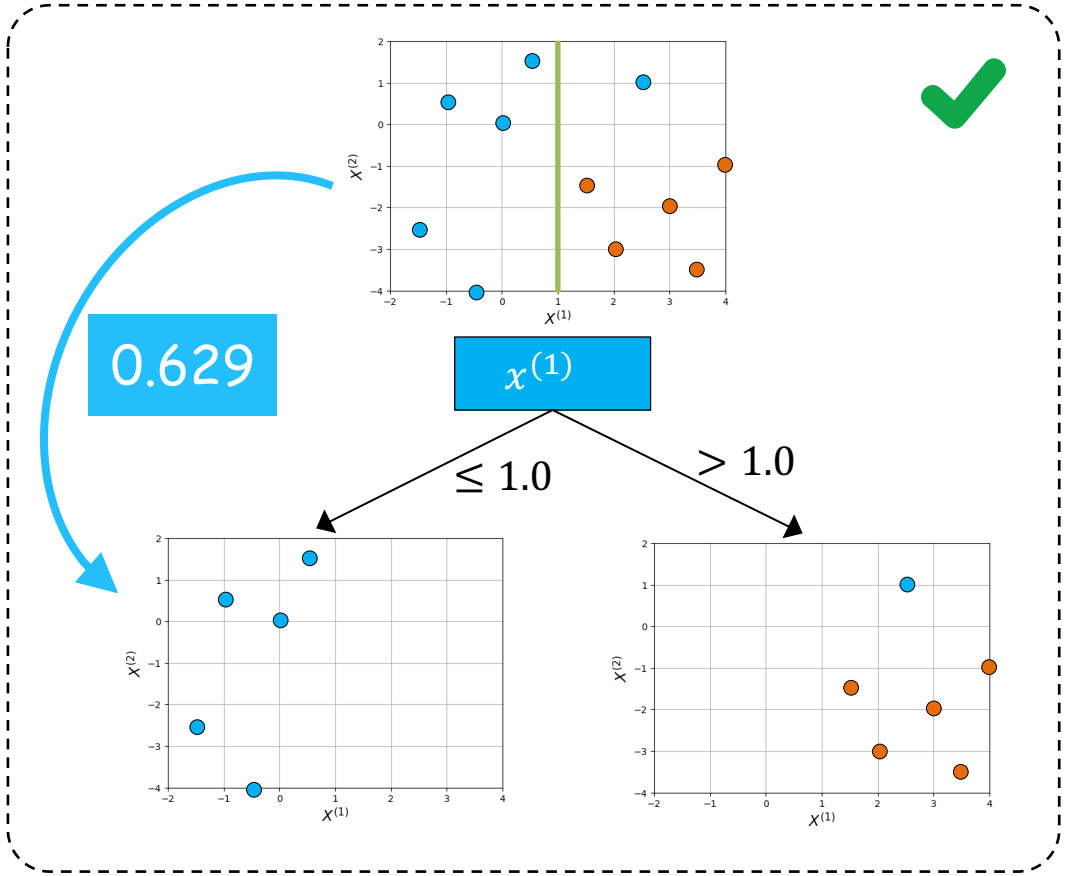
$= \frac{8}{11} \times 0.964 + \frac{3}{11} \times 0.00$

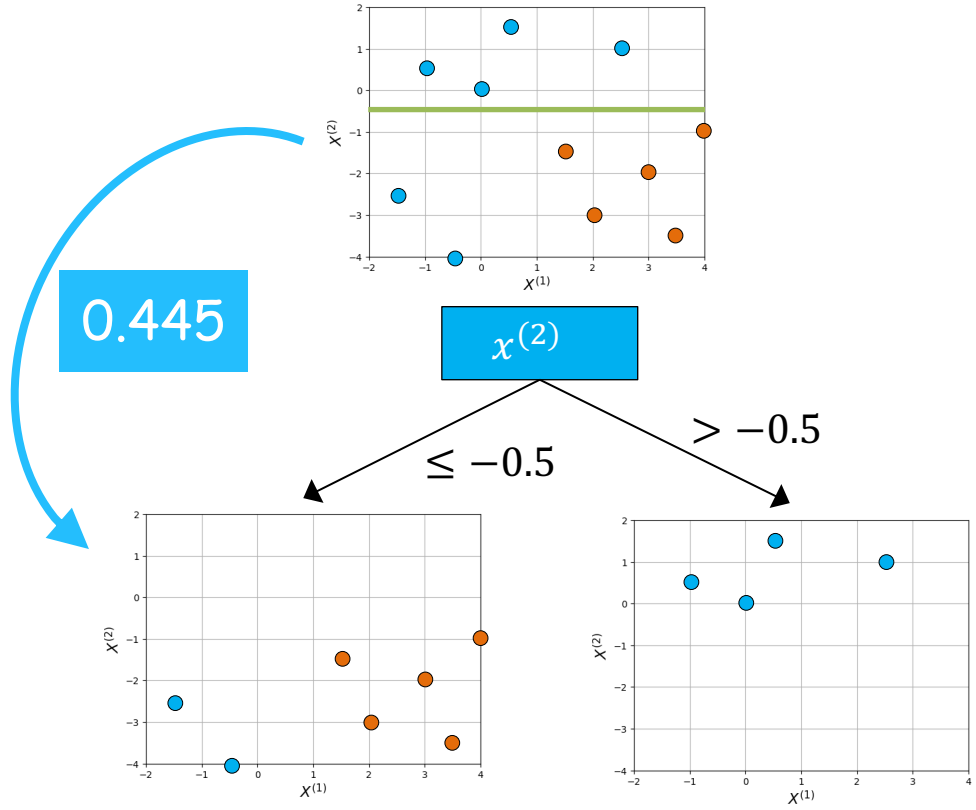
$= 0.701$

Unique Values of $x^{(2)}$ : `[-4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, 0.0, 0.5, 1.0, 1.5]`

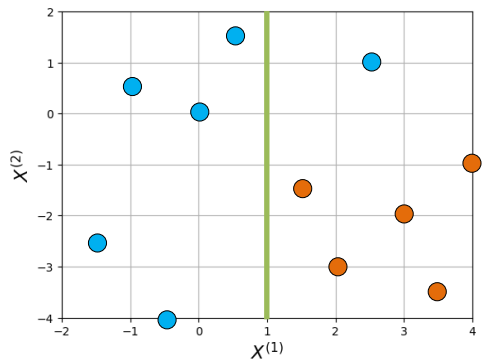
Candidate Splits : `[-3.75, -3.25, -2.75, -2.25, -1.75, -1.25, -0.5, 0.25, 0.75, 1.25]`

# Decision Tree Learning for Numerical Features (5 of 7)

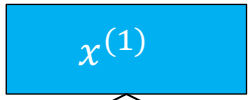


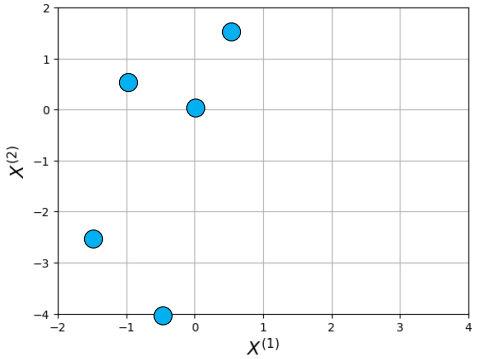
# Decision Tree Learning for Numerical Features (6 of 7)

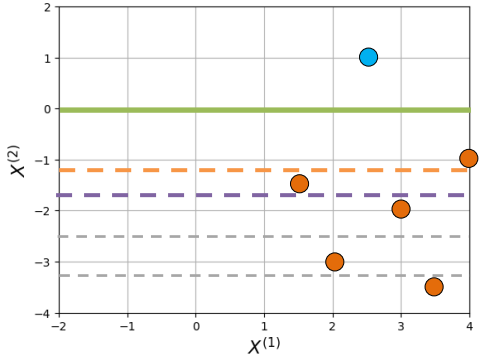



$E = 0.355$




$\leq 1.0$ $\quad > 1.0$




$t_5 = 0.0$ : $IG = 0.355 - 0.0 = 0.355$

$t_3 = -1.75$ : $E = \frac{3}{6} \times -\left(\frac{3}{3}\log_2\frac{3}{3}\right) + \frac{3}{6} \times -\left(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right)$

$= \frac{3}{6} \times 0.0 + \frac{3}{6} \times 0.918$

$= 0.459$

$t_4 = -1.025$ : $E = \frac{4}{6} \times -\left(\frac{4}{4}\log_2\frac{4}{4}\right) + \frac{2}{6} \times -\left(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right)$

$= \frac{4}{6} \times 0.0 + \frac{2}{6} \times 0.0$

$= 0.333$

$t_5 = 0.0$ : $E = \frac{5}{6} \times -\left(\frac{5}{5}\log_2\frac{5}{5}\right) + \frac{1}{6} \times -\left(\frac{1}{1}\log_2\frac{1}{1}\right)$

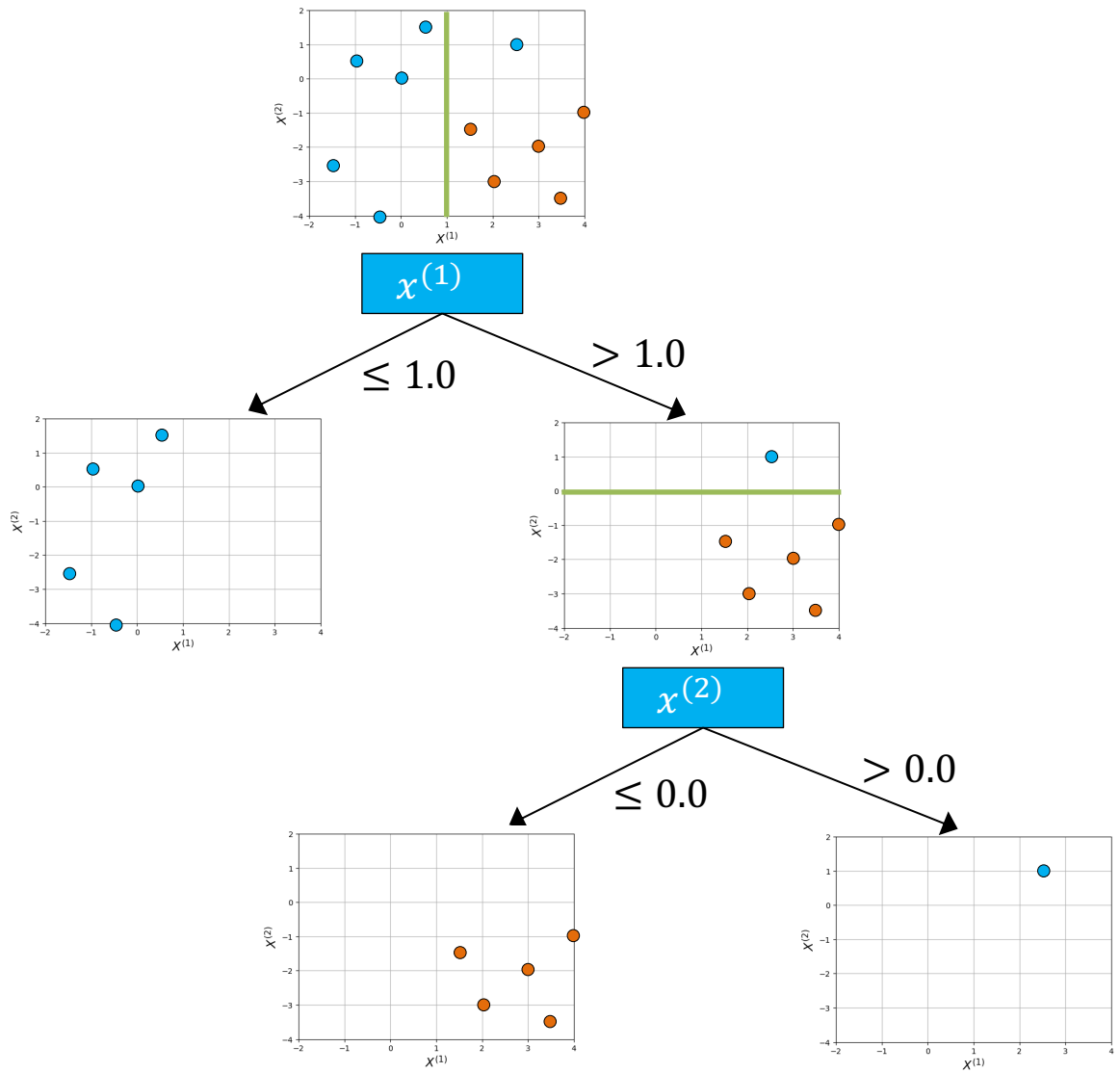
$= \frac{5}{6} \times 0.0 + \frac{1}{6} \times 0.0$

$= 0.0$

Unique Values of $x^{(2)}$ : `[-3.5, -3.0, -2.0, -1.5, -1.0, 1.0]`

Candidate Splits : `[-3.25, -2.5, -1.75, -1.25, 0.0]`

# Decision Tree Learning for Numerical Features (7 of 7)




0: 1:

```
IF x(1) ⩽ 0.00
    → predict 0
ELSE  # x(1) > 0.0
    IF x(2) ⩽ 0.0
        → predict 1
    ELSE
        → predict 0
```

# Pseudocode for Decision Tree Classifier

```
Function DecisionTree_Classify_Numeric(new_point, data, features, labels)
Begin
    Step 1: Stopping Criteria
        1.1 If all labels in `labels` are identical OR `features` is empty Then
            Return majority_label(labels)

    Step 2: Compute Parent Entropy
        base_entropy ← Entropy(labels)

    Step 3: Find Best Numeric Split by Information Gain
        best_gain      ← −∞
        best_feature   ← None
        best_threshold ← None

        For each feature F in features Do
            3.1 Generate candidate thresholds
                unique_vals    ← sorted unique values of data[F]
                thresholds     ← []
                For i from 1 to length(unique_vals)-1 Do
                    thresholds.append( (unique_vals[i] + unique_vals[i-1]) / 2 )

            3.2 Evaluate each threshold t
                For each t in thresholds Do
                    left_labels  ← labels of rows where data[F] ≤ t
                    right_labels ← labels of rows where data[F] >  t
                    weight_left  ← |left_labels|  ÷ |labels|
                    weight_right ← |right_labels| ÷ |labels|
                    child_entropy ← weight_left × Entropy(left_labels)
                                  + weight_right × Entropy(right_labels)
                    info_gain     ← base_entropy − child_entropy

                    If info_gain > best_gain Then
                        best_gain      ← info_gain
                        best_feature   ← F
                        best_threshold ← t
                    End If
                End For
        End For
```

```
    Step 4: Split on best_feature at best_threshold and Recurse
        value ← new_point[best_feature]

        If value ≤ best_threshold Then
            subset_data   ← rows in data where data[best_feature] ≤ best_threshold
            subset_labels ← corresponding labels
        Else
            subset_data   ← rows in data where data[best_feature] > best_threshold
            subset_labels ← corresponding labels
        End If

        If subset_data is empty Then
            Return majority_label(labels)   // fallback if no examples
        Else
            remaining_features ← features minus {best_feature}
            Return DecisionTree_Classify_Numeric(
                new_point,
                subset_data,
                remaining_features,
                subset_labels
            )
        End If
End
```

# Café Example Dataset (Revisit)

| Seating | Drink | TimeOfDay | Dessert |
|---|---|---|---|
| Indoor | Coffee | Morning | Muffin |
| Indoor | Coffee | Morning | Muffin |
| Indoor | Coffee | Morning | Muffin |
| Indoor | Coffee | Morning | Muffin |
| Indoor | Coffee | Morning | Muffin |
| Indoor | Coffee | Afternoon | Muffin |
| Indoor | Coffee | Afternoon | Muffin |
| Indoor | Coffee | Afternoon | Muffin |
| Indoor | Coffee | Evening | Muffin |
| Indoor | Coffee | Evening | Muffin |
| Indoor | Tea | Morning | Cake |
| Indoor | Tea | Afternoon | Cake |
| Indoor | Tea | Afternoon | Cake |

| Seating | Drink | TimeOfDay | Dessert |
|---|---|---|---|
| Indoor | Tea | Evening | Cake |
| Indoor | Tea | Evening | Cake |
| Indoor | Tea | Evening | Cake |
| Outdoor | Coffee | Morning | Cookie |
| Outdoor | Coffee | Morning | Cookie |
| Outdoor | Coffee | Afternoon | Cookie |
| Outdoor | Coffee | Afternoon | Cookie |
| Outdoor | Coffee | Evening | Cookie |
| Outdoor | Coffee | Evening | Cookie |
| Outdoor | Tea | Afternoon | Cookie |
| Outdoor | Tea | Evening | Cookie |
| Outdoor | Tea | Evening | Cookie |

*Q: Do we really need to learn differently with categorical features?*

# Label Encoder

| Seating | Drink | TimeOfDay | Dessert |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 2 | 0 |
| 0 | 0 | 2 | 0 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 2 |
| 0 | 1 | 1 | 2 |
| 0 | 1 | 2 | 2 |
| 0 | 1 | 2 | 2 |
| 0 | 1 | 2 | 2 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |

| Seating | Drink | TimeOfDay | Dessert |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
| 1 | 0 | 2 | 1 |
| 1 | 0 | 2 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 1 |
| 1 | 1 | 2 | 1 |

*Label encoding is a simple way to convert categorical (non-numeric) data into numeric form by assigning each unique category an integer label.*

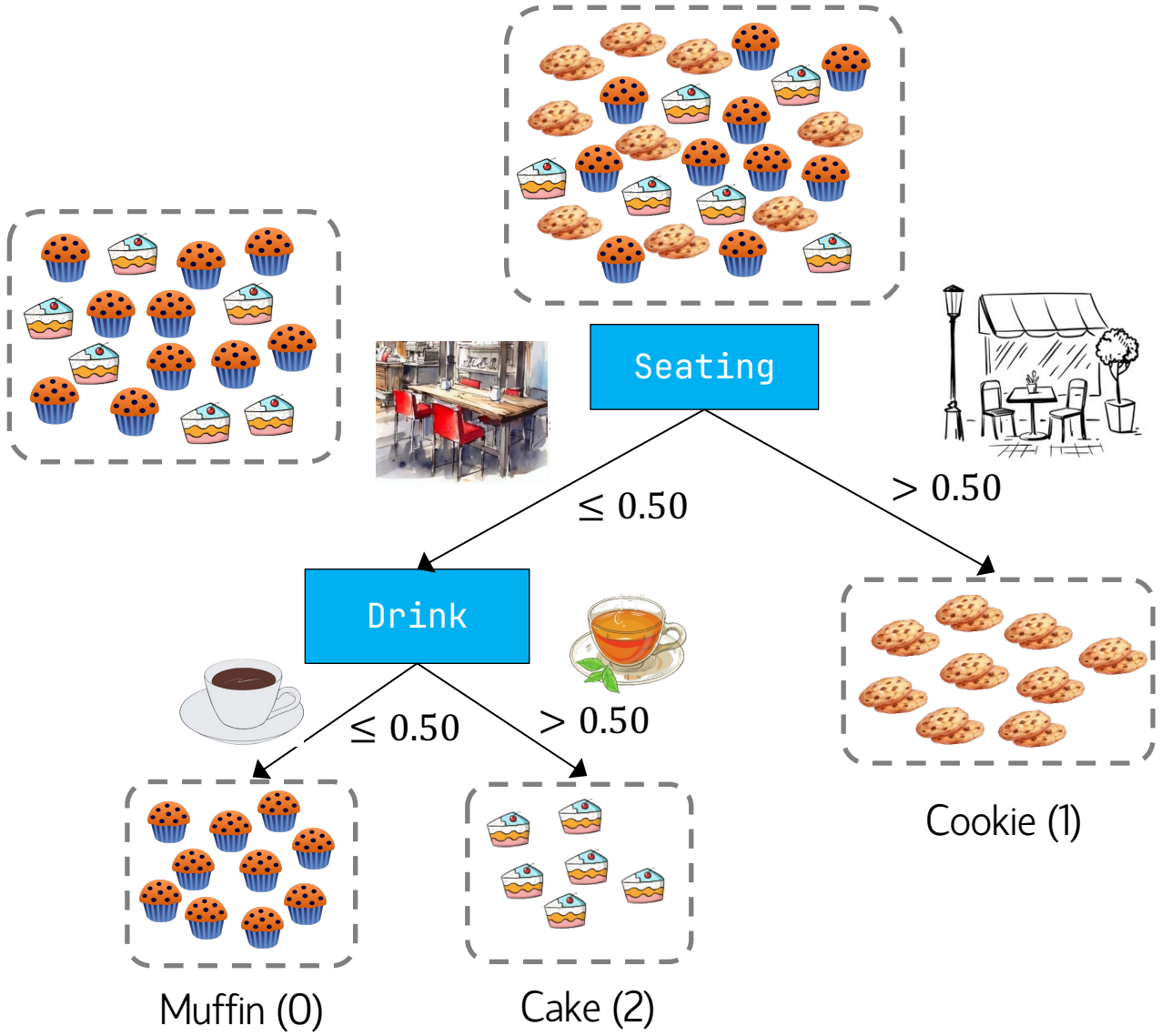
Seating
0: Indoor
1: Outdoor

Drink
0: Coffee
1: Tea

TimeOfDay
0: Morning
1: Afternoon
2: Evening

Dessert
0: Muffin
1: Cookie
2: Cake

# Decision Tree with Encoded Labels




| Seating | Drink |
|---|---|
| 0: Indoor<br>1: Outdoor | 0: Coffee<br>1: Tea |
| **TimeOfDay** | **Dessert** |
| 0: Morning<br>1: Afternoon<br>2: Evening | 0: Muffin<br>1: Cookie<br>2: Cake |

```
IF Seating > 0.5
    → predict 1
ELSE  # Seating ⩽0.5
   IF Drink ⩽ 0.5
       → predict 0
   ELSE
       → predict 2
```

# Python Code Snippet

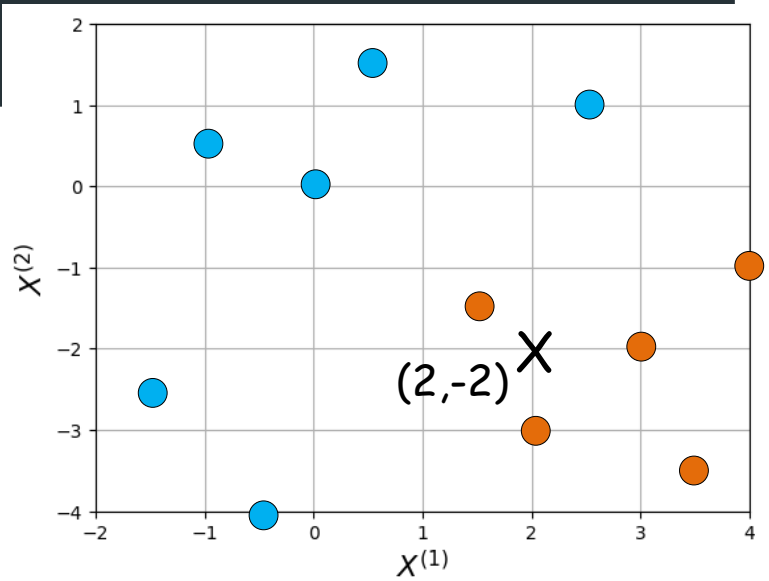
```
import math
from collections import Counter
import numpy as np

def entropy(labels):
    total = len(labels)
    counts = Counter(labels)
    ent = 0.0
    for count in counts.values():
        p = count / total
        ent -= p * math.log2(p)
    return ent


def decision_tree_classify_numeric(new_point, X, y, feature_idx=None):
    if feature_idx is None:
        feature_idx = list(range(X.shape[1]))

    # Step 1: Stopping criteria
    if len(set(y)) == 1 or not feature_idx:
        return Counter(y).most_common(1)[0][0]           # majority label

    # Step 2: Parent entropy
    base_entropy = entropy(y)

    # Step 3: Best split search
    best_gain = -1.0
    best_feature = None
    best_thresh = None

    for f in feature_idx:
        # 3.1 candidate thresholds (mid-points between sorted unique values)
        vals = np.unique(X[:, f])
        if vals.size < 2:
            continue
        thresholds = (vals[:-1] + vals[1:]) / 2.0

        for t in thresholds:
            left_mask = X[:, f] <= t
            right_mask = ~left_mask
            if not left_mask.any() or not right_mask.any():
                continue

            left_y, right_y = y[left_mask], y[right_mask]
```

```
...cont...

            # 3.2 weighted child entropy → information gain
            w_left = len(left_y) / len(y)
            w_right = 1 - w_left
            gain = base_entropy - (w_left * entropy(left_y) +
                                   w_right * entropy(right_y))

            if gain > best_gain:
                best_gain, best_feature, best_thresh = gain, f, t

    # No valid split → leaf
    if best_feature is None:
        return Counter(y).most_common(1)[0][0]

    # Step 4: Split & recurse down the branch matching new_point
    if new_point[best_feature] <= best_thresh:
        mask = X[:, best_feature] <= best_thresh
    else:
        mask = X[:, best_feature] > best_thresh

    return decision_tree_classify_numeric(
        new_point,
        X[mask],
        y[mask],
        [f for f in feature_idx if f != best_feature]
    )
```

# Usage Example

```
# Example Usage

# Define training data and labels
X = np.array([[-0.5, -4.0], [-1.5, -2.5], [0.0, 0.0], [-1.0, 0.5], [0.5, 1.5], [2.5, 1.0],
              [3.5, -3.5], [2.0, -3.0], [3.0, -2.0], [1.5, -1.5], [4.0, -1.0]])
y = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1])  # Two classes: 0 and 1

# Define a new point to classify
new_point = np.array([2.0, -2.0])

# Predict the class for the new point
predicted_class = decision_tree_classify_numeric(new_point, X, y):

# Output
print(f"New Point: {new_point}")
print(f"Predicted Class: {predicted_class}")
```

```
New Point: [2. -2.]

Predicted Class: 1
```



# Step 1: Zero Impurity

```
# Step 1: Stopping criteria
if len(set(y)) == 1 or not feature_idx:
    return Counter(y).most_common(1)[0][0]          # majority label
```

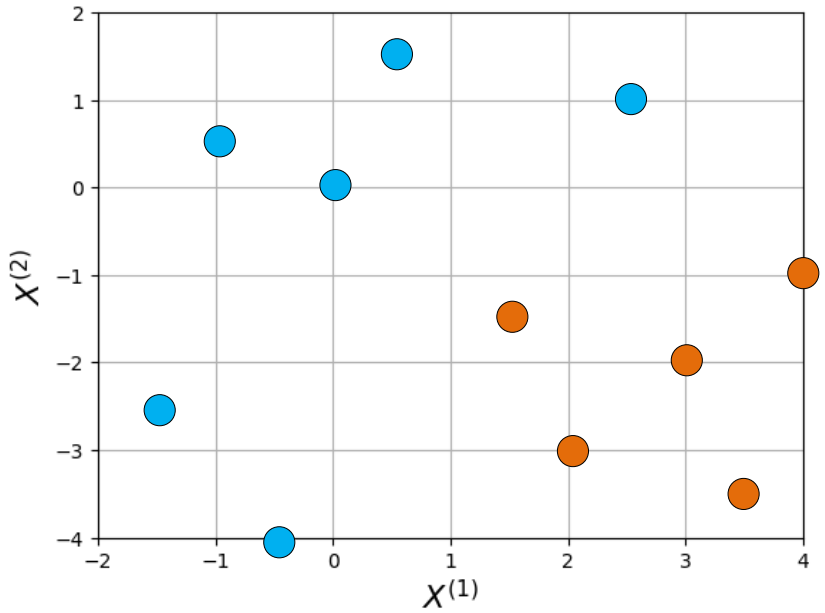
$$y = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^T$$

$$\{0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1\} \Rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

*We have 2 classes, hence we skip the return.*

# Step 2: Parent Entropy

```
# Step 2: Parent entropy
base_entropy = entropy(y)
```



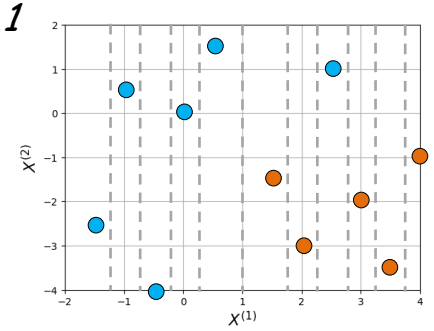

$$E = -\left(\frac{6}{11}\log_2\left(\frac{6}{11}\right) + \frac{5}{11}\log_2\left(\frac{5}{11}\right)\right)$$

$$= 0.994$$

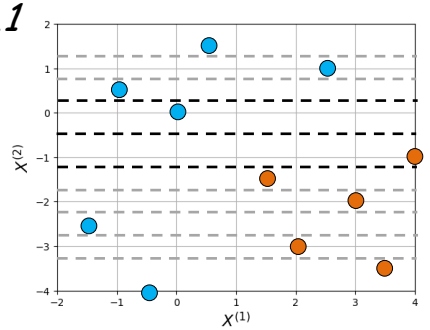
# Step 3: Feature & Split with Highest Information Gain

```
for f in feature_idx:
    # 3.1 candidate thresholds (mid-points between sorted unique values)
    vals = np.unique(X[:, f])
    if vals.size < 2:
        continue
    thresholds = (vals[:-1] + vals[1:]) / 2.0

    for t in thresholds:
        left_mask = X[:, f] ⩽ t
        right_mask = ~left_mask
        if not left_mask.any() or not right_mask.any():
            continue

        left_y, right_y = y[left_mask], y[right_mask]

        # 3.2 weighted child entropy → information gain
        w_left = len(left_y) / len(y)
        w_right = 1 - w_left
        gain = base_entropy - (w_left * entropy(left_y) +
                               w_right * entropy(right_y))

        if gain > best_gain:
            best_gain, best_feature, best_thresh = gain, f, t
```

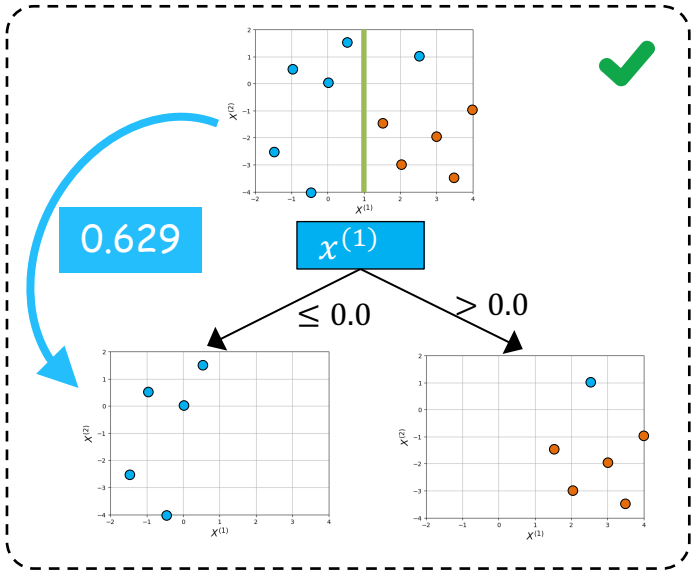
$x^{(1)}$:3.1



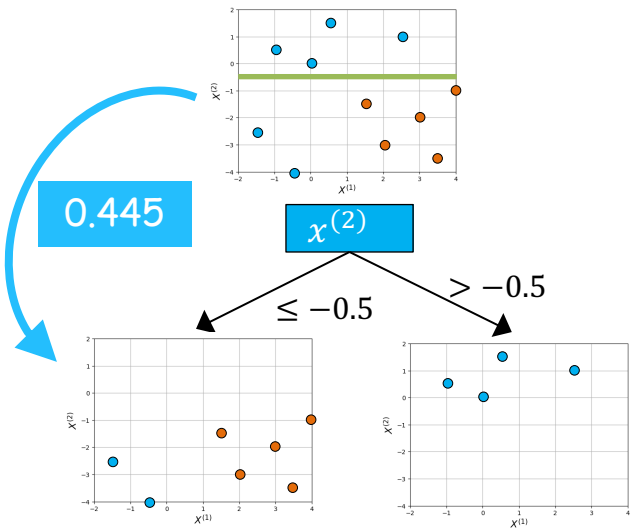

$x^{(2)}$:3.1



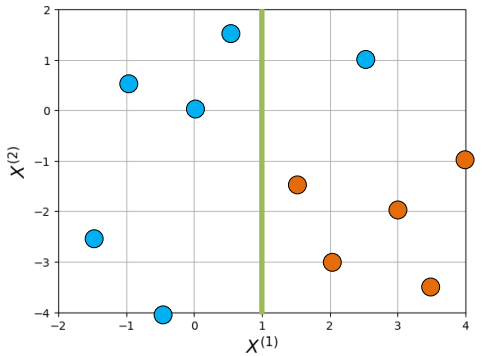

$x^{(1)}$:3.2




$x^{(2)}$:3.2



# Step 1 & 4: Zero Impurity & Recurse Down

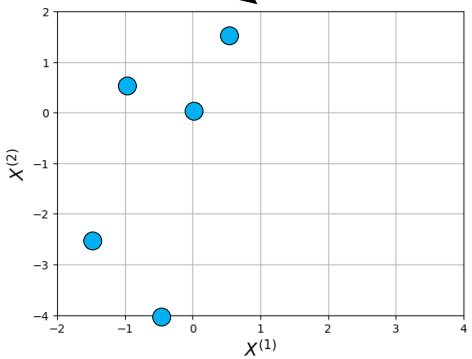


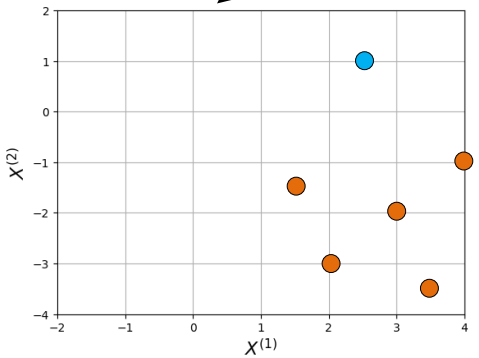

$E = 0.355$

$x^{(1)}$

$\leq 0.0$

$> 0.0$

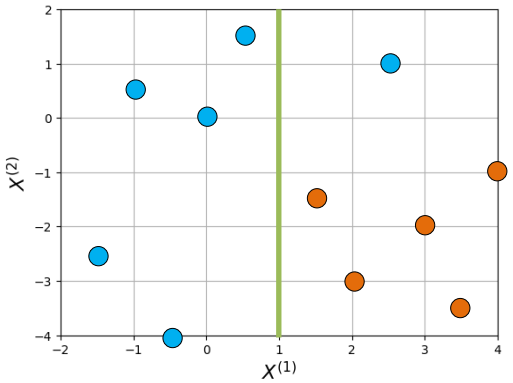



```
# Step 4: Split & recurse down the branch matching new_point
    if new_point[best_feature] ⩽ best_thresh:
        mask = X[:, best_feature] ⩽ best_thresh
    else:
        mask = X[:, best_feature] > best_thresh

return decision_tree_classify_numeric(
        new_point,
        X[mask],
        y[mask],
        [f for f in feature_idx if f ≠ best_feature]
    )
```

```
# Step 1: Stopping criteria
if len(set(y)) == 1 or not feature_idx:
    return Counter(y).most_common(1)[0][0]
```

# Step 2: Parent Entropy

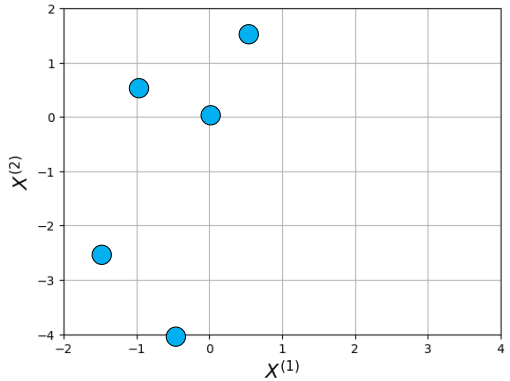


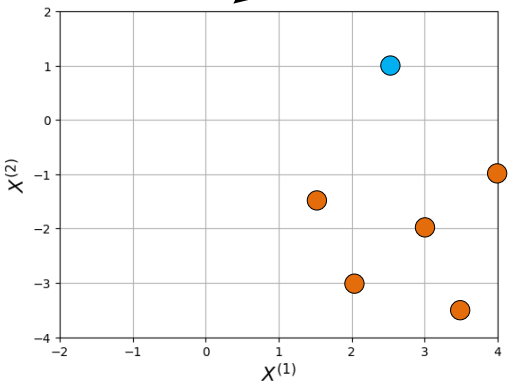

$E = 0.355$

$x^{(1)}$

$\leq 0.0$

$> 0.0$



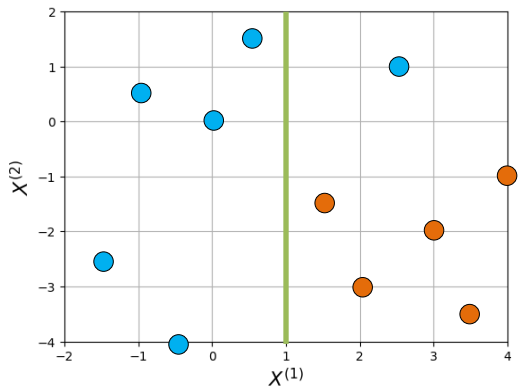

```
# Step 2: Parent entropy
base_entropy = entropy(y)
```

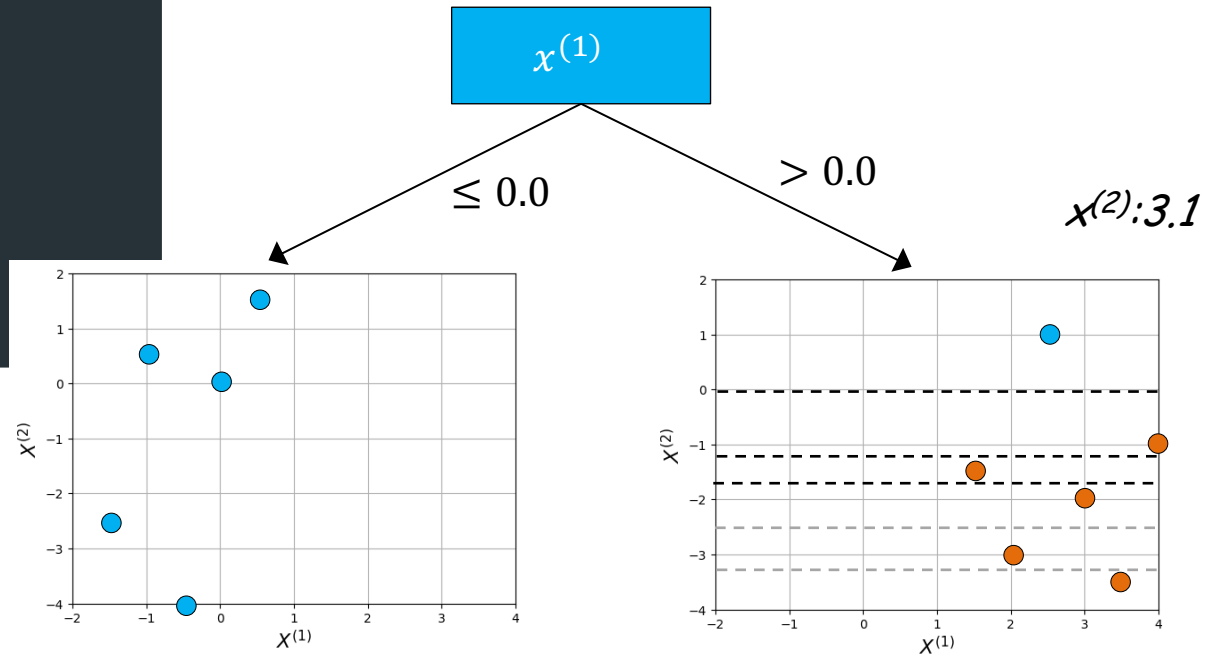
$$E = -\left(\frac{5}{6}\log_2\frac{5}{6} + \frac{1}{6}\log_2\frac{1}{6}\right)$$

$$= 0.355$$

# Step 3.1: Candidate Splits

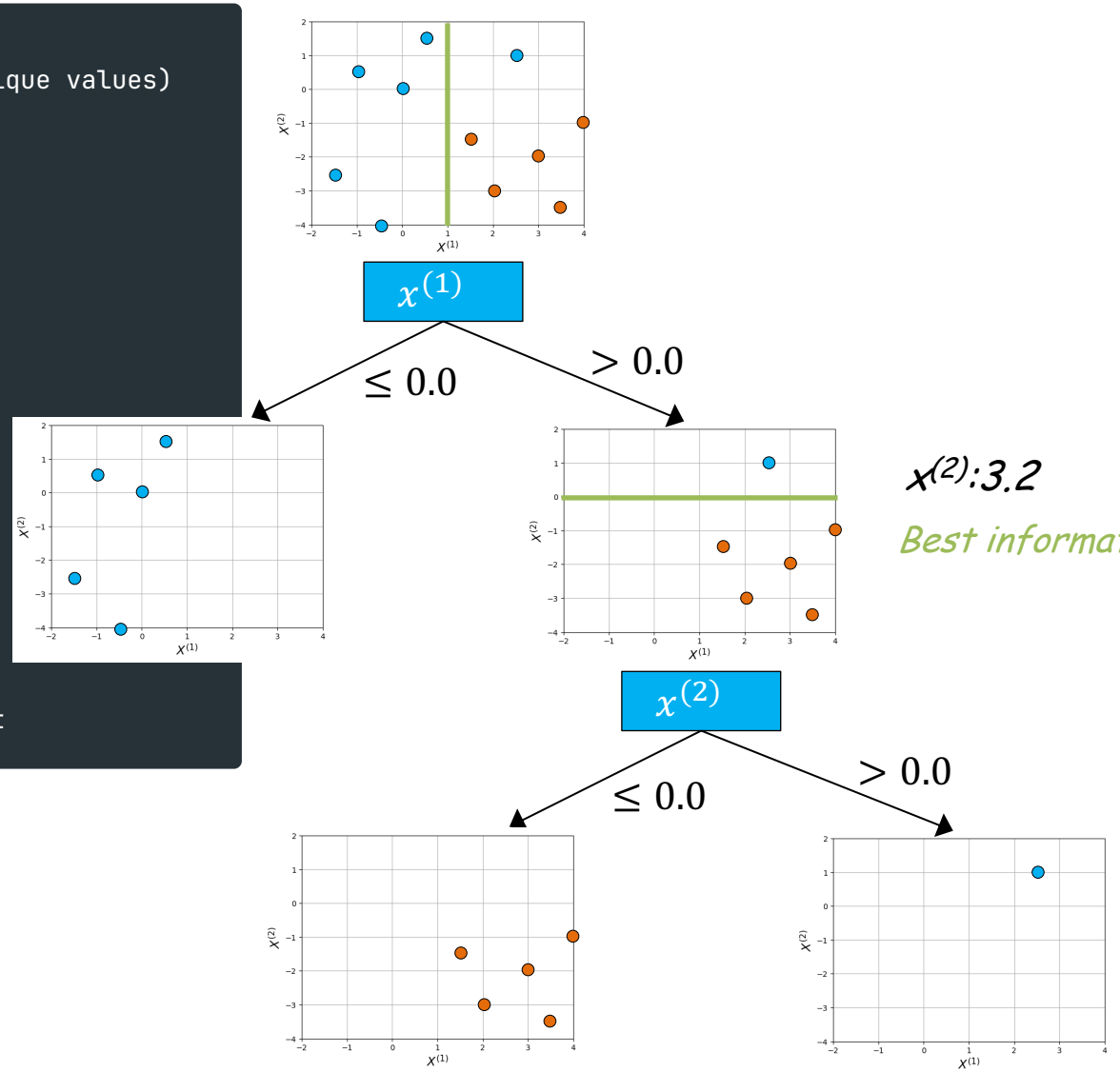
```
for f in feature_idx:
    # 3.1 candidate thresholds (mid-points between sorted unique values)
    vals = np.unique(X[:, f])
    if vals.size < 2:
        continue
    thresholds = (vals[:-1] + vals[1:]) / 2.0

    for t in thresholds:
        left_mask = X[:, f] ⩽ t
        right_mask = ~left_mask
        if not left_mask.any() or not right_mask.any():
            continue

        left_y, right_y = y[left_mask], y[right_mask]

        # 3.2 weighted child entropy → information gain
        w_left = len(left_y) / len(y)
        w_right = 1 - w_left
        gain = base_entropy - (w_left * entropy(left_y) +
                               w_right * entropy(right_y))

        if gain > best_gain:
            best_gain, best_feature, best_thresh = gain, f, t
```



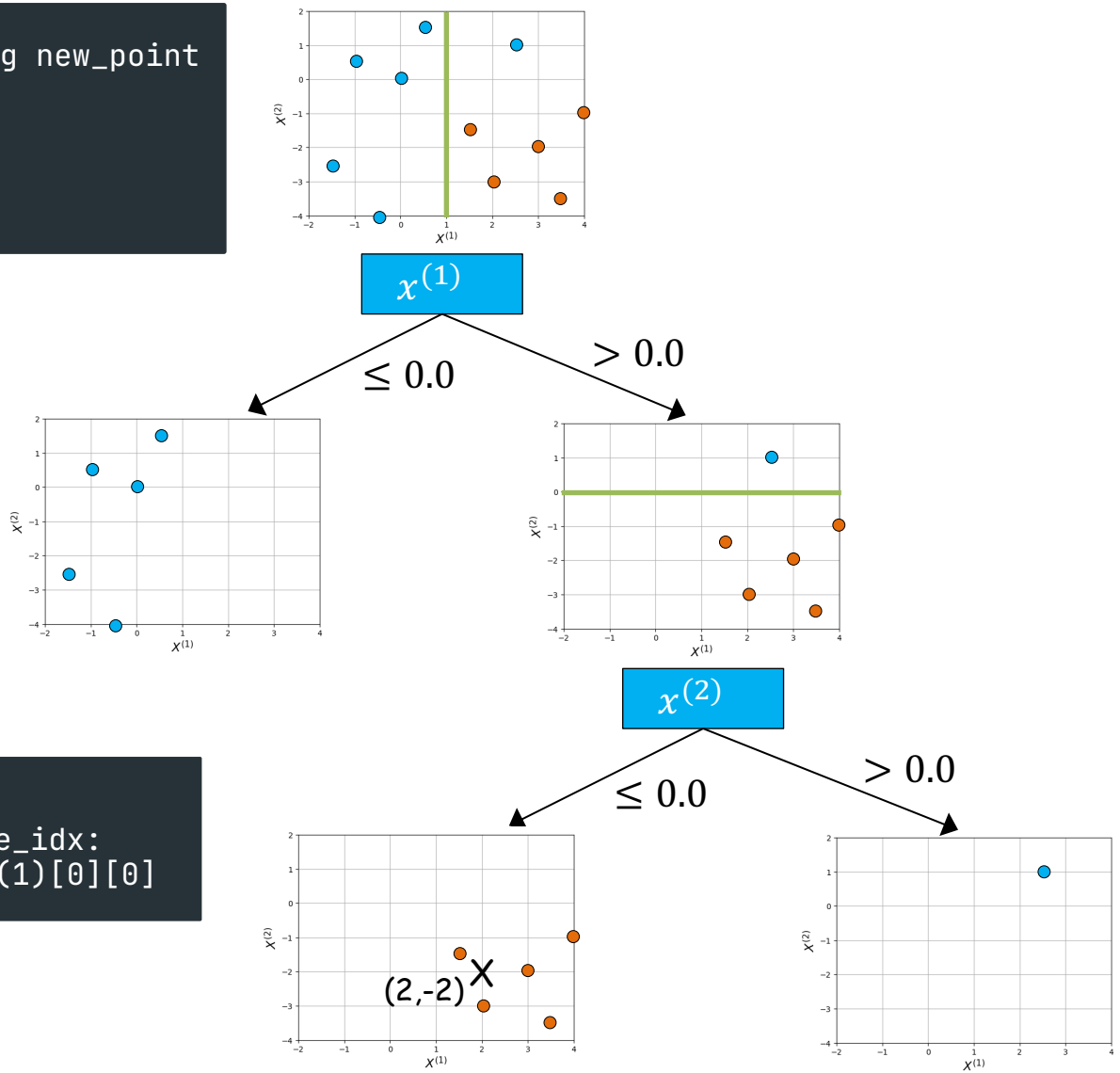
# Step 3.2: Candidate Split with Highest Information Gain

```
for f in feature_idx:
    # 3.1 candidate thresholds (mid-points between sorted unique values)
    vals = np.unique(X[:, f])
    if vals.size < 2:
        continue
    thresholds = (vals[:-1] + vals[1:]) / 2.0

    for t in thresholds:
        left_mask = X[:, f] ⩽ t
        right_mask = ~left_mask
        if not left_mask.any() or not right_mask.any():
            continue

        left_y, right_y = y[left_mask], y[right_mask]

        # 3.2 weighted child entropy → information gain
        w_left = len(left_y) / len(y)
        w_right = 1 - w_left
        gain = base_entropy - (w_left * entropy(left_y) +
                               w_right * entropy(right_y))

        if gain > best_gain:
            best_gain, best_feature, best_thresh = gain, f, t
```



# Step 4 & 1: Matching Branch & Stopping Criteria

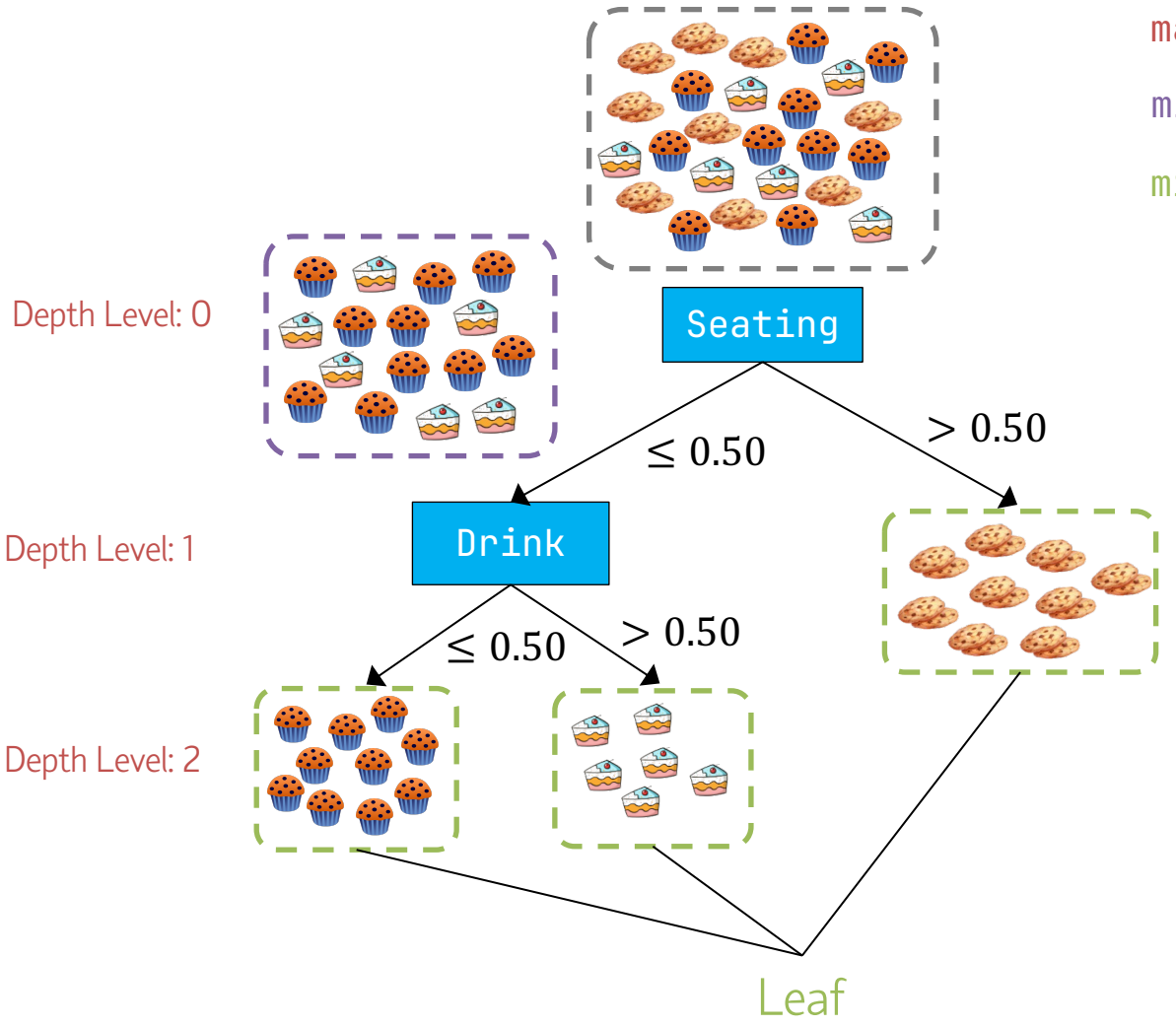
```
# Step 4: Split & recurse down the branch matching new_point
    if new_point[best_feature] ⩽ best_thresh:
        mask = X[:, best_feature] ⩽ best_thresh
    else:
        mask = X[:, best_feature] > best_thresh
```




```
# Step 1: Stopping criteria
if len(set(y)) == 1 or not feature_idx:
    return Counter(y).most_common(1)[0][0]
```

1

# Hyperparameters




`max_depth`: Maximum Tree Depth

`min_samples_split`: Minimum # Samples required to Create a Decision Rule

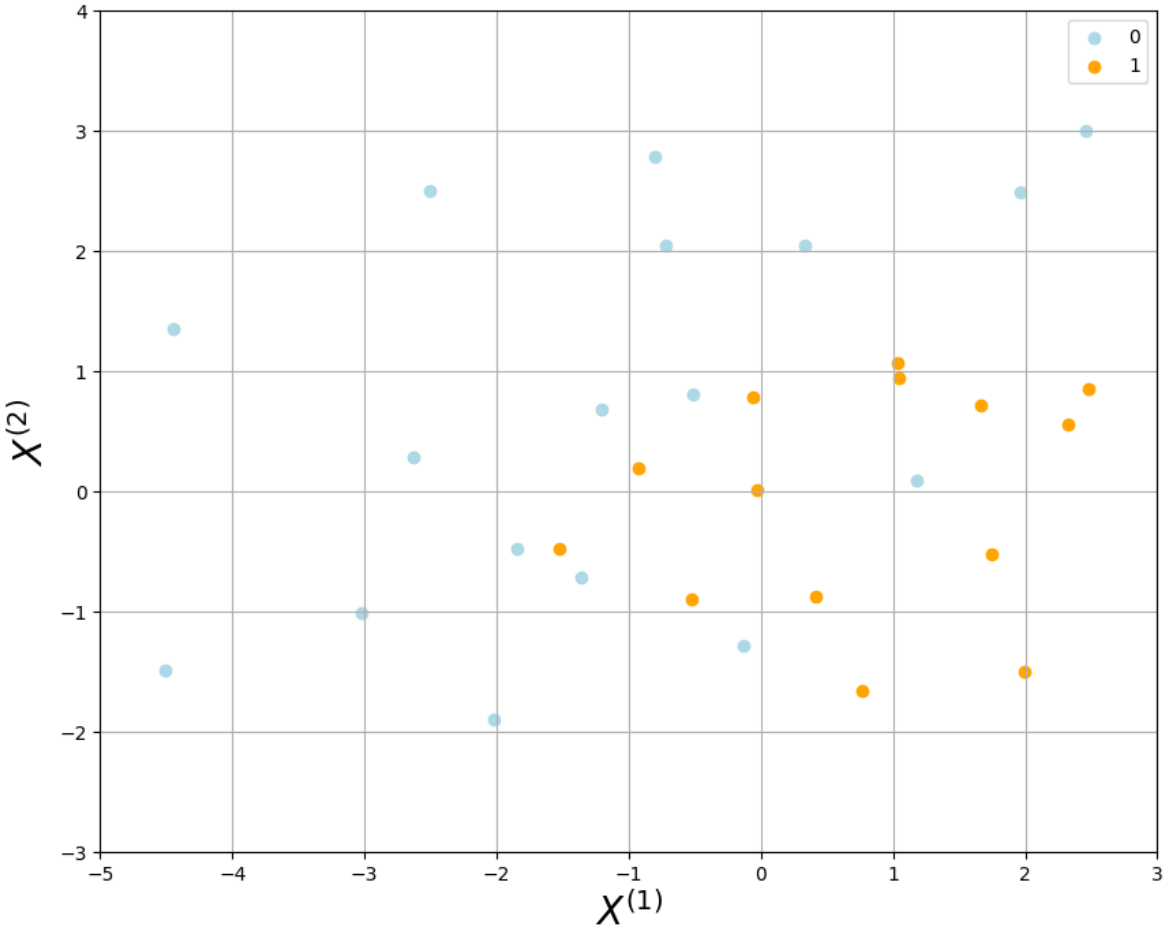
`min_samples_leaf`: Minimum # Samples required to be in a Leaf

`min_samples_split` → Tree Depth and # Samples in a Leaf

`max_depth` → # Samples in a Leaf

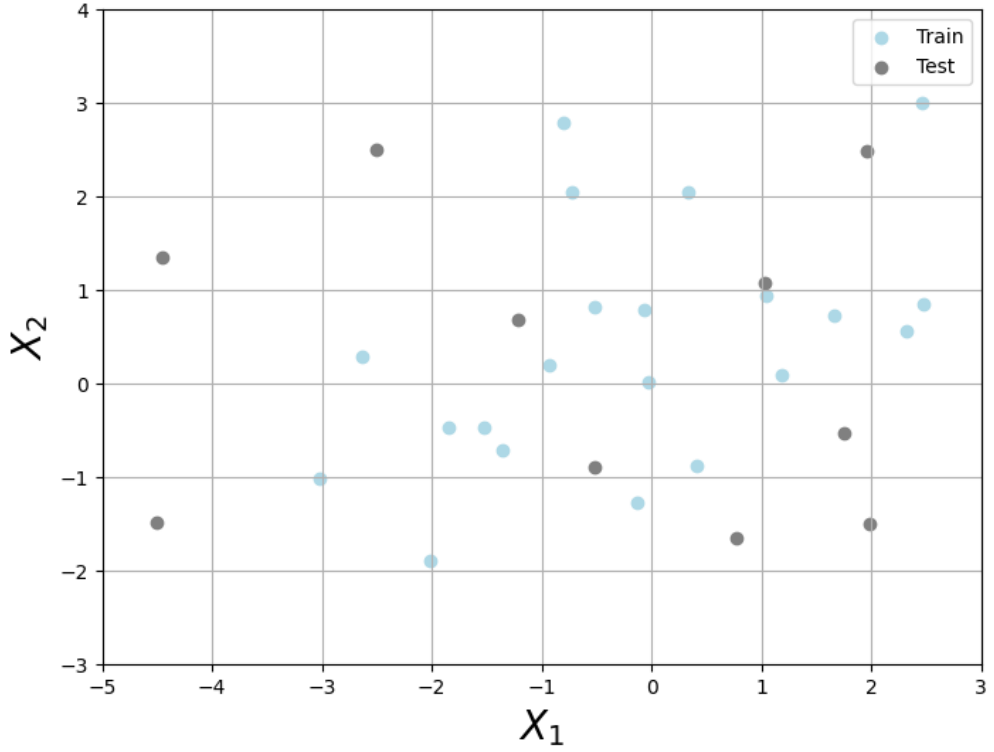
`min_samples_leaf` → Tree Depth

# Example Dataset



# Train, Validation and Test Datasets

```
from sklearn.model_selection import train_test_split

# First, split the data into train (70%) and test (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```





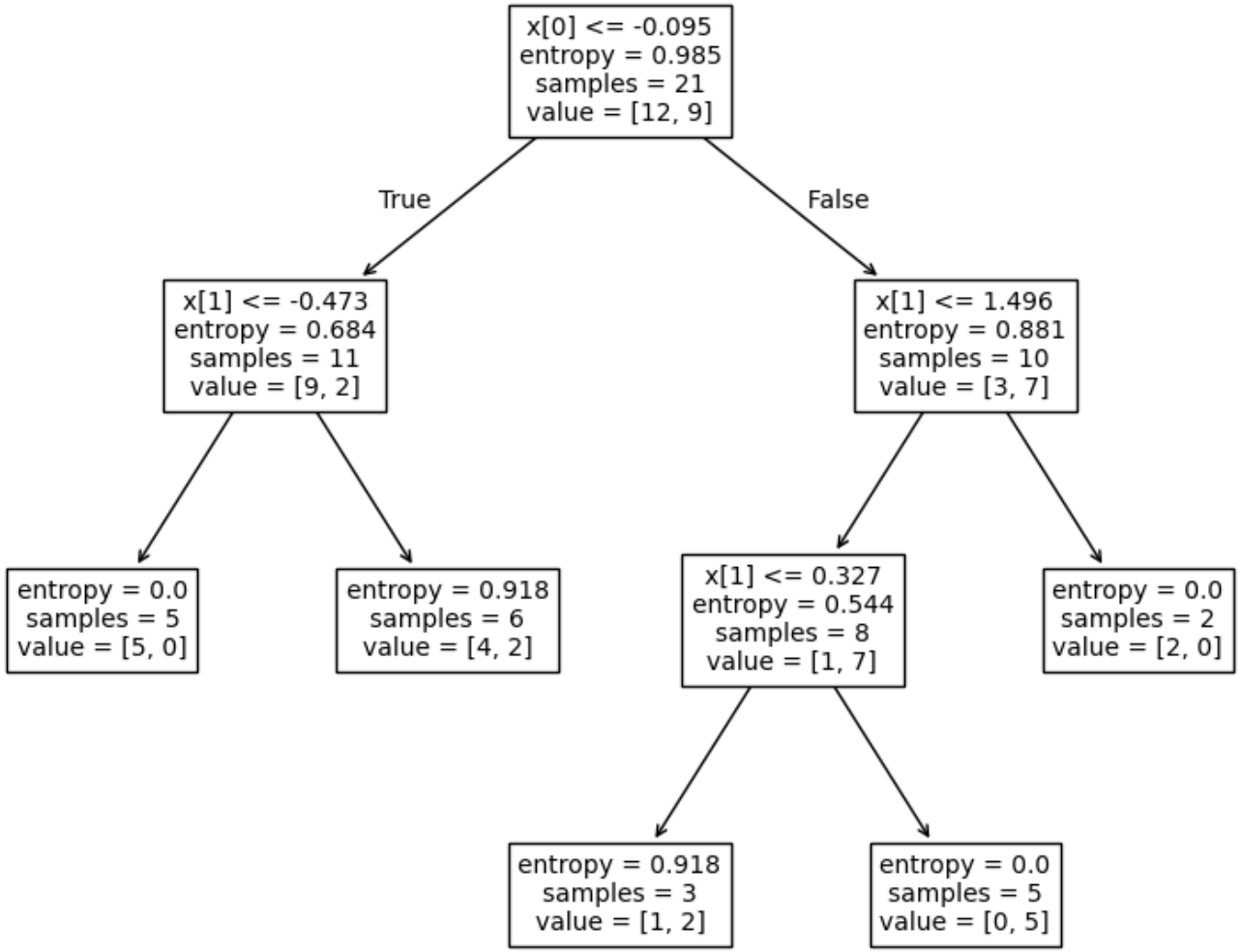
- Train:Test is typicaly either 0.8:0.2 or 0.7:0.3.
- **Test** dataset is a proxy of unseen data, and it will only be used in the final evaluation.
- **Train** dataset is further divided into k folds (e.g. 3 or 5).For each fold, the model is trained on k-1 folds and validated on the remaining fold.
- We use **K-Fold Cross-Validation** to fine-tune or optimise the ML model. Here, we find a set of hyper-parameters, e.g. **max_depth**, **min_samples_split** and **min_samples_leaf**, based on the average performance across folds.

# scikit-learn: Decision Tree Classifier

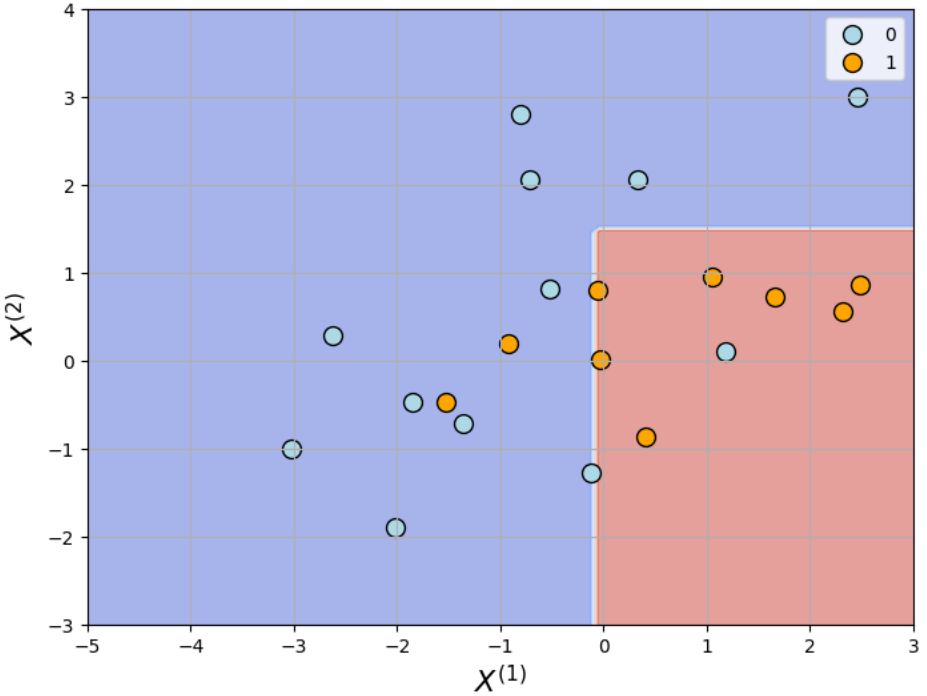
```
# Step 1: Create an instance of DecisionTreeClassifier
min_samples_split = 8
model = DecisionTreeClassifier(criterion='entropy', min_samples_split=min_samples_split random_state=204)

# Step 2: Define cross-validation strategy
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Step 3: Perform cross-validation on the training data
cv_scores = cross_val_score(model, X_train, y_train, cv=kf, scoring='accuracy')
print("Mean Cross-Validation Accuracy: {:.2f}".format(np.mean(cv_scores)))
```

```
Mean Cross-Validation Accuracy: 0.66
```

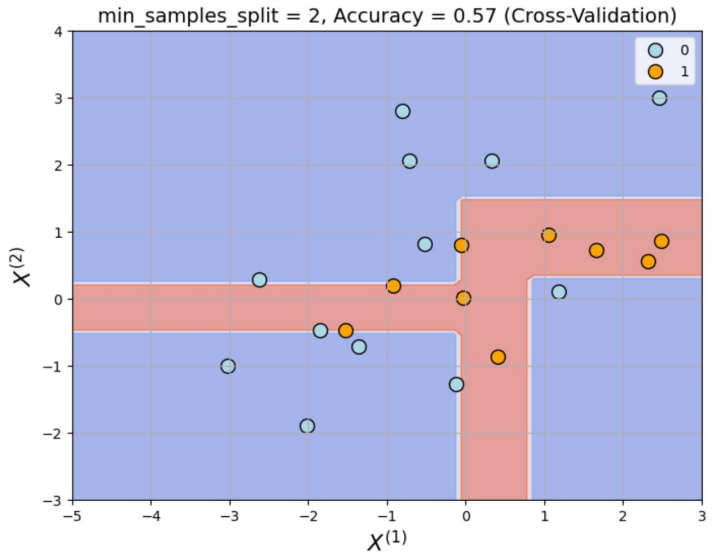
# Learned Decision Tree

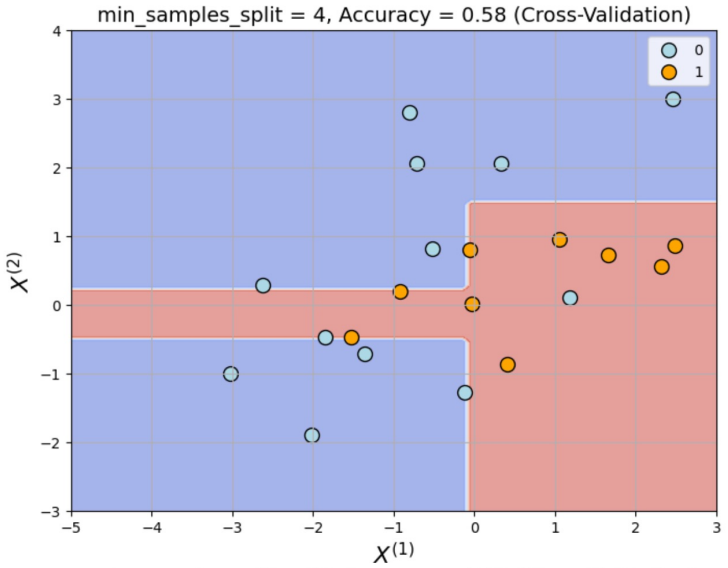


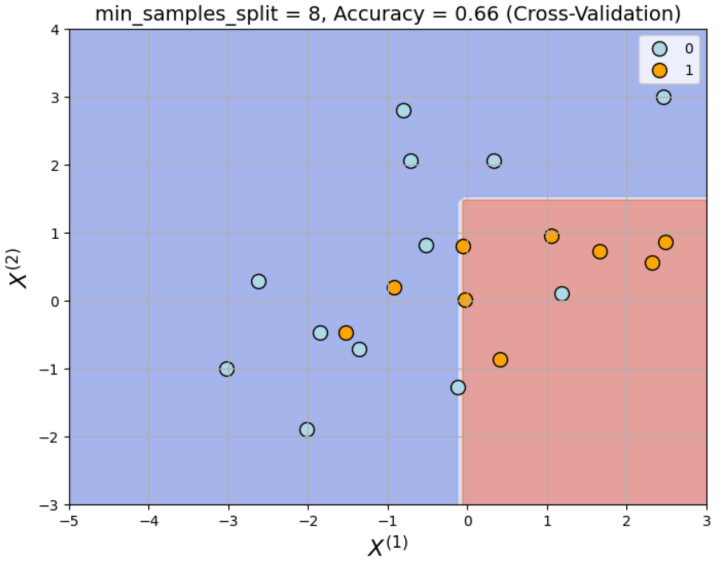

min_samples_split = 8 | Accuracy (CV) = 0.66

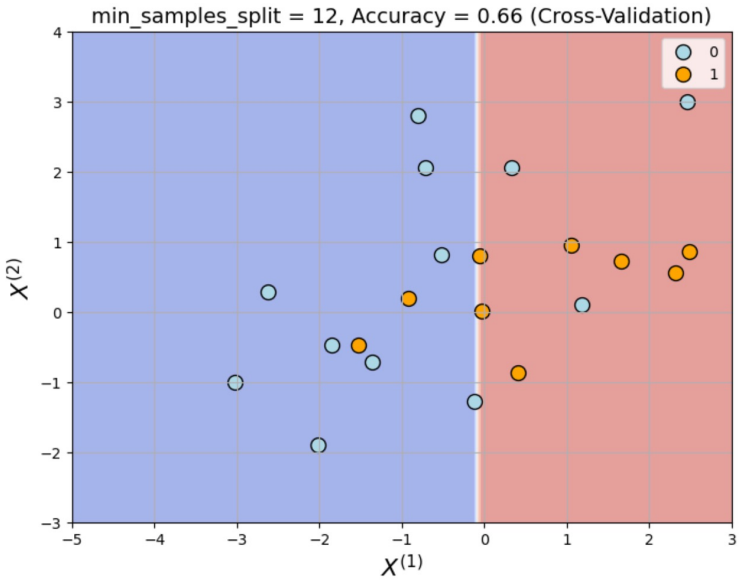


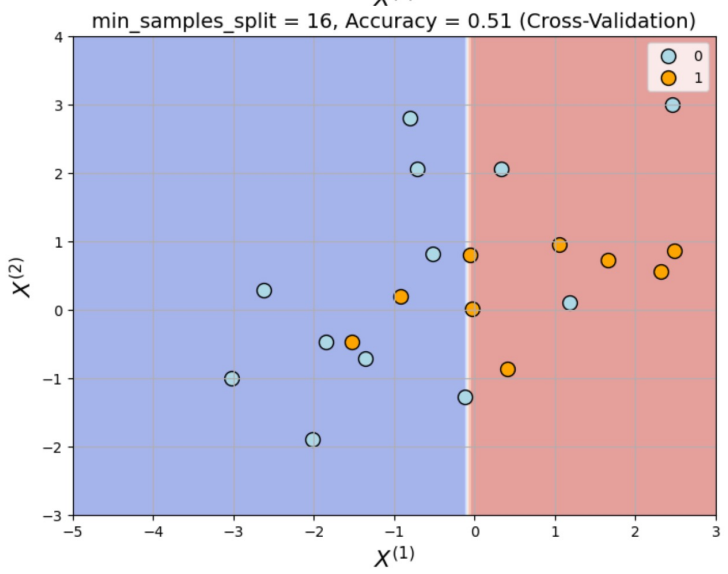
# Model Complexity

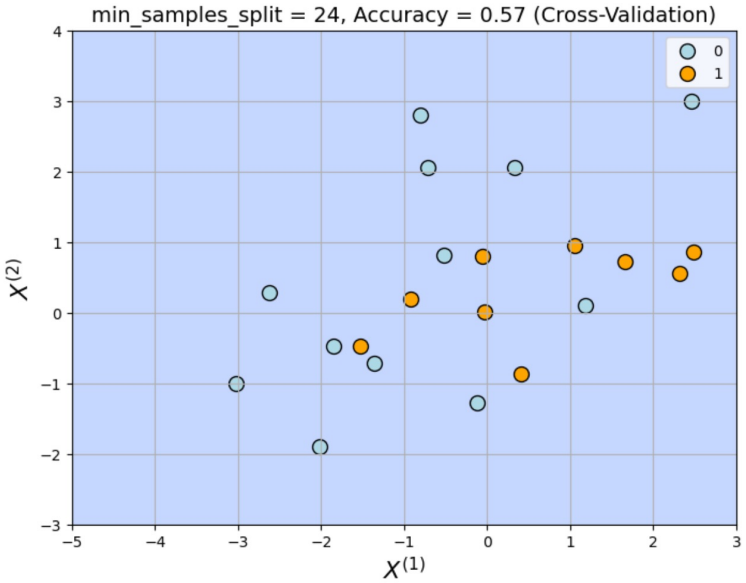


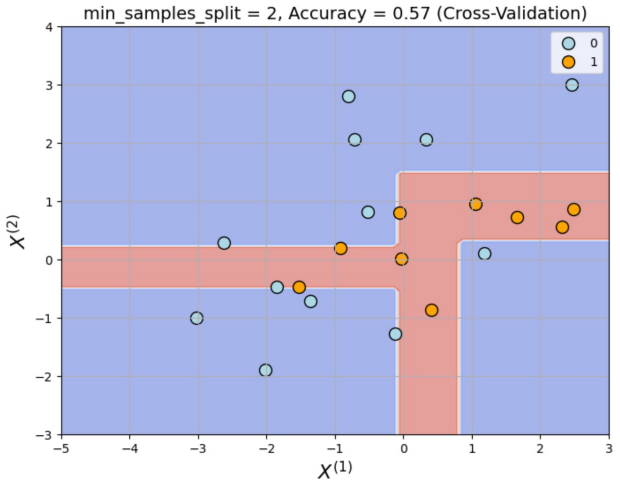



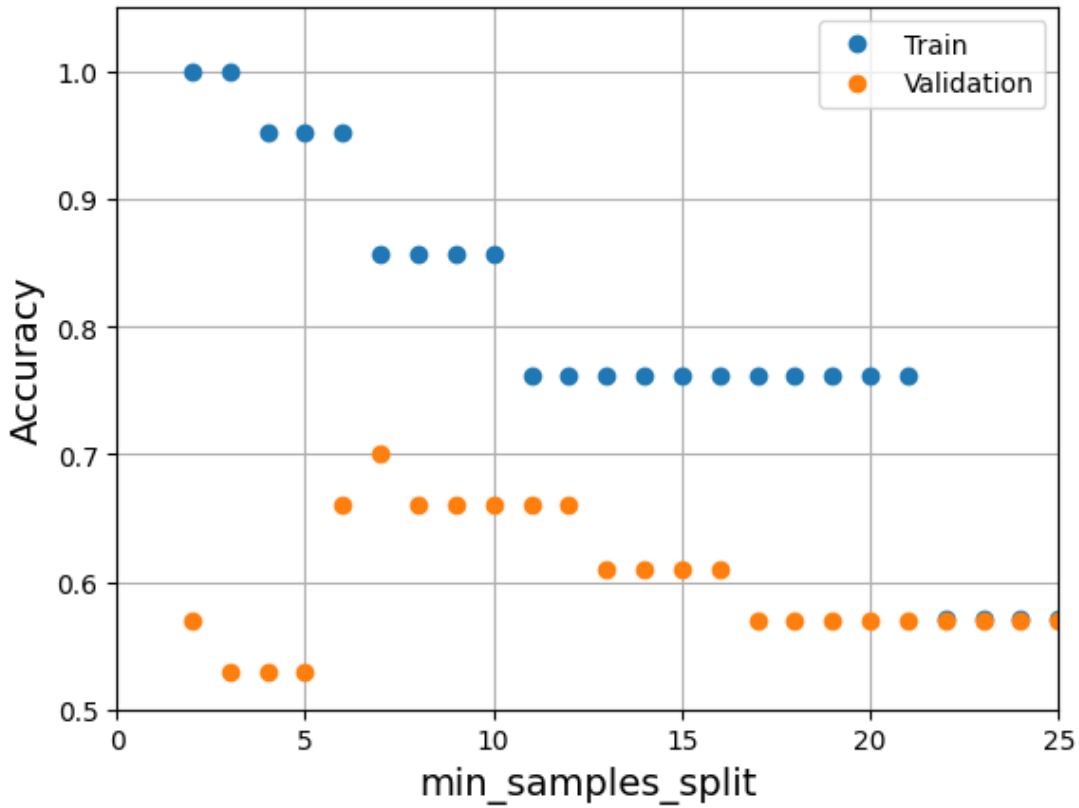



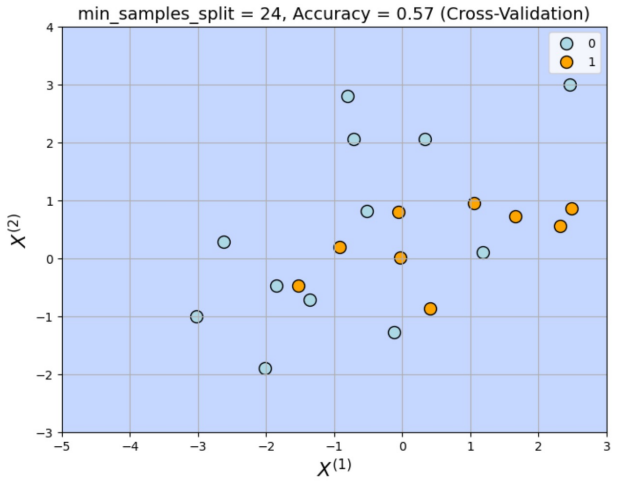



# Bias-Variance Trade-Offs



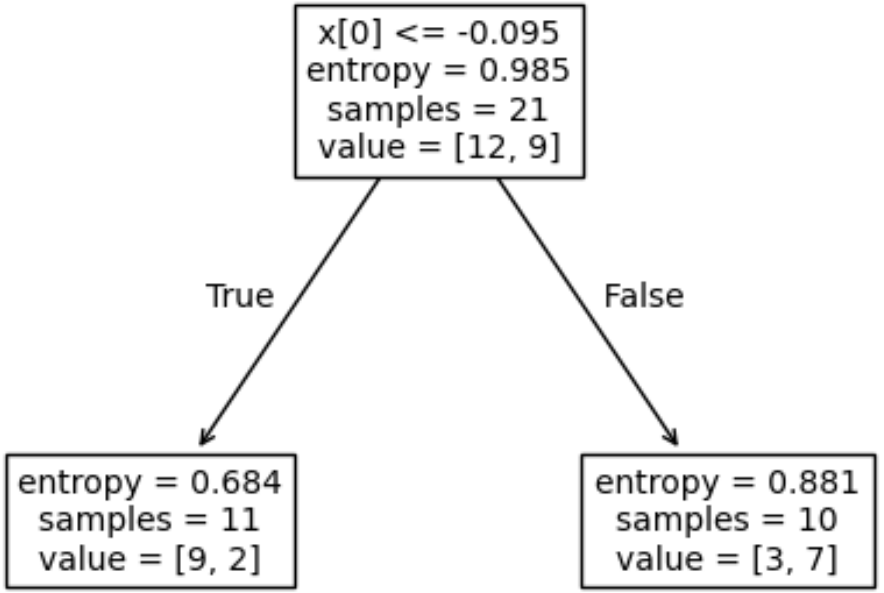

- High Variance, Low Bias
- Complex Decision Boundary
- High Train Accuracy
- Low Validation Accuracy

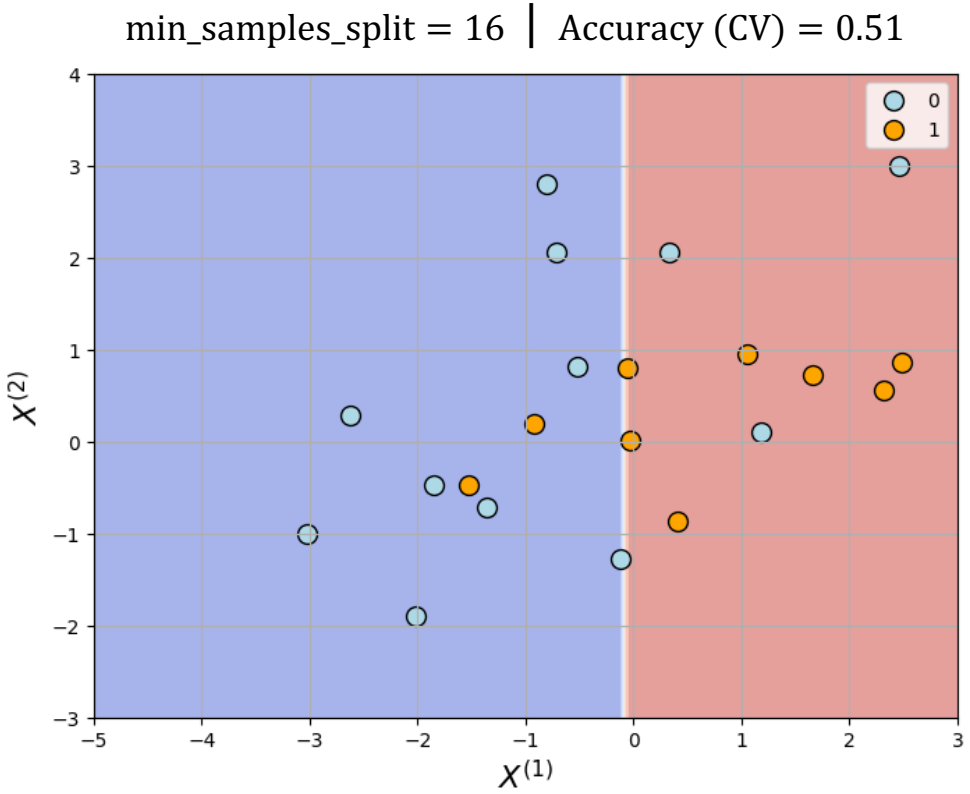


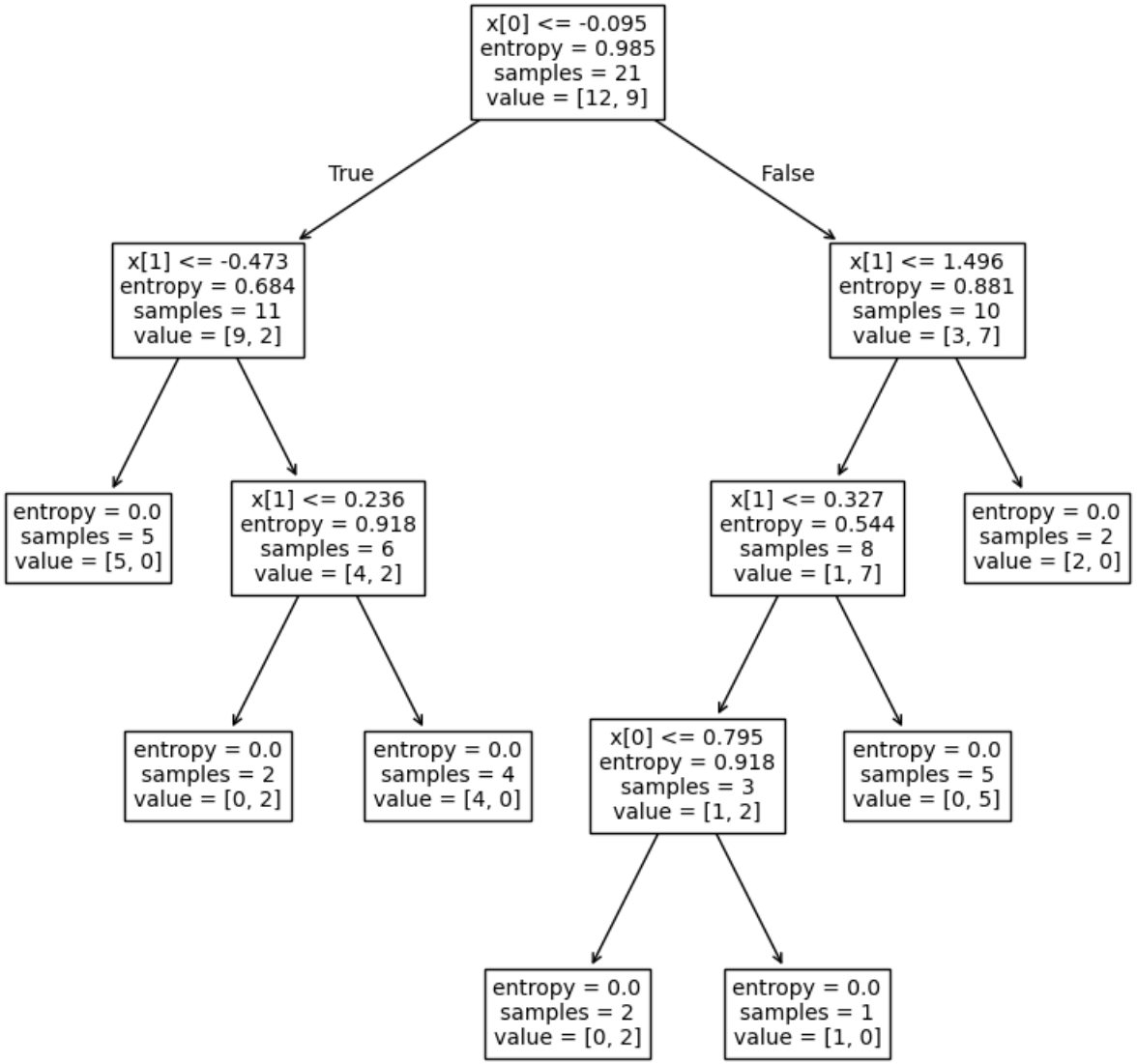

- High Bias, Low Variance
- Simple Decision Boundary
- Low Train Accuracy
- Low Validation Accuracy

# Decision Boundary (min_samples_split = 16)

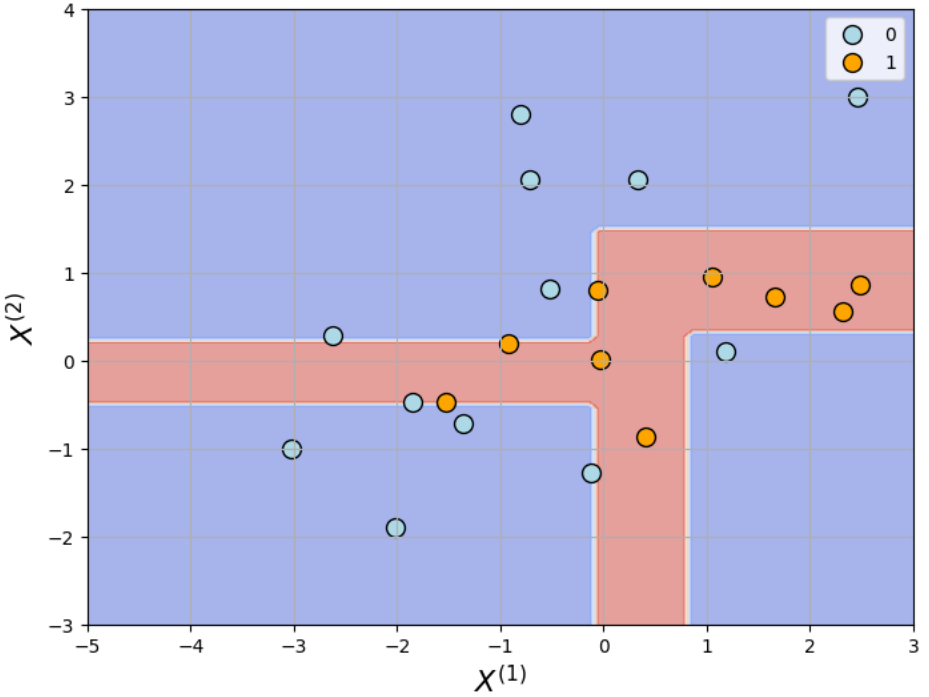


# Decision Boundary (min_samples_split = 2)

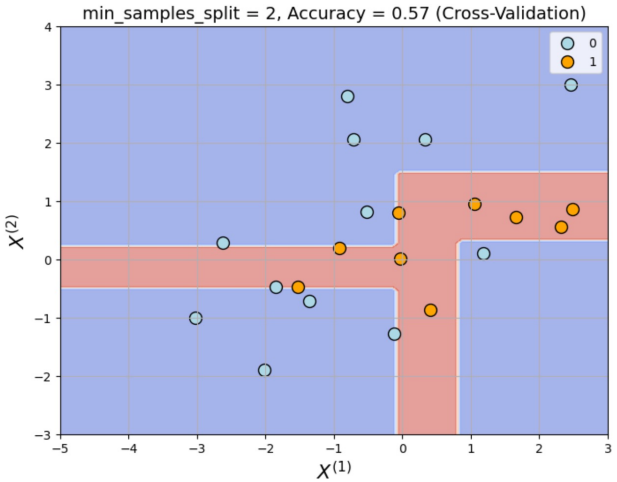



min_samples_split = 2 | Accuracy (CV) = 0.57

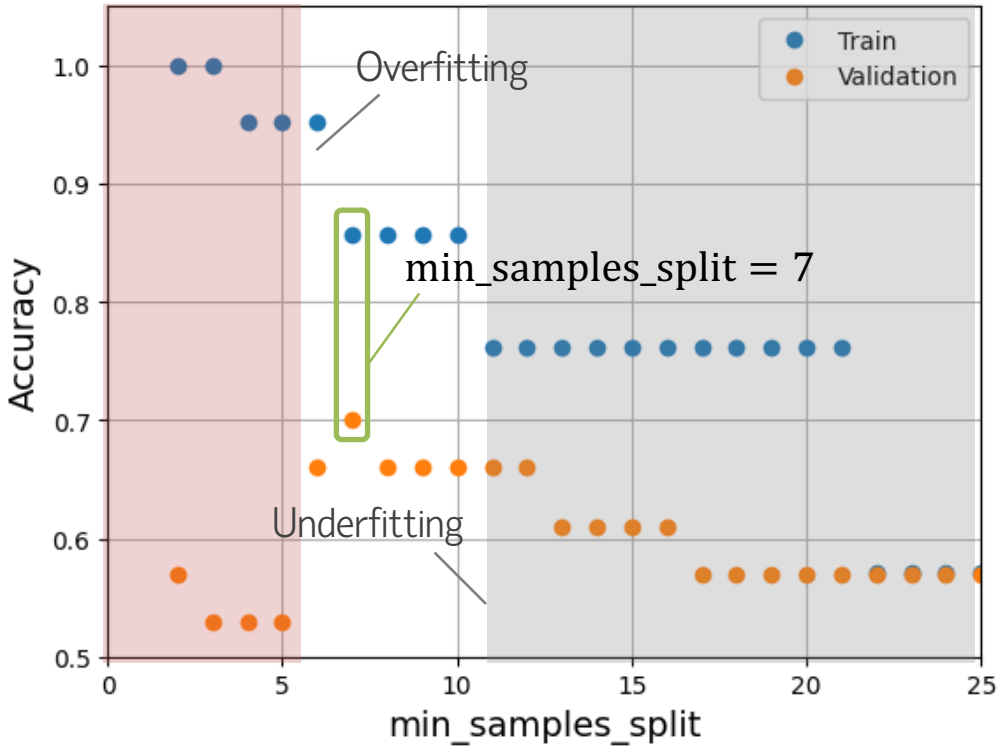


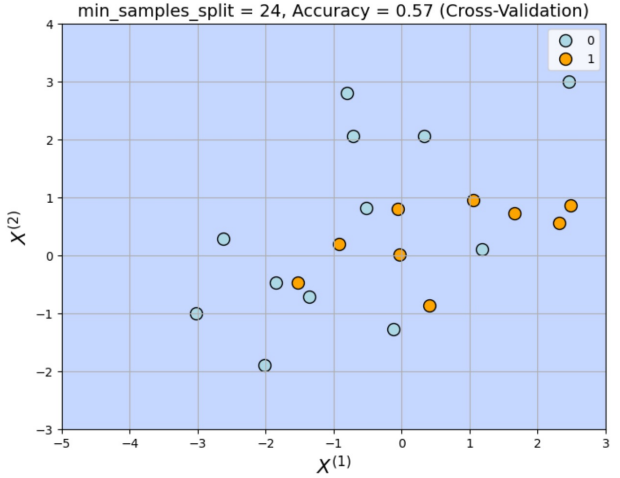
# Overfitting vs Underfitting

- Overfitting: If **min_samples_split** is too low, the tree continually splits on very few samples and ends up "memorizing" random noise in the training data. This drives up training accuracy but hurts validation accuracy due to poor generalization.
- Underfitting: If **min_samples_split** is too high, the tree is restricted from making potentially informative splits. It remains overly simplistic, capturing only broad patterns and missing important nuances—leading to low accuracy on both the training and validation sets.




- High Variance, Low Bias
- Complex Decision Boundary
- High Train Accuracy
- Low Validation Accuracy




- High Bias, Low Variance
- Simple Decision Boundary
- Low Train Accuracy
- Low Validation Accuracy

- Optimal: A balanced setting for **min_samples_split** allows the tree to make enough splits to capture meaningful structure, but not so many that it chases noise. Training and validation scores are closely aligned, indicating a more generalizable model.

# MyDecisionTreeClassifier (Revised)

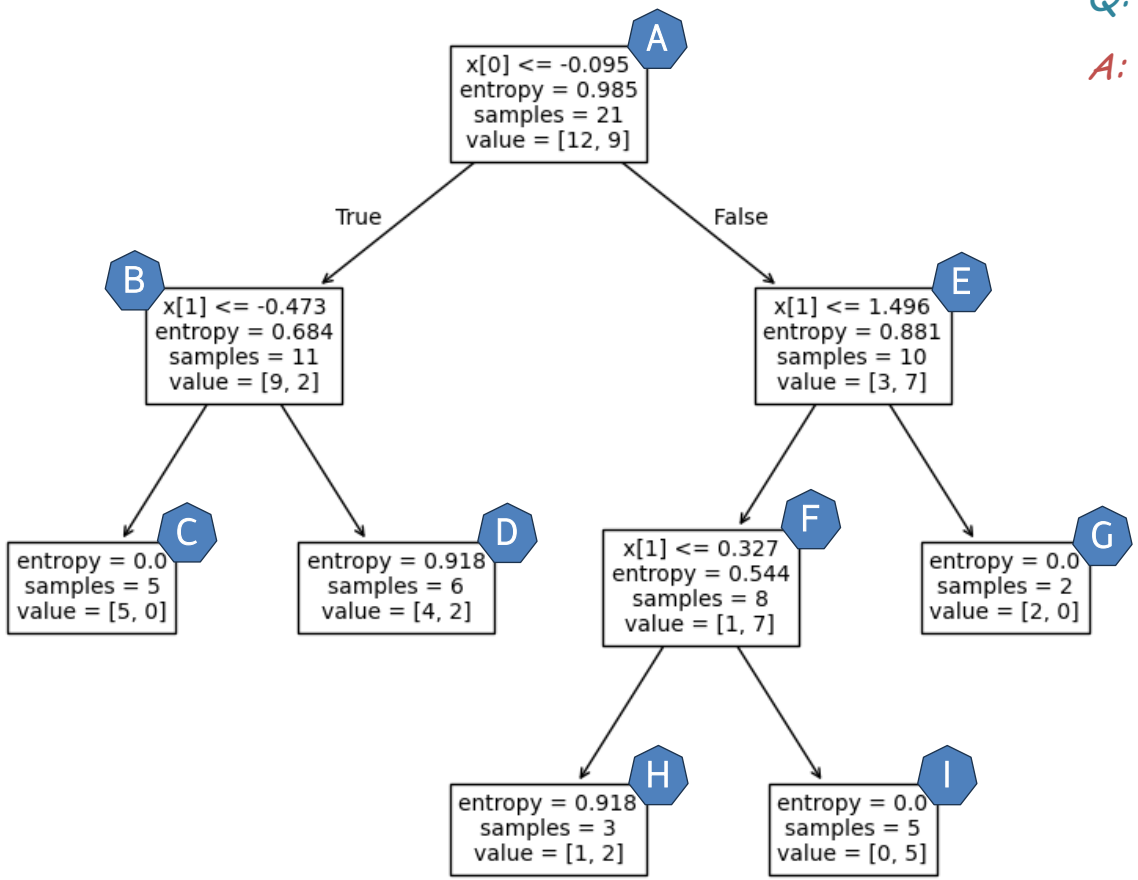
```
import numpy as np
from collections import Counter

def _entropy(y):
    counts = np.bincount(y)
    p = counts[counts > 0] / len(y)
    return -np.sum(p * np.log2(p))


def _gini(y):
    counts = np.bincount(y)
    p = counts[counts > 0] / len(y)
    return 1.0 - np.sum(p ** 2)


class MyDecisionTreeClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self,
                 criterion: str = "entropy",
                 max_depth: int | None = None,
                 min_samples_split: int = 2,
                 random_state: int | None = None):
        self.criterion = criterion
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.random_state = random_state

    def fit(self, X, y):
        X = np.asarray(X)
        y = np.asarray(y)

        # store meta-data required by sklearn
        self.n_features_in_ = X.shape[1]
        self.classes_, y_enc = np.unique(y, return_inverse=True)

        # choose impurity function
        self._impurity = _entropy if self.criterion == "entropy" else _gini
        rng = np.random.RandomState(self.random_state)

        # build the tree
        self.tree_ = self._grow_tree(X, y_enc, depth=0, rng=rng)
        return self
```

```
...cont...

    def predict(self, X):
        X = np.asarray(X)
        preds = np.array([self._predict_one(row, self.tree_) for row in X])
        return self.classes_[preds]

    def _best_split(self, X, y):
        best_gain = -np.inf
        best_feat = None
        best_thresh = None
        parent_imp = self._impurity(y)
        n_samples, n_features = X.shape

        for j in range(n_features):
            # unique sorted feature values
            vals = np.unique(X[:, j])
            if vals.size == 1:
                continue
            # mid-points between consecutive values
            thresholds = (vals[:-1] + vals[1:]) / 2.0
            for t in thresholds:
                left_mask = X[:, j] <= t
                right_mask = ~left_mask
                if not left_mask.any() or not right_mask.any():
                    continue
                y_left, y_right = y[left_mask], y[right_mask]
                w_left = len(y_left) / n_samples
                w_right = 1 - w_left
                gain = (parent_imp
                        - w_left * self._impurity(y_left)
                        - w_right * self._impurity(y_right))
                if gain > best_gain:
                    best_gain, best_feat, best_thresh = gain, j, t
        return best_feat, best_thresh, best_gain
```

# MyDecisionTreeClassifier (cont.)

```
...cont...


    def _grow_tree(self, X, y, depth, rng):
        """Recursively build the tree; returns a nested-dict node."""
        node = {}
        # Step-1 stopping criteria
        if (len(np.unique(y)) == 1
                or (self.max_depth is not None and depth >= self.max_depth)
                or len(y) < self.min_samples_split):
            node["type"] = "leaf"
            node["class"] = Counter(y).most_common(1)[0][0]
            return node

        # Step-3: best split search
        feat, thresh, gain = self._best_split(X, y)
        if feat is None or gain <= 0:
            node["type"] = "leaf"
            node["class"] = Counter(y).most_common(1)[0][0]
            return node

        node.update(type="split", feature=feat, threshold=thresh)

        left_idx = X[:, feat] <= thresh
        right_idx = ~left_idx
        node["left"] = self._grow_tree(X[left_idx], y[left_idx], depth + 1, rng)
        node["right"] = self._grow_tree(X[right_idx], y[right_idx], depth + 1, rng)
        return node

    def _predict_one(self, x_row, node):
        """Traverse tree until reaching a leaf."""
        while node["type"] == "split":
            node = node["left"] if x_row[node["feature"]] <= node["threshold"] else node["right"]
        return node["class"]
```

# Feature Importances




*Q: How do we know which features are importance and which ones are not?*

*A: We find how much information this feature gained across the whole tree.*

Information Gain at every Split

| Node | Feature | Information Gain $IG_t$ | Node Weight $w_t = \frac{N_t}{N_{\text{Root}}}$ | Contribution $w_t IG_t$ |
|---|---|---|---|---|
| A | $x[0]$ | $0.985 - \left(\frac{11}{21}0.684 + \frac{10}{21}0.881\right) = 0.2078$ | $21/21 = 1$ | 0.2078 |
| B | $x[1]$ | $0.684 - \left(\frac{5}{11}0 + \frac{6}{11}0.918\right) = 0.1830$ | $11/21 = 0.5238$ | 0.0954 |
| E | $x[1]$ | $0.881 - \left(\frac{8}{10}0.544 + \frac{2}{10}0\right) = 0.4458$ | $10/21 = 0.4762$ | 0.2123 |
| F | $x[1]$ | $0.544 - \left(\frac{3}{8}0.918 + \frac{5}{8}0\right) = 0.1998$ | $8/21 = 0.3810$ | 0.0761 |

Contribution by Feature

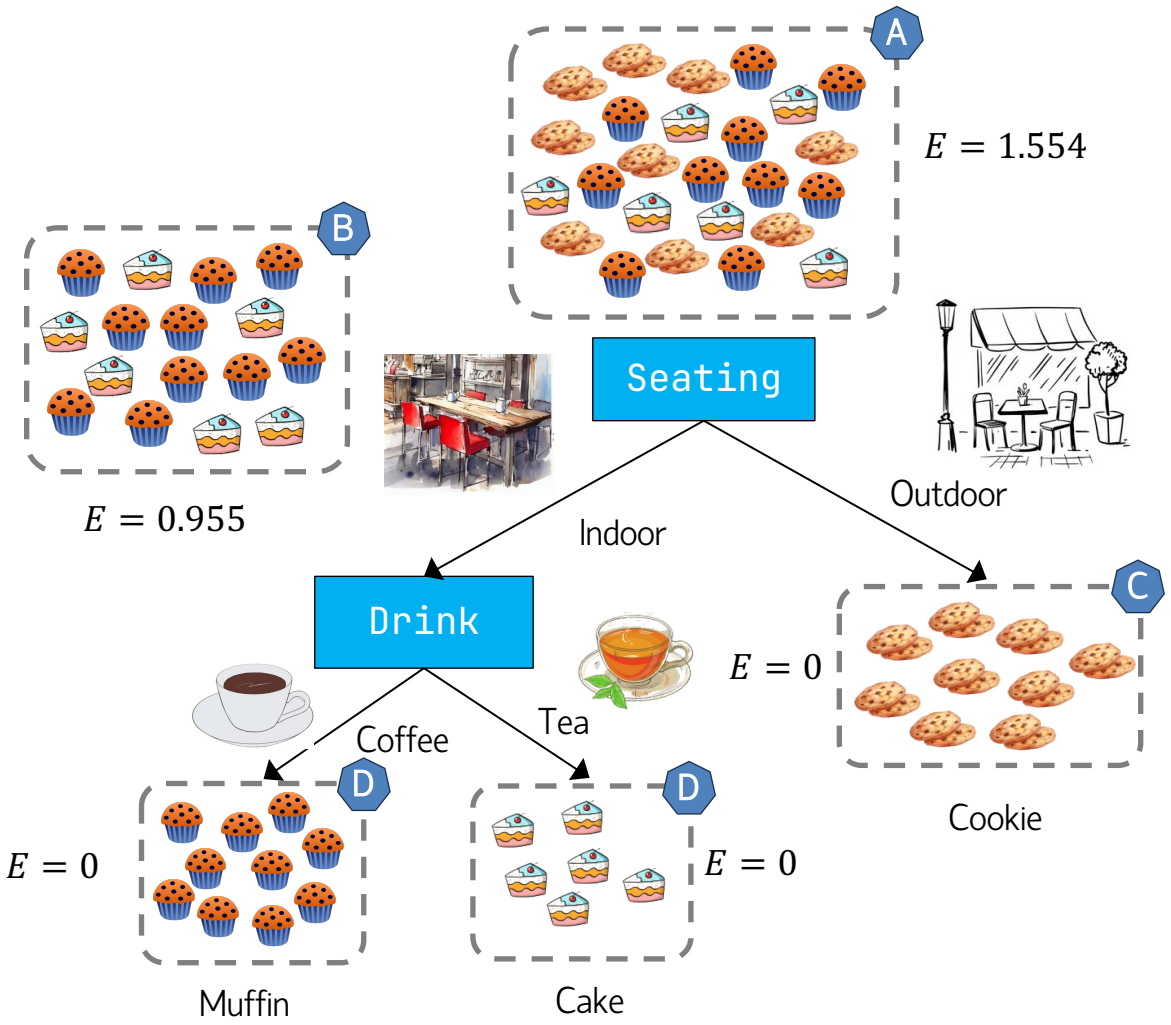
| Feature | Sum of Contribution $\sum_t w_t IG_t$ | Normalised Importance |
|---|---|---|
| $x[0]$ | 0.2078 | $0.2078/0.5916 \approx 0.352$ |
| $x[1]$ | $0.0954 + 0.2123 + 0.0761 = 0.3838$ | $0.3838/0.5916 \approx 0.648$ |

Total Information Gain across the Tree $= 0.2078 + 0.3838 = 0.5916.$

Mean Information Increment:

- $x[1]$ accounts for ~65% of the tree's overall information gain.
- $x[0]$ contributes the remaining ~35%.

# Try-It-Yourself: Feature Importances




Information Gain at every Split

| Node | Feature | Information Gain $IG_t$ | Node Weight $w_t = \frac{N_t}{N_{\text{Root}}}$ | Contribution $w_t IG_t$ |
|---|---|---|---|---|
| A | Seating | | | |
| B | Drink | | | |

Contribution by Feature

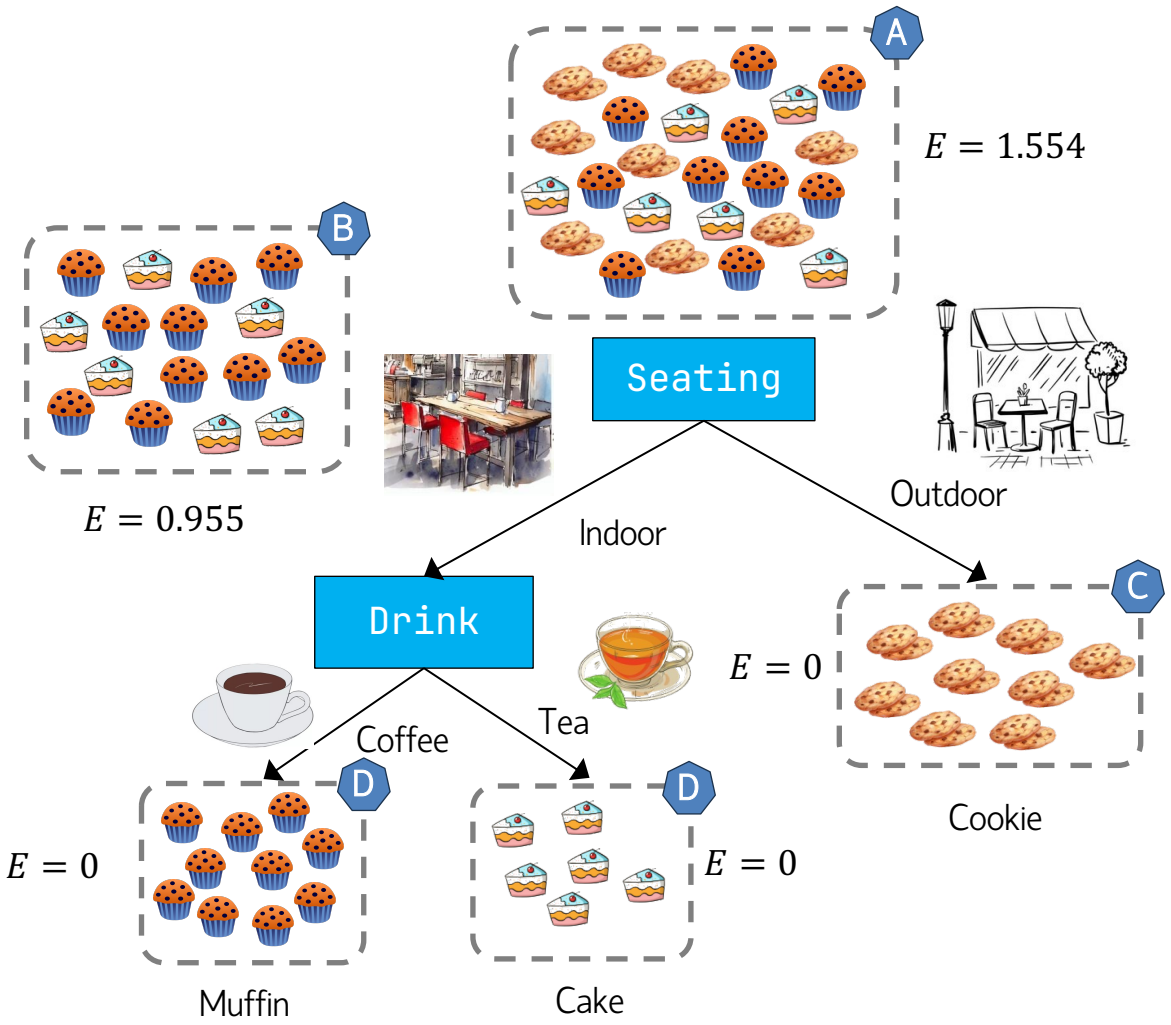
| Feature | Sum of Contribution $\sum_t w_t IG_t$ | Normalised Importance |
|---|---|---|
| Seating | | |
| Drink | | |

Total Information Gain across the Tree = __ + __ = __.

Mean Information Increment:

- Seating accounts for __% of the tree's overall information gain.
- Drink contributes the remaining __%.

# Solution: Feature Importances




Information Gain at every Split

| Node | Feature | Information Gain $IG_t$ | Node Weight $w_t = \frac{N_t}{N_{\text{Root}}}$ | Contribution $w_t IG_t$ |
|---|---|---|---|---|
| A | Seating | $1.554 - \left(\frac{16}{25}0.955 + \frac{9}{25}0\right) = 0.9428$ | $25/25 = 1$ | 0.9428 |
| B | Drink | $0.955 - \left(\frac{10}{16}0 + \frac{6}{16}0\right) = 0.955$ | $16/25 = 0.64$ | 0.6112 |

Contribution by Feature

| Feature | Sum of Contribution $\sum_t w_t IG_t$ | Normalised Importance |
|---|---|---|
| Seating | 0.9428 | $0.9428/1.554 \approx 0.607$ |
| Drink | 0.6112 | $0.6112/1.554 \approx 0.393$ |

Total Information Gain across the Tree $= 0.9428 + 0.6112 = 1.554.$

Mean Information Increment:

- Seating accounts for ~61% of the tree's overall information gain.
- Drink contributes the remaining ~39%.

# Summary

- A decision tree learns a series of if/else questions that maximise information gain at every split, steadily reducing the entropy of the data until each leaf is (almost) pure.
- Controlling complexity with hyper-parameters such as max_depth, min_samples_split, and min_samples_leaf is essential: tiny values give a deep, jagged boundary (high variance), while large values give an overly simple tree (high bias).
- Use k-fold cross-validation on the training split to pick those hyper-parameters; the test set remains untouched until the very end, providing an unbiased snapshot of real-world performance. A well-tuned tree achieves the sweet spot of high accuracy on both train and validation data, avoiding over-fitting while still capturing the key patterns in the dataset.
- Decision trees work natively with both numeric and (label-encoded) categorical inputs, selecting optimal thresholds for numbers and optimal groupings for categories—so feature scaling or normalisation is not required.
- Each feature's variable importance is transparent: sum the weighted impurity drops it provides across the tree. This "mean decrease in impurity" highlights which predictors truly drive the model.