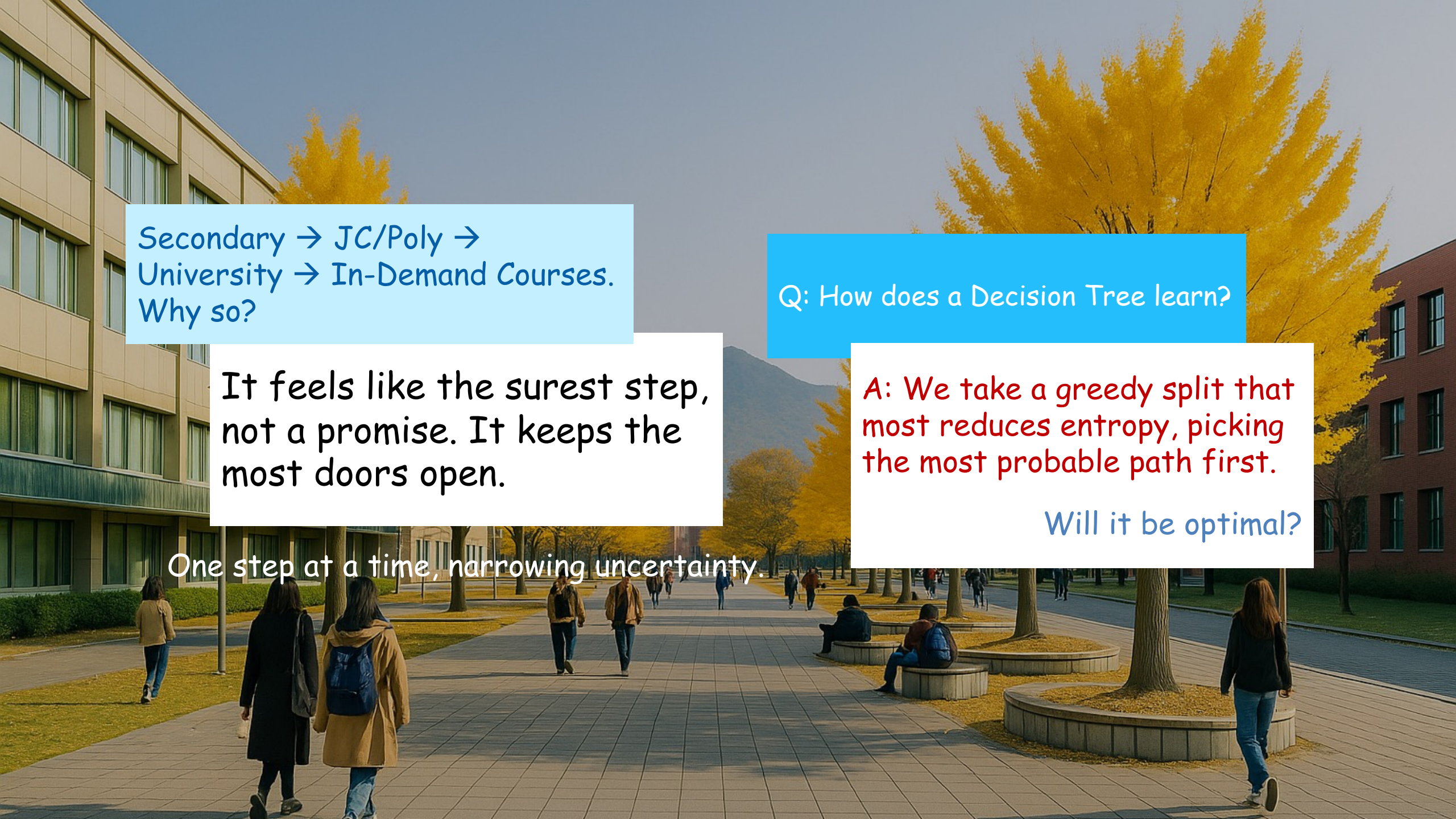
Machine Learning

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?

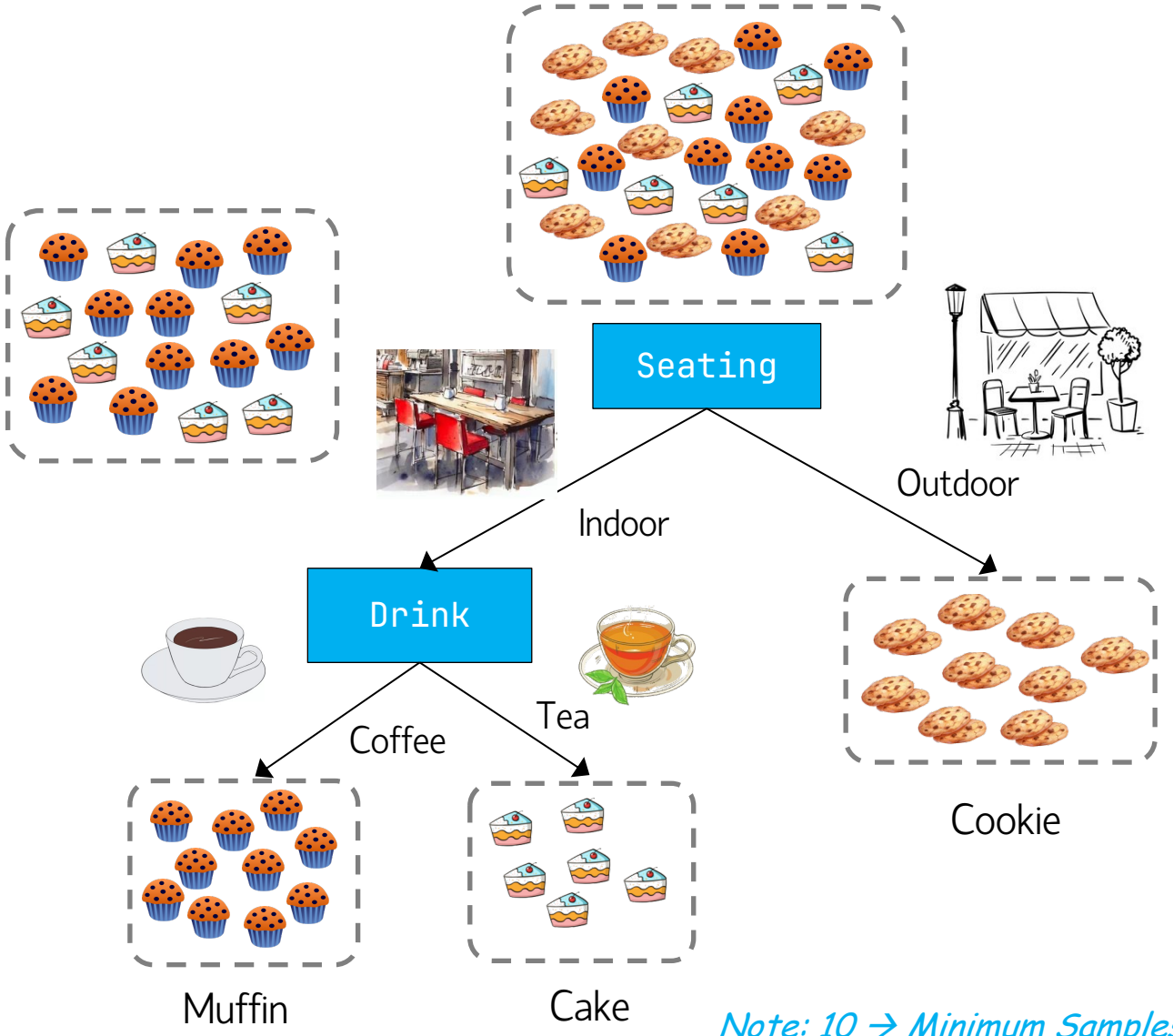
Café Example Dataset

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



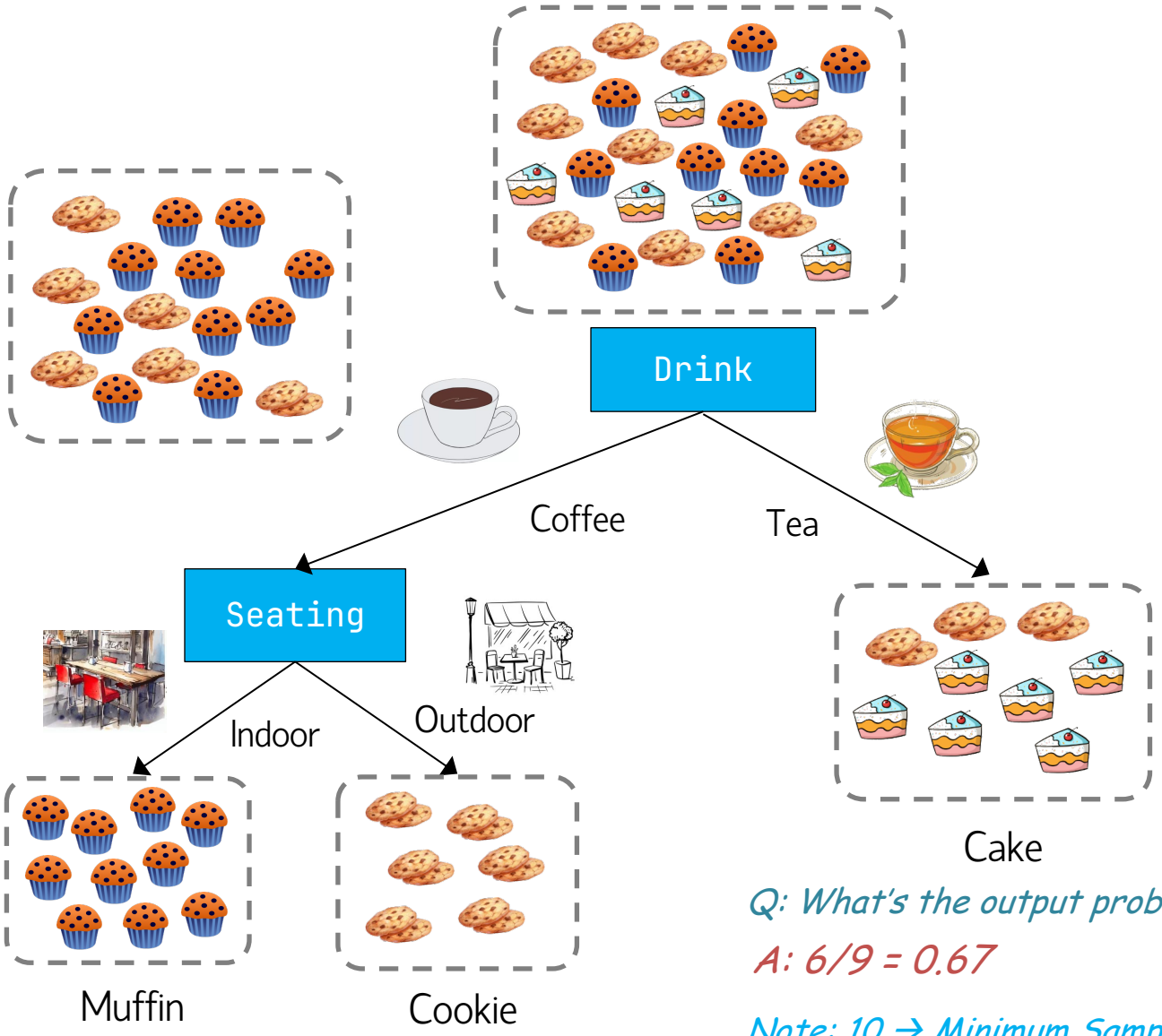
Seating	Drink	Dessert
Outdoor	Tea	Cookie
Outdoor	Tea	Cookie
Outdoor	Tea	Cookie
Outdoor	Tea	Cookie
Outdoor	Coffee	Cookie
Indoor	Coffee	Muffin
Indoor	Coffee	Muffin
Indoor	Tea	Cake
Indoor	Tea	Cake
Indoor	Tea	Cake

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?

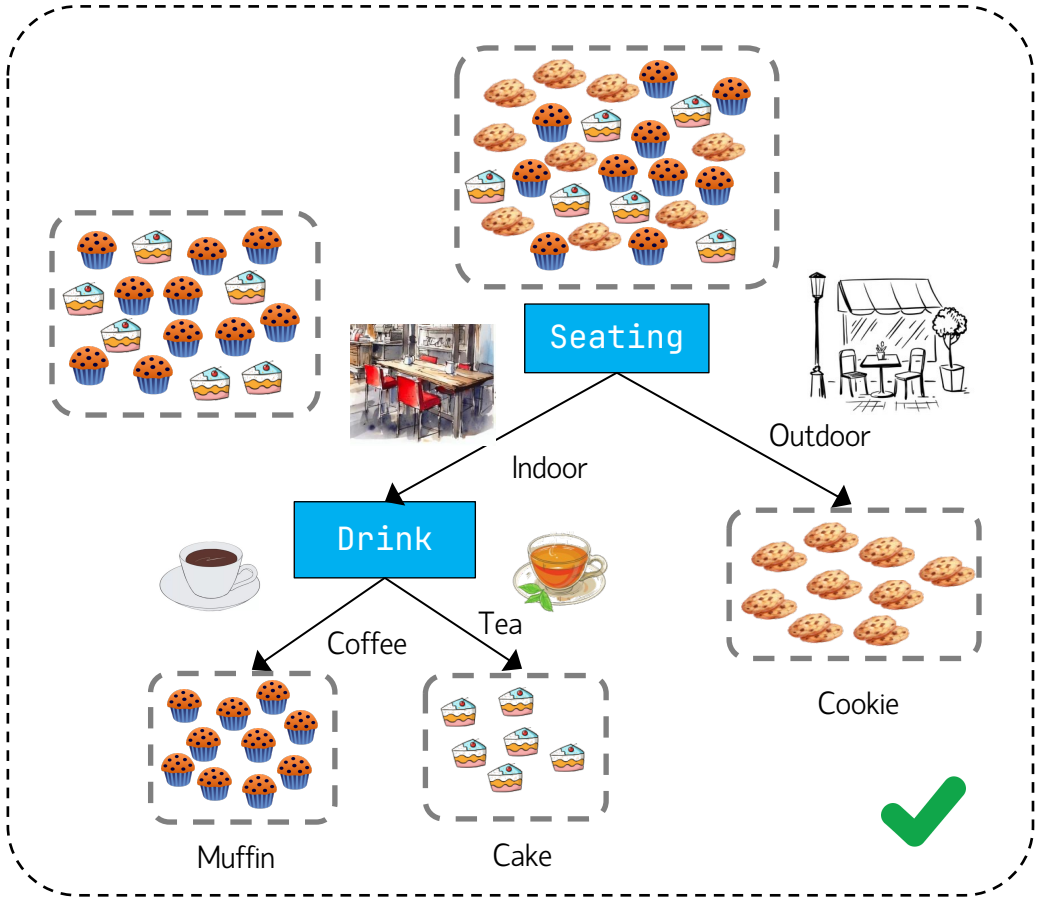
A: 6/9 = 0.67

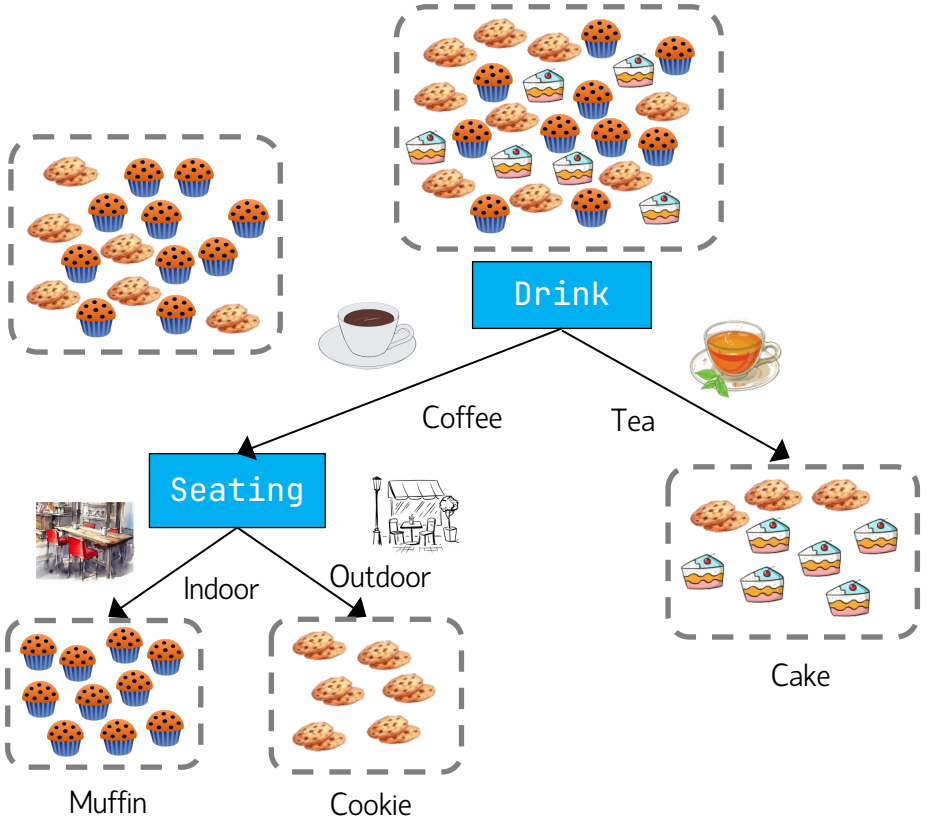
Note: 10 → Minimum Samples Split

Q: How many did we classify it correctly?

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

Which Tree Is Better?





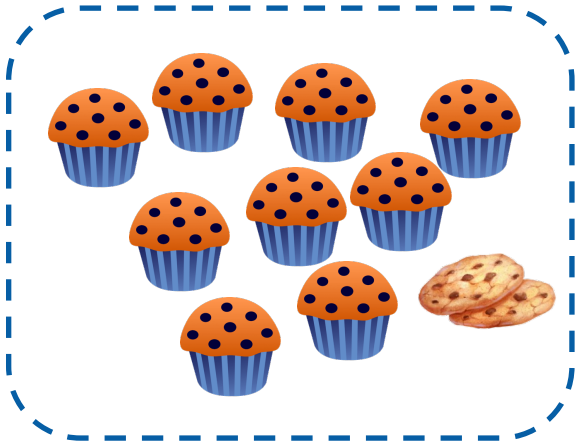
Note: 10 → Minimum Samples Split

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

A: Purer Leaves.

Classification Errors

Low Entropy

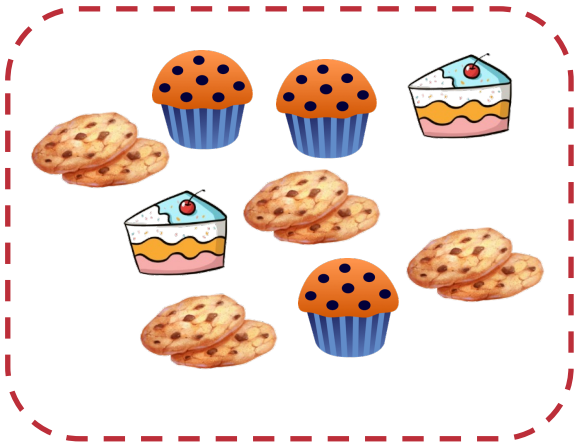


10 customers who ordered Latte

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

Muffins. 9 out of 10 customers did that.

High Entropy



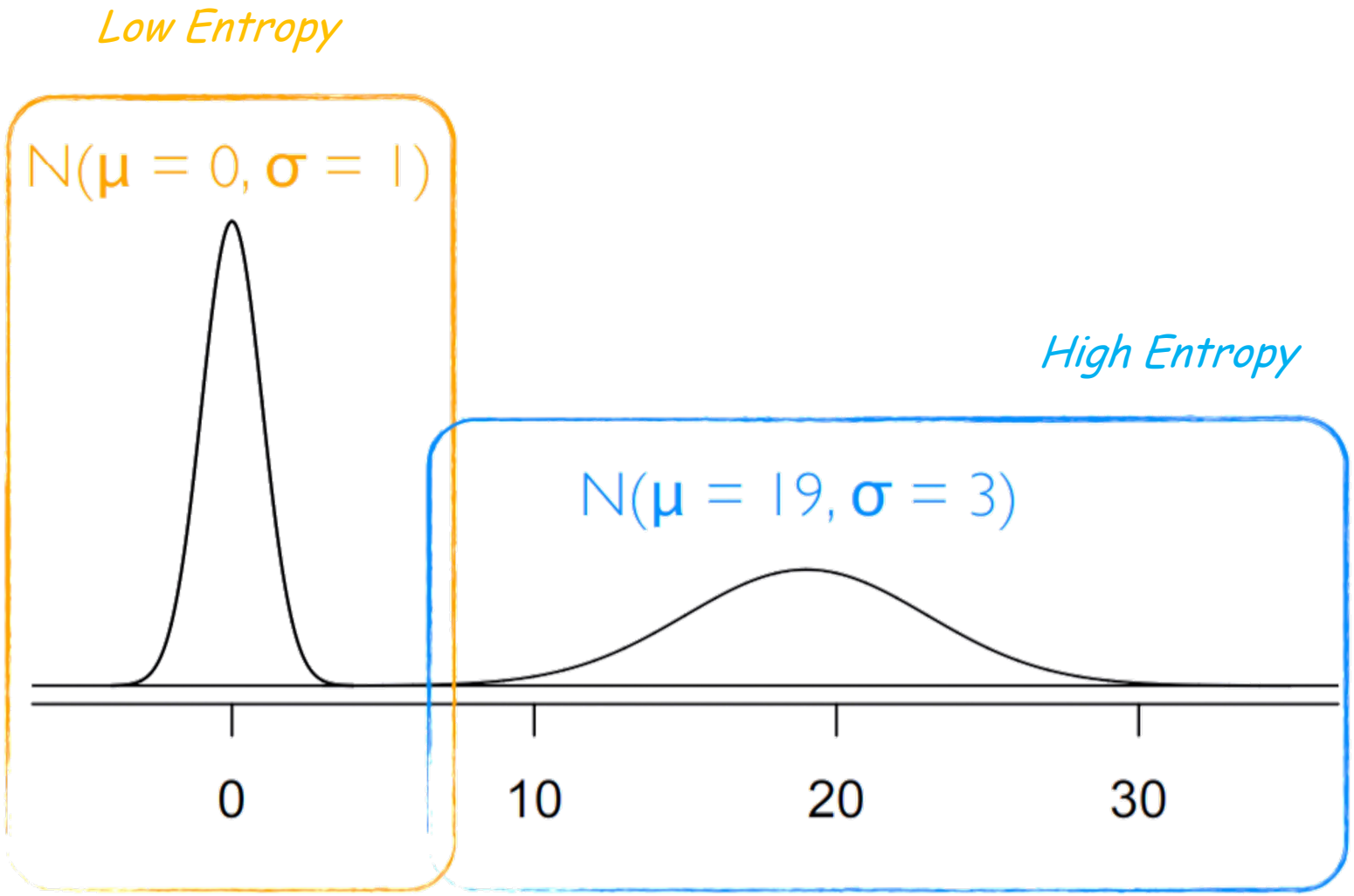
9 customers who ordered Mocha

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

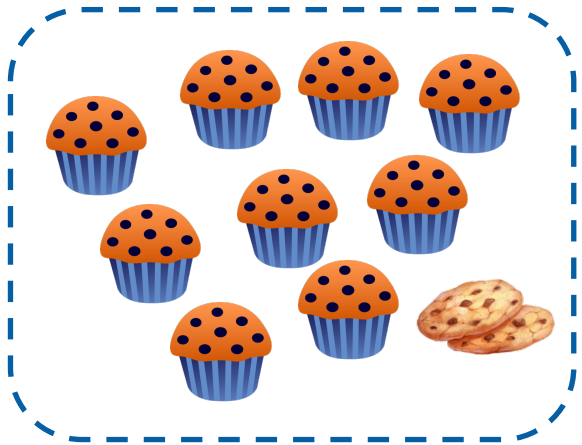


Entropy

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

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

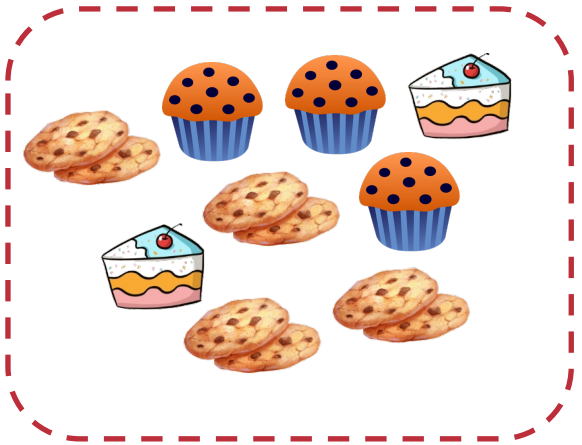
Low Impurity → Low Entropy



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

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

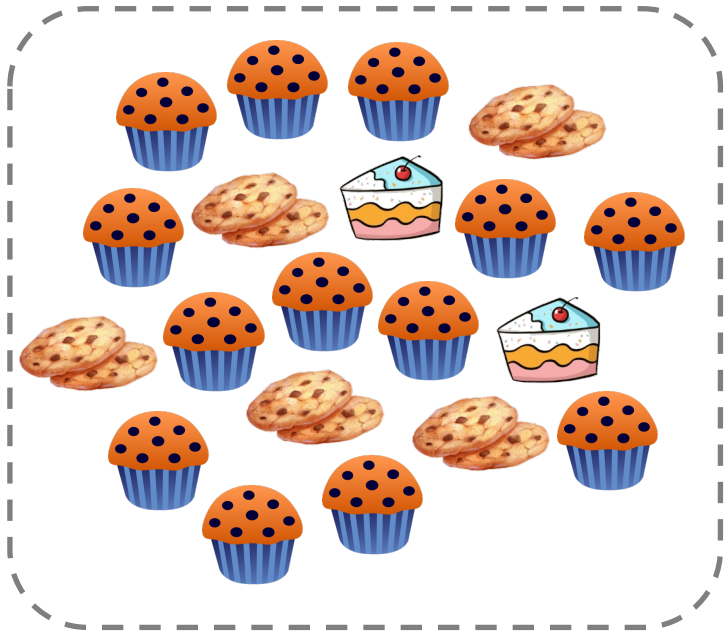
High Impurity → High Entropy



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

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

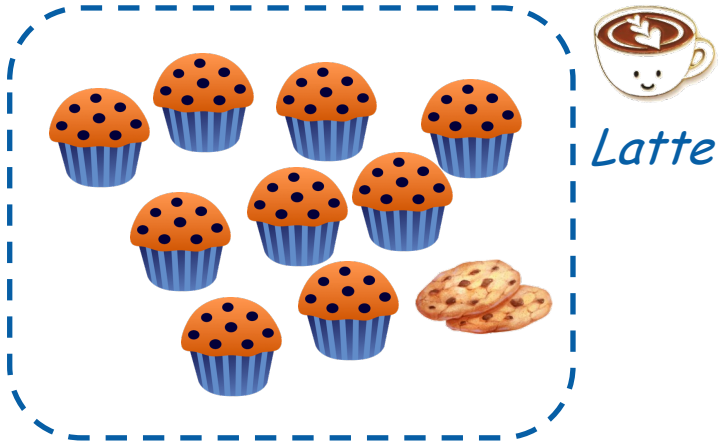
Information Gain



$E = 1.23$

0.23

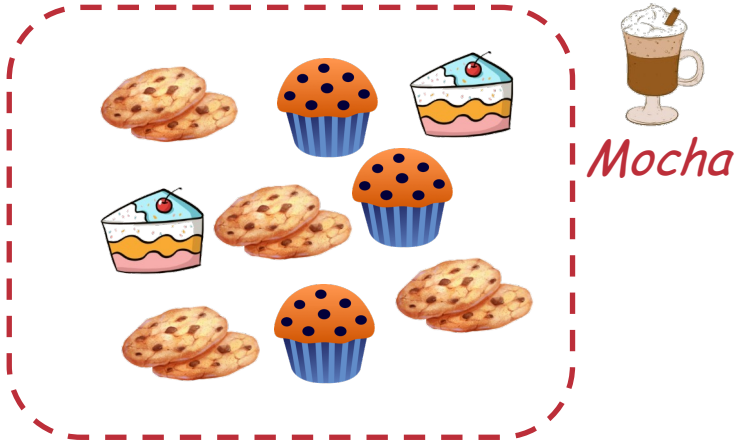
$E = 0.47$



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

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

$E = 1.52$



Information Gain $IG = 1.23 - 1.00$

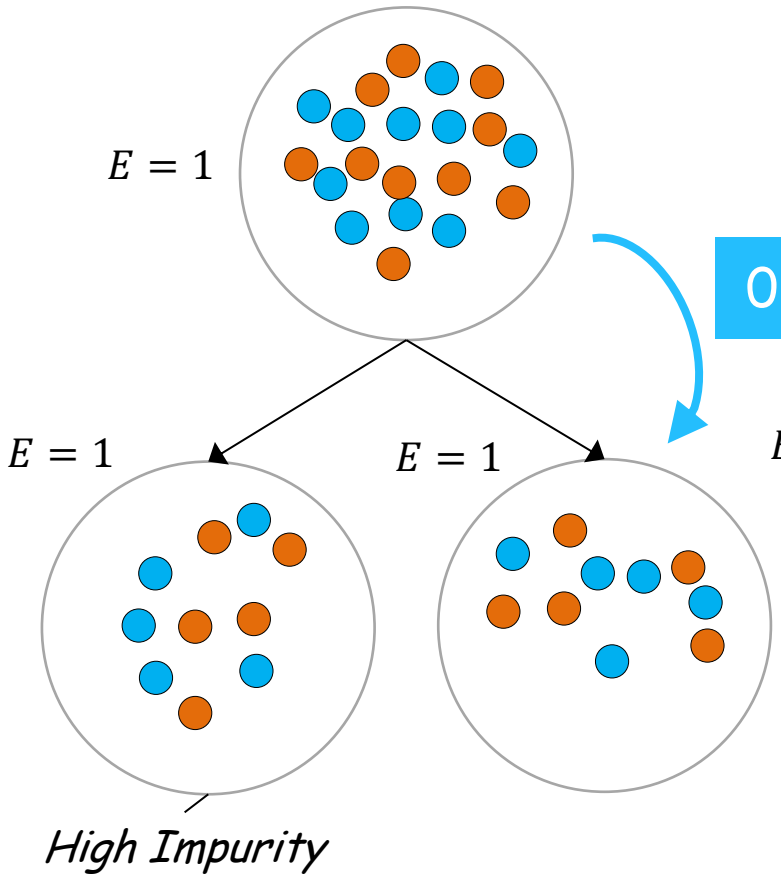
$= 0.23$

$$E = \frac{10}{19}0.47 + \frac{9}{19}1.52$$ (Weighted Sum)

$$= 0.24 + 0.76 = 1.00$$

Information Gain (cont.)

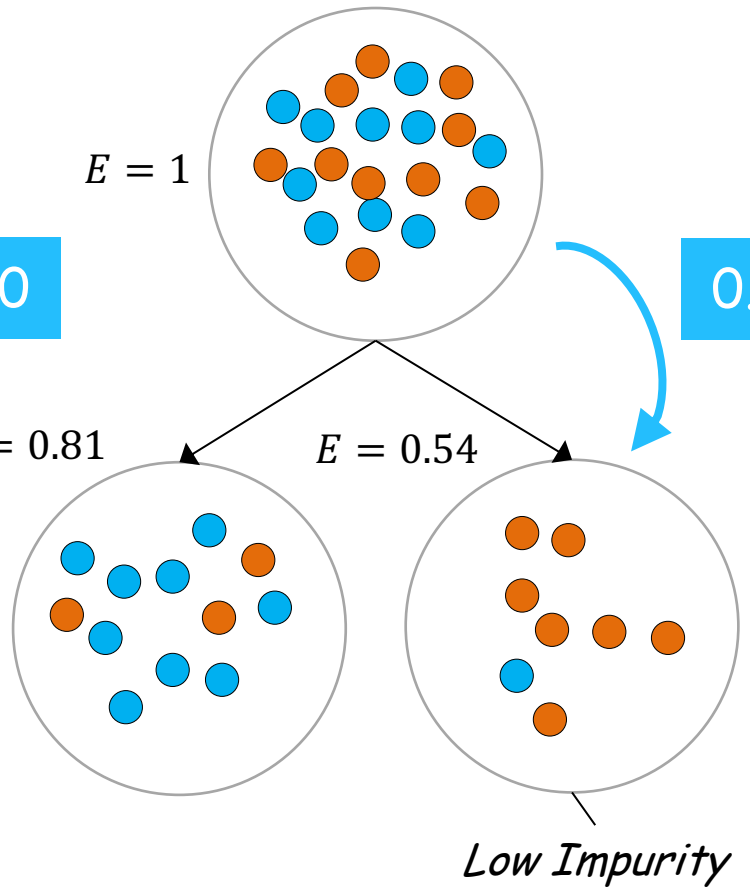
Entropy: $E = -\sum_{i=1}^{N} p_i \log_2(p_i)$



0.00

High Impurity

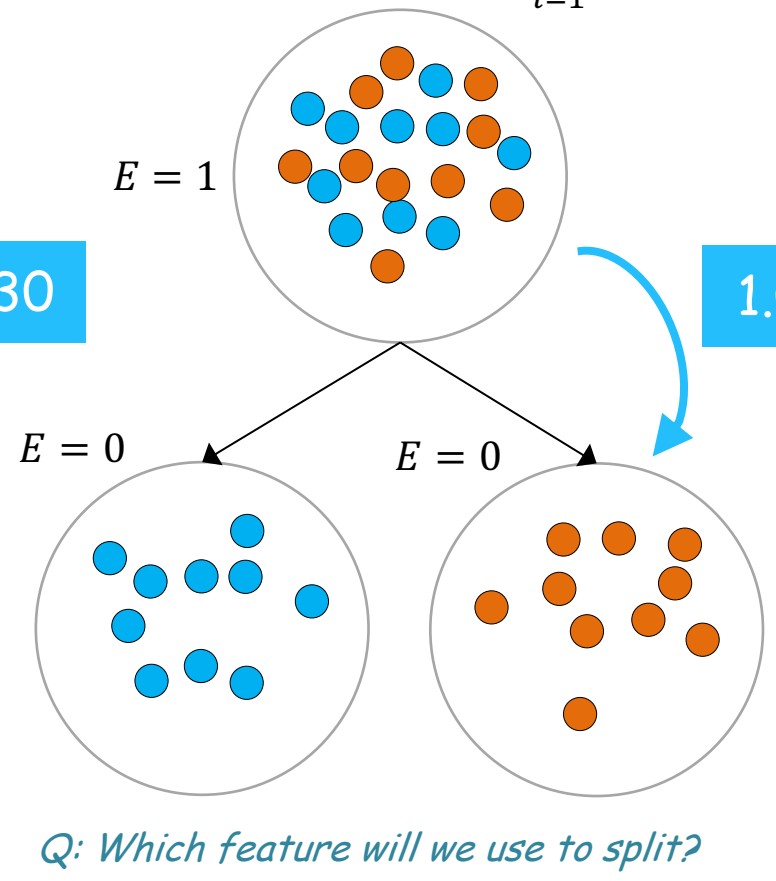
$$E = \frac{10}{20}1.00 + \frac{10}{20}1.00 = 1.00$$



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.

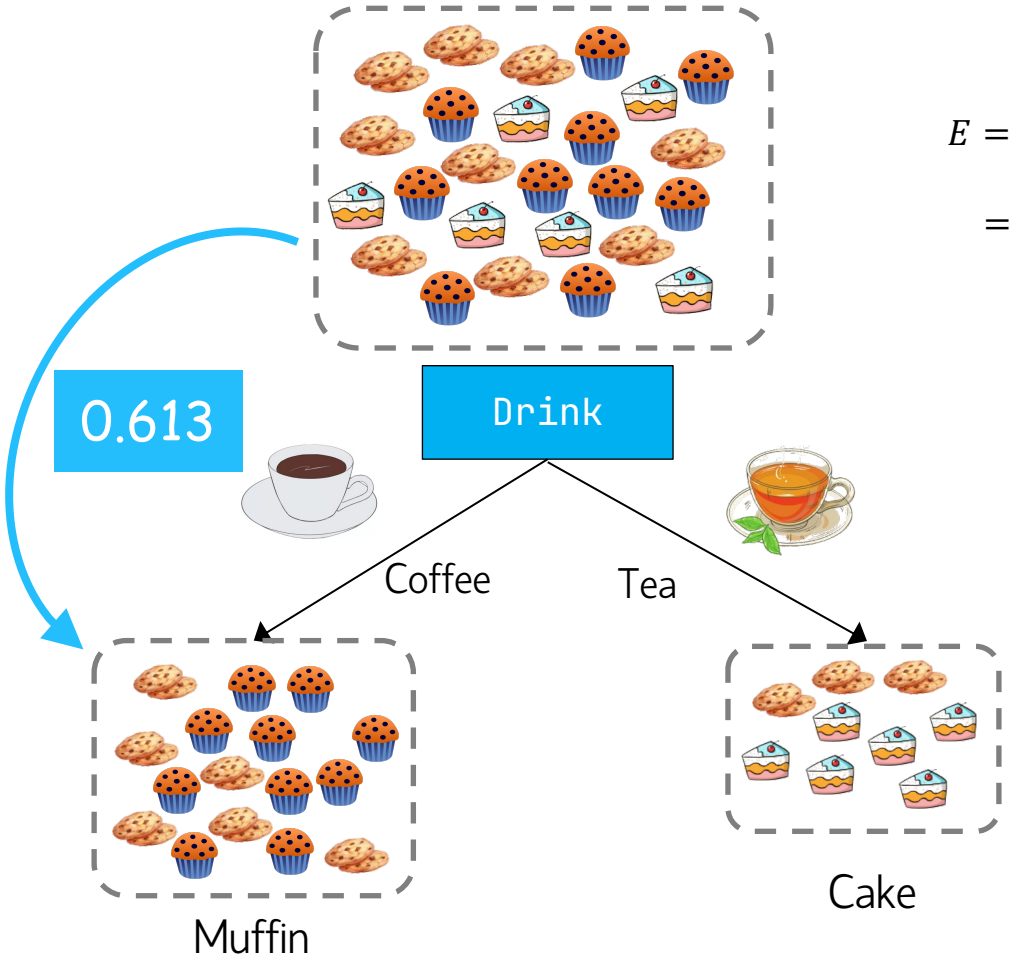
Café Example Dataset

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)



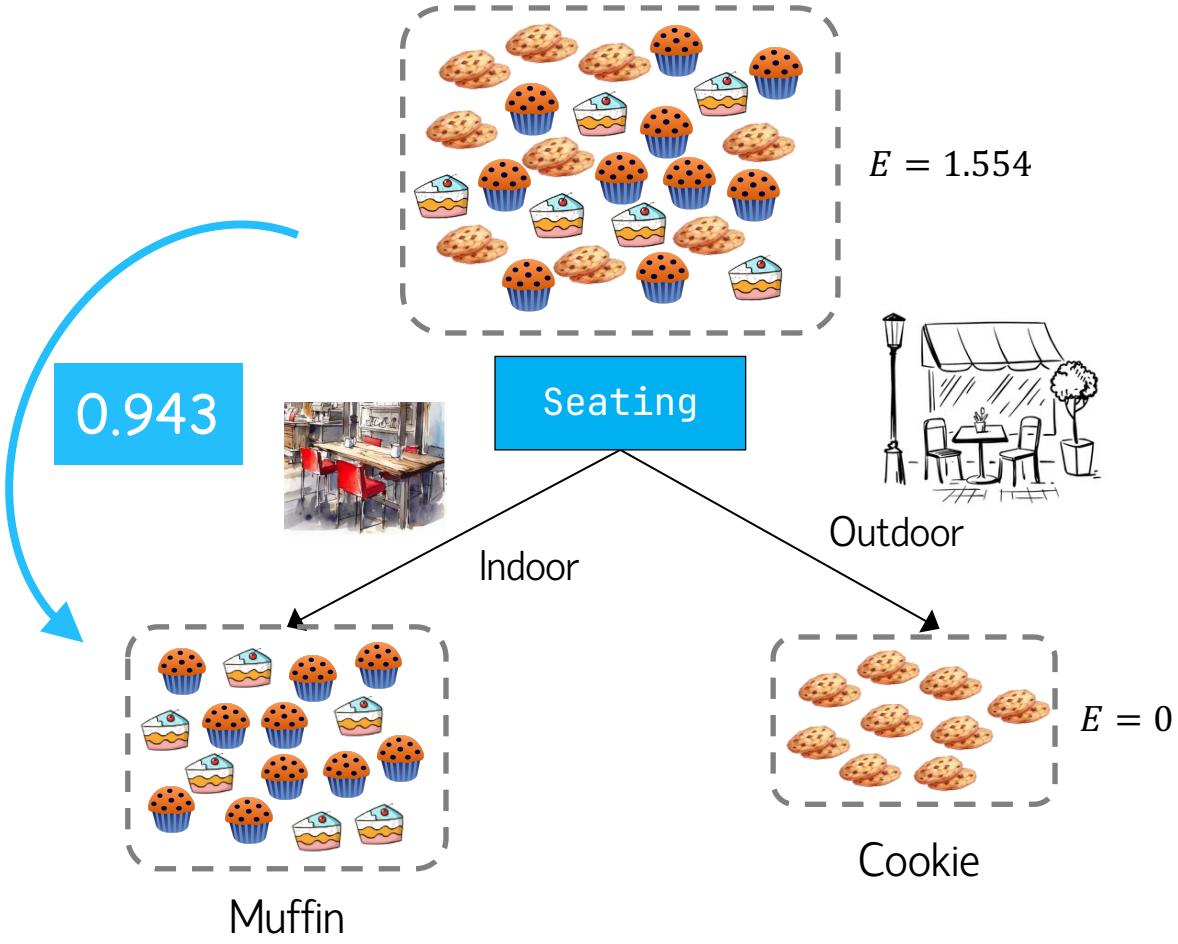
$$E = -\left(\frac{10}{25}\log_2\left(\frac{10}{25}\right) + \frac{9}{25}\log_2\left(\frac{9}{25}\right) + \frac{6}{25}\log_2\left(\frac{6}{25}\right)\right)$$
$$= -(-0.529 - 0.531 - 0.494) = 1.554$$

$$E = -\left(\frac{6}{9}\log_2\left(\frac{6}{9}\right) + \frac{3}{9}\log_2\left(\frac{3}{9}\right)\right)$$
$$= -(-0.390 - 0.528) = 0.918$$

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

$$IG = 1.554 - \left(\frac{16}{25}0.955 + \frac{9}{25}0.918\right)$$
$$= 1.554 - (0.611 + 0.330) = 0.613$$

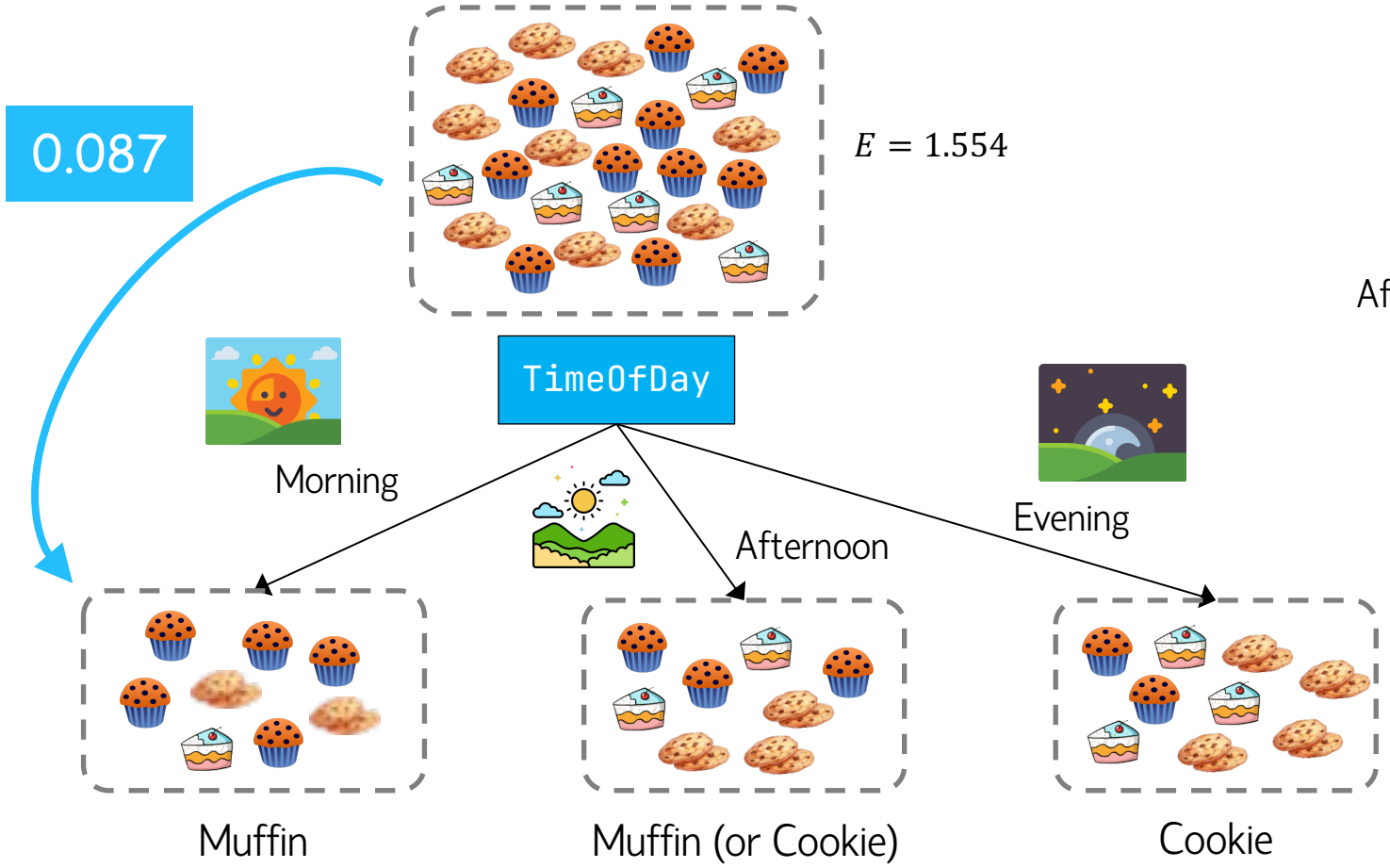
Decision Tree Learning (2 of 8)



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

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

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:

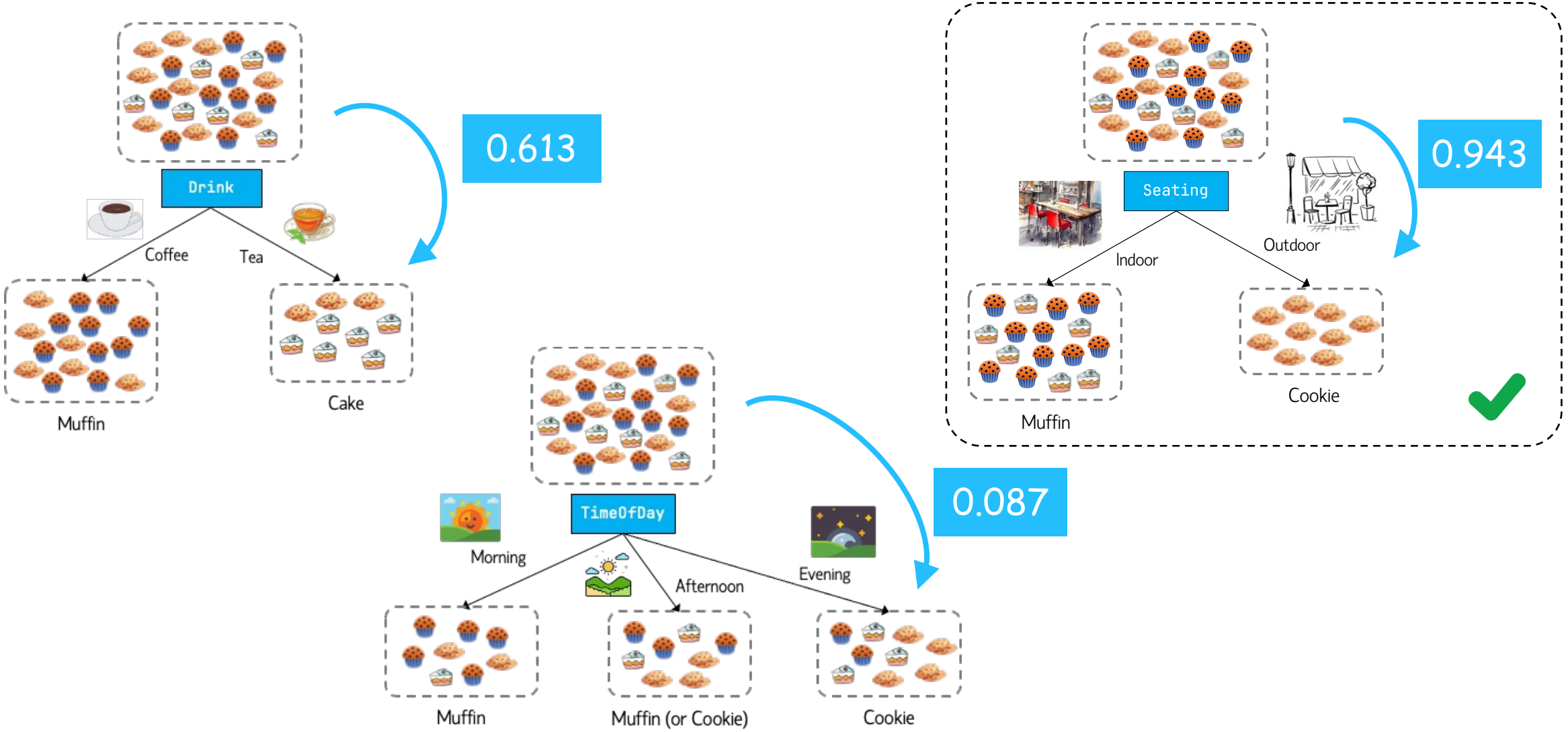
$$E = -\left(\frac{2}{9}\log_2\left(\frac{2}{9}\right) + \frac{4}{9}\log_2\left(\frac{4}{9}\right) + \frac{3}{9}\log_2\left(\frac{3}{9}\right)\right)$$

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

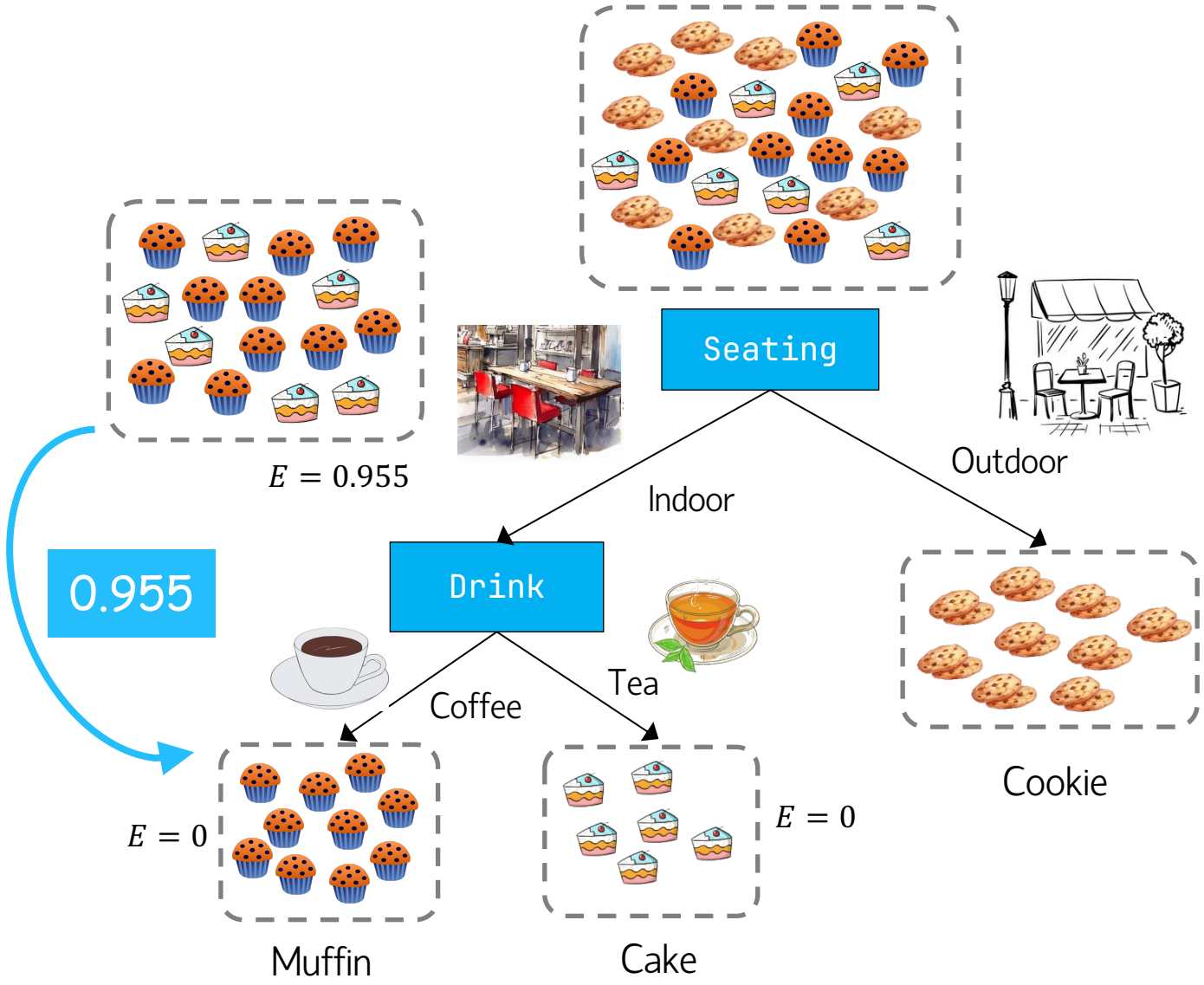
$$IG = 1.554 - \left(\frac{8}{25}1.299 + \frac{8}{25}1.562 + \frac{9}{25}1.530\right)$$

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

Decision Tree Learning (4 of 8)



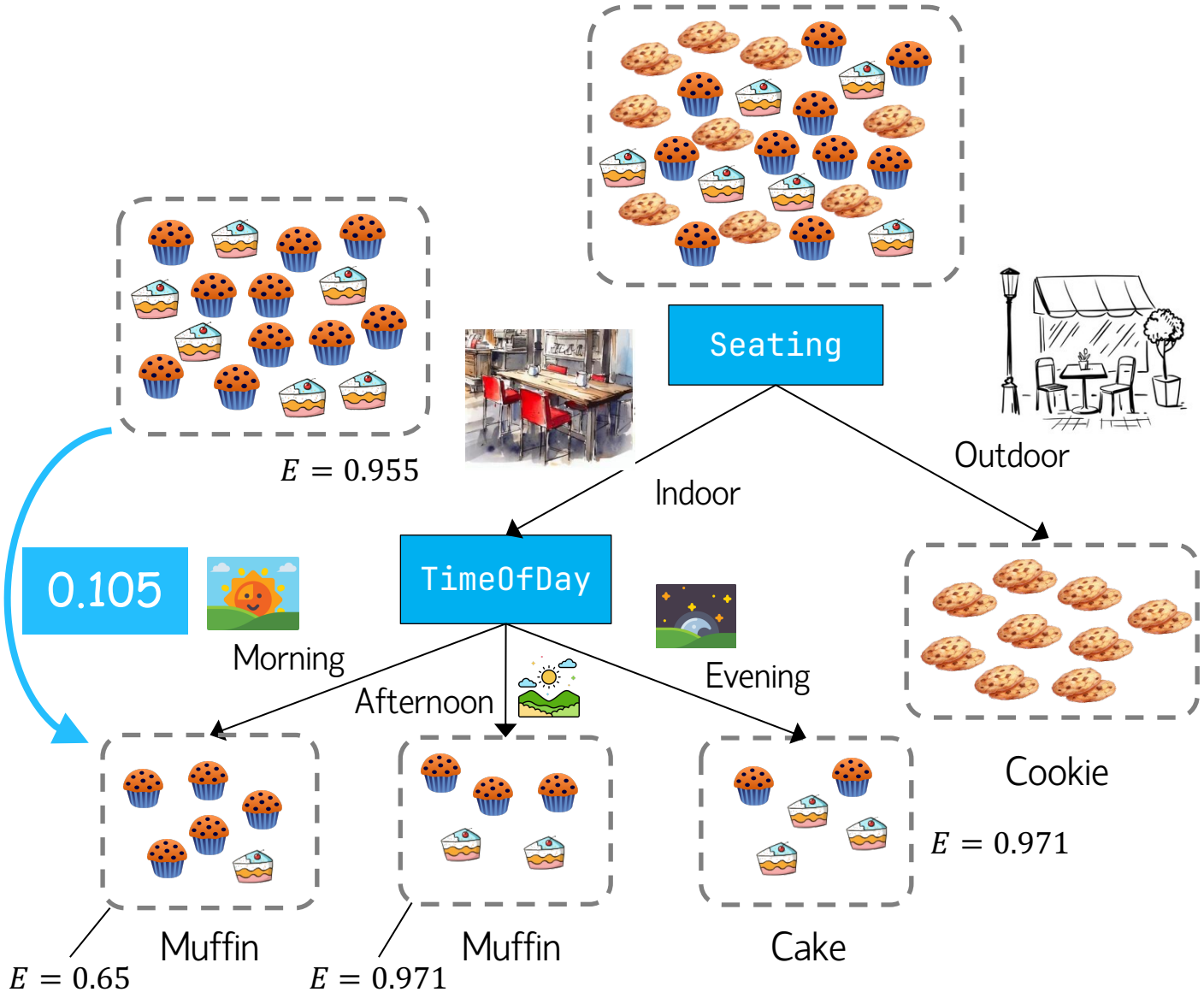
Decision Tree Learning (5 of 8)



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

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

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

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

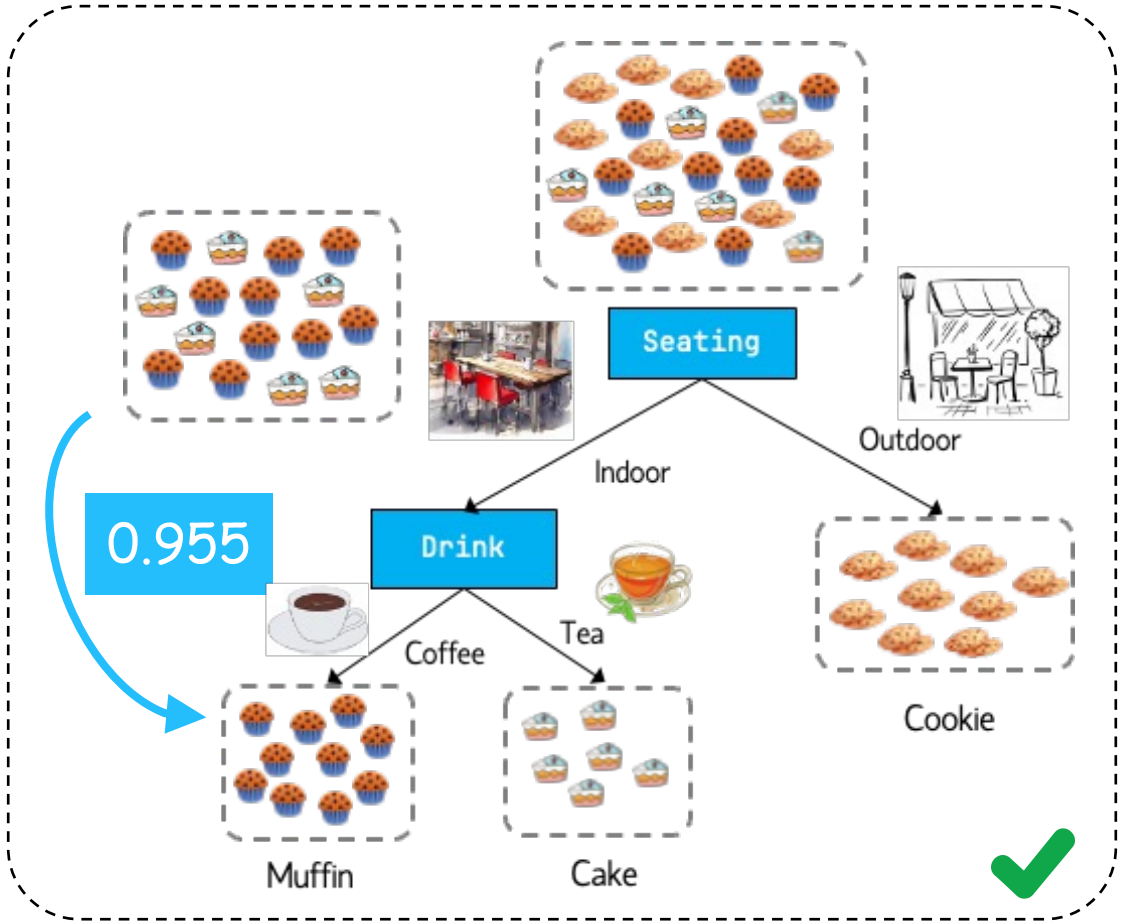
Evening: $E = -\left(\frac{2}{5}\log_2\left(\frac{2}{5}\right) + \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right)$

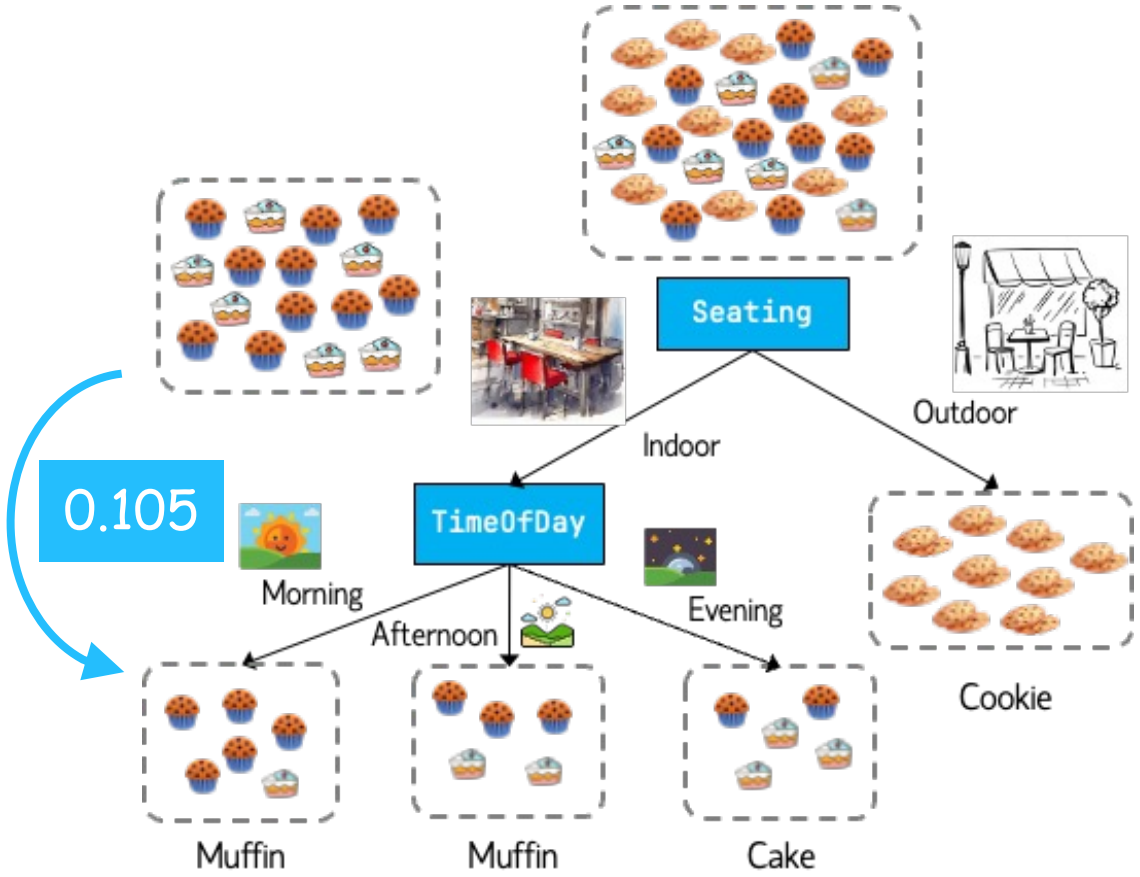
$= -(-0.529 - 0.442) = 0.971$

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

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

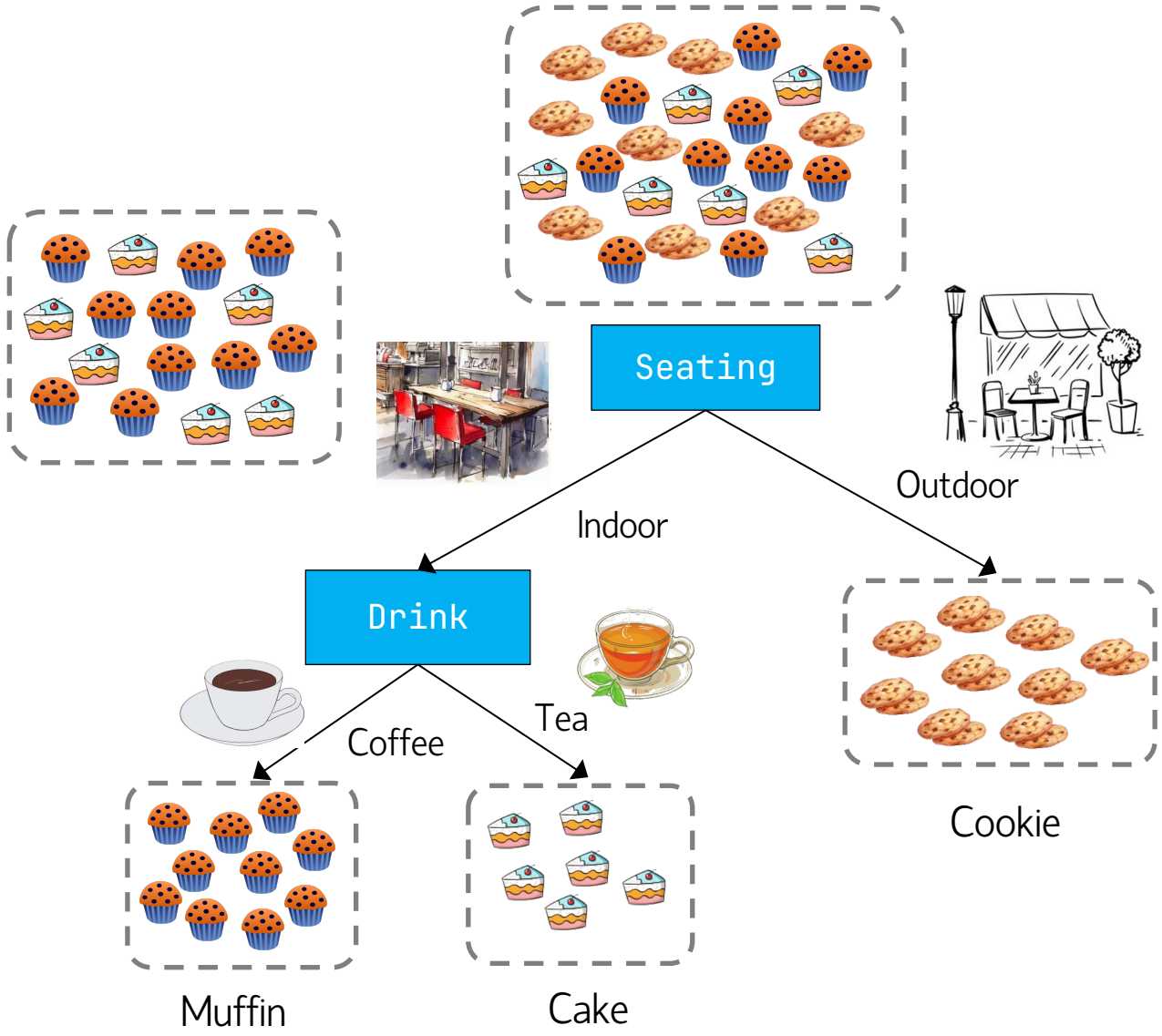
Decision Tree Learning (7 of 8)





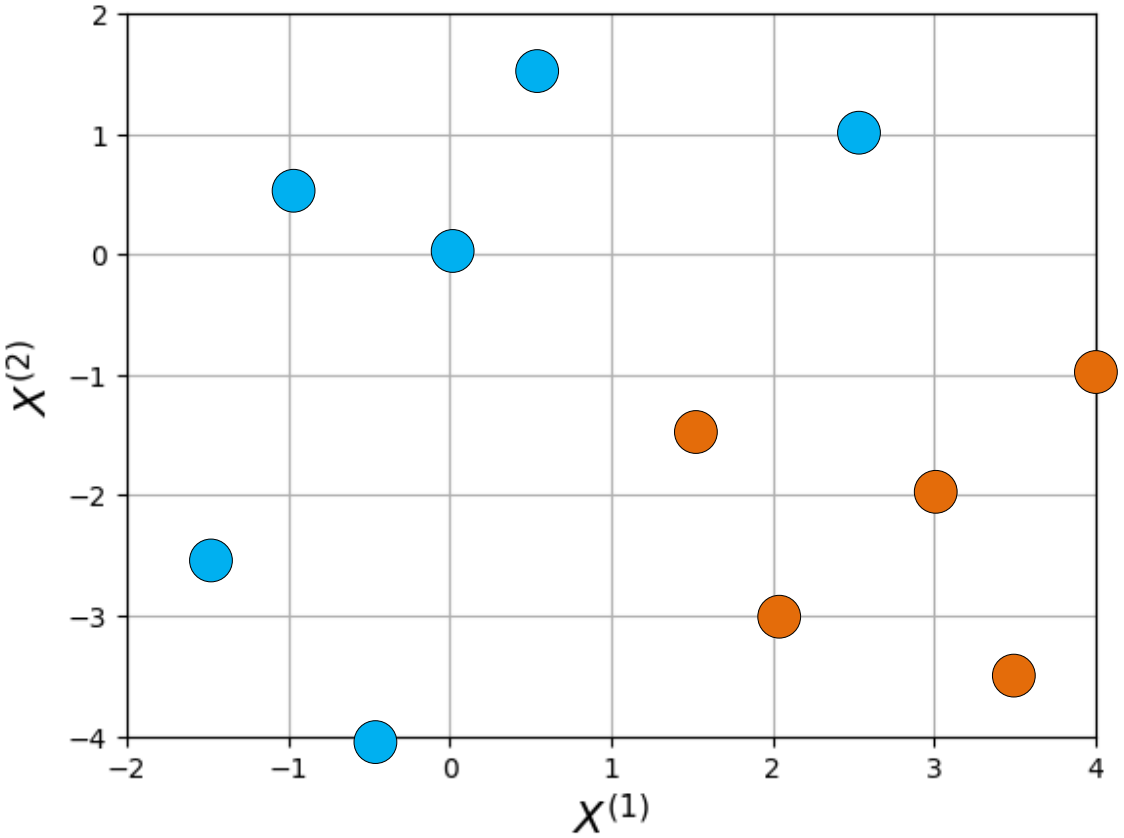
Note: 10 → Minimum Samples Split

Decision Tree Learning (8 of 8)




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

Decision Tree Learning for Numerical Features (1 of 7)



x_1 and x_2 are numerical features.

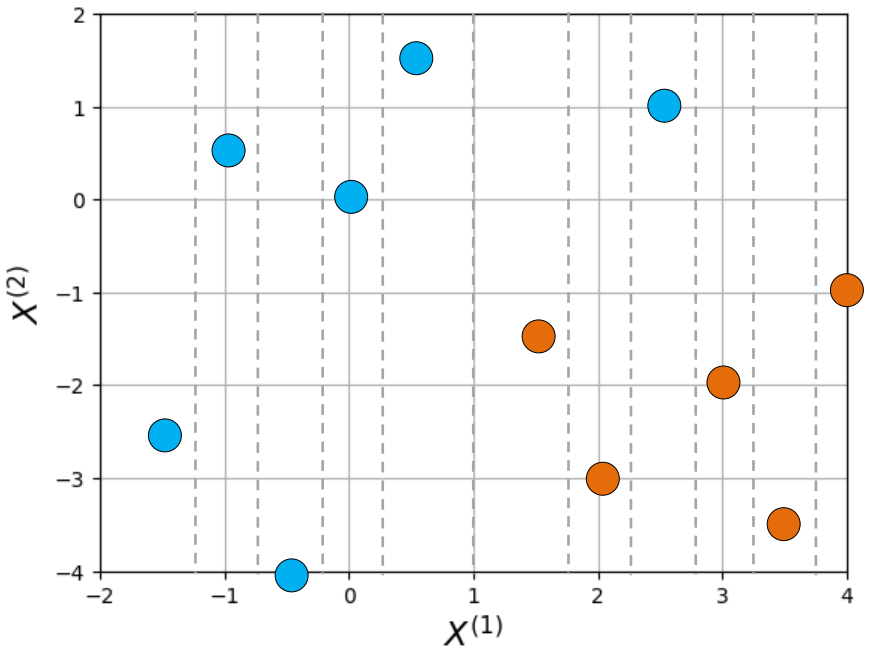
$$E = -\left(\frac{6}{11}\log_2\left(\frac{6}{11}\right) + \frac{5}{11}\log_2\left(\frac{5}{11}\right)\right)$$

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

$$= 0.994$$

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.

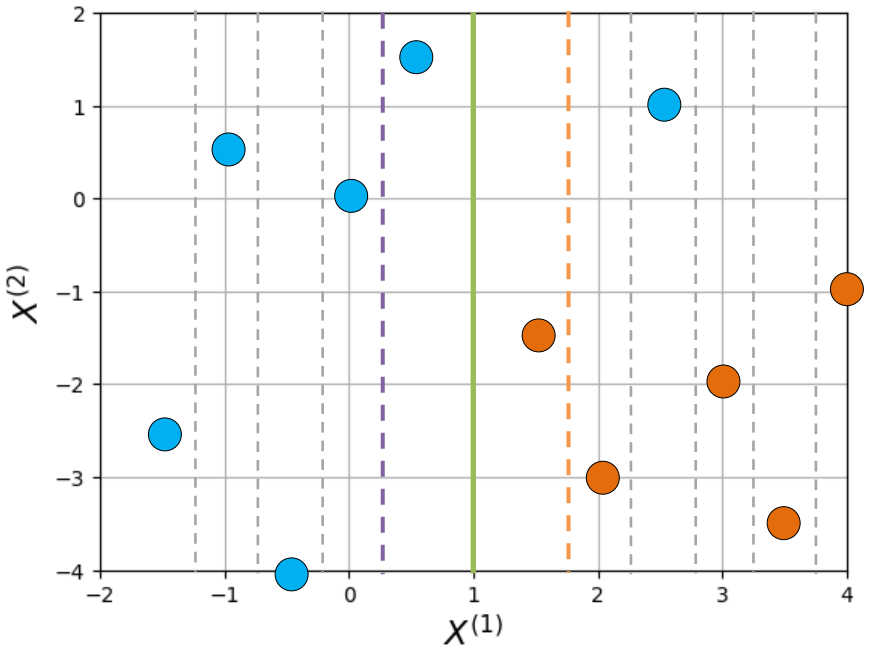
For each threshold t_k, compute entropy for each child and then their weighted average to find information gain from the split.

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)



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

$t_4 = 0.25$: $E = \frac{4}{11} \times -\left(\frac{4}{4}\log_2\frac{4}{4}\right) + \frac{7}{11} \times -\left(\frac{5}{7}\log_2\frac{5}{7} + \frac{2}{7}\log_2\frac{2}{7}\right)$

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

$= 0.549$

$t_5 = 1.0$: $E = \frac{5}{11} \times -\left(\frac{5}{5}\log_2\frac{5}{5}\right) + \frac{6}{11} \times -\left(\frac{5}{6}\log_2\frac{5}{6} + \frac{1}{6}\log_2\frac{1}{6}\right)$

$= \frac{5}{11} \times 0.00 + \frac{6}{11} \times 0.650$

$= 0.355$

$t_6 = 1.75$: $E = \frac{6}{11} \times -\left(\frac{5}{6}\log_2\frac{5}{6} + \frac{1}{6}\log_2\frac{1}{6}\right) + \frac{5}{11} \times -\left(\frac{4}{5}\log_2\frac{4}{5} + \frac{1}{5}\log_2\frac{1}{5}\right)$

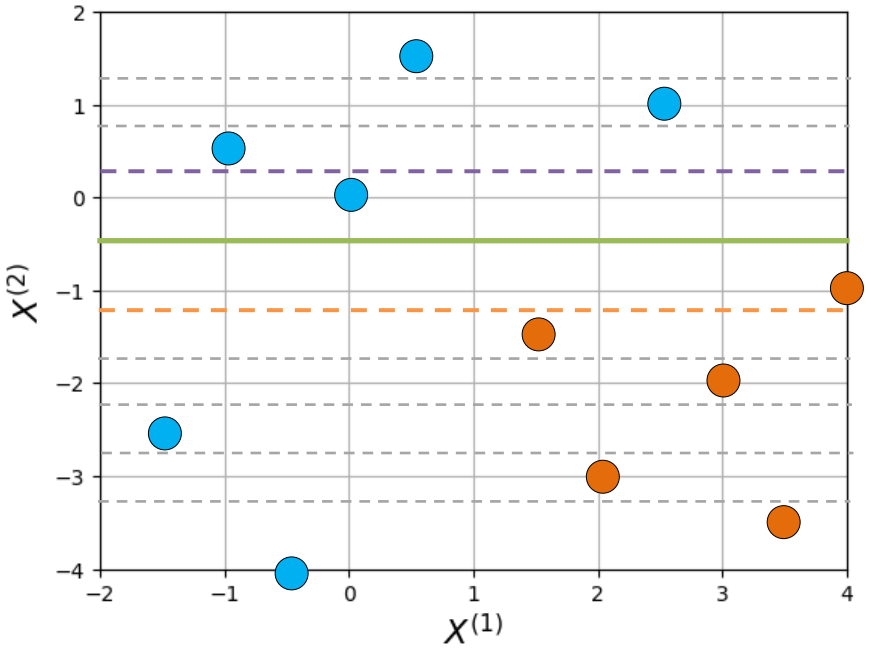
$= \frac{6}{11} \times 0.650 + \frac{5}{11} \times 0.722$

$= 0.683$

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)



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

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

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

$= 0.829$

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

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

$= 0.549$

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

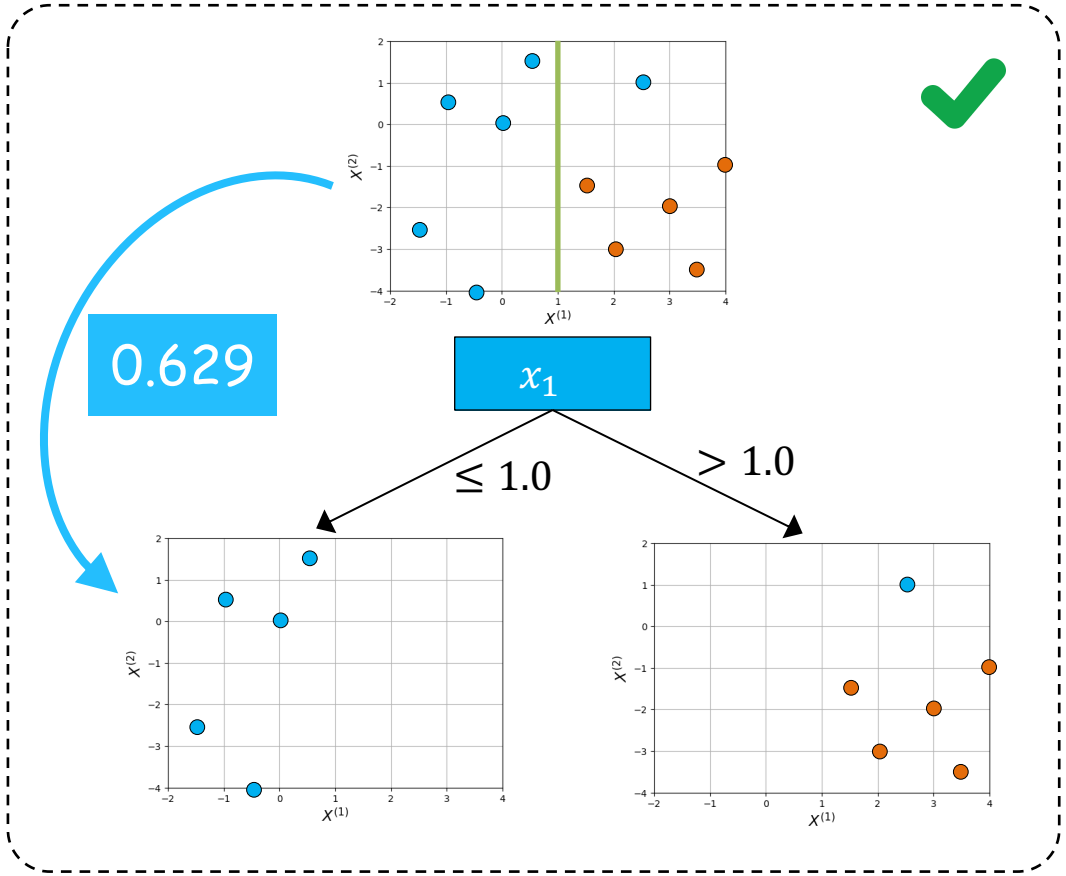
$= \frac{8}{11} \times 0.964 + \frac{3}{11} \times 0.00$

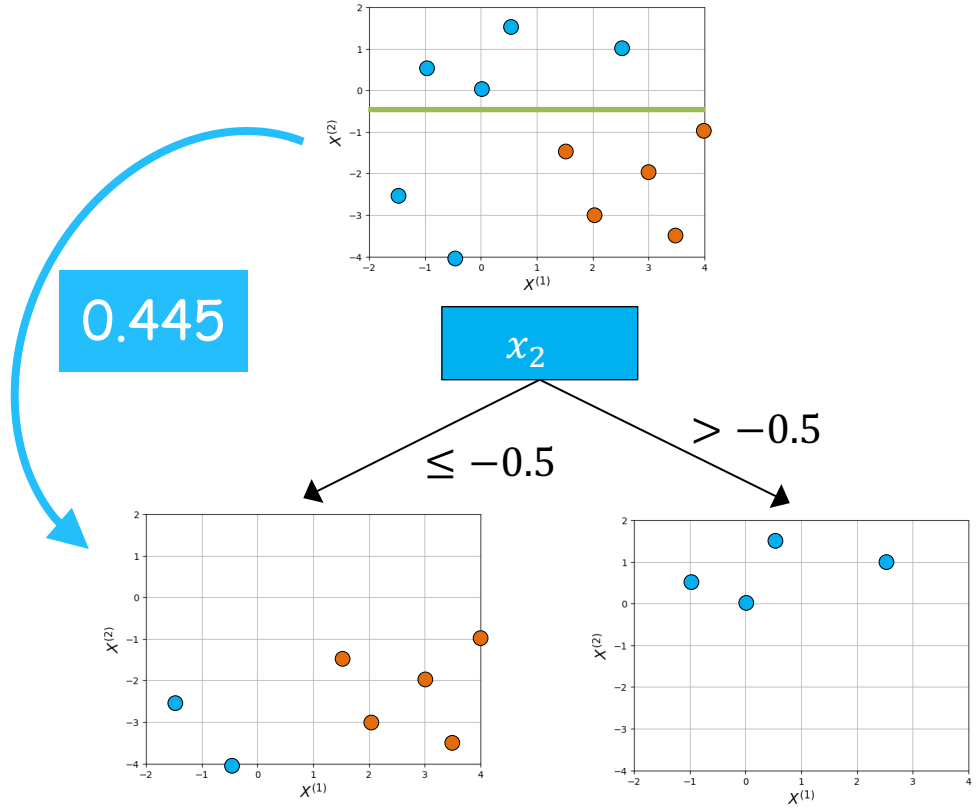
$= 0.701$

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]`

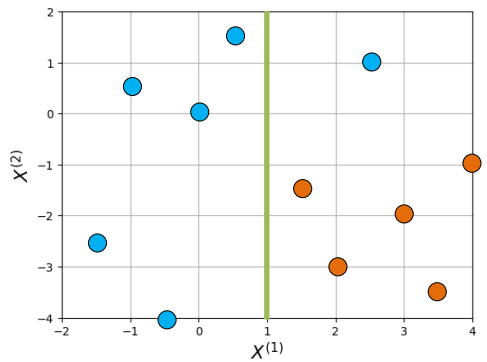
Candidate Splits : `[-3.75, -3.25, -2.75, -2.25, -1.75, -1.25, -0.5, 0.25, 0.75, 1.25]`

Decision Tree Learning for Numerical Features (5 of 7)

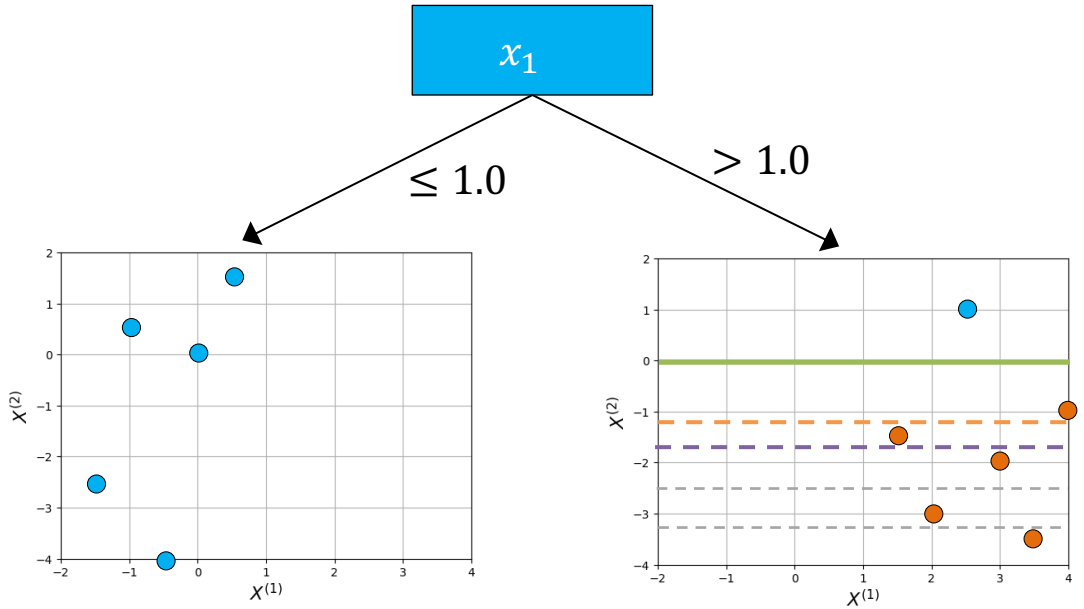




Decision Tree Learning for Numerical Features (6 of 7)



$E = 0.355$



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

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

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

$= 0.459$

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

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

$= 0.333$

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

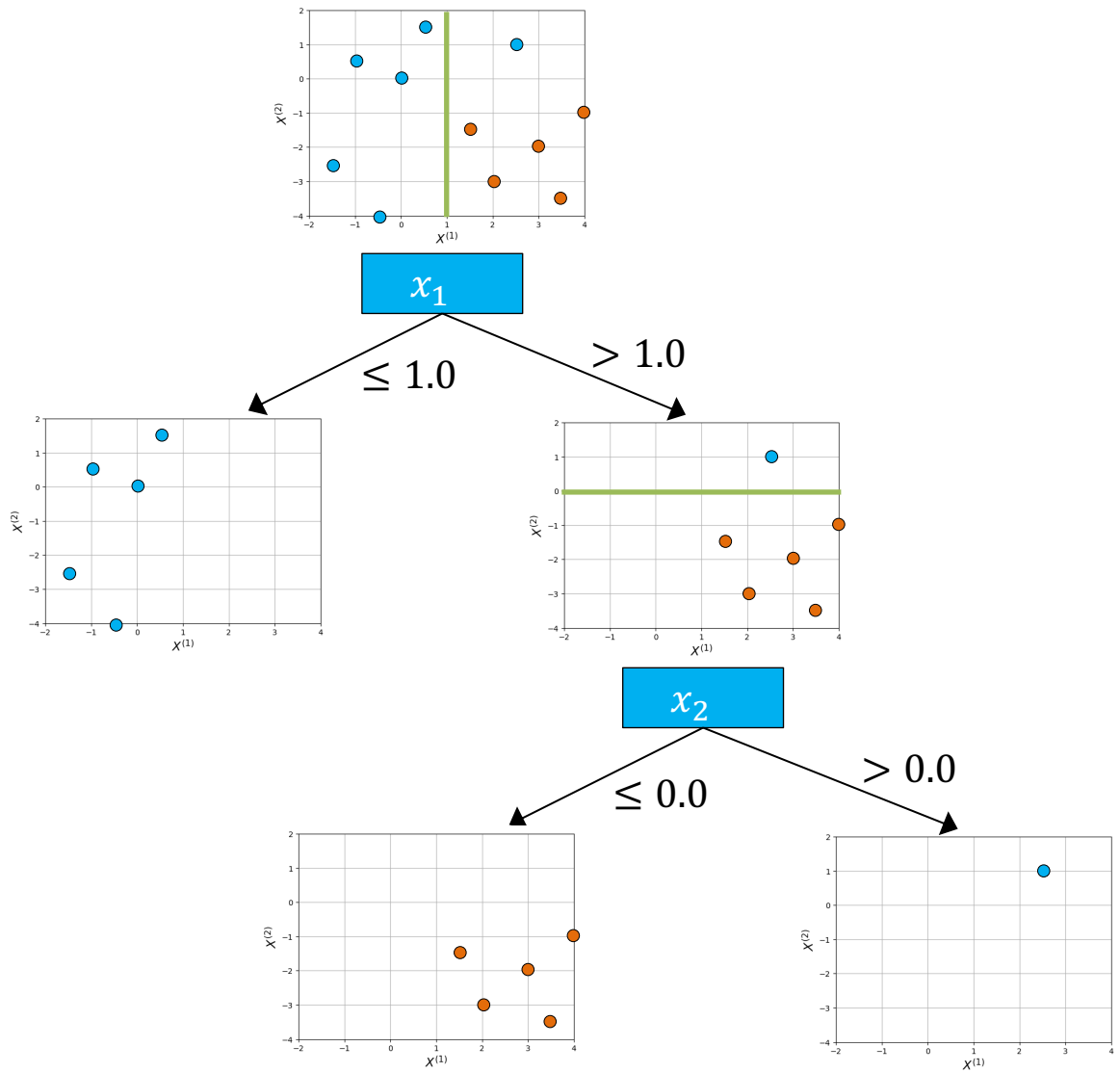
$= \frac{5}{6} \times 0.0 + \frac{1}{6} \times 0.0$

$= 0.0$

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
```

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.

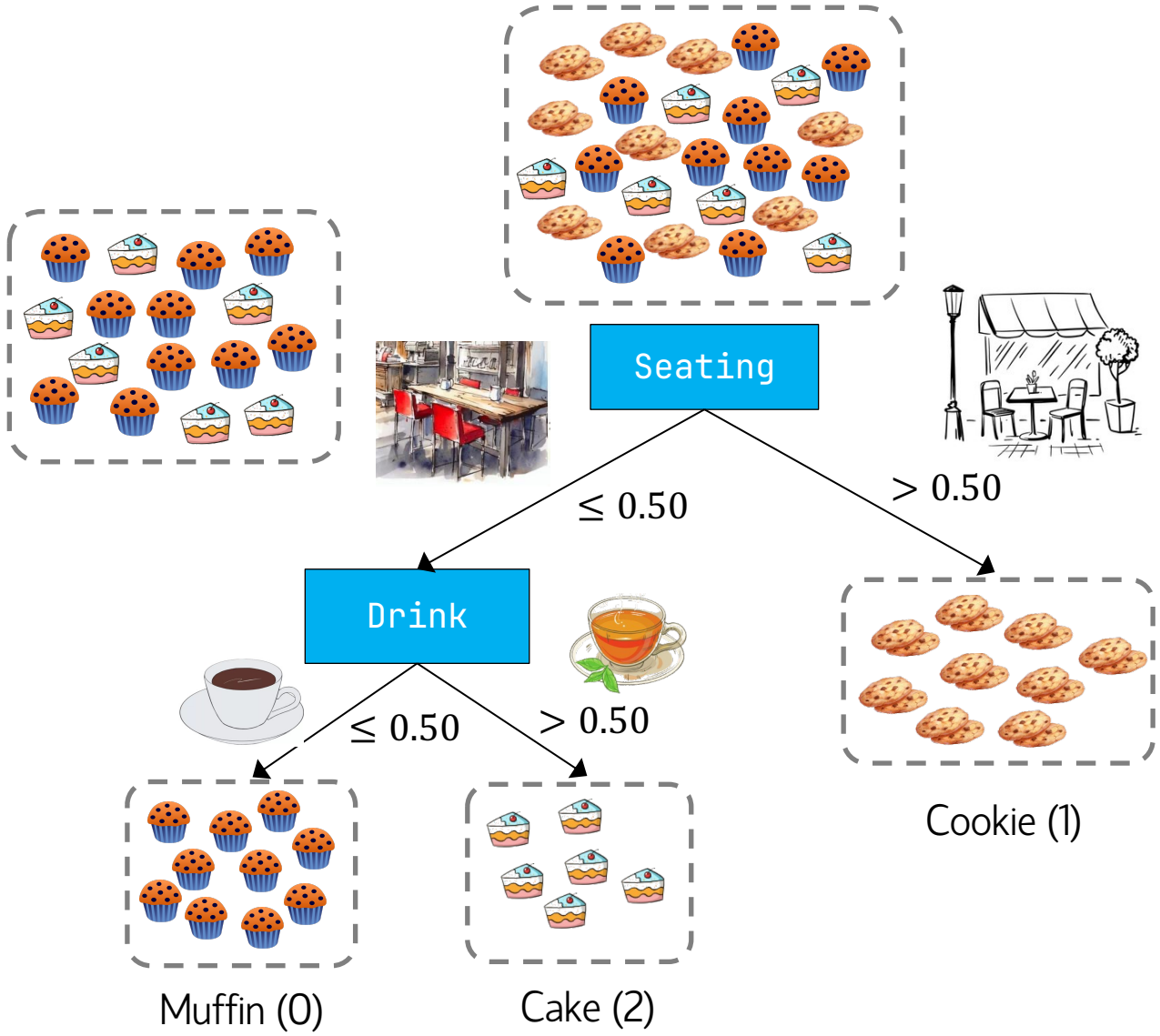
Seating
0: Indoor
1: Outdoor

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 1: Outdoor	0: Coffee 1: Tea
TimeOfDay	**Dessert**
0: Morning 1: Afternoon 2: Evening	0: Muffin 1: Cookie 2: Cake

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

Pseudocode for Decision Tree Classifier

```
# Inputs
#  X, y           ← data (N×D, N)
#  max_depth
#  min_leaf
#  X_query        ← examples to predict

# ----- fit -----
Define BUILD(X, y, depth):
  IF all labels in y equal OR depth = max_depth OR |y| < 2·min_leaf THEN
    RETURN Leaf( class = majority(y) )

  N    ← |y|
  base ← - Σ_k p_k log2 p_k            # p_k = freq(y=k)/N ; define 0·log 0 = 0
  best_gain ← 0 ; best ← NONE

  FOR j = 1 TO D DO
    thresholds ← midpoints of sorted unique X[:, j]
    FOR each t IN thresholds DO
      L ← {i : X[i, j] ≤ t} ; R ← {i : X[i, j] > t}
      IF |L| < min_leaf OR |R| < min_leaf THEN CONTINUE
      H_L ← - Σ_k p_Lk log2 p_Lk        # p_Lk = freq(y[L]=k)/|L|
      H_R ← - Σ_k p_Rk log2 p_Rk        # p_Rk = freq(y[R]=k)/|R|
      gain ← base - (|L|/N)·H_L - (|R|/N)·H_R
      IF gain > best_gain THEN best_gain ← gain ; best ← (j, t, L, R)

  IF best = NONE THEN
    RETURN Leaf( class = majority(y) )

  (j*, t*, L, R) ← best
  left  ← BUILD(X[L, :], y[L], depth + 1)
  right ← BUILD(X[R, :], y[R], depth + 1)
  RETURN Node(feature=j*, threshold=t*, left=left, right=right)

tree ← BUILD(X, y, 0)
```

```
...cont...

# ----- predict -----
ŷ ← list of length |X_query|
FOR i = 1 TO |X_query| DO
  node ← tree
  WHILE node is Node DO
    IF X_query[i][node.feature] ≤ node.threshold THEN node ← node.left
    ELSE node ← node.right
  END WHILE
  ŷ[i] ← node.class
END FOR

RETURN ŷ
```

Decision Tree Classification from Scratch

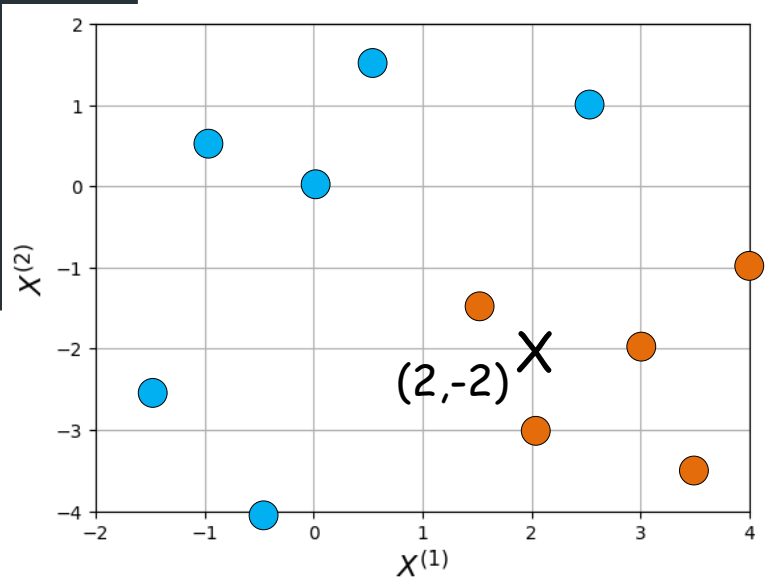
```
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin

class MyDecisionTreeClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, max_depth=3, min_samples_leaf=1):
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf

    def fit(self, X, y):
        X = np.asarray(X, float); y = np.asarray(y)
        self.classes_, y_idx = np.unique(y, return_inverse=True)
        self.n_classes_ = self.classes_.size
        self.tree_ = self._build(X, y_idx, 0)
        return self

    def predict(self, X):
        proba = self.predict_proba(X)
        return self.classes_[np.argmax(proba, axis=1)]

    def predict_proba(self, X):
        X = np.asarray(X, float)
        out = np.zeros((X.shape[0], self.n_classes_), float)
        for i, x in enumerate(X):
            node = self.tree_
            while not node["leaf"]:
                node = node["left"] if x[node["feat"]] ⩽ node["thr"] else
node["right"]
            out[i] = node["proba"]
        return out

    # ---- internals ----
    def _entropy(self, y_idx):
        counts = np.bincount(y_idx, minlength=self.n_classes_).astype(float)
        s = counts.sum()
        if s == 0: return 0.0
        p = counts / s
        p = p[p > 0]
        return float(-(p * np.log2(p)).sum())
```

```
...cont...

    def _leaf(self, y_idx):
        counts = np.bincount(y_idx, minlength=self.n_classes_).astype(float)
        proba = counts / counts.sum()
        return {"leaf": True, "proba": proba}


    def _build(self, X, y_idx, depth):
        N, D = X.shape
        if depth ⩾ self.max_depth or N < 2 * self.min_samples_leaf or
np.all(y_idx == y_idx[0]):
            return self._leaf(y_idx)

        base = self._entropy(y_idx)
        best_gain, best = 0.0, None

        for j in range(D):
            vals = np.unique(X[:, j])
            if vals.size ⩽ 1: continue
            thrs = (vals[:-1] + vals[1:]) / 2.0
            for t in thrs:
                L = X[:, j] ⩽ t
                nL, nR = int(L.sum()), N - int(L.sum())
                if nL < self.min_samples_leaf or nR < self.min_samples_leaf:
continue
                H_L = self._entropy(y_idx[L])
                H_R = self._entropy(y_idx[~L])
                gain = base - (nL / N) * H_L - (nR / N) * H_R
                if gain > best_gain:
                    best_gain, best = gain, (j, t, L)

        if best is None:
            return self._leaf(y_idx)

        j, t, L = best
        left  = self._build(X[L],   y_idx[L],   depth + 1)
        right = self._build(X[~L],  y_idx[~L],  depth + 1)
        return {"leaf": False, "feat": int(j), "thr": float(t), "left":
left, "right": right}
```

Usage Example

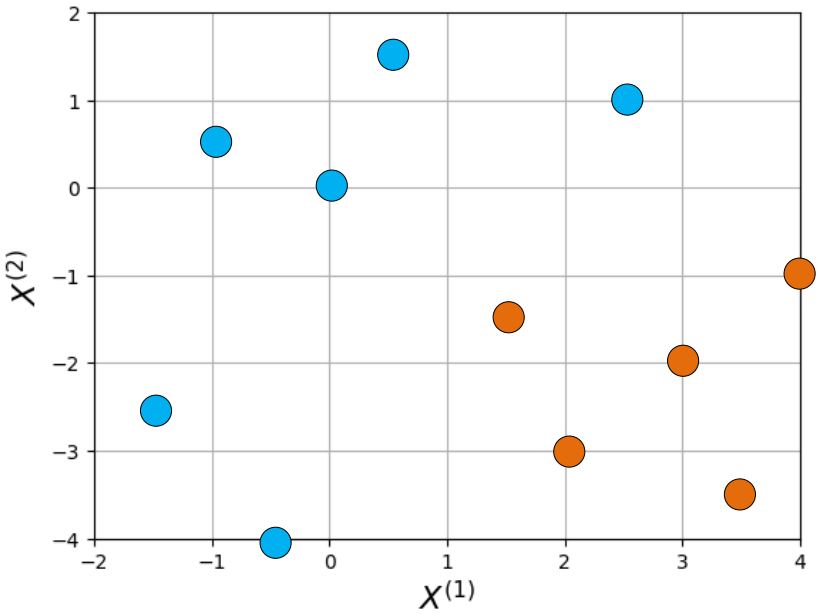
```
import numpy as np

# ----- training data -----
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, 1

# ----- fit tree (entropy, binary thresholds) -----
clf = MyDecisionTreeClassifier(max_depth=3, min_samples_leaf=1)
clf.fit(X, y)

# ----- classify a new point -----
X_test = np.array([[2.0, -2.0]])
proba = clf.predict_proba(X_test)[0]      # [P(class=0), P(class=1)]
pred  = clf.predict(X_test)[0]

# ----- outputs -----
print("Classes order:", clf.classes_)                # → [0 1]
print("New Point:", X_test[0].tolist())
print("Probabilities [class 0, class 1]:", proba.tolist())
print("Predicted Class:", int(pred))
```

```
Classes order: [0 1]
New Point: [2.0, -2.0]
# leaf is pure on the right-lower region
Probabilities [class 0, class 1]: [0.0, 1.0]
Predicted Class: 1
```



Step 1: Dataset & Root Entropy (H = 0.994)

```
def _entropy(self, y_idx):
    counts = np.bincount(y_idx, minlength=self.n_classes_).astype(float)
    p = counts / counts.sum()                          # class probs
    p = p[p > 0]
    return float(-(p * np.log2(p)).sum())              # -Σ p log₂ p

def _build(self, X, y_idx, depth):
    N, D = X.shape
    if ( depth ⩾ self.max_depth
         or N < 2*self.min_samples_leaf
         or np.all(y_idx == y_idx[0]) ):
        return self._leaf(y_idx)
    base = self._entropy(y_idx)                        # root entropy
    best_gain, best = 0.0, None
    ...
```



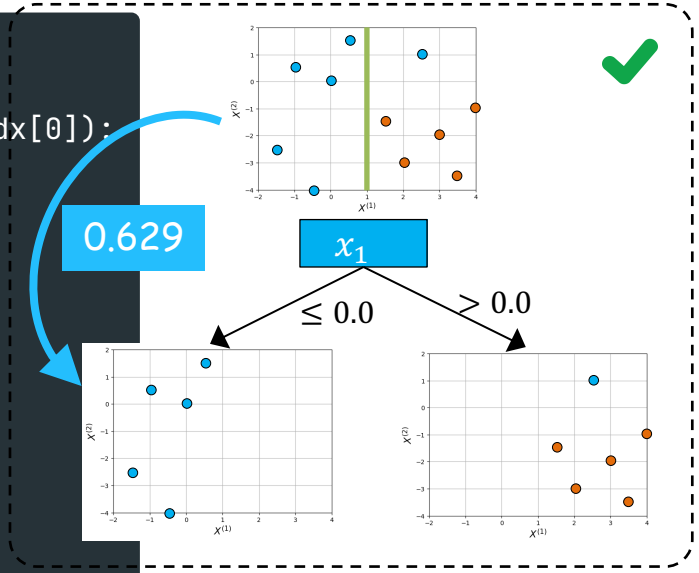
$$E = -\left(\frac{6}{11}\log_2\left(\frac{6}{11}\right) + \frac{5}{11}\log_2\left(\frac{5}{11}\right)\right)$$

$$= 0.994$$

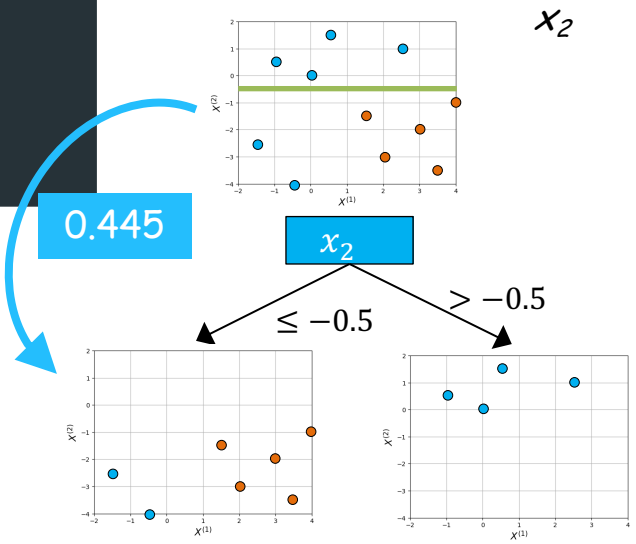
Step 2: Evaluate Candidate Splits (Information Gain)

```python
def _build(self, X, y_idx, depth):
    N, D = X.shape
    if depth ⩾ self.max_depth or N < 2*self.min_samples_leaf or np.all(y_idx == y_idx[0]):
        return self._leaf(y_idx)
    base = self._entropy(y_idx)

    best_gain, best = 0.0, None
    for j in range(D):
        vals = np.unique(X[:, j])
        if vals.size ⩽ 1:
            continue
        thrs = (vals[:-1] + vals[1:]) / 2.0           # mid-point thresholds
        for t in thrs:
            L = X[:, j] ⩽ t                           # split mask
            nL, nR = int(L.sum()), N - int(L.sum())   # child sizes
            if nL < self.min_samples_leaf or nR < self.min_samples_leaf:
                continue
            H_L = self._entropy(y_idx[L])             # left entropy
            H_R = self._entropy(y_idx[~L])            # right entropy
            gain = base - (nL/N)*H_L - (nR/N)*H_R     # information gain
            if gain > best_gain:                      # keep best
                best_gain, best = gain, (j, t, L)
```

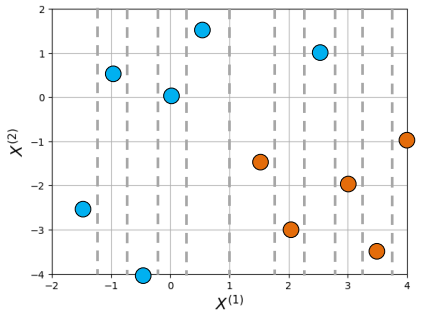
x_1



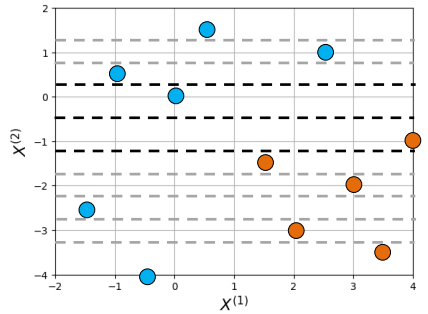
x_2



x_1

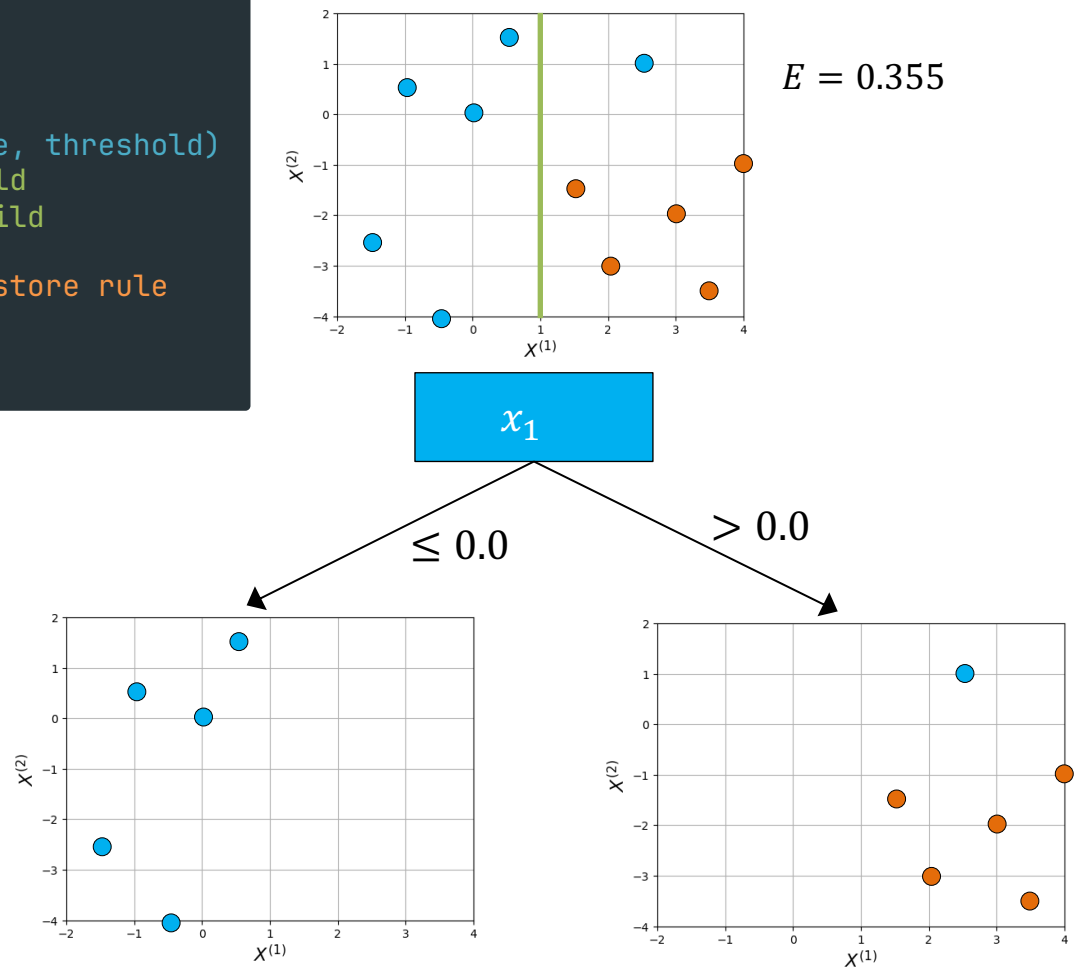


x_2



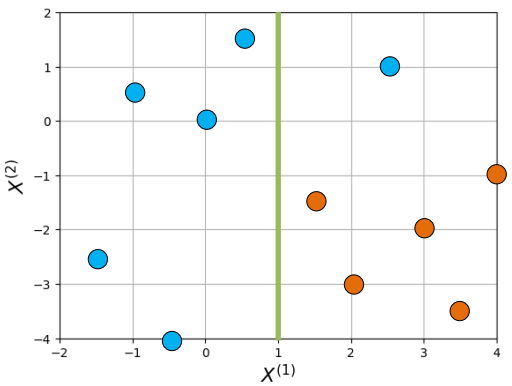
Step 3: Apply Best Split: x_1 @ 1.0

```
...

if best is None:
    return self._leaf(y_idx)

j, t, L = best                                          # chosen (feature, threshold)
left  = self._build(X[L],  y_idx[L],  depth + 1)        # build left child
right = self._build(X[~L], y_idx[~L], depth + 1)        # build right child
return {
    "leaf": False, "feat": int(j), "thr": float(t),     # ← highlight: store rule
    "left": left, "right": right
}
```

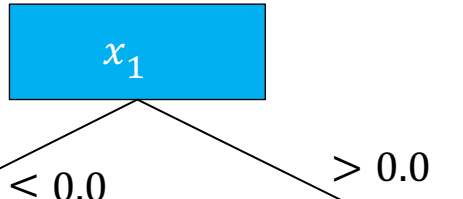


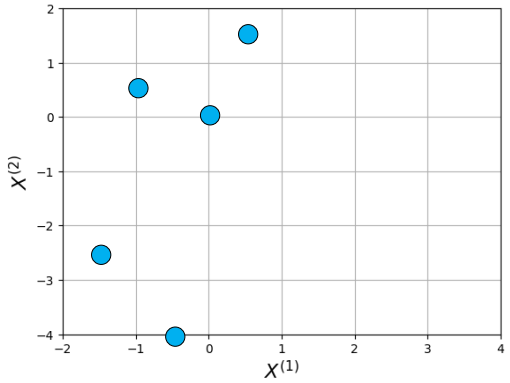
Step 4 — Child Entropies & Weighted Remainder (E = 0.355)

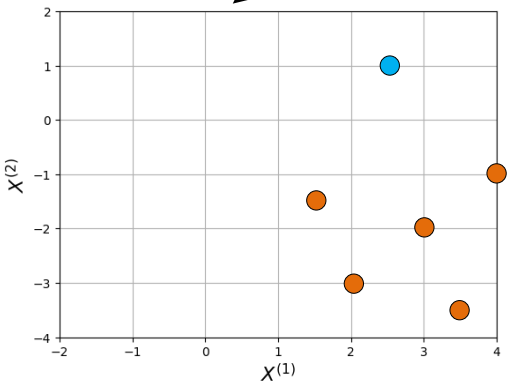
```
H_L = self._entropy(y_idx[L])
H_R = self._entropy(y_idx[~L])
E_after = (nL/N)*H_L + (nR/N)*H_R  # weighted remainder

# (info gain shown for completeness)
gain = base - E_after
```



$E = 0.355$





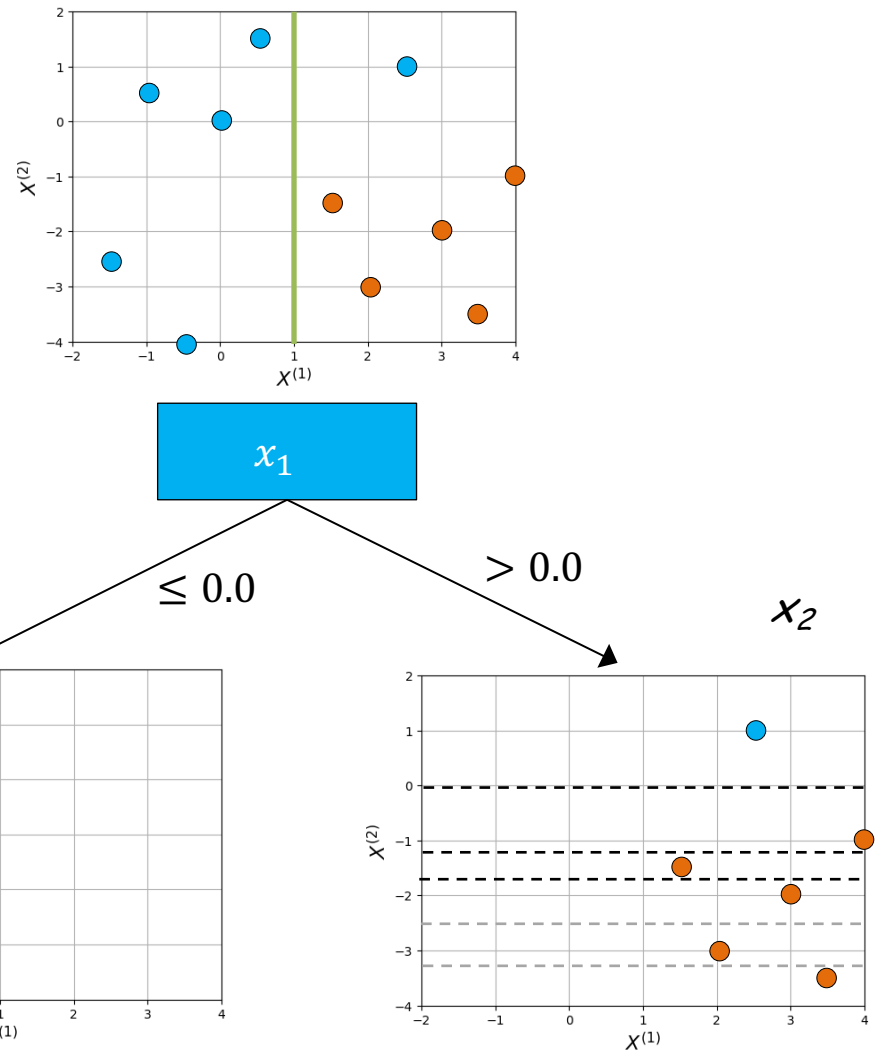


$$E = -\left(\frac{5}{6}\log_2\frac{5}{6} + \frac{1}{6}\log_2\frac{1}{6}\right)$$

$$= 0.355$$

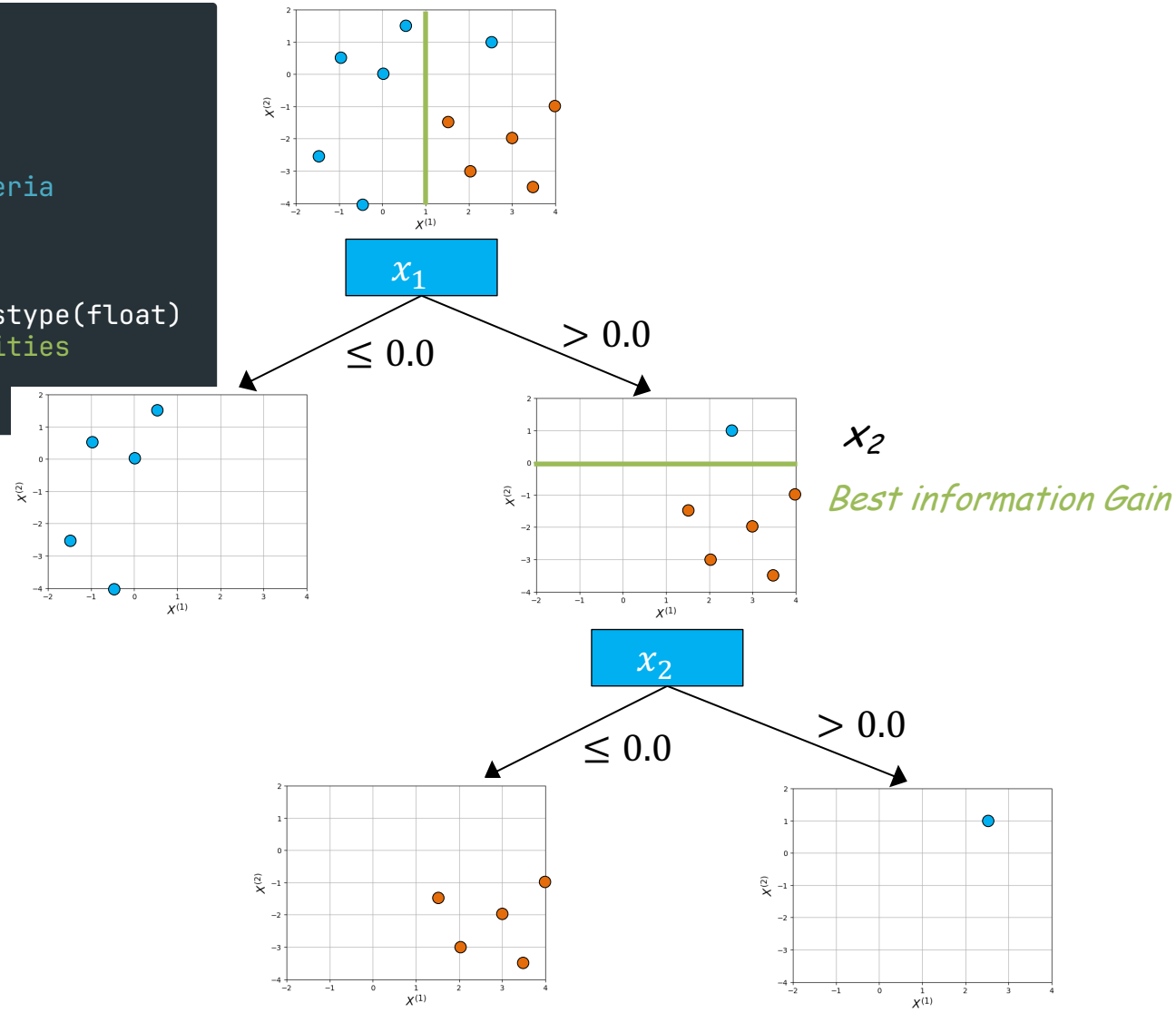
Step 5: Search Next Split on Right Child

```
right = self._build(X[~L], y_idx[~L], depth + 1)       # recurse on right

# The same loop from Slide 2 runs inside this call; best becomes x₂ @ 0.0
```



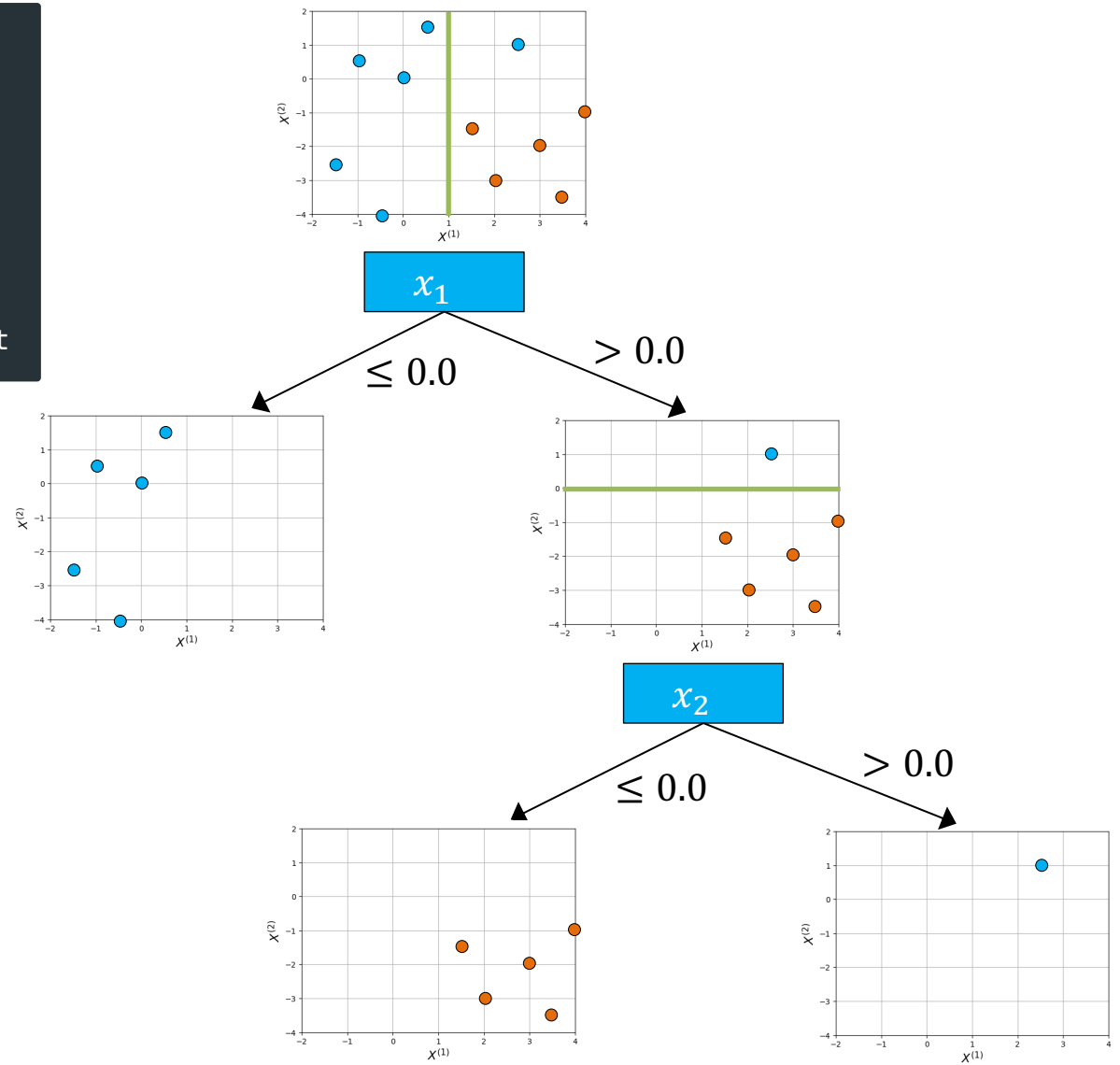
Step 6: Apply Best Second Split: x_2 @ 0.0 → Leaves

```
def _build(self, X, y_idx, depth):
    N, D = X.shape
    if (depth ⩾ self.max_depth or
        N < 2*self.min_samples_leaf or
        np.all(y_idx == y_idx[0])):          # stopping criteria
        return self._leaf(y_idx)

def _leaf(self, y_idx):
    counts = np.bincount(y_idx, minlength=self.n_classes_).astype(float)
    proba = counts / counts.sum()            # leaf probabilities
    return {"leaf": True, "proba": proba}    # leaf node
```



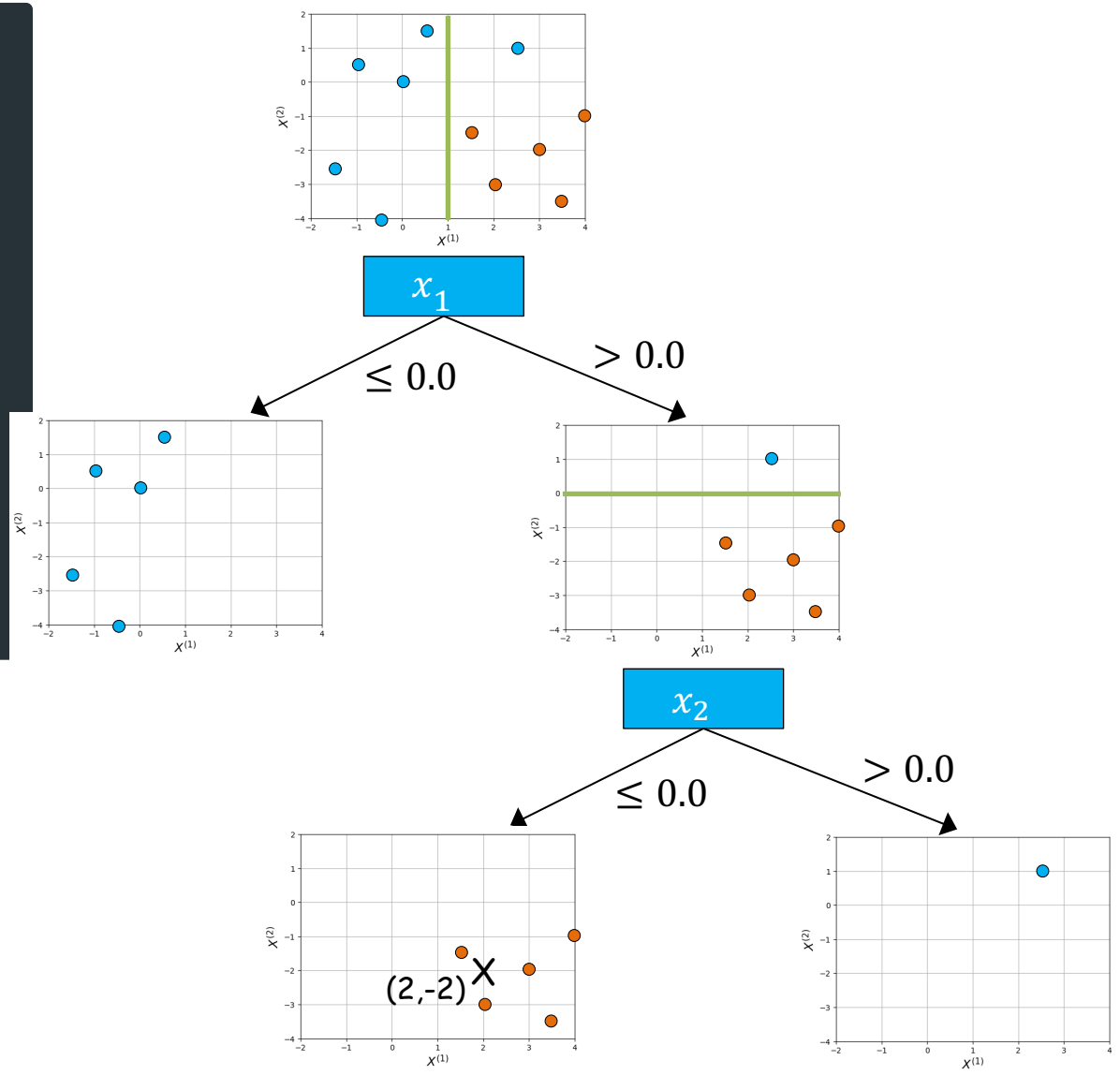
Step 7: (Optional) Tree snapshot & decision rules

```
# (what an internal node looks like in the learned tree)
node = {
    "leaf": False,
    "feat": 0,     # e.g., x₁
    "thr": 1.0,    # e.g., split at 1.0
    "left": {...}, # subtree if x[feat] ≤ thr
    "right": {...} # subtree if x[feat] > thr
}

# Rule: if x[node["feat"]] ≤ node["thr"] → left else right
```

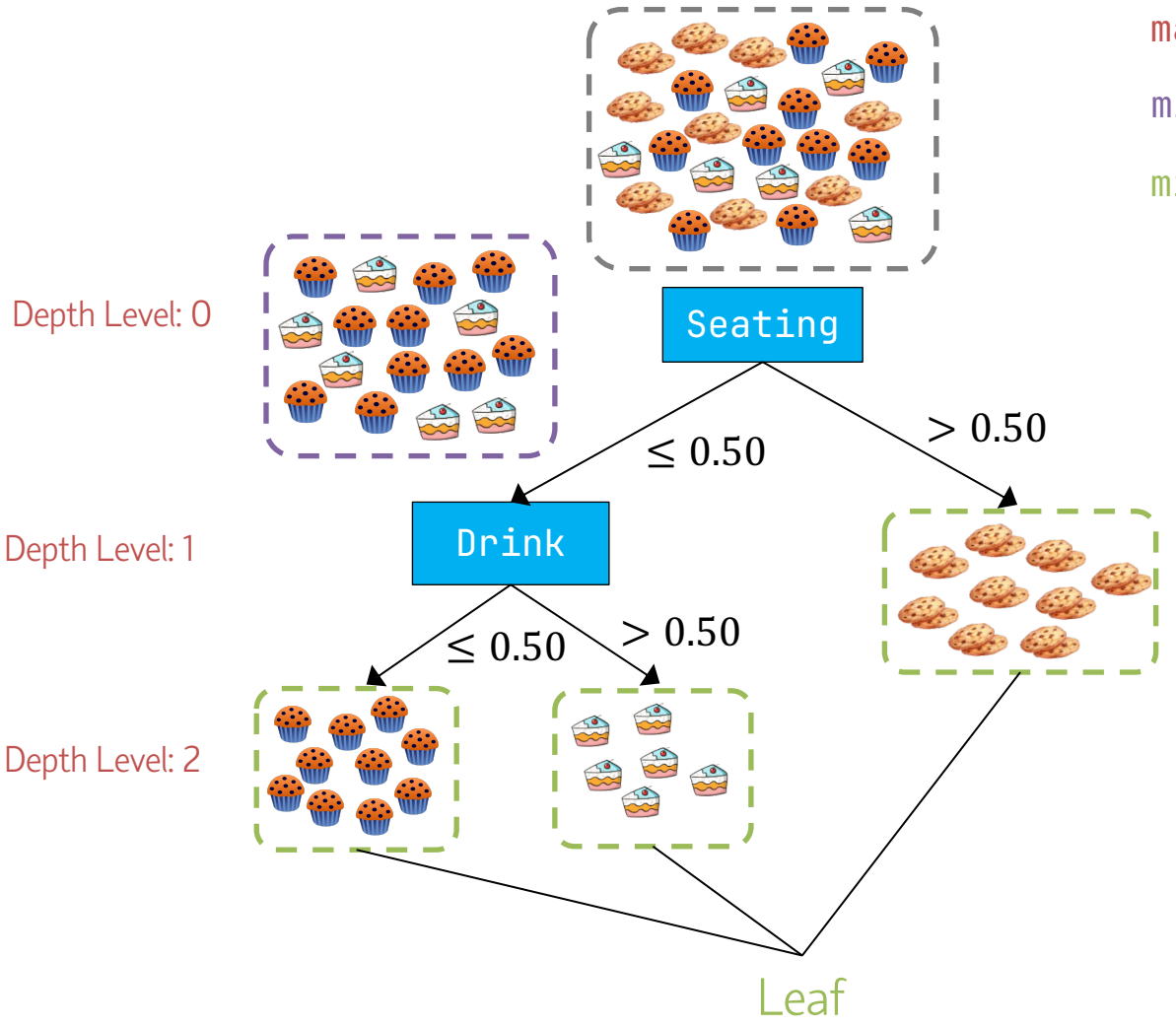


Step 8: Prediction Walk-Through: classify (2, −2)

```
def predict_proba(self, X):
    X = np.asarray(X, float)
    out = np.zeros((X.shape[0], self.n_classes_), float)
    for i, x in enumerate(X):
        node = self.tree_
        while not node["leaf"]:
            node = (
                    node["left"]
                    if x[node["feat"]] ⩽ node["thr"]
                    else node["right"]
                   )                                    # traverse
        out[i] = node["proba"]                          # leaf probs
    return out

def predict(self, X):
    proba = self.predict_proba(X)
    return self.classes_[np.argmax(proba, axis=1)]  # argmax class
```



Hyperparameters



`max_depth`: Maximum Tree Depth

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

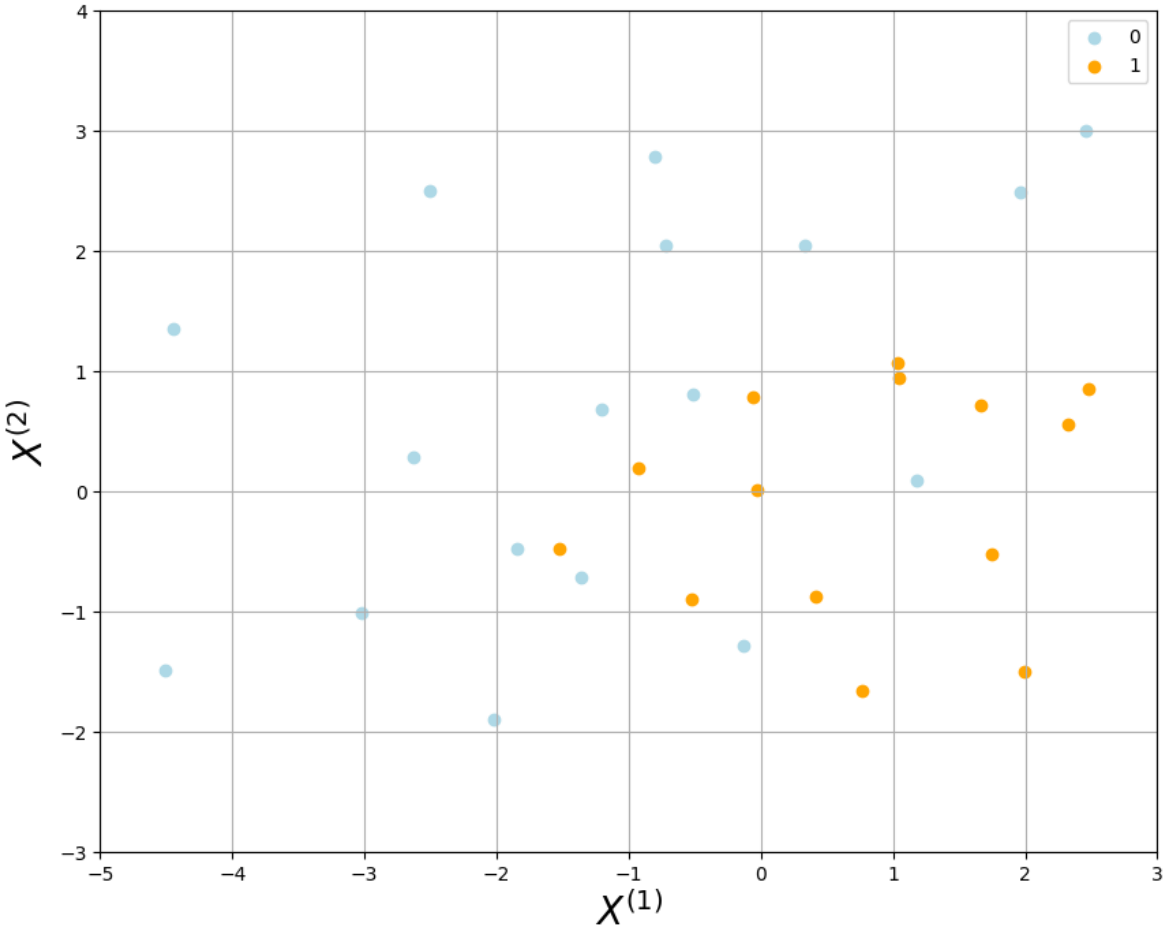
`min_samples_leaf`: Minimum # Samples required to be in a Leaf

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

`max_depth` → # Samples in a Leaf

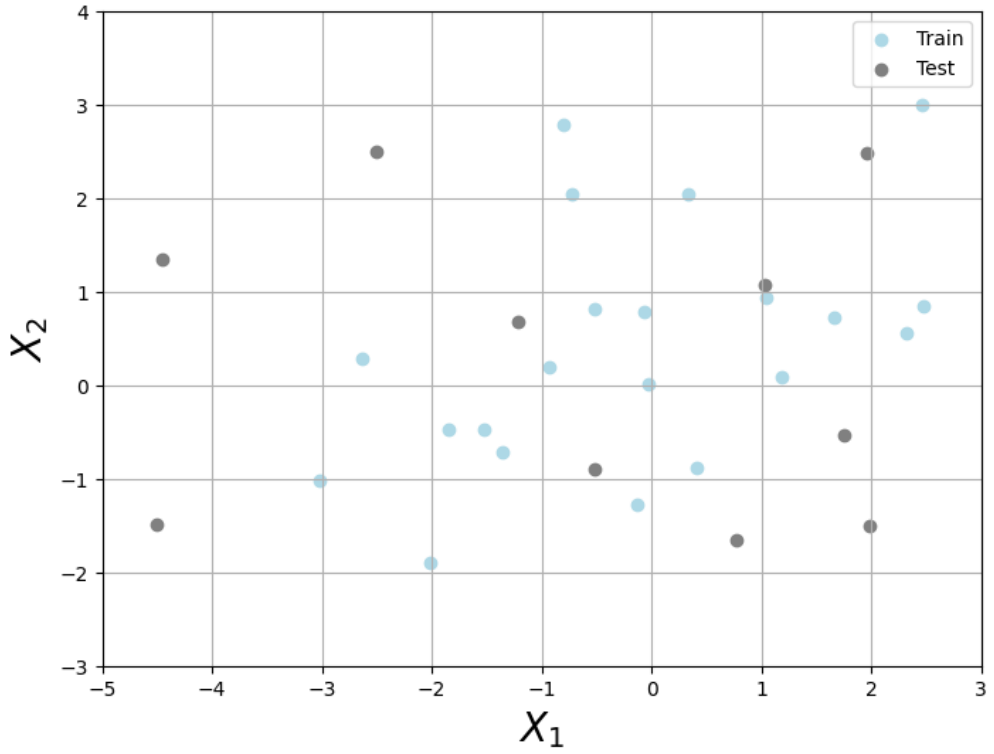
`min_samples_leaf` → Tree Depth

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)
```



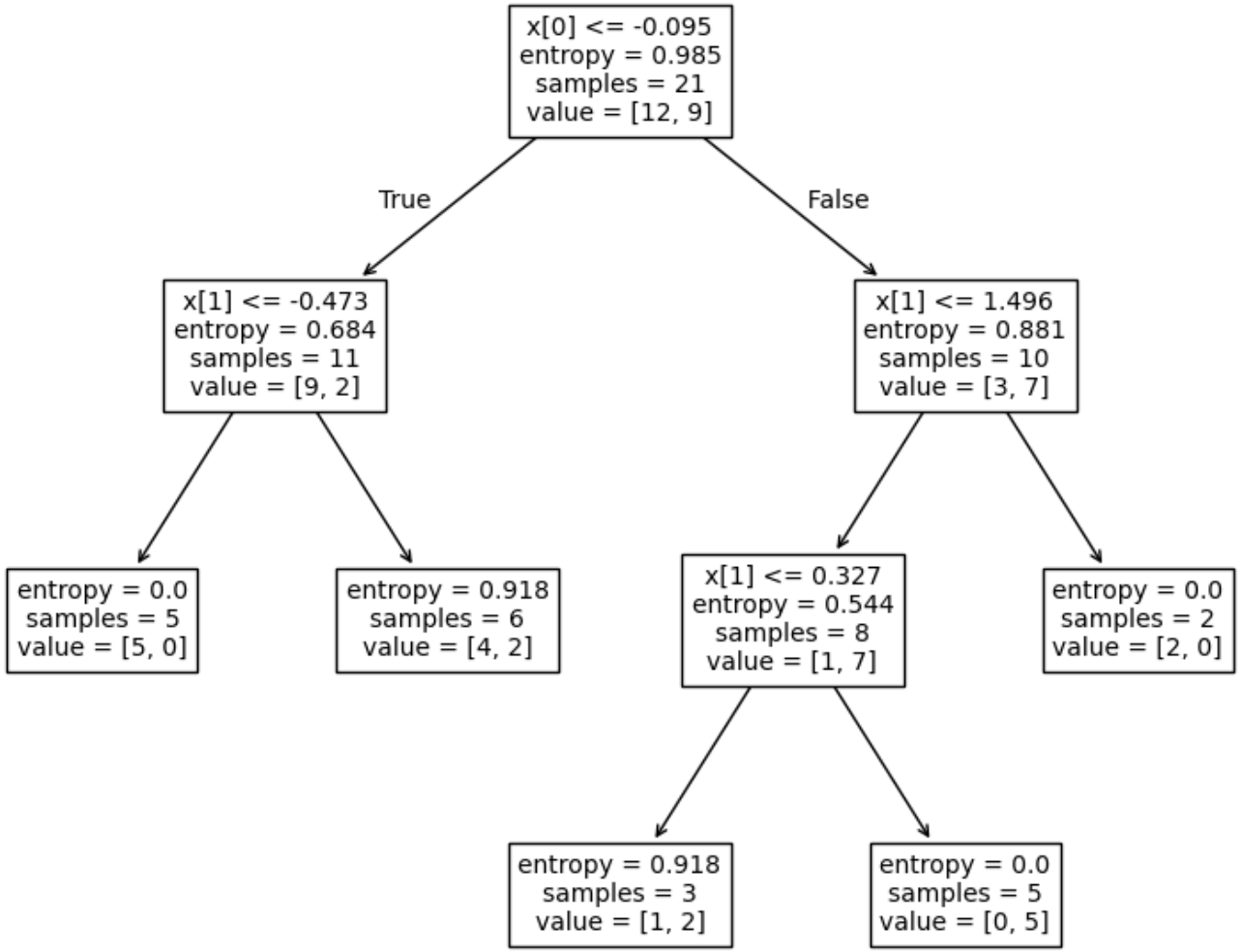
- Train:Test is typicaly either 0.8:0.2 or 0.7:0.3.
- **Test** dataset is a proxy of unseen data, and it will only be used in the final evaluation.
- **Train** dataset is further divided into k folds (e.g. 3 or 5).For each fold, the model is trained on k-1 folds and validated on the remaining fold.
- 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

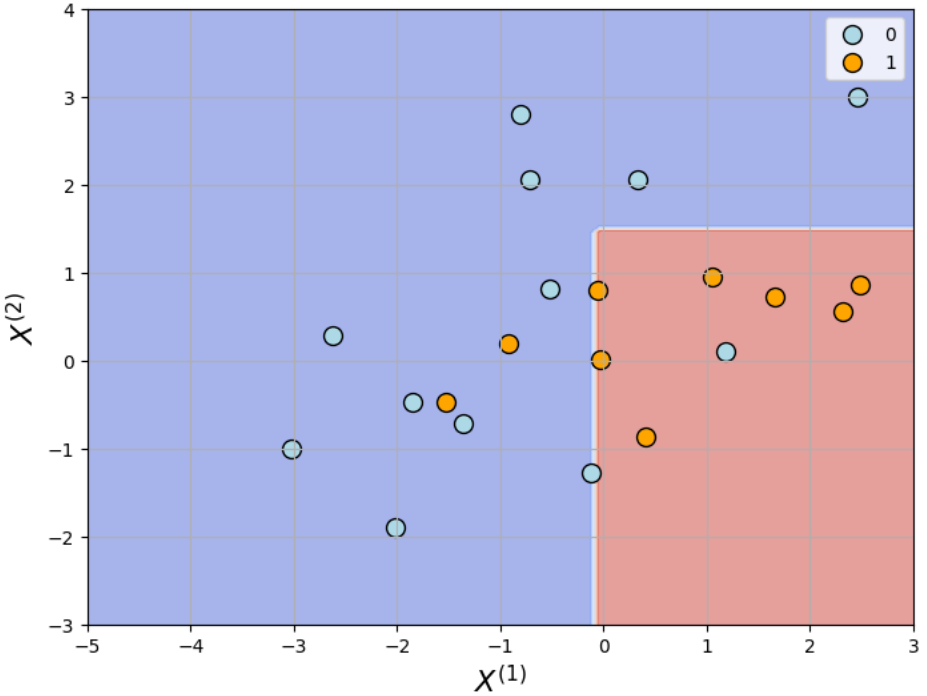
```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_val_score
# (Optional, better for classification)
# from sklearn.model_selection import StratifiedKFold as KFold

# model
min_samples_split = 8
model = DecisionTreeClassifier(
    criterion='entropy',
    min_samples_split=min_samples_split,
    random_state=204
)

# 5-fold CV
kf = KFold(n_splits=5, shuffle=True, random_state=42)
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
```

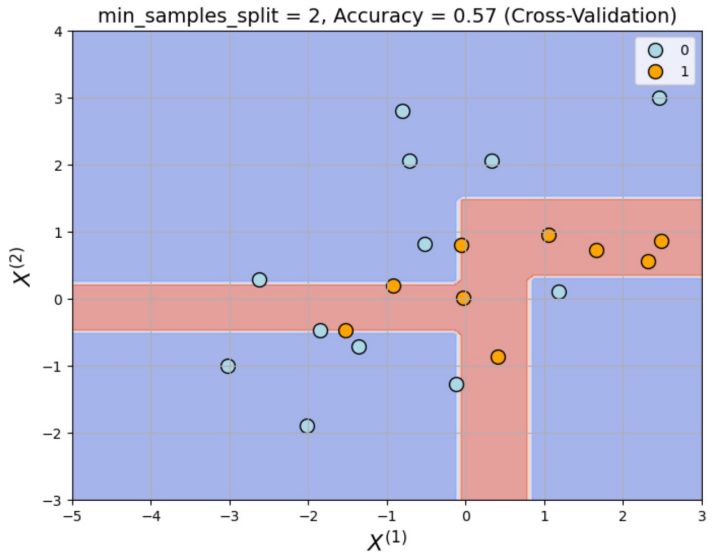
Learned Decision Tree

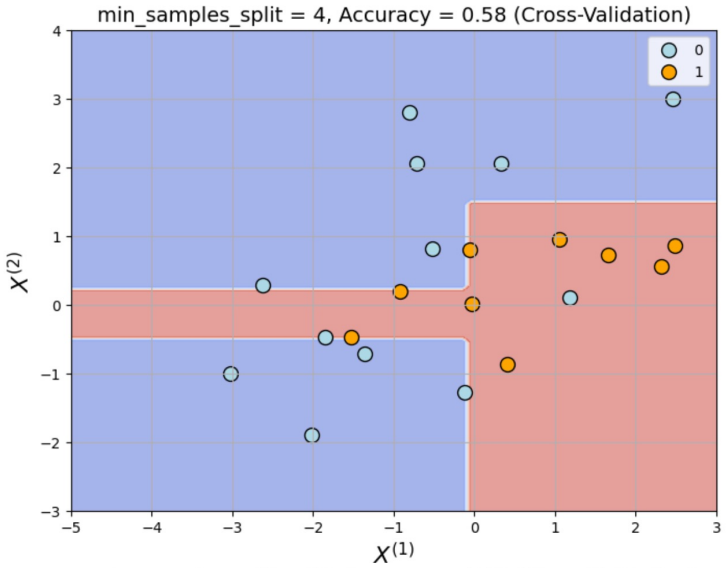


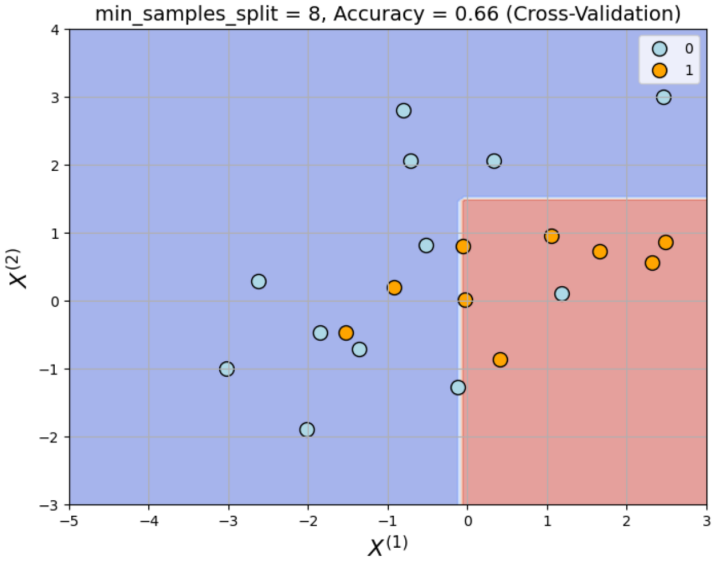
min_samples_split = 8 | Accuracy (CV) = 0.66

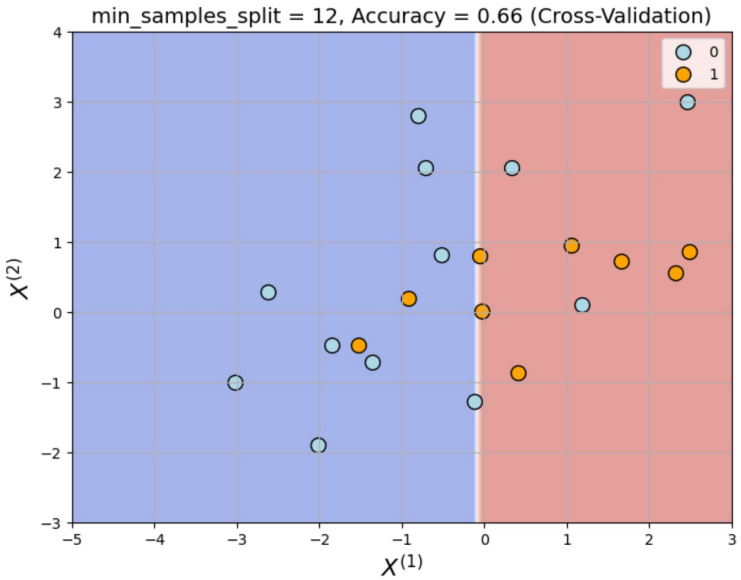


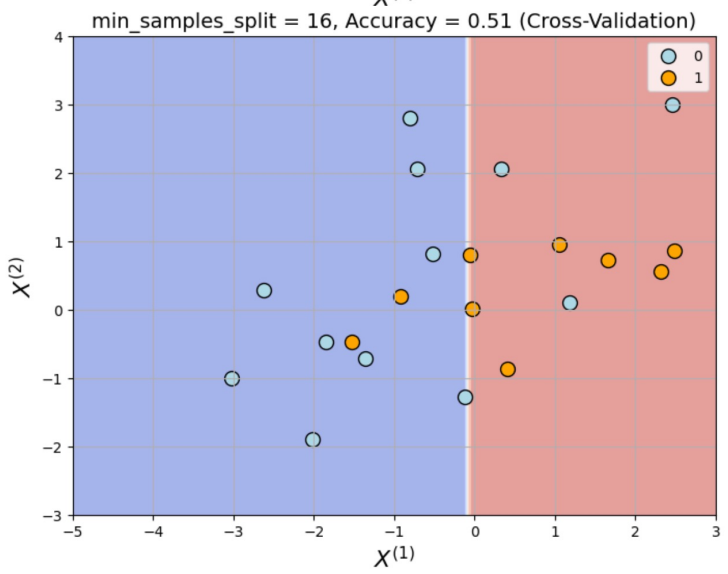
Model Complexity

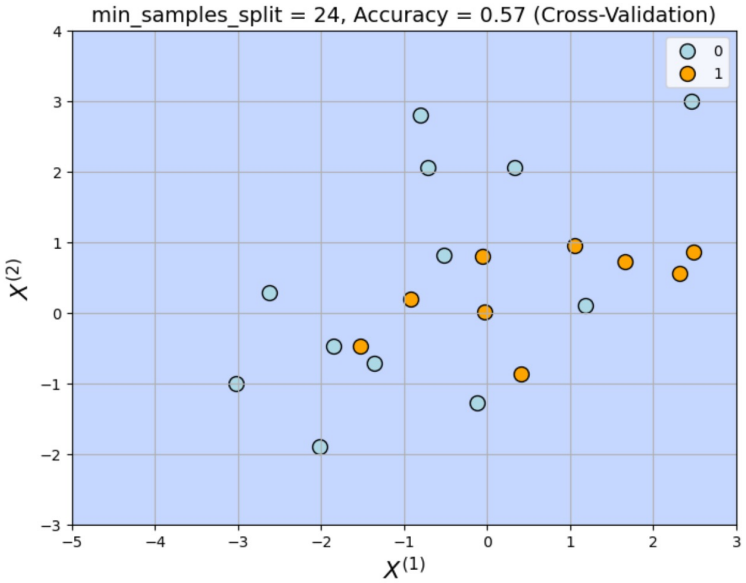




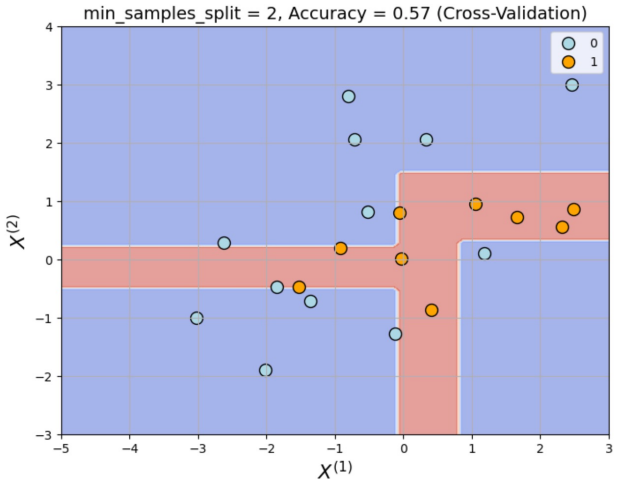




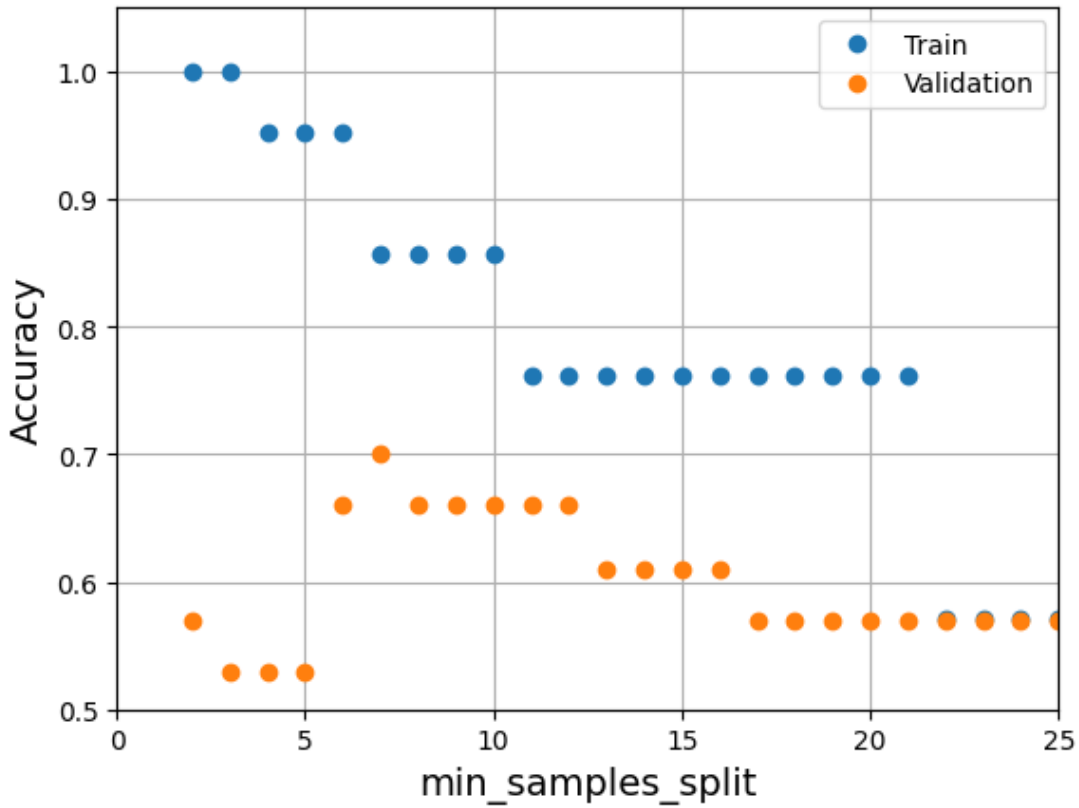


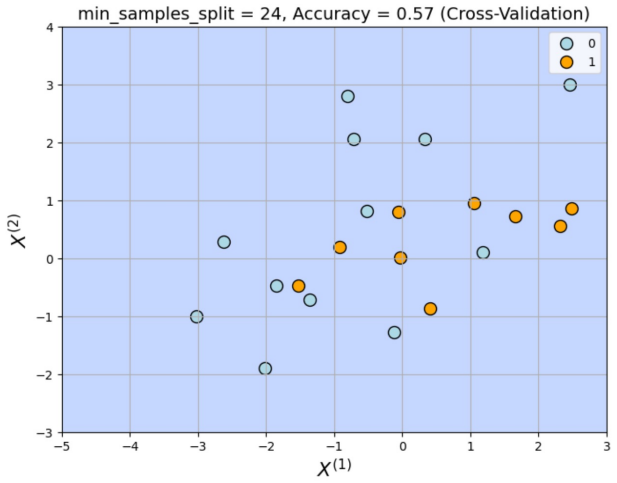


Bias-Variance Trade-Offs



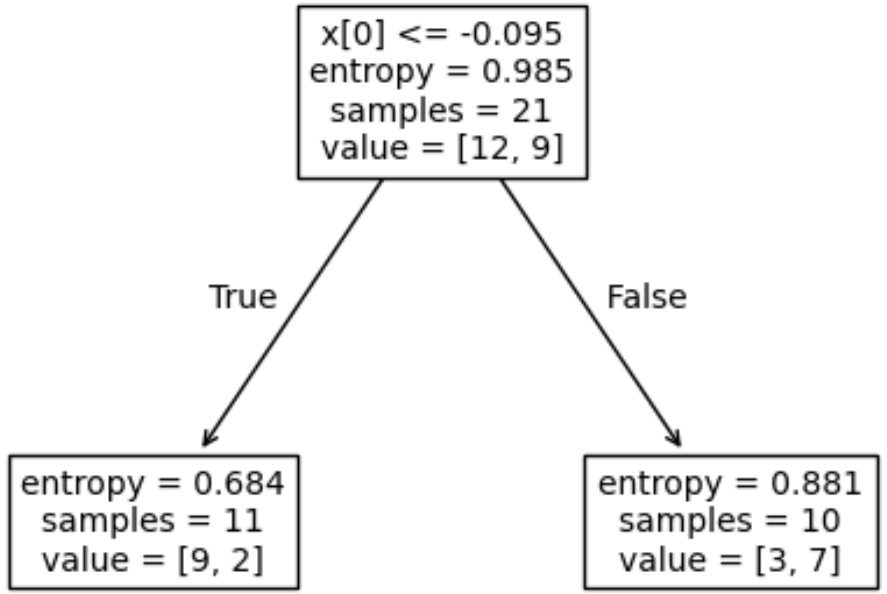
- High Variance, Low Bias
- Complex Decision Boundary
- High Train Accuracy
- Low Validation Accuracy

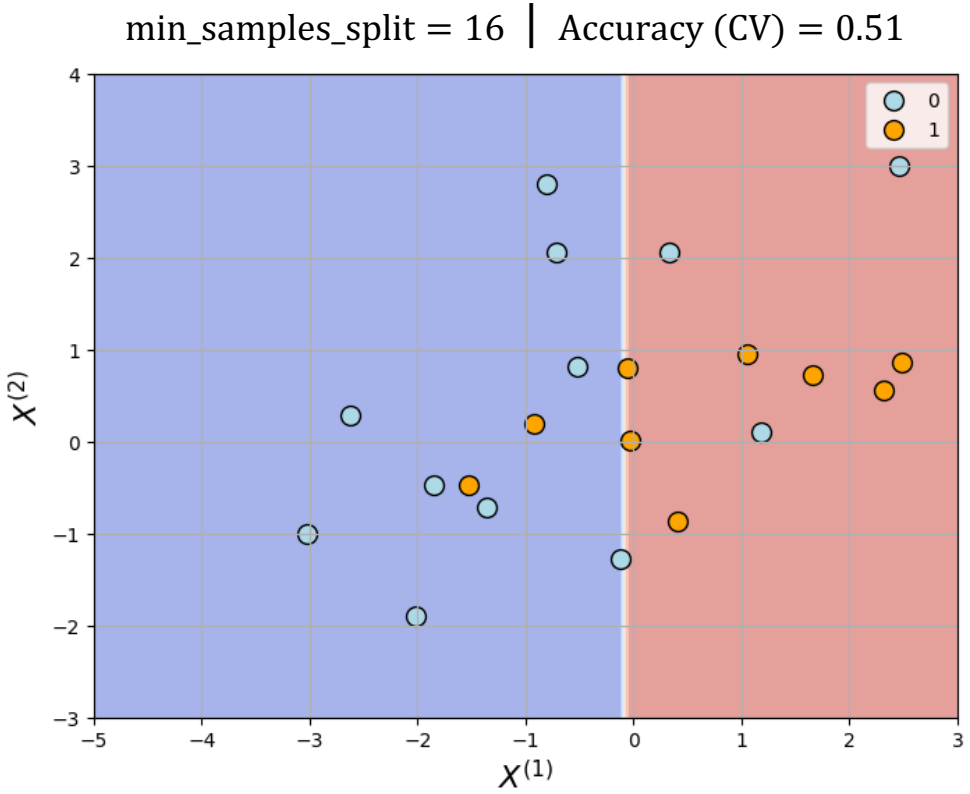




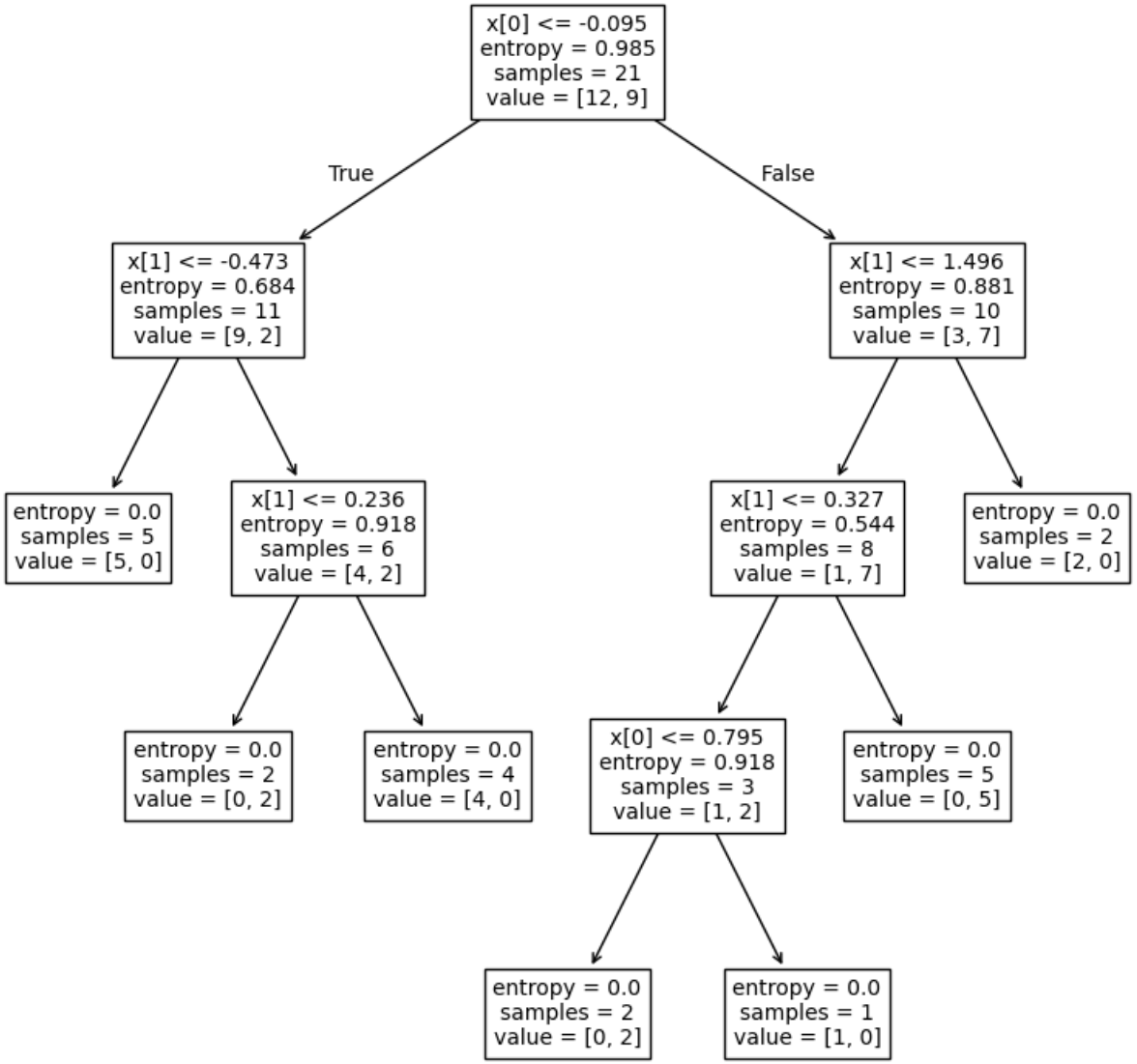
- High Bias, Low Variance
- Simple Decision Boundary
- Low Train Accuracy
- Low Validation Accuracy

Decision Boundary (min_samples_split = 16)

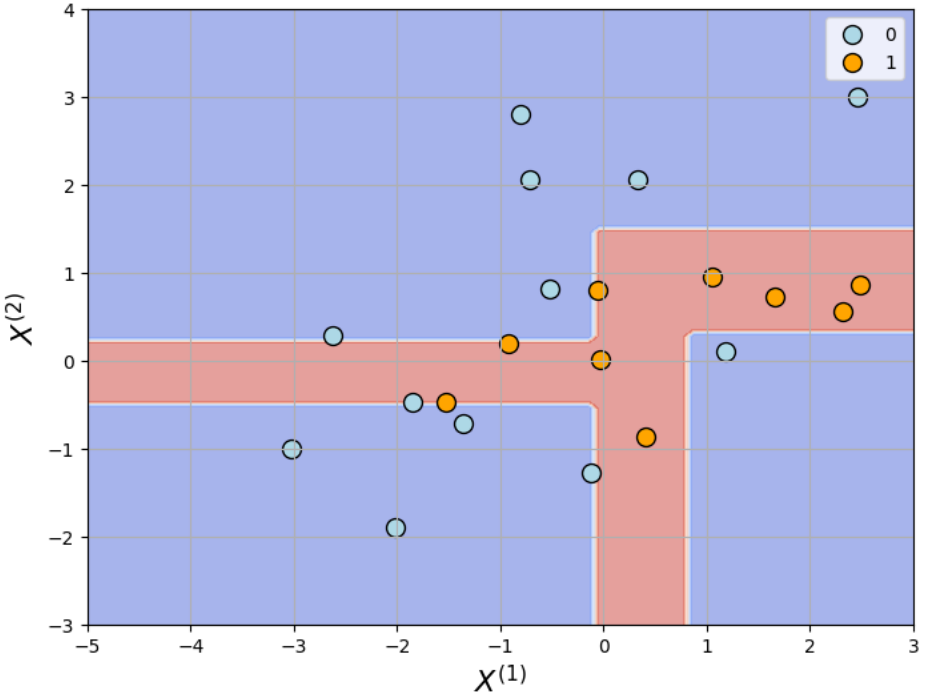




Decision Boundary (min_samples_split = 2)

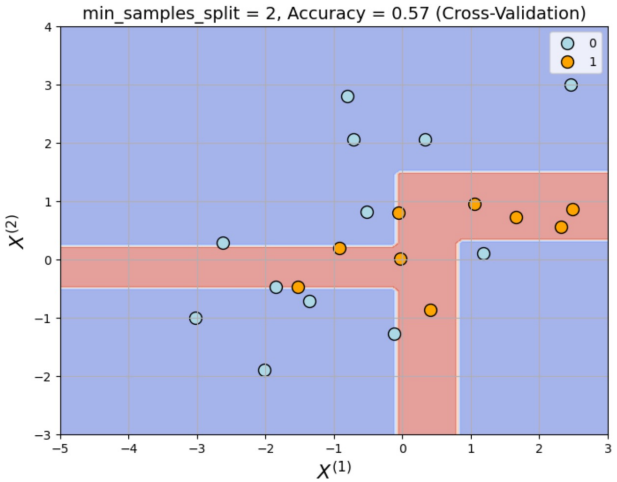


min_samples_split = 2 | Accuracy (CV) = 0.57

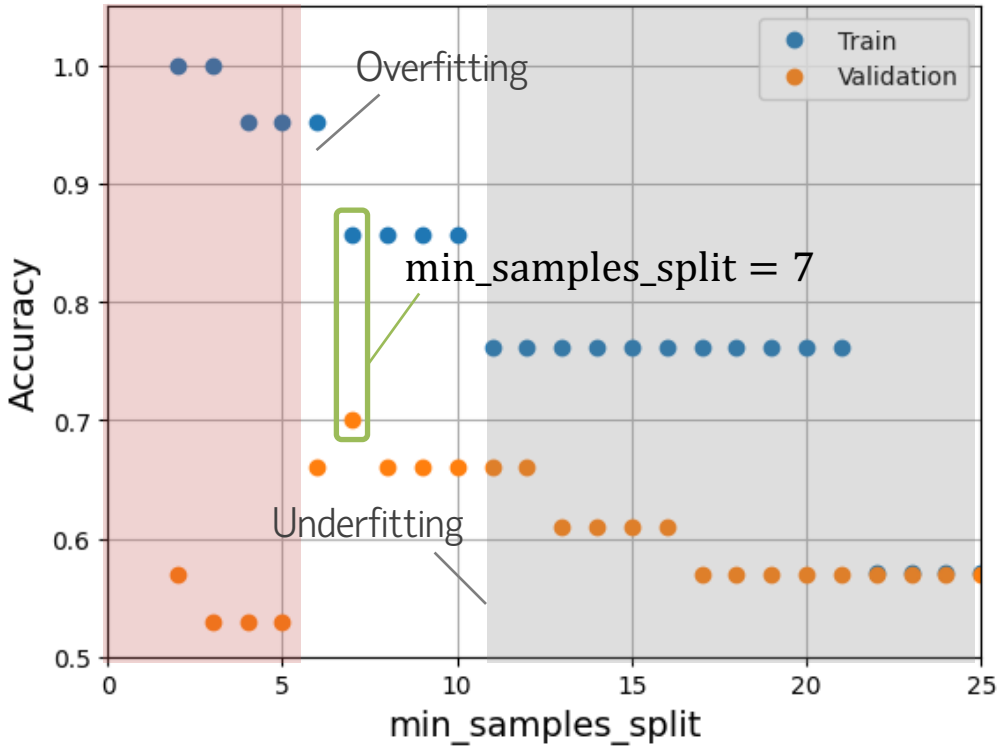


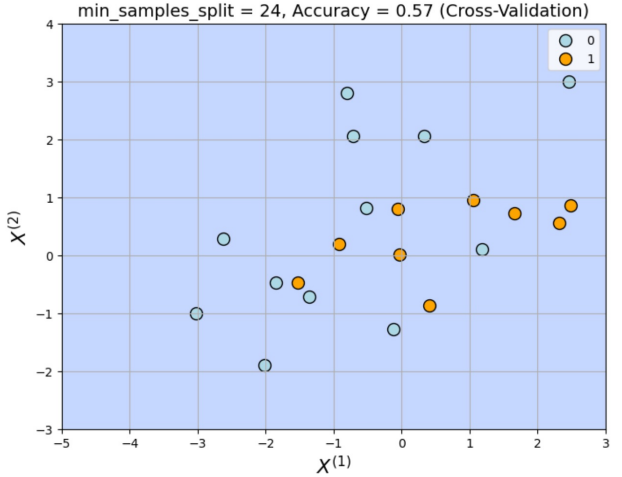
Overfitting vs Underfitting

- 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

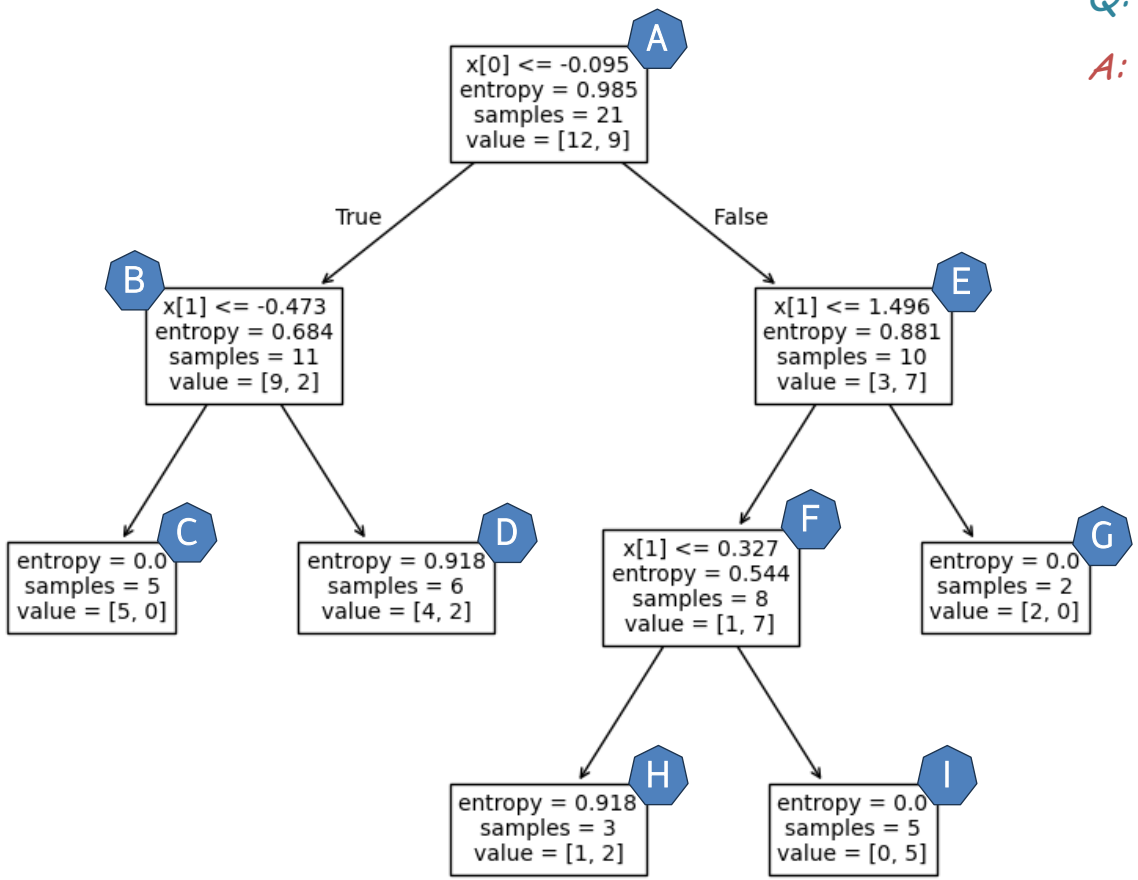




- High Bias, Low Variance
- Simple Decision Boundary
- Low Train Accuracy
- Low Validation Accuracy

- 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.

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

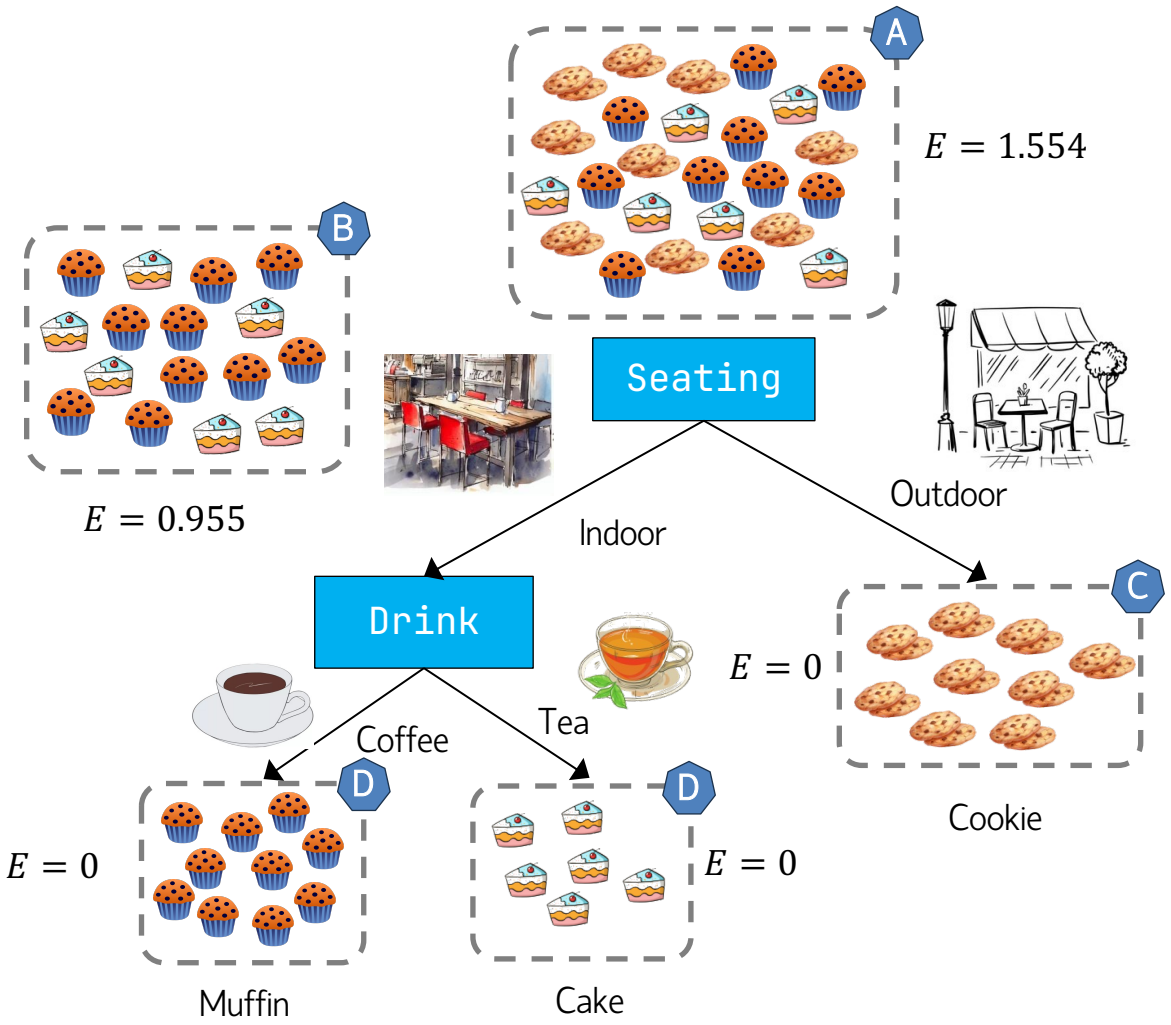
Feature	Sum of Contribution $\sum_t w_t IG_t$	Normalised Importance
$x[0]$	0.2078	$0.2078/0.5916 \approx 0.352$
$x[1]$	$0.0954 + 0.2123 + 0.0761 = 0.3838$	$0.3838/0.5916 \approx 0.648$

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

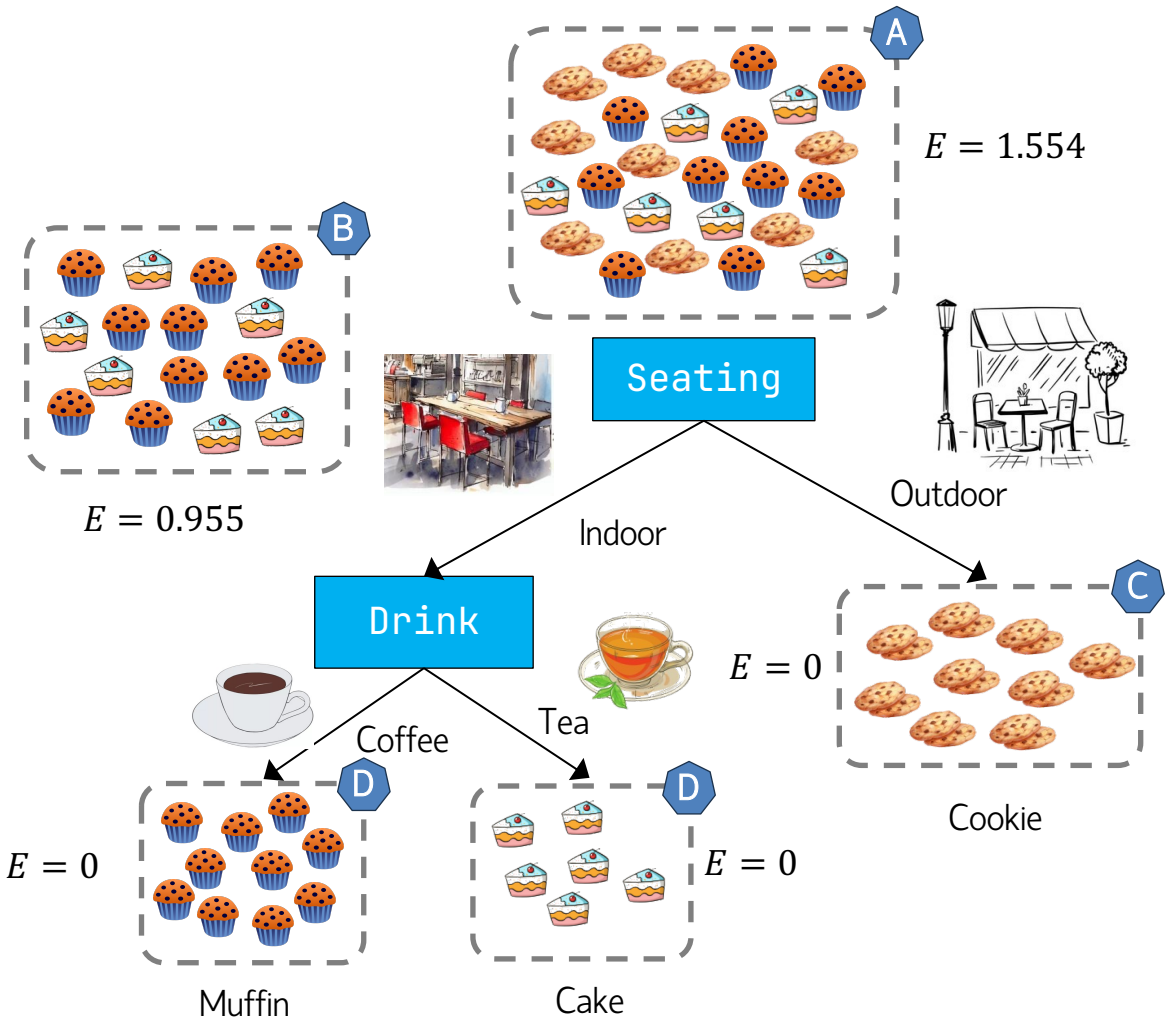
Feature	Sum of Contribution $\sum_t w_t IG_t$	Normalised Importance
Seating		
Drink		

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.