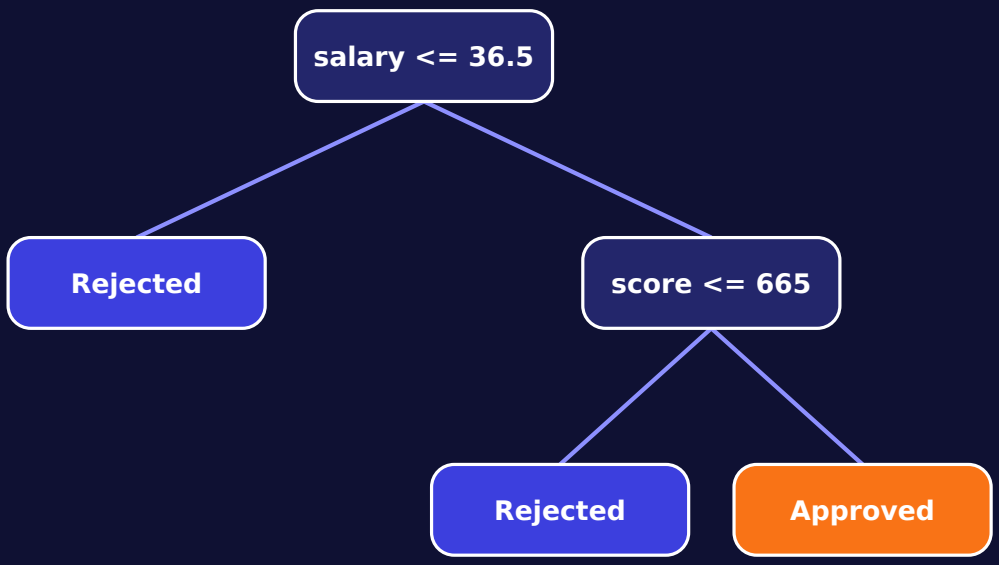
Decision Tree

Notes, Python code and interview questions



Code and Debug

codeanddebug.in/blog | YouTube: @codeanddebug

1. The idea

A decision tree predicts by asking a series of yes or no questions about the features, such as "is the salary above ₹36,500?" and then "is the credit score above 665?", until it reaches a leaf that holds the answer. It is a supervised learning model and works for classification (a class in each leaf) and regression (an average value in each leaf).

It learns the questions from labelled data. At every node it tries every feature and every cut-off, keeps the split that makes the two groups purest, and repeats on each group until it hits a stopping rule. This is recursive binary splitting, and it is greedy: each split is the best for that node only.

2. Key terms

Term	Meaning
Root node	The first question. It sees all the training data.
Decision (internal) node	Any node that asks a question and splits the data.
Leaf (terminal) node	No more questions. Holds the prediction.
Branch	The path from a node to a child, usually yes or no.
Depth	Number of questions from the root to the deepest leaf.
Pruning	Removing branches to make the tree simpler and stop overfitting.

3. Gini, entropy and information gain

Gini impurity (0 = pure, 0.5 = 50/50 for two classes):

$$Gini = 1 - \Sigma\ p_i^2$$

Entropy (0 = pure, 1 = 50/50 for two classes):

$$Entropy = -\ \Sigma\ p_i \cdot \log2(p_i)$$

Information gain of a split:

$$IG = Entropy(parent) - \Sigma\ (n_{child} / n_{parent}) \cdot Entropy(child)$$

Example: a node with 4 pass and 1 fail has Gini = $1 - (0.8^2 + 0.2^2) = 0.32$ and entropy = 0.722. The tree picks the split with the lowest weighted Gini or the highest information gain. Gini and entropy choose the same split in most cases; Gini is the scikit-learn default and is slightly faster.

4. Worked example: picking the root

Student	Studied	Revision	Slept 7+	Result
1	Yes	Yes	Yes	Pass
2	Yes	No	Yes	Pass
3	Yes	Yes	No	Pass
4	Yes	No	No	Fail
5	Yes	Yes	Yes	Pass
6	No	No	Yes	Pass
7	No	No	Yes	Fail
8	No	Yes	No	Fail
9	No	No	No	Fail
10	No	No	Yes	Fail

- Root: 5 pass, 5 fail. Gini 0.5, entropy 1.
- Studied: Yes 4/1 (Gini 0.32), No 1/4 (0.32). Weighted Gini **0.32**, information gain 0.278.
- Revision: Yes 3/1 (0.375), No 2/4 (0.444). Weighted Gini 0.417, information gain 0.125.
- Slept: Yes 4/2 (0.444), No 1/3 (0.375). Weighted Gini 0.417, information gain 0.125.
- Lowest weighted Gini and highest gain: **Studied** becomes the root.

5. DecisionTreeClassifier cheat sheet

Parameter	What it does	Typical values
`criterion`	How splits are scored	`"gini"`, `"entropy"`, `"log_loss"`
`max_depth`	Maximum depth	3 to 10, or None
`min_samples_split`	Points needed to split a node	2 to 20
`min_samples_leaf`	Points every leaf must keep	1 to 20
`max_features`	Features tried at each split	None (all)
`ccp_alpha`	Cost complexity pruning strength	Tune with the pruning path
`class_weight`	Weight for rare classes	None or "balanced"

6. Overfitting and pruning

- An unlimited tree splits until every leaf is pure and memorises noise: training accuracy near 100%, test accuracy lower.
- **Pre-pruning** stops growth early: max_depth, min_samples_leaf, min_samples_split, max_leaf_nodes.
- **Post-pruning** grows the full tree, then removes weak branches. Cost complexity pruning scores a subtree as error + ccp_alpha × leaves.
- Choose the settings with cross-validation (GridSearchCV), then check the test set once.

7. Regression trees

Splits reduce the squared error of the target, and each leaf predicts the mean of its training targets, so predictions form a staircase. A regression tree cannot predict outside the range of the

targets it was trained on.

8. ID3, C4.5 and CART

Algorithm	Criterion	Splits	Notes
ID3	Information gain	Multi-way	Categorical features only, no pruning
C4.5	Gain ratio	Multi-way	Numbers, missing values, post-pruning
CART	Gini / squared error	Binary	Classification and regression; used by scikit-learn

9. Pros and cons

Advantages	Disadvantages
Easy to explain and draw	Overfits without pruning
No feature scaling needed	Unstable: small data changes, different tree
Captures non-linear patterns and interactions	Axis-parallel staircase boundaries
Very fast predictions	Greedy, not guaranteed optimal
Classification and regression	Weaker than random forests and boosting on accuracy

10. Code examples (all tested)

Your first tree: loan approval

PYTHON

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier, export_text

# Monthly salary (in thousand rupees) and credit score for 14 loan applicants
X = np.array([
    [25, 620], [30, 710], [18, 580], [45, 640], [22, 700], [35, 600], [28, 560],  # rejected
    [60, 720], [55, 760], [40, 780], [75, 690], [50, 740], [90, 810], [38, 750],  # approved
])
y = np.array([0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])  # 0 = rejected, 1 = approved

model = DecisionTreeClassifier(max_depth=2, random_state=42)
model.fit(X, y)

print(export_text(model, feature_names=["salary_k", "credit_score"]))

new_applicants = np.array([[42, 745], [80, 610]])
print("Predictions:", model.predict(new_applicants))
print("Depth:", model.get_depth(), "| Leaves:", model.get_n_leaves())
```

OUTPUT

```
|--- salary_k <= 36.50
|   |--- class: 0
|--- salary_k >  36.50
|   |--- credit_score <= 665.00
|   |   |--- class: 0
|   |--- credit_score >  665.00
|   |   |--- class: 1

Predictions: [1 0]
Depth: 2 | Leaves: 3
```

Gini and information gain by hand

PYTHON

```
import numpy as np

# 10 students: did they study, attend the revision class, sleep 7+ hours? 1 = pass, 0 = fail
studied  = np.array(["Y", "Y", "Y", "Y", "Y", "N", "N", "N", "N", "N"])
revision = np.array(["Y", "N", "Y", "N", "Y", "N", "N", "Y", "N", "N"])
slept    = np.array(["Y", "Y", "N", "N", "Y", "Y", "Y", "N", "N", "Y"])
passed   = np.array([1, 1, 1, 0, 1, 1, 0, 0, 0, 0])

def gini(labels):
    p = np.bincount(labels, minlength=2) / len(labels)
    return 1 - np.sum(p ** 2)

def entropy(labels):
    p = np.bincount(labels, minlength=2) / len(labels)
    p = p[p > 0]
    return -np.sum(p * np.log2(p))

print(f"Root: Gini = {gini(passed):.3f}, Entropy = {entropy(passed):.3f}\n")

for name, feature in [("studied", studied), ("revision", revision), ("slept", slept)]:
    left, right = passed[feature == "Y"], passed[feature == "N"]
    wl, wr = len(left) / len(passed), len(right) / len(passed)
    w_gini = wl * gini(left) + wr * gini(right)
    gain = entropy(passed) - (wl * entropy(left) + wr * entropy(right))
    print(f"{name:9s} Yes: {left.sum()} pass / {len(left) - left.sum()} fail | "
          f"No: {right.sum()} pass / {len(right) - right.sum()} fail | "
          f"weighted Gini = {w_gini:.3f} | information gain = {gain:.3f}")
```

OUTPUT

```
Root: Gini = 0.500, Entropy = 1.000

studied   Yes: 4 pass / 1 fail | No: 1 pass / 4 fail | weighted Gini = 0.320 | information gain = 0.278
revision  Yes: 3 pass / 1 fail | No: 2 pass / 4 fail | weighted Gini = 0.417 | information gain = 0.125
slept     Yes: 4 pass / 2 fail | No: 1 pass / 3 fail | weighted Gini = 0.417 | information gain = 0.125
```

max_depth and overfitting

PYTHON

```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

X, y = make_moons(n_samples=600, noise=0.35, random_state=7)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=7)

print("max_depth | leaves | train acc | test acc")
for depth in [1, 2, 3, 4, 6, 8, 12, None]:
    tree = DecisionTreeClassifier(max_depth=depth, random_state=7).fit(X_train, y_train)
    train_acc = tree.score(X_train, y_train)
    test_acc = tree.score(X_test, y_test)
    print(f"{str(depth):>9} | {tree.get_n_leaves():>6} | {train_acc:>9.3f} | {test_acc:.3f}")
```

OUTPUT

```
max_depth | leaves | train acc | test acc
        1 |      2 |     0.826 | 0.772
        2 |      4 |     0.914 | 0.844
        3 |      6 |     0.914 | 0.844
        4 |     10 |     0.919 | 0.844
        6 |     19 |     0.933 | 0.861
        8 |     31 |     0.964 | 0.872
       12 |     49 |     1.000 | 0.850
     None |     49 |     1.000 | 0.850
```

Pre-pruning and post-pruning (breast cancer)

PYTHON

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y, random_state=0)

# 1. A fully grown tree
full = DecisionTreeClassifier(random_state=0).fit(X_train, y_train)
print(f"Full tree:   depth {full.get_depth()}, {full.get_n_leaves()} leaves, "
      f"test acc {full.score(X_test, y_test):.3f}")

# 2. Pre-pruning: limit the tree while it grows
params = {"max_depth": [2, 3, 4, 5, 6, None], "min_samples_leaf": [1, 3, 5, 10]}
grid = GridSearchCV(DecisionTreeClassifier(random_state=0), params, cv=5)
grid.fit(X_train, y_train)
pre = grid.best_estimator_
print(f"Pre-pruned:  {grid.best_params_}, {pre.get_n_leaves()} leaves, "
      f"test acc {pre.score(X_test, y_test):.3f}")

# 3. Post-pruning: grow fully, then cut back weak branches (cost complexity pruning)
alphas = full.cost_complexity_pruning_path(X_train, y_train).ccp_alphas
grid2 = GridSearchCV(DecisionTreeClassifier(random_state=0), {"ccp_alpha": alphas[:-1]}, cv=5)
grid2.fit(X_train, y_train)
post = grid2.best_estimator_
best_alpha = grid2.best_params_["ccp_alpha"]
print(f"Post-pruned: ccp_alpha={best_alpha:.4f}, {post.get_n_leaves()} leaves, "
      f"test acc {post.score(X_test, y_test):.3f}")
```

OUTPUT

```
Full tree:   depth 5, 18 leaves, test acc 0.902
Pre-pruned:  {'max_depth': 4, 'min_samples_leaf': 5}, 11 leaves, test acc 0.923
Post-pruned: ccp_alpha=0.0138, 5 leaves, test acc 0.902
```

Regression tree: rent from flat size

PYTHON

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

# Flat size (sq ft) vs monthly rent (thousand rupees), with some noise
rng = np.random.default_rng(3)
size = rng.uniform(400, 2000, 200)
rent = 6 + 0.012 * size + 4 * np.sin(size / 250) + rng.normal(0, 2, 200)
X = size.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, rent, test_size=0.3, random_state=3)

for depth in [1, 3, 5, None]:
    reg = DecisionTreeRegressor(max_depth=depth, random_state=3).fit(X_train, y_train)
    print(f"max_depth={str(depth):4s} leaves={reg.get_n_leaves():3d}  "
          f"train MAE={mean_absolute_error(y_train, reg.predict(X_train)):.2f}  "
          f"test MAE={mean_absolute_error(y_test, reg.predict(X_test)):.2f}")

reg = DecisionTreeRegressor(max_depth=3, random_state=3).fit(X_train, y_train)
print("Rent for 900 and 1500 sq ft:", reg.predict([[900], [1500]]).round(1))
```

OUTPUT

```
max_depth=1    leaves=  2  train MAE=2.55  test MAE=2.65
max_depth=3    leaves=  8  train MAE=1.52  test MAE=1.65
max_depth=5    leaves= 31  train MAE=1.16  test MAE=1.90
max_depth=None leaves=140  train MAE=0.00  test MAE=2.30
Rent for 900 and 1500 sq ft: [15.  24.1]
```

Decision tree from scratch with NumPy

PYTHON

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

class SimpleTree:
    def __init__(self, max_depth=3, min_samples=2):
        self.max_depth, self.min_samples = max_depth, min_samples

    def gini(self, y):
        p = np.bincount(y) / len(y)
        return 1 - np.sum(p ** 2)

    def best_split(self, X, y):
        best = (None, None, self.gini(y))          # feature, threshold, impurity
        for f in range(X.shape[1]):
            values = np.unique(X[:, f])
            for t in (values[:-1] + values[1:]) / 2:  # midpoints between sorted values
                left = X[:, f] <= t
                w = (left.sum() * self.gini(y[left]) + (~left).sum() * self.gini(y[~left])) / len(y)
                if w < best[2]:
                    best = (f, t, w)
        return best

    def build(self, X, y, depth):
        f, t, _ = self.best_split(X, y)
        if depth == self.max_depth or len(y) < self.min_samples or f is None:
            return np.bincount(y).argmax()           # leaf: majority class
        left = X[:, f] <= t
        return (f, t, self.build(X[left], y[left], depth + 1), self.build(X[~left], y[~left], depth + 1))

    def fit(self, X, y):
        self.root = self.build(X, y, 0)
        return self

    def predict_one(self, x, node):
        while isinstance(node, tuple):
            f, t, l, r = node
            node = l if x[f] <= t else r
        return node

    def predict(self, X):
        return np.array([self.predict_one(x, self.root) for x in X])

X, y = make_moons(n_samples=400, noise=0.3, random_state=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)

mine = SimpleTree(max_depth=4).fit(X_train, y_train)
sk = DecisionTreeClassifier(max_depth=4, random_state=1).fit(X_train, y_train)
print("Root split (feature, threshold):", mine.root[0], round(mine.root[1], 3))
print("Our tree test accuracy:     ", round(np.mean(mine.predict(X_test) == y_test), 3))
print("scikit-learn test accuracy: ", round(sk.score(X_test, y_test), 3))
print("Same predictions on test set:", np.mean(mine.predict(X_test) == sk.predict(X_test)))
```

OUTPUT

```
Root split (feature, threshold): 1 0.124
Our tree test accuracy:      0.83
scikit-learn test accuracy:  0.83
Same predictions on test set: 1.0
```

Wine classification: tuning and feature importance

PYTHON

```
import numpy as np
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.metrics import classification_report

data = load_wine()
X_train, X_test, y_train, y_test = train_test_split(
    data.data, data.target, test_size=0.3, stratify=data.target, random_state=42)

grid = GridSearchCV(DecisionTreeClassifier(random_state=42),
                    {"criterion": ["gini", "entropy"], "max_depth": [2, 3, 4, 5, None],
                     "min_samples_leaf": [1, 2, 5]}, cv=5)
grid.fit(X_train, y_train)
tree = grid.best_estimator_
print("Best settings:", grid.best_params_)
print("Test accuracy:", round(tree.score(X_test, y_test), 3), "\n")
print(classification_report(y_test, tree.predict(X_test), target_names=data.target_names))

top = np.argsort(tree.feature_importances_)[::-1][:4]
print("Top features:")
for i in top:
    print(f"  {data.feature_names[i]:30s} {tree.feature_importances_[i]:.3f}")
print()
print(export_text(tree, feature_names=list(data.feature_names), max_depth=2))
```

OUTPUT

```
Best settings: {'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 2}
Test accuracy: 0.907

              precision    recall  f1-score   support

     class_0       0.94      0.89      0.91        18
     class_1       0.86      0.90      0.88        21
     class_2       0.93      0.93      0.93        15

    accuracy                           0.91        54
   macro avg       0.91      0.91      0.91        54
weighted avg       0.91      0.91      0.91        54

Top features:
  flavanoids                     0.413
  alcohol                        0.335
  hue                            0.150
  proline                        0.101

|--- flavanoids <= 1.58
|   |--- hue <= 0.90
|   |   |--- class: 2
|   |--- hue >  0.90
|   |   |--- alcohol <= 12.62
|   |   |   |--- class: 1
|   |   |--- alcohol >  12.62
|   |   |   |--- class: 1
|--- flavanoids >  1.58
|   |--- alcohol <= 13.02
|   |   |--- class: 1
|   |--- alcohol >  13.02
|   |   |--- proline <= 597.50
|   |   |   |--- class: 1
|   |   |--- proline >  597.50
|   |   |   |--- class: 0
```

11. Decision tree interview questions

1. What is a decision tree?

A supervised learning model that predicts by asking a sequence of questions about the features, each splitting the data into smaller groups, until it reaches a leaf that holds the prediction. It can be used for classification and regression.

2. How does a decision tree decide where to split?

At each node it tries every feature and every candidate threshold, measures the impurity of the resulting child nodes, and keeps the split that reduces impurity the most. For classification that is Gini or entropy; for regression it is usually squared error.

3. What is Gini impurity?

One minus the sum of squared class proportions in a node. It is the chance of labelling a random point wrongly if you guess using the node's class distribution. It is 0 for a pure node and 0.5 for a 50/50 node with two classes.

4. What is entropy?

A measure of uncertainty: minus the sum of p times log base 2 of p over the classes. It is 0 for a pure node and 1 for a 50/50 node with two classes.

5. What is information gain?

The drop in entropy from a parent node to its children, where each child's entropy is weighted by its share of the points. The split with the highest information gain is chosen.

6. Gini or entropy: which is better?

Neither is clearly better. They produce almost the same tree in most cases. Gini is slightly faster because it avoids logarithms, which is why it is the default in scikit-learn.

7. Why do decision trees overfit, and how do you stop it?

Without limits a tree keeps splitting until each leaf is pure, which means it learns the noise in the training data. You stop it with pre-pruning (max_depth, min_samples_leaf, min_samples_split, max_leaf_nodes) or post-pruning (cost complexity pruning with ccp_alpha), tuned with cross-validation.

8. What is the difference between pre-pruning and post-pruning?

Pre-pruning stops the tree from growing past certain limits. Post-pruning grows the full tree first and then removes branches that do not improve validation performance enough to justify their size.

9. What is cost complexity pruning?

A post-pruning method that scores each subtree by its training error plus alpha times its number of leaves. As alpha increases, more branches are removed. The best alpha is chosen by cross-validation.

10. Do decision trees need feature scaling?

No. Each split compares one feature with a threshold, so rescaling a feature only moves the threshold and gives the same tree.

11. How does a regression tree make predictions?

It splits the data to reduce the squared error of the target, and each leaf predicts the mean target value of the training points in it. The result is a step-shaped function.

12. What are ID3, C4.5 and CART?

Three tree-building algorithms. ID3 uses information gain and multi-way splits on categorical features. C4.5 uses gain ratio, handles numeric features and missing values, and prunes. CART makes binary splits, uses Gini or squared error, and supports regression; scikit-learn uses CART.

13. What is feature importance in a decision tree?

The total impurity reduction a feature contributes across all its splits, weighted by the number of samples reaching those splits, normalised to sum to 1. It is biased towards high-cardinality features, so permutation importance is a useful check.

14. Why is a decision tree called a greedy algorithm?

Because it chooses the best split at each node based only on that node, without considering how the choice affects later splits. Finding the globally optimal tree is NP-hard, so greedy splitting is used instead.

15. Are decision trees stable?

No. They have high variance: small changes in the data can change the top split and therefore the whole tree. Bagging, as in random forests, reduces this variance by averaging many trees.

16. What is the difference between a decision tree and a random forest?

A decision tree is a single model. A random forest trains many trees on bootstrap samples of the data, with a random subset of features at each split, and averages their predictions. The forest is more accurate and stable but much harder to interpret.

17. How do decision trees handle categorical features?

In theory they can split on categories directly, as ID3 and C4.5 do. In scikit-learn you usually encode them first, with one-hot encoding or ordinal encoding. Recent versions of scikit-learn also add native categorical support in some tree-based models.

18. What is the time complexity of a decision tree?

Training is roughly $O(n \times m \times \log n)$ for n rows and m features when the tree is balanced, because each level sorts and scans the features. Prediction is O(depth), which is about O(log n) for a balanced tree.

Grow your own tree in the interactive playground, try the Gini vs entropy calculator and take the quiz on the blog: codeanddebug.in/blog/decision-tree-in-machine-learning