Decision Trees: Theory and Practice

September 9, 2025

1 Introduction

Decision trees are non-parametric models that recursively partition the feature space to produce piecewise-constant predictions. They are easy to interpret, handle mixed feature types, and require little preprocessing.

2 Theory and Formulas

For classification, a tree chooses splits by maximizing impurity reduction. Let $\mathcal{D}$ be a node's dataset with class proportions p_k. Common impurities include Gini and entropy:

$$\mathrm{Gini}(\mathcal{D}) = 1 - \sum_k p_k^2, \tag{1}$$

$$\mathrm{Entropy}(\mathcal{D}) = -\sum_k p_k \log p_k. \tag{2}$$

For a split into left/right children L, R, the impurity after the split is

$$I_{\text{split}} = \frac{|L|}{|\mathcal{D}|} I(L) + \frac{|R|}{|\mathcal{D}|} I(R), \tag{3}$$

and the best split maximizes $\Delta I = I(\mathcal{D}) - I_{\text{split}}$. Stopping criteria include maximum depth, minimum samples per leaf, and minimal impurity decrease.

3 Applications and Tips

- **Pros:** interpretability, handles non-linear boundaries, little preprocessing.
- **Cons:** high variance, prone to overfitting; consider ensembles.
- **Regularization:** use `max_depth`, `min_samples_leaf`, or cost-complexity pruning.
- **Features:** no scaling required; can mix categorical (encoded) and numerical features.
- **Baselines:** compare against logistic regression, SVM, or random forests.

4 Python Practice

Run the script in this chapter directory to generate figures into `figures/`.

Listing 1: Generate Decision Tree figures

```
python gen_decision_tree_figures.py
```

Listing 2: gen_decision_tree_figures.py

```
"""
Figure generator for the Decision Tree chapter.

Generates illustrative figures and saves them into the chapter's 'figures/'
folder next to this script, regardless of current working directory.

Requirements:
- Python 3.8+
- numpy, matplotlib, scikit-learn

Install (if needed):
  pip install numpy matplotlib scikit-learn

This script avoids newer or experimental APIs for broader compatibility.
"""
from __future__ import annotations

import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

try:
    from sklearn.datasets import make_moons, make_classification
    from sklearn.tree import DecisionTreeClassifier, plot_tree
    from sklearn.ensemble import RandomForestClassifier
except Exception as e:
    raise SystemExit(
        "Missing scikit-learn. Please install with: pip install scikit-learn"
    )


def _ensure_figures_dir(path: str | None = None) -> str:
    """Create figures directory under this chapter regardless of CWD."""
    if path is None:
        base = os.path.dirname(os.path.abspath(__file__))
        path = os.path.join(base, "figures")
    os.makedirs(path, exist_ok=True)
    return path


def _plot_decision_boundary(ax, clf, X, y, title: str):
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(
```

```python
        np.linspace(x_min, x_max, 400), np.linspace(y_min, y_max, 400)
    )
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    cmap_light = ListedColormap(["#FFEEEE", "#EEEEFF"])
    cmap_bold = ListedColormap(["#E74C3C", "#3498DB"])
    ax.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8, levels=np.unique(Z).
        size)
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolors="k", s=20)
    ax.set_title(title)
    ax.set_xlabel("Feature 1")
    ax.set_ylabel("Feature 2")


def fig_dt_decision_boundary_2class(out_dir: str) -> str:
    np.random.seed(0)
    X, y = make_moons(n_samples=400, noise=0.25, random_state=0)
    clf = DecisionTreeClassifier(max_depth=4, random_state=0)
    clf.fit(X, y)

    fig, ax = plt.subplots(figsize=(5.5, 4.5), dpi=150)
    _plot_decision_boundary(ax, clf, X, y, "Decision Tree boundary (max_depth
        =4)")
    out_path = os.path.join(out_dir, "dt_decision_boundary_2class.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_dt_depth_compare(out_dir: str) -> str:
    np.random.seed(1)
    X, y = make_moons(n_samples=500, noise=0.3, random_state=1)
    models = [
        (DecisionTreeClassifier(max_depth=3, random_state=1), "max_depth=3"),
        (DecisionTreeClassifier(random_state=1), "max_depth=None (deep)")
    ]
    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True,
        sharey=True)
    for ax, (m, title) in zip(axes, models):
        m.fit(X, y)
        _plot_decision_boundary(ax, m, X, y, f"Decision Tree: {title}")
    fig.suptitle("Depth and overfitting")
    out_path = os.path.join(out_dir, "dt_depth_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_dt_feature_importances(out_dir: str) -> str:
    X, y = make_classification(
        n_samples=600,
        n_features=8,
        n_informative=3,
```

```
        n_redundant=2,
        n_repeated=0,
        random_state=7,
        shuffle=True,
    )
    clf = DecisionTreeClassifier(max_depth=5, random_state=7)
    clf.fit(X, y)
    importances = clf.feature_importances_

    fig, ax = plt.subplots(figsize=(6.5, 3.8), dpi=160)
    idx = np.arange(importances.size)
    ax.bar(idx, importances, color="#3498DB")
    ax.set_xticks(idx)
    ax.set_xticklabels([f"f{i}" for i in idx])
    ax.set_ylabel("importance")
    ax.set_title("Decision Tree feature importances")
    ax.set_ylim(0, max(0.25, importances.max() + 0.05))
    for i, v in enumerate(importances):
        ax.text(i, v + 0.01, f"{v:.2f}", ha="center", va="bottom", fontsize=8)
    out_path = os.path.join(out_dir, "dt_feature_importances.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_dt_vs_rf_boundary(out_dir: str) -> str:
    np.random.seed(2)
    X, y = make_moons(n_samples=500, noise=0.3, random_state=2)
    dt = DecisionTreeClassifier(max_depth=5, random_state=2).fit(X, y)
    rf = RandomForestClassifier(n_estimators=100, random_state=2).fit(X, y)

    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True, sharey=True)
    _plot_decision_boundary(axes[0], dt, X, y, "Decision Tree")
    _plot_decision_boundary(axes[1], rf, X, y, "Random Forest")
    fig.suptitle("Decision Tree vs Random Forest")
    out_path = os.path.join(out_dir, "dt_vs_rf_boundary.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_dt_tree_plot(out_dir: str) -> str:
    # Small depth to keep the plot readable
    X, y = make_moons(n_samples=200, noise=0.25, random_state=3)
    clf = DecisionTreeClassifier(max_depth=3, random_state=3).fit(X, y)

    fig, ax = plt.subplots(figsize=(10, 6), dpi=150)
    plot_tree(clf, filled=True, feature_names=["x1", "x2"], class_names=["0", "1"], ax=ax)
    ax.set_title("Decision Tree (max_depth=3)")
    out_path = os.path.join(out_dir, "dt_tree_plot.png")
```

```python
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def main():
    out_dir = _ensure_figures_dir(None)
    generators = [
        fig_dt_decision_boundary_2class,
        fig_dt_depth_compare,
        fig_dt_feature_importances,
        fig_dt_vs_rf_boundary,
        fig_dt_tree_plot,
    ]
    print("Generating figures into:", os.path.abspath(out_dir))
    for gen in generators:
        try:
            p = gen(out_dir)
            print("Saved:", p)
        except Exception as e:
            print("Failed generating", gen.__name__, ":", e)


if __name__ == "__main__":
    main()
```

Listing 3: gen_decision_tree_figures.py

```python
"""
Figure generator for the Decision Tree chapter.

Generates illustrative figures and saves them into the chapter's 'figures/'
folder next to this script, regardless of current working directory.

Requirements:
- Python 3.8+
- numpy, matplotlib, scikit-learn

Install (if needed):
  pip install numpy matplotlib scikit-learn

This script avoids newer or experimental APIs for broader compatibility.
"""
from __future__ import annotations

import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

try:
    from sklearn.datasets import make_moons, make_classification
    from sklearn.tree import DecisionTreeClassifier, plot_tree
```

```
    from sklearn.ensemble import RandomForestClassifier
except Exception as e:
    raise SystemExit(
        "Missing scikit-learn. Please install with: pip install scikit-learn"
    )


def _ensure_figures_dir(path: str | None = None) -> str:
    """Create figures directory under this chapter regardless of CWD."""
    if path is None:
        base = os.path.dirname(os.path.abspath(__file__))
        path = os.path.join(base, "figures")
    os.makedirs(path, exist_ok=True)
    return path


def _plot_decision_boundary(ax, clf, X, y, title: str):
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 400), np.linspace(y_min, y_max, 400)
    )
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    cmap_light = ListedColormap(["#FFEEEE", "#EEEEFF"])
    cmap_bold = ListedColormap(["#E74C3C", "#3498DB"])
    ax.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8, levels=np.unique(Z).size)
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolors="k", s=20)
    ax.set_title(title)
    ax.set_xlabel("Feature 1")
    ax.set_ylabel("Feature 2")


def fig_dt_decision_boundary_2class(out_dir: str) -> str:
    np.random.seed(0)
    X, y = make_moons(n_samples=400, noise=0.25, random_state=0)
    clf = DecisionTreeClassifier(max_depth=4, random_state=0)
    clf.fit(X, y)

    fig, ax = plt.subplots(figsize=(5.5, 4.5), dpi=150)
    _plot_decision_boundary(ax, clf, X, y, "Decision Tree boundary (max_depth=4)")
    out_path = os.path.join(out_dir, "dt_decision_boundary_2class.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_dt_depth_compare(out_dir: str) -> str:
    np.random.seed(1)
    X, y = make_moons(n_samples=500, noise=0.3, random_state=1)
    models = [
        (DecisionTreeClassifier(max_depth=3, random_state=1), "max_depth=3"),
```

```
        (DecisionTreeClassifier(random_state=1), "max_depth=None (deep)")
    ]
    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True,
        sharey=True)
    for ax, (m, title) in zip(axes, models):
        m.fit(X, y)
        _plot_decision_boundary(ax, m, X, y, f"Decision Tree: {title}")
    fig.suptitle("Depth and overfitting")
    out_path = os.path.join(out_dir, "dt_depth_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_dt_feature_importances(out_dir: str) -> str:
    X, y = make_classification(
        n_samples=600,
        n_features=8,
        n_informative=3,
        n_redundant=2,
        n_repeated=0,
        random_state=7,
        shuffle=True,
    )
    clf = DecisionTreeClassifier(max_depth=5, random_state=7)
    clf.fit(X, y)
    importances = clf.feature_importances_

    fig, ax = plt.subplots(figsize=(6.5, 3.8), dpi=160)
    idx = np.arange(importances.size)
    ax.bar(idx, importances, color="#3498DB")
    ax.set_xticks(idx)
    ax.set_xticklabels([f"f{i}" for i in idx])
    ax.set_ylabel("importance")
    ax.set_title("Decision Tree feature importances")
    ax.set_ylim(0, max(0.25, importances.max() + 0.05))
    for i, v in enumerate(importances):
        ax.text(i, v + 0.01, f"{v:.2f}", ha="center", va="bottom", fontsize=8)
    out_path = os.path.join(out_dir, "dt_feature_importances.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_dt_vs_rf_boundary(out_dir: str) -> str:
    np.random.seed(2)
    X, y = make_moons(n_samples=500, noise=0.3, random_state=2)
    dt = DecisionTreeClassifier(max_depth=5, random_state=2).fit(X, y)
    rf = RandomForestClassifier(n_estimators=100, random_state=2).fit(X, y)

    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True,
        sharey=True)
```

```python
    _plot_decision_boundary(axes[0], dt, X, y, "Decision Tree")
    _plot_decision_boundary(axes[1], rf, X, y, "Random Forest")
    fig.suptitle("Decision Tree vs Random Forest")
    out_path = os.path.join(out_dir, "dt_vs_rf_boundary.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_dt_tree_plot(out_dir: str) -> str:
    # Small depth to keep the plot readable
    X, y = make_moons(n_samples=200, noise=0.25, random_state=3)
    clf = DecisionTreeClassifier(max_depth=3, random_state=3).fit(X, y)

    fig, ax = plt.subplots(figsize=(10, 6), dpi=150)
    plot_tree(clf, filled=True, feature_names=["x1", "x2"], class_names=["0",
        "1"], ax=ax)
    ax.set_title("Decision Tree (max_depth=3)")
    out_path = os.path.join(out_dir, "dt_tree_plot.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def main():
    out_dir = _ensure_figures_dir(None)
    generators = [
        fig_dt_decision_boundary_2class,
        fig_dt_depth_compare,
        fig_dt_feature_importances,
        fig_dt_vs_rf_boundary,
        fig_dt_tree_plot,
    ]
    print("Generating figures into:", os.path.abspath(out_dir))
    for gen in generators:
        try:
            p = gen(out_dir)
            print("Saved:", p)
        except Exception as e:
            print("Failed generating", gen.__name__, ":", e)


if __name__ == "__main__":
    main()
```

5 Result

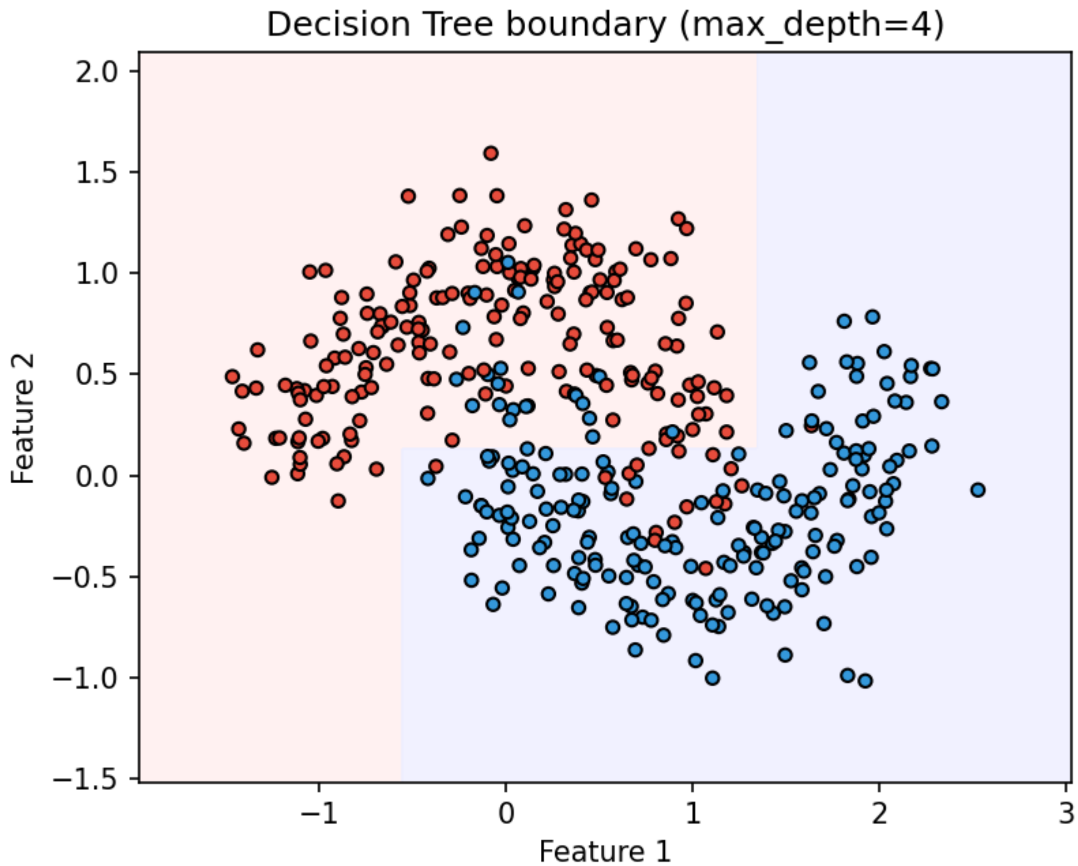

Figure 1: Decision tree decision boundary on a 2-class dataset.

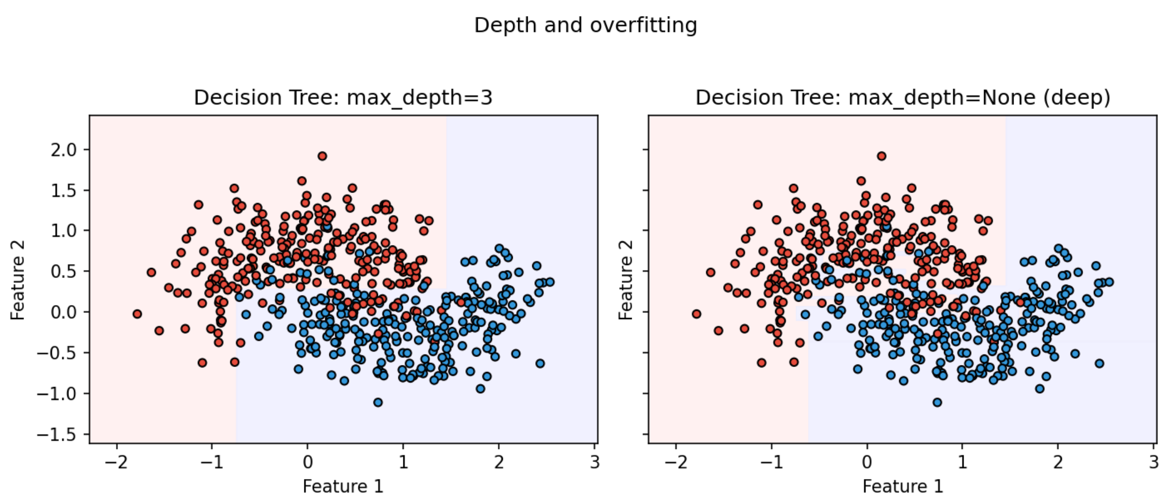

Figure 2: Effect of depth: shallow vs deep tree (overfitting).

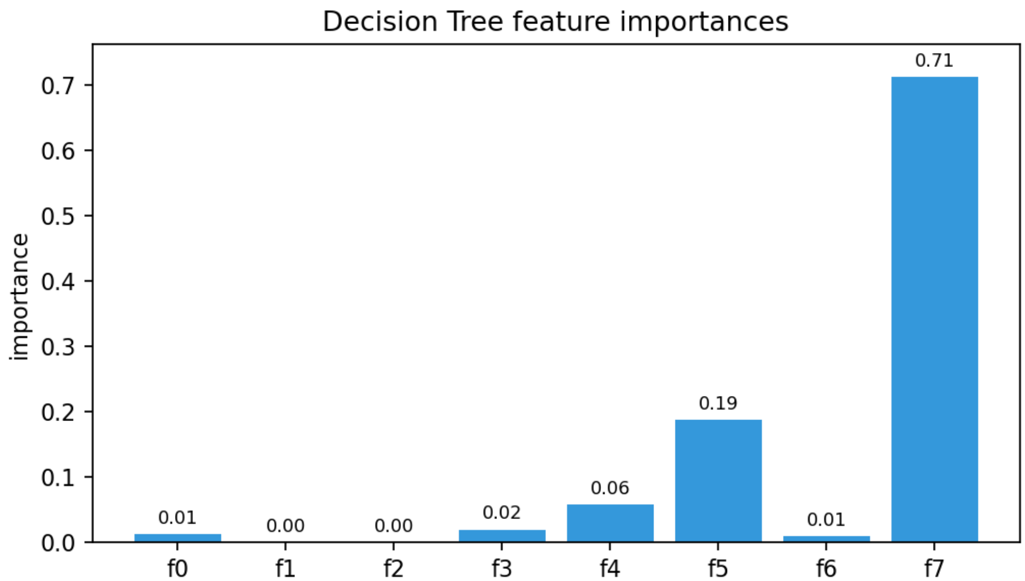

Figure 3: Feature importances from a decision tree.

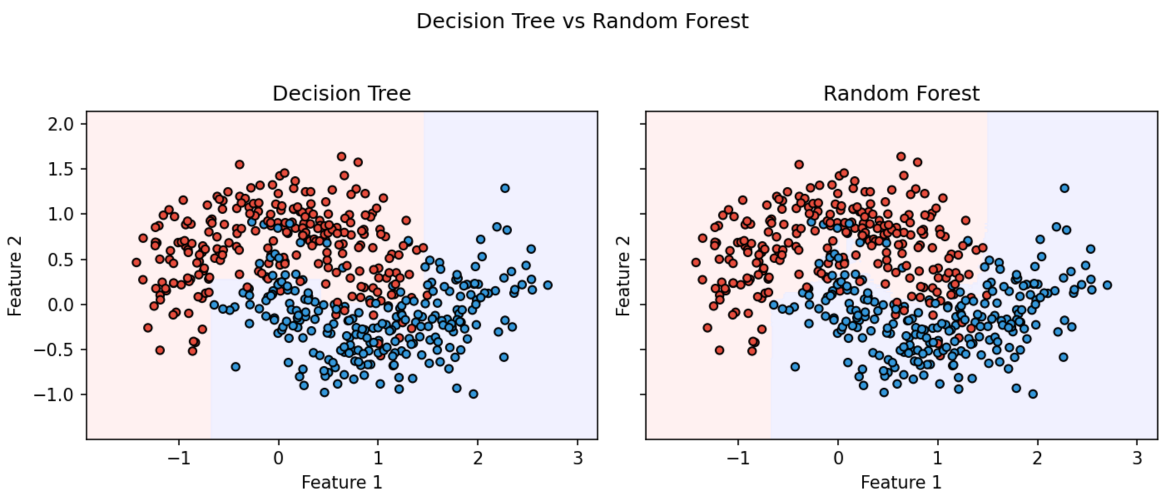

Figure 4: Decision boundary: single tree vs random forest.

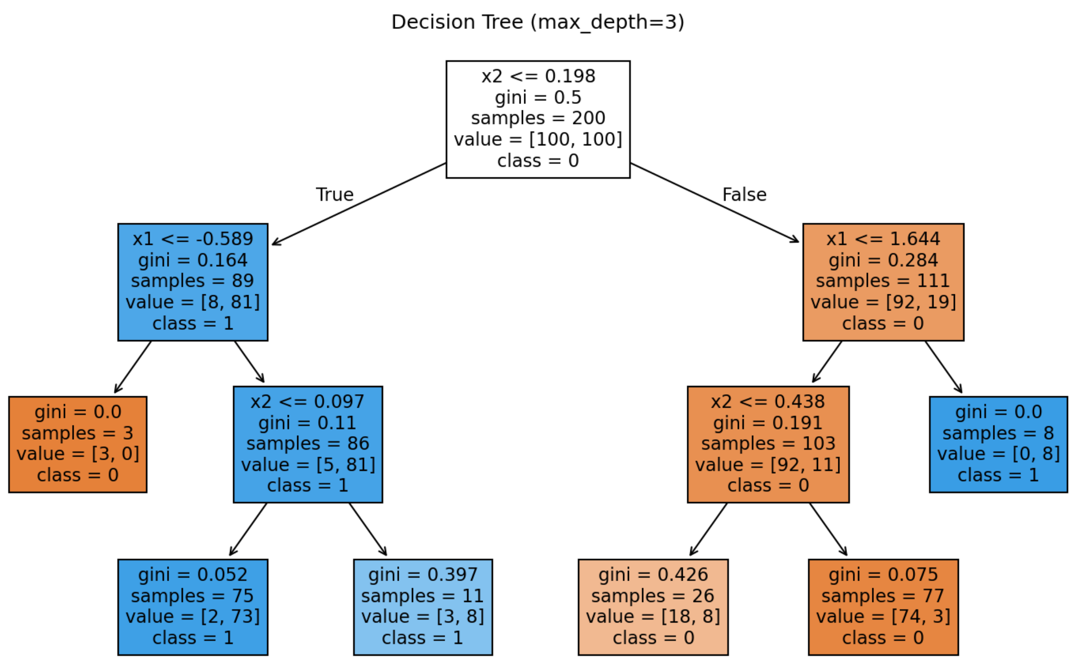

Figure 5: Tree structure visualization (max_depth=3).

6 Summary

Decision trees provide interpretable, flexible baselines. With appropriate regularization or by using ensembles (random forests, gradient boosting), they become powerful general-purpose learners.