决策树：理论与实践

2025 年 9 月 9 日

目录

1 引言

决策树（Decision Tree）通过递归划分特征空间，形成分段常数的预测模型。其优点是可解释性强、对数据预处理要求低，并能处理非线性边界。

2 原理与公式

以分类树为例，在每个节点选择能够最大化“纯度提升”的划分。设节点数据集为 $\mathcal{D}$，类别占比为 p_k。常见纯度指标有基尼与熵：

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

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

若划分为左右子节点 L, R，则划分后的纯度为

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

最优划分使得 $\Delta I = I(\mathcal{D}) - I_{\text{split}}$ 最大。停止条件常包括：最大深度、叶子最小样本数、最小纯度提升等。

3 应用与技巧

- **优点**：可解释、能处理非线性、对特征尺度不敏感、可处理类别与数值特征（需编码）。
- **缺点**：容易过拟合、方差较大；可用集成方法缓解。
- **正则化**：调整 `max_depth`、`min_samples_leaf`，或使用复杂度剪枝。
- **基线对比**：与逻辑回归、SVM、随机森林等模型对比评估。

4 Python 实战

在本章节目录运行下述命令，图片将保存到本目录的 `figures/`：

Listing 1: 生成决策树配图

```
python gen_decision_tree_figures.py
```

Listing 2: gen_decision_tree_figures.py 源码

```
"""
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
```

```python
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")


```

```python
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)
```

```python
    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 结果

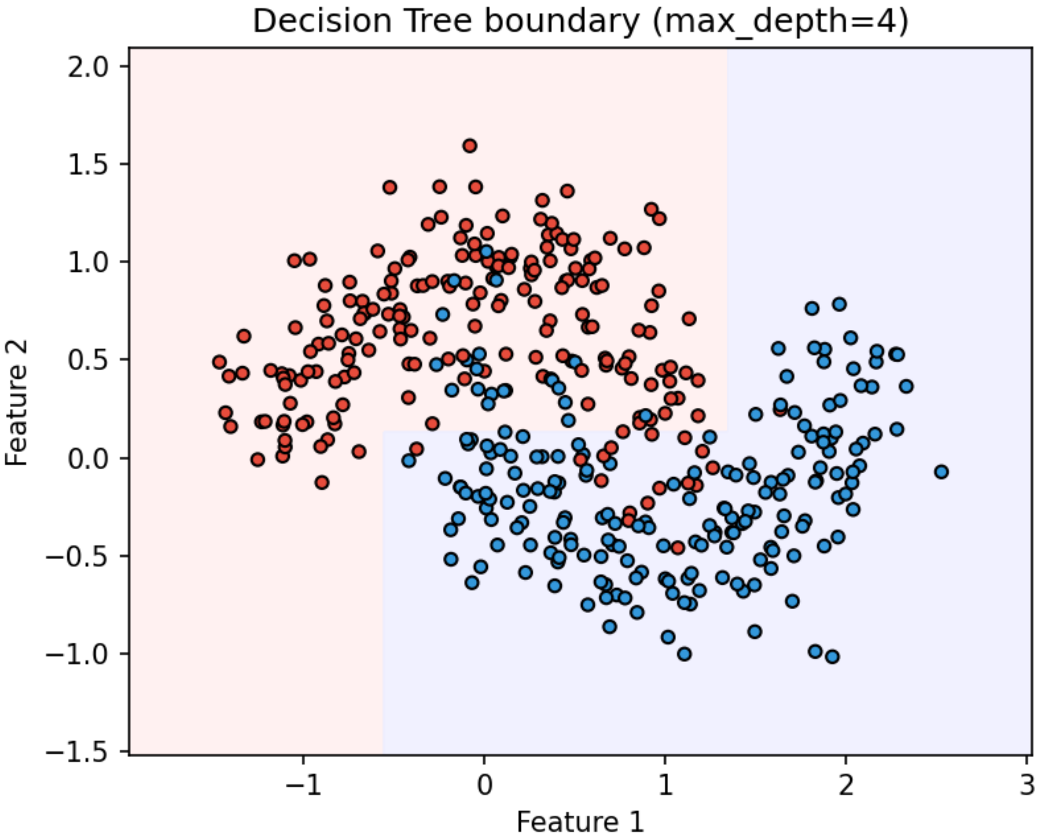

图 1: 决策树在两类数据上的决策边界。

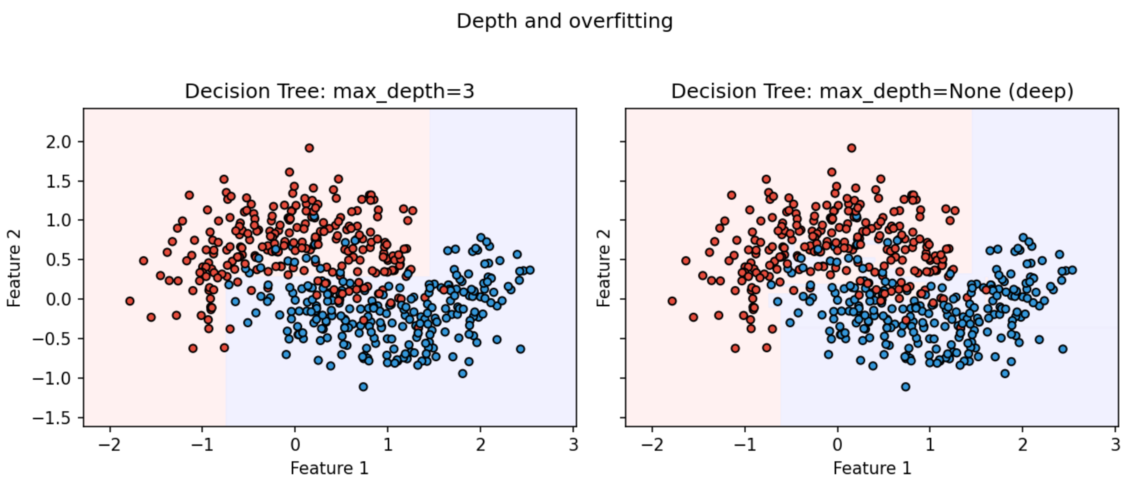

图 2: 树深度影响：浅层与深层（过拟合）对比。

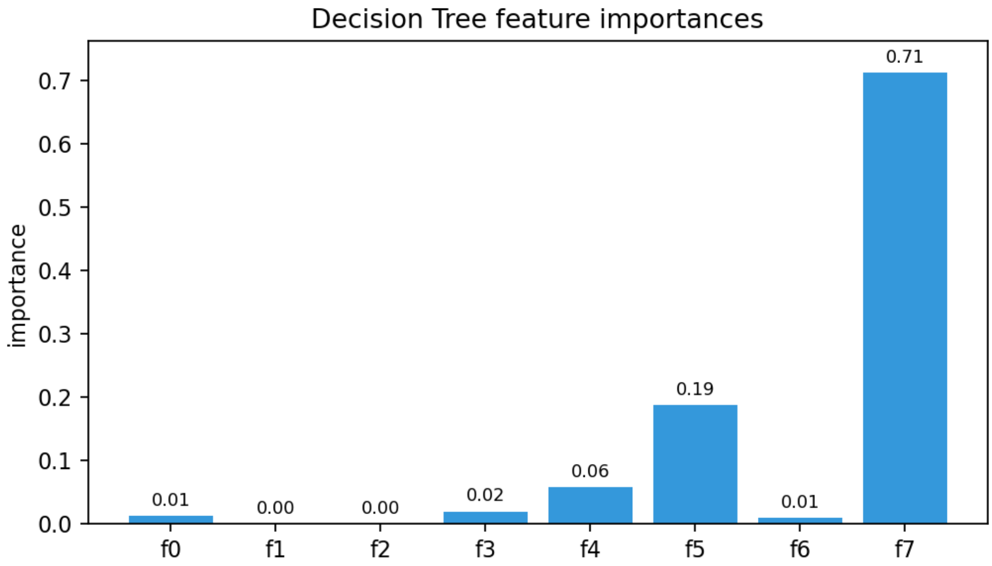

图 3: 决策树的特征重要性可视化。

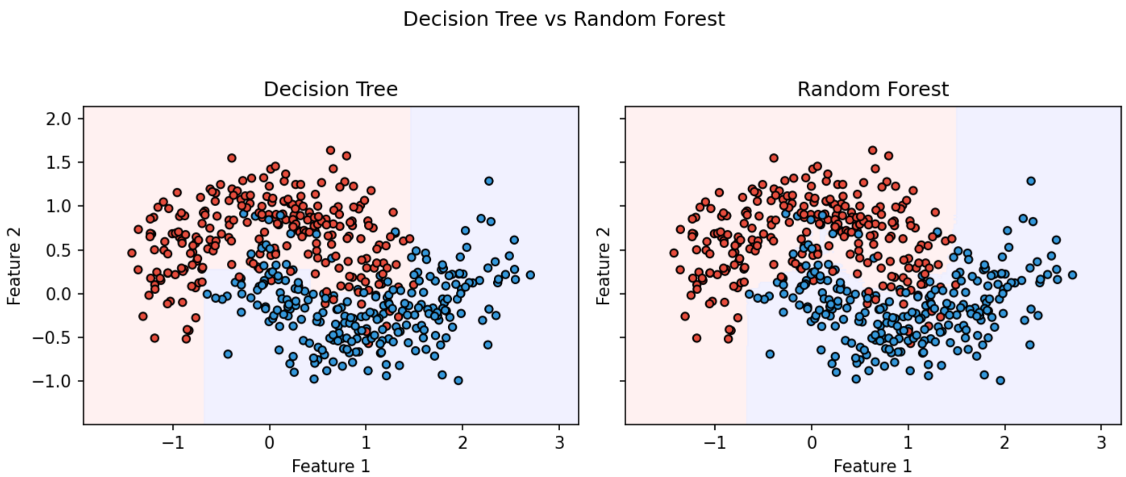

图 4: 单棵决策树与随机森林的决策边界对比。

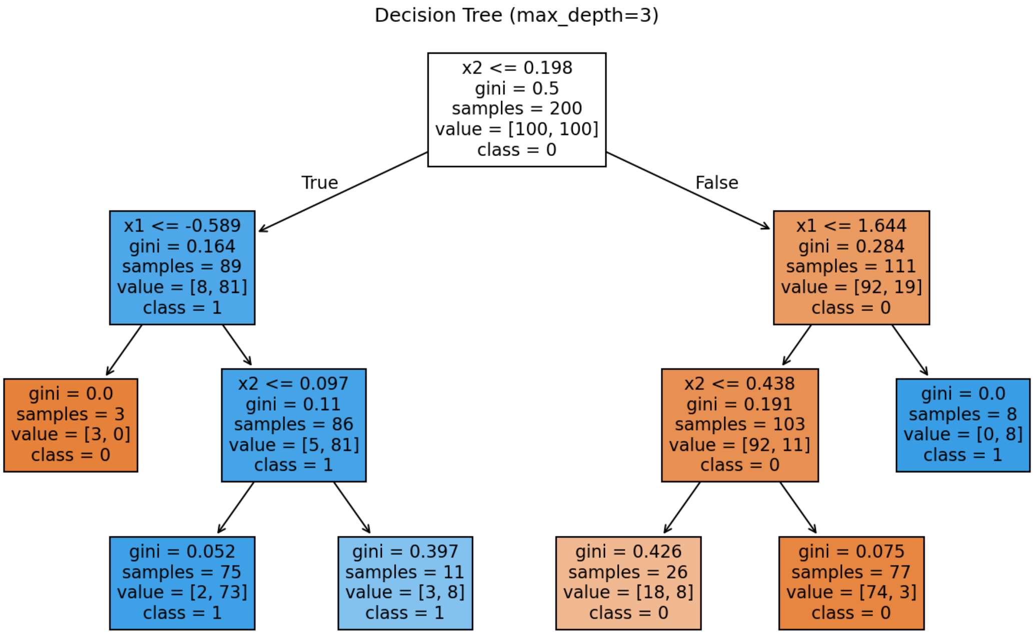

图 5: 树结构可视化（max_depth=3）。

6 总结

决策树作为可解释且灵活的基线模型，在适当的正则化或与集成方法（随机森林、梯度提升）结合时，能在多种任务上取得具有竞争力的表现。