随机森林：理论与实践

2025 年 9 月 10 日

1 引言

随机森林（Random Forest）是基于“装袋”（bagging）与“特征子采样”的集成学习方法。它通过对多个随机化的决策树进行投票（分类）或求均值（回归），显著降低方差、提升泛化能力，同时对数据预处理的要求较低。

2 原理与公式

随机森林的两大关键：自助采样（bootstrap）与特征子采样（每次划分仅在随机抽取的特征子集上搜索最优划分）。设有 B 棵树 $\{T_b\}_{b=1}^{B}$，每棵树在各自的自助样本 $\mathcal{D}_b$ 上训练。分类任务的集成预测为多数表决：

$$\hat{y} = \operatorname{mode}\big(T_1(\mathbf{x}), \dots, T_B(\mathbf{x})\big), \tag{1}$$

回归任务的集成预测为简单平均：

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} T_b(\mathbf{x}). \tag{2}$$

由于特征子采样降低了各基学习器的相关性，随着树数 B 的增加，整体方差可进一步下降。分类中常用的经验选择是 `max_features`=$\sqrt{p}$（p 为特征数）。

袋外（OOB）估计：每次自助采样平均会留下约 $\approx 36\%$ 的“袋外”样本。利用袋外样本做内部验证可得到接近无偏的泛化评估，无需额外划分验证集或交叉验证。

常用超参数包括:树的数量 B(`n_estimators`)、每次划分考虑的特征数(`max_features`)、树深与叶子最小样本数（正则化）、以及 `bootstrap`/`oob_score` 设置等。

3 应用与技巧

- **优点：**默认表现强、鲁棒性好、对特征尺度不敏感、能处理混合类型特征。

- **缺点**：预测与存储开销较单棵树更大；可解释性低于浅层树。
- **正则化与调参**：调整 `max_depth`、`min_samples_leaf`、`max_features`；增大 `n_estimators` 直至 OOB/测试指标收敛。
- **诊断**：使用 OOB 分数、随树数变化的学习曲线；查看特征重要性（必要时用置换重要性）。
- **预处理**：类别特征需独热编码；数值特征一般无需标准化。

4 Python 实战

在本章节目录运行下述命令，图片将保存到 `figures/`：

Listing 1: 生成随机森林配图

```
python gen_random_forest_figures.py
```

Listing 2: gen_random_forest_figures.py 源码

```
"""
Figure generator for the Random Forest 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

```

```python
try:
    from sklearn.datasets import make_moons, make_classification
    from sklearn.ensemble import RandomForestClassifier
except Exception:
    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_rf_decision_boundary_2class(out_dir: str) -> str:
    np.random.seed(0)
    X, y = make_moons(n_samples=500, noise=0.3, random_state=0)
    clf = RandomForestClassifier(
        n_estimators=150, max_depth=None, 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, "Random Forest boundary (n_estimators=150)")
    out_path = os.path.join(out_dir, "rf_decision_boundary_2class.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_rf_n_estimators_compare(out_dir: str) -> str:
    np.random.seed(1)
    X, y = make_moons(n_samples=600, noise=0.28, random_state=1)
    models = [
        (RandomForestClassifier(n_estimators=5, random_state=1), "n_estimators=5"),
        (RandomForestClassifier(n_estimators=200, random_state=1), "n_estimators=200"),
    ]
    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"Random Forest: {title}")
    fig.suptitle("Effect of number of trees")
    out_path = os.path.join(out_dir, "rf_n_estimators_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_rf_max_features_compare(out_dir: str) -> str:
    np.random.seed(2)
    X, y = make_moons(n_samples=600, noise=0.32, random_state=2)
    models = [
        (RandomForestClassifier(max_features=1, n_estimators=150, random_state=2), "max_features=1"),
        (RandomForestClassifier(max_features="sqrt", n_estimators=150, random_state=2), "max_features=sqrt"),
```

```python
    ]
    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"Random Forest: {title}")
    fig.suptitle("Effect of feature subsampling (max_features)")
    out_path = os.path.join(out_dir, "rf_max_features_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_rf_feature_importances(out_dir: str) -> str:
    X, y = make_classification(
        n_samples=800,
        n_features=10,
        n_informative=4,
        n_redundant=3,
        n_repeated=0,
        random_state=7,
        shuffle=True,
    )
    clf = RandomForestClassifier(n_estimators=200, random_state=7)
    clf.fit(X, y)
    importances = clf.feature_importances_

    fig, ax = plt.subplots(figsize=(7.0, 4.0), dpi=160)
    idx = np.arange(importances.size)
    ax.bar(idx, importances, color="#2ECC71")
    ax.set_xticks(idx)
    ax.set_xticklabels([f"f{i}" for i in idx])
    ax.set_ylabel("importance")
    ax.set_title("Random Forest 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, "rf_feature_importances.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
```

```
    return out_path


def fig_rf_oob_curve(out_dir: str) -> str:
    np.random.seed(3)
    X, y = make_classification(
        n_samples=1200,
        n_features=15,
        n_informative=5,
        n_redundant=5,
        random_state=3,
    )

    trees = np.unique(np.linspace(5, 300, 15).astype(int))
    oob_scores = []
    for n in trees:
        # OOB requires bootstrap=True
        rf = RandomForestClassifier(
            n_estimators=n, oob_score=True, bootstrap=True,
                random_state=3
        )
        rf.fit(X, y)
        oob_scores.append(rf.oob_score_)

    fig, ax = plt.subplots(figsize=(6.5, 4.0), dpi=160)
    ax.plot(trees, oob_scores, marker="o", color="#9B59B6")
    ax.set_xlabel("n_estimators")
    ax.set_ylabel("OOB score")
    ax.set_title("Out-of-bag score vs number of trees")
    ax.grid(True, linestyle=":", alpha=0.4)
    out_path = os.path.join(out_dir, "rf_oob_curve.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def main():
    out_dir = _ensure_figures_dir(None)
    generators = [
        fig_rf_decision_boundary_2class,
        fig_rf_n_estimators_compare,
        fig_rf_max_features_compare,
```

```
        fig_rf_feature_importances,
        fig_rf_oob_curve,
    ]
    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

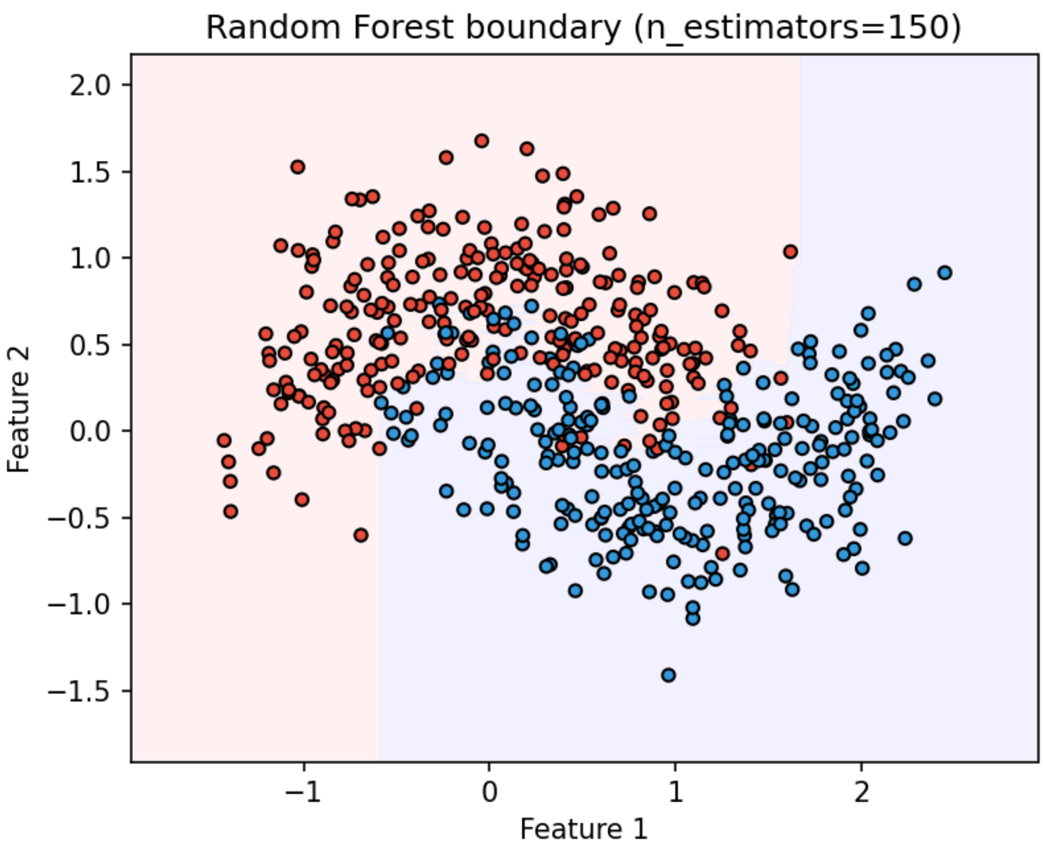

图 1: 随机森林在两类数据上的决策边界。

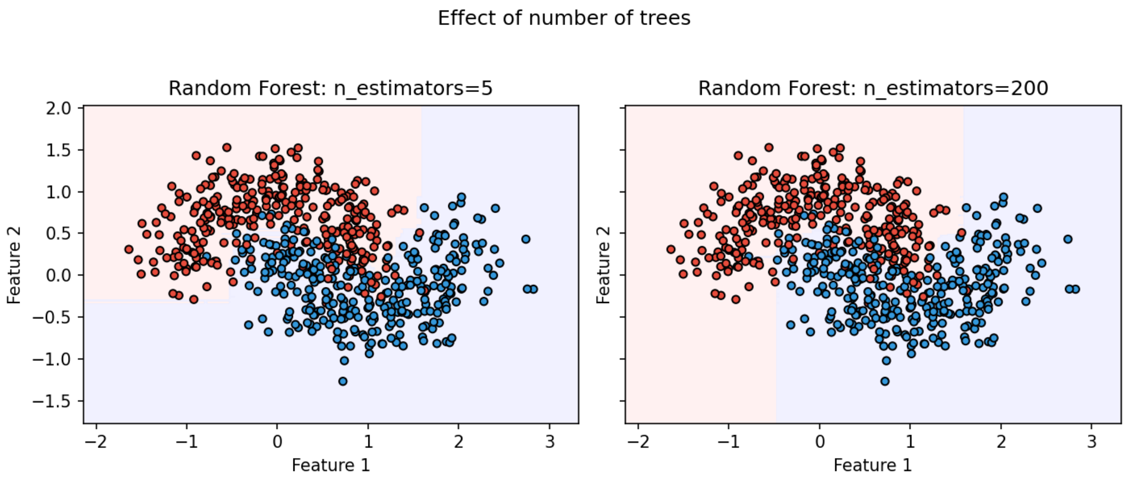

图 2: 树数量影响：少树 vs 多树的对比。

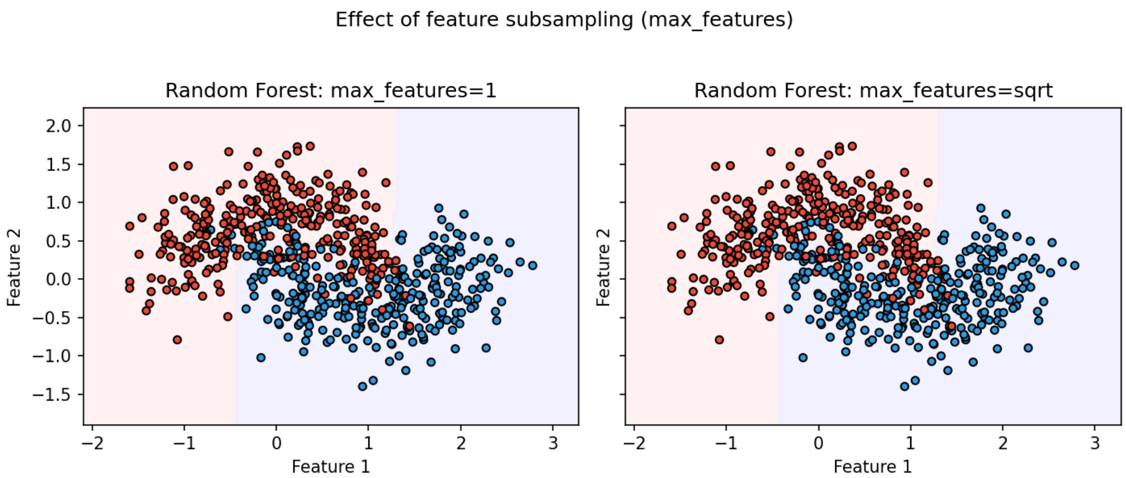

图 3: 不同 `max_features` 下的决策边界对比。

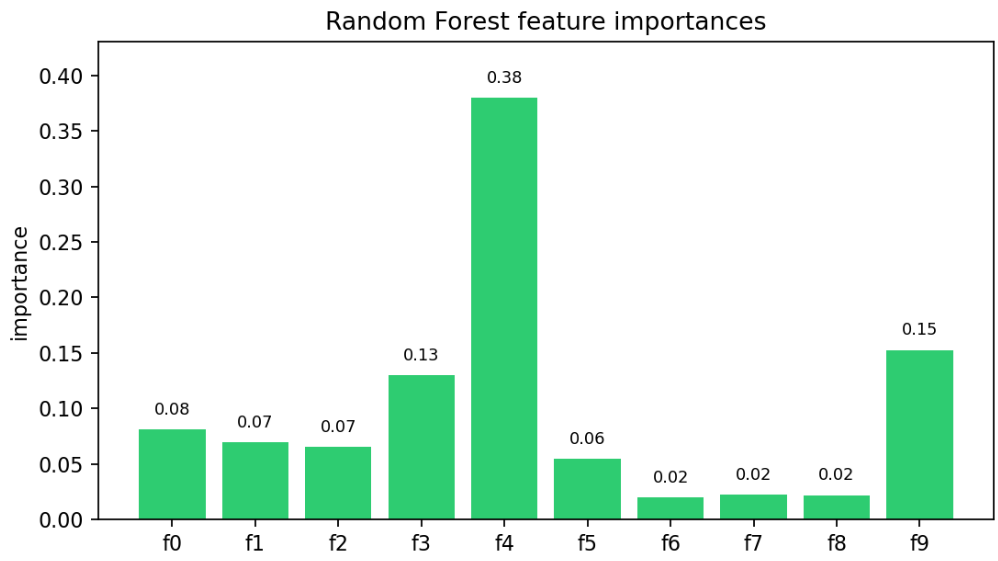

图 4: 随机森林的特征重要性可视化。

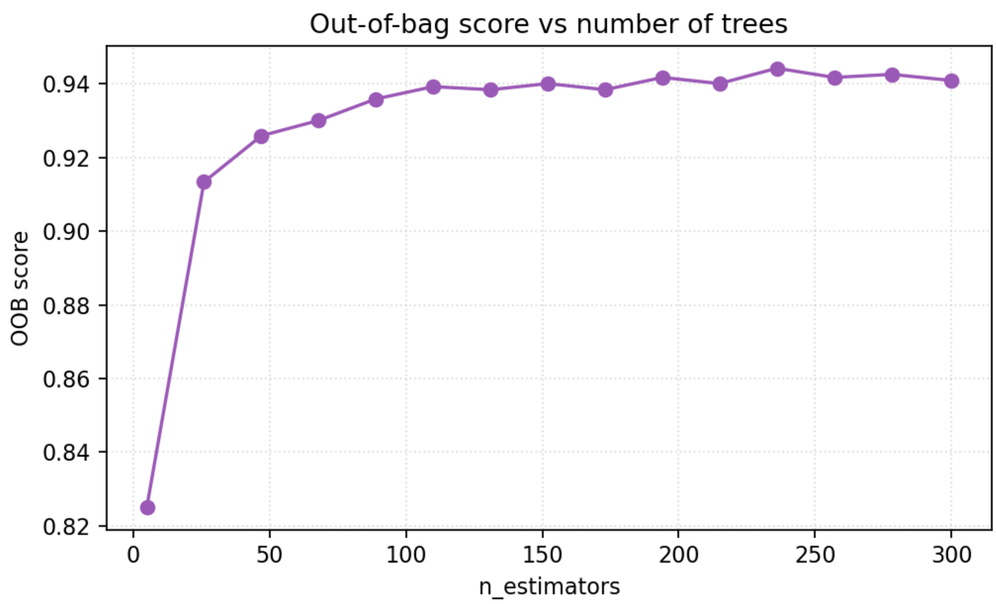

图 5: 袋外（OOB）分数随树数量变化的曲线。

6 总结

随机森林通过“装袋 + 特征子采样”有效降低方差，提供 OOB 内部验证与可用的特征重要性，是工程中可靠的通用基线模型。通过合适的树数与正则化参数，可在准确率与效率之间取得良好平衡。