Random Forests: Theory and Practice

September 10, 2025

1 Introduction

Random forests are ensemble models that aggregate many decision trees trained on bootstrap samples and with randomized feature selection. By averaging (regression) or majority voting (classification), they reduce variance, improve generalization, and retain robustness without extensive preprocessing.

2 Theory and Formulas

Random forests combine *bagging* and *feature subsampling*. Let $\{T_b\}_{b=1}^{B}$ be trees trained on bootstrap samples $\mathcal{D}_b$ and using a random subset of features at each split. For classification, the prediction is the majority vote

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

and for regression, the average

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

Variance reduces as B increases when individual trees are not perfectly correlated (promoted by feature subsampling). A common choice is `max_features` $= \sqrt{p}$ for classification with p features.

Out-of-bag (OOB) estimation: each bootstrap leaves out roughly $\approx 36\%$ of samples. The model's OOB score uses these held-out samples as an internal validation, providing a nearly unbiased generalization estimate without extra cross-validation.

Key hyperparameters: number of trees B (`n_estimators`), split feature count (`max_features`), tree depth and leaf size (regularization), and bootstrap/OOB settings.

3 Applications and Tips

- **Pros:** strong out-of-the-box performance, handles mixed feature types, robust to outliers and monotone transforms, little scaling required.
- **Cons:** larger memory and inference cost than a single tree; less interpretable than a shallow tree.
- **Regularization:** tune `max_depth`, `min_samples_leaf`, `max_features`; increase `n_estimators` until OOB/test metrics stabilize.
- **Diagnostics:** use OOB score, learning curves vs trees, and check feature importances (and, when needed, permutation importance).

- **Preprocessing:** one-hot encode categorical features; no need to standardize numeric features.

4 Python Practice

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

Listing 1: Generate Random Forest figures

```
python gen_random_forest_figures.py
```

Listing 2: gen_random_forest_figures.py

```python
"""
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

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):
```

```python
    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"),
    ]
    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

```

```python

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 Result

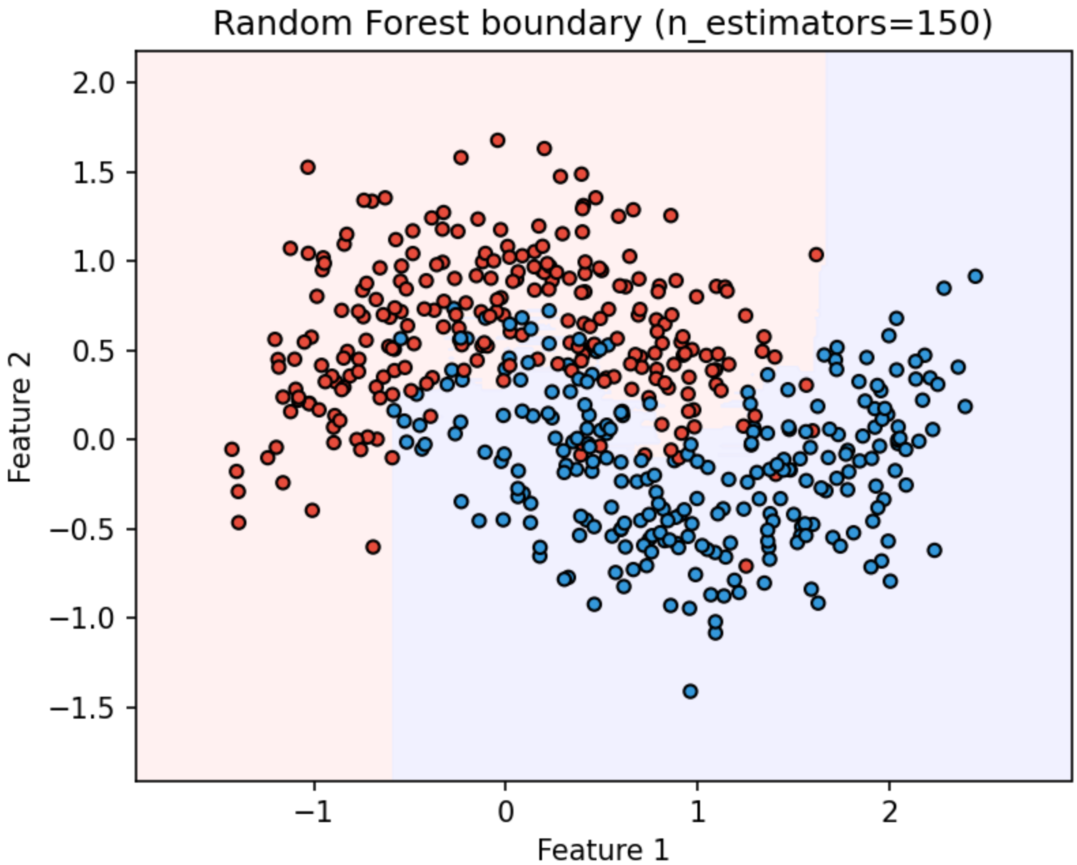

Figure 1: Random forest decision boundary on a 2-class dataset.

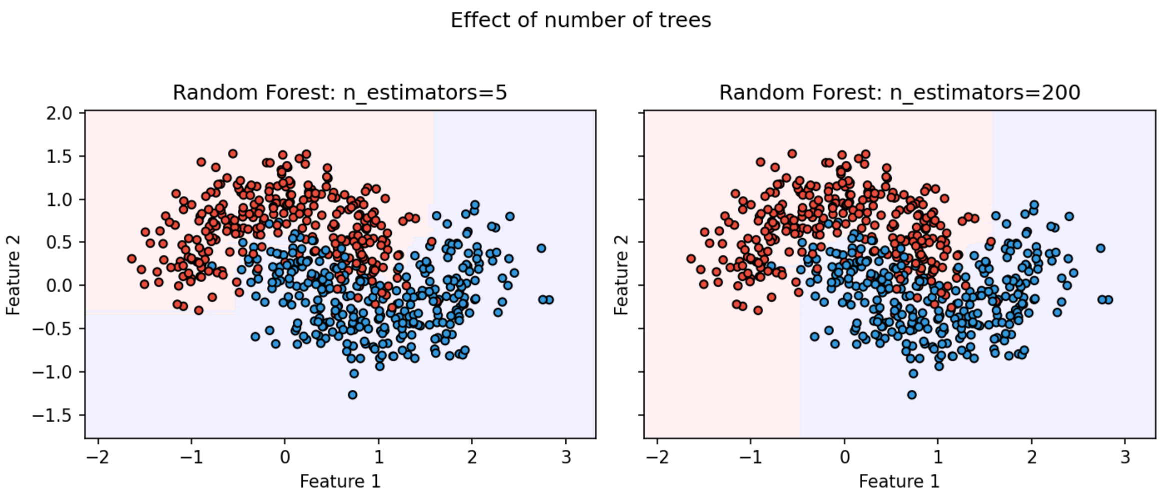

Figure 2: Effect of number of trees: small vs large ensemble.

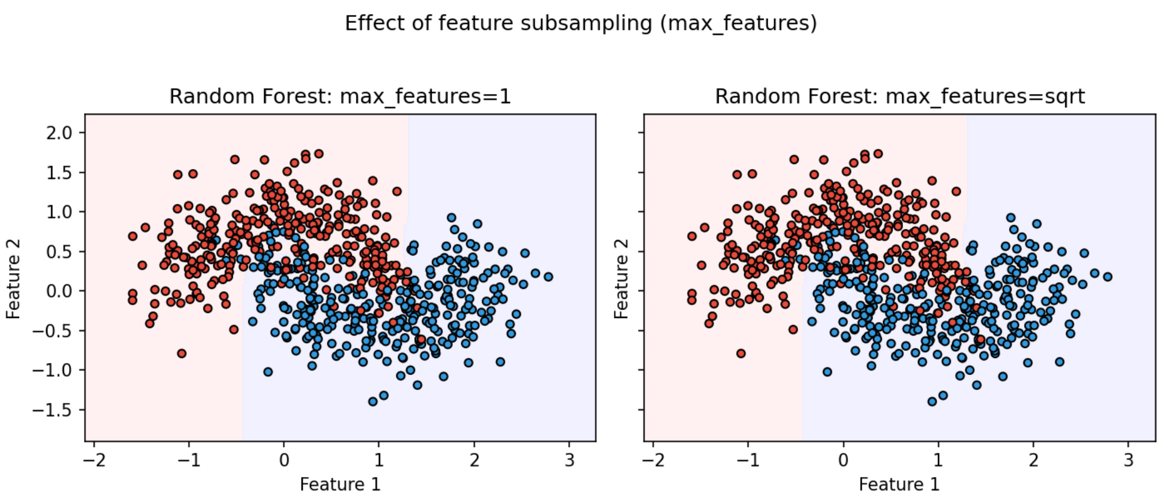

Figure 3: Decision boundaries with different `max_features`.

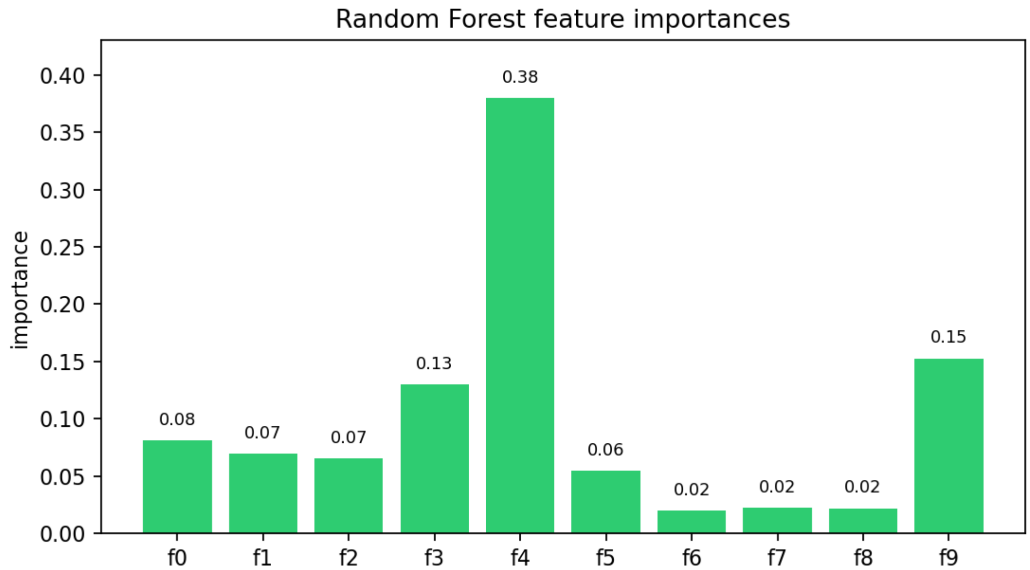

Figure 4: Feature importances from a random forest.

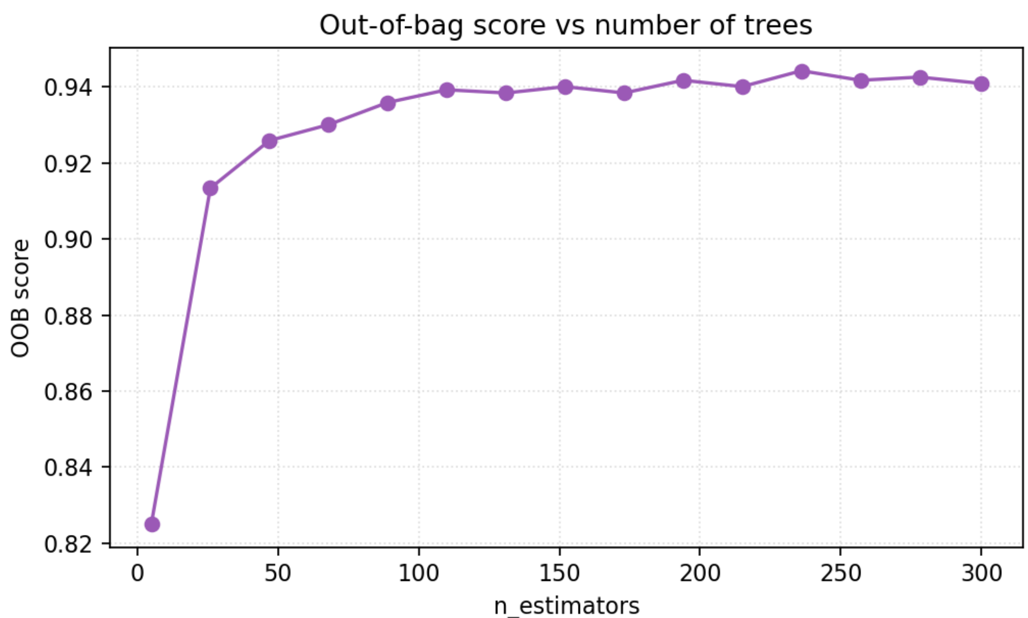

Figure 5: Out-of-bag score vs number of trees.

6 Summary

Random forests provide a strong, reliable baseline for many tasks. They reduce variance through bagging and feature subsampling, offer OOB validation, and yield useful importance measures. Tune tree count and regularization for the best trade-off between accuracy and efficiency.