

Random Forests in Practice

Data Science Lab

2025-11-21

Table of contents

Overview	1
Mathematical Model	1
Environment Setup	2
Data Loading and Preparation	3
Model Training	4
Diagnostics	5
Receiver Operating Characteristic	5
Confusion Matrix	6
Feature Importance Visualization	7
Complex Visualization: Decision Boundaries	8
Error Rate vs. Number of Trees	10
Hyperparameter Considerations	11
Practical Tips	12
References	12

Overview

Random forests are ensemble models that aggregate many decision trees to reduce variance and improve generalization.¹ This document walks through training and interpreting a random forest classifier in Python, with a mix of narrative, math, and visuals.

Mathematical Model

Each tree T_b is trained on a bootstrap sample \mathcal{D}_b and a random subset of features. The forest prediction for a classification task with B trees is the majority vote:

¹Introduced the random forest algorithm with theoretical justification and empirical benchmarks.

$$\hat{y} = \text{mode}(\{T_b(\mathbf{x})\}_{b=1}^B)$$

For regression, the trees are averaged:

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

The randomization across bootstrapped data and feature subsampling drives decorrelation between trees, delivering lower variance than single-tree models.

Environment Setup

```
import importlib
import subprocess
import sys

def ensure(package):
    try:
        importlib.import_module(package)
    except ImportError:
        subprocess.check_call([sys.executable, "-m", "pip", "install", package])

for pkg in ("numpy", "pandas", "seaborn", "matplotlib", "scikit-learn"):
    ensure(pkg)
```

```
Requirement already satisfied: scikit-learn in /Users/luciusjmorningstar/Desktop/GIT-
REPOSITORY/JJB_Gallery/.venv/lib/python3.13/site-packages (1.7.2)
Requirement already satisfied: numpy>=1.22.0 in /Users/luciusjmorningstar/Desktop/GIT-
REPOSITORY/JJB_Gallery/.venv/lib/python3.13/site-packages (from scikit-
learn) (2.3.5)
Requirement already satisfied: scipy>=1.8.0 in /Users/luciusjmorningstar/Desktop/GIT-
REPOSITORY/JJB_Gallery/.venv/lib/python3.13/site-packages (from scikit-
learn) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in /Users/luciusjmorningstar/Desktop/GIT-
REPOSITORY/JJB_Gallery/.venv/lib/python3.13/site-packages (from scikit-
learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /Users/luciusjmorningstar/Desktop/GIT-
REPOSITORY/JJB_Gallery/.venv/lib/python3.13/site-packages (from scikit-
learn) (3.6.0)
```

```

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer, make_classification
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import RocCurveDisplay, ConfusionMatrixDisplay, classification_report
from sklearn.decomposition import PCA
from sklearn.inspection import DecisionBoundaryDisplay

sns.set_theme(style="whitegrid")

```

Data Loading and Preparation

We will use the Breast Cancer Wisconsin dataset bundled with scikit-learn, which contains 30 features computed from digitized fine needle aspirate images.²

```

dataset = load_breast_cancer(as_frame=True)
df = dataset.frame
df.head()

```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.198

Split the data into training and testing sets (stratified to maintain label balance).

```

X_train, X_test, y_train, y_test = train_test_split(
    df.drop(columns="target"),
    df["target"],
    test_size=0.25,
    random_state=42,
    stratify=df["target"]
)

```

²Official description of the dataset, feature definitions, and usage considerations.

```
)  
  
X_train.shape, X_test.shape
```

((426, 30), (143, 30))

Model Training

```
rf = RandomForestClassifier(  
    n_estimators=400,  
    max_features="sqrt",  
    min_samples_leaf=2,  
    random_state=42,  
    n_jobs=-1  
)  
rf.fit(X_train, y_train)
```

n_estimators	400
criterion	'gini'
max_depth	None
min_samples_split	2
min_samples_leaf	2
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	-1
random_state	42
verbose	0
warm_start	False
class_weight	None
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

Evaluate cross-validated training performance to estimate generalization ability.

```
cv_scores = cross_val_score(rf, X_train, y_train, cv=5)
cv_scores.mean(), cv_scores.std()
```

```
(np.float64(0.9600547195622434), np.float64(0.020556647327639923))
```

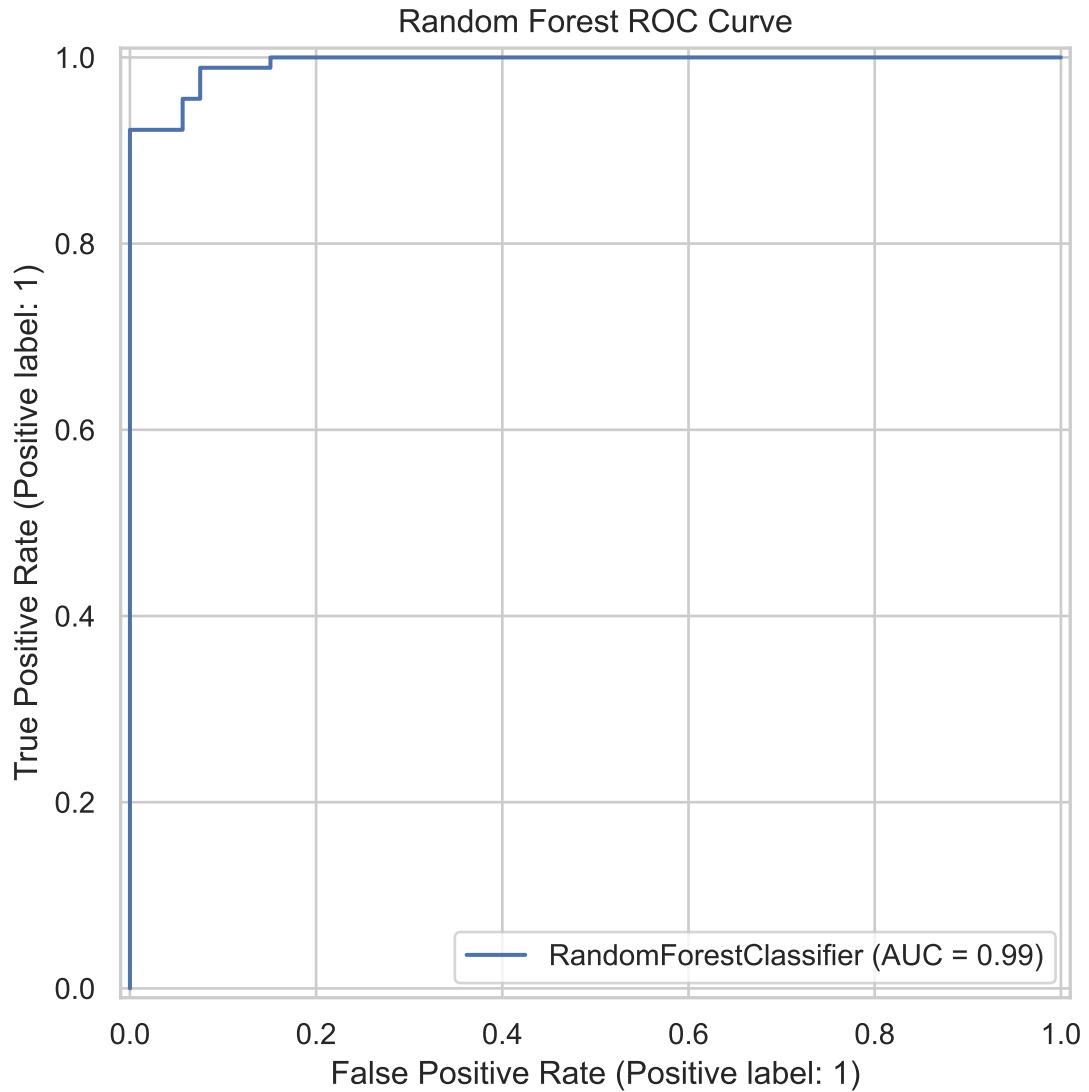
Diagnostics

```
y_pred = rf.predict(X_test)
print(classification_report(y_test, y_pred, target_names=dataset.target_names))
```

	precision	recall	f1-score	support
malignant	0.96	0.92	0.94	53
benign	0.96	0.98	0.97	90
accuracy			0.96	143
macro avg	0.96	0.95	0.95	143
weighted avg	0.96	0.96	0.96	143

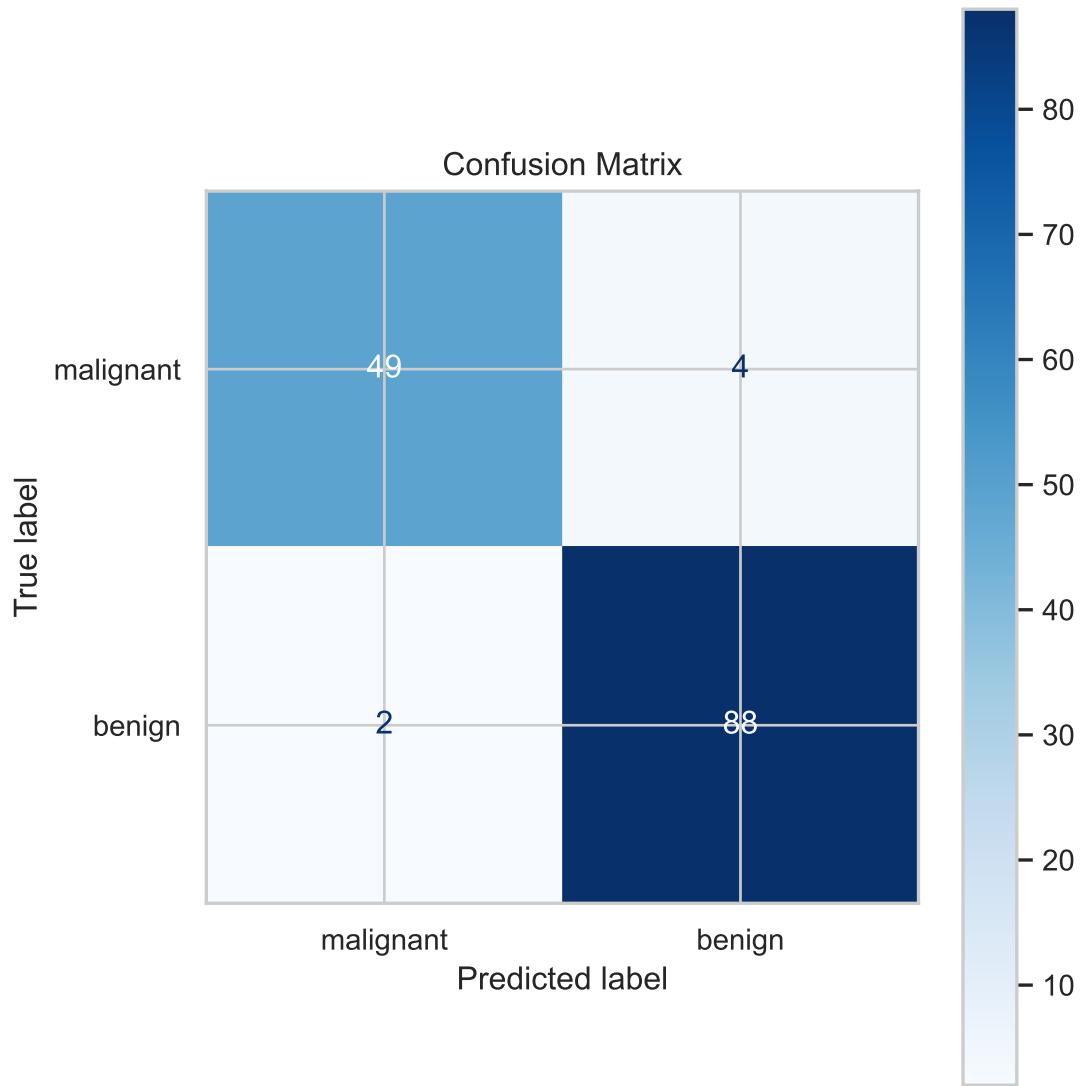
Receiver Operating Characteristic

```
fig, ax = plt.subplots(figsize=(8, 6))
RocCurveDisplay.from_estimator(rf, X_test, y_test, ax=ax)
ax.set_title("Random Forest ROC Curve")
plt.tight_layout()
plt.show()
```



Confusion Matrix

```
fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_estimator(rf, X_test, y_test, display_labels=dataset.target_names)
ax.set_title("Confusion Matrix")
plt.tight_layout()
plt.show()
```



Feature Importance Visualization

```

importances = pd.Series(rf.feature_importances_, index=df.columns[:-1]).sort_values(ascending=False)
top_features = importances.head(15)

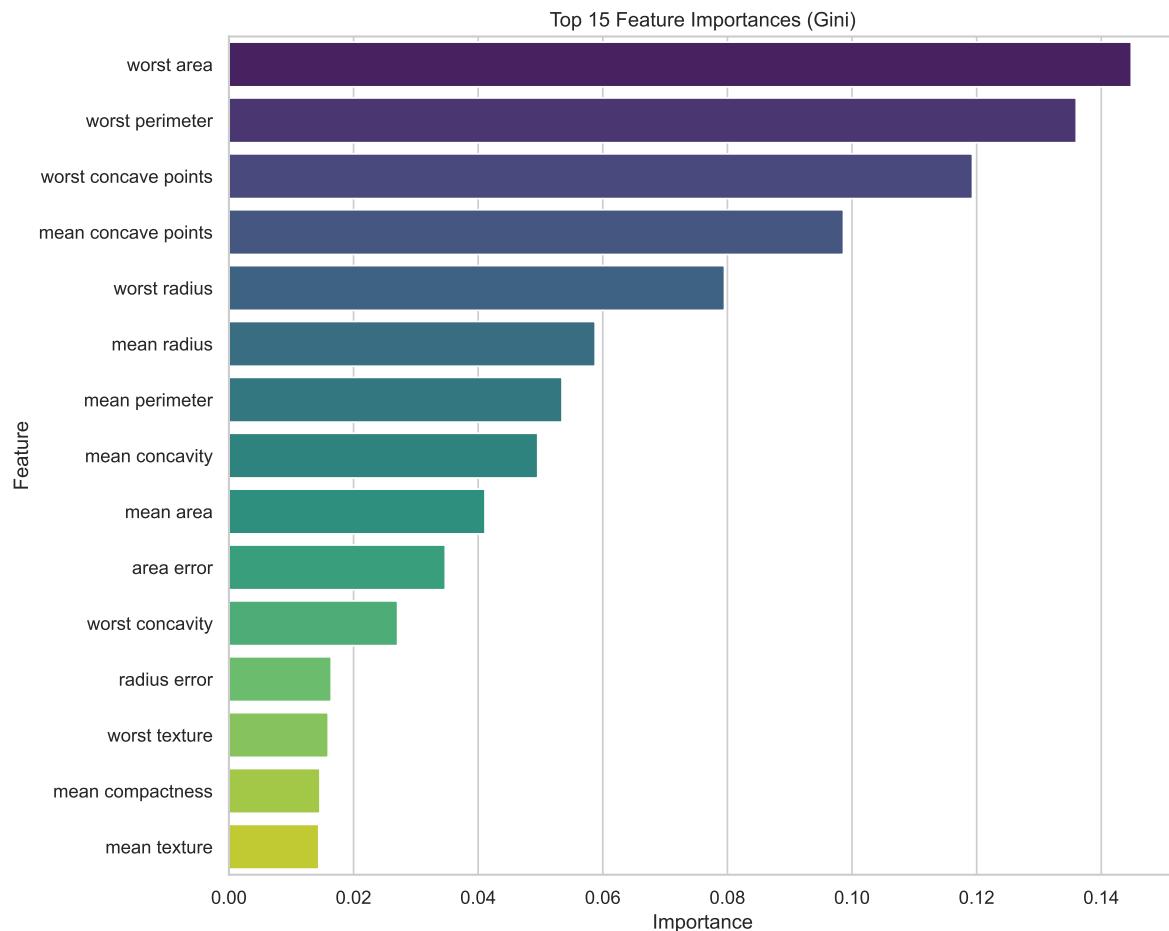
plt.figure(figsize=(10, 8))
sns.barplot(x=top_features.values, y=top_features.index, palette="viridis")
plt.title("Top 15 Feature Importances (Gini)")
plt.xlabel("Importance")

```

```

plt.ylabel("Feature")
plt.tight_layout()
plt.show()

```



Complex Visualization: Decision Boundaries

To visualize the decision boundaries of the high-dimensional Random Forest model, we project the data onto its first two Principal Components (PCA). This allows us to see how the model separates classes in a 2D latent space.

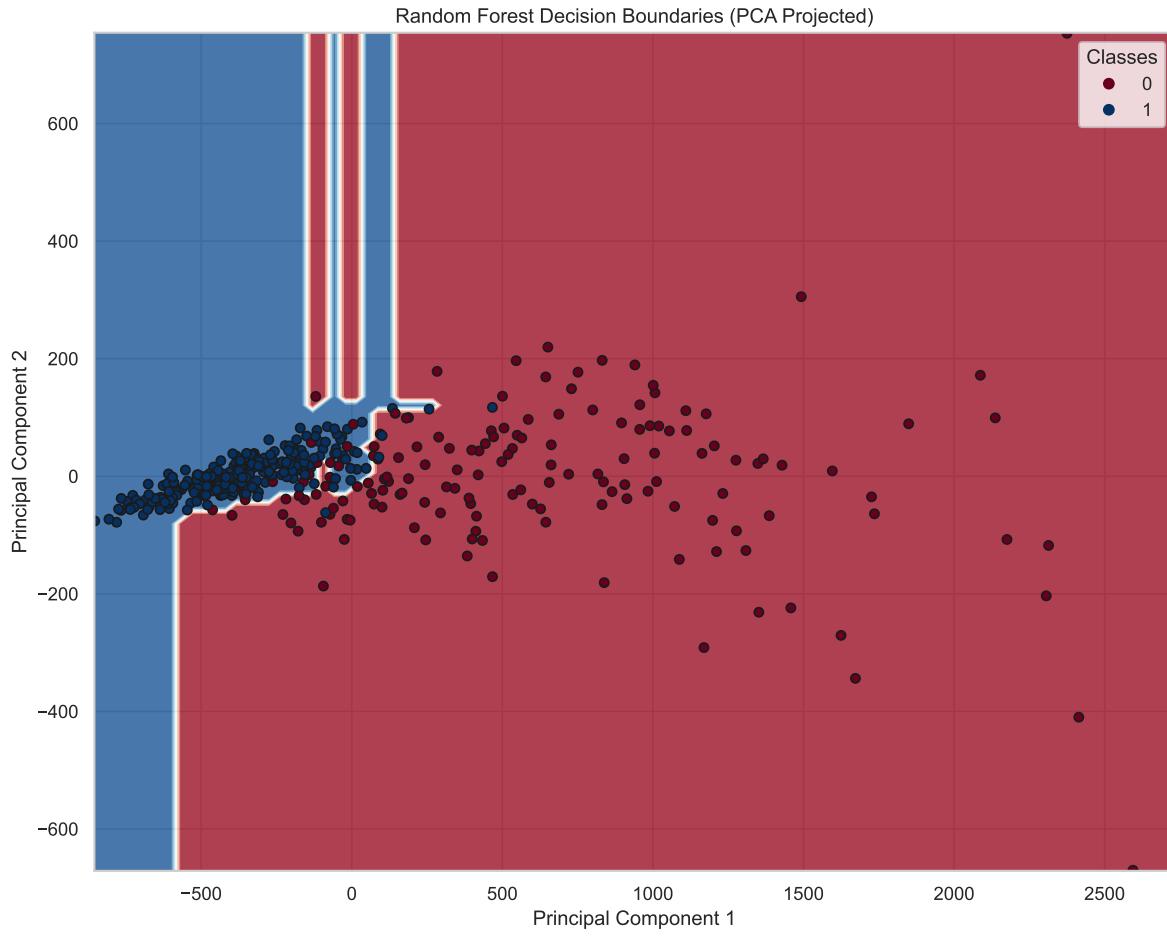
```

# PCA Projection
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train)

```

```
# Train a new RF on PCA components for visualization
rf_pca = RandomForestClassifier(n_estimators=100, random_state=42)
rf_pca.fit(X_pca, y_train)

# Plot Decision Boundary
fig, ax = plt.subplots(figsize=(10, 8))
DecisionBoundaryDisplay.from_estimator(
    rf_pca,
    X_pca,
    response_method="predict",
    cmap="RdBu",
    alpha=0.8,
    ax=ax,
    xlabel="Principal Component 1",
    ylabel="Principal Component 2",
)
scatter = ax.scatter(X_pca[:, 0], X_pca[:, 1], c=y_train, cmap="RdBu", edgecolors="k", s=30)
ax.set_title("Random Forest Decision Boundaries (PCA Projected)")
plt.legend(*scatter.legend_elements(), title="Classes")
plt.tight_layout()
plt.show()
```



Error Rate vs. Number of Trees

This graph illustrates how the model's Out-of-Bag (OOB) error rate stabilizes as the number of trees in the forest increases, demonstrating the ensemble effect.

```
n_estimators_range = range(15, 300, 10)
oob_errors = []

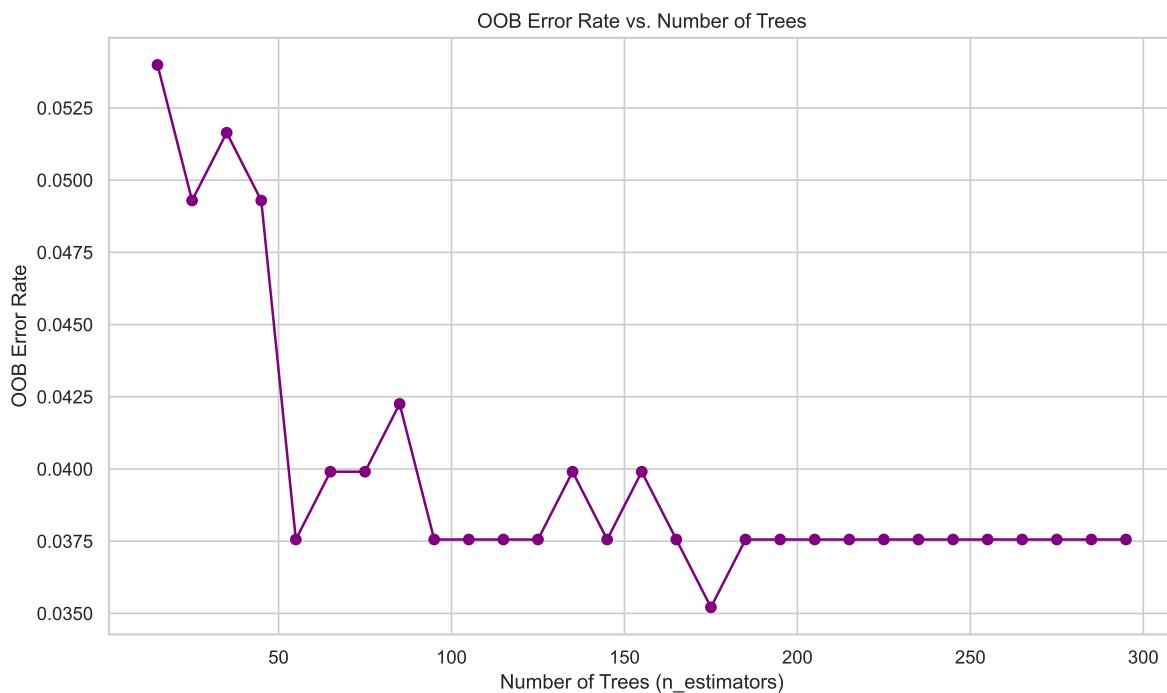
for n in n_estimators_range:
    rf_oob = RandomForestClassifier(n_estimators=n, warm_start=True, oob_score=True, random_
    rf_oob.fit(X_train, y_train)
    oob_errors.append(1 - rf_oob.oob_score_)

plt.figure(figsize=(10, 6))
```

```

plt.plot(n_estimators_range, oob_errors, marker='o', linestyle='-', color='purple')
plt.title("OOB Error Rate vs. Number of Trees")
plt.xlabel("Number of Trees (n_estimators)")
plt.ylabel("OOB Error Rate")
plt.grid(True)
plt.tight_layout()
plt.show()

```



Hyperparameter Considerations

- `n_estimators`: Increasing trees generally improves stability until diminishing returns set in.
- `max_depth` or `min_samples_leaf`: Control tree complexity, mitigating overfitting.
- `max_features`: Governs the degree of feature randomness; `sqrt` is typical for classification.
- `class_weight`: Useful for imbalanced datasets to penalize misclassification of minority classes.

Grid search or Bayesian optimization can systematically explore these settings.³

³Demonstrated the efficiency gains of random search over grid search for hyperparameter tuning.

Practical Tips

- **Feature scaling:** Not required because trees are invariant to monotonic transformations.
- **Missing values:** scikit-learn's implementation does not handle NaNs; impute beforehand.
- **Interpretability:** Use SHAP values or permutation importance for richer explanations.
- **Out-of-bag (OOB) estimates:** Enable `oob_score=True` to get a built-in validation metric without a separate hold-out set.

References

- Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32. [https://doi.org/10.1023/A:1010933404324⁴](https://doi.org/10.1023/A:1010933404324)
- scikit-learn Breast Cancer Dataset docs. [https://scikit-learn.org/stable/datasets/toy_dataset.html⁵](https://scikit-learn.org/stable/datasets/toy_dataset.html)
- Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13, 281–305. [https://jmlr.org/papers/v13/bergstra12a.html⁶](https://jmlr.org/papers/v13/bergstra12a.html)

⁴breiman2001

⁵sklearn_breast

⁶bergstra2012