# Random Forests in Practice

Data Science Lab

2025-11-21

## Table of contents

### 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:

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

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

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

```python
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
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import RocCurveDisplay, ConfusionMatrixDisplay, classification_report

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 | 0.300 |
| 1 | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.086 |
| 2 | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.197 |
| 3 | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.241 |
| 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"]
)

X_train.shape, X_test.shape
```

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

```
((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))
```

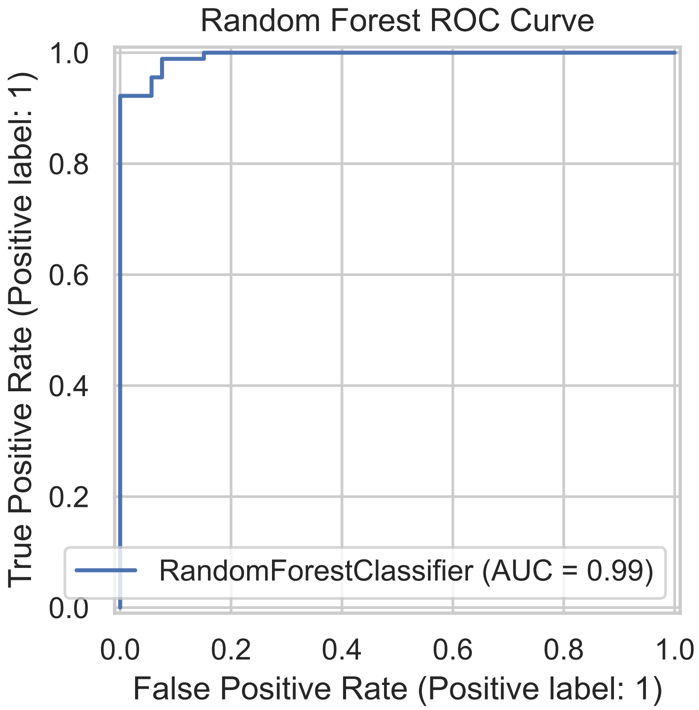
```
              precision    recall  f1-score   support

   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
```

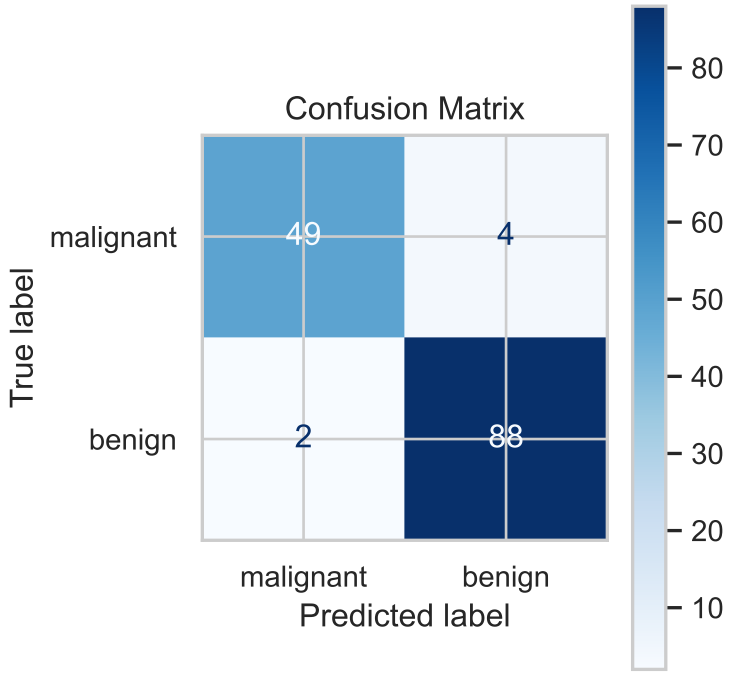
### Receiver Operating Characteristic

```
fig, ax = plt.subplots(figsize=(6, 4))
RocCurveDisplay.from_estimator(rf, X_test, y_test, ax=ax)
ax.set_title("Random Forest ROC Curve")
plt.tight_layout()
```



**Confusion Matrix**

```
fig, ax = plt.subplots(figsize=(4, 4))
ConfusionMatrixDisplay.from_estimator(rf, X_test, y_test, display_labels=dataset.target_names
ax.set_title("Confusion Matrix")
plt.tight_layout()
```

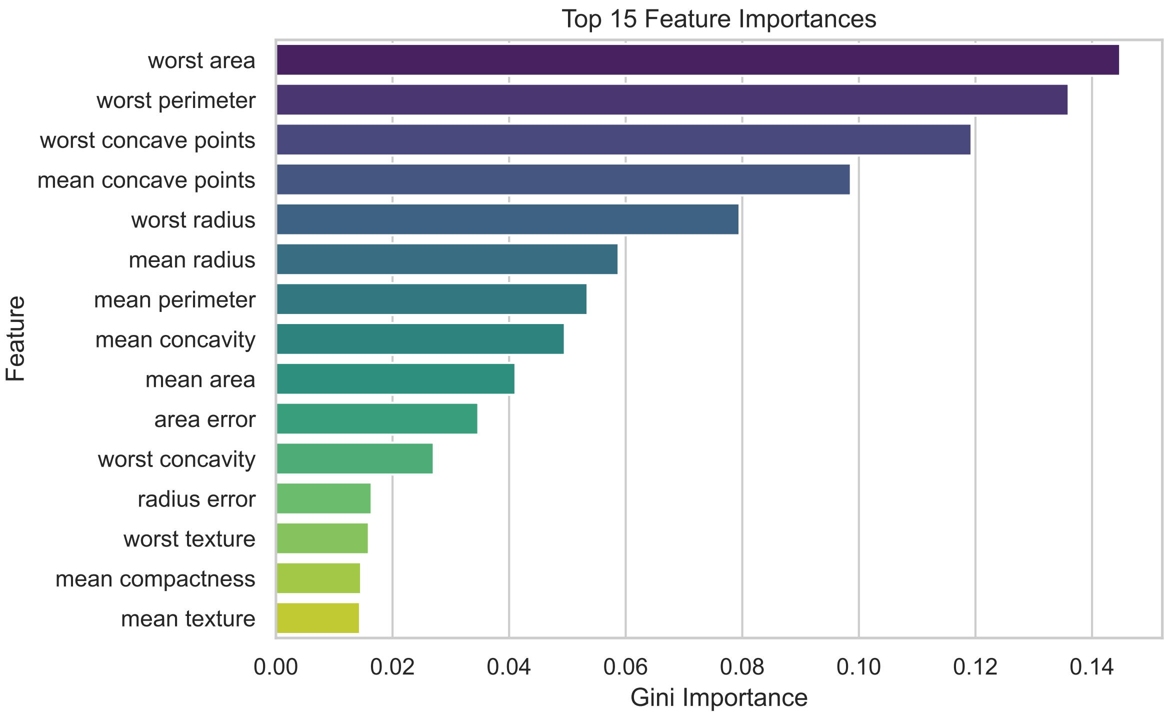


### Feature Importance Visualization

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

try:
    sns
except NameError:
    import seaborn as sns
    sns.set_theme(style="whitegrid")

plt.figure(figsize=(8, 5))
sns.barplot(x=top_features.values, y=top_features.index, palette="viridis")
plt.title("Top 15 Feature Importances")
plt.xlabel("Gini Importance")
plt.ylabel("Feature")
plt.tight_layout()
```



## 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]

## 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.

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

## References

- Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324[4]
- scikit-learn Breast Cancer Dataset docs. https://scikit-learn.org/stable/datasets/toy_dataset.html[5]
- 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.html[6]

[4] b̆reiman2001

[5] ŝklearn_breast

[6] b̆ergstra2012