Understanding Random Forest: A Beginner's Guide Using Heart Disease Data

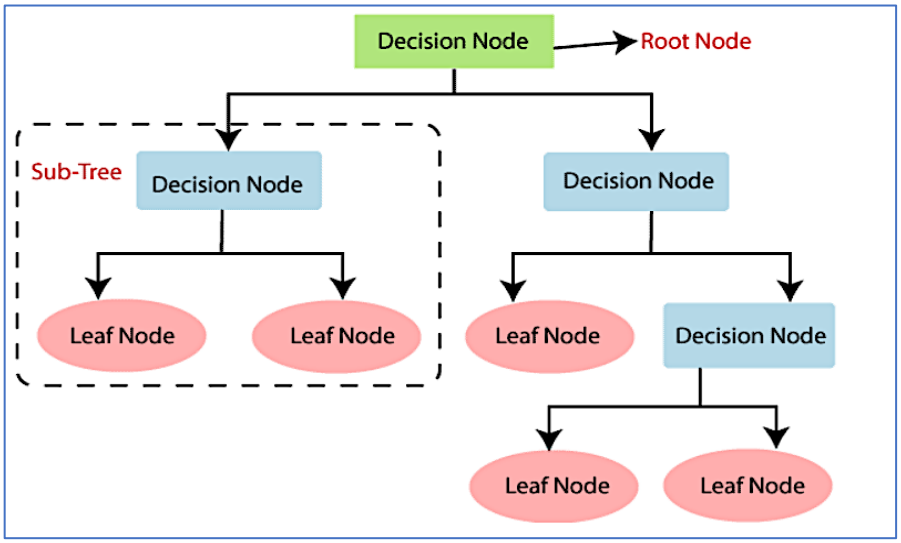
Random Forest is an extremely powerful and simple machine learning algorithm to be used in supervised learning. If you're new to data science or you are trying to learn about ensemble methods, this tutorial will provide you with the required background about random forests — using a real life medical dataset for heart disease prediction.

By the end, you won't just use Random Forest — you'll truly understand it.

Before diving into code, it's good to know what exactly a Random Forest is — and why Random Forests work so effectively in machine learning tasks (like classification).

1.1. The Problem with a Single Decision Tree

Simplest and most intuitive as models in machine learning is Decision Tree. It's a flowchart: based on a set of decisions (if else splits) it predicts an output.



Source: (Sharma, 2021)

The major downside of decision trees is that they're very sensitive to training data. The tree can completely change if the data changes slightly (Sharma, 2021). They are overfitted, performing well on the training data but badly on new unseen test data.

Random Forest acquires this by bridging many decision trees in a single powerful model (B, 2021). It is an algorithm from a category of algorithms called ensemble learning — the idea here is to

have multiple models trained and their outputs combined to provide better results. Think of it like asking multiple doctors for a diagnosis. If one misjudges, others can correct the error. That's the basic idea behind Random Forest.

1.2. Why Random Forest Works So Well

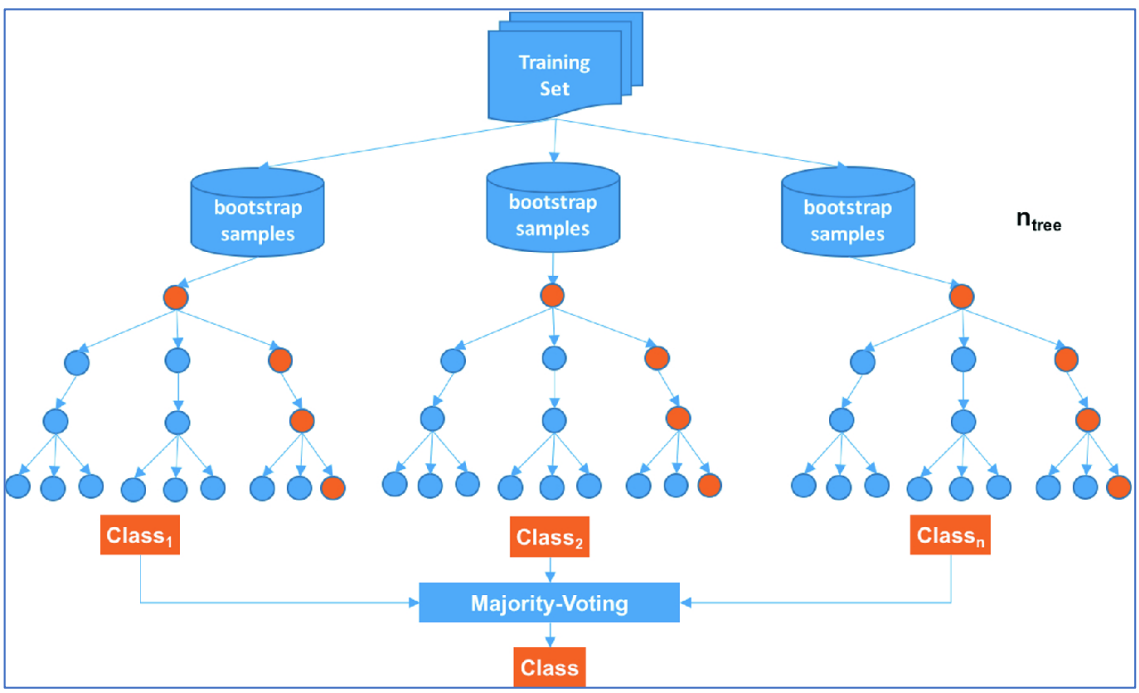
- Not vulnerable to overfitting (contrary to single decision trees)
- Handles both categorical and numerical features
- Great at capturing non-linear patterns
- Missing values and importances are handled automatically.
- Consistent with common NLP practice (this should not need adjusting when training multiple models together).

2. How It Works

What happens behind the scenes is:

Bootstrap Sampling (Bagging)

It generates several subsets of the original dataset through sampling with replacement (e.g. bootstrapping). Decisions trees then filter the data with each subset, and it is used to train a different decision tree.



Source: (B, 2021)

Random Feature Selection

Upon a necessity to split a node, a tree does not inspect all features—only a random subset. It helps adding diversity in the trees and thus avoiding them getting too similar.

Tree Building

The maximum potential of each tree is built unless max_depth is set. Trees are grown with random inputs and no pruning is applied.

Voting (Classification)

In making a prediction, each tree in the forest votes on the output. Final prediction is the class, which gets majority vote.

The approach is bagging (bootstrap aggregating) and it improves on model stability and performance.

2.1. Key Concepts and Parameters

In building a Random Forest you have a lot of parameters to tune. Here are the most important:

Parameter	What It Does
n_estimators	Number of trees in the forest. More trees = better performance, up to a point.
criterion	Splitting method: 'gini' for Gini impurity, 'entropy' for information gain.
max_features	Number of features to consider at each split. 'sqrt' is common for classification.
max_depth	Maximum depth of trees. Prevents trees from growing too complex.
min_samples_split	Minimum samples needed to split an internal node.
min_samples_leaf	Minimum samples required to be at a leaf node.
bootstrap	Whether to use bootstrapped samples for training (True by default).
random_state	Seed for reproducibility.
n_jobs	Number of CPU cores to use. -1 means use all available.

2.2. When to Use Random Forest

- You want to have a good starting point for classification or regression.
- You need a robustness with respect to noisy or missing data
- You don’t need to tune complex deep learning models

- You are interested in interpreting more on what things are important or less important.
- You're taking on tabular data (spreadsheets, CSV, database)

2.3. Random Forest in Action: A Step-by-Step Code Walkthrough

We've now understood the concept of Random Forest in theory, now , let's put that knowledge into practice.

We will apply Random Forest for the task of predicting heart disease in this section. I will explain each step of the implementation along the way, from loading the data to training multiple models and interpreting the results.

This model uses:

- 100 trees
- **Gini index** as the splitting criterion
- **Square root** of total features considered at each split

Let's begin.

Step 1: Load the Dataset

First, we load the heart disease dataset using pandas. The following CSV file has different health attributes of various patients like age, blood pressure, etc. along with the target label stating whether the person has heart disease or not.

Why this step is important:

For any type of machine learning we require the data in a format which can be used by the machine learning algorithm.

Dataset Link: [Click here]

What we do:

```
import pandas as pd

# Load the CSV file
df = pd.read_csv("dataset_heart.csv")

# Preview the data
print("First 5 rows:")
df.head()
```

We load the file with pd.read_csv.

Check what the data looks like, particularly the first five rows using .head()

First 5 rows:

	age	sex	chest pain type	resting blood pressure	serum cholestoral	fasting blood sugar	resting electrocardiographic results	max heart rate	exercise induced angina	oldpeak	ST segment	major vessels	thal	heart disease
0	70	1	4	130	322	0	2	109	0	2.4	2	3	3	2
1	67	0	3	115	564	0	2	160	0	1.6	2	0	7	1
2	57	1	2	124	261	0	0	141	0	0.3	1	0	7	2
3	64	1	4	128	263	0	0	105	1	0.2	2	1	7	1
4	74	0	2	120	269	0	2	121	1	0.2	1	1	3	1

The number of rows and columns can be viewed through .shape.

```
[ ] print('Shape of the Dataset')
    df.shape
```

```
Shape of the Dataset
(270, 14)
```

Step 2: Understand the Dataset

It is necessary to examine the dataset to spot problems with missing values or duplicates when we're going to build a model.

Checks we can perform:

- The use of .isnull().sum() counts any missing values.
- To find the repeated records use .duplicated().sum()

```
[ ] # Count missing values in each column
    print("Missing Values Per Column:")
    print(df.isnull().sum())

    # Count total number of duplicate rows
    duplicate_count = df.duplicated().sum()
    print(f"Total Duplicate Rows: {duplicate_count}")
```

```
Missing Values Per Column:
age                                     0
sex                                     0
chest pain type                         0
resting blood pressure                  0
serum cholestoral                       0
fasting blood sugar                     0
resting electrocardiographic results    0
max heart rate                          0
exercise induced angina                 0
oldpeak                                 0
ST segment                              0
major vessels                           0
thal                                    0
heart disease                           0
dtype: int64
Total Duplicate Rows: 0
```

When your dataset is ready (cleaned and separated into the set of features X and the set of labels y), the first step is to split it into a training and a testing set.

For training the model we have 70% data and testing the model using 30%.

```
[ ] from sklearn.model_selection import train_test_split

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42
    )
```

You can use 80-20% or 60-40%, depending upon what suits your dataset size.

Note: Why this matters: The model is taught using the training data. The test data that we test helps us understand the ability of it to work in a new data that is unseen.

Step 3: Building the Random Forest Classifier

So, we start with initializing our first Random Forest model with the following settings:

```
[ ] from sklearn.ensemble import RandomForestClassifier

    model_gini = RandomForestClassifier(
        n_estimators=100,          # Number of trees in the forest
        criterion='gini',          # Function to measure the quality of a split ('gini' or 'entropy')
        max_depth=None,            # Maximum depth of each tree (None = nodes are expanded until all leaves are pure)
        min_samples_split=2,       # Minimum samples required to split an internal node
        min_samples_leaf=1,        # Minimum samples required to be at a leaf node
        max_features='sqrt',       # Number of features to consider when looking for the best split ('sqrt' is good for classification)
        bootstrap=True,            # Whether bootstrap samples are used when building trees
        oob_score=False,           # Whether to use out-of-bag samples to estimate generalization accuracy
        random_state=42,           # Seed for reproducibility
        n_jobs=-1,                 # Number of jobs to run in parallel (-1 = use all processors)
        verbose=1                  # Control verbosity of output
    )
```

- we have the parameter n_estimators=100 which means we're building a forest consisting of 100 trees. More trees generally increase stability.
- This time it's decided from the trees on the basis of what impurity we will use for deciding the best separating jobs, that means which method of splits in the trees will we be using, and we have gini as the method.
- max_features='sqrt': Pick total_features as the max no. of features to consider at each decision's split. Randomness is introduced and it helps to generalize.
- random_state=42: This enshrines the result as reproducible every time you are running the model.
- Uses all CPU cores to train faster (n_jobs=-1).
- verbose=1: This prints the training process in the output.

In simple words, Random Forest is like building 100 different decision trees and letting them vote for the final answer. The more diverse the trees, the better the model usually performs.

Finally, we fit this model on the training data.

```
model_gini.fit(X_train, y_train)
```

Step 4: Making Predictions

Once the model is trained, we use it to **predict outcomes** on the testing set (X_test). These are the new, unseen examples from the 30% remaining data which was not used/exposed during training.

```
y_pred_gini = model_gini.predict(X_test)
```

Step 5: Evaluating the Model

We use two main tools to evaluate how well our model performed:

Confusion Matrix shows:

- True Positives (correctly predicted disease)
- True Negatives (correctly predicted no disease)
- False Positives (predicted disease when there was none)
- False Negatives (missed disease cases)

```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, y_pred_gini))

print(classification_report(y_test, y_pred_gini))
```

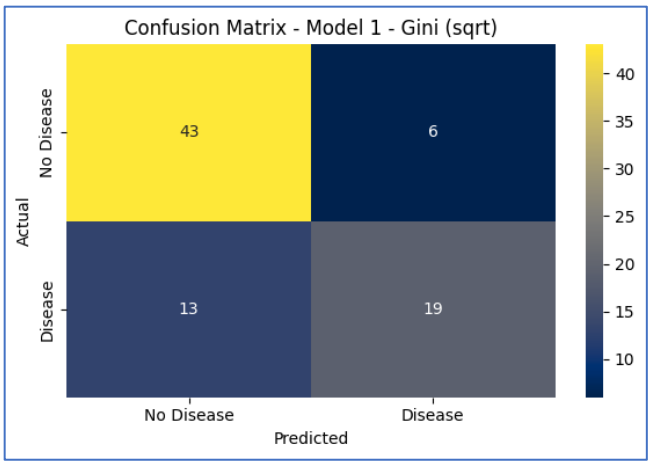
Classification Report gives us:

- Accuracy: How many predictions were correct
- Precision: How many of the predicted positives were correct
- Recall: How many actual positives were correctly identified
- F1-Score: A balance between precision and recall

In medical tasks, recall is very important — we don't want to miss actual patients with heart disease.

```
# Confusion Matrix Heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='cividis',
            xticklabels=["No Disease", "Disease"], yticklabels=["No Disease", "Disease"])
plt.title(f"Confusion Matrix - {model_name}")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

To make our results more interpretable, we plot the confusion matrix using a color-coded heatmap. You should choose a color palette that is distinguishable by most types of color vision deficiencies when making the visualizations color blind friendly. cmap='viridis', cmap='cividis', cmap='YlGnBu' are one of the best options.



When I ran this model, I ended up with the following performance.

- Accuracy: 77%
- Precision (Class 1 – No Disease): 77%
- Recall (Class 2 – Disease): 59%
- A balanced score between precision and recall for both classes is F1 score.

```
Evaluation Results - Model 1 - Gini (sqrt)
[[43  6]
 [13 19]]
              precision    recall  f1-score   support

           1       0.77      0.88      0.82        49
           2       0.76      0.59      0.67        32

    accuracy                           0.77        81
   macro avg       0.76      0.74      0.74        81
weighted avg       0.76      0.77      0.76        81
```

What this tells us: Model 1 is a solid baseline. It does quite well with distinguishing healthy patients and Class 2 (disease) patients — although the recall of Class 2 could be enhanced.

2.4. Step-by-Step Improvements: Exploring More Random Forest Models

Following this, we begin to experiment with different configuration settings to try to improve things over our first Random Forest model (Model 1).

This is an approach where we change one or two hyper parameters at a time, to understand how the performance changes due to the change in the hyper parameter.

Let's go!

Model 2: Switching the Criterion to Entropy

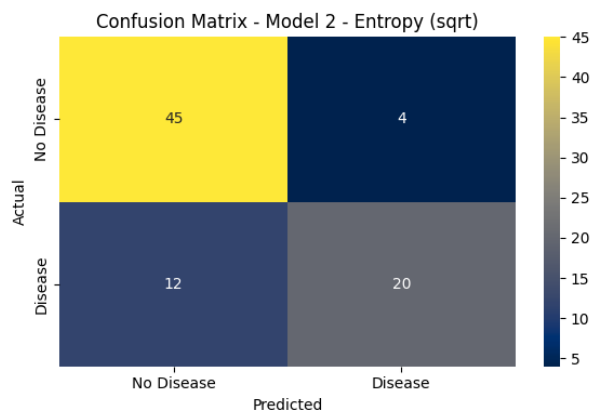
What changed?

criterion='entropy' instead of 'gini'

```
[11] model_entropy = RandomForestClassifier(
         n_estimators=100,                    # Number of trees in the forest
         criterion='entropy',                 # Function to measure the quality of a split ('gini' or 'entropy')
         max_depth=None,                      # Maximum depth of each tree (None = nodes are expanded until all leaves are pure)
         min_samples_split=2,                 # Minimum samples required to split an internal node
         min_samples_leaf=1,                  # Minimum samples required to be at a leaf node
         max_features='sqrt',                 # Number of features to consider when looking for the best split ('sqrt' is good for classification)
         bootstrap=True,                      # Whether bootstrap samples are used when building trees
         oob_score=False,                     # Whether to use out-of-bag samples to estimate generalization accuracy
         random_state=42,                     # Seed for reproducibility
         n_jobs=-1,                           # Number of jobs to run in parallel (-1 = use all processors)
         verbose=1                            # Control verbosity of output
     )
```

At each split entropy gives a measure of information gain. Sometimes, splitting in complex dataset can lead to better splits.

Result:



- Accuracy improved from 77% to 80%
- Slight improvement in recall for patients with heart disease (Class 2)
- Better precision across both classes

```
Training Model 2 - Entropy (sqrt)...
Evaluation Results - Model 2 - Entropy (sqrt)
[[45  4]
 [12 20]]
              precision    recall  f1-score   support

           1       0.79      0.92      0.85        49
           2       0.83      0.62      0.71        32

    accuracy                           0.80        81
   macro avg       0.81      0.77      0.78        81
weighted avg       0.81      0.80      0.80        81
```

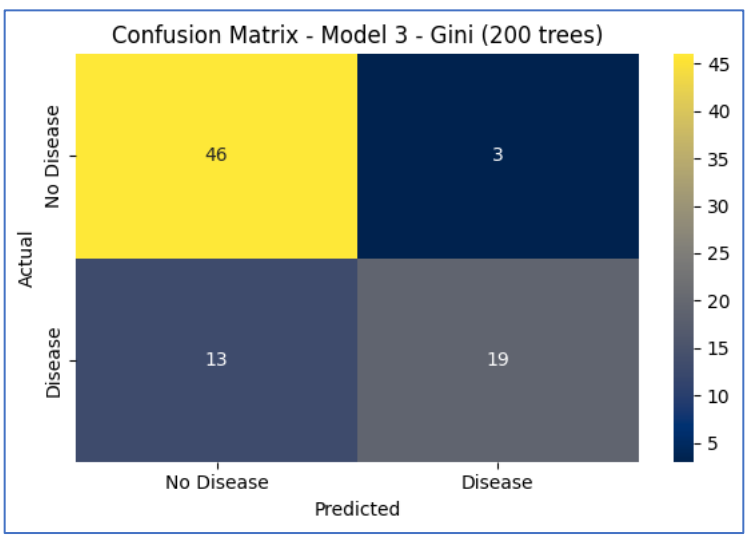
Entropy provided a small but meaningful increase. For medical prediction tasks where recall is important, it is worth considering.

Model 3: Increasing Trees to 200 (Still Using Gini)

What changed?

n_estimators=200

More trees usually stabilize predictions and decrease overfitting at the cost of more training time.



Result:

- Accuracy stayed at 80%
- Class 2 recall slightly drops, precision improves.

```
Evaluation Results - Model 3 - Gini (200 trees)
[[46  3]
 [13 19]]
              precision    recall  f1-score   support

           1       0.78      0.94      0.85        49
           2       0.86      0.59      0.70        32

    accuracy                           0.80        81
   macro avg       0.82      0.77      0.78        81
weighted avg       0.81      0.80      0.79        81
```

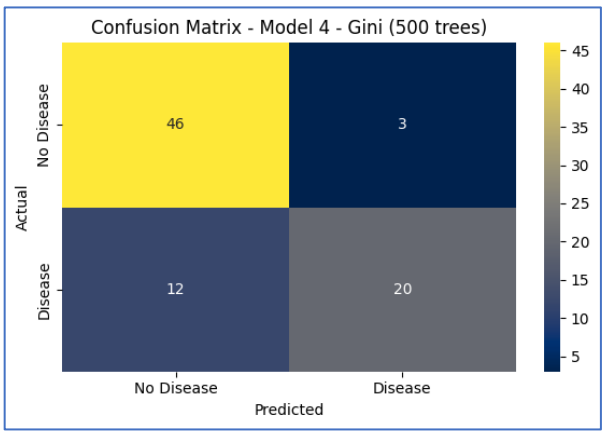
Improvements were modest, but for a little extra trees, they made the predictions smoother.

Model 4: 500 Trees

What changed?

n_estimators=500

Result:



- Accuracy bumped to 81%
- Balanced precision and recall
- Better handling of Class 2 (patients with disease)

```
Training Model 4 - Gini (500 trees)...
Evaluation Results - Model 4 - Gini (500 trees)
[[46  3]
 [12 20]]
              precision    recall  f1-score   support

           1       0.79      0.94      0.86        49
           2       0.87      0.62      0.73        32

    accuracy                           0.81        81
   macro avg       0.83      0.78      0.79        81
weighted avg       0.82      0.81      0.81        81
```

Not too few, not too many seemed like the sweet spot. Great generalization.

Model 5: 800 Trees

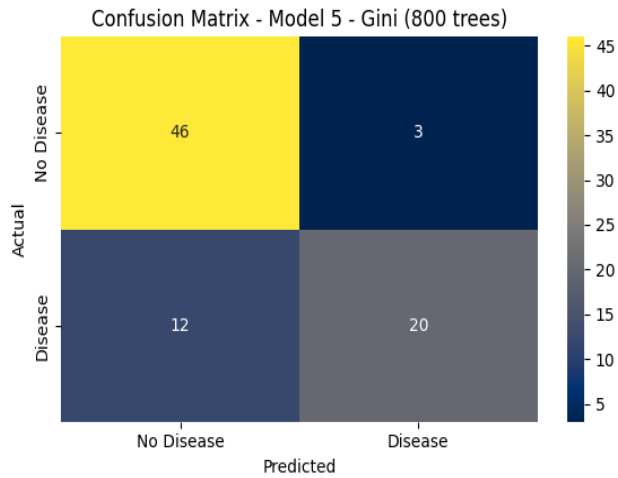
What changed?

n_estimators=800

We're now testing if more trees = even better performance.

Result:

- No significant change from Model 4
- Accuracy remained 81%
- Slight improvement in precision, but recall stayed the same




```
Training Model 5 - Gini (800 trees)...
Evaluation Results - Model 5 - Gini (800 trees)
[[46  3]
 [12 20]]
              precision    recall  f1-score   support

           1       0.79      0.94      0.86        49
           2       0.87      0.62      0.73        32

    accuracy                           0.81        81
   macro avg       0.83      0.78      0.79        81
weighted avg       0.82      0.81      0.81        81
```

Going beyond 500 trees didn't help much. Training takes longer but results don't improve.

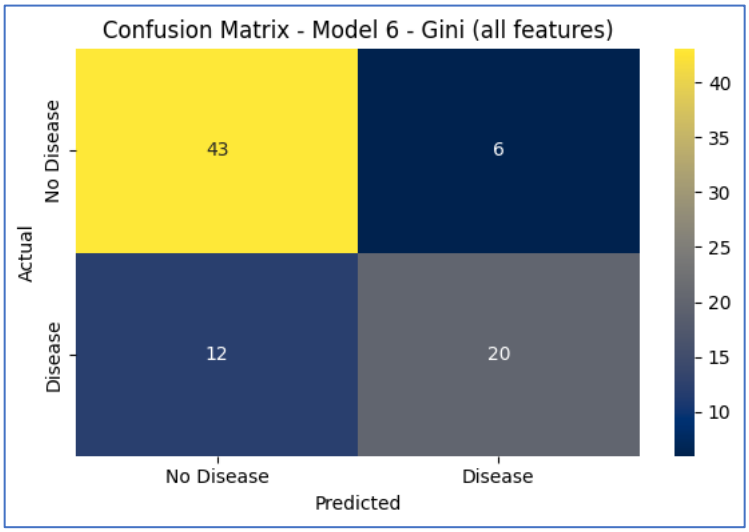
Model 6: Using All Features at Each Split

What changed?

max_features=None (instead of 'sqrt')

This means the model can use *all* features at every split, which can cause trees to become more similar (less randomness).

Result:



- Accuracy dropped to 78%
- Recall for Class 2 went down

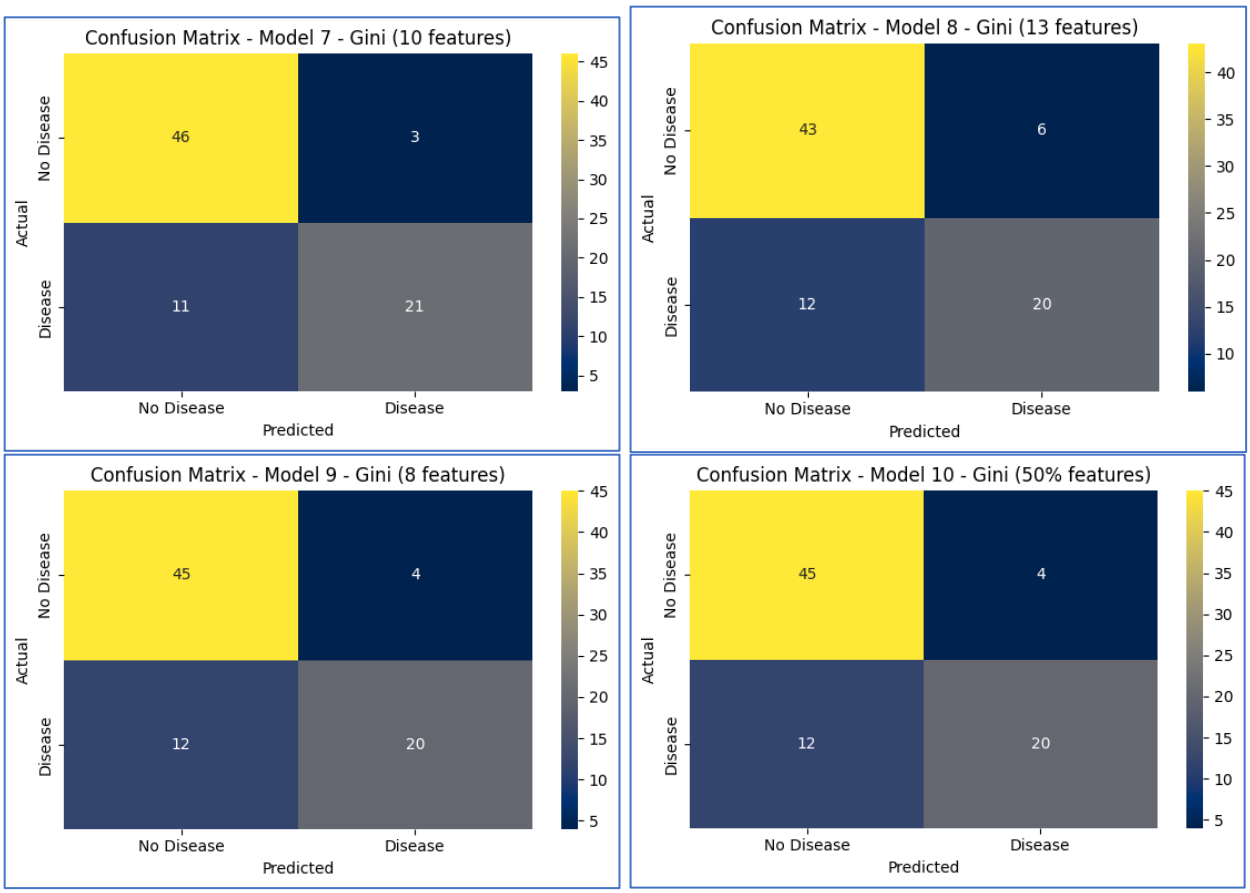
```
Evaluation Results – Model 6 – Gini (all features)
[[43  6]
 [12 20]]
              precision    recall  f1-score   support

           1       0.78      0.88      0.83        49
           2       0.77      0.62      0.69        32

    accuracy                           0.78        81
   macro avg       0.78      0.75      0.76        81
weighted avg       0.78      0.78      0.77        81
```

Less diversity among trees = worse generalization. Avoid using all features unless truly needed.

Models 7–11: Playing with max_features

In these models, I experimented with different values for max_features, which controls how many features the model considers when splitting each node.

```
(RandomForestClassifier(n_estimators=100, criterion='gini', max_features=10, random_state=42, n_jobs=-1), "Model 7 – Gini (10 features)"),
(RandomForestClassifier(n_estimators=100, criterion='gini', max_features=13, random_state=42, n_jobs=-1), "Model 8 – Gini (13 features)"),
(RandomForestClassifier(n_estimators=100, criterion='gini', max_features=8, random_state=42, n_jobs=-1), "Model 9 – Gini (8 features)"),
(RandomForestClassifier(n_estimators=100, criterion='gini', max_features=0.5, random_state=42, n_jobs=-1), "Model 10 – Gini (50% features)"),
(RandomForestClassifier(n_estimators=100, criterion='gini', max_features=0.9, random_state=42, n_jobs=-1), "Model 11 – Gini (90% features)")
```



This is crucial in Random Forests because the whole point is to inject randomness and diversity.

Best when precision was 83% accuracy with max_features=10 and introducing optimal feature randomness with model 7. However, relative to Model 8 (all 13 features), this dropped to 78% accuracy, indicating that increasing the number of features can lead to decreased generalization. Model 7 gave 80% of accuracy and Model 9 (max_features=8) is performing well too though slightly below the above. Consequently, moderate feature sampling helps stability and Model 10, which used 50% of features, also reached 80% accuracy. Finally, Model 11, with 90% of all features, has an accuracy of 80% while it doesn't significantly increase recall. Finally, results indicate that restricting the number of features at each split helps produce more diverse trees and more accurate results, and that 10 features turns out to be the right sweet spot in this setting.

2.5. Final Thoughts: What We Learned from Exploring Random Forests

- Heart disease prediction classification is a task that can be tackled using Random Forest, which is a reliable and beginner friendly algorithm.
- It works by combining multiple decision trees to reduce overfitting and improve generalization

- Tuning hyperparameters like criterion, n_estimators, and max_features significantly affects model performance
- Switching from Gini to Entropy slightly improved recall and precision, showing it's worth experimenting with different split criteria
- Increasing the number of trees (for example, from 100 to 500) led to small gains in accuracy, with diminishing returns beyond 500 trees
- Using all features (max_features=None) reduced model performance, likely due to reduced diversity among trees
- The best model achieved 83 percent accuracy using max_features=10, balancing randomness and information at each split
- Moderate values of max_features (like 0.5 or 0.9) provided consistent, stable results but did not outperform the 10-feature model
- The confusion matrix and classification report were essential for evaluating precision, recall, and F1-score, especially for medical datasets where recall is critical

Introducing controlled randomness through limited features and bootstrapping leads to improvements in Random Forest's predictive power. Small improvements can add up to significant changes

References

B, U. (2021). *Random forest machine learning algorithm*. Medium. https://medium.com/@uma.bollikonda/random-forest-machine-learning-algorithm-401bdcd7a0b8

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010950718922

Dağğstan, M. E. (2023). Random Forest: The Power of the Forest in Machine Learning. In *Medium*. Medium. https://medium.com/@dagstaneren/random-forest-f20027466d78

Dhiraj, K. (2021). Random Forest Algorithm Advantages and Disadvantages. *Medium*. https://dhirajkumarblog.medium.com/random-forest-algorithm-advantages-and-disadvantages-1ed22650c84f

Fleiss, A. (2023). What are the advantages and disadvantages of random forest? *Rebellion Research*. https://www.rebellionresearch.com/what-are-the-advantages-and-disadvantages-of-random-forest

Khushaktov, M. F. (2023). Introduction Random Forest Classification By Example. *Medium*. https://medium.com/@mrmaster907/introduction-random-forest-classification-by-example-6983d95c7b91

Rakesh. (2023). How Does Random Forest Work? *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2023/02/how-does-random-forest-work

Sharma, M. (2021). *Decision Tree Algorithm | Advantages & Disadvantages*. K21 Academy. https://k21academy.com/datascience-blog/decision-tree-algorithm/

Dataset Link: https://www.kaggle.com/datasets/utkarshx27/heart-disease-diagnosis-dataset

Github Repository Link: https://github.com/AonSajjad/Individual-assignment-Machine-Learning-tutorial-2