Machine Learning

GitHub: https://github.com/Sharjeel-Analyst/Machine_Learning[i]

Name: Muhammad Sharjeel
Student_ID: 23086732

Understanding of Random Forest Algorithm with Examples

What is Random Forest Algorithm?

Random Forest is a widely-used machine learning algorithm developed by Leo Breiman and Adele Cutler that is used in regression and classification problems and produces, even without hyperparameter tuning a great result most of the time. It is perhaps the most used algorithm because of its simplicity.
It is a powerful tree learning technique in Machine Learning to make predictions and then we do vote of all the tress to make prediction.
It is a type of classifier that uses many decision trees to make predictions.
It takes different random parts of the dataset to train each tree and then it combines the results by averaging them. This approach helps improve the accuracy of predictions. Random Forest is based on ensemble learning.

> ***"Ensemble learning combines the predictions of multiple models (called "weak learners" or "base models") to make a stronger, more reliable prediction. The goal is to reduce errors and improve performance."***

Types of Ensemble Learning in Machine Learning:

1. Bagging: Models are trained independently on different subsets of the data, and them results are averaged or voted on. This is also called Bootstrap Aggregating.
2. Boosting: Models are trained sequentially, with each one learning from the mistakes of the previous model.

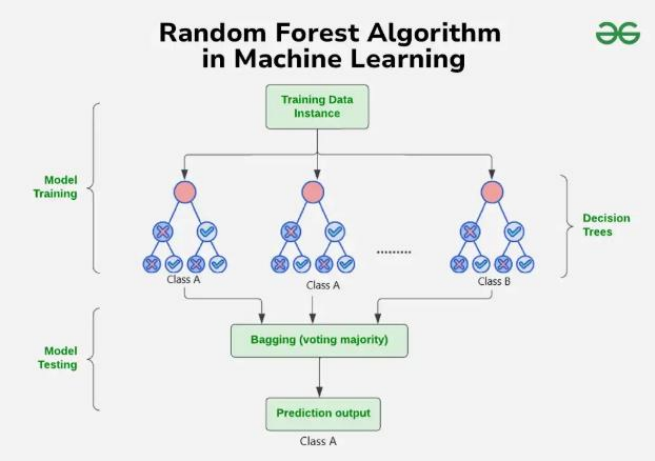


Fig1.0: Random Forest Algorithm Intro

As explained in image: Process starts with a dataset with rows and their corresponding class labels (columns).
Then - Multiple Decision Trees are created from the training data. Each tree is trained on a random subset of the data (with replacement) and a random subset of features. This process is known as bagging or bootstrap aggregating.

- Each Decision Tree in the ensemble learns to make predictions independently.
- When presented with a new, unseen instance, each Decision Tree in the ensemble makes a prediction.
- The final prediction is made by combining the predictions of all the Decision Trees. This is typically done through a majority vote (for classification) or averaging (for regression).

Daily life Example:

If someone wants to play golf (Yes/NO) is the tree. If yes then its further nodes come that includes different parameters which he thinks should he have for playing.
like how is the weather (sunny or not). If weathers are sunny then has he got golf stick and gloves.
By aggregating these predictions from decision trees is simply a random forest algorithm in daily life.

Learning Objectives:

In this tutorial, we will explore the Random Forest model, a powerful machine learning technique.

- Learn the working of random forest with an example by using python coding.
- Understand the impact of different hyperparameters.
- Implement them on a classification problem using scikit-learn.

Overview of Decision Trees:

Decision trees are a simple machine learning tool used for classification and regression tasks. They break complex decisions into smaller steps, making them easy to understand and implement.

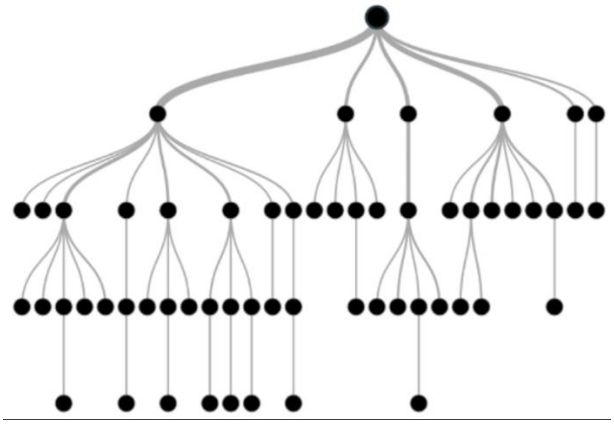


Fig 1.1: Decision Tree

Decision Tree Terminologies:

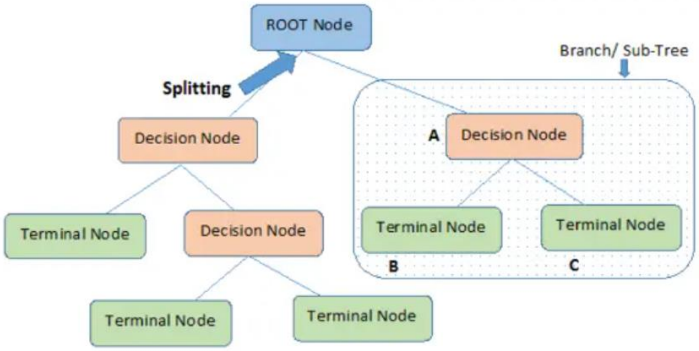


Fig 1.2: Decision Tree Specifications

Root Node: The initial node at the beginning of a decision tree, where the entire population or dataset starts dividing based on various features or conditions.
Decision Nodes: Nodes resulting from the splitting of root nodes are known as decision nodes.
Terminal or leaf Nodes: Nodes where further splitting is not possible, often indicating the final classification or outcome. Leaf nodes are also referred to as terminal nodes.
Sub-Tree: Similar to a subsection of a graph being called a sub-graph, a sub-section of these tree is referred to as a sub-tree. It represents a specific portion of the decision tree.
Branch / Sub-Tree: A subsection of the entire is referred to as a branch or sub-tree.
Parent and Child Node: In a decision tree, a node that is divided into sub-nodes is known as a parent node, and the sub-nodes emerging from it are referred to as child nodes.

In Random Forest Algorithm many decision trees are used to get best possible outcome.

Assumptions of Random Forest:

To effectively use Random Forest, it is important to understand the underlying assumptions of the algorithm.

- **Independence of Trees**: The decision trees in the forest should be independent of each other. We achieve this through bootstrap sampling and feature randomness.
- **Sufficient Data**: Random Forest requires a large amount of data to build diverse trees and achieve optimal performance.
- **Balanced Trees**: The algorithm assumes that individual trees grow sufficiently deep to capture the underlying patterns in the data.
- **Noisy Data Handling**: Random Forest can handle noisy data, but it assumes that the noise randomly distributes and is not systematic.

Random Forest Applications:

- Customer churn prediction
- Fraud detection
- Stock price prediction
- Medical diagnosis

In this tutorial we will work of medical diagnosis (Heart Attack Risk) by implementing all steps of Random Forest Algorithm.

Steps Involved in Random Forest Algorithm:

- **Step 1:** In this model, we select a subset of data points and a subset of features to construct each decision tree. Simply put, we take n random records and m features from a dataset containing k records.
- **Step 2:** We construct individual decision trees for each sample.
- **Step 3:** Each decision tree will generate an output.
- **Step 4:** We consider the final output based on majority voting for classification and averaging for regression, respectively.

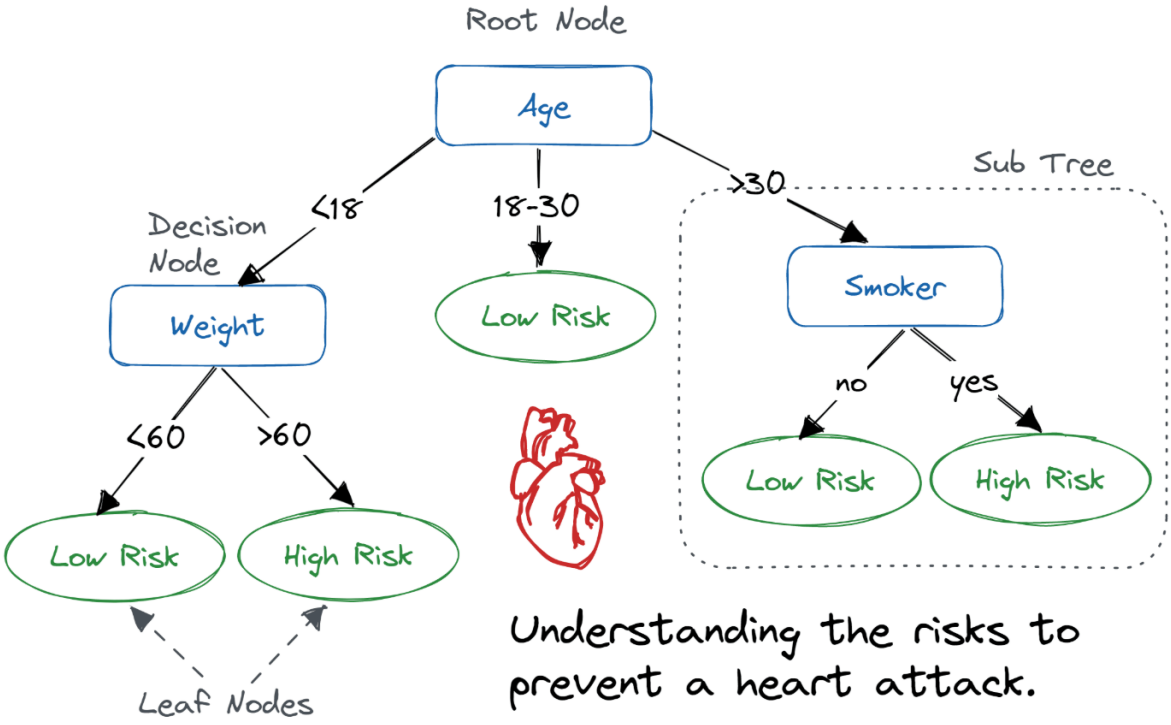


Fig 1.3: Decision Tree of heart risk by age

Key Features of Random Forest:

Random Forest stands out as a powerful and flexible machine learning technique.

- Diverse Decision Trees: Instead of relying on a single model, Random Forest generates multiple decision trees, each using different subsets of data and features. This diversity helps reduce overfitting and enhances the model's ability to generalize to new data.
- Robust Predictions: By combining the outcomes from multiple trees, Random Forest delivers predictions that are more stable and less prone to errors caused by variance.
- Handling Missing Data: Random Forest manages these seamlessly, either by using surrogate splits or by averaging results from trees unaffected by missing data for the same inputs.
- Feature Importance Insights: One valuable aspect of Random Forest is its ability to show how influential each feature is in making predictions. This makes it an excellent tool for understanding your data better and selecting the most relevant features.
- Scalable to Big Data: The algorithm is well-suited for large and complex datasets. Since each tree is constructed independently, Random Forest can be parallelized, making it efficient for tasks involving high-dimensional data.
- Versatility in Applications: Whether you're working on classification or regression tasks, Random Forest can handle both with ease. It's also adept at working with a mix of categorical and numerical data.
- Ensemble Stability: Unlike a single decision tree, Random Forest is less sensitive to changes in the training data, making it a reliable option for consistent predictions.
- Built-In Error Estimation: A handy feature of Random Forest is its ability to estimate errors internally. Using out-of-bag (OOB) samples—data points left out when building each tree—it provides a quick and effective error assessment without requiring a separate validation set.

Important Hyperparameters in Random Forest:

Random Forest uses hyperparameters to enhance its accuracy or speed.

1. **For Better Predictions:**

- n_estimators: Sets the number of trees in the forest. More trees can improve accuracy but take longer to train.
- max_features: Limits the number of features considered when splitting a node, influencing tree diversity and accuracy
- min_samples_leaf: Determines the minimum number of samples required to form a leaf node, preventing overfitting
- criterion: Defines how to split nodes in each tree (e.g., Gini impurity, Entropy, or Log Loss).
- max_leaf_nodes: Caps the number of leaf nodes, simplifying the trees.

2. **For Faster Performance:**

- n_jobs: Controls the number of processors used. Set to 1 use one processor while -1 use all available for quicker training.
- random_state: Ensures consistent results by fixing the randomness for reproducibility.
- oob_score: Uses "out-of-bag" samples (data not used during training) to estimate model performance without needing a separate validation set.

Dataset Summary

This dataset contains **8,763** records with **26 attributes**, primarily focused on cardiovascular health factors. The target variable for classification is **"Heart Attack Risk"**, which is binary (0 = No Risk, 1 = High Risk).

1. **Demographics:**
    - **Patient ID:** Unique identifier (not useful for modeling).
    - **Age:** Integer representing patient's age.
    - **Sex**: Male/Female.
2. **Health Indicators:**
    - **Cholesterol:** Numeric value indicating cholesterol level.
    - **Blood Pressure:** Categorical string (e.g., "158/88"), may need transformation.
    - **Heart Rate:** Numeric, beats per minute.
    - **Diabetes:** Binary (0 = No, 1 = Yes).
    - **Family History:** Binary indicator.
    - **Obesity:** Binary (0 = No, 1 = Yes).
    - **BMI**: Continuous numeric value.
3. **Lifestyle Factors:**
    - **Smoking, Alcohol Consumption:** Binary (0 = No, 1 = Yes).
    - **Exercise Hours Per Week:** Continuous numeric value.
    - **Physical Activity Days Per Week:** Integer.
    - **Sedentary Hours Per Day:** Continuous numeric value.
    - **Sleep Hours Per Day:** Integer.
    - **Diet:** Categorical (needs encoding).
4. **Medical & Social Factors:**
    - Previous Heart Problems, Medication Use, Stress Level (all integer values).
    - **Triglycerides:** Numeric.
    - **Income**: Numeric value.
    - Country, Continent, Hemisphere: Categorical (could be used for regional insights).
5. **Target Variable:**
    - **Heart Attack Risk:** Binary classification (0 = No, 1 = Yes).

Why Use Random Forest for Heart Attack Prediction?

- Handles High-Dimensional Data: The dataset has 25+ features, and Random Forest is good at selecting the most important ones.
- Reduces Overfitting: Unlike individual decision trees, it generalizes well to new data.
- Feature Importance Ranking: It helps identify the most critical risk factors (e.g., Cholesterol, Smoking, Family History).
- Works Well with Missing Data: It can handle missing values by averaging results from multiple trees.

Feature Importance in Heart Attack Prediction:

Random Forest assigns importance scores to each feature. Likely top predictors of heart attack risk from this dataset:

- Cholesterol Levels
- Blood Pressure (Systolic/Diastolic)
- BMI (Obesity Indicator)
- Smoking & Alcohol Consumption
- Family History of Heart Disease
- Previous Heart Problems

Coding in Python

1. Building of Random Forest Classifier:

```
# instantiating the random forest classifier
rf = RandomForestClassifier()
```

```
# fitting the model on the training set
rf.fit(X_train, y_train)
```

```
# making predictions on the testing set
y_pred = rf.predict(X_test)
```

```
# checking the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f'The accuracy of the model is: {accuracy}')
```

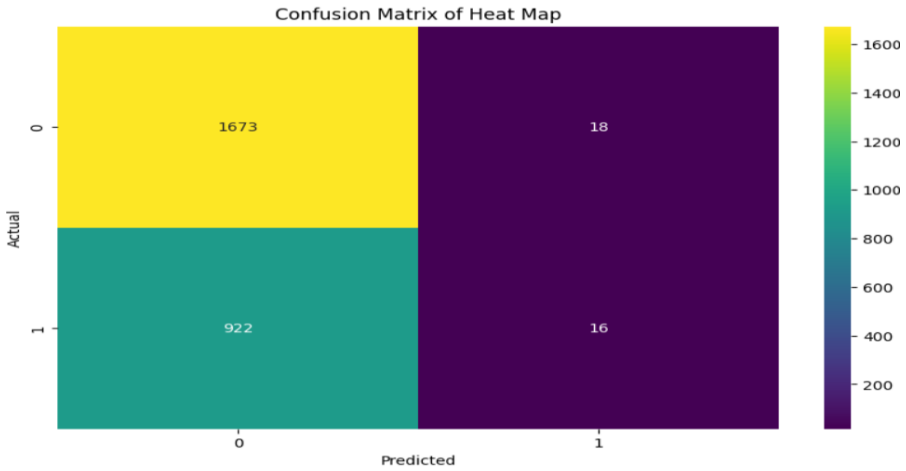
```
The accuracy of the model is: 0.6424496006085965
```

```
# checking the classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.64      0.99      0.78      1691
           1       0.47      0.02      0.03       938

    accuracy                           0.64      2629
   macro avg       0.56      0.50      0.41      2629
weighted avg       0.58      0.64      0.51      2629
```

2. Confusion Matrix:

```
# checking the confusion matrix
confusion_matrix(y_test, y_pred)
# plotting the confusion matrix
plt.figure(figsize=(10, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='viridis')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix of Heat Map')
plt.show()
```



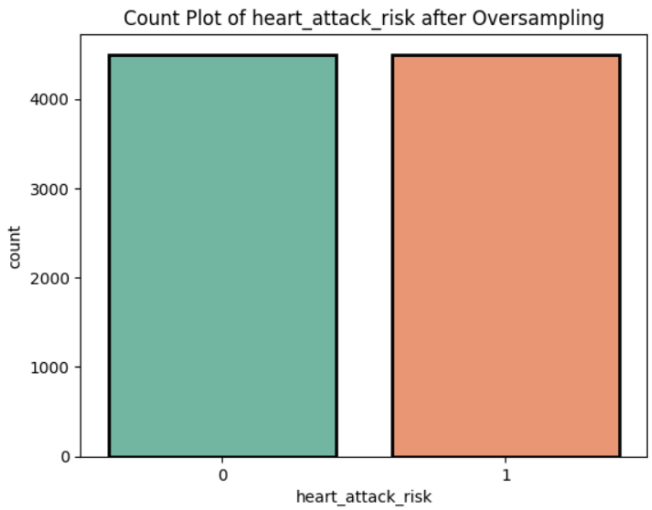
- confusion matrix shows the number of correct and incorrect predictions for each class.
- it tells us the number of true positives, true negatives, false positives, and false negatives.
- from the above results it is clear that model is not performing well on the testing set. As due to class imbalance it misclassifies the data.
- As this model identifies 920 records which is a larger number

3. **Applying SMOTE to deal with class imbalance:**

```
# Define features and target
X_resample = df.drop('heart_attack_risk', axis=1)
y_resample = df['heart_attack_risk']

# Ensure all columns are float64
X_resample = X_resample.astype(float)

# Split data
X_train_resample, X_test_resample, y_train_resample, y_test_resample = train_test_split(
    X_resample, y_resample, test_size=0.2, random_state=42
)

# Apply SMOTE
smote = SMOTE(random_state=42)
X_train_resample, y_train_resample = smote.fit_resample(X_train_resample, y_train_resample)
```

```
# plotting the count plot
sns.countplot(x=y_train_resample , palette='Set2' , edgecolor='black' , linewidth=2 )
plt.title('Count Plot of heart_attack_risk after Oversampling')
plt.show()
```

4. **After SMOTE:**



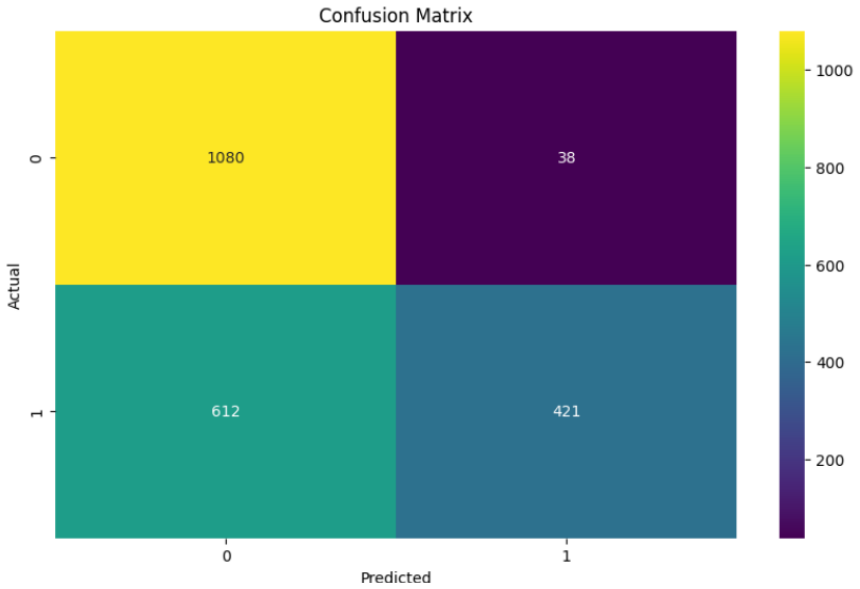
5. After SMOTE again Random Forest:

```
# Splitting the resampled dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Initializing the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Training the model
rf_classifier.fit(X_train, y_train)

# Making predictions
y_pred = rf_classifier.predict(X_test)
```

```
# Evaluating the model
print(f'Accuracy Score:{accuracy_score(y_test, y_pred)}')

(confusion_matrix(y_test, y_pred))
class_report = classification_report(y_test, y_pred)
```

```
Accuracy Score:0.6978149697814969
```

```
# plotting the confusion matrix
plt.figure(figsize=(10, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='viridis')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

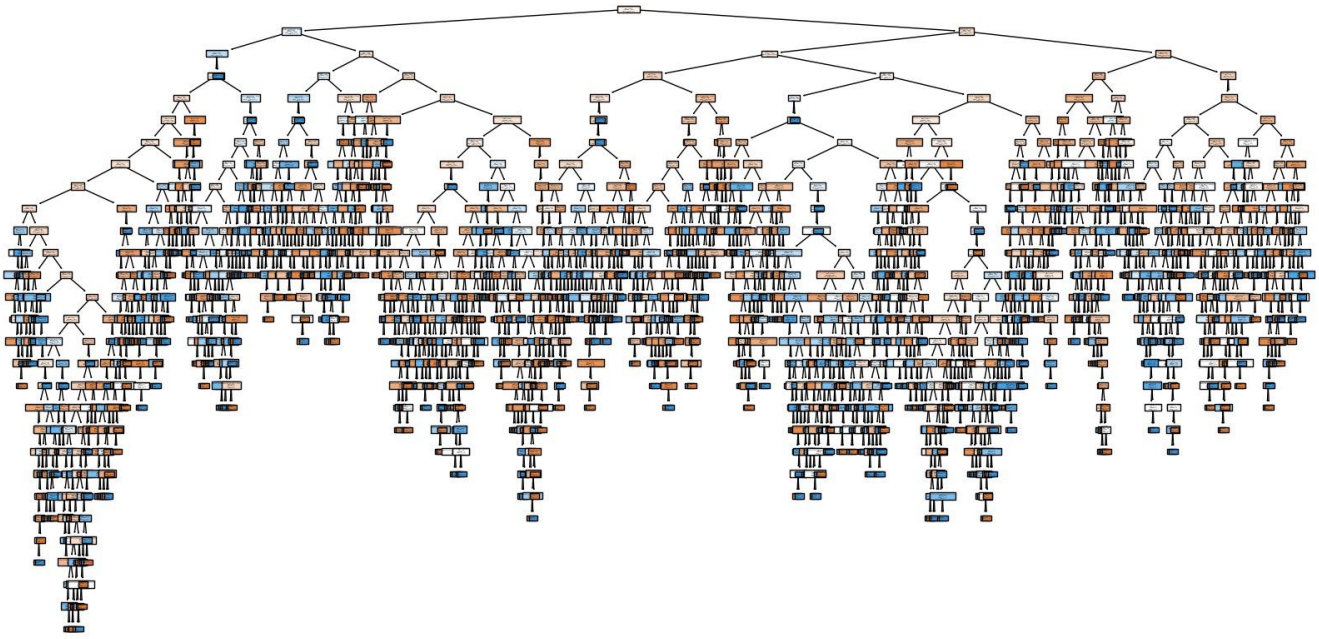


6. Hyper Parameter Tunning:

```
# calling the random forest and adjusting the parameters
model = RandomForestClassifier(
    n_estimators=150,      # Number of trees
    max_depth=10,          # Maximum depth of each tree
    min_samples_split=10,  # Minimum samples required to split an internal node
    min_samples_leaf=5,    # Minimum samples required to be at a leaf node
    max_features=5,        # Number of features to consider when looking for the best split
    criterion='entropy',   # Splitting criterion
    random_state=42
)

# fitting the model
model.fit(X_train, y_train)

# calculating the train and test accuracy
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# printing the train and test accuracy
print(f"Train accuracy: {accuracy_score(y_train, y_pred_train)}")
print(f"Test accuracy: {accuracy_score(y_test, y_pred_test)}")
```

```
Train accuracy: 0.7296511627906976
Test accuracy: 0.694560669456067
```

7. **Let's Visualize:**



Key Outcomes:

- Works well for classification and regression tasks
- Handles missing data effectively
- Reduces overfitting through majority voting or averaging
- Executes in parallel due to independent decision trees.
- Offers stability by averaging multiple tree outputs
- Resistant to the curse of dimensionality by reducing feature space.
- Automatically separates part of the data for validation during training.
- Complex compared to decision trees, making interpretation harder.
- Longer training time due to multiple tree evaluations.

Conclusion:

Random forest is a versatile, high-performing machine-learning algorithm. It efficiently handles various data types (binary, continuous, categorical), and is especially valuable when speed and robustness are required. Despite its limitations, it remains a flexible and widely used tool.

---

[i] **References:**

https://www.youtube.com/watch?v=gkXX4h3qYm4&t=189s
https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-random-forest-algorithm-for-beginners/
https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/
https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/
https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/?ref=ml_lbp
https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/
https://www.geeksforgeeks.org/a-comprehensive-guide-to-ensemble-learning/