# Random Forest Models

A Comprehensive Guide 🤓

Piyush Hole

# Introduction

## Overview

Brief introduction to machine learning and the importance of Random Forest models.

## Objective

Provide a detailed guide on building and optimizing a Random Forest model.

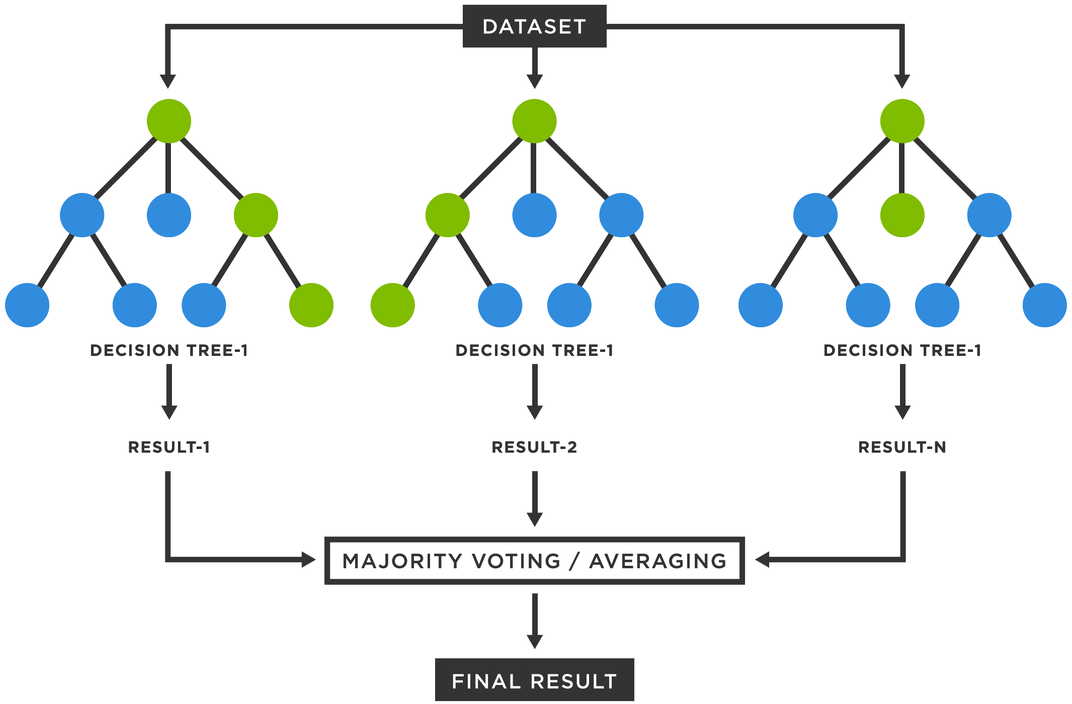
# What is a Random Forest?

**Definition**: An ensemble learning method that combines multiple decision trees.

- ensemble → multiple weak models combine into a stronger one
- averages the predictions of individual trees to reduce variance and improve accuracy

**Advantages**:

- Reduces overfitting by averaging multiple trees
  - overfitting → learns the training data too well and thus performs poor on new data
- Provides feature importance
- Robust to noise and outliers



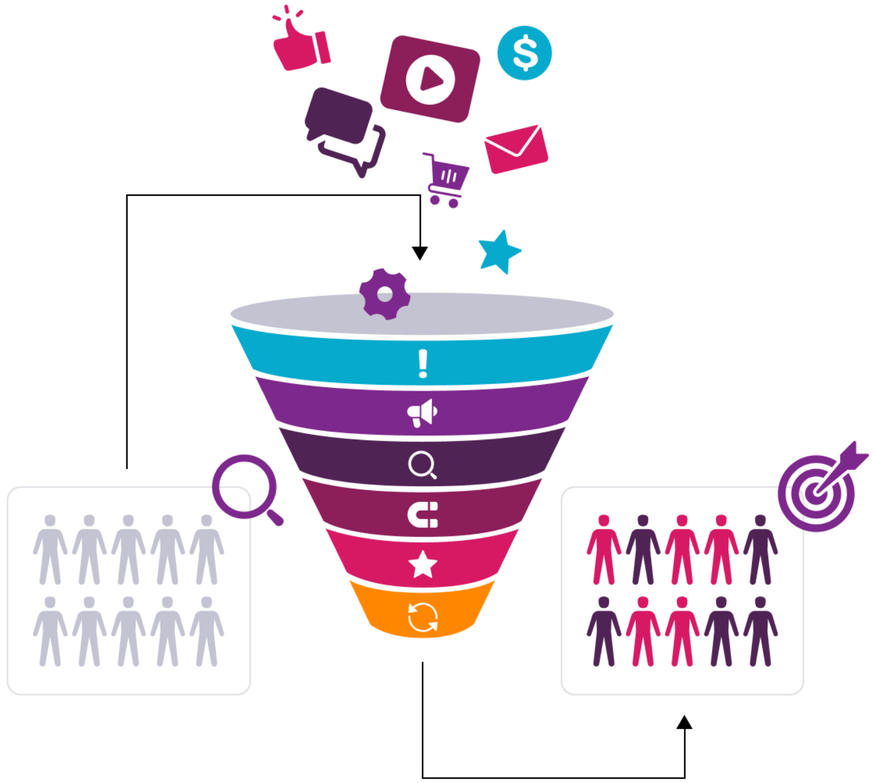
# Step 1

Preparing the Data

# Data Collection

- **Sources**: Data can be collected from various sources such as databases, APIs, web scraping, or even manual data entry.
  - The quality and relevance of your data significantly impact the model's performance.

- **Data Understanding**: Before cleaning, it's crucial to understand the data.
  - This involves exploring the dataset to identify patterns, trends, and potential issues.



# Data Cleaning

- **Handling Missing Values:**
  - Imputation: Filling missing values with statistical measures like mean, median, or mode.
  - Removal: Discarding rows or columns with missing values, especially if they constitute a small portion of the dataset.

- **Removing Duplicates**: Identifying and removing duplicate entries to avoid bias in the model.


# Data Preprocessing

- **Normalization/Standardization:**
  - Normalization: Scaling features to a range of [0, 1] using techniques like Min-Max scaling.
  - Standardization: Transforming features to have a mean of 0 and a standard deviation of 1, using Z-score normalization.

- **Encoding Categorical Variables:**
  - One-Hot Encoding: Converting categorical variables into binary vectors.
  - Label Encoding: Assigning a unique integer to each category.

- **Feature Engineering**: Creating new features from existing ones to better represent the underlying patterns in the data.

- **Splitting Data**: Dividing the dataset into training and testing sets to evaluate the model's performance on unseen data. Common splits include 80/20 or 70/30.

# Step 2

Train the Model

# Scikit-Learn

Scikit-learn is a popular machine learning library in Python that provides simple and efficient tools for data mining and data analysis.

**Initialization:**

- Define the model structure using RandomForestClassifier or RandomForestRegressor from scikit-learn.
- Key parameters include n_estimators (number of trees) and max_depth (maximum depth of each tree).

**Training Process:**

- Fitting the Model: Use the fit method to train the model on the training data.
- Bootstrapping: Scikit-learn handles the bootstrapping process internally, creating diverse subsets of the data for each tree.

Initialization

Python

```python
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest Classifier model
# n_estimators:  Number of trees in the forest
# max_depth:    Maximum depth of a tree
# random_state:  Controls the randomness of the trees
rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)

# Train (fit) the model using the training data
# X_train:       Features of the training data.
# y_train:       Target labels of the training data.
rf.fit(X_train, y_train)
```

# More Complex Models

**TensorFlow and Keras:**

- TensorFlow: An open-source library for numerical computation and machine learning, developed by Google.
- Keras: A high-level API for building and training deep learning models, built on top of TensorFlow.

**PyTorch:**

- An open-source machine learning library developed by Facebook's AI Research lab, known for its dynamic computation graph and ease of use.



Neural Network — Python

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Initialize the model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=10)
```

# Step 3

Evaluate the Model

# Confusion Matrix

A table that summarizes the performance of a classification model, showing true vs. predicted classifications.

**Metrics:**

- Accuracy: Measures the overall correctness of the model but can be misleading if the classes are imbalanced.
- Precision: Focuses on the accuracy of positive predictions. High precision indicates a low false positive rate.
- Recall (Sensitivity): Measures the model's ability to identify all relevant instances. High recall indicates a low false negative rate.
- F1 Score: Provides a single metric that balances precision and recall, useful when there is an uneven class distribution.
    - F1 = 2 * (precision * recall) / (precision + recall)

| | Predicted Positive | Predicted Negative | |
|---|---|---|---|
| Actual Positive | TP<br>*True Positive* | FN<br>*False Negative* | Sensitivity<br>$\frac{TP}{(TP+FN)}$ |
| Actual Negative | FP<br>*False Positive* | TN<br>*True Negative* | Specificity<br>$\frac{TN}{(TN+FP)}$ |
| | Precision<br>$\frac{TP}{(TP+FP)}$ | Negative Predictive Value<br>$\frac{TN}{(TN+FN)}$ | Accuracy<br>$\frac{TP+TN}{(TP+TN+FP+FN)}$ |

# Step 4

Optimize Hyperparameters

# Hyperparameter Tuning

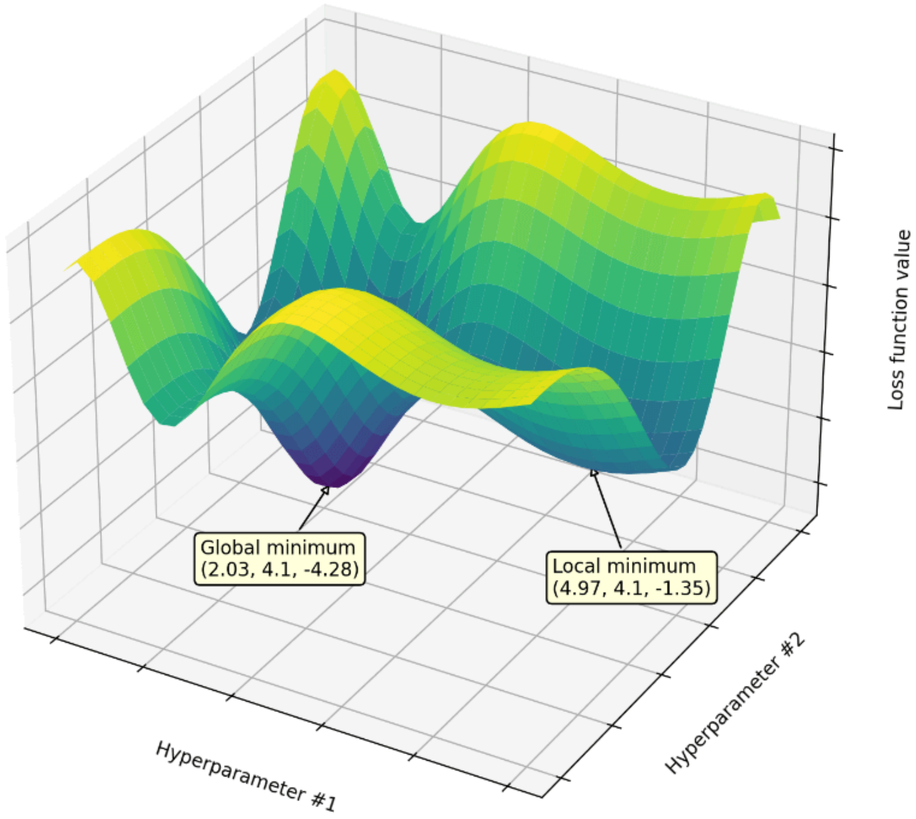
Variables that are set before the training process and control aspects of the learning algorithm

**GridSearchCV:**

- Performs an exhaustive search over a specified parameter grid.
- Computationally expensive but ensures the best combination of hyperparameters is found.

**RandomizedSearchCV:**

- Samples a fixed number of parameter settings from specified distributions.
- Faster than GridSearchCV but may not find the optimal combination.



# Important Hyperparameter

- **n_estimators:** Number of trees in the forest. More trees can improve performance but increase computational cost.
- **max_depth:** Maximum depth of each tree. Controls the complexity of the model.
- **min_samples_split:** Minimum number of samples required to split an internal node. Prevents overfitting by stopping the tree from splitting too much.
- **min_samples_leaf:** Minimum number of samples required to be at a leaf node. Helps in creating more generalized trees.
- **max_features:** Number of features to consider when looking for the best split. Can be a fraction or an integer.

# Cross-Validation

Used to assess performance on unseen data by dividing the dataset into multiple folds.

**K-Fold Cross-Validation:**

- The dataset is divided into k equally sized folds.
- The model is trained k times, each time using k−1 folds for training and the remaining fold for validation.
- The results are averaged to produce a single estimation of model performance.
- Common choices for k are 5 or 10.

$$cv_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

**Mean Squared Error (MSE):** MSEi represents the error on the i-th fold.

**Summation ∑:** Adds the MSEs from all k folds.

**Averaging 1/k:** Takes the mean of all MSE values across folds to get an overall estimate of model performance.