**RANDOM FOREST.**

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML.

It is based on the concept of **ensemble learning**, which is a **process of combining multiple classifiers to solve a complex problem and to improve the performance of the model**.

Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

The greater the number of trees in a Random Forest Algorithm, the higher its accuracy and problem-solving ability.

Random Forest is a classifier that contains several decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.

It contains many decision trees representing a distinct instance of the classification of data input into the random forest. The random forest technique considers the instances individually, taking the one with the majority of votes as the selected prediction.

Random forest algorithms have three main **hyperparameters**, which need to be set before training. Hyperparameters are those parameters that are explicitly defined by the user to control the learning process. The value of the Hyperparameter is selected and set by the machine learning engineer before the learning algorithm begins training the model. Hence, these are external to the model, and their values cannot be changed during the training process. Examples of hyperparameters include:

1. The k in kNN or K-Nearest Neighbour algorithm
2. Learning rate for training a neural network
3. Train-test split ratio
4. Batch Size
5. Number of Epochs
6. Branches in Decision Tree
7. Number of clusters in Clustering Algorithm

RANDOM FOREST HYPERPARAMETERS.

1. n\_estimators = number of trees in the forest.
2. max\_features = max number of features considered for splitting a node.
3. max\_depth = max number of levels in each decision tree.
4. min\_samples\_split = min number of data points placed in a node before the node is split.
5. min\_samples\_leaf = min number of data points allowed in a leaf node.
6. bootstrap = method for sampling data points (with or without replacement).

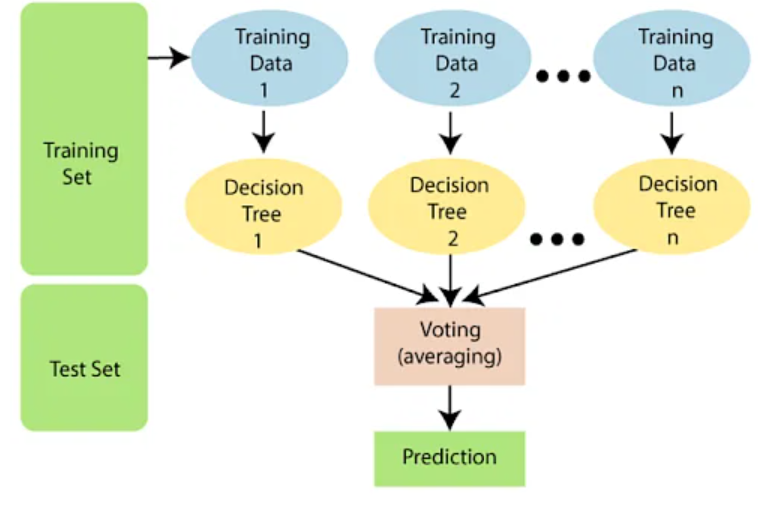
The following steps explain the working Random Forest Algorithm:

Step 1: Select random samples from a given data or training set.

Step 2: This algorithm will construct a decision tree for every training data.

Step 3: Voting will take place by averaging the decision tree.

Step 4: Finally, select the most voted prediction result as the final prediction result.



**EXAMPLE CODE.**

The code below is for diabetes prediction and the associated dataset is attached with this description.

1. **Importing Libraries.**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

These lines import necessary Python libraries:

- `**pandas**` for data manipulation and analysis.

- `**numpy**` for numerical operations.

- `**matplotlib.pyplot**` for data visualization.

- `**seaborn**` for statistical data visualization.

2. **Importing Dataset.**

df = pd.read\_csv('diabetes.csv')

df.head()

- `**pd.read\_csv('diabetes.csv')**` reads a CSV file named 'diabetes.csv' into a Pandas DataFrame called `df`.

- `**df.head()**` displays the first few rows of the DataFrame.

3. **Checking Data Types.**

df.dtypes

- `**df.dtypes**` displays the data types of each column in the DataFrame.

4. **Checking Data Information.**

df.info()

- `**df.info()**` provides a concise summary of the DataFrame, including data types and non-null values.

5. **Handling Missing Values.**

df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)

- Replaces 0 values in specific columns (`Glucose`, `BloodPressure`, `SkinThickness`, `Insulin`, `BMI`) with NaN (missing values).

6. **Displaying Missing Values Summary.**

for i in df.columns:

print(f"{i: <50}{df[i].isnull().sum():<30}{((df[i].isnull().sum())\*100)/df.shape[0]: .2f}")

- Prints a summary of missing values for each column, including the total number and percentage of missing values.

7. **Handling Missing Values – Imputation.**

df['Glucose'].fillna(df['Glucose'].mean(), inplace=True)

- Fills missing values with the mean of each respective column after replacing 0 values with NaN.

8. **Displaying Missing Values Summary after Imputation.**

for i in df.columns:

print(f"{i: <50}{df[i].isnull().sum():<30}{((df[i].isnull().sum())\*100)/df.shape[0]: .2f}")

- Prints an updated summary of missing values after imputation.

9. **Visualizing Correlation Matrix.**

plt.figure(figsize=(8, 6), dpi=80)

sns.heatmap(df.corr())

plt.show()

- Plots a heatmap of the correlation matrix to visualize relationships between numerical features.

10. **Visualizing Pair Plot.**

sns.pairplot(df,hue='Outcome')

- Creates a pair plot to visualize relationships between pairs of features, colored by the 'Outcome' variable.

11. **Visualizing Boxplot.**

plt.figure(figsize=(18, 6), dpi=80)

sns.boxplot(data=df, orient="h", palette="Set1")

plt.show()

- Displays a boxplot for each feature, horizontally oriented.

12. **Visualizing Outcome Count.**

sns.countplot(x="Outcome", data=df)

- Plots a bar chart showing the count of each class in the 'Outcome' variable.

13. **Data Splitting.**

X=df.drop('Outcome',axis=1)

y=df['Outcome']

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,test\_size=0.2,random\_state=0)

- Splits the data into features (`X`) and target variable (`y`). Then, further splits it into training and testing sets using `train\_test\_split`.

14. **Data Scaling.**

from sklearn.preprocessing import StandardScaler

scaling\_x=StandardScaler()

X\_train=scaling\_x.fit\_transform(X\_train)

X\_test=scaling\_x.transform(X\_test)

- Scales the features using `StandardScaler` to standardize the range of independent variables.

15. **Building a Random Forest Classifier.**

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier()

rfc.fit(X\_train, y\_train)

- Creates and fits a Random Forest Classifier model to the training data.

16. **Model Prediction and Accuracy Score.**

rfc.predict(X\_test)

rfc.score(X\_test, y\_test)

- Makes predictions on the test set and calculates the accuracy score.

17. **Confusion Matrix Visualization.**

from sklearn.metrics import confusion\_matrix

mat = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(7, 5))

sns.heatmap(mat, annot=True)

- Plots a heatmap of the confusion matrix.

18. **Classification Report.**

from sklearn.metrics import classification\_report

target\_names = ['Diabetes', 'Normal']

print(classification\_report(y\_test, y\_pred, target\_names=target\_names))

- Prints a classification report including precision, recall, and F1-score.

19. **ROC Curve.**

from sklearn.metrics import roc\_curve

y\_pred\_proba = rfc.predict\_proba(X\_test)[:,1]

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

- Calculates the Receiver Operating Characteristic (ROC) curve.

20. **Plotting ROC Curve.**

plt.plot([0,1],[0,1],'k-')

plt.plot(fpr,tpr, label='Knn')

plt.xlabel('fpr')

plt.ylabel('tpr')

plt.title('ROC curve')

plt.show()

- Plots the ROC curve.

21. **Area under ROC Curve.**

from sklearn.metrics import roc\_auc\_score

roc\_auc\_score(y\_test,y\_pred\_proba)

- Calculates and prints the area under the ROC curve.

22. **Scatter Plot of True vs. Predicted Values.**

index = np.arange(0,len(y\_test))

fig, ax = plt.subplots(1,1,figsize=(15,5))

plt.scatter(index,y\_test,c="red",label = 'True Value')

plt.scatter(index,y\_pred,c="blue", label = 'Predicted Value')

plt.legend()

- Plots a scatter plot comparing true and predicted values.