Experiment No. 6
Apply Random Forest algorithm for credit fraud detection
Date of Performance:
Date of Submission:

Aim: Apply Random Forest algorithm for credit fraud detection.

Objective: Ability to implement ensemble learning algorithms.

Theory:

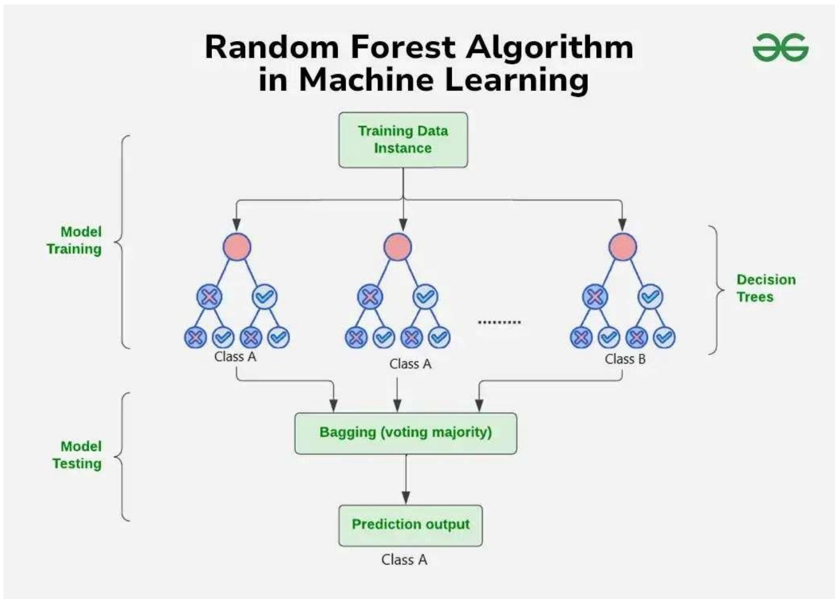
A Random Forest is a collection of decision trees that work together to make predictions. In this article, we'll explain how the Random Forest algorithm works and how to use it.

Understanding Intuition for Random Forest Algorithm

Random Forest algorithm is a powerful tree learning technique in Machine Learning to make predictions and then we do voting of all the tress to make prediction. They are widely used for classification and regression task.

- 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.

Imagine asking a group of friends for advice on where to go for vacation. Each friend gives their recommendation based on their unique perspective and preferences (decision trees trained on different subsets of data). You then make your final decision by considering the majority opinion or averaging their suggestions (ensemble prediction).



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).

Key Features of Random Forest

- Handles Missing Data: Automatically handles missing values during training, eliminating the need for manual imputation.
- Algorithm ranks features based on their importance in making predictions offering valuable insights for feature selection and interpretability.
- Scales Well with Large and Complex Data without significant performance degradation.

- Algorithm is versatile and can be applied to both classification tasks (e.g., predicting categories) and regression tasks (e.g., predicting continuous values).

How Random Forest Algorithm Works?

The random Forest algorithm works in several steps:

- Random Forest builds multiple decision trees using random samples of the data. Each tree is trained on a different subset of the data which makes each tree unique.
- When creating each tree the algorithm randomly selects a subset of features or variables to split the data rather than using all available features at a time. This adds diversity to the trees.
- Each decision tree in the forest makes a prediction based on the data it was trained on. When making final prediction random forest combines the results from all the trees. o For classification tasks the final prediction is decided by a majority vote. This means that the category predicted by most trees is the final prediction. o For regression tasks the final prediction is the average of the predictions from all the trees.
- The randomness in data samples and feature selection helps to prevent the model from overfitting making the predictions more accurate and reliable.

Assumptions of Random Forest

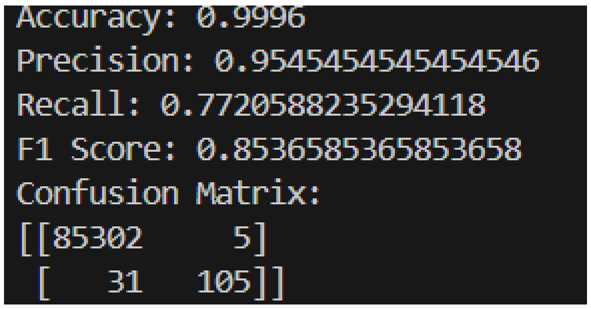
- Each tree makes its own decisions: Every tree in the forest makes its own predictions without relying on others.
- Random parts of the data are used: Each tree is built using random samples and features to reduce mistakes.
- Enough data is needed: Sufficient data ensures the trees are different and learn unique patterns and variety.
- Different predictions improve accuracy: Combining the predictions from different trees leads to a more accurate final results.

Code:

```
# Importing Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,
recall_score, f1_score

# Step 1: Load the dataset
df = pd.read_csv('creditcard.csv')
df = df.dropna()

# Step 2: Separate features and target variable
X = df.drop('Class', axis=1)
y = df['Class']

# Step 3: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 4: Initialize and train the Random Forest model with optimized parameters
rf_model = RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42,
n_jobs=-1)
rf_model.fit(X_train, y_train)

# Step 5: Predict using the model
y_pred = rf_model.predict(X_test)

# Step 6: Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Step 7: Print the results in the required format
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.16f}')
print(f'Recall: {recall:.16f}')
print(f'F1 Score: {f1:.16f}')
print('Confusion Matrix:')
print(conf_matrix)
```

Output:



```
Accuracy: 0.9996
Precision: 0.9545454545454546
Recall: 0.7720588235294118
F1 Score: 0.8536585365853658
Confusion Matrix:
[[85302     5]
 [   31   105]]
```

Conclusion:

Comment on the architecture and the results obtained.

The Random Forest model used for classification demonstrated excellent accuracy (0.9996) with a high precision of 0.9545, indicating a low false positive rate. However, the recall (0.7721) suggests that some actual positives were missed. The F1 Score of 0.8537 reflects a good balance between precision and recall. The confusion matrix highlights minimal misclassifications, indicating the model's robustness. Overall, the architecture is well-suited for the dataset, but further fine-tuning could enhance recall for better performance.