Random Forest

(Code: Subhajit Das)

What is Random Forest:

“Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.” Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

Random Forest works in two phases:

1. To create the random forest by combining N decision trees
2. To make predictions for each tree created in the first phase.

For new data points, it finds the predictions of each decision tree, and assigns the new data points to the category that wins the majority votes.

One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables, as in the case of regression, and categorical variables, as in the case of classification. It performs better for classification and regression tasks.

Where we can use Random Forest:

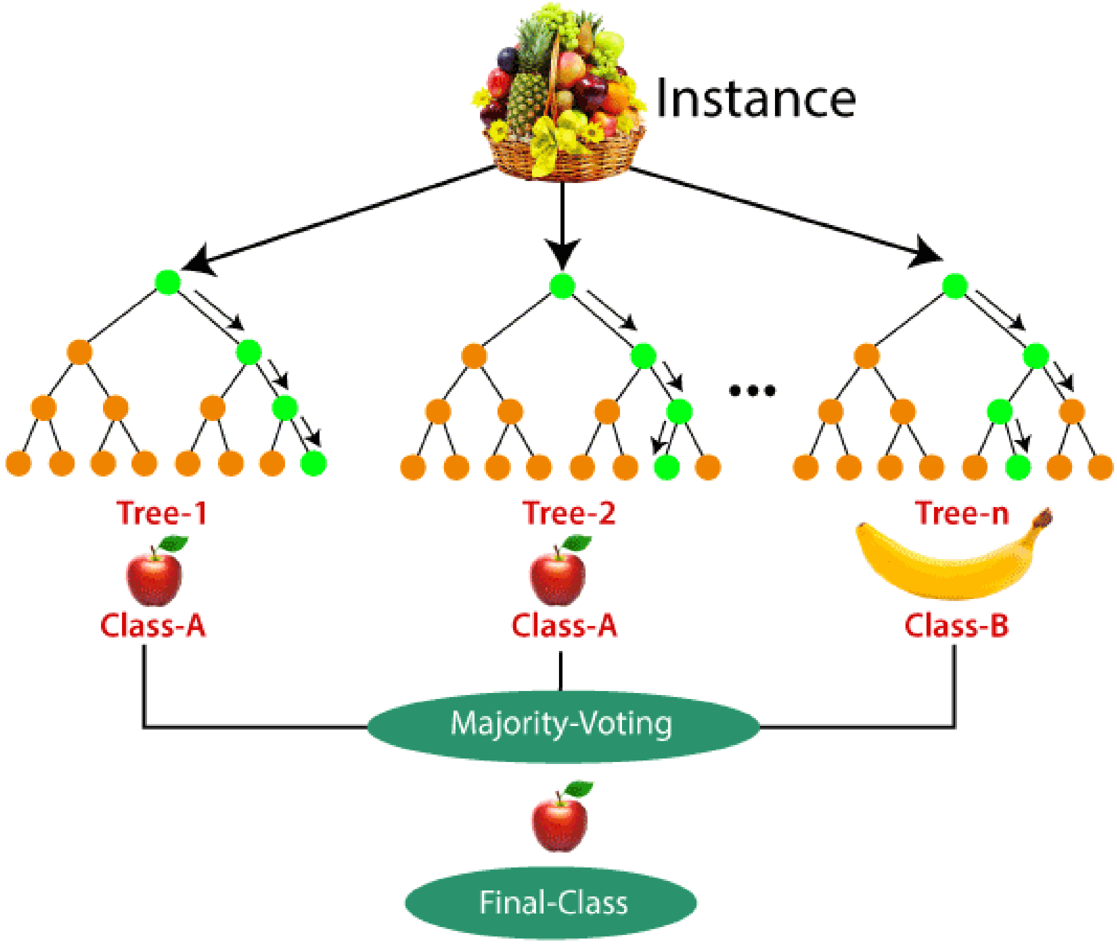
Random Forest has a wide range of applications across various domains. Here are some of the sectors where it is commonly used:

1. **Banking Industry**: Random Forest can be used for credit card fraud detection and customer segmentation. It can also help in predicting loan defaults.
2. **Healthcare and Medicine**: It finds applications in predicting various diseases like cardiovascular disease, diabetes, and breast cancer..
3. **Stock Market**: Random Forest can be used for stock market prediction and sentiment analysis.
4. **E-Commerce**: It can be used for product recommendation, price optimization, and search ranking.
5. **Finance**: Random Forests are widely used for tasks like fraud detection.
6. **Marketing**: They are widely used for tasks like customer churn prediction.
7. **Image Classification**: They are widely used for tasks like image classification.
8. **Regression**: Random Forests are effective in predicting continuous values, such as house prices or stock prices.

Use of Decision Tree in Random Forest:

Supervised Learning: 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.

How Random Forest Algorithm works:



In [ ]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
```

In [ ]:
```python
tennis_df = pd.read_csv("/content/drive/MyDrive/ML and DL DataSets/6_Play_Te
tennis_df
```

Out[2]:

	outlook	temp	humidity	wind	play
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

LabelEncoder

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
```

In [ ]:

```
Le = LabelEncoder()
```

In [ ]:

```
# 1. fit the encoder using the '.fit()' method on your data. This determines
# 2. transform your data into numerical labels using the '.transform()' meth

#  A categorical feature with values ‘low’, ‘medium’, and ‘high’, LabelEncod
tennis_df['outlook'] = Le.fit_transform(tennis_df['outlook'])
tennis_df['temp'] = Le.fit_transform(tennis_df['temp'])
tennis_df['humidity'] = Le.fit_transform(tennis_df['humidity'])
tennis_df['wind'] = Le.fit_transform(tennis_df['wind'])
tennis_df['play'] = Le.fit_transform(tennis_df['play'])
tennis_df
```

Out[5]:

	outlook	temp	humidity	wind	play
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1
5	1	0	1	0	0
6	0	0	1	0	1
7	2	2	0	1	0
8	2	0	1	1	1
9	1	2	1	1	1
10	2	2	1	0	1
11	0	2	0	0	1
12	0	1	1	1	1
13	1	2	0	0	0

Separating features and labels

In [ ]:

```
x = tennis_df[['outlook', 'temp', 'humidity',   'wind']]
x
```

Out[6]:

	outlook	temp	humidity	wind
0	2	1	0	1
1	2	1	0	0
2	0	1	0	1
3	1	2	0	1
4	1	0	1	1
5	1	0	1	0
6	0	0	1	0
7	2	2	0	1
8	2	0	1	1
9	1	2	1	1
10	2	2	1	0
11	0	2	0	0
12	0	1	1	1
13	1	2	0	0

In [ ]:

```
y = tennis_df['play']
y
```

Out[7]:

```
0     0
1     0
2     1
3     1
4     1
5     0
6     1
7     0
8     1
9     1
10    1
11    1
12    1
13    0
Name: play, dtype: int64
```

Spliting train and test datasets

In [ ]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

In [ ]:

```
# length or sample of train dataset
len(x_train)
```

Out[10]:

```
11
```

In [ ]:

```
# length or sample of test dataset
len(x_test)
```

Out[11]:

```
3
```

Using Random Forest Classifier

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:

```
rfc = RandomForestClassifier(n_estimators = 100,
                             criterion = "gini")
```

In [ ]:

```
rfc.fit(x_train, y_train)
```

Out[14]:

```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:

```
rfc.predict(x_test)
```

Out[15]:

```
array([1, 1, 0])
```

In [ ]:

```
x_test
```

Out[16]:

	outlook	temp	humidity	wind
4	1	0	1	1
13	1	2	0	0
8	2	0	1	1

In [ ]:

```
y_test
```

Out[17]:

```
4     1
13    0
8     1
Name: play, dtype: int64
```

In [ ]:

```
rfc.score(x_test, y_test)
```

Out[18]:

```
0.3333333333333333
```

Random Forest in Iris Dataset

In [ ]:

```
iris_df = sns.load_dataset('iris')
iris_df.head()
```

Out[19]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [ ]:

```
# We can also import in this way
from sklearn import datasets
iris = datasets.load_iris()
type(iris) # But here we have to convert this into DataFrame first
```

Out[20]:

```
sklearn.utils._bunch.Bunch
```

In [ ]:

```
print(iris.target_names)
print(iris.feature_names)
```

```
['setosa' 'versicolor' 'virginica']
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal widt
h (cm)']
```

In [ ]:

```
iris_df.shape
```

Out[22]:

```
(150, 5)
```

In [ ]:

```
iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [ ]:

```
iris_df.isnull().sum() # Iris dataset contains no null value
```

Out[24]:

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

Feature Engineering

Label Encoder transforming categorical to numerical value

In [ ]:

```
from sklearn.preprocessing import OneHotEncoder
```

In [ ]:

```
Le = LabelEncoder()
```

In [ ]:

```
iris_df['species'] = Le.fit_transform(iris_df[['species']])
iris_df
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:11
6: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

Out[27]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

Separating features and labels

In [ ]:

```
x = iris_df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
x.head()
```

Out[28]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [ ]:

```
iris_df.species.unique()
```

Out[29]:

```
array([0, 1, 2])
```

In [ ]:

```python
y = iris_df[['species']]
y.head()
```

Out[30]:

	species
0	0
1	0
2	0
3	0
4	0

Spliting train and test datasets

In [ ]:

```python
from sklearn.model_selection import train_test_split
```

In [ ]:

```python
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

In [ ]:

```python
# length or sample of train dataset
len(x_train)
```

Out[33]:

```
105
```

In [ ]:

```python
# length or sample of test dataset
len(x_test)
```

Out[34]:

```
45
```

Using Random Forest Classifier

Parameters used in RandomForestClassifier():

The `RandomForestClassifier` in `sklearn` has several parameters:

1. **n_estimators**: The number of trees in the forest.
2. **criterion**: The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
3. **max_depth**: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
4. **min_samples_split**: The minimum number of samples required to split an internal node.
5. **min_samples_leaf**: The minimum number of samples required to be at a leaf node.
6. **min_weight_fraction_leaf**: The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node.
7. **max_features**: The number of features to consider when looking for the best split.
8. **random_state**: Controls the randomness of the estimator.
9. **bootstrap**: Whether bootstrap samples are used when building trees.
10. **oob_score**: Whether to use out-of-bag samples to estimate the generalization accuracy.
11. **n_jobs**: The number of jobs to run in parallel.
12. **verbose**: Controls the verbosity when fitting and predicting.
13. **warm_start**: When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble.
14. **class_weight**: Weights associated with classes in the form `{class_label: weight}` .
15. **ccp_alpha**: Complexity parameter used for Minimal Cost-Complexity Pruning.

16. **max_samples**: If bootstrap is True, the number of samples to draw from X to train each base estimator.

These parameters allow you to control the behavior of the random forest and can be tuned to improve the performance of the model on your specific task.

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:

```
iris_rfc = RandomForestClassifier(n_estimators = 100, # Specifies the number
                                                      # The more trees there
                       criterion = "gini")
```

In [ ]:

```
iris_rfc.fit(x_train, y_train)
```

```
<ipython-input-37-f59c81ba2419>:1: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  iris_rfc.fit(x_train, y_train)
```

Out[37]:

```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:

```
iris_rfc.predict(x_test)
```

Out[38]:

```
array([0, 2, 2, 2, 1, 2, 0, 1, 2, 2, 0, 1, 2, 1, 2, 1, 1, 0, 1, 2, 1, 2,
       0, 0, 0, 1, 2, 0, 1, 0, 2, 1, 0, 1, 2, 0, 0, 2, 1, 0, 1, 0, 0, 0,
       2])
```

In [ ]:

```
x_test.head()
```

Out[39]:

	sepal_length	sepal_width	petal_length	petal_width
6	4.6	3.4	1.4	0.3
130	7.4	2.8	6.1	1.9
116	6.5	3.0	5.5	1.8
128	6.4	2.8	5.6	2.1
73	6.1	2.8	4.7	1.2

In [ ]:

```
y_test.head()
```

Out[40]:

	species
6	0
130	2
116	2
128	2
73	1

Viewing the prediction score

In [ ]:
```
iris_rfc.score(x_test, y_test)
```

Out[41]:
```
0.9555555555555556
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classificatior
```

In [ ]:
```
pred_iris = iris_rfc.predict(x_test) # storing the predictions that your moc
```

In [ ]:
```
# Classification metrics can't handle a mix of multiclass-multioutput and mu
confusion_matrix(y_test, pred_iris)
```

Out[44]:
```
array([[16,  0,  0],
       [ 0, 13,  1],
       [ 0,  1, 14]])
```

In [ ]:
```
accuracy_score(y_test, pred_iris)
```

Out[45]:
```
0.9555555555555556
```

In [ ]:
```
print(classification_report(y_test, pred_iris))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.93      0.93      0.93        14
           2       0.93      0.93      0.93        15

    accuracy                           0.96        45
   macro avg       0.95      0.95      0.95        45
weighted avg       0.96      0.96      0.96        45
```

Predict Iris Flower with user input

In [ ]:
```
iris_predict = LabelEncoder()
categories = ['setosa' 'versicolor' 'virginica']
iris_predict.fit(categories)
```

Out[47]:
```
LabelEncoder()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:

```python
sepal_length = input("Enter the sepal_length: ")

sepal_width = input("Enter the sepal_width: ")

petal_length = input("Enter the petal_length: ")

petal_width = input("Enter the petal_width: ")

label_map = {0: 'setosa', 1: 'versicolor', 2: 'virginica'}
predict_value = iris_rfc.predict([[sepal_length, sepal_width, petal_length,

# Map the numerical prediction back to a string label
label_value = label_map[predict_value[0]]

print(label_value) # 0: 'setosa', 1: 'versicolor', 2: 'virginica'
```

```
Enter the sepal_length: 16
Enter the sepal_width: 20
Enter the petal_length: 10
Enter the petal_width: 8
virginica
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:
X does not have valid feature names, but RandomForestClassifier was fitted
with feature names
  warnings.warn(
```