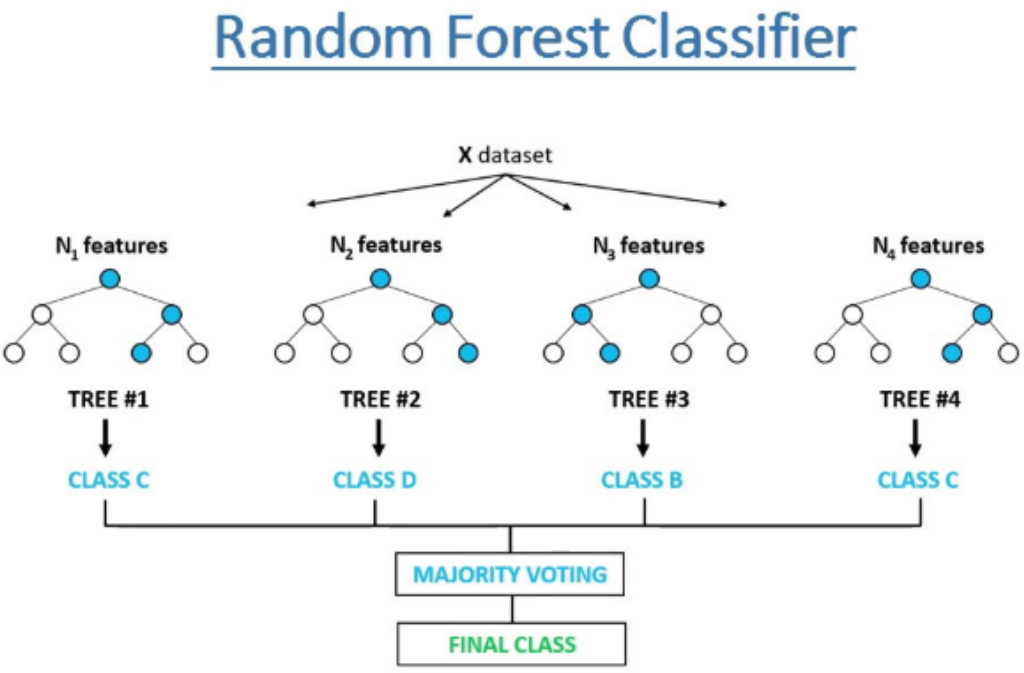
Random Forest



Random Forest is a robust machine learning algorithm that can be **used for** a **variety** of tasks **including regression and classification**.

It is an **ensemble method**, **meaning** that a **random forest model is made up of a large number of small decision trees**, **called estimators**, which each **produce** their **own predictions**.

The random forest model **combines** the **predictions** of the **estimators** to **produce** a **more accurate prediction**.

Standard decision tree classifiers have the **disadvantage** that they are **prone to overfitting to the training set.**

The **random forest's ensemble design** allows the **random forest to compensate** for this and **generalize well to unseen data**, **including data with missing values.**

Random forests are also **good at handling large datasets** with high **dimensionality and heterogeneous feature types** (for example, if one column is categorical and another is numerical).

Features of a Random Forest Algorithm

1.It's **more accurate** than the **decision tree algorithm**.

2.It **provides** an **effective way** of **handling missing data**.

3.It can **produce** a **reasonable prediction without hyper-parameter tuning**.

4.It **solves** the **issue of overfitting** in **decision trees**.

5.In **every random forest tree**, a **subset of features** is **selected randomly at the node's splitting point**.

Random forests are **very good** for **classification problems** but are **slightly less good** at **regression problems**.

In contrast to linear regression, a random forest regressor is unable to make predictions outside the range of its training data.

How random forest algorithm works?

The **main difference between** the **decision tree algorithm** and the **random forest algorithm** is that establishing **root nodes and segregating nodes** is done **randomly** in the latter.

The **random forest employs** the **bagging method to generate the required prediction.**

Bagging involves **using different samples of data** (training data) **rather than just one sample. **

A **training dataset** comprises **observations and features** that are used for **making predictions**.

The **decision trees produce different outputs**, **depending** on the **training data fed to the random forest algorithm**.

These **outputs** will be **ranked**, and the **highest** will be **selected as the final output.**

Random Forest is **provided** via the **RandomForestRegressor** and **RandomForestClassifier** classes.

Both **models operate** the **same way** and **take the same arguments** that **influence** how the **decision trees are created**.

Randomness is used in the **construction of the model**. This means that each **time the algorithm is run on the same data**, **it will produce a slightly different model.**

When **using machine learning algorithms** that have a **stochastic learning algorithm**, it is **good practice** to evaluate **them by averaging their performance across multiple runs or repeats of cross-validation**.

When fitting a final model, it may be **desirable** to **either increase the number of trees until** the **variance** of the model is **reduced across repeated evaluations**, **or** to **fit multiple final models** and **average their predictions**.

Random Forest for Classification

Classification in random forests **employs** an **ensemble methodology** to **attain** the **outcome**.

The **training data** is fed to **train various decision trees**.

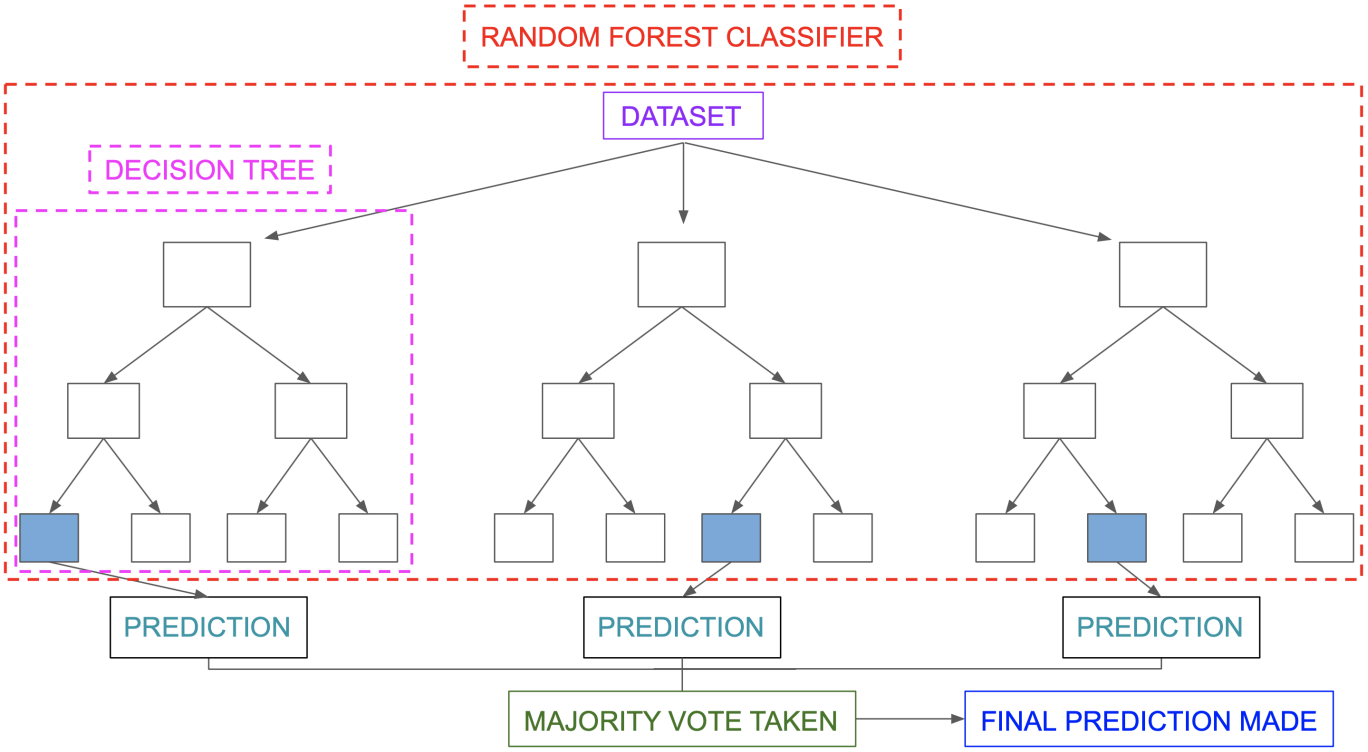
This **dataset** consists of observations and features that will be selected randomly during the splitting of nodes.

A rain **forest system** relies on **various decision trees**. Every decision tree consists of decision nodes, leaf nodes, and a root node.

The leaf node of each tree is the final output produced by that specific decision tree.

The **selection** of the **final output** follows the **majority-voting system**.

In this case, the output chosen by the majority of the decision trees becomes the final output of the rain forest system.



Random Forest Regression

Random forest is an ensemble of decision trees. This is to say that many trees, constructed in a certain “random” way form a Random Forest.

1.Each tree is created from a **different sample of rows and at each node**, a different **sample of features is selected for splitting**.

2.Each of the **trees makes its own individual prediction**.

3.These predictions are then **averaged** to produce a single result.

The **averaging makes a Random Forest better than a single Decision Tree hence improves its accuracy and reduces overfitting**.

A **prediction from the Random Forest Regressor** is an **average of the predictions** produced by the **trees in the forest**.

Important Terms to Know

There are different ways that Random Forest algorithm makes data decisions, and consequently, there are some important related terms to know. Some of these terms include:

Entropy:

It is a **measure of randomness** or unpredictability in the data set.

Information Gain:

A **measure of the decrease** in the **entropy after** the **data set is split is the information gain**.

Leaf Node:

A **leaf node** is a **node** that **carries the classification or the decision**.

Decision Node:

A **node** that **has two or more branches**.

Root Node:

The **root node is the topmost decision node**, which is where you have all of your data.

credits:https://www.simplilearn.com/tutorials/machine-learning-tutorial/random-forest-algorithm

Advantages and Disadvantages of Random Forest

1.It **reduces overfitting** in decision trees and helps to **improve the accuracy**

2.It is **flexible to both classification and regression problems**

3.It works well with **both categorical and continuous values**

4.It **automates missing values present** in the data

5.**Normalising of data is not required as it uses a rule-based approach.**

Disadvanatages

1.It requires much **computational power** as well as resources as it builds numerous trees to combine their outputs.

2.It also **requires much time for training as it combines** a lot of decision trees to determine the class.

3.Due to the **ensemble of decision trees**, it also **suffers interpretability and fails to determine the significance of each variable**.

CUSTOMER SEGMENTATION USING Random Forest

DATASET:https://www.kaggle.com/prathmeshpatil12/customersegmentation

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.ensemble import RandomForestClassifier

# Create the model with 100 trees
model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True,
                               max_features = 'sqrt')
```

```
df=pd.read_csv("/content/Train (2).csv")
```

```
df.head()
```

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Sc
0	462809	Male	No	22	No	Healthcare	1.0	
1	462643	Female	Yes	38	Yes	Engineer	NaN	Ave
2	466315	Female	Yes	67	Yes	Engineer	1.0	
3	461735	Male	Yes	67	Yes	Lawyer	0.0	
4	462669	Female	Yes	40	Yes	Entertainment	NaN	

```
df.shape
```

```
(8068, 11)
```

```
df.columns
```

```
Index(['ID', 'Gender', 'Ever_Married', 'Age', 'Graduated', 'Profession',
       'Work_Experience', 'Spending_Score', 'Family_Size', 'Var_1',
       'Segmentation'],
      dtype='object')
```

```
df.drop(['ID'], axis=1)
```

	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score
0	Male	No	22	No	Healthcare	1.0	Low
1	Female	Yes	38	Yes	Engineer	NaN	Average
2	Female	Yes	67	Yes	Engineer	1.0	Low
3	Male	Yes	67	Yes	Lawyer	0.0	High
4	Female	Yes	40	Yes	Entertainment	NaN	High
...	...	...	...	...	...	...	...
8063	Male	No	22	No	NaN	0.0	Low
8064	Male	No	35	No	Executive	3.0	Low
8065	Female	No	33	Yes	Healthcare	1.0	Low

```
df['Segmentation'].value_counts()
```

```
D    2268
A    1972
C    1970
B    1858
Name: Segmentation, dtype: int64
```

```
df.isnull().sum()
```

```
ID                 0
Gender             0
Ever_Married     140
Age                0
Graduated         78
Profession       124
Work_Experience  829
Spending_Score     0
Family_Size      335
Var_1             76
Segmentation       0
dtype: int64
```

```
df.isnull().sum()
```

```
ID                 0
Gender             0
Ever_Married     140
Age                0
Graduated         78
Profession       124
Work_Experience  829
Spending_Score     0
Family_Size      335
Var_1             76
Segmentation       0
dtype: int64
```

```
df.dtypes
```

```
ID                int64
Gender           object
```

```
Ever_Married        object
Age                  int64
Graduated           object
Profession          object
Work_Experience    float64
Spending_Score      object
Family_Size        float64
Var_1               object
Segmentation        object
dtype: object
```

```
categorical_features = df.select_dtypes(include=[np.object]).columns
print("total categorical_features",len(categorical_features))
```

```
total categorical_features 7
```

```
df['Spending_Score'].value_counts()
```

```
Low        4878
Average    1974
High       1216
Name: Spending_Score, dtype: int64
```

```
df['Profession'].value_counts()
```

```
Artist           2516
Healthcare       1332
Entertainment     949
Engineer          699
Doctor            688
Lawyer            623
Executive         599
Marketing         292
Homemaker         246
Name: Profession, dtype: int64
```

```
df['Var_1'].value_counts()
```

```
Cat_6    5238
Cat_4    1089
Cat_3     822
Cat_2     422
Cat_7     203
Cat_1     133
Cat_5      85
Name: Var_1, dtype: int64
```

```
df['Family_Size'].value_counts()
```

```
2.0    2390
3.0    1497
1.0    1453
4.0    1379
5.0     612
6.0     212
7.0      96
8.0      50
9.0      44
Name: Family_Size, dtype: int64
```

Encoding Categorical Data

```
# Replacing Male with 0 and Female with 1 in Gender column
df['Gender'] = df['Gender'].replace({'Male':0,'Female':1})


# Replacing No with 0 and Yes with 1 in Ever_Married column
df['Ever_Married'] = df['Ever_Married'].replace({'No':0,'Yes':1})


# Replacing No with 0 and Yes with 1 in Graduated column
df['Graduated'] = df['Graduated'].replace({'No':0,'Yes':1})


# Replacing Low with 0, Average with 1 and High with 2 in Spending_Score column
df['Spending_Score'] = df['Spending_Score'].replace({'Low':0,'Average':1,'High':2})


# Replacing Cat_1 with 1, Cat_2 with 2, Cat_3 with 3, Cat4 with 4, Cat_5 with 5, Cat_6 with 6, a
df['Var_1'] = df['Var_1'].replace({'Cat_1':1,'Cat_2':2,'Cat_3':3,'Cat_4':4,'Cat_5':5,'Cat_6':6,'


# Replacing Artist with 0, Healthcare with 1, Entertainment with 2, Engineer with 3, Doctor with
df['Profession'] = df['Profession'].replace({'Artist':0,'Healthcare':1,'Entertainment':2,'Engine


# Replacing A with 0, B with 1, C with 2 and D with 3 in Spending_Score column
df['Segmentation'] = df['Segmentation'].replace({'A':0,'B':1,'C':2,'D':3})


df
```

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_
0	462809	0	0.0	22	0.0	1.0	1.0	
1	462643	1	1.0	38	1.0	3.0	NaN	
2	466315	1	1.0	67	1.0	3.0	1.0	
3	461735	0	1.0	67	1.0	5.0	0.0	
4	462669	1	1.0	40	1.0	2.0	NaN	
...	...	...	...	...	...	...	...	
8063	464018	0	0.0	22	0.0	NaN	0.0	
8064	464685	0	0.0	35	0.0	6.0	3.0	
8065	465406	1	0.0	33	1.0	1.0	1.0	
8066	467299	1	0.0	27	1.0	1.0	1.0	
8067	461879	0	1.0	37	1.0	6.0	0.0	

8068 rows × 11 columns

Filling missing data

```
df.isnull().sum()
```

```
ID                  0
Gender              0
Ever_Married      140
Age                 0
Graduated          78
Profession        124
Work_Experience   829
Spending_Score      0
Family_Size       335
Var_1              76
Segmentation        0
dtype: int64
```

```
df['Ever_Married'].fillna(int(df['Ever_Married'].mean()), inplace=True)
```

```
df['Ever_Married'].value_counts()
```

```
1.0    4643
0.0    3425
Name: Ever_Married, dtype: int64
```

```
df['Graduated'].fillna(int(df['Graduated'].mean()), inplace=True)
df['Profession'].fillna(int(df['Profession'].mean()), inplace=True)
df['Work_Experience'].fillna(int(df['Work_Experience'].mean()), inplace=True)
df['Family_Size'].fillna(int(df['Family_Size'].mean()), inplace=True)
df['Var_1'].fillna(int(df['Var_1'].mean()), inplace=True)
```

```
df.isna().sum()
```

```
ID                0
Gender            0
Ever_Married      0
Age               0
Graduated         0
Profession        0
Work_Experience   0
Spending_Score    0
Family_Size       0
Var_1             0
Segmentation      0
dtype: int64
```

Splitting the data

```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

```
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=42)
```

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier()
classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
y_pred = classifier.predict(X_test)
```

```
df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
df
```

	Actual	Predicted
0	3	3
1	0	0
2	3	3
3	2	1
4	3	2
...	...	...
1609	0	0
1610	1	1
1611	0	1
1612	0	3
1613	3	0

1614 rows × 2 columns

Evaluation Metrics

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[156  77  47 111]
 [ 88 123 111  47]
 [ 50  79 210  41]
 [ 84  31  25 334]]
0.509913258983891
```

```
from sklearn.metrics import precision_score
print("Precision Score : ",precision_score(y_test, y_pred,
                                           pos_label='positive',average='micro'))
```

```
Precision Score :  0.509913258983891
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1321: UserWarnin
  % (pos_label, average), UserWarning)
```

```
from sklearn.metrics import recall_score
print("Recall Score : ",recall_score(y_test, y_pred,
                                           pos_label='positive',average='micro'))
```

```
Recall Score :  0.509913258983891
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1321: UserWarnin
  % (pos_label, average), UserWarning)
```

```
from sklearn.metrics import f1_score
print("f1_Score : ",f1_score(y_test, y_pred,
                                           pos_label='positive',average='micro'))
```

```
f1_Score :  0.509913258983891
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1321: UserWarnin
  % (pos_label, average), UserWarning)
```