Machine Learning 🖥️

Classification and Regression Tree (CART) Algorithm

Decision Tree

By ⟹ PRINCE 👑❤️

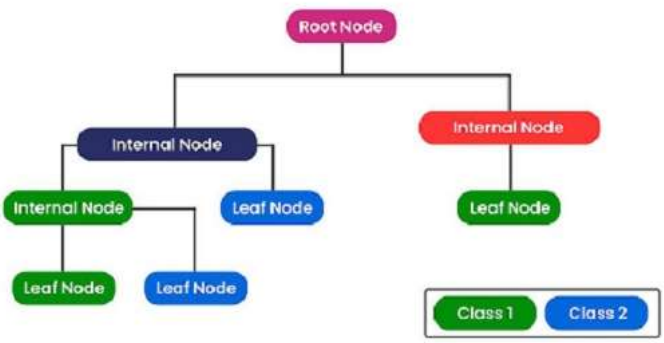
Classification and Regression Tree (CART)⟶

- CART is a variation of the decision tree algorithm that can handle both classification and regression tasks. Scikit-Learn uses the Classification And Regression Tree (CART) algorithm to train Decision Trees (also called "growing" trees).

CART Algorithm ⟶

- CART is a predictive algorithm that is used in Machine learning. It explains how the target variable's values can be predicted based on other matters. A decision tree where each fork is split into a predictor variable and each node has a prediction for the target variable at the end.
- In the decision tree, the root node is taken as the training set and is split into two by considering the best attribute and threshold value. The subsets are also split using the same logic. This continues till the last pure sub-set is found in the tree or the maximum number of leaves possible in that growing tree.
- CART ⟹ Decision Tree.
- Usage in Classification ⟹ 98 %
- Usage in Regression ⟹ 2 %
- An algorithm that initiates human thinking in machine learning.

Decision Tree Architecture



Max Depth : max_depth ⟶

- A parameter of Decision Tree.
- This parameter in Python defines the depth of the decision tree or the number of level of nodes we want in a decision tree.

Disadvantages of Decision Tree :

1. Overfitting ⟶

- When train accuracy is higher than the test accuracy then overfitting occurs in any Machine Learning Algorithm.
- It is a high variance algorithm, it means that it can easily overfit because it has no inherent mechanism to stop, thereby creating complex decision rules.
- A decision tree can be highly time-consuming in its training phase.
- Decision Tree is prone to obverfitting, i.e overfitting is common in the algorithm.

▶ *How to overcome with overfitting :*

Pruning Technique ⟶ `The cutting of decision trees from the bottom or the cutting of level of nodes of decision trees.`

- If decision tree is left empty without its parameters then it is bound to be overfitted. **If DecisionTreeClassifier( ) ⟹ Overfitting**

- **max_depth** is a parameter that defines the level of nodes and that is also used for **pruning** (cutting of decision tree).

2. Optimization ⟶

At every level, the decision tree algorithm looks for the pure node and doesn't consider how the recent decision will affect the next few stages of splitting.

3. Decision Trees are unstable.

4. Limited performance in regression.

Advantages of Decision Tree :

1. Highly intutive and easy to understand.

2. Less number of data preparation steps.

3. It does not required lot of assumptions.

4. Decision trees can create complex decision boundaries, allowing them to easily solve non-linear problems.

Parameters for Decision Tree :

1. Max Depth : The maximum depth of trees allowed. The default value is set to none.

2. Minimum Sample Split : The minimum number of observations required in a node to be allowed in a leaf node.

3. Min Samples Leaf : The minimum number of observations allowed in a leaf node.

4. Max Features : The maximum number of features allowed for splitting the nodes.

5. Criterion : This parameter determines how the impurity of a split will be measured.

- Default value is **'gini'** but we can also use **'entropy'** as a metric for impurity.

6. Splitter : This is how the decision tree searches the features for a split. The default value is set to **'best'**.

Split Decision Tree ⟶

Criterion Parameter :

Criterion has two types/parameters :

1. **Gini Index/Impurity : Criterion = 'gini'**
2. **Entropy : Criterion = 'entropy'**

→ Both of them are probabilistic in nature.

→ Based on these two parameters a decision tree splits.

→ Ranges from 0 to 1

Gini Impurity : G I ⟶ criterion = 'gini'

- **A measurement used to build Decision Trees to determine how the features of a dataset should split nodes to form the tree.**
- **The Gini Impurity of a dataset is a number between 0-0.5, which indicates the likelihood of new, random data being misclassified if it were given a random class label according to the class distribution in the dataset.**

Formula :

Gini Impurity ⟶ G I ⟶ D = 1 - Gini Index

$$Gini\ Impurity = 1 - \sum_{i=1}^{n} p_i^2$$

Where,

- n is the number of class labels
- pi is the proportion of the class label that belongs to class k for a particular mode

---

Example :

Boys ⟶ b ⟶ 12

Girls ⟶ g ⟶ 17

G I = D = ?

Solution :

P(b) = 12 / 29

P(g) = 17 / 29

G I = 1 - [$(P(b)^2 + P(g)^2$]

G I = 1 - [$(0.413)^2 + (0.586)^2$]

G I = 1 - [0.17 + 0.34]

G I = 0.49

49 % chance ⟹ Impure Data, So Less chance of overfitting.

Entropy ⟶ Information Gain : criterion = 'entropy'

- In the context of Decision Trees, entropy is a measure of disorder or impurity in a node.
- A node with more variable composition would be consider to have higher entropy.
- Level of Entropy ranges from Zero(0) to One(1) .

Formula :

Information Gain = $\text{Entropy}^{\text{parent}}$ - $\text{Entropy}^{\text{children}}$

$$E = -\sum_{i=1}^{n} p_i \, log_2(p_i)$$

Example :

Boys ⟶ b ⟶ 12

Girls ⟶ g ⟶ 17

Entropy ⟶ E = ?

Solution :

P(b) = 12 / 29

P(g) = 17 / 29

E = - [$(0.413 \log_2(0.413) + (0.586 \log_2(0.586)$]

E = - [(0.41 * -1.28) + (0.58 * -0.78)]

E = - [-0.524 - 0.4524]

E = 0.98

98 % chance ⟹ Impure Data, So, 98 % chance of overfitting

Python Impelementation for CART (Decision Tree)

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import matplotlib.pyplot as plt
```

Mounting Google Drive

```
# Mounting Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
df = pd.read_csv('/content/drive/MyDrive/Dataset/Flight_Satisfaction.csv')
```

```
df.head()
```

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3

```
# To show all column
pd.set_option('display.max_columns',None)
df.head()
```

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3

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

```
Unnamed: 0                             0
id                                     0
Gender                                 0
Customer Type                          0
Age                                    0
Type of Travel                         0
Class                                  0
Flight Distance                        0
Inflight wifi service                  0
Departure/Arrival time convenient      0
Ease of Online booking                 0
Gate location                          0
Food and drink                         0
Online boarding                        0
Seat comfort                           0
Inflight entertainment                 0
On-board service                       0
Leg room service                       0
Baggage handling                       0
Checkin service                        0
Inflight service                       0
Cleanliness                            0
Departure Delay in Minutes             0
Arrival Delay in Minutes             310
satisfaction                           0
dtype: int64
```

Column 'Arrival Delay in Minutes' has 310 missing (null) values.

So, we have to fill these null values first.

```
df['Arrival Delay in Minutes'].fillna(df['Arrival Delay in Minutes'].median(), inplace = True)
df.isna().apply(pd.value_counts).T
```

	False
Unnamed: 0	103904
id	103904
Gender	103904
Customer Type	103904
Age	103904
Type of Travel	103904
Class	103904
Flight Distance	103904
Inflight wifi service	103904
Departure/Arrival time convenient	103904
Ease of Online booking	103904
Gate location	103904
Food and drink	103904
Online boarding	103904
Seat comfort	103904
Inflight entertainment	103904
On-board service	103904
Leg room service	103904
Baggage handling	103904
Checkin service	103904
Inflight service	103904
Cleanliness	103904
Departure Delay in Minutes	103904
Arrival Delay in Minutes	103904

```
df.shape
```

```
(103904, 25)
```

Showing Profile Report using Pandas

How to install Pandas-Profiling in Google Colab

Copy below command and paste it in Google Colab cell.

! pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip

---

```
# !pip uninstall pandas-profiling -y
```

```
#!pip install pandas-profiling
```

```
# How to install Pandas-Profiling in Google Colab
! pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
```

After Installing Restart the Kernel

```
# import pandas_profiling
# pandas_profiling.ProfileReport(df)
```

```
from pandas_profiling import ProfileReport
profile = ProfileReport(df, title='Flight Satisfaction', html={'style':{'full_width':True}})
profile.to_file(output_file = 'FLightSatisfaction.html')
profile
```

```
Summarize dataset: 100%                                        395/395 [01:47<00:00, 3.36it/s, Completed]
Generate report structure: 100%                                        1/1 [00:11<00:00, 11.75s/it]
Render HTML: 100%                                        1/1 [00:12<00:00, 12.16s/it]
Export report to file: 100%                                        1/1 [00:00<00:00, 7.67it/s]
```

Value	Count	Frequency (%)
female	52727	50.7%
male	51177	49.3%

Most occurring characters

Value	Count	Frequency (%)
e	156631	30.1%
a	103904	19.9%
l	103904	19.9%
F	52727	10.1%
m	52727	10.1%
M	51177	9.8%

Data Pre-Processing ⟹ Label Encoder

Converting object to int

```
# Object => int
# Data Pre-processing

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
df['Customer Type'] = le.fit_transform(df['Customer Type'])
df['Type of Travel'] = le.fit_transform(df['Type of Travel'])
df['Class'] = le.fit_transform(df['Class'])
df['satisfaction'] = le.fit_transform(df['satisfaction'])

df.head()
```

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease c Onlir bookir
0	0	70172	1	0	13	1	2	460	3	4	
1	1	5047	1	1	25	0	0	235	3	2	
2	2	110028	0	0	26	0	0	1142	2	2	
3	3	24026	0	0	25	0	0	562	2	5	
4	4	119299	1	0	61	0	0	214	3	3	

Now object is converted into int.

Correlation

```
corr = df.corr()
corr[['satisfaction']]
```

	satisfaction
Unnamed: 0	-0.004731
id	0.013734
Gender	0.012211
Customer Type	-0.187638
Age	0.137167
Type of Travel	-0.449000
Class	-0.449321
Flight Distance	0.298780
Inflight wifi service	0.284245
Departure/Arrival time convenient	-0.051601
Ease of Online booking	0.171705
Gate location	0.000682
Food and drink	0.209936
Online boarding	0.503557
Seat comfort	0.349459
Inflight entertainment	0.398059
On-board service	0.322383
Leg room service	0.313131
Baggage handling	0.247749
Checkin service	0.236174
Inflight service	0.244741
Cleanliness	0.305198
Departure Delay in Minutes	-0.050494
Arrival Delay in Minutes	-0.057435
satisfaction	1.000000

Applying Sixth Machine Learning Algorithm

```
x = df.drop(['Unnamed: 0', 'id', 'satisfaction'],axis = 1)
y = df[['satisfaction']]

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x,y,test_size = 0.3, random_state = 1)
```

```
xtrain.shape, xtest.shape
```

```
((72732, 22), (31172, 22))
```

```
ytrain.shape, ytest.shape
```

```
((72732, 1), (31172, 1))
```

Applying Decision Tree without any parameter

```
# Machine Learning Algorithm
# Applying Decision Tree without any parameter

from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier()
```

```
ypred = model.predict(xtest)
ypred
```

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

Training Accuracy

```
model.score(xtrain, ytrain)     # 100 % Accuracy => Clear case of overfitting
```

```
1.0
```

Training Accuracy = 1.0, 100 % Accuracy ⟶ Clear case of overfitting

Test Accuracy

```
model.score(xtest, ytest)
```

```
0.9441485948928525
```

- **Train Accuracy >> Test Accuracy**
- **Train Accuracy is 6 % higher than Test Accuracy, It means model is overfitted.**
- **0.1 % to 0.8 - 0.9 % higher value of Train accuracy can be taken as underfitted model.**

Pruning (Overcome with Overfitting)

```
model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 9)
# criterion = 'entropy' or 'gini' can be use.
model.fit(xtrain, ytrain)
ypred = model.predict(xtest)

print("Percentage of Training Accuracy=",model.score(xtrain, ytrain)*100)
print("Percentage of Test Accuracy=",model.score(xtest, ytest)*100)
```

```
Percentage of Training Accuracy= 94.71484353517022
Percentage of Test Accuracy= 94.26729115873219
```

We can take 'entropy' or 'gini' any one parameter to overcome with Overfitting, But here 'entropy' will give more accurate and better prediction than 'gini'.

- Percentage of Training Accuracy= 94.71
- Percentage of Test Accuracy= 94.27

Training Accuracy is only 0.24 % higher than Test Accuracy, So now model is not overfitted.

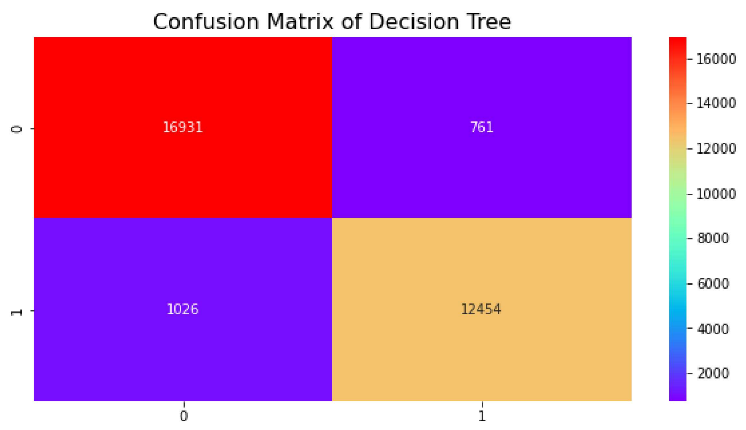
Confusion Matrix and Classification Report

```
from sklearn.metrics import confusion_matrix, classification_report
cf = confusion_matrix(ytest, ypred)
print(cf)
```

```
[[16931   761]
 [ 1026 12454]]
```

Plotting Confusion Matrix

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix of Decision Tree', fontsize = 16)
sns.heatmap(cf, fmt = 'g', annot = True, cmap = 'rainbow')
plt.show()
```



Classification Report

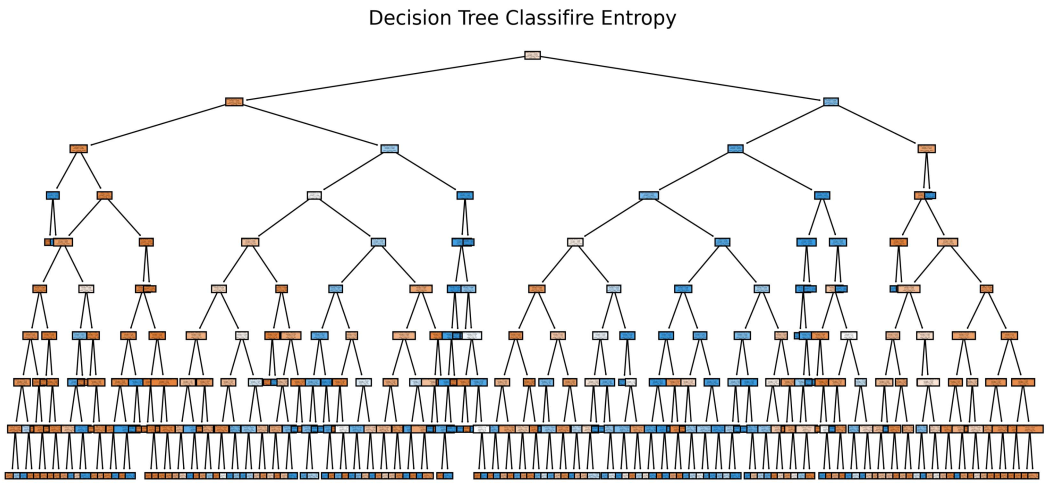
```
print(classification_report(ytest, ypred))

              precision    recall  f1-score   support

           0       0.94      0.96      0.95     17692
           1       0.94      0.92      0.93     13480

    accuracy                           0.94     31172
   macro avg       0.94      0.94      0.94     31172
weighted avg       0.94      0.94      0.94     31172
```

Plotting Decision Tree

Note :

In Python, we can plot Decision Tree and Linear Regression.

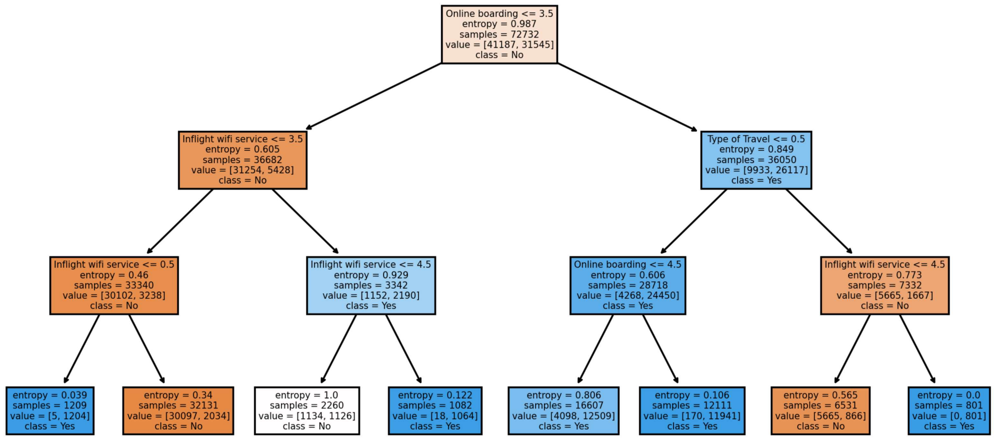
```
from sklearn.tree import plot_tree
f = list(xtrain)
c = ['No', 'Yes']
fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (15, 7), dpi = 300)
plot_tree(model, feature_names = f, class_names = c, filled = True)
plt.title('Decision Tree Classifire Entropy', fontsize = 16)
plt.show()
```



To better understand the concept of DT Plotting, taking max_depth = 3

```
model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 3)
model.fit(xtrain, ytrain)
ypred = model.predict(xtest)
f = list(xtrain)
c = ['No', 'Yes']
fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (10, 5), dpi = 300)
plot_tree(model, feature_names = f, class_names = c, filled = True)
plt.title('Decision Tree Classifire Entropy', fontsize = 12)
plt.show()
```

Decision Tree Classifire Entropy



0s completed at 4:56 AM