Decision Tree Algorithms - II

Contents

1. Decision Tree in Python :
    i. Classification Tree
    ii. Regression Tree

Case Study – Predicting Loan Defaulters

Background

- The bank possesses demographic and transactional data of its loan customers. If the bank has a robust model to predict defaulters it can undertake better resource allocation.

Objective

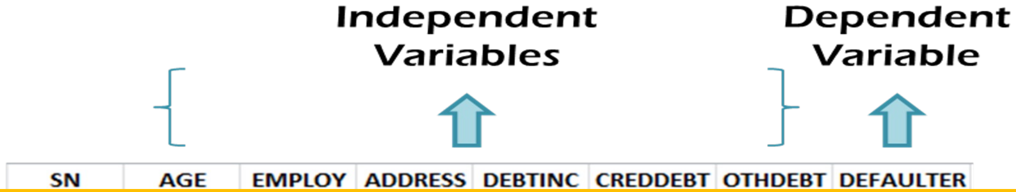
- To predict whether the customer applying for the loan will be a defaulter

Available Information

- Sample size is 700
- Age group, Years at current address, Years at current employer, Debt to Income Ratio, Credit Card Debts, Other Debts are the independent variables
- Defaulter (=1 if defaulter, 0 otherwise) is the dependent variable

Data Snapshot

BANK LOAN



Column	Description	Type	Measurement	Possible Values
SN	Serial Number	Integer	-	-
AGE	Age Groups	Integer	1(<28 years), 2(28-40 years), 3(>40 years)	3
EMPLOY	Number of years customer working at current employer	Integer	-	Positive value
ADDRESS	Number of years customer staying at current address	Integer	-	Positive value
DEBTINC	Debt to Income Ratio	Continuous	-	Positive value
CREDDEBT	Credit to Debit Ratio	Continuous	-	Positive value
OTHDEBT	Other Debt	Continuous	-	Positive value
DEFAULTER	Whether customer defaulted on loan	Integer	1(Defaulter), 0(Non-Defaulter)	2

Classification Tree in Python

```
# Importing the Libraries
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier,
DecisionTreeRegressor, plot_tree

from sklearn.metrics import confusion_matrix, precision_score,
recall_score, accuracy_score,roc_curve, roc_auc_score
```

- **sklearn.tree** module includes Decision Tree – based models for classification and regression

Classification Tree in Python

```
# Importing and Readying the Data for Modeling
bankloan = pd.read_csv("BANK LOAN.csv")


bankloan1 = bankloan.drop(['SN'], axis = 1)

bankloan1['AGE'] = bankloan1['AGE'].astype('category')

bankloan2 = pd.get_dummies(bankloan1)
bankloan2.head()
```

drop() is used to remove unwanted variables.

pd.get_dummies() converts categorical variables into dummy variables. Since AGE is a categorical variable, it is converted.

```
# Output
```

	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER	AGE_1	AGE_2	AGE_3
0	17	12	9.3	11.36	5.01	1	0	0	1
1	10	6	17.3	1.36	4.00	0	1	0	0
2	15	14	5.5	0.86	2.17	0	0	1	0
3	15	14	2.9	2.66	0.82	0	0	0	1
4	2	0	17.3	1.79	3.06	1	1	0	0

Classification Tree Using Information Gain

```
# Creating Data Partitions
X = bankloan2.loc[:,bankloan2.columns != 'DEFAULTER']
y = bankloan2.loc[:, 'DEFAULTER']

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.30,
                                                    random_state =
999)
```

- **train_test_split()** from sklearn.model_selection is used to split dataset into random train and test sets.
- **test_size** represents the proportion of dataset to be included in the test set.
- **random_state** sets the seed for the random number generator.

Classification Tree Using Information Gain

```
# Classification Tree Using Information Gain
dtcl = DecisionTreeClassifier(criterion='entropy',
                              min_samples_split=
int(len(X_train)*.10))
```

- **DecisionTreeClassifier()** from sklearn.tree fits a classification tree.
- **criterion=** ‘entropy’ specifies the function to measure the split. Default is ‘gini’ for Gini impurity. ‘entropy’ stands for information gain.
- **min_samples_split=** minimum number of samples required to split an internal node. This number is set to be 10% of the sample size.
- The output displays model specifications.

```
# Output
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=49,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Classification Tree in Python – Prediction

```
# Generating Predictions for the model
```

```
y_pred = dtcl.predict(X_test)
y_pred_probs = dtcl.predict_proba(X_test)

cutoff = 0.3
pred_test = np.where(y_pred_probs[:,1] > cutoff, 1, 0)
pred_test
```

```
# Output
```

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

Classification Tree in Python – Confusion Matrix

```
# Confusion Matrix
confusion_matrix(y_test, pred_test, labels=[0, 1])
array([[107,  50],
       [ 14,  39]], dtype=int64)

accuracy_score(y_test, pred_test)
0.6952380952380952

precision_score(y_test, pred_test)
0.43820224719101125

recall_score(y_test, pred_test)
0.7358490566037735
```

- **accuracy_score() =** number of correct predictions out of total predictions
- **precision_score()** = true positives / (true positives + false positives)
- **recall_score()** also known as 'Sensitivity' = true positives / (true positives + false negatives)

```
# Area Under ROC Curve
auc = roc_auc_score(y_test, y_pred_probs[:,1])
print('AUC: %.3f' % auc)
AUC: 0.720
```

Classification Tree in Python – ROC Curve

```
# Area Under ROC Curve
```

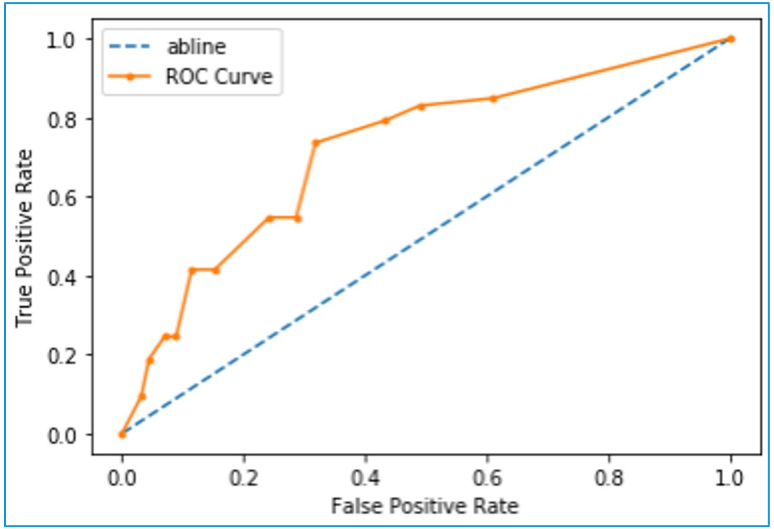
```
DTfpr, DTtpr, thresholds = roc_curve(y_test, y_pred_probs[:,1])

abline_probs = [0 for _ in range(len(y_test))]
abline_auc = roc_auc_score(y_test, abline_probs)
abline_fpr, abline_tpr, _ = roc_curve(y_test, abline_probs)

plt.plot(abline_fpr, abline_tpr, linestyle='--', label='abline')
plt.plot(DTfpr, DTtpr, marker='.', label='ROC Curve')
plt.xlabel('False Positive Rate');plt.ylabel('True Positive Rate')
plt.legend(); plt.show()
```

Classification Tree in Python – ROC Curve

```
# Output
```




```
# Plotting The Tree
dtcl_infgain = DecisionTreeClassifier(criterion='entropy', min_samples_split
= int(len(X_train)*.10))
dtcl_infgain.fit(X_train, y_train)
```

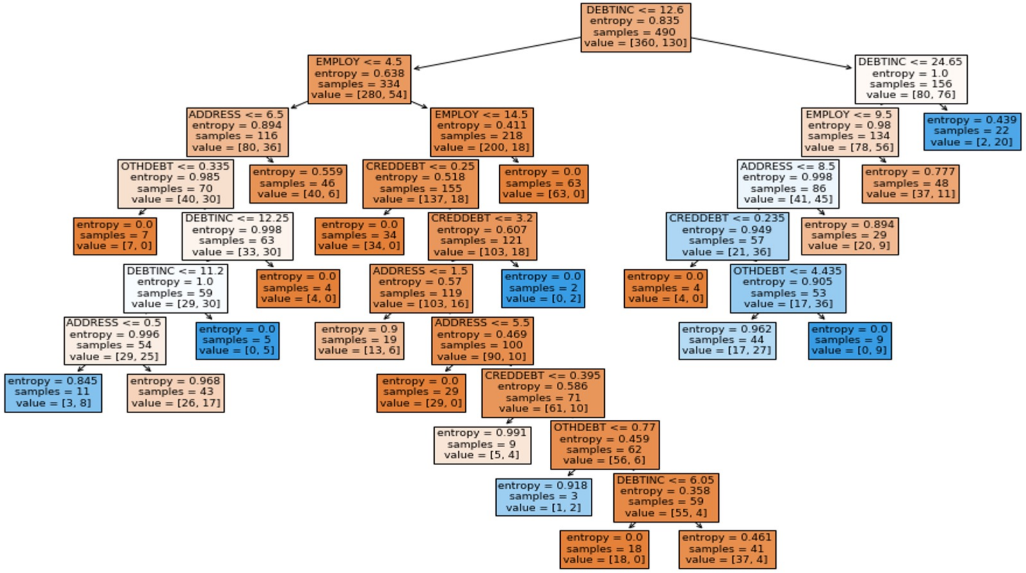
Classification Tree Using Information Gain

```
# Plotting The Tree
```

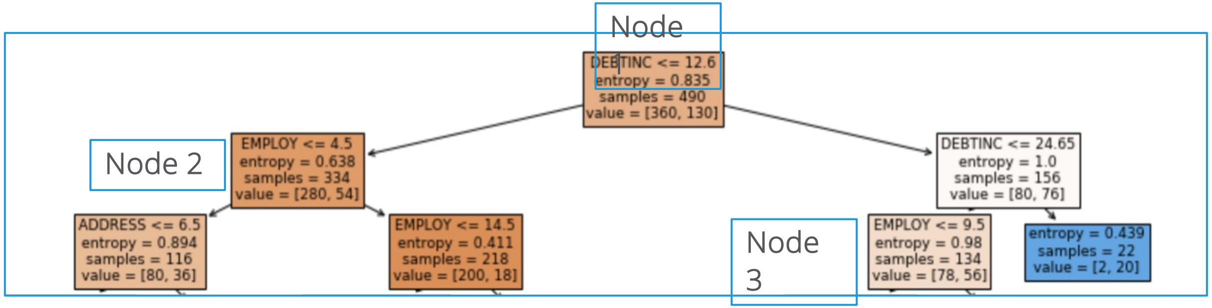
```python
from sklearn.tree import plot_tree
plt.figure(figsize = (16,10))
plot_tree(dtcl_infgain, filled = True, feature_names = list(X.columns))
plt.show();
```

- **plot_tree** is used to plot the decision tree.
- **filled= True** paints nodes to indicate majority class for classification and **feature_names** is used to mention the feature names.

```
# Output
```



Classification Tree Interpretation



Interpretation :

- Due to a large number of continuous predictors, a tree with several nodes and branches is generated.
- Tree starts with all 490 observations (Train set). 360 are non-defaulters (0) and the remaining 130 are defaulters (1).
- DEBTINC is the first split variable, left branch is <=12.6 and right branch is >12.6. 334/490 have DEBTINC<=12.6.
- EMPLOY is the second split on left branch, which further divides 334 obs. into 280 non-defaulters (0) and the remaining 54 as defaulters (1).
- The algorithm progresses till no further variable split is left.

Case Study – Modeling Motor Insurance Claims

Background

- A car insurance company collects range of information from their customers at the time of buying and claiming insurance. The company wishes to check if there any of it can be used to model and predict claim amount

Objective

- To model motor insurance claim amount based on vehicle related information collected at the time of registering and claiming insurance

Available Information

- Sample size is 1000
- Independent Variables: Vehicle Information – Vehicle Age, Engine Capacity, Length and Weight of the Vehicle
- Dependent Variable: Claim Amount

Data Snapshot

Motor_Claims

Independent variables

Dependent variable

Observations

vehage	CC	Length	Weight	claimamt
4	1495	4250	1023	72000
2	1061	3495	875	72000
2	1405	3675	980	50400
7	1298	4090	930	39960
2	1495	4250	1023	106800
1	1086	3565	854	69592.8
4	796	3495	740	38400

Columns	Description	Type	Measurement	Possible values
vehage	Age of the vehicle at the time of claim	Integer	Years	positive values
CC	Engine capacity	Integer	cc	positive values
Length	Length of the vehicle	Integer	mm	positive values
Weight	Weight of the vehicle	Integer	kg	positive values
claimamt	Claim amount	Continuous	INR	positive values

Regression Tree in Python

```
# Importing and Readying the Data for Modeling
motor = pd.read_csv("Motor Claims.csv")
motor.head()
```

```
# Output
```

	vehage	CC	Length	Weight	claimamt
0	4	1495	4250	1023	72000.0
1	2	1061	3495	875	72000.0
2	2	1405	3675	980	50400.0
3	7	1298	4090	930	39960.0
4	2	1495	4250	1023	106800.0

Since all variables are continuous, no further processing is needed.

```
# Creating Data Partitions
X = motor.loc[:,motor.columns != 'claimamt']
y = motor.loc[:, 'claimamt']
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                     test_size = 0.30,
                                                     random_state =
999)
```

Regression Tree in Python

```
# Regression Tree Using MSE
dtreg = DecisionTreeRegressor(min_samples_split =
                              int(len(X_train)*.10))
dtreg.fit(X_train, y_train)
```

- **DecisionTreeRegressor()** from sklearn.tree fits a regression tree.
- **min_samples_split=** minimum number of samples required to split an internal node. This number is set to be 10% of the sample size.
- The output displays model specifications.

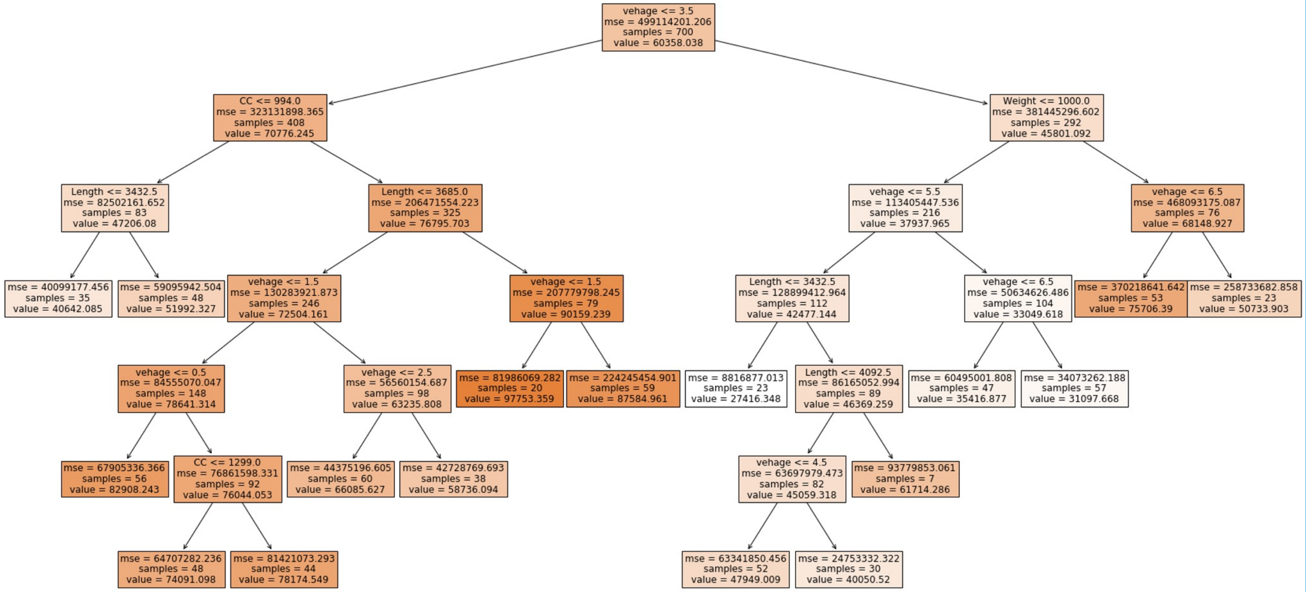
```
# Output
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=70,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

Regression Tree in Python

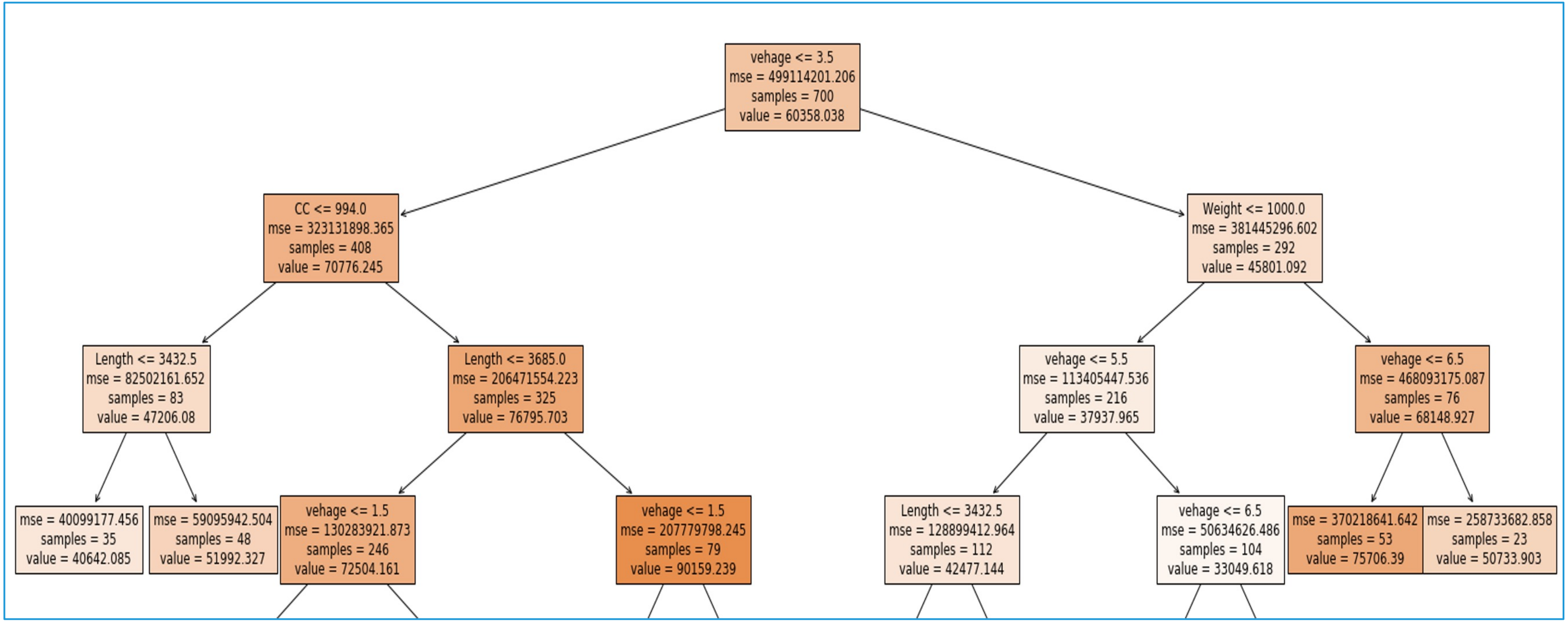
Plotting The Tree

```
plt.figure(figsize = (30,15))
plot_tree(dtreg, filled = True, feature_names = list(X.columns))
plt.show();
```

Output



Regression Tree in Python



Interpretation :

- Tree starts with all 700 training observations, 60358.038 is the average claim amount of these observations.
- vehage is the first split variable, left branch is <=3.5 and right branch is >3.5.
- 408 have vehage <=3.5 which has 70776.246 average claim amount .
- The process continues till there is no variable left for splitting.

Regression Tree in Python

```
# Predictions
y_pred_reg = dtreg.predict(X_test)
y_pred_reg
```

- **predict()** returns predicted regression value for X. Output is an array.

```
# Output
```

```
array([47949.00923077, 74091.09775    , 50733.9026087 , 58736.09431579,
       50733.9026087 , 35416.87659574, 74091.09775    , 31097.668     ,
       58736.09431579, 82908.24257143, 87584.96054237, 87584.96054237,
       51992.32725    , 82908.24257143, 66085.6268     , 75706.38973585,
       74091.09775    , 58736.09431579, 75706.38973585, 31097.668     ,
       31097.668     , 97753.359     , 82908.24257143, 74091.09775    ,
       47949.00923077, 40050.52      , 35416.87659574, 58736.09431579,
       35416.87659574, 87584.96054237, 40050.52      , 35416.87659574,
       75706.38973585, 40642.0848    , 35416.87659574, 31097.668     ,
       31097.668     , 35416.87659574, 40642.0848    , 97753.359     ,
```

Quick Recap

CART in Python

- **`DecisionTreeClassifier`** and **`DecisionTreeRegressor`** from **`sklearn.tree`** library are used for classification and regression respectively.
- **`plot_tree`** from **`sklearn.tree`** library is used for plotting decision tree.