**Decision Tree Models in Python — Build, Visualize, Evaluate**

**Guide and example from MITx Analytics Edge using Python**

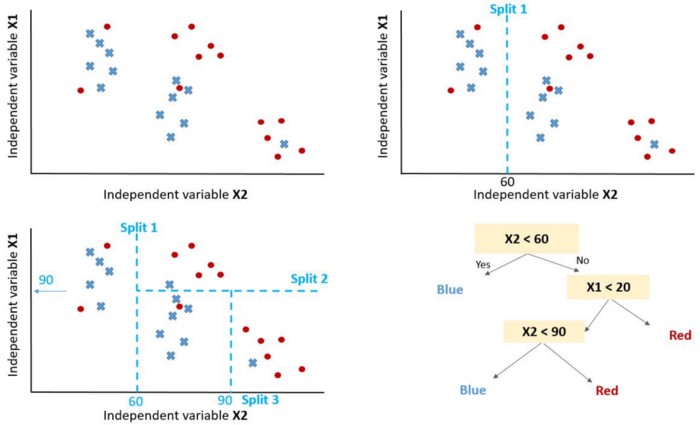
Classification and Regression Trees (CART) can be translated into a graph or set of rules for predictive classification. They help when logistic regression models cannot provide sufficient decision boundaries to predict the label. In addition, decision tree models are more interpretable as they simulate the human decision-making process. In addition, decision tree regression can capture non-linear relationships, thus allowing for more complex models.



Modified by the author based on a Photo by [Johann Siemens](https://unsplash.com/@emben?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

**How do CART models work?**

Consider the case of two independent variables X1 and X2. We want to predict whether the outcome is red or blue. CART tries to split this data into subsets so that each subset is as pure or homogeneous as possible.



Decision tree logic and data splitting — Image by author.

The first split (split1) splits the data in a way that if variable X2 is less than 60 will lead to a blue outcome and if not will lead to looking at the second split (*split2*). *Split2* guides to predicting red when **X1>20** considering **X2<60**. *Split3* will predict blue if **X2<90** and red otherwise.

**How to control the model performance?**

After you select the variables to consider for the model through discipline knowledge or feature selection process you will need to define the optimum number of splits.

the target of splitting is to increase the homogeneity of the outcome from each node. Increase its ability to classify the data. In other words, increase the purity after each split. If we predict blue and red, choose the number of splits that give all blue and all red if possible. Choose the number of splits that will generate pure results.

A pure node is one that results in perfect prediction.

But how to quantify purity after splits to make sure we have pure nodes as much as possible.

We aim at reducing uncertainty after each split. A bad split will make the outcome 50% blue and 50% red. the perfect split will give 100% blue for example.

To measure how the splits are behaving in terms of increasing information after the split we can rely on the following measures:

1 — **Entropy** [entropy = -1\*sum(p\*log(p)) ]

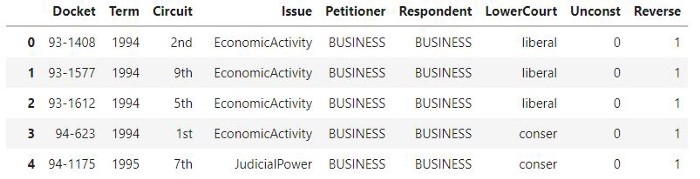
2 — **Gini impurity** [Gini = sum(p(1-p)), where p is the proportion of misclassified observation within the sub partition]

**Example: Predicting Judge Stevens Decision**

The target is to predict whether or not Justice Steven voted to reverse the court decision with **1 means voted to reverse the decision** and **0 means he affirmed** the decision of the court.

The code and the data are available at [**GitHub**](https://github.com/muadelm/MITx_Analytics_Edge/tree/main/CART%20model_Supreme%20Court%20Forecasting).

The data frame appears as below with the target variable (Reverse).



Judge Steven Court Decision — Image by author

**Important Note**: Decision Tree (DT) can handle both continuous and numeric variables. But if you are using Python Scikit Learn, you might get a ValueError for categorical.

The features have many categorical values that we will convert into numerical values using the function below:

**def** convert\_cat(df,col):  
 """  
 input: dataframe and col list of categorical columns  
 output: dataframw with numerical values  
 """  
 **for** c **in** col:  
 item\_list **=** df[c]**.**unique()**.**tolist()  
 enum**=**enumerate(item\_list)  
 d **=** dict((j,i) **for** i,j **in** enum)  
 print(c)  
 print(d)  
   
 df[c]**.**replace(d, inplace**=True**)  
 **return** dfconvert\_cat(df,['Circuit', 'Issue', 'Petitioner', 'Respondent',  
 'LowerCourt'])

Split the data into training and testing

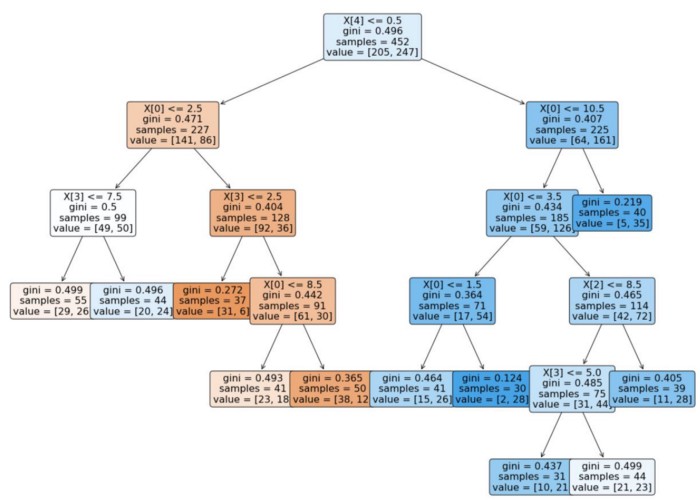
X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, Y, test\_size**=**0.2, random\_state**=**0)

Build a decision tree model on the training data

clf **=** tree**.**DecisionTreeClassifier('gini', min\_samples\_leaf**=**30, random\_state**=**0)  
clf **=** clf**.**fit(X\_train, y\_train)

Plot the decision tree model

**from** sklearn **import** tree *# for decision tree models*plt**.**figure(figsize **=** (20,16))  
tree**.**plot\_tree(clf, fontsize **=** 16,rounded **=** **True** , filled **=** **True**);



Decision tree model — Image by author

Use the classification report to assess the model.

report **=** classification\_report(predTree, y\_test)  
print(report)

**References**

MITx Analytics Course on edX