Decision Trees

This lesson will focus on training decision tree models in Python.

WE'LL COVER THE FOLLOWING

Decision trees

In the first lesson of this chapter, we talked about how linear regression models focus only on linear relationships between the dependent and independent variables; they fail to capture nonlinear relationships. Decision trees are made to capture nonlinear relationships.

Decision trees model data as a **tree** of hierarchical branches. It is a flowchart-like structure in which each internal node represents a *test* on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from the root to the leaf represent classification rules. Decision Trees can adapt to both regression and classification tasks.

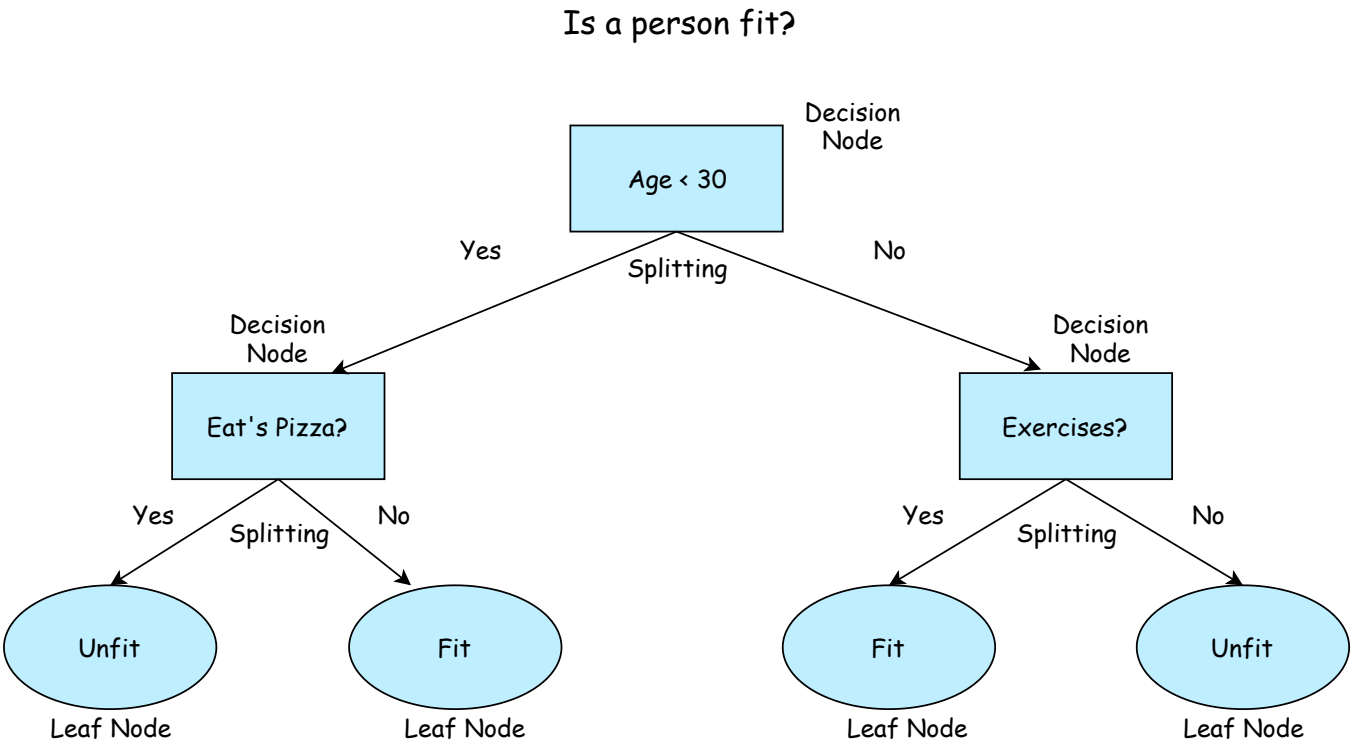
Common terms used with Decision trees:

- **Root node**: It represents the entire population or sample, and this further gets divided into two or more homogeneous sets.
- **Splitting**: It is a process of dividing a node into two or more sub-nodes.
- **Decision node**: When a sub-node splits into further sub-nodes, then it is called a decision node.
- **Leaf/Terminal node**: Nodes that do not split are called Leaf or Terminal

node.

- **Pruning**: When we remove sub-nodes of a decision node, this process is called pruning. It is the opposite process of splitting.
- **Branch/Sub-tree**: A subsection of the entire tree is called branch or sub-tree.
- **Parent** and **Child** node: A node, which is divided into sub-nodes is called a parent node of sub-nodes, whereas sub-nodes are the children of the parent node.

A very common example that is given in the context of decision trees is that we want to classify a person as unfit or fit based on the person's age, whether he/she eats pizza, and whether he/she exercises in the morning. A decision tree of this could be:



From the diagram, we can see that at every node there is a yes/no decision. We keep moving in the tree until we reach the leaf nodes, where the observation is classified into a class.

Decision trees in Python

We will be using the Audit Risk Dataset of different firms. We will be performing the binary classification task of predicting whether a company is fraudulent or not. The dataset has the following attributes:

```
Audit Risk Dataset
# Sector_score
# LOCATION_ID
# PARA_A
# SCORE_A
# PARA_B
# SCORE_B
# TOTAL
# numbers
# Marks
# Money_Value
# MONEY_Marks
# District
# Loss
# LOSS_SCORE
# History
# History_score
# Score
# Risk : BInary Target Variable
```

In python, the `sklearn` package has a lot of machine learning models implemented as classes. We will be importing the `DecisionTreeClassifier` class from `sklearn.tree`. We will also be importing a function named `train_test_split` from `sklearn.model_selection` that divides our dataset into train and test sets.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report

df = pd.read_csv('audit_data.csv')

# MAKE DATA
X = df.drop(columns = ['Risk','LOCATION_ID'])
Y = df[['Risk']]

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2, random_state = 3)

# MAKE MODEL
d_tree = DecisionTreeClassifier()
d_tree.fit(X_train,Y_train)

# CALCULATE AND PRINT RESULTS
preds = d_tree.predict(X_test)
acc = accuracy_score(y_true = Y_test,y_pred = preds)
print(acc)
print(classification_report(y_true = Y_test,y_pred = preds))
```

After we read the data in **line 6**, we separate our target variable as `Y`. We drop the `LOCATION_ID` column since it would not provide any useful information to the model. To split the data into training and test sets, we use the function `train_test_split`. We provide our inputs, `X`, and the labels, `Y`, to the function in **line 12**. We also provide the test set size as `test_size`. 0.2 implies that 20% data will be included in the testing set, while the rest 80% will form the training set. The function outputs 4 items that we can retrieve directly into 4 variables. These are:

1. Inputs for the training data that we store in `X_train`
2. Inputs for the testing data that we store in `X_test`
3. Labels for the training data that we store in `Y_train`
4. Labels for the testing data that we store in `Y_test`

In **line 15**, we make our Decision Tree model just like we did in the last lesson. We call `DecisionTreeClassifier` without any arguments. Then in the next line, we call the `fit` function of the model. We provide the training examples and labels to the function. After this, we need to evaluate our model. Therefore, we use the `predict` function of the model in **line 19** to store predictions in `preds`. We give the testing inputs `X_test` to `predict` as an argument. We use the `accuracy_score` function to measure the accuracy of the predictions. We print the accuracy with the classification report, which we obtained by using the `classification_report` function, in the last two lines.

From the outputs, we can see that the model performs excellent on the testing data. The model has learned all patterns and relationships in the dataset and gives correct results 100% of the time.

> Note that this was a relatively easier task for the model. For bigger and more complex datasets, we might not get 100% accuracy.

This brings an end to this lesson where we looked at Decision Trees. In the next lesson, we will look at another machine learning model called *Random Forests*.