Decision_trees

October 13, 2025

1 Decision Trees

Estimated time needed: **25** minutes

1.1 Objectives

After completing this lab you will be able to:

- Develop a classification model using Decision Tree Algorithm
- Apply Decision Tree classification on a real world dataset.

1.2 Introduction

This lab explores decision tree classification, a powerful machine learning technique for making data-driven decisions. You will learn to build, visualize, and evaluate decision trees using a real-world dataset. The dataset used in this lab is that of Drug prediction based on the health parameters of a patient.

1.2.1 Importing Libraries

First, to make sure that the required libraries are available, execute the cell below.

[1]:
```
!pip install numpy==2.2.0
!pip install pandas==2.2.3
!pip install scikit-learn==1.6.0
!pip install matplotlib==3.9.3
```

```
Collecting numpy==2.2.0
  Downloading
numpy-2.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(62 kB)
Downloading
numpy-2.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.1 MB)
                         16.1/16.1 MB
177.3 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-2.2.0
Collecting pandas==2.2.3
  Downloading
pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
```

```
(89 kB)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-
packages (from pandas==2.2.3) (2.2.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.12/site-packages (from pandas==2.2.3) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas==2.2.3) (2024.2)
Collecting tzdata>=2022.7 (from pandas==2.2.3)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas==2.2.3) (1.17.0)
Downloading
pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.7
MB)
                         12.7/12.7 MB
171.1 MB/s eta 0:00:00
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: tzdata, pandas
Successfully installed pandas-2.2.3 tzdata-2025.2
Collecting scikit-learn==1.6.0
  Downloading scikit_learn-1.6.0-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Requirement already satisfied: numpy>=1.19.5 in /opt/conda/lib/python3.12/site-
packages (from scikit-learn==1.6.0) (2.2.0)
Collecting scipy>=1.6.0 (from scikit-learn==1.6.0)
  Downloading
scipy-1.16.2-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata
(62 kB)
Collecting joblib>=1.2.0 (from scikit-learn==1.6.0)
  Downloading joblib-1.5.2-py3-none-any.whl.metadata (5.6 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn==1.6.0)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading
scikit_learn-1.6.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(13.1 MB)
                         13.1/13.1 MB
138.3 MB/s eta 0:00:00
Downloading joblib-1.5.2-py3-none-any.whl (308 kB)
Downloading
scipy-1.16.2-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (35.7
MB)
                         35.7/35.7 MB
131.3 MB/s eta 0:00:0000:01
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.5.2 scikit-learn-1.6.0 scipy-1.16.2
threadpoolctl-3.6.0
Collecting matplotlib==3.9.3
```

```
  Downloading matplotlib-3.9.3-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib==3.9.3)
  Downloading contourpy-1.3.3-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib==3.9.3)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib==3.9.3)
  Downloading fonttools-4.60.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl.metadata (111 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib==3.9.3)
  Downloading kiwisolver-1.4.9-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-
packages (from matplotlib==3.9.3) (2.2.0)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib==3.9.3) (24.2)
Collecting pillow>=8 (from matplotlib==3.9.3)
  Downloading pillow-11.3.0-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (9.0 kB)
Collecting pyparsing>=2.3.1 (from matplotlib==3.9.3)
  Downloading pyparsing-3.2.5-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from matplotlib==3.9.3) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib==3.9.3) (1.17.0)
Downloading
matplotlib-3.9.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.3
MB)
                        8.3/8.3 MB
142.7 MB/s eta 0:00:00
Downloading
contourpy-1.3.3-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (362
kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.60.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl (4.9 MB)
                        4.9/4.9 MB
69.3 MB/s eta 0:00:00
Downloading
kiwisolver-1.4.9-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (1.5
MB)
                        1.5/1.5 MB
33.0 MB/s eta 0:00:00
Downloading
pillow-11.3.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (6.6
```

```
MB)
                                      6.6/6.6 MB
96.7 MB/s eta 0:00:00
Downloading pyparsing-3.2.5-py3-none-any.whl (113 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler,
contourpy, matplotlib
Successfully installed contourpy-1.3.3 cycler-0.12.1 fonttools-4.60.0
kiwisolver-1.4.9 matplotlib-3.9.3 pillow-11.3.0 pyparsing-3.2.5
```

Now import the required libraries for this lab.

[3]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import metrics

%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

1.2.2 About the dataset

Imagine that you are a medical researcher compiling data for a study. You have collected data about a set of patients, all of whom suffered from the same illness. During their course of treatment, each patient responded to one of 5 medications, Drug A, Drug B, Drug C, Drug X and Drug Y.

Part of your job is to build a model to find out which drug might be appropriate for a future patient with the same illness. The features of this dataset are the Age, Sex, Blood Pressure, and Cholesterol of the patients, and the target is the drug that each patient responded to.

It is a sample of a multiclass classifier, and you can use the training part of the dataset to build a decision tree, and then use it to predict the class of an unknown patient or to prescribe a drug to a new patient.

```
<h2>Downloading the Data</h2>
To download the data, we will use !wget to download it from IBM Object Storage.
```

[5]:
```
path= 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
 ↪IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/
 ↪drug200.csv'
my_data = pd.read_csv(path)
my_data
```

[5]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC

```
2     47   M     LOW        HIGH   10.114  drugC
3     28   F  NORMAL        HIGH    7.798  drugX
4     61   F     LOW        HIGH   18.043  drugY
..   ...  ..     ...         ...      ...     ...
195   56   F     LOW        HIGH   11.567  drugC
196   16   M     LOW        HIGH   12.006  drugC
197   52   M  NORMAL        HIGH    9.894  drugX
198   23   M  NORMAL      NORMAL   14.020  drugX
199   40   F     LOW      NORMAL   11.349  drugX

[200 rows x 6 columns]
```

1.3 Data Analysis and pre-processing

You should apply some basic analytics steps to understand the data better. First, let us gather some basic information about the dataset.

[6]:
```
my_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

This tells us that 4 out of the 6 features of this dataset are categorical, which will have to be converted into numerical ones to be used for modeling. For this, we can make use of **LabelEncoder** from the Scikit-Learn library.

[7]:
```
label_encoder = LabelEncoder()
my_data['Sex'] = label_encoder.fit_transform(my_data['Sex'])
my_data['BP'] = label_encoder.fit_transform(my_data['BP'])
my_data['Cholesterol'] = label_encoder.fit_transform(my_data['Cholesterol'])
my_data
```

[7]:
```
     Age  Sex  BP  Cholesterol  Na_to_K   Drug
0     23    0   0            0   25.355  drugY
1     47    1   1            0   13.093  drugC
2     47    1   1            0   10.114  drugC
3     28    0   2            0    7.798  drugX
4     61    0   1            0   18.043  drugY
```

```
..   …  …  ..          …      …    …
195   56    0   1            0   11.567  drugC
196   16    1   1            0   12.006  drugC
197   52    1   2            0    9.894  drugX
198   23    1   2            1   14.020  drugX
199   40    0   1            1   11.349  drugX

[200 rows x 6 columns]
```

With this, you now have 5 parameters that can be used for modeling and 1 feature as the target variable. We can see from comparison of the data before Label encoding and after it, to note the following mapping. For parameter 'Sex' : $M \rightarrow 1, F \rightarrow 0$ For parameter 'BP' : $High \rightarrow 0, Low \rightarrow 1, Normal \rightarrow 2$ For parameter 'Cholesterol' : $High \rightarrow 0, Normal \rightarrow 1$

You can also check if there are any missing values in the dataset.

[8]:
```
my_data.isnull().sum()
```

[8]:
```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

This tells us that there are no missing values in any of the fields.

To evaluate the correlation of the target variable with the input features, it will be convenient to map the different drugs to a numerical value. Execute the following cell to achieve the same.

[11]:
```
custom_map = {'drugA':0,'drugB':1,'drugC':2,'drugX':3,'drugY':4}
my_data['Drug_num'] = my_data['Drug'].map(custom_map)
my_data
```

[11]:
```
     Age  Sex  BP  Cholesterol  Na_to_K   Drug  Drug_num
0     23    0   0            0   25.355  drugY         4
1     47    1   1            0   13.093  drugC         2
2     47    1   1            0   10.114  drugC         2
3     28    0   2            0    7.798  drugX         3
4     61    0   1            0   18.043  drugY         4
..   …  …  ..          …      …    …       …
195   56    0   1            0   11.567  drugC         2
196   16    1   1            0   12.006  drugC         2
197   52    1   2            0    9.894  drugX         3
198   23    1   2            1   14.020  drugX         3
199   40    0   1            1   11.349  drugX         3

[200 rows x 7 columns]
```

You can now use the **corr()** function to find the correlation of the input variables with the target variable.

Practice question Write the code to find the correlation of the input variables with the target variable and identify the features most significantly affecting the target.

[16]:
```
# your code here
my_data.drop("Drug", axis = 1).corr()['Drug_num']
```

[16]:
```
Age            -0.004828
Sex            -0.098573
BP              0.372868
Cholesterol     0.055629
Na_to_K         0.589120
Drug_num        1.000000
Name: Drug_num, dtype: float64
```

Click here for the solution

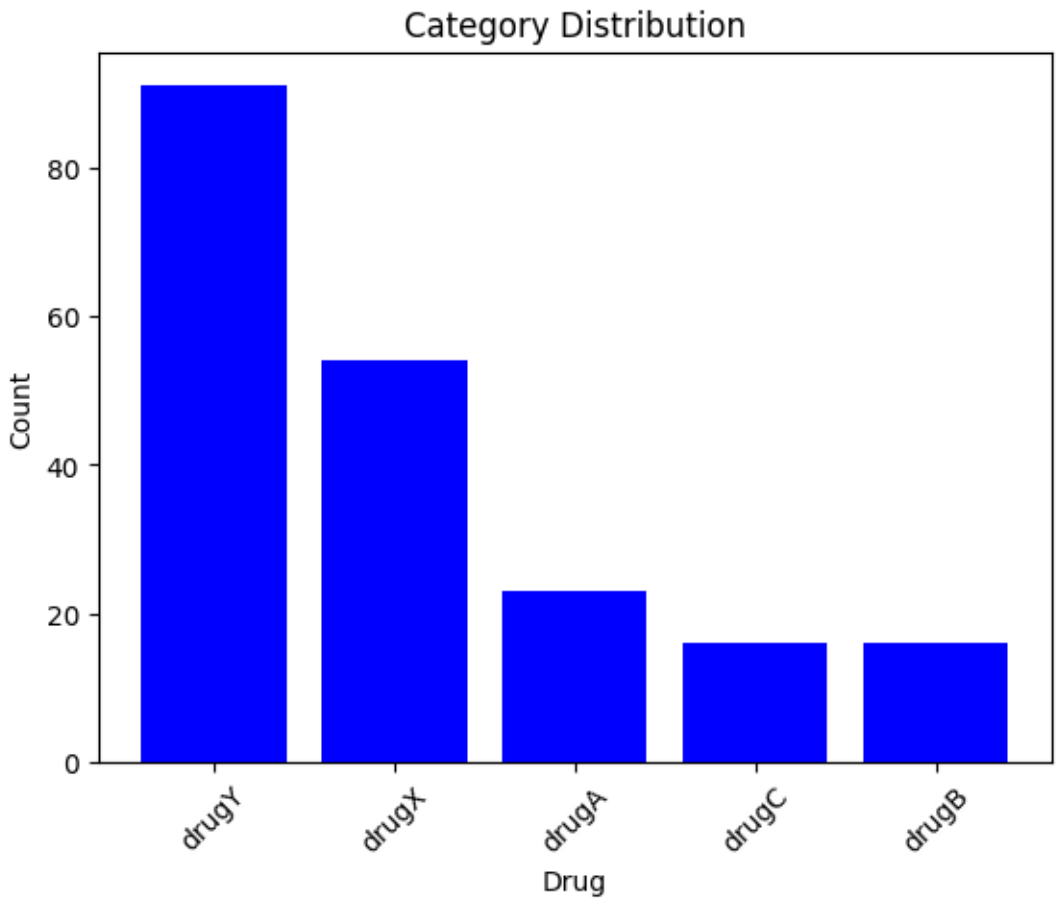
```
my_data.drop('Drug',axis=1).corr()['Drug_num']
```

This shows that the drug recommendation is mostly correlated with the `Na_to_K` and `BP` features.

We can also understand the distribution of the dataset by plotting the count of the records with each drug recommendation.

[17]:
```
category_counts = my_data['Drug'].value_counts()

# Plot the count plot
plt.bar(category_counts.index, category_counts.values, color='blue')
plt.xlabel('Drug')
plt.ylabel('Count')
plt.title('Category Distribution')
plt.xticks(rotation=45)  # Rotate labels for better readability if needed
plt.show()
```



This shows us the distribution of the different classes, clearly indicating that Drug X and Drug Y have many more records in comparison to the other 3.

1.4 Modeling

For modeling this dataset with a Decision tree classifier, we first split the dataset into training and testing subsets. For this, we separate the target variable from the input variables.

[23]:
```
y = my_data['Drug']
X = my_data.drop(['Drug','Drug_num'], axis=1)
```

Now, use the **train_test_split()** function to separate the training data from the testing data. We can make use of 30% of the data for testing and the rest for training the Decision tree.

[24]:
```
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y,␣
↪test_size=0.3, random_state=32)
```

You can now define the Decision tree classifier as **drugTree** and train it with the training data.

[25]:
```
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
```

[26]:
```
drugTree.fit(X_trainset,y_trainset)
```

[26]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

1.4.1 Evaluation

Now that you have trained the decision tree, we can use it to generate the predictions on the test set.

[29]:
```
tree_predictions = drugTree.predict(X_testset)
```

We can now check the accuracy of our model by using the accuracy metric.

[28]:
```
print("Decision Trees's Accuracy: ", metrics.accuracy_score(y_testset,␣
↪tree_predictions))
```

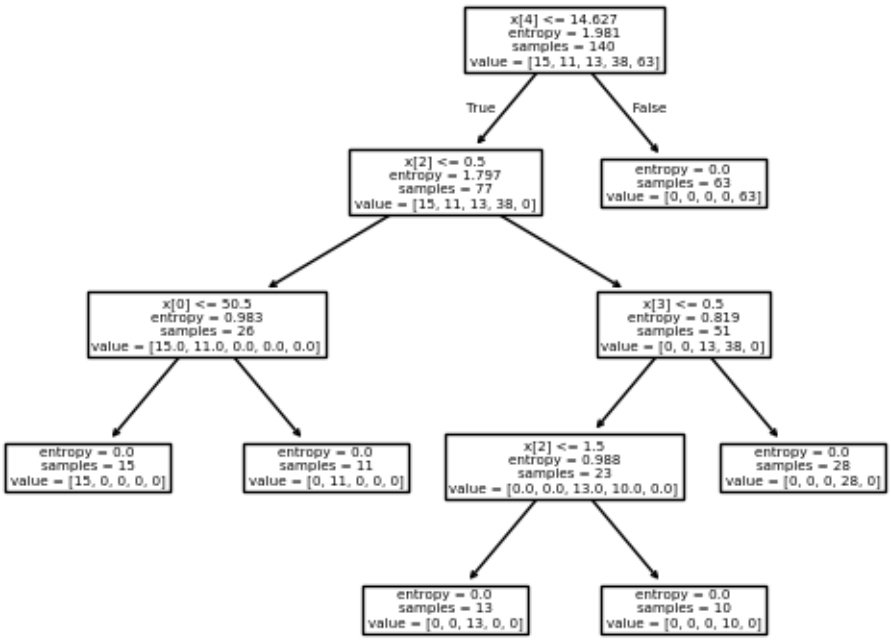
```
Decision Trees's Accuracy:  0.9833333333333333
```

This means that the model was able to correctly identify the labels of 98.33%, i.e. 59 out of 60 test samples.

1.4.2 Visualize the tree

To understand the classification criteria derived by the Decision Tree, we may generate the tree plot.

[30]:
```
plot_tree(drugTree)
plt.show()
```



From this tree, we can derive the criteria developed by the model to identify the class of each training sample. We can interpret them by tracing the criteria defined by tracing down from the root to the tree's leaf nodes.

For instance, the decision criterion for Drug Y is $Na\_to\_K > 14.627$.

Practice Question: Along similar lines, identify the decision criteria for all other classes.

[33]:
```
Drug A : $Na\_to\_K <= 14.627,\ BP = High,\ Age <= 50.5$
Drug B : $Na\_to\_K <= 14.627,\ BP = High,\ Age > 50.5$
Drug C : $Na\_to\_K <= 14.627,\ BP = Low,\ Cholesterol <= High$
Drug X : $Na\_to\_K <= 14.627,\ BP = Normal,\ Cholesterol = High$
```

```
  Cell In[33], line 1
    Drug A : $Na\_to\_K <= 14.627,\ BP = High,\ Age <= 50.5$
         ^
SyntaxError: invalid syntax
```

Click here for the solution

Drug A : $Na\_to\_K <= 14.627,\ BP = High,\ Age <= 50.5$ Drug B : $Na\_to\_K <= 14.627,\ BP = High,\ Age > 50.5$ Drug C : $Na\_to\_K <= 14.627,\ BP = Low,\ Cholesterol <= High$ Drug X : $Na\_to\_K <= 14.627,\ BP = Normal,\ Cholesterol = High$

Practice Question: If the max depth of the tree is reduced to 3, how would the performance of the model be affected?

[36]:
```
# your code here
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 3)
drugTree.fit(X_trainset,y_trainset)
tree_predictions = drugTree.predict(X_testset)
print("Decision Trees's Accuracy: ", metrics.accuracy_score(y_testset,␣
 ↪tree_predictions))
```

```
Decision Trees's Accuracy:  0.9833333333333333
```

Click here for the solution

```
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 3)
drugTree.fit(X_trainset,y_trainset)
tree_predictions = drugTree.predict(X_testset)
print("Decision Trees's Accuracy: ", metrics.accuracy_score(y_testset, tree_predictions))
```

1.4.3 Congratulations! You're ready to move on to your next lesson!

1.5 Author

Abhishek Gagneja ### Other Contributors Jeff Grossman

<!– ## Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2025-05-13	3.1	Anita Verma	Corrected the decision tree in the solution cell
2024-10-31	3.0	Abhishek Gagneja	Rewrite
2020-11-03	2.1	Lakshmi	Made changes in URL
2020-11-03	2.1	Lakshmi	Made changes in URL
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab