Data Mining and Machine Learning(SBEN454)

Assignment-6.1

Student Name: Abdulrehman Mahmoud Suliman

Student ID: 1180140

Submitted to: Eng. Peter Salah

Pre-Processing :

- Check that there is no null values.
- Drop id column
- Remove rows with negative values
- Consider any height < 90 and weight < 30 outliers and remove them
- Create new column for body mass index after calculating it from the height and weight
- Drop the height and weight columns
- Calculate the Age with years then categorize them in to three categories
    - catigory 1 30-50 years
    - catigory 2 50-60 years
    - catigory 3 >60 years
- Then remove the outliers from the ap_hi and ap_lo so that
    - ap_hi be from 90 to 180
    - ap_lo be from 60 to 110
    - 1 - Normal ap_hi 90-120 and ap_low 60-80
    - 2 - Elevated ap_hi 120-130 and ap_lo 60-80
    - 3 - At Risk (prehypertension) ap_hi: 130-140 and ap_lo : 80-90
    - 4- High Blood Pressure (hypertension) ap_hi >140 and ap_lo >90
- Tried to combine the ap_hi and ap_lo in one record but found that I will loose to much records that won't fall in any of the categories

- Check that all records are unique and there is no duplication.

Node Class:

This class will be used in the decision tree class as a tree in made up of nodes.

Each node contains information about the:

- attribute or feature
- Threshold upon which the record will be in the left or right tree
- Left data
- Right data
- Information gain of the attribute
- Decision if it is a leaf node

Decision Tree Class:

Functions:

- __init__:
  - Initialize the root and defines maximum depth
- Entropy:
  - Calculates the entropy for all classes in a given dataset
  - Returns entropy
- calculateGain:
  - calculates the information gain of a given attribute using the entropy function
  - returns gain

- **getPartition:**
    - **This function loop over all features and unique categories in the feature column to get the best feature to be represented in this node and the best threshold for each feature for the categories upon which we can divide the data to left and right subtrees**
    - **Return partition list containing:**
        - **Feature**
        - **Threshold of partitioning data**
        - **Left dataset**
        - **Right dataset**
        - **Information Gain**
- **buildTree:**
    - **This function builds the tree by partitioning the data then checks that the gain of the chosen feature is a positive value and sets the nodes and for the last node or the leaf node it sets one additional thing with is the decision.**
    - **Returns Node**
- **fit:**
    - **concatenates the labels column and the data entered by the user and calls the buildTree function.**
- **getDecision:**
    - **recursively calls itself and get the decisions for the left and right subtree for each node until it reaches the leaf node and gets the final decision.**

- **Predict:**
    - **This function used to get the test inputs from the user and passes it to the getDecision function to get the predictions.**

Why chose this approach?

After searching about how the sklearn library works specially if we have more than two children I found that sklearn only supports binary splitting for many reasons one of them is to support more than one splitting criteria like the gini impurity which only supports the binary splits and it wouldn't make any problem as a series of binary splits can model any number of simultaneous splits.

Comparing the results with sklearn:

By predicting the test samples using the built-in function and the from scratch class I found that the results are nearly the same.

From scratch:

Accuracy = 0.7176456753101664

Built-in :

Accuracy = 0.7171752807667431