NAME: Bhavesh Prashant Chaudhari ,
Kunal Bhatia,
Ayush Bodade

UID: 2021300018,
2021300010,
2021300015

Data Structures Mini-Project

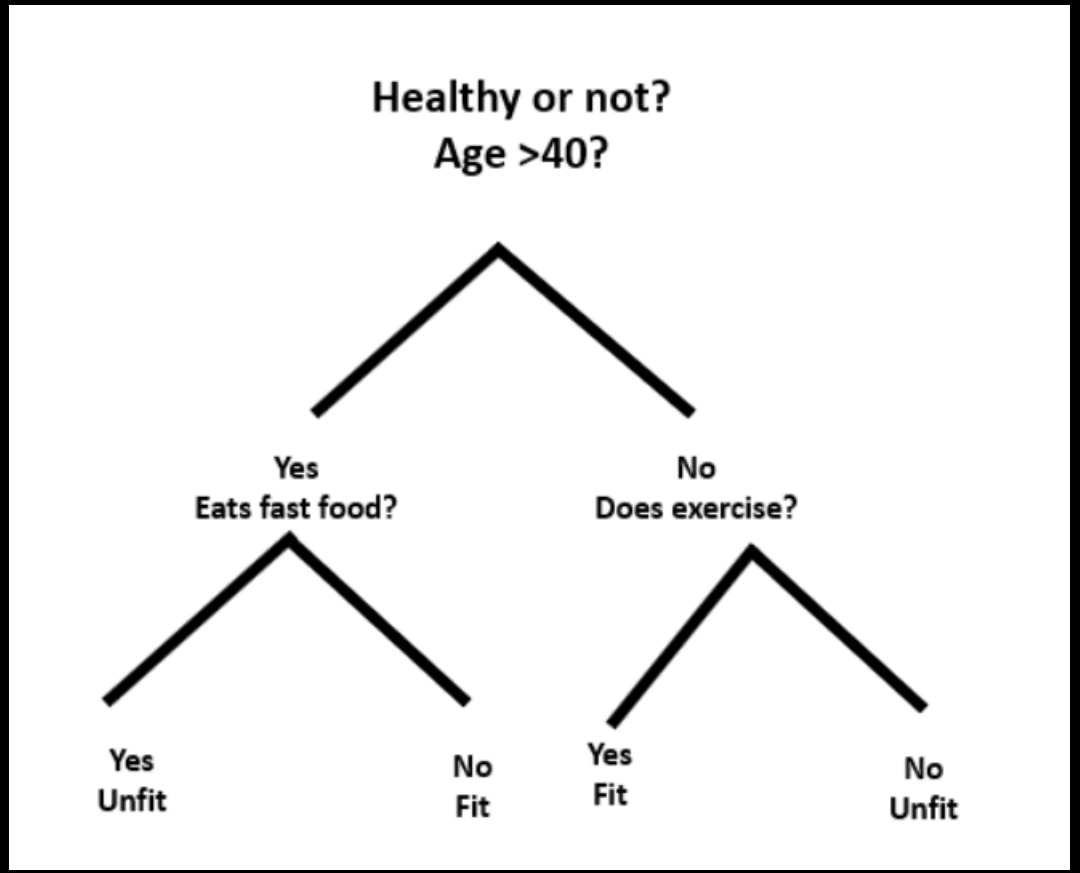
Topic: Decision Tree

Teacher In-Charge: Anand Godbole

What is Decision Tree?

Decision Tree is a Supervised ML that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.

Example:
Prediciting if a person is fit or unfit based on what he eats,if he exercise and his age.



Construction of Decision Tree

A tree can be “learned” by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of a decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high-dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

HOW DECISION TREE MAKES DECISION?

Each internal node of tree makes a decision with the help of concept known as information gain. Information gain is defined as the difference between entropy of parent node and sum of weighted entropy of child nodes.

Entropy is the measure of uncertainity.It is calculated using following formula:

Entropy(E(y)) = -p(y) * log(p(y))

where p(y) is the probability of getting certain label y(output) after the spliting is done.

Therefore Information Gain is given as:

IG = E(parent) - E(children)

Splitting which gives highest information gain is choosen.

Implementing Decision Tree In Python From Scratch

Creating a Node Class:

```python
class Node:
    def __init__(self,left=None,right=None,threshold=None,feature=None,*,value=None):
        self.left = left
        self.right = right
        self.threshold = threshold
        self.feature = feature
        self.value = value

    def is_leaf_node(self):
        return self.value is not None
```

-Left and Right variables are used to store the left and right child of the parent node.
-Threshold variable is used to set threshold of the node.If the threshold criteria is satisfed then the data points statisfying the criteria go to the left child of parent.Similarly data points which do not satisfy the threshold go to the right child of parent.
-Feature variable stores the feature which was used for splitting.
-Value store the output.Only leaf nodes have appropriate values while internal nodes have NULL value indicating that it is not a leaf node.
-The is_leaf_node function is used to know if the node is leaf node or not.

Creating Tree:

```
class DecisionTree:
    def __init__(self,max_depth=100):
        self.max_depth = max_depth
        self.root = None


    def fit(self,X,y):
        self.root = self._create_tree(X,y)

    def _create_tree(self, X, y, depth=0):
        sample_count, feature_count = X.shape
        label_count = len(np.unique(y))

        if label_count == 1 or depth>=self.max_depth:
            leaf_value = self._most_common(y)
            return Node(value = leaf_value)

        best_feature, threshold = self._best_split(X, y, feature_count)
        left_idxs = np.where(X[:, best_feature] <= threshold)[0]
        right_idxs = np.where(X[:, best_feature] > threshold)[0]
        left = self._create_tree(X[left_idxs, :], y[left_idxs], depth+1)
        right = self._create_tree(X[right_idxs, :], y[right_idxs], depth+1)

        return Node(left, right, threshold, best_feature)
```

Creating tree recursively by splitting dataset.Splitting is done till we obtain pure node i.e a leaf node with only one output or till the depth of the tree reaches max depth value.

Finding Best Split:

```
def _best_split(self,X,y,feature_count):
    best_info_gain = -1
    split_idx, split_threshold = None, None

    for feature_idx in range(feature_count):
        X_column = X[:, feature_idx]
        thresholds = np.unique(X_column)

        for threshold in thresholds:
            info_gain = self._cal_info_gain(X_column,y,threshold)
            if(info_gain > best_info_gain):
                best_info_gain = info_gain
                split_idx = feature_idx
                split_threshold = threshold
    return split_idx,split_threshold
```

Calculating Information Gain:

```
def _cal_info_gain(self,X_column,y,threshold):
    parent_entropy = self._entropy(y)
    left_idxs = np.where(X_column <= threshold)[0]
    right_idxs = np.where(X_column > threshold)[0]
    left_size, right_size = left_idxs.size, right_idxs.size
    n = len(y)
    left_entropy, right_entropy = self._entropy(y[left_idxs]), self._entropy(y[right_idxs])
    weighted_child_entropy = (left_size/n)*left_entropy + (right_size/n)*right_entropy
    info_gain = parent_entropy - weighted_child_entropy
    return info_gain
```

Finding Entropy of node:

```
def _entropy(self,y):
    unique_y = np.unique(y)
    cal = []
    for i in unique_y:
        p_num = (np.where(y == i))[0].size
        p_den = len(y)
        p = p_num / p_den
        cal.append(p*np.log2(p))
    return -np.sum(cal)
```

Traversing tree to predict the output after training the model

```
def predict(self, X):
    return np.array([self._traverse_tree(x, self.root) for x in X])


def _traverse_tree(self, x, node):
    if node.is_leaf_node():
        return node.value

    if(x[node.feature] > node.threshold):
        return self._traverse_tree(x, node.right)
    return self._traverse_tree(x, node.left)
```

Loading the dataset and training the model:

```
import pandas as pd
from sklearn import datasets
data = pd.read_csv("E:/Sem3/DS_Mini/heart/heart.csv")
```

```
import numpy as np
from sklearn.model_selection import train_test_split

X, y = data.values[:, 0:13], data.values[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

clf = DecisionTree(max_depth=10)
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)

def accuracy(y_test, y_pred):
    return np.sum(y_test == y_pred) / len(y_test)

acc = accuracy(y_test, predictions)
print("Accuracy of model: ",acc)
print("\nSome Values predicted by model:")
results = np.c_[y_test, predictions]
compared_data = pd.DataFrame(results, columns=["Actual Value", "Predicted Value"])
print(compared_data.head(10))
```

Dataset used: Heart Disease

Predicitons by model:

```
Accuracy of model:  0.819672131147541

Some Values predicted by model:
   Actual Value  Predicted Value
0           0.0              0.0
1           0.0              0.0
2           1.0              0.0
3           0.0              0.0
4           1.0              0.0
5           1.0              1.0
6           1.0              1.0
7           0.0              0.0
8           0.0              0.0
9           1.0              0.0
```

1 means positive(has heart disease)
0 means negative(no heart disease)

After training the accuray was found out to be 82% which is pretty decent.

Concept used: Binary Tree and recursion.

THANK YOU!!