DOKUZ EYLÜL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME 3402 CONCENPTS OF PROGRAMING LANGUAGES

Decision Tree with Python Programming Language

By:

2022510013 – Yasamin Valishariatpanahi (GO)

2022510046 – Sara Mahyanbakhshayesh (JAVA)

2021510070 – Ege Yıldırım (PYTHON)

Lecturer: Doç. Dr. YUNUS DOĞAN

Spring 2025
İzmir

INTRODUCTION

Decision trees provide an explicit approach to solving the decision-making situation of a set of outputs through specific inputs.

Key components such as entropy (used to measure the purity of the data) and information gain (used to determine the most informative attribute for splitting) were calculated manually without the help of external libraries. Based on these calculations, the decision tree was also built manually step by step (using Python's Node library).

The end result is a command line application with the following features:

- Reads input data from CSV or TXT files

- Calculates and displays entropy and information gain at each decision point

- Manually builds the decision tree according to the calculated values

- Prints the tree in a clear, hierarchical text format

- Accepts new user input to make predictions using the generated tree.

-Creates a PNG image of the tree based on user input

This project deepened the understanding of how decision trees work and provided valuable experience with the Python programming language.

Building the Decision Tree Step by Step

ENTROPY: UNDERSTANDING DATA IMPURITY IN DECISION TREES

Entropy is a fundamental concept from information theory that quantifies the amount of uncertainty or impurity in a dataset. In the context of decision trees, entropy is used to measure

how mixed the class labels are within a given subset of data. It serves as a criterion to determine the quality of a split: the lower the entropy, the purer the subset.

A perfectly pure dataset—where all instances belong to the same class—has an entropy of 0, indicating no disorder or uncertainty. Conversely, if the dataset is evenly distributed across multiple classes, entropy is at its maximum, indicating the highest level of disorder and uncertainty in predicting the target class.

Mathematical Definition

The entropy $H(S)$ of a dataset S is calculated using the following formula:

$$H(S) = -\sum_{i=1}^{c} p_i \, log_2^{p_i}$$

Where:

- $H(S)$ is the entropy of dataset S
- c is the total number of classes
- p_i is the proportion of instances in class i relative to the total number of instances in the dataset

Interpreting Entropy Values

- **Entropy = 0**: The dataset is completely pure (all instances belong to a single class).
- **Entropy > 0 and < 1**: The dataset contains a mix of classes, but one class may dominate.
- **Entropy = 1 (for binary classification)**: The dataset is equally split between two classes, representing maximum impurity.

Why Entropy Matters

Entropy is used during the decision tree construction process to evaluate how informative an attribute is in classifying the dataset. Attributes that result in lower entropy after splitting the data are considered more effective, as they produce purer subsets. This forms the basis for calculating Information Gain, which guides the algorithm which is made by group in choosing the best attribute at each step of the tree-building process.

INFORMATION GAIN: SELECTING THE MOST INFORMATIVE ATTRIBUTE

Information Gain is a key metric used in decision tree algorithms to determine which attribute best separates the data into classes. It quantifies the reduction in uncertainty or impurity that results from splitting the dataset based on a particular attribute.

In essence, Information Gain measures how much knowing the value of an attribute improves the prediction of the target class. The attribute with the highest Information Gain is chosen at each step of the tree construction because it most effectively organizes the data into purer subsets.

Mathematical Definition

Information Gain $IG(S, A)$ for an attribute A on dataset S is calculated as:

$$IG(S, A) = H(S) - \sum_{v \, \epsilon \, Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

Where:

- $IG(S, A)$ is the Information Gain of attribute A on dataset S
- $H(S)$ is the entropy of the entire dataset before the split
- $Values(A)$ is the set of all possible values for attribute A
- SvSv is the subset of S where attribute A takes the value v
- $|S_v|$ and $|S|$ are the number of samples in S_v and S respectively
- $H(S_v)$ is the entropy of subset S_v

How It Works

- First, calculate the entropy of the full dataset, $H(S)$, which represents the current level of impurity.
- Then, calculate the weighted entropy of each subset S_v created by splitting the dataset on attribute A.
- The weighted sum of these subset entropies represents the expected entropy after splitting.
- Subtracting this from the original entropy gives the Information Gain.
- The attribute with the largest Information Gain is selected for splitting, as it yields the greatest reduction in uncertainty.

FUNCTIONS, MANUEL CONSTRUCTION AND EXPLANATIONS

Functions:

```
def create_colored_tree(root, highlighted_path_objects):  1 usage
    dot = Digraph()
    added_gv_nodes = set()

    def add_nodes_edges(node):
        node_gv_id = str(id(node))

        color = 'red' if node in highlighted_path_objects else 'black'

        if node_gv_id not in added_gv_nodes:
            label_node=node.name
            for items in title:
                if items==node.name:
                    label_node="["+node.name+"]"
            dot.node(node_gv_id, label=label_node, color=color)
            added_gv_nodes.add(node_gv_id)

        for child in node.children:
            child_gv_id = str(id(child))
            dot.edge(node_gv_id, child_gv_id)
            add_nodes_edges(child)

    add_nodes_edges(root)
    return dot
```

This function keeps a list of the elements of the resulting tree structure, and if there is an element of the list in this list, it is colored red, if not, it is colored black and the path specified by the user is displayed. After returning this value, it is printed as png using certain functions from the Digraph library. The specified library is required for the program to run.

```
def entropy(dictionary):  5 usages
    total = 0
    answer=0
    for a in dictionary:
        total+=dictionary[a]
    for b in dictionary:
        p=dictionary[b]/total
        answer+=-p*cmath.log(p, base: 2)
    answer=round(answer.real,3)
    return answer
"""
A parameter called values_dict is used to specify how many output states there are.
part of the gain calculation is the entropy calculation of values_dict,
and when calculating from main_dict you need the sum of the values of the elements in values_dict.
The variable parameter called
main_dict is the dictionary structure created for the first column we have in each calculation.
"""
def information_gain(main_dict,values_dict):  1 usage
    total=0
    answer=0
    for a in values_dict:
        total+=values_dict[a]
    for b in main_dict:
        p=0
        for c in main_dict[b]:
            p+=main_dict[b][c]/total
        print(b,":",main_dict[b],"->Entropy:",entropy(main_dict[b]))
        answer+=-(p*entropy(main_dict[b]))
    answer+=entropy(values_dict)
    answer = round(answer.real, 3)
    return answer
```

The two basic structures that enable the formation of the tree are entropy and information gain.Their calculation is mentioned above and in the assignment document.The mathematical calculation is done as a function in the code.The formation of the decision tree depends only on the states of these two parameters.No external library was used at this stage.

```
def tree_maker(using_list,set_of_attr,values_dict):  2 usages
    temp_result = 0
    top = ""
    final_list=[]
    for k in range(len(title) - 1):
        temp_dict = {}
        if k not in set_of_attr:
            for j in range(len(using_list)):
                if using_list[j][k] not in temp_dict:
                    temp_dict[using_list[j][k]] = {}
                if using_list[j][len(title) - 1] not in temp_dict[using_list[j][k]]:
                    temp_dict[using_list[j][k]][using_list[j][len(title) - 1]] = 0
                temp_dict[using_list[j][k]][using_list[j][len(title) - 1]] += 1
            print("For:",title[k])
            result = information_gain(temp_dict, values_dict)
            print("Gain:",result,"\n")
            if result > temp_result:
                temp_result = result
                final_list = list(temp_dict.items())
                top = k
    return [top,final_list]
```

The purpose of this tree maker function is actually to calculate the gains and find out which column has the highest gain.While doing this, the structure of the list sent to the function, which attributes have been used before and the general output structure of that list are important.The loop calculates the entropy and gain calculations according to the list for a column at each stage, and if it is greater than the highest value before it, it enters the if condition and it becomes the new highest gain. The footnote here is that the variable called top is initially specified as a string and then, relying on Python's dynamic variable type, it is equated to the value that carries the index value of that column.The top variable has an important function in the loop that creates the structure of the whole tree because any gain value that is not greater than 0 indicates that there is a conflict situation in the current stage of the tree.

```
def child_list_maker(tree_maker_list,using_list,ascendant_set):  2 usages
    temp_set=ascendant_set.copy()
    for j in range(len(tree_maker_list[1])):
        sub_list=[]
        for k in range(len(using_list)):
            if tree_maker_list[1][j][0]==using_list[k][tree_maker_list[0]]:
                sub_list.append(using_list[k])
        temp_set.add(tree_maker_list[0])
        element=TreeNodes(sub_list,temp_set)
        Tree_Nodes_List.append(element)
```

After determining the column and structures with the highest gain, the sub_list of the list structure should be taken for the different variables in this column and the column used should not be used again.

```
def traverse_tree(list_of_inputs):  1 usage
    temp=root
    highlight_nodes = [temp]
    main_flag=True
    temp_child=root
    while main_flag:
        for k in range(len(list_of_inputs)):
            tree_flag=False
            if list_of_inputs[k][0]==temp.name:

                for l in range(len(temp.children)):
                    if temp.children[l].name==list_of_inputs[k][1]:
                        temp=temp.children[l]
                        highlight_nodes.append(temp)
                        tree_flag=True
                        break
                if not tree_flag:
                    main_flag=False
                    print("One of the data you have entered is incorrect.")
                    break
                if temp.children:
                    if len(temp.children) > 1 and not temp.children[0].children:
                        temp_child=temp.children[0]
                        backup=temp
                        for c in range(len(temp.children)):
                            temp=temp.children[c]
                            highlight_nodes.append(temp)
                            temp=backup
                        temp=temp_child
```

```
                if not tree_flag:
                    main_flag=False
                    print("One of the data you have entered is incorrect.")
                    break
                if temp.children:
                    if len(temp.children) > 1 and not temp.children[0].children:
                        temp_child=temp.children[0]
                        backup=temp
                        for c in range(len(temp.children)):
                            temp=temp.children[c]
                            highlight_nodes.append(temp)
                            temp=backup
                        temp=temp_child
                    else:
                        temp = temp.children[0]
                        highlight_nodes.append(temp)
            if not temp.children: #IMPORTANT FACTOR:If tree includes some conflict factors,the tree can not decide which way it should
                answer=temp.name
                main_flag=False
                print("--------FINAL ANSWER--------")
                if temp_child is not root:print("The final result can not be determined because of inputs")
                else:print(title[len(title) - 1] + ":" + answer)
                dot = create_colored_tree(root, highlight_nodes)
                dot.render( filename: 'decision_tree', format='png', cleanup=True,view=True)
```

The tree traverse function is important for generating the png according to the user's specific inputs after the decision tree is printed to the console and for generating the final output according to the given inputs. The reason for the complexity in terms of variable compared to other functions is the conflict situations called anomalies in the breast_cancer.csv file. In order to display these situations accurately by the user, the functionality of the structure has been increased but the readability has decreased.

```
while len(dictionary_values_list)>0:

  entropy_answer=entropy(dictionary_values_list[0][1])
  print("----- Subtree for:",dictionary_values_list[0][0],":",dictionary_values_list[0][1],"Parent:",node_list[counter].parent.name,"En
  if entropy_answer==0:

      key = next(iter(dictionary_values_list[0][1]))
      print("All",key,"-> Leaf Node\n")
      node = Node(key,parent=node_list[counter])
  else:
      answer_list=tree_maker(Tree_Nodes_List[0].sub_list,Tree_Nodes_List[0].ascendant,dictionary_values_list[0][1])
      if isinstance(answer_list[0],str):
          print("In this case,The Decision Tree can not decide which parameter will be divided")
          for key in dictionary_values_list[0][1].keys():
              node = Node(key, parent=node_list[counter])
      else:
          print("Highest Gain is:", title[answer_list[0]], "Split based on:", title[answer_list[0]], "\n")
          node = Node(title[answer_list[0]], parent=node_list[counter])
          dictionary_values_list.extend(answer_list[1])
          child_list_maker(answer_list, Tree_Nodes_List[0].sub_list, Tree_Nodes_List[0].ascendant)
          for item in answer_list[1]:
              node2 = Node(item[0], parent=node)
              node_list.append(node2)
  Tree_Nodes_List.pop(0)
  dictionary_values_list.pop(0)
  counter=counter+1
```

Finally, Looking for the general loop structure, this is not normally a function definition, but it is included because it is a general loop and it is the main factor in the structure of the whole tree except for the root and the first children.The main purpose is to enable the formation of the decision tree.It is prepared by making use of Python's various data structures.The main design is made with 3 different lists and their synchronization is provided.2 lists are used as a queue structure and memory is saved.

STEP-BY-STEP CONSTRUCTION WITH EXAMPLE OUTPUT

Step-by-Step Construction

The structure of the program works according to the order of the items mentioned below. According to the user's request, article 6 can repeat itself as many times as desired. After the 5th article,root and the first children are formed, the whole tree rotates in a certain loop until it is completed.

1.Data Entry and Flexibility:

Although the system was mainly tested on three different CSV files (weather.csv, contact_lenses.csv, breast_cancer.csv), as mentioned in the introduction, it was designed to handle CSV files as well as files in other formats such as TXT. Each file has its own attributes and output states, and the program is able to respond consistently to these different file structures and contents.

2. Delimiter Detection and Data Reading:

After the user specifies the file they want to process, a function called Sniffer in Python's csv library automatically detects which delimiter character (e.g. comma, semicolon, etc.) is used in the specified file. After this detection, the standard file reading process commonly used in programming languages is performed. Each line read is stored in a list, which in Python is a direct data structure (Python does not have a direct 'array' structure; lists fulfill this need).

3.Initial Entropy Calculation and Root Node Identification:

An initial entropy calculation is performed for the last column of the generated master list (the entire dataset), i.e. the target attribute (output class). This entropy value is of critical importance as it will be used in the information gain calculations to determine the root node of the decision tree. This is explained more clearly in the "Information Gain Calculation" section of the introduction. Once the entropy in the last column of the master list is calculated, the possible information gains for all other attribute columns in the dataset are calculated separately and the attribute with the highest information gain is determined as the root node of the tree.

4.Creation of First Child Nodes and Use of Data Structures:

The identification of the root node naturally leads to the identification of the first child nodes (branches/children) connected to this root. This stage constitutes the first structural layer of the decision tree. Starting from the child nodes, various data structures are used to ensure synchronization and data consistency in the branching process of the tree structure. For example, dictionary structures are converted into lists or lists are kept together with special objects.

5. Main Cycle: Tree Expansion and Termination Conditions:

After the creation of the root node and the first child nodes, the main loop begins. Again, entropy and possible information gain results play a decisive role in the progress of this loop. The main loop first checks whether the entropy of the subset being processed is 0 (zero). An entropy of 0, as mentioned in the introduction, indicates that the subset is in a “pure” state, i.e. all instances belong to the same class. In this case, that point is directly identified as a leaf node and the branching process for that branch is terminated. The entropy calculation in the loop is not based on the entire main dataset, but on the subset that is currently being processed and derived from the previous node.

An important point, also mentioned in the “Functions” section, is that there are certain conflict situations that can be encountered during the information gain calculation. In such cases, instead of branching the node structure to the end, the program assigns the possible output states as leaf nodes at the first point where the conflict starts. This structure ensures that the data structures that hold the nodes to be processed, which are determined during the creation of the root node, work in a queue fashion until there are no more elements to process (the number of elements drops to zero).
Finally, it should be noted that since the main structure is a loop, information gain will enter a state of continuous and continuous computation if the entropy of the list created by that node is not 0, and thus the tree structure will be formed correctly.

6. Tree Visualization and User Interaction:

The decision tree resulting from the execution of the algorithm can be easily printed on the console screen using Python's bigtree library. (Note: The purpose of using the bigtree library is certainly not to influence the basic logic of the algorithm, but only to model the parent-child relationships and to enable the hierarchical structure of the tree to be easily displayed on

the console). After the tree structure is printed on the screen, the user is asked to enter certain inputs (attribute values). According to these values entered by the user, the result (predicted class) is determined by following the path on the decision tree, and this path can optionally be created as a visual file in PNG format. The system allows the user to query again with different inputs.

Example from output

This report part outlines how a program builds decision trees using data from **CSV or TXT files**. While it can handle various datasets, the examples here mostly use weather.csv because its output is easier to manage and read than that of the two other available files. The program is **reliable and works well with different types of datasets**, consistently showing results similar to the examples provided. This document part explains the program's steps with example output: how it takes in files, picks the main starting point (the root node), processes subsequent branches (child nodes), deals with clear-cut outcomes (pure nodes), shows the tree using the bigtree library, and lets users get predictions.

File Input and Program Kick-off

- **What happens:** The program starts up when the user types in the name of the data file.
- **Handling the File:** If the filename isn't right (maybe it's misspelled or the file isn't there), the program will show an error and finish the process.
- **Be Careful:** Make sure to include the file extension (like .csv or .txt) because the program looks at the whole name.

```
Finding for Root: 0.94   {'no': 5, 'yes': 9}

For: outlook
sunny : {'no': 3, 'yes': 2} ->Entropy: 0.971
overcast : {'yes': 4} ->Entropy: 0.0
rainy : {'yes': 3, 'no': 2} ->Entropy: 0.971
Gain: 0.246

For: temperature
hot : {'no': 2, 'yes': 2} ->Entropy: 1.0
mild : {'yes': 4, 'no': 2} ->Entropy: 0.918
cool : {'yes': 3, 'no': 1} ->Entropy: 0.811
Gain: 0.029

For: humidity
high : {'no': 4, 'yes': 3} ->Entropy: 0.985
normal : {'yes': 6, 'no': 1} ->Entropy: 0.592
Gain: 0.151

For: windy
false : {'no': 2, 'yes': 6} ->Entropy: 0.811
true : {'no': 3, 'yes': 3} ->Entropy: 1.0
Gain: 0.048

Highest Gain is: outlook Split based on: outlook
```

Figuring Out the Starting Point (Root Node)

- **What this shows:** This output is all about how the first main decision point (the root node) is chosen.
- **Highlights:** For every column (or attribute) in the data:
    - It lists out all the unique things (elements) in that column.
    - It calculates "entropy" for each of these unique things.

    - Finally, it figures out an "Information Gain" for the whole column.

- **Making the Choice:** The column that gets the highest Information Gain score is picked as the "highest gain" column, and that becomes the root node of the decision tree.
- **Iteration:** After the root node is set, this whole calculation process is **repeated in a loop** to figure out the child nodes for each branch coming off the root.

```
----- Subtree for: sunny : {'no': 3, 'yes': 2} Parent: outlook Entropy: 0.971 -----

For: temperature
hot : {'no': 2} ->Entropy: 0.0
mild : {'no': 1, 'yes': 1} ->Entropy: 1.0
cool : {'yes': 1} ->Entropy: 0.0
Gain: 0.571

For: humidity
high : {'no': 3} ->Entropy: 0.0
normal : {'yes': 2} ->Entropy: 0.0
Gain: 0.971

For: windy
false : {'no': 2, 'yes': 1} ->Entropy: 0.918
true : {'no': 1, 'yes': 1} ->Entropy: 1.0
Gain: 0.02

Highest Gain is: humidity Split based on: humidity
```

Dealing with Child Nodes and How It Looks

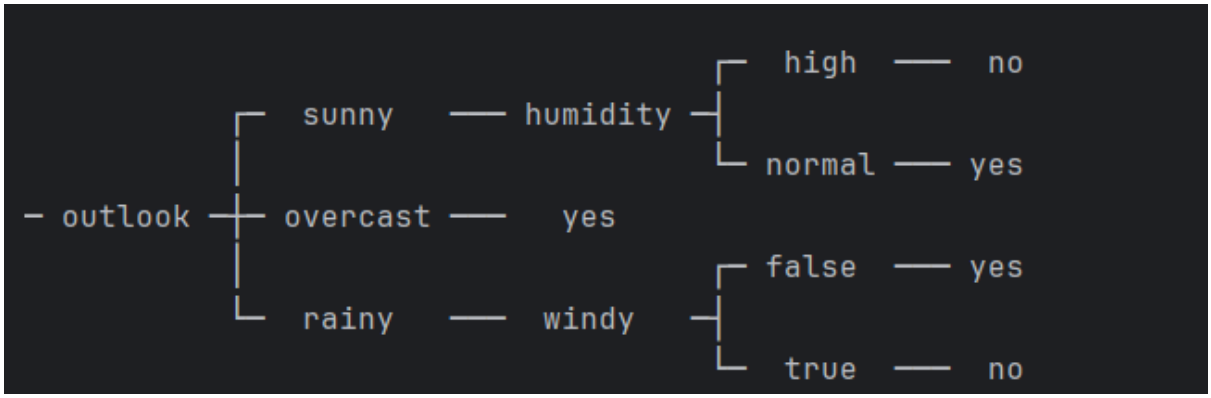
- **What happens next:** After the root node is decided, the program looks at each child element (which matches up with the unique values of the root attribute). It goes through them in an order set by a dictionary_values_list (this is part of how the function works).
- **Keeping it Clear:**
    - To make things easier to read, especially when there are lots of calculations, the program prints the parent node's name when it's showing the math for its child nodes.

- The numbers you see (like entropy and gain) will be different for each child node's part of the tree because they're working with different smaller chunks of the data (sublists).

```
----- Subtree for: high : {'no': 3} Parent: humidity Entropy: 0.0 -----

All no -> Leaf Node
```

When Things Are Simple (Pure Subsets) and Reaching the End (Leaf Nodes)

- **What's Next:** Sometimes, as the tree gets built, a small dataset(a sub_list) will only have one kind of outcome (like all "yes" or all "no").
- **Zero Entropy:** When that happens, the entropy for that dataset is 0, and the program shows this.
- **Leaf Nodes:** If a node is a final decision point (a leaf node), the program doesn't bother doing or showing any more entropy calculations for it in this output.(The calculation is finished only for that part of the tree and the calculation is continued for the rest of the tree parts.)



Showing the Tree

- **How it's made:** The decision tree's structure is put together by using the Node class from a Python tool called bigtree.
- **About Libraries:**
  - The main part of the program that builds the tree (the part that does the entropy/gain math) **doesn't use any outside libraries**.

- The bigtree library is **only used to show or print** the tree once it's built.

- **How it looks:** The tree is printed out sideways (horizontally) in the console using bigtree's root.hshow() command.

```
Please enter outlook ['sunny', 'overcast', 'rainy'] type: sunny
Please enter temperature ['hot', 'mild', 'cool'] type: mild
Please enter humidity ['high', 'normal'] type: high
Please enter windy ['false', 'true'] type: something


--------FINAL ANSWER--------
play:no
```

Users Enter Some Inputs

- **What users do:** The user can type in a set of values for the different attributes in the dataset.

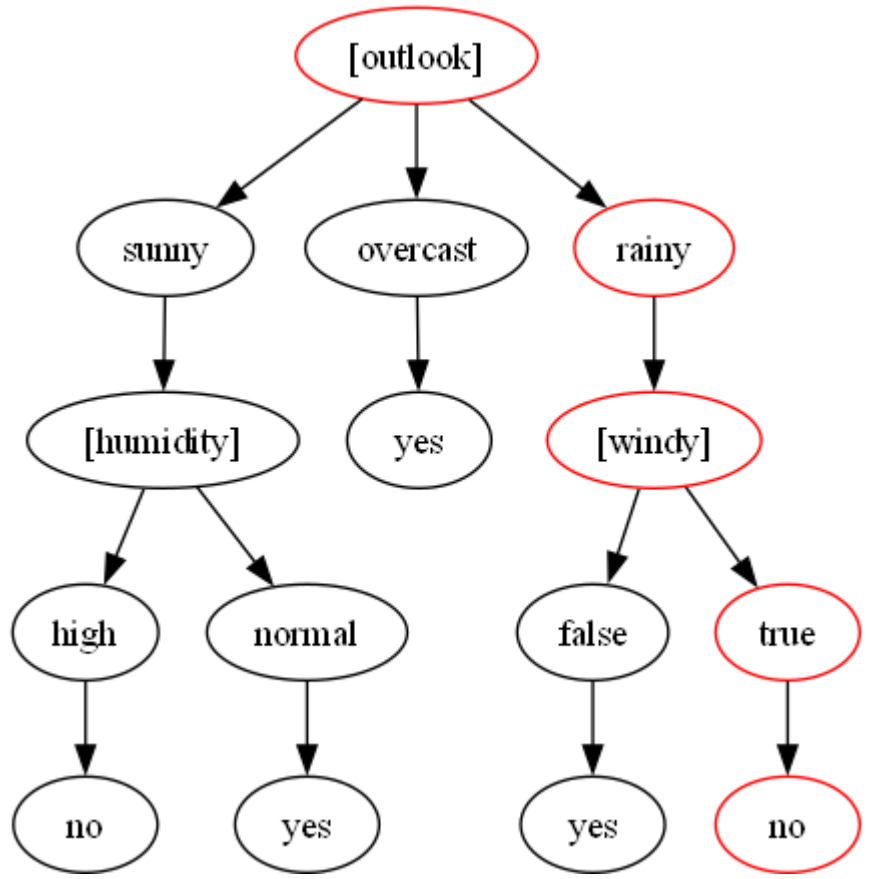
- **Input Flexibility:** The program accepts these inputs even if:

  - Certain values do not change how the tree itself is constructed (if these values do not affect the structure of the tree, it does not matter what it writes.)

  - Input values are not case sensitive.

- **Getting Answers:** Based on what the user types in, the program follows the branches of the tree and prints out what it thinks the outcome or classification should be.

- **More Questions?:** The user can then choose to type in new inputs to get more predictions if they want.



Final PNG Output

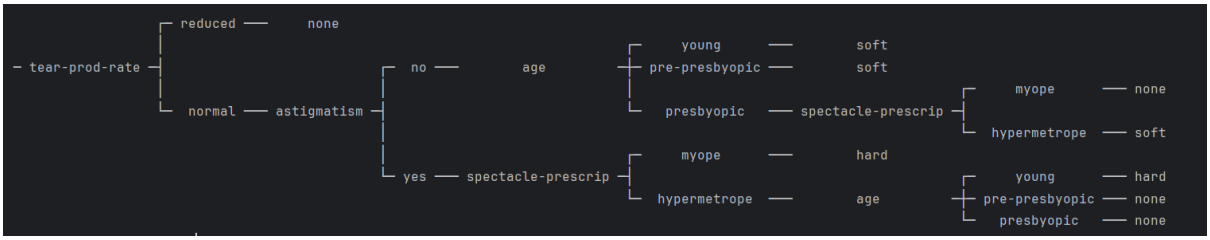
- **What's this about:** After doing its thing (like building a tree or making a prediction), the program makes a Tree PNG. According to the input status specified by the user, the pattern is indicated in color.

OTHER OUTPUT EXAMPLES WITH SCREENSHOTS

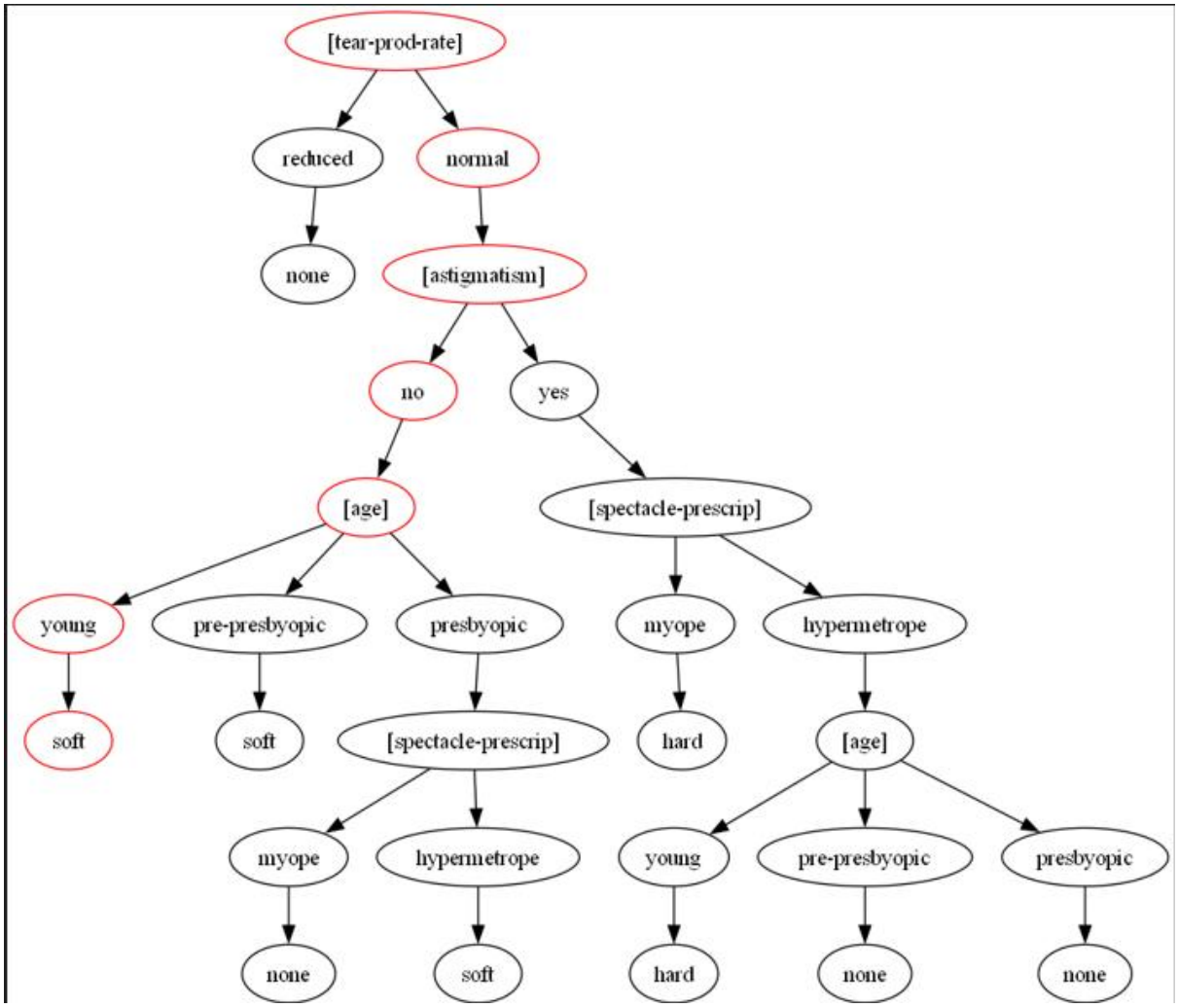
Contact_lenses.csv

```
Finding for Root: 1.326   {'none': 15, 'soft': 5, 'hard': 4}

For: age
young : {'none': 4, 'soft': 2, 'hard': 2} ->Entropy: 1.5
pre-presbyopic : {'none': 5, 'soft': 2, 'hard': 1} ->Entropy: 1.299
presbyopic : {'none': 6, 'hard': 1, 'soft': 1} ->Entropy: 1.061
Gain: 0.039

For: spectacle-prescrip
myope : {'none': 7, 'soft': 2, 'hard': 3} ->Entropy: 1.384
hypermetrope : {'none': 8, 'soft': 3, 'hard': 1} ->Entropy: 1.189
Gain: 0.04

For: astigmatism
no : {'none': 7, 'soft': 5} ->Entropy: 0.98
yes : {'none': 8, 'hard': 4} ->Entropy: 0.918
Gain: 0.377

For: tear-prod-rate
reduced : {'none': 12} ->Entropy: 0.0
normal : {'soft': 5, 'hard': 4, 'none': 3} ->Entropy: 1.555
Gain: 0.549

Highest Gain is: tear-prod-rate Split based on: tear-prod-rate
```

```
----- Subtree for: reduced : {'none': 12} Parent: tear-prod-rate Entropy: 0.0 -----

All none -> Leaf Node

----- Subtree for: normal : {'soft': 5, 'hard': 4, 'none': 3} Parent: tear-prod-rate Entropy: 1.555 -----

For: age
young : {'soft': 2, 'hard': 2} ->Entropy: 1.0
pre-presbyopic : {'soft': 2, 'hard': 1, 'none': 1} ->Entropy: 1.5
presbyopic : {'none': 2, 'hard': 1, 'soft': 1} ->Entropy: 1.5
Gain: 0.222

For: spectacle-prescrip
myope : {'soft': 2, 'hard': 3, 'none': 1} ->Entropy: 1.459
hypermetrope : {'soft': 3, 'hard': 1, 'none': 2} ->Entropy: 1.459
Gain: 0.096

For: astigmatism
no : {'soft': 5, 'none': 1} ->Entropy: 0.65
yes : {'hard': 4, 'none': 2} ->Entropy: 0.918
Gain: 0.771

Highest Gain is: astigmatism Split based on: astigmatism
```




```
Please enter age ['young', 'pre-presbyopic', 'presbyopic'] type: young
Please enter spectacle-prescrip ['myope', 'hypermetrope'] type: fake_data
Please enter astigmatism ['no', 'yes'] type: no
Please enter tear-prod-rate ['reduced', 'normal'] type: normal


--------FINAL ANSWER--------
contact-lenses:soft
```

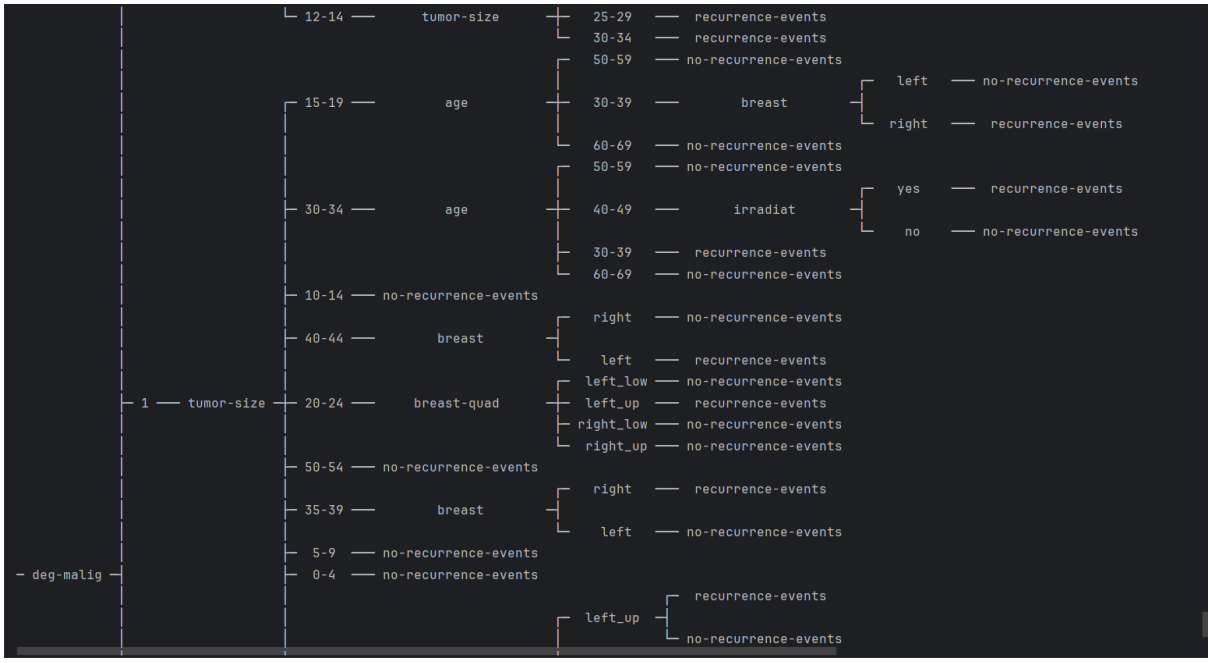


Breast_Cancer.csv

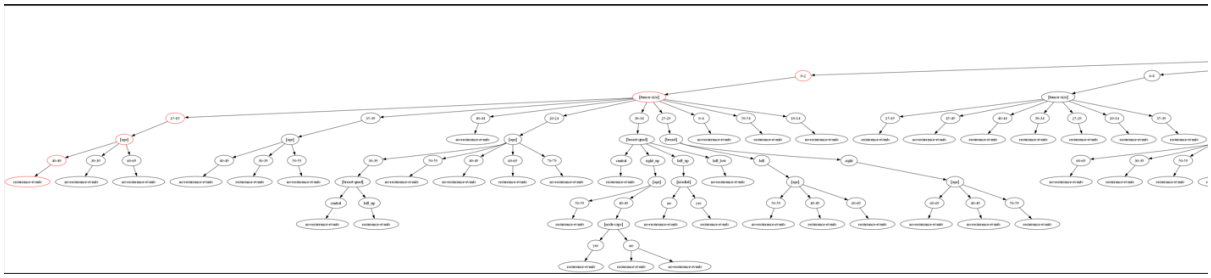
```
Finding for Root: 0.872   {'recurrence-events': 81, 'no-recurrence-events': 196}

For: age
40-49 : {'recurrence-events': 27, 'no-recurrence-events': 62} ->Entropy: 0.885
50-59 : {'no-recurrence-events': 69, 'recurrence-events': 22} ->Entropy: 0.798
60-69 : {'no-recurrence-events': 38, 'recurrence-events': 17} ->Entropy: 0.892
30-39 : {'no-recurrence-events': 21, 'recurrence-events': 15} ->Entropy: 0.98
70-79 : {'no-recurrence-events': 5} ->Entropy: 0.0
20-29 : {'no-recurrence-events': 1} ->Entropy: 0.0
Gain: 0.021

For: menopause
premeno : {'recurrence-events': 48, 'no-recurrence-events': 101} ->Entropy: 0.907
ge40 : {'no-recurrence-events': 90, 'recurrence-events': 33} ->Entropy: 0.839
lt40 : {'no-recurrence-events': 5} ->Entropy: 0.0
Gain: 0.012

For: tumor-size
15-19 : {'recurrence-events': 6, 'no-recurrence-events': 23} ->Entropy: 0.736
35-39 : {'recurrence-events': 7, 'no-recurrence-events': 12} ->Entropy: 0.949
30-34 : {'recurrence-events': 24, 'no-recurrence-events': 33} ->Entropy: 0.982
25-29 : {'no-recurrence-events': 33, 'recurrence-events': 18} ->Entropy: 0.937
40-44 : {'no-recurrence-events': 16, 'recurrence-events': 6} ->Entropy: 0.845
10-14 : {'no-recurrence-events': 27, 'recurrence-events': 1} ->Entropy: 0.222
0-4 : {'no-recurrence-events': 7, 'recurrence-events': 1} ->Entropy: 0.544
20-24 : {'no-recurrence-events': 34, 'recurrence-events': 14} ->Entropy: 0.871
45-49 : {'no-recurrence-events': 2, 'recurrence-events': 1} ->Entropy: 0.918
```

```
----- Subtree for: 45-49 : {'no-recurrence-events': 1} Parent: tumor-size Entropy: 0.0 -----

All no-recurrence-events -> Leaf Node

----- Subtree for: 5-9 : {'no-recurrence-events': 2} Parent: tumor-size Entropy: 0.0 -----

All no-recurrence-events -> Leaf Node

----- Subtree for: 15-19 : {'recurrence-events': 1, 'no-recurrence-events': 2} Parent: tumor-size Entropy: 0.918 -----
```

```
----- Subtree for: 30-39 : {'no-recurrence-events': 2, 'recurrence-events': 1} Parent: age Entropy: 0.918 -----

For: menopause
premeno : {'no-recurrence-events': 2, 'recurrence-events': 1} ->Entropy: 0.918
Gain: 0.0

For: inv-nodes
0-2 : {'no-recurrence-events': 2, 'recurrence-events': 1} ->Entropy: 0.918
Gain: 0.0

For: node-caps
no : {'no-recurrence-events': 2, 'recurrence-events': 1} ->Entropy: 0.918
Gain: 0.0

For: breast
left : {'no-recurrence-events': 2} ->Entropy: 0.0
right : {'recurrence-events': 1} ->Entropy: 0.0
Gain: 0.918

For: breast-quad
left_low : {'no-recurrence-events': 2, 'recurrence-events': 1} ->Entropy: 0.918
Gain: 0.0

For: irradiat
no : {'no-recurrence-events': 2, 'recurrence-events': 1} ->Entropy: 0.918
Gain: 0.0

Highest Gain is: breast Split based on: breast
```

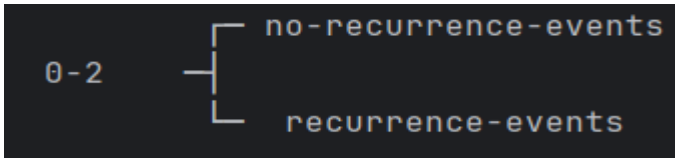


```
Please enter age ['40-49', '50-59', '60-69', '30-39', '70-79', '20-29'] type: 40-49
Please enter menopause ['premeno', 'ge40', 'lt40'] type: premeno
Please enter tumor-size ['15-19', '35-39', '30-34', '25-29', '40-44', '10-14', '0-4', '20-24', '45-49', '50-54', '5-9'] type: 15-19
Please enter inv-nodes ['0-2', '3-5', '15-17', '6-8', '9-11', '24-26', '12-14'] type: 0-2
Please enter node-caps ['yes', 'no'] type: yes
Please enter deg-malig ['3', '1', '2'] type: 3
Please enter breast ['right', 'left'] type: right
Please enter breast-quad ['left_up', 'central', 'left_low', 'right_up', 'right_low'] type: left_up
Please enter irradiat ['no', 'yes'] type: no


--------FINAL ANSWER--------
Class:recurrence-events
```

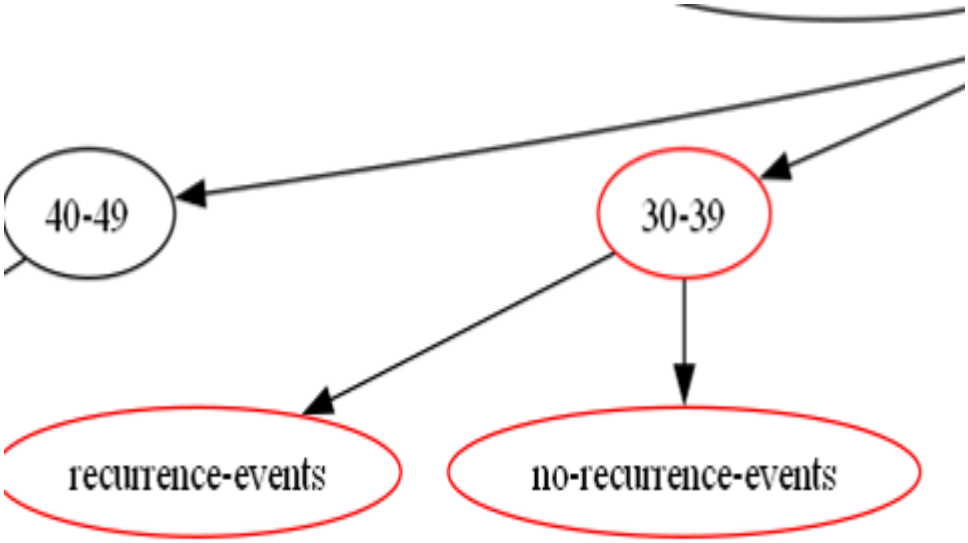


Important information about Breast_Cancer file

There are a total of 6 anomalies in the dataset. To summarize this situation, the inputs are identical but the output result is different. In this case, the program returns a special console message and while the tree structure is shown as PNG, all output situations that can occur in that situation are specified as leaf node.



```
Please enter age ['40-49', '50-59', '60-69', '30-39', '70-79', '20-29'] type: 30-39
Please enter menopause ['premeno', 'ge40', 'lt40'] type: premeno
Please enter tumor-size ['15-19', '35-39', '30-34', '25-29', '40-44', '10-14', '0-4', '20-24', '45-49', '50-54', '5-9'] type: 0-4
Please enter inv-nodes ['0-2', '3-5', '15-17', '6-8', '9-11', '24-26', '12-14'] type: 0-2
Please enter node-caps ['yes', 'no'] type: no
Please enter deg-malig ['3', '1', '2'] type: 2
Please enter breast ['right', 'left'] type: right
Please enter breast-quad ['left_up', 'central', 'left_low', 'right_up', 'right_low'] type: central
Please enter irradiat ['no', 'yes'] type: no


--------FINAL ANSWER--------
The final result can not be determined because of inputs
```



CONCLUSION

More than just a coding exercise, this project offered an look at fundamental machine learning ideas and software development practices. By building an algorithm with using Python, a solid understanding of concepts such as entropy, information gain, and the incremental nature of decision tree construction was gained. This hands-on method helped to appreciate the math behind the algorithm.

Doing each step manually, such as calculating entropy, selecting the best data splits and building the tree, significantly improved understanding and coding skills. The choice of Python for this project was highly beneficial; its flexibility made it easy to build the algorithm step by step. This practical Python experience provided valuable skills for future software and data work.

The result is a user-friendly command-line tool that can read data from CSV or TXT files, accurately calculate entropy and information gain, build a decision tree that can be visualized, and provide instant classification results for new data. This work demonstrates how algorithmic understanding,programming language like Python, and practical combine. It provides a solid foundation for future projects in the field of machine learning and data science, and the necessary knowledge and tools for more complex tasks.