Shagun Gupta
Lisa Meng
Reid Pattis
INF 552
HW #1 - Decision Trees

Report

1. The following data structures were used to implement the decision tree algorithm: Class Node to define all nodes of the decision tree except the leaf nodes, Class leaf_node to define all the leaf nodes, and Class Tree to define, build, visualize and test the decision tree itself.

- Class Node:
    - Consists of the data variables: attribute name (att_name), information gain (ig), and the **incoming** branch value (branch_val) (-1 for root node).
    - Has a function is_leaf() to ask if objects of this class are leaf nodes or not; always returns No.
- Class Leaf_node:
    - Consists of the label value (label_val) and the **incoming** branch value (branch_val)..
    - Has a function is_leaf() to ask if objects of this class are leaf nodes or not; always returns Yes.

- Class Tree:

    - Using a defaultdict to store the main tree where the dictionary is of the format- parent object: [a list of children objects that could belong to both Class node and Class leaf_node].

    - Using a dictionary (features) to store all the unique values per column as well.

    - Consists of functions: calculate_root that calculates the new root node for every dataset (recursive function), visualization function, test function.

Pandas was used to handle data and regular expressions were used to clean the data. Classes were used to organize nodes and collections.defaultdict was used instead of a normal dictionary to optimize the code by avoiding errors during retrieval and initialization.

We followed the ID3 algorithm to implement the decision tree and organized the code to first handle edge cases and if none apply, we then calculated entropy to decide the next node to split the tree on. We also included the case where, for example, if splitting the node based on Location, and City-Centre is not part of any of the examples in that subset, then the algorithm creates that branch and points to a leaf-node with majority label value in the parent subset.

The code was developed in multiple phases. Initially, we did not have clarity on all the edge cases and they were added subsequently as we realised these edge cases also exist. We also initially could not decide how to store and visualize this tree. We solved this dilemma after a lot of brainstorming and experimenting. We first worked on calculating the entropy successfully and then linked that to choosing the root node and calling the recursion function. Then, we added the edge cases in the end. We also ran the ID3 library and compared the tree produced to get some clarity on what kind of result we should expect. We decided to visualize the tree through simple indentation. The format of the decision tree being printed is:

```
 -1 -> Root Node
        - branch value -> next attribute
                - branch value -> next attribute
                        -branch value -> leaf node
                        -branch value -> leaf node
                -branch value -> leaf node
        -branch value -> leaf node
```

The result we got on running the test case: (occupied = Moderate; price = Cheap; music = Loud; location = City-Center; VIP = No; favorite beer = No) - **YES**

Final Tree:

```
--1->Occupied
        -High->Location
                -Ein-Karem->Yes
                -Talpiot->No
                -City-Center->Yes
                -Mahane-Yehuda->Yes
                -German-Colony->No
        -Moderate->Location
                -City-Center->Yes
```

```
    -German-Colony->VIP
        -No->No
        -Yes->Yes
    -Ein-Karem->Yes
    -Mahane-Yehuda->Yes
    -Talpiot->Price
        -Expensive->No
        -Cheap->No
        -Normal->Yes
-Low->Location
    -German-Colony->No
    -Ein-Karem->Price
        -Expensive->No
        -Normal->No
        -Cheap->Yes
    -City-Center->Price
        -Expensive->No
        -Cheap->No
        -Normal->Music
            -Loud->No
            -Quiet->VIP
                -Yes->No
                -No->Favorite Beer
                    -Yes->No
                    -No->No
    -Talpiot->No
    -Mahane-Yehuda->No
```

2. Our implementation of a decision tree uses the ID3 algorithm, which was taught in class. When researching which library to use, we discovered that the most popular library to use when building decision trees is “sklearn”. But we quickly learned that sklearn’s algorithm uses CART. Because ID3 uses Entropy and Information gain to construct a decision tree and CART uses Gini Impurity, we decided to forego sklearn. Upon further research, we finally found a library, called “decision-tree-id3”, that uses ID3 in its algorithm but required us to pip install decision-tree-id3.

Due to the fact that the training data had discrepancies, i.e. 2 rows of data had the same attribute values but different labels, the id3 library handled this by checking if the attribute values are same and if the labels are same, but in the case that the labels are not same, the library goes to the parent node and takes the majority label to create a leaf node and provides the distribution of the label values (eg. 1 YES/1 NO). Differently, our code accounts for this discrepancy by making subsets of our initial training dataset (dropping an attribute everytime) until we are left with only the label value. Thus when there are no more attributes to be selected, and the instances still do not have the same label, a leaf node is created and labelled with the most common class of the examples in the subset.

Ways to improve our code:

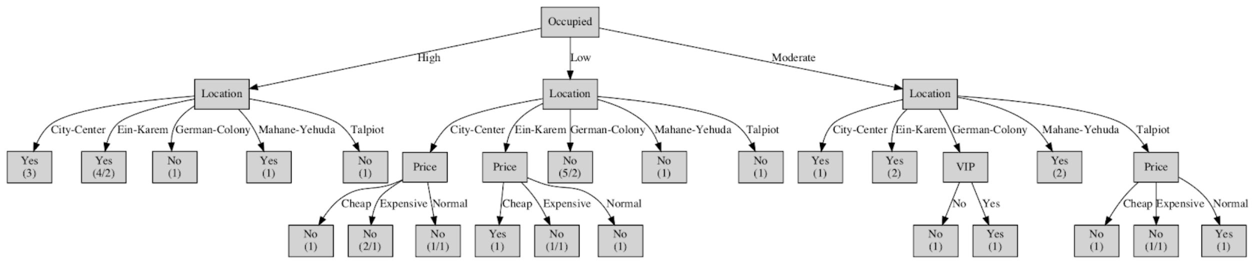
1. Instead of using maximum entropy gain we could use minimum average entropy after splitting on the basis of a node. Since I.G. = Entropy at root node - Average Entropy and Entropy at root node is constant for a certain dataset

2. To handle the edge case where all attribute values are constant but the label values are different (discrepancy in data), we created the

tree till only the label column was left in the dataset and then chose the majority value, but that part of the tree could be replaced with the majority value as well - decreasing the tree size.

But similar to the library, when every attribute value in the subset belongs to the same class, the node is turned into a leaf node and labelled with the class of the instances.

In addition, when implementing the id3 library and visualizing the tree, we noticed that a leaf was created and labelled with the most common class/label of the instance in the parent node's set when no instance in the parent set was found to match a specific value of the selected attribute. This leaf also gave a distribution of the label values. Our code does the same with the exception of the distribution.



3. As stated in its name, decision trees are useful as a decision-making tool. At a basic level, it can be used ubiquitously to make and support decisions, and visualize all possible outcomes of a decision and the consequences along each branch. For example, a day-to-day application of a decision tree is using it to determine if you should go out that night based on historical data regarding attributes such as time, homework, location, food, activity, and which friend(s) is coming along. At a deeper level, decision trees is a Machine Learning algorithm used for Classification and Regression problems. For example, in terms of classification, with data about Iris flower sepal and petal length and

width (attributes) and its different type of species (class/label), a decision tree can be applied to determine the species of a new Iris flower. Another example, with physical characteristics (attributes) and edibility(class/label), a decision tree can be applied to determine if an unprecedented mushroom is edible or not. And because decision trees use supervised learning, it is very useful as explainable AI. Thus models that use decision tree algorithms are easier to visualize and explain.

INDIVIDUAL CONTRIBUTIONS

For this assignment we started out by coding individually and then picking the best code but came down to taking Shagun's rough code that worked until inclusion of edge cases and debugged and fixed it by changing some data structures and adding the edge cases.

Before and while coding was done, we discussed the algorithm multiple times, manually implemented it to understand it's working and differentiated between various pseudo-codes that we could adopt.

The part 2 of the assignment was researched by Lisa and Reid where they decided to implement id3. It was implemented by Reid and checked and confirmed by Lisa and Shagun.

The third part of the assignment was done by Lisa.

The report was compiled and written collectively.