INF 552 HW1

Teammates:
He Chang 5670527576
Ziqiao Gao 2157371827
Fanlin Qin 5317973858

Part 1.

Data Structure:
Based on the training data, our Decision Tree is a general tree, each tree node has following properties:

self.question
self.last_answer
self.children
self.isLeaf

question: *question* means the name of the tree node. When the node is not a leaf, the *question* is the question that we are going to ask when visiting this tree node.

last_answer: *last_answer* notes the choice of this tree node's parent question.

children: *children* note the children nodes of this node. The *children* property is a list and the structure of its element is a simple dictionary looking like: {choice: next_node}.

isLeaf: *isLeaf* notes if this node is a leaf. If it is, the *question* property would be the answer.

e.g A tree node could be:
{
self.question: "Location"
self.last_answer: "High"
self.children: [{City-Center: Next_node}, {Mahane-Yehuda: Next_node2}]
isLeaf: false
}

Optimization:

1. Tree node design: At the beginning, we applied normal general tree node data structure (children is a list of child nodes) to our decision tree, but there was a problem that we might need an extra property, a map to record the choice of current question and its relevant next node, otherwise we won't know which answer leads to next node. Therefore, instead of simply putting nodes in the children list, we changed the child from

next_node to a dictionary {choice: next_node}. E.g [next_node1, next_node2] -> [{ “yes”: next_node1}, {“no”: next_node2}]

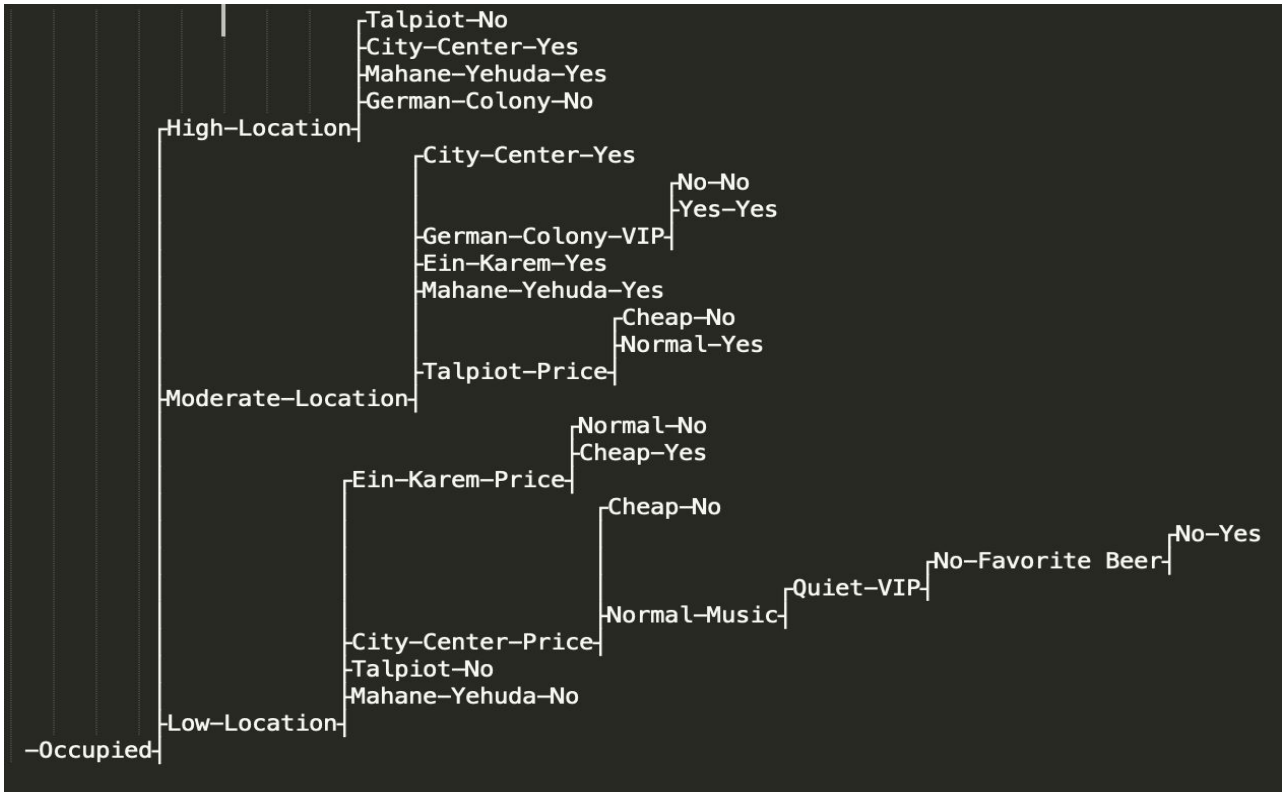
2. Dataset: In each recursion, we used to give full dataset as an input to each recursion, this method requires extra property to denote previous choices and attributes which is too complicated and confusing. Therefore, we decided to produce a new dataset which will filter irrelevant data and attributes for each new recursion. In the new method, we won’t need to clean the dataset before using it.
3. Printing the tree: Instead of manually drawing the decision tree, we printed the tree using recursion.

Challenge:

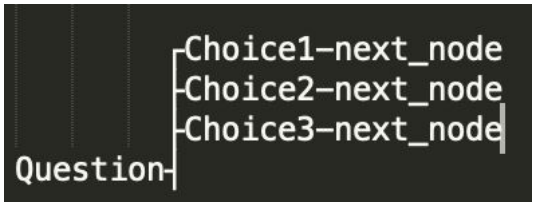
1. We spend a long time designing a good tree-node design.
2. We have to keep generating sub-dataset for each node because we do not want to consider previously used attributes.
3. There is a data conflict in this training data, that is, two identical data results in two opposite results. We have to randomly pick one of them as the result of the leaf node.
4. Printing such a decision tree is a huge challenge. We used to think of using first order traversal to print the tree and then manually draw the structure of the tree, however, we conquered this difficulty.

Our Prediction: **YES**

Decision Tree Graph:

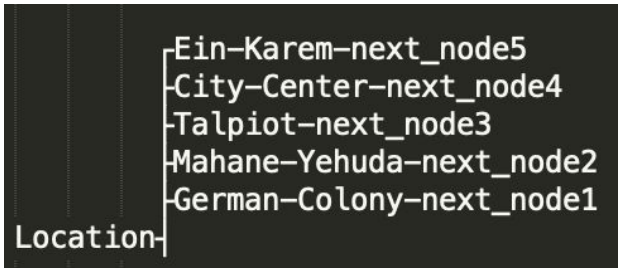


Explanation: A typical tree node can be seen as following structure:



On the left is the question (name) of the node, and then we can see that three child nodes are listed from top to bottom on the right. As we explained before, each child node is a dictionary {choice: next_node}. One thing we need to pay attention to is that all child nodes are above the line of the question, and any word below the question line is not part of the subtree.

e.g:



In this example, we can see the question of this tree node is Location, and there are five child nodes listed on the right, and each child node represents a choice for the question Location and the next node if we choose this location, like if we choose German-Colony for the question Location, we will go to the next_node1.

Part 2.

The open source python library we use to implement the Decision Tree algorithm is called **sklearn.tree.DecisionTreeClassifier().**

Compared to our own implementation, sklearn DecisionTreeClassifier makes building a decision tree model easy by reducing the amount of code. After importing the sklearn library along with pandas, we create a decision tree model to fit the training data with the label values. Then we use the model to predict the result of (occupied = Moderate; price = Cheap; music = Loud; location = City-Center; VIP = No; favorite beer = No). The predicted answer for this is also '**Yes**'.

The tree structure sklearn library builds is a little bit different. It uses a binary tree structure where there will only be two branches for each node. Also sklearn decision tree only works on numerical values so we have to translate the word choices of each attribute into numbers by using a labelencoder object sklearn provides.

Also we noticed something valuable that we could consider to improve our own decision tree construction. When preparing for the training data for sklearn decisiontree, the sklearn library suggests dividing the original dataset into one training set and one test set with 70% to be

training set and 30% as test set. We believe this is a good strategy for implementing decision tree algorithm, because it prevents the decision tree from being over-fitting. Over-fitting happens when the tree is strictly built to fit all samples in the original training set and it ends up with branches with strict rules of sparse data. The method used to address over-fitting problem is called pruning where we segregate dataset into training part and test part randomly.

In part1, we have a situation where we run out of attributes to make new branches. After our collective analysis, we believe the original training data contains conflicting rows. Randomly separating the original dataset helps to purify the trained decision tree and thus increases the accuracy.

Part 3.

There are many interesting applications of decision trees. The first example is that we can apply decision trees in business. The special feature of decision trees is helpful for management to make future decisions based on analytical techniques such as discounted cash flow and present value methods. Management can easily and clearly consider various action opinions through decision trees. The second example is the classification system for serial criminal patterns. This system can analyse the previous criminal records with decision trees to obtain the possible characteristics of criminal. It can get ten times the data summarized by the team of experienced detectives.

Reference:

[1] John.F Magee, “Decision Trees for Decision Making”, *Harvard Business Review* Available: *https://hbr.org/1964/07/decision-trees-for-decision-making*

[2] Kamal Dahbur and Thomas Muscarello, “Classification System for Serial Criminal Patterns”, *Artificial Intelligence and Law* 11(4):251-269, Dec-2003

Individual contribution:

He Chang, Ziqiao Gao, Fanlin Qin: Design tree structure and implement decision tree algorithm in python for part1.

Ziqiao Gao: Optimize, implemente and debug the decision tree algorithm and printing method in part 1.

He Chang: Research online and use open source python library to implement decision tree algorithm for part 2.

Fanlin Qin: Research for decision tree algorithm applications and complete part 3.