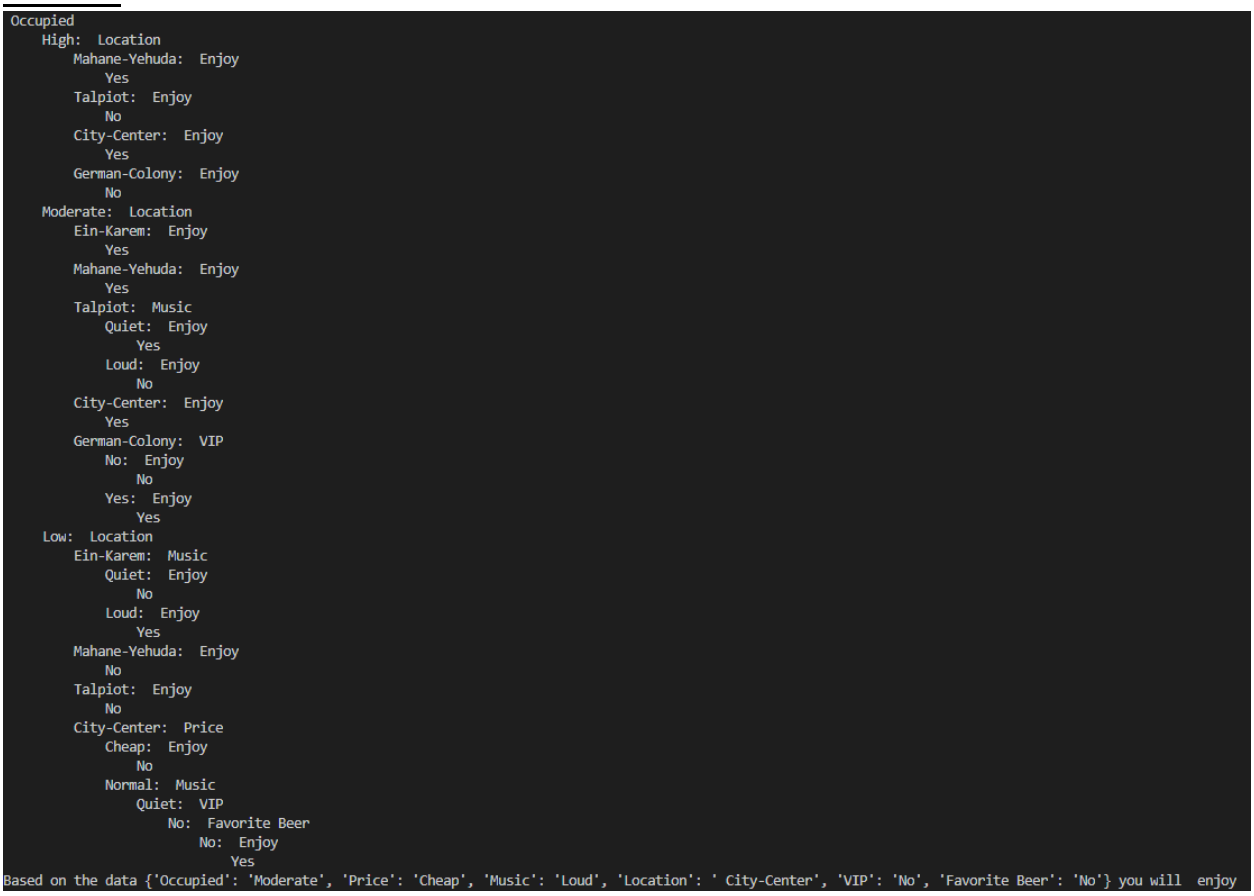
Name: Gaurav Baney

Student ID: 1258384414

Part 1



Prediction: Yes you will enjoy

Implementation:

I created two classes for my implementation – DecisionTreeNode and DecisionTree.

The Node class is used to keep track of node specific information such - attribute chosen for level, children nodes, the name of the branch that led to that attribute and whether we have an answer for our prediction or not.

The Tree class does all the work required for the building of the tree according to the algorithm and predicting the outcome of the desired search query. I used the pseudo coded ID3 algorithm on Wikipedia as my base to coding out this assignment.

Tree Methods –

parseData(file) – Method takes in a filename and returns a parsed version of the list of given attributes, target attribute(label), attribute values and a json version of the inputted data. I've used lists and dictionaries to keep track of most of the data for easy access, the only special structure would be a set used to keep track of the attributes.

predictLeafNode(data) – Using the data provided (subset of data at current node of tree), this function calculates a value for the label if the traversal of the tree were to stop.

dataEntropy(currentData) – Using the provided subset of the data this function calculates the entropy of the data using the formula learned in class

build() – starter function for the tree building process, once it's done building it calls print

buildhelper(node, remainingAttributes) – helper function that uses recursion to build the decision tree. It uses the information from the currentNode in conjuction with the information gain and entropy functions to figure out what the best attribute is. Once it's done that it splits the dataset and creates depth first branch. Function completes when its done building out the entire tree and returns when it hits a leaf node at which it assigns a prediction value.

findHighestInformationGain(node, currentEntropy, remainingAttributes) – Uses remaining attributes to find the attribute that gives us the highest gain value so that we can split on it.

findAttributeEntropy(currentData, currAttr) – Uses the formula learned in class to calculate the entropy of the attribute present in the current dataset.

printTree(node, indent) - Prints the tree in a root first and indented node structure following the format. Node : Selected Branch.

predictQuery(query, currNode) – Uses tree traversal to predict the label value for a passed in search query. If tree does not contain needed branch it returns majority result of label value at present node.

Challenges faced:

I decided to use python for this assignment as a way to get more comfortable with the language. A majority of my time went in figuring out language specific semantics. It was also somewhat difficult for me to translate the math so I followed the steps in:

https://www.youtube.com/watch?v=UdTKxGQvYdc&t=364s

Apart from that it was tiny bugs that I was trying to fix.

Part 2

A package I found that implemented this algorithm is –

https://pypi.org/project/decision-tree-id3/

Based on the documentation provided it looks like this implementation is a lot more robust when it comes to larger datasets with parameters that can be used to fine tune the result such as

max_depth	(int)
min_samples_split	(int)
prune	(bool)
gain_ratio	(bool)
min_entropy_decrease	(float)
is_repeating	(bool)

My implementation in comparison just runs with the base math and can't fine tuned to any length and cannot handle repeating elements in the tree.

Their tree visualization functions also have a lot more parameters and can be used to graph certain elements such as occurrences and traversal probabilities.

Part 3

Decision Trees have extremely high use cases. I have done a lot of game development and one use I can see if for AI decision making in easy to play games where you could map winning combinations of moves or even prune actions making it easier for an AI to decide what it wants to do in open world game.

Visual depictions of decision trees can even be used to in schools to help children understand simple yes / no style algorithms in their math and engineering classes.