Decision Tree Using ID3 Implementation and Applications

Chaoyu Li, chaoyuli@usc.edu, 6641732094

DSCI-552: Machine Learning for Data Science

Instructor: Satish Kumar Thittamaranahalli

Feb. 2, 2022

Table of Contents

Basic Algorithm and Code Functions Description

I implemented the decision tree with the ID3 algorithm. The libraries I used are *Math*, *Operator* and *Matplotlib.* The flow of the algorithm is as follows: First, the program will read data from dt_data.txt into a 2D list and a dictionary. The features of the decision tree are stored in the dictionary, and the corresponding values of the features in different cases are stored in the 2D list. Next, I use DFS to recursively build a decision tree, each time taking the feature with the largest entropy value as the current node of the tree and performing subtree recursion according to the possible values of this feature. I use a dictionary to hold decision trees. After completing the construction of the entire decision tree, I display the entire tree in the form of graphics through *matplotlib.pyplot*.

The major functions of my code are:

- **calcEnt(dataSet)**

It calculates the entropy for each feature. It accepts a 2D list as a dataset and calculates the entropy for the specific feature. It returns the value of entropy of the feature. The Math.log() is used here.

- **splitDataSet(dataSet,feature,value)**

It splits the dataset by one specific feature. It accepts the whole dataset, one integer number representing the index of the feature in the dataset, and the value of the feature that need to be splited. It returns a subset without the split feature.

- **chooseBestFeatureToSplit(dataSet)**

It decides which feature to split based on the entropy. It accepts the whole dataset and calls function calcEnt() to calculate the entropies of each feature. Then it calls function

splitDataSet() to split the feature with the maximum entropy. It returns the index of the best feature.

- **majorityCnt(feature)**

  It is used to deal with such a situation: when there are no any other features, but the remaining data is not exactly the same value, It returns the value with the most occurrences.

- **createTree(dataSet, labels)**

  It is the main function of building a decision tree. It is a DFS function and it accepts a 2D list as the dataset and a dictionary of labels representing the features of the dataset. For each iteration, it calls function chooseBestFeatureToSplit() to find the index of the best feature, and it uses the best feature as the current node of the decision tree. Then, it deletes the best feature from the dataset to get a subset to run the next level of recursion until there is no more feature. It stores the decision tree in a dictionary and returns it.

- **loadTXT(file)**

  It reads the dt_data.txt and splits it into two parts: features and values. It accepts the path of the dataset and returns a dictionary and a 2D list.

- **plotNode(string,centerPt,upPos, type, size)**

  It sets the position and style of the node on the plot. It accepts a string of text to display on the node, the center position of the node that need to be displayed, the position of the up level node, the style type of the node and the size of the node.

- **plotTextPosition(cntPos, upPos, string)**

  It finds the position to show the text between parent nodes and child nodes.

- **getNumLeafs(dTree)**

It counts the number of the leaf nodes of the decision tree to calculate the width of the tree on the plot. It is a DFS function.

- **getTreeDepth(dTree)**

It counts the depth of the decision tree to calculate the height of the tree on the plot. It is a DFS function.

- **plotTree(dTree, upPos, string)**

It displays the decision tree on the plot. It is a DFS function. It accepts the current decision tree dictionary, the position of the up level node, and the text between parent nodes and child nodes.

- **createPlot(dTree)**

It is the main function of displaying the whole decision tree. It accepts the decision tree dictionary and create the plot.

- **predictor(dTree, query)**

It uses the dfs algorithm to give a final “yes” or “no” result based on the input query. It accepts the decision tree dictionary and the query dictionary.

Data Structure Used

The implementation of the code has been done in Python 3.9. The data structures used for implementing the ID3 algorithm are:

- Dictionary:

Dictionaries in Python are essentially key value pairs. There are certain values which are mapped to specific keys. The decision tree has been implemented as a recursive dictionary *i.e.*, each key is the root child and the values of the key correspond to the children of the root. I use dictionary to store decision tree. The format as follow:

```
{'Occupied': {
        'Low': {
                'Location': {
                        'City-Center': {
                                'Price': {
                                        'Cheap': 'No',
                                        'Normal': {
                                                'Favorite Beer': {
                                                        'No': 'No',
                                                        'Yes': 'Yes'}}}
```

- 2D List:

  We have used 2D lists to store our training data. The format as follow:

  - Data: [['High', 'Expensive', 'Loud', 'Talpiot', 'No', 'No', 'No']...['Low', 'Cheap', 'Loud', 'Ein-Karem', 'Yes', 'Yes', 'Yes']]
  - Labels: ['Occupied', 'Price', 'Music', 'Location', 'VIP', 'Favorite Beer']

Challenges Faced and Optimizations

- Challenges:
    - There are a lot of DFS functions and their stop conditions are very sensitive. Therefore, it is not that easy to determine the boundary conditions.
    - Since DFS traverses the entire dataset, it takes much more time when the depth is deeper.
    - Sometimes it can be difficult to decide whether to store data in lists or dictionaries, because lists are easier to access and manipulate but dictionaries better preserve the structure of the data.
- Optimizations:
    - Pruning: Pruning of some of the branches can be done to optimize the tree structure and make it balanced. It reduces the time complexity efficiently.
    - Sets: Use sets appropriately to store data. Because the set can filter out duplicate elements and reduce the amount of computation.

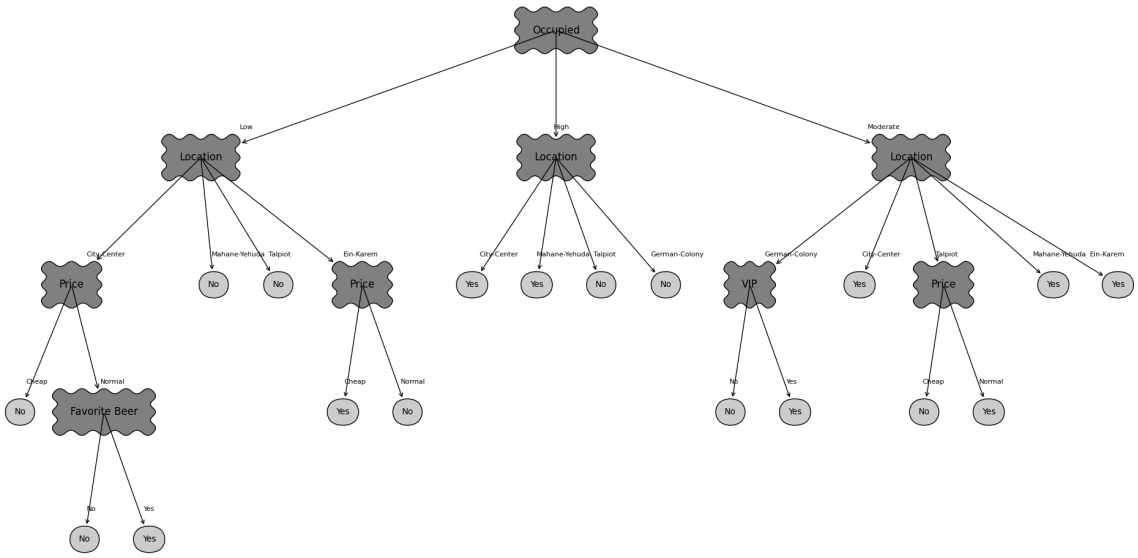
Execution of my algorithm and results

The output of my program is a decision tree in dictionary format and a .png image:

- Decision tree:

{'Occupied': {'Low': {'Location': {'City-Center': {'Price': {'Cheap': 'No', 'Normal': {'Favorite Beer': {'No': 'No', 'Yes': 'Yes'}}}}, 'Mahane-Yehuda': 'No', 'Talpiot': 'No', 'Ein-Karem': {'Price': {'Cheap': 'Yes', 'Normal': 'No'}}}}, 'High': {'Location': {'City-Center': 'Yes', 'Mahane-Yehuda': 'Yes', 'Talpiot': 'No', 'German-Colony': 'No'}}, 'Moderate': {'Location': {'German-Colony': {'VIP': {'No': 'No', 'Yes': 'Yes'}}, 'City-Center': 'Yes', 'Talpiot': {'Price': {'Cheap': 'No', 'Normal': 'Yes'}}, 'Mahane-Yehuda': 'Yes', 'Ein-Karem': 'Yes'}}}}

- Decision tree image:



- **Input the query:**

{'Occupied':'Moderate','Price':'Cheap','Music':'Loud','Location':'City-Center','VIP':'No','Favorite Beer':'No'}

Output: <u>Yes</u>

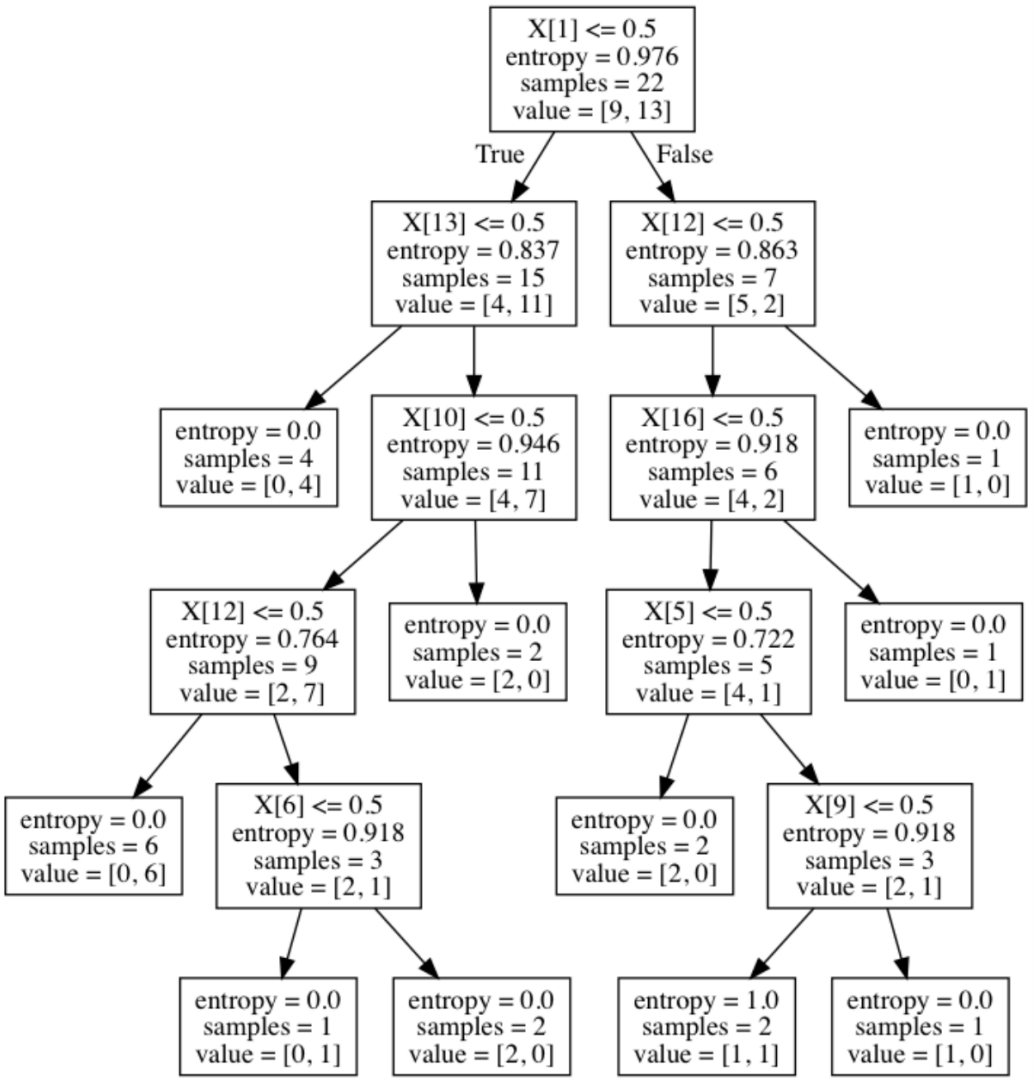
Comparison between my algorithm and scikit-learn

I found a good python library named scikit-learn which contains decision tree algorithm to compare with my algorithm. There are some differences between my algorithm and the decision tree algorithm in scikit-learn.

1. The structure of tree are different: the decision tree of my algorithm is a multi-forked tree; however, the scikit-learn algorithm will create a binary tree because sklearn's approach is to work with numerical features, not categorical.
2. The complexity of sklearn's algorithm is O(n), but the complexity of my algorithm is $O(n^2)$. Here n means the number of training examples. Therefore, it is obvious that the sklearn's algorithm is better than mine.
3. The sklearn's algorithm offer more methods to build a decision tree. For example, the decision tree can be implemented by calculating Gini Impurity instead of entropy. Actually I can improve my algorithm like this.

Implementation using scikit-learn

The decision tree displayed by sklearn's algorithm is shown below:



I inputed the test case to the sklearn's program and also got a "**Yes**".

Application of Decision Tree

1. Buy goods: For example, when buying a car, we consider factors such as budget, safety, color, speed, appearance, etc. For example, when buying a car, we consider factors such as budget, safety, color, speed, appearance, etc. Some of these factors are more important and some are less important, and together they form a decision tree. According to the decision tree, we can quickly filter out the cars that meet our expectations.
2. Medicine: Decision trees are widely used in the medical field. Doctors can use decision trees to quickly diagnose different diseases to increase the efficiency of diagnosis.
3. Observation of celestial bodies: Decision trees also have corresponding applications in astronomy. Scientists use decision trees to filter objects seen by Hubble. For example, they can quickly find out which of the observed objects are stars.

Contributions

Part	Contributer
ID3 algorithm	Chaoyu Li
Sklearn’s algorithm	Chaoyu Li
Application of Decision Tree	Chaoyu Li
Report	Chaoyu Li