Project Specification
COMP 3401
Fall 2019

Posted: Oct. 23
Due: Ecopy: 11:59pm, Nov. 28
    Hardcopy: In class, Nov. 28

1. Goal

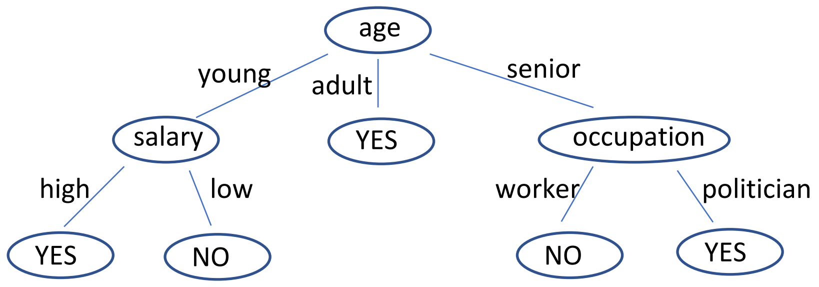
A decision tree is a data structure on which the classifier runs to make a prediction. Each internal node is attached with a 'testing attribute' and the branches are the outcomes for the testing. The leaf nodes are attached with the class labels. The object whose class label is being predicted starts from the root, and is guided at each node by the testing outcome to follow a specific path to a leaf node, where it is assigned the class label. The goal of this project is to implement a decision tree based on following application: customers are assessed for their *RISK* level, *high* or *low*, depending on the five attributes: age, credit_history, income, race, health. The subsequent section contains some details that you will need to know in completing your project.

2. How to represent a decision tree

A crucial step for building a decision tree is to select the testing attribute at each node. This has been discussed in detail in the class. An issue that has not been discussed in the class, however, is exactly how to use a data structure to represent a decision tree in a programming environment. This concerns with the implementation, and depends on the language you will use. In the general case, you need to represent the following information for each node: testing attribute, the values in the domain of the testing attribute, and for each branch the identity of the child that it leads to. In Python, you can use sequence type for this purpose. For each internal node, the list of the form below may be used:

$$[A, \{v_1: B_1, \ldots, v_p: B_p\}]$$

where A is the testing attribute at the node. The key-value pairs in the dictionary represent branches where v_i is a value in Domain(A) and B_i is a reference to a child. For a leaf node, simply use [L] to represent it where L is the class label attached to the node. For example, consider the following decision tree:



The above decision tree contains a total of eight nodes. They are represented as follows:

A: [‘age’, {‘young’: B_1, ‘adult’: B_2, ‘senior’: B_3}]
B_1: [‘salary’, {‘high’: C_1, ‘low’: C_2}]
B_2: [‘YES’]
B_3: [‘occupation’, {‘worker’: C_3, ‘politician’: C_4}]
C_1: [‘YES’]
C_2: [‘NO’]
C_3: [‘NO’]
C_4: [‘YES’]

In almost all the cases, the root is used to identify the entire decision tree. The above representation is easy to work with. For example, suppose each person is represented by a dictionary of the form {‘age’: v_1, ‘salary’: v_2, ‘occupation’: v_3}. To predict the class label for an individual p, we can simply use the following loop:

```
node ← A
while node is a list
    v ← p[node[0]]
    node ← node[1][v]
return node[0]
```

It’s equally easy to predict the class labels for a group of instances, such as those represented in a matrix. (Detail omitted.)

3. Encoding the values for nominal attributes

In many cases, you need to test the accuracy of the decision tree on a testing set. The testing set is usually represented as a mxn matrix where the rows are attributes and the columns are objects. In numpy, the most commonly used input function that operates on matrix is loadtxt(*). But this function requires the matrix to have numerical values while in most cases the values for a nominal attribute are symbols. One way to cope with this problem is to encode the symbols as numerical values. For example, the following is an encoding scheme for the above example:

{'buy':{'YES':1, 'NO':2}, 'age': {'young':1, 'adult':2, 'senior':3}, ‘salary’: {‘high’:1, ‘low’:2}, 'occupation': {'worker':1, 'politician':2}}

With the above encoding, the representation of the decision tree in section 2 will be the following.

A: [‘age’, {1: B_1, 2: B_2, 3: B_3}]
B_1: [‘salary’, {1: C_1, 2: C_2}]
B_2: [1]
B_3: [‘occupation’, {1: C_3, 2: C_4}]
C_1: [1]
C_2: [2]
C_3: [2]
C_4: [1]

If necessary, additional data structures can be used for decoding purpose, such as the following:

[[‘buy’, (1, 2)], [‘age’, (1,2,3)], [‘salary’, (1,2)], [‘occupation’, (1,2)]

{‘buy’: {1: ‘YES’, 2: ‘NO’}, ‘age’: {1: ‘young’, 2: ‘adult’, 3: ‘senior’}, ‘salary’: {1: ‘high’, 2: ‘low’}, ‘occupation’: {1: ‘worker’, 2: ‘politician’}}

The first list gives the encoded domain for each attribute, and the second tells how each value is decoded to its original symbol.

For your convenience, the data matrix I supply to you contains the already-encoded values.

4. How to store a decision tree in a file

A decision tree, once constructed, will be used many times. Therefore, it must be a persistent structure that survives program closure. One way to do it is to save it in a file. An important requirement here is that after we save it we must be able to read it back in exactly the same format as the one saved. Numpy provides a convenient mechanism, json, to just do that. For example, the following code saves a list to a file.

```
Import json

a = 1
b = 2
c = [‘abc’, {‘c’: a, ‘d’: b}]
with open(‘pathname’, ‘w’) as f:
    json.dump(c, f)
```

The list [‘abc’, {‘c’:1, ‘d’:2}] has been stored in a file located at ‘pathname’. If later on you want to retrieve that list, you can use

```
with open(‘pathname’, ‘r’) as f:
     v = json.load(f)
```

Now variable v contains a reference to the list [‘abc’, {‘c’:1, ‘d’:2}].

Note that there is a slight discrepancy between the read and the write versions if you save a dictionary with a numerical value as a key, being that the version you read will have the numerical keys be enclosed in a pair of quotes, i,e., their types become string now. For example, consider the following code.

```
a = 1
b = 2
c = [‘abc’, {1: a, ‘d’: b}]
with open(‘pathname’, ‘w’) as f:
    json.dump(c, f)
```

The first key in the dictionary within list c is 1, which is of integer type. When this list is read back:

```
with open(‘pathname’, ‘r’) as f:
     v = json.load(f)
```

you will find v refers to list [‘abc’, {‘1’: 1, ‘d’: 2}], where 1 as the key in the dictionary has been quoted, so is a string type. As a result, you should use function str() to change a numerical value to string type when you try to match it to a key which you bring back from a file using json.

5. Requirements

5.1. Task

The overall work related to your project consists of three parts: 1. Grow a decision tree to its maximum level based on a training set, and then test its accuracy on a test set; 2. Prune the decision tree based on the test set; 3. Plot the decision trees before and after pruning in two separate files. Your main task is growing the decision tree and testing accuracies for the full and pruned versions. I will supply functions for you to do the pruning and plotting the decision tree.

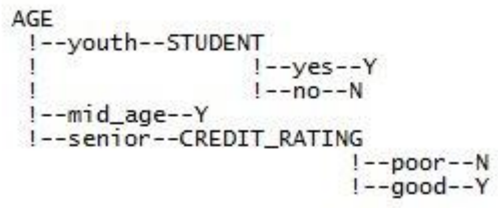
5.2. Organization of the file system for your project

Place all the work under a single folder with the name <your user id>. (The root folder in the file structure I provide to you has the name *decision-tree*, change it to <your user id>.) Under this folder, create two sub-folders with the names *program* and *data*. Put all the Python files in the *program* folder and all the data files in the *data* folder. Create a file, say, *growTree.py* to include all the functions that you will use to grow the tree. The programs for pruning and plotting should be placed into another two separate files. The following diagram is more specific for such a directory and file structure.

```
<your user id>
    program
        growTree.py
        evaluate.py
        pruneTree.py
        disp.py
    data
        train.txt
        test.txt
        deDomain.txt
        dataDesc.txt
        you-select.txt
        treeFilePruned.txt
        treePicFull.txt
        tree.PicPruned.txt
        readme.pdf
```

The Python files in red are the ones you must create as the main job for your project. The text files in red will be created when you run the four programs in *program* folder. I will provide all the files in black. The following is a description about the functionality of each file.

- *growTree.py*: grow a decision tree to its full depth. It contains a function, *main*(), which takes no parameter and returns the file name where you save the decision tree you created.
- *evaluate.py*: evaluate the accuracy of a decision tree. It contains a function, *main*(*fname*), where *fname* is the file in which a (full or pruned) decision tree is saved. The function returns the accuracy of the decision tree when running on the test set.
- *pruneTree.py*: prune a decision tree to its minimum depth. It contains a function, *main*(*fname*), where *fname* is the file where you save the full decision tree. It returns the name of the file where the pruned tree is stored.
- *disp.py*: Plot a decision tree. It contains a function, *showIt*(*fname*), where *fname* is the file name where you store the full decision tree. It returns the file name in which your decision tree is plotted. The plot for a decision tree is similar to the following:



```
AGE
 !--youth--STUDENT
 !                !--yes--Y
 !                !--no--N
 !--mid_age--Y
 !--senior--CREDIT_RATING
                        !--poor--N
                        !--good--Y
```

- *train.txt*: the training set. It is an nxm matrix where the rows are attributes and columns are objects. The values in the matrix are the encodings for the values in the original domains of the attributes.
- *test.txt*: the test set. The interpretation of its structure is the same as the above training set.
- *deDomain.txt*: It tells how to decode the attribute values. Refer to *readme.txt* for detail.
- *dataDesc.txt:* an interpretation for *train.txt* and *test.txt*. Refer to *readme.txt* for detail.
- *you-select.txt*: the file where you save the decision tree. Select a name at your discretion.
- *treeFilePruned.txt*: the file where the pruned decision tree is saved.
- *treePicFull.txt*: the file where your fully-grown decision tree is plotted.
- *treePicPruned.txt*: the file where the pruned decision tree is plotted.

Since the program and data sub-folders are under the same folder, whenever your programs need to access the data files, you should use relative path name '../data/filename'. This is necessary for marking purpose: if you use absolute path name, your program will not work on my machine.

5.3. Calling functions from Python shell

To ensure the modularity, and to make the debugging easier, do not include any code other than function definitions in your Python files. This means you can call a function only from the IDLE shell. Also, the functionality for each of the four modules in *program* folder is well separated. Their interaction should occur at the IDLE shell. Do not try to place one module inside the other, since doing so would make the interaction implicit.

To call a function defined in a file from IDLE shell, you must first import it. This in term requires the path that leads to the file be inserted manually into the sys.path. To avoid this hassle, you can click on run->run module. This automatically inserts the path for the current file into the sys.path, and also sets it as the current working directory.

A final remainder: to use the programs I supply to you, it is important that you represent the decision tree in exactly the same format as described in section 2, and use the training/testing sets in a compatible way to what the file *dataDesc.txt* describes, or else the supplied programs will not work.

5.4. A summery of your task

The following list summarizes your task for the project. (In the list, the functionalities of the files and the functions are described in section 5.2.)

1. Create *growTree.py*, and run *growTree.main()* to save the fully-grown decision tree to a .txt file with a name of your choice.
2. Create *evaluate.py*, and run *evaluate.main(fname1)* where fname1 is the file name in which you save your fully-grown decision tree, and print the accuracy in the IDLE shell.
3. Call *disp.showIt(fname1)* where *fname1* is the file name in which you save your fully-grown decision tree. This call plots your fully-grown decision tree in file *treePicFull.txt*.
4. Call *pruneTree.main(fname1)* where *fname1* is the file name in which you save your fully-grown decision tree. This call prunes your decision tree to the minimum depth, and save the pruned decision tree in file *treeFilePruned.txt*.
5. Call *disp.showIt(fname2)* where *fname2* is the file name in which the pruned decision is saved, i.e, *treeFilePruned.txt*. This call plots the pruned decision tree in file *treePicPruned.txt*.
6. Call *evaluate.main(fname2)* where *fname2* is the file name in which the pruned decision is saved, i.e, *treeFilePruned.txt.* This call prints the accuracy of the pruned decision tree to the IDLE shell.

5.5. A summary of specific requirements in your programming

This list of requirements is for purpose of ‘regularizing’ the setting for your program. The reason for such a regularization is simple: you do the project on your own machines, and I will mark your projects by running your programs on my machine. If everyone uses different setting, marking is not possible.

1. Use exactly the same format as was described in section 2&3 to represent a decision tree where the attribute values should be encoded as integers.
2. Explained in file *readme.pdf* is how to use file *dataDesc.txt* to interpret the train.txt and test.txt. Follow the explanation there when you manipulate the contents in train.txt and test.txt.
3. Whenever your programs need to access a file in a different folder, say *anyfile.txt* in *data* folder, use the relative path name like ‘../data/*anyfile.txt*’. DO NOT use absolute path.
4. The parameters for functions *main*(*) in *evaluate.py*, *showIt*(*) in *disp.py*, and *main*(*) in *pruneTree.py*, are plain file names, like *anyfile.txt* in the *data* folder. Do not include path in it like ../data/anyfile.txt. (Refer to section 5.2 for the functionalities of these functions.)
5. Do not include any code other than definitions for functions in files *growTree.py* and *evaluate.py*.

5.6. Submission

E-submission: the entire directory tree with the structure described in Section 5.2, zipped in a single .zip file.

Hardcopy: A report of the project that includes the following:

1. Print your and your patner's name, students' id, on the cover page
2. screenshots of the following function calls:
   - v1 = growTree.main(), evaluate.main(v1), disp.showIt(v1), v2 = pruneTree(v1), evaluate(v2), disp.showIt(v2) on the IDLE shell
   - The outputs from the two calls for evaluate.main(*) to the IDLE shell.
3. The following .txt files:
   - The file you save your fully-grown decision tree.
   - *tree.PicFull.txt*
   - *tree.PicPruned.txt*