VNUHCM – UNIVERSITY OF SCIENCE

FALCUTY OF INFORMATION TECHNOLOGY

ARTIFICIAL INTELLIGENT

Lab 02: Decision Tree

Teacher: **Nguyễn Ngọc Thảo**

Lê Ngọc Thành

Nguyễn Trần Duy Minh

Nguyễn Hải Đăng

Contents

I. Student information

- Full name: Võ Nguyễn Hoàng Kim
- Student ID: 21127090
- Class: 21CLC07

II. Complete progress

No.	Specification	Completion
1	Preparing the data sets	100%
2	Building the decision tree classifiers	100%
3	Evaluating the decision tree classifiers Classification report and confusion matrix Comments	100%
4	The depth and accuracy of a decision tree Trees, tables, and charts Comments	100%

III. Description

In this assignment, the student have to build a decision tree on the UCI Nursery Data Set, with support from the scikit-learn library.

There are 12960 records in the data set. Nursery Database was derived from a hierarchical decision model originally developed to rank applications for nursery schools.

The dataset is downloaded from: https://archive.ics.uci.edu/dataset/76/nursery.

IV. Libraries

In this program, I've used these libraries:

- import matplotlib.pylot as plt.
- import pandas as pd.
- seaborn as sns.
- scikit-learn:
    - from sklearn.model_selection import train_test_split
    - from sklearn.preprocessing import LabelEncoder
    - from sklearn.tree import DecisionTreeClassifier
    - from sklearn import tree
    - from sklearn.metrics import classification_report, confusion_matrix,accuracy_score
- graphviz

V. Solving assignment

1. Preparing the data sets.

I downloaded the dataset from https://archive.ics.uci.edu/dataset/76/nursery (to prepare data sets) and put it in the "input" folder. These files in this folder are the original files (not be added extension "csv").

For ease to use, I made a copy of the file "nursery.data" in the same directory as the source code and added the "csv" extension (it becomes "nursery.data.csv") to read the file.

In my code, I read the data set by name, therefore, the data set file must be in the same directory as the source code. If not, the data sets can not be read and maybe arise some problems (error)

a. Read data sets.

Because "nursery.data.csv" just store the data without label, I create a *label_data* includes labels of the data set.

Read the data set with *label_data*, we have 9 columns (include 8 features and 1 class). Then, I put these columns in the corresponding variables:

- X: 8 columns of feature.
- y: the last column of class.

Create function **preDataSet**(*trainSize, testSize*):

- This function has 2 parameters: size of training set, size of test set.
- Using **train_test_split** (from libraries) to split training set and test set with the corresponding size (from parameters).

There are 4 proportions, including (train/test): 40/60, 60/40, 80/20, 90/10, thus I have a list *listProportion* to store 4 proportions.

Use **for** loop to split data by elements in *listProportion*, each of proportion we have 4 subsets: *feature_train*, *feature_test*, *label_train*, *label_test*.

- *feature_train:* a set of training examples, each of which is a tuple of 8 attribute values (target attribute excluded).
- *feature_test*: a set of labels corresponding to the examples in *feature_train.*
- *label_train:* a set of test examples, it is of similar structure to *feature_train*
- *label_test*: a set of labels corresponding to the examples in *feature_test*

I use a dictionary *subset* to store subsets of each proportion. Thus, there are 4 elements in the *subset* list and 16 subsets, in total, for 4 proportions.

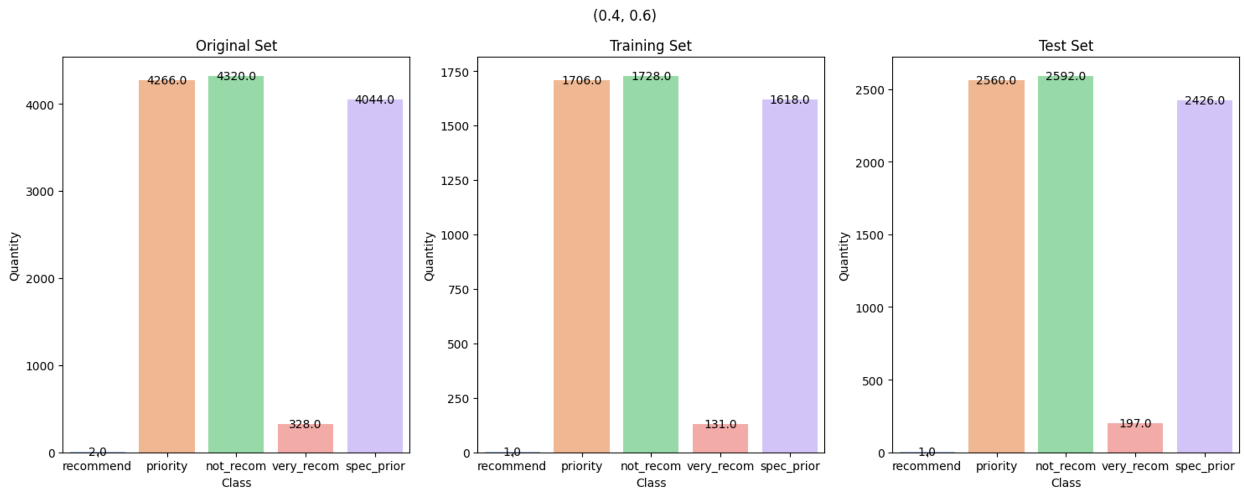
b. Visualization

To show that I have prepared the data set appropriately, I had visualized the distributions of classes in all the data sets (the original set, training set, and test set) of all proportions.

I use the bar chart to plot the data set. In this chart, axis x presents data in class column (y), axis y presents the quantity of each.

For each proportion, I have 3 charts to present for the original set, training set, and test set.

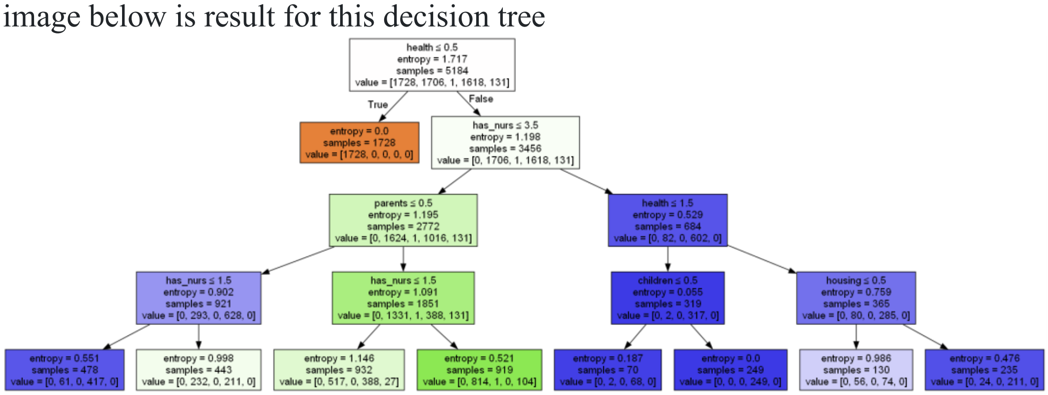
The image below is the result of visualization for proportion 40/60 (train/test).



2. Build the decision tree classifiers.

With each proportion from the *subset* above:

- Initialize LabelEncoder instances for categorical features.
- Build the decision tree by **DecisionTreeClassifer** with *criterion* is "entropy" and set this tree with the *max_depth* is 5. With the *max_depth* value, I have tried to set this with "None", but the exported image with the tree is hard to see. Because with "None", nodes are expanded until all leaves are pure, the tree is too large. Therefore, I set this *max_depth* smaller to fit the scale of image.
- Use **fit** (in libraries) to build a decision tree classifier from the *(feature_train*, *label_train)*.
- Use **export_graphviz** to set some tree's information (such as feature names)
- I save this result of decision tree in format "png". Besides, use **render** to render this image in the folder "DecisionTree" with the name "Decision_Tree_<corresponding proportion>". For instance, with the proportion 40/60 (train/test), the image for this decision tree will be rendered in folder "DecisionTree" with the name "Decision_Tree_(0.4,0.6).png". And this image below is result for this decision tree



3. Evaluating the decision tree classifiers.

a. Interpret idea.

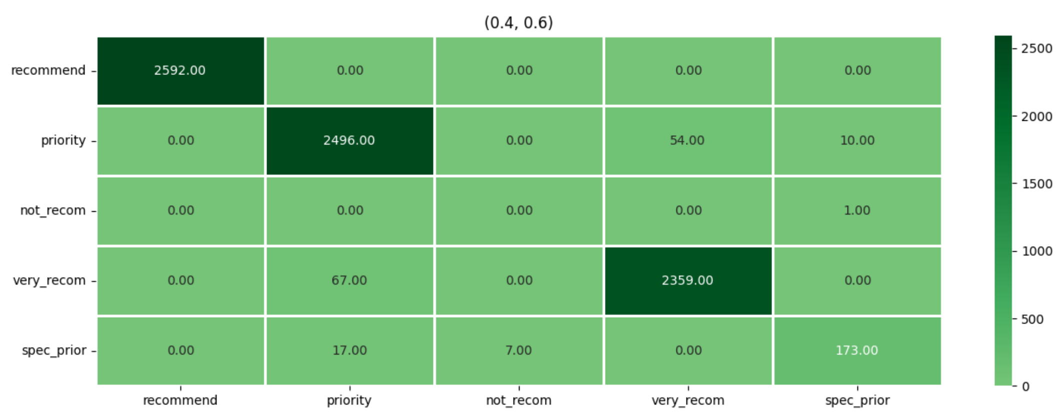
For each proportion in the *subset* dictionary above:

- Initialize LabelEncoder instances for categorical features (for *feature_train* and *feature_test*).
- Build the decision tree by **DecisionTreeClassifer** (in libraries) with *criterion* is "entropy".
- Use **fit** (in libraries) to build a decision tree classifier from the *(feature_train*, *label_train)*.
- Use **predict** (in libraries) to make predictions.
- Use **classification_report** (in libraries) to make the report for each proportion and **confusion_matrix** (in libraries) to create the confusion matrix.
- This is the classification report and confusion matrix for proportion 40/60 (train/test)

```
Classification Report for (0.4, 0.6) proportion:
              precision    recall  f1-score   support

   not_recom       1.00      1.00      1.00      2592
    priority       0.97      0.97      0.97      2560
   recommend       0.00      0.00      0.00         1
  spec_prior       0.98      0.97      0.97      2426
  very_recom       0.94      0.88      0.91       197

    accuracy                           0.98      7776
   macro avg       0.78      0.77      0.77      7776
weighted avg       0.98      0.98      0.98      7776
```



b. Comments

The confusion matrix is a great way to examine and verify the data presented in the classification report. Hence, we can compute some attribute:

- **Precision** is the percentage of correct positive predictions relative to total positive predictions.
- **Recall** is the number of true positives divided by the sum of true positives and false negatives.
- **F1 score** is the harmonic mean of precision and recall.
- **Support** is the number of samples in each class.

The reported averages include:

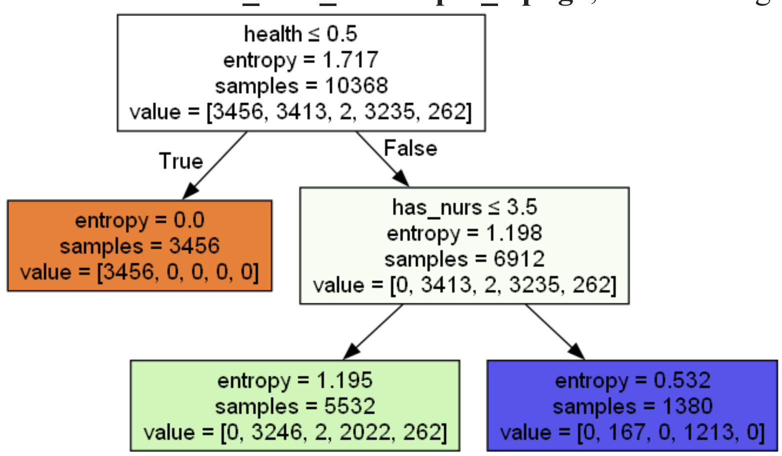
- **Accuracy**: is a general measure of how well the classification performs for all classes.
- **Macro Average:** When you want to assess the effectiveness of the classifier without considering the class imbalance, macro average is helpful.
- Weight Average It is especially helpful when working with datasets that are unbalanced since it takes each class's effect into consideration based on how prevalent it is.

➔ From this definition about these attributes, I can conclude that: with accuracy, we just have a general observation of dataset, while macro average and weight average do it better. Thus, if we want to adjust a model (or a data set), we need to consider the two attributes above. It will generate a complete evaluation.

4. The depth and accuracy of a decision tree.

a. Build the code.

- At this task, I use the proportion 80/20 (train/test) to build decision trees with different *max_depth*: None, 2, 3, 4, 5, 6, 7.
- To build the decision tree with these depth, I repeat steps (in 2 or 3), I have trees with the depth None, 2, 3, 4, 5, 6, 7, respectively and store it into *treeList*.
- Use **render** (in libraries) to render these tree results in image (png file).
- I save this image in the folder "DecisionTree_Tree_Accuracy" with the name: "Decision_Tree_MaxDepth_<corresponding *max_depth*>". For instance, with *max_depth* = 2, the image name will be "**Decision_Tree_MaxDepth_2.png"**, and the image will be



- Use **accuracy_score** (in libraries) to compute the accuracy. For each accuracy score, I have round it to 3 decimals.
- Then, I have the table below:

Max_depth	None	2	3	4	5	6	7
Accuracy	0.993	0.764	0.825	0.861	0.876	0.891	0.918

b. Comments

- From the table above, we can observe that: with the max_depth = 2, the accuracy is lowest, but when we change the max_depth (increase), the accuracy will be greater. And the best value for accuracy is max_depth = None, with the corresponding accuracy is 0.993 (approximately 1, almost absolute)

→ Thus, we can conclude that the max_depth affects accuracy.

- However, to get the high accuracy, we need to spend a lot of time to build the decision tree. The nodes will be expanded until all leaves are pure. With the small data, this may not take much time, but with the large data, it will arise some problems.

VI. References

1. https://machinelearningcoban.com/2017/08/31/evaluation/
2. https://websitehcm.com/confusion-matrix-la-gi-cac-yeu-to-quan-trong/
3. https://scikit-learn.org/stable/index.html
4. https://matplotlib.org/