Introduction to ML - Decision Tree Coursework

October 2020

1 Overview

In this assignment, you will implement a decision tree algorithm and use it to determine one of the indoor locations based on WIFI signal strengths collected from a mobile phone. See Figure 1 for an illustration of the experimental scenario. The results of your experiments should be discussed in the report. You should also deliver the code you have written.

Deadline: Fri - 06 Nov 2020 (7pm) on CATE

2 Setup

We do recommend you work on the Ubuntu workstations in the lab. This assignment and all code were tested for Linux and Mac OS machines. We cannot guarantee compatibility with Windows machine and cannot promise any support if you do choose to work on a Windows machine.

2.1 Working on DoC lab workstations (recommended)

You can also work from home and use the lab workstations. See this list of `https://www.doc.ic.ac.uk/csg/facilities/lab/workstations` to ssh into one of the machines.

In order to load all the packages that you might need for the course work, you can run the following command:

```
export PYTHONUSERBASE=/vol/lab/ml/mlenv
```

We installed numpy, matplotlib, pytorch, which should cover most of the packages you will need for all the courseworks in this course. If you don't want to type that command every time that you connect to your lab machine, you can add it to your bashrc:

```
echo "export PYTHONUSERBASE=/vol/lab/ml/mlenv" >> ~/.bashrc
```

This way, every terminal you open will have that environment variable set. It is recommended to use ”python3” exclusively. The current python3 version in lab machines is 3.8.5. To test the configuration:

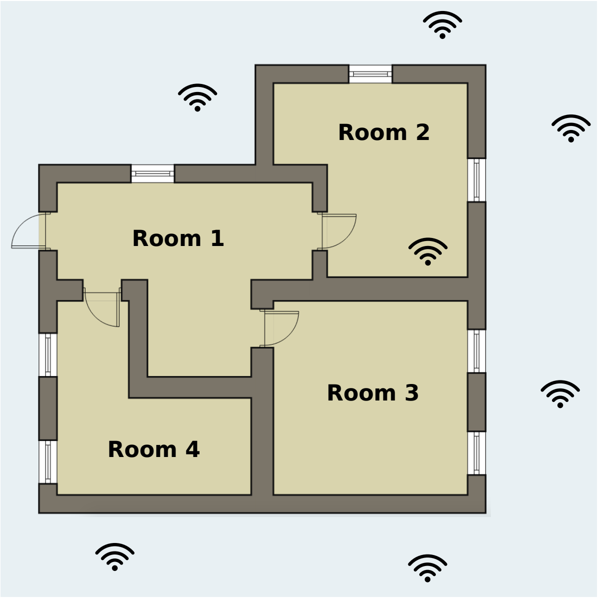

Figure 1: Illustration of the scenario. The WIFI signal strength from 7 emitters are recorded from a mobile phone. The objective of this coursework is to learn decision tree that predict in which of the 4 rooms the user is standing.

```
python3 -c "import numpy as np; print(np)"
```

This should print:

```
<module 'numpy' from '/vol/lab/ml/mlenv/lib/python3.8/site-packages/numpy/__init__.py'>
```

2.2 Working on your own system

If you decide to work locally on your machine, then you will need to give us explicit instructions how to run your code. Anything we cannot run will result to your points of the question reduced by 30%.

Python>= 3.5: All provided code has been tested on Python versions 3.5 or 3.6. Make sure to install Python version 3.5 or 3.6 on your local machine. Otherwise, you might encounter errors! If you are working on Mac OSX, you can use Homebrew to brew install python3.

Part 1: Implementation

Step 1: Loading data

You can load the datasets from the files "WIFI_db/clean_dataset.txt" and "WIFI_db/noisy_dataset.txt". They contain a 2000x8 array. This array represents a dataset of 2000 samples. Each sample is composed of 7 WIFI signal strength while the last column indicates the room number in which the user is standing (i.e., the label of the sample). **All the features in the dataset are continuous except the room number.** You can load the text file with the "loadtxt" function from Numpy. Given the nature of the dataset you will have to build decision trees capable of dealing with continuous attributes and multiple labels.

Step 2: Creating Decision Trees

To create the decision tree, you will write a recursive function called *decision_tree_learning()*, that takes as arguments a matrix containing the dataset and a *depth* variable (which is used to compute the maximal depth of the tree, for plotting purposes for instance). The label of the training dataset is assumed to be the last column of the matrix. The pseudo-code of this function is described below. This pseudo-code is taken from *Artificial Intelligence: A Modern Approach* by Stuart Russell and Peter Norvig (Figure 18.5), but modified to take into account that the considered dataset contains continuous attributes (see section 18.3.6 of the book).

Algorithm 1 Decision Tree creating

```
procedure DECISION_TREE_LEARNING(training_dataset, depth)
    if all samples have the same label then
        return (a leaf node with this value, depth)
    else
        split← FIND_SPLIT(training_dataset)
        node← a new decision tree with root as split value
        l_branch, l_depth ← DECISION_TREE_LEARNING(l_dataset, depth+1)
        r_branch, r_depth ← DECISION_TREE_LEARNING(r_dataset, depth+1)
        return (node, max(l_depth, r_depth))
    end if
end procedure
```

The function FIND_SPLIT chooses the attribute and the value that results in the highest information gain. Because the dataset has continuous attributes, the decision-tree learning algorithms search for the split point (defined by an attribute and a value) that gives the highest information gain. For instance, if you have two attributes (A0 and A1) with values that both range from 0 to 10, the algorithm might determine that splitting the dataset according to "A1>4" gives the most information. An efficient method for finding good split points is to sort the values of the attribute, and then consider only split points that are between two examples in sorted order, while keeping track of the running totals of positive and negative examples on each side of the split point.

To evaluate the information gain, suppose that the training dataset S_{all} has K different labels. We can define two subsets (S_{left} and S_{right}) of the dataset depending on the splitting rule (for instance "A1>4") and for each dataset and subset, we can compute the distribution (or probability) of each label. For instance, $\{p^1 p^2 \dots p^K\}$ (p^k is the number of samples with the label k divided by the total number of samples from the initial dataset). The information gain is defined by using the general definition of the entropy as follow:

$$\text{Gain}(S_{\text{all}}, S_{\text{left}}, S_{\text{right}}) = H(S_{\text{all}}) - \text{Remainder}(S_{\text{left}}, S_{\text{right}})$$

$$H(dataset) = -\sum_{k=1}^{k=K} p_k * \log_2(p_k)$$

$$\text{Remainder}(S_{left}, S_{right}) = \frac{|S_{\text{left}}|}{|S_{\text{left}}| + |S_{\text{right}}|} H(S_{\text{left}}) + \frac{|S_{\text{right}}|}{|S_{\text{left}}| + |S_{\text{right}}|} H(S_{\text{right}})$$

Where $|S|$ represents the number of samples in subset S.

Implementation:

You are only allowed to use Numpy and Matplotlib for this coursework. Any other library, like scikit learn is NOT allowed. For the implementation of the tree (in Python), it is advised to use dictionaries to store nodes as a single object, for instance by using this kind of structure {'attribute', 'value', 'left', 'right'}. In this case, 'left' and 'right' will also be nodes. You might also want to add a Boolean field name "leaf" that indicates whether or not the node is a leaf (terminal node) or not. However, this is not strictly necessary, as there are other methods to determine if a node is terminal or not.

Bonus (5 points): Write a function to visualize the tree. See Figure 2 for an example. You can only use Numpy and Matplotlib for this question as well.

Step 3: Evaluation

Now that you have an automatic process to train decision trees, you can evaluate the accuracy of your tree on the provided datasets. For that, evaluate your decision tree using 10-fold cross validation on both the clean and noisy datasets. You should expect that slightly different trees will be created per each fold, since the training data that you use each time will be slightly different. Use your resulting decision tree to classify your data in your test set.

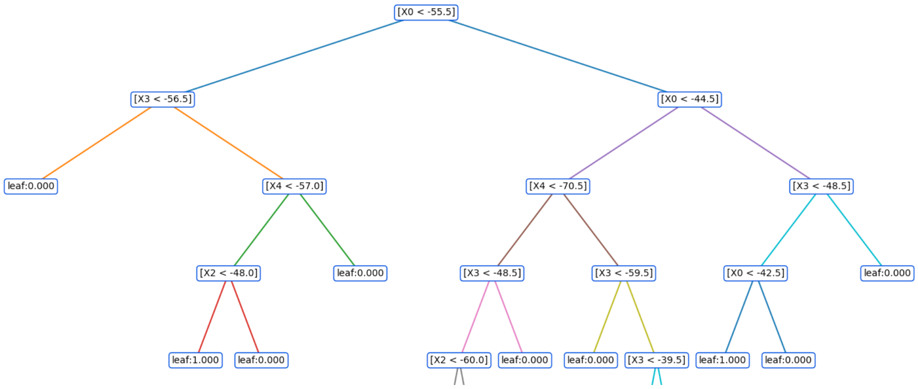

Figure 2: example of Decision tree visualization (not all the tree is displayed, and this tree has been trained on a different dataset.)

Implementation:

Implement an evaluation function that takes a trained tree and a test dataset: evaluate(test_db, trained_tree) and that returns the accuracy of the tree.

Write report the average cross validation classification results (for both clean and noisy data):

- Confusion matrix. (Hint: you should get a single 4x4 matrix)
- Average recall and precision rates per class. (Hint: you can derive them directly from the previously computed confusion matrix)
- The F1-measures derived from the recall and precision rates of the previous step.
- Average classification rate (NOTE: classification rate $= 1 -$ classification error).

Comment on the results of both datasets, e.g. which rooms are recognized with high/low accuracy, which rooms are confused.

Step 4: Pruning (and evaluation again)

In order to reduce the performance difference of our decision tree between the clean and noisy dataset, you will implement a pruning function based on reducing the validation error. This approach works as follow: for each node directly connected to two leaves, evaluate the benefits on the validation error of substituting this node with a single leaf (defined according to the training set). If a single leaf reduces the validation error, then the node in pruned and replaced by a single leaf. The tree needs to be parsed several times until there is no more node connected to two leaves (HINT: when you prune a node, the parent node might now verify this condition).

Compare the performances of your tree before and after pruning on a 10-fold cross validation and comment in your report.

Part 2: Questions

Answer to the following questions in your report:

Noisy-Clean Datasets Question Is there any difference in the performance when using the clean and noisy datasets? If yes/no explain why. Discuss the differences in the overall performance and per class.

Pruning Question Briefly explain how you implemented your pruning function and details the influence of the pruning on the accuracy of your tree on both datasets.

Depth Question Comment on the maximal depth of the tree that you generated for both datasets and before and after pruning. What can you tell about the relationship between maximal depth and prediction accuracy?

Part 3: Deliverables

For the completion of this part of the coursework, the following has to be submitted electronically via CATE:

- All the code you have written
- A README file (in txt) to explain how to run your code.
- A report of approximately 4-5 pages (excluding figures and tables) containing the following:
  - Brief summary of implementation details (e.g., how you performed cross-validation, how you selected the best attribute in each node, how you compute the average results, anything that you think it is important in your system implementation);
  - Commented results of the evaluation including the average confusion matrix, the average classification rate and the average precision, recall rates and F1-measure for each of the four classes; for both clean and noisy datasets.
  - Diagrams of the trees trained on the entire dataset (bonus points)
  - Answers to the three questions of Part 2.

Grading scheme

Final Grade = 0.75* Report content + 0.15* Code quality + 0.1* Report quality

- Code (total : 100)
  - Results on new test data: 50 (Make sure that your code runs. If not, you will be asked to resubmit the code and lose 30% of the code mark)
  - Presentation of the code (comments, indentation, structure): 20
  - README instructions: 30
- Report content (total : 100)
  - Implementation details : 20
  - Confusion matrix: 7
  - Recall/precision/F measure/Classification rate: 8
  - Analysis of the cross validation evaluation: 10
  - Answer to the clean-noisy question: 15
  - Answer to the pruning question: 20
  - Answer to the depth Question: 20
  - Visualization of tree: BONUS 5
- Report quality (total : 100)
  - Quality of presentation.

Acknowledgements

This coursework is inspired from the coursework designed by Maja Pantic.