CS 6350/DS 4350: HW2-Q2 Decision Trees on the Nursery Dataset

Samir Abdelrahman

Yujin Song, Shubham Sanjay Sawant

Fall 2025

1 Overview

In this assignment, you will explore the fundamental concepts of decision tree learning by implementing and evaluating classifiers on the Nursery dataset. The goal of this exercise is not only to give you hands-on experience with building a machine learning model from scratch, but also to strengthen your understanding of key ideas such as entropy, information gain, hyperparameter selection, and cross-validation. The concept of Cross-Validation has already been covered in our lectures. Please review the relevant lecture slides for background before starting this part of the homework.
By the end of this assignment, you should be able to
(1) implement baseline and decision tree models,
(2) analyze their performance on training and test datasets,
(3) apply cross-validation to tune model parameters.
This assignment emphasizes both correctness of implementation and clarity of reporting your results.

2 Dataset

We will work with the Nursery dataset from the UCI Machine Learning Repository (`https://archive.ics.uci.edu/dataset/76/nursery`). This dataset was derived from a hierarchical decision model used to rank applications for nursery schools. Each row in the dataset has the following attributes:

- Parents: {pretentious, great_pret, usual}.
- Has_nurs: {critical, very_crit, less_proper, proper, improper}.
- Form: {incomplete, foster, completed, complete}.
- Children: {1, 2, 3, more}.
- Housing: {critical, less_conv, convenient}.
- Finance: {inconv, convenient}.
- Social: {problematic, nonprob, slightly_prob}.
- Health: {priority, not_recom, recommended}.

The goal is to classify applications into one of five classes: `not_recom`, `recommend`, `very_recom`, `priority`, `spec_prior`.

3 Files and Directories Provided

The following files and directories are provided to you for this homework:

- **model.py**: Implement the `MajorityBaseline` and `DecisionTree` models.
  - Implement the train() and predict() logic for the MajorityBaseline model first. Once you've completed this, and implemented the necessary functions in train.py below, you should be able to train and evaluate your MajorityBaseline model. See README.md for more details.
  - Implement the train() and predict() logic for the DecisionTree model. You should be able to train and evaluate your DecisionTree model by running the completed train.py script. See README.md for more details.

- **train.py**: Contains training and evaluation code. You must implement `calculate_accuracy()` method to receive full points..
  - You may run train.py to test and debug your implementations. See README.md for more details on how to run this script. It's up to you whether you want to use the train() and evaluate() functions in this file (we strongly recommend that you do).

- **cross_validation.py**: Implement `cross_validation()`.
  - This file contains the code for running cross-validation on the decision tree model. See README.md for details on how to run this script.

- **data.py**: Contains helper methods for reading training, test, and cross-validation datasets. You do not need to make any changes to this file.

- **data/**: This directory contains the dataset files required for the assignment. You do not need to make any changes to this file.
  - train.csv: The main training dataset.
  - test.csv: The test dataset to evaluate your models.
  - cv/fold1.csv, fold2.csv, ..., fold5.csv: Contains 5 files, one for each fold during cross-validation. These files are created by splitting the train.csv dataset into 5 folds. You will use these files for cross-validation experiments.

4 Tasks and Points Distribution (60 points)

Task	**Points**
Accuracy (implement `calculate_accuracy`)	6
Majority Baseline Accuracy	8
Simple Decision Tree (no depth limit, entropy)	14
Decision Tree with Cross-Validation (depth 1–6)	20
Decision Tree with Best Depth from CV	12
Total	**60**

4.1 Accuracy [6 points]

Implement the method `calculate_accuracy()` in `train.py`. This should compute accuracy of predictions compared to ground truth labels.

4.2 Majority Baseline Accuracy [8 points]

Implement the `MajorityBaseline` class in `model.py`. You will also implement the generic `train()` and `evaluate()` functions in `train.py` that will be used to run all models in this assignment.

- Clearly state the majority class and the corresponding accuracy.
- Implement the `MajorityBaseline` class in `model.py`, and `train()` and `evaluate()` functions in `train.py`.
- Report accuracy on both train and test splits.

In your report, address the following:

- The label distribution seems pretty imbalanced. Why might accuracy be a bad metric for measuring the quality of a model on this dataset?

4.3 Simple Decision Tree [14 points]

Implement the `DecisionTree` class in `model.py`. The generic `train()` and `evaluate()` functions you implemented in the previous step should work for your decision tree model without any changes.
The decision tree classifier that:

- does not have any depth limit.
- is trained on the training data using entropy as the information gain criterion.
- is evaluated on both the train and test data once it's trained.

If you do it right, you shouldn't have to change anything in train.py from your majority baseline implementation.

- Report your accuracy on both the train and test splits.

4.4 Decision Tree with Cross-Validation [20 points]

Implement the depth limit in your `DecisionTree` class in `model.py`, and the cross-validation logic in `cross_validation.py`. For this task, your goal is to find the optimal `depth_limit` for your decision tree and analyze its performance.

In your `cross_validation.py` script, you will test a range of `depth_limit` values from 1 to 12. After running your experiment, you must provide two things in your report:

- The Result: State the best depth you found and its corresponding average cross-validation accuracy.
- The Analysis: You will likely observe that the accuracy stops improving after a certain depth. In a separate paragraph, provide a brief analysis explaining your hypothesis for why this performance plateau occurs. Consider what happens to the tree's growth and the purity of the nodes when the depth becomes sufficiently large. (Hint: Think about the natural stopping conditions of the decision tree algorithm).

4.5 Decision Tree with Best Depth from CV [12 points]

Use your optimal depth limit learned during cross-validation, re-run your training and evaluation code in train.py

- Report your accuracy on both the train and test splits.

Your submission will be evaluated on:

- The accuracy of your decision tree on the training and test splits.
- The accuracy on a hidden test split (provided by us during grading).

5 Bonus: Decision Trees with Collision Entropy [10 points]

Closely related to the Gini Impurity is the Collision Entropy which can be defined as:

$$\text{CollisionEntropy} = -\log_2\left(\sum_i p_i^2\right)$$

5.1 Simple Decision Tree with Collision Entropy [3 points]

Implement a decision tree classifier that:

- does not have any depth limit.
- is trained using Collision Entropy as the information gain criterion

- is evaluated on both the train and test data once it's trained.

You should be able to tweak your existing decision tree code in model.py, using the self.ig_criterion variable to decide whether to use entropy or collision entropy. You shouldn't need to change your code train.py if you've already completed Sections.
See README.md for instructions on how to run train.py with the –ig_criterion or -i_flag.

- Report your accuracy on both the train and test splits.

5.2 Decision Tree with Cross-Validation [3 points]

Run cross-validation on your Collision Entropy decision tree using the cross-validation folds we've provided and depth limit values [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12].
You shouldn't need to change your code cross_validation.py if you've already completed Section.

- Report the optimal depth limit learned from cross-validation and the corresponding best cross-validation average accuracy.

See REAMDE.md for instructions on how to run cross_validation.py with the –ig_criterion or -i_flag. This flag is only needed for the Bonus part.

5.3 Decision Tree with Best Depth from CV [4 points]

Re-run your training and evaluation code from train.py with Collision Entropy, but using your optimal depth limit learned during cross-validation.

- Report your accuracy on both the train and test splits.

6 Programming Notes

1. We strongly encourage you to use Python. If you choose not to, see the note at the end of this document. For this assignment's coding questions, you will upload your file code in Canvas.
2. You are not allowed to use any machine learning libraries (e.g., scikit-learn, tensorflow, pytorch), but can use libraries with mathematical functions like `numpy` and data management libraries like `pandas`. By default, you can use any packages provided in the `requirements.txt` file. We will evaluate your code using Python 3.12.8, but you can use any version $\geq$ 3.7. If you want to use an additional package not specified, we must approve it first.

7 Submission Instructions

You will submit two components on Canvas:

- **Report**: Submit your report containing answers to the questions, your findings/results from the coding sections, and screenshots. It needs to be included in the assignment2 folder.
- **Code**: Submit the full zip file we provide on Canvas if you are working in Python.
- **Report Format template**: We will give you two format(docx and latex) template, you can modify it to complete your report. You can also use your own template. Just leave one format in your zip file.
  If you use Word (.docx): submit `.docx + PDF`. The file name is: `yourname_uid_HW2_Q2_report.pdf/docx`.
  If you use LaTeX: submit `.zip (source) + PDF`. The file name is: `yourname_uid_HW2_Q2_report.zip/pdf`.

  Please delete all irrelevant content and keep only the questions numbers and your analysis in the file.

Note for Non-Python Users

If not using Python:

- Make your code CLI executable on CADE machines.
- Add a `README.md` with instructions.
- Executable should train and evaluate both baseline and decision tree.
- Print train/test accuracies (the same values you include in your report) to console.
- Accept the following flags:
  - `-t TRAIN_PATH`: path to train csv file
  - `-e EVAL_PATH` : path to test csv file
  - `-m MODEL_TYPE` (`majority baseline`, `decision tree`) : takes one of [”majority baseline”, ”decision tree”]
  - `-d DEPTH_LIMIT` : takes an integer and sets the depth-limit hyperparameter
  - `-i IG_CRITERION` (`entropy`, `gini`) : takes one of [”entropy”, ”gini”] (only needed if you’re in CS 6350)
- A separate executable should run cross-validation with flags:
  - `-c CV_PATH` pointing to your cv/ directory,
  - `-d DEPTH_LIMIT`
  - `-i IG_CRITERION` (only needed for CS 6350)

It should print the best CV accuracy and corresponding hyperparameters (the same values your include in your report) to the console.
We won't be able to assign partial credit for non-Python submissions if the code doesn't run. We also won't be able to provide debugging assistance during office hours. See the FAQ section on the class website for instructions on how to access CADE, and how to turn your code into a command-line executable.