CS 6350, DS 4350: Machine Learning Spring 2025

Homework 1

Handed out: January 14, 2025
Due date: January 28, 2025

1 Decision Trees

1. [25 points] Catherine loves going to the beach, but she keeps going on days when the conditions aren't right. She needs your help. You need to build a decision tree that will help Catherine decide if it's a good day to go to the beach. You need to make the decision based on four features described below:

   (a) **Weather** (*Sunny, Cloudy, Rainy*): Describes the weather.

   (b) **Temp** (*Hot, Warm, Cold*): Describes the temperature.

   (c) **Crowd** (*Busy, Empty*): Describes how many people are at the beach.

   (d) **Time** (*Morning, Afternoon*): Describes what time Catherine will get to the beach.

   You are given the following dataset which contains data for 10 different days. For each day, the values of the above four features are listed. The label of whether the conditions were ideal for Catherine to go to the beach is also provided.

Weather	Temp	Crowd	Time	Conditions
Cloudy	Hot	Empty	Morning	+
Sunny	Hot	Busy	Afternoon	−
Cloudy	Warm	Busy	Morning	−
Sunny	Cold	Empty	Morning	+
Rainy	Cold	Busy	Afternoon	−
Cloudy	Warm	Empty	Morning	+
Rainy	Warm	Empty	Morning	+
Cloudy	Cold	Busy	Morning	−
Sunny	Warm	Busy	Afternoon	+
Cloudy	Hot	Empty	Morning	+

Table 1: Training data for the beach prediction problem.

   (a) [5 points] How many possible functions are there to map these four features to a boolean decision? How many functions are consistent with the given training dataset?

(b) [3 points] What is the entropy of the labels in this data? When calculating entropy, the base of the logarithm should be base 2.

(c) [4 points] Compute the information gain of each feature and enter it into Table 2. Specify upto 3 decimal places.

Feature	Information Gain
Weather	
Temp	
Crowd	
Time	

Table 2: Information gain for each feature.

(d) [1 points] Which attribute will you use to construct the root of the tree using the information gain heuristic of the ID3 algorithm?

(e) [8 points] Using the root that you selected in the previous question, construct a decision tree that represents the data. You do not have to use the ID3 algorithm here, you can show any tree with the chosen root.

(f) [4 points] Suppose you are given three more examples, listed in Table 3. Use your decision tree to predict the label for each example. Also report the accuracy of the classifier that you have learned.

Weather	Temp	Crowd	Time	Conditions
Cloudy	Hot	Busy	Morning	$-$
Sunny	Cold	Busy	Afternoon	$-$
Rainy	Warm	Empty	Afternoon	$+$

Table 3: Test data for beach prediction problem

2. [10 points] Recall that in the ID3 algorithm, we want to identify the best attribute that splits the examples that are relatively pure in one label. Aside from entropy, which we saw in class and you used in the previous question, there are other methods to measure impurity.

   We will now develop a variant of the ID3 algorithm that does not use entropy. If, at some node, we stopped growing the tree and assign the most common label of the remaining examples at that node, then the impurity at that node can be measured using the Gini impurity, defined as:

$$GiniImpurity = 1 - \sum_i p_i^2$$

   where p_i is the fraction of examples that are labeled with the i^{th} label. In other words, the gini impurity measures the frequency at which any element of the dataset will be mislabelled when it is randomly labeled.

(a) [2 points] Notice that *GiniImpurity* can be thought of as a measure of impurity just like entropy. Just like we used entropy to define information gain, we can define a new version of information gain that uses *GiniImpurity* in place of entropy. Write down an expression that defines a new version of information gain that uses *GiniImpurity* in place of entropy.

(b) [6 points] Calculate the value of your newly defined information gain from the previous question for the four features in the beach dataset from 1. Use 3 significant digits. Enter the information gain into Table 4.

Feature	Information Gain (using Gini impurity)
Weather	
Temp	
Crowd	
Time	

Table 4: Information gain for each feature.

(c) [2 points] According to your results in the last question, which attribute should be the root for the decision tree? Do these two measures (entropy and Gini impurity) lead to the same tree?

2 Decision Trees on the Car Evaluation Dataset

In this assignment, we will work with the Car Evaluation dataset from the UCI Machine Learning Repository (`https://archive.ics.uci.edu/dataset/19/car+evaluation`). This dataset consists of examples with various attributes related to cars and their evaluations. Each row in the dataset has the following attributes:

- **Buying**: The buying price of the car (values: vhigh, high, med, low).
- **Maint**: The maintenance price of the car (values: vhigh, high, med, low).
- **Doors**: The number of doors (values: 2, 3, 4, 5more).
- **Persons**: The capacity of persons (values: 2, 4, more).
- **Lug_boot**: The size of the luggage boot (values: small, med, big).
- **Safety**: The safety rating of the car (values: low, med, high).

The goal is to classify cars into one of four classes: `unacc` (unacceptable), `acc` (acceptable), `good` (good), `vgood` (very good). **For this assignment, you will implement and evaluate various decision tree models on the Car Evaluation dataset.**

Programming notes. We **strongly** encourage you to use Python. If you choose not to, see the note at the end of this document for additional instructions, and skip the rest of this paragraph. For this assignment's coding questions, we will use autograder to run and grade your code on Gradescope. This means as soon as you submit, autograder runs your code and assigns you points. Remember that you are not allowed to use any machine learning libraries (e.g. `scikit-learn`, `tensorflow`, `pytorch`, etc.), 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 ¿= 3.7. If you want to use an additional package not specified, we must approve it first.

Files and Directories Provided

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

- `model.py`:
  - This file will contain most of your code changes for this homework. This is where you'll implement your baseline and decision tree 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 `MajorityBaseline` model by running the completed `train.py` script. See `README.md` for more details.

  - Focus on writing modular and reusable methods. You can add any additional functions, classes, etc. that you need, but **do not change the function signatures we've provided**. If you do, your code may not run with our autograder.

- `train.py`:
  - This file contains training and evaluation code. During the homework, 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), but you must implement the `calculate_accuracy()` method to receive full points.

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

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

- `data/` (Directory):
  - This directory contains the dataset files required for the assignment.
  - `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.

Cross-Validation

The depth of the tree is a *hyper-parameter* to the decision tree algorithm that helps reduce overfitting. By depth, we refer to the maximum path length from the root to any leaf. That is, a tree with just a single node has depth 0, a tree with a root attribute directly leading to labels in one step has depth 1 and so on. You will see later in the semester that many machine learning algorithm (SVM, logistic-regression, etc.) require choosing hyper-parameters before training commences, and this choice can make a big difference in the performance of the learners. One way to determine a good hyper-parameter values to use a technique called *cross-validation*.

As usual, we have a training set and a test set. Our goal is to discover good hyperparameters using the training set *only*. Suppose we have a hyperparameter (e.g. the depth of a decision tree) and we wish to ascertain whether it is a good choice or not. To do so, we can set aside some of the training data into a subset called the *validation* set and train on the rest of the training data. When training is finished, we can test the resulting classifier on the validation data. This allows us to get an idea of how well the particular choice of hyper-parameters does.

However, since we did not train on the whole dataset, we may have introduced a statistical bias in the classifier caused by the choice of the validation set. To correct for this, we will need to repeat this process multiple times for different choices of the validation set. That is, we need to train many classifiers with different subsets of the training data removed and average out the accuracy across these trials.

For problems with small data sets, a popular method is the leave-one-out approach. For each example, a classifier is trained on the rest of the data and the chosen example is then evaluated. The performance of the classifier is the average accuracy on all the examples. The downside to this method is for a data set with n examples we must train n different classifiers. Of course, this is not practical in general, so we will hold out subsets of the data many times instead.

Specifically, for this problem, you should implement k-fold cross-validation.

The general approach for k-fold cross-validation is the following: Suppose we want to evaluate how good a particular hyper-parameter is. We randomly split the training data into k equal sized parts. Now, we will train the model on all but one part with the chosen hyper-parameter and evaluate the trained model on the remaining part. We should repeat this k times, choosing a different part for evaluation each time. This will give us k values of accuracy. Their average cross-validation accuracy gives we an idea of how good this choice of the hyper-parameter is. To find the best value of the hyper-parameter, we will need to repeat this procedure for different choices of the hyper-parameter. Once we find the best value of the hyper-parameter, we can use the value to retrain we classifier using the entire training set.

With these points in mind, here are the coding challenges you will implement as a part of this assignment.

1. **Accuracy [10 points]**
   To ensure that your implementation of accuracy calculation is correct, you will implement the method `calculate_accuracy()` in `train.py`. This method should compute the accuracy of predictions compared to the ground truth labels. We will test your code using hidden gold labels and prediction labels.

2. **Majority Baseline Accuracy [10 points]**
   Compute the baseline accuracy of the dataset by always predicting the majority class from the training data. Clearly state the majority class and the corresponding accuracy. Implement the `MajorityBaseline` class in `model.py`, along with the `train()` and `evaluate()` functions in `train.py`. Report the accuracy on both the 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?

3. **Simple Decision Tree [15 points]**
   Implement a 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.

Implement the `DecisionTree` class in `model.py`, along with the `train()` and `evaluate()` functions in `train.py`. 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. **Decision Tree with Cross-Validation** [**15 points**]
Implement a depth limit in your decision tree implementation. Next, run cross-validation on your decision tree using the cross-validation folds we've provided and depth limit values $[1, 2, 3, 4, 5, 6]$. You will implement the depth limit in the `DecisionTree` class in `model.py`, and cross-validation in `cross_validation.py`. Report the optimal depth limit learned from cross-validation [**5 points**] and the corresponding best cross-validation average accuracy [**10 points**].

5. **Decision Tree with Best Depth from CV** [**15 points**]
Re-run your training and evaluation code in `train.py`, but using your optimal depth limit learned during cross-validation. 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).

3 CS 6350 only: Decision Trees with Collision Entropy

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

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

Repeat the steps from Section 2, but using Collision Entropy to define information gain criterion instead of the standard entropy we saw in class:

1. **Simple Decision Tree w/ 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 Section 2. See `REAMDE.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.

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 $[1, 2, 3, 4, 5, 6]$. You shouldn't need to change your code `cross_validation.py` if you've already completed Section 2. See `REAMDE.md` for instructions on how to run `cross_validation.py` with the `--ig_criterion` or `-i` flag. Report the optimal depth limit learned from cross-validation and the corresponding best cross-validation average accuracy.

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.

4 Submission Instructions

You will submit two components on Gradescope:

- **Report**: Submit your report as a pdf, containing your answers to the questions in Section 1 and your findings/results from the coding sections to the corresponding assignment in Gradescope.
- **Code**: If you're working in Python, submit your `cross_validation.py`, `model.py`, and `train.py` files to the corresponding assignment in Gradescope. If you're not using python, submit your executable files following the instructions below.

Note for non-Python Users

If you're not using Python, we ask that you make your code CLI executable from the CADE machines. Please also add a REAMDE.md to your submission with instructions on how to run your code.

Specifically, you need to have an executable file that trains and evaluates both the majority baseline classifier and your decision tree. It should print the train and test accuracies (the same values your include in your report) to the console. It should accept the following command-line flags:

- `-t TRAIN_PATH`: path to train csv file
- `-e EVAL_PATH`: path to test csv file
- `-m MODEL_TYPE`: takes one of `["majority_baseline", "decision_tree"]`
- `-d DEPTH_LIMIT`: takes an integer and sets the depth-limit hyperparameter
- `-i IG_CRITERTION`: takes one of `["entropy", "gini"]` (only needed if you're in CS 6350)

Similarly, you need to have a different executable file that runs cross-validation on your decision tree, and accepts the flags `-c CV_PATH` pointing to your `cv/` directory, `-d DEPTH_LIMIT` and `-i IG_CRITERTION` (only needed if you're in 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.