CS 334: Homework #3

Submission Instructions: Your submission should consist of two steps (detailed in Homeworks #1, #2). If *either of the two steps is late, then your assignment is late.*

1. **Decision Tree Implementation** (6+12+10+15+8+7+8=66 points): For this problem, you will implement the decision tree algorithm. You *are not allowed* to use any `scikit-learn` modules in this problem. The algorithm should work on both of the datasets from homework 2 (i.e., `simxTrain.csv, simyTrain.csv, simxTest.csv, simyTest.csv` and `space_trainx.csv, space_trainy.csv, space_testx.csv, space_testy.csv`). The template code is found in `dt.py`. The program takes two required arguments: maximum depth and minimum leaf samples and datasets as optional arguments (by default it uses the simulation dataset). You can either execute the file from the command line by running the following command `python dt.py <max depth> <min leaf samples>` or use another Python file to call the DecisionTree class methods.

   (a) (Code) Implement the `calculate_score` helper function. The arguments to the function are `y`, a 1D numpy array of shape $(n,)$ that contains the class labels (0 or 1) and the `criteria` ('gini' or 'entropy'). The function should output the gini or entropy score associated with the array. The array can be of any size and you can assume a binary classification (i.e., 0 or 1 only). Hint: make sure your function works for the pure splits!

   (b) (Code) Implement the `find_best_splitval` helper function. The arguments to the function are `xcol`, a 1d numpy array that contains the measurements for a column, `y`, a 1D numpy array of shape $(n,)$ that contains the class labels (0 or 1), the `criteria` ('gini' or 'entropy'), and the `minLeafSample`. The function should output the best split value, `v`, for the feature such that given this value v, the data will be split into $xcol \leq v$ and $xcol > v$, and also out the score resulting from using this split value.

   (c) (Code) Implement the `train` function of decision tree. The arguments to the function are `xFeat`, a numpy 2D array of size $n \times d$ where each row represents a sample, and each column a feature, and y, a numpy 1D array of size n that contains which class (i.e., 0 or 1). You can add variables to the class to store whatever information you need. No pruning is required in your implementation.

   (d) (Code) Implement the `predict` function of decision tree. This will take in a numpy 2D array (m, d), and should output a numpy 1D array $(m,)$ that represents the predicted class of each sample.

   (e) (Written) Train your decision tree model using a max depth of 3 and minimum leaf samples to be 5 on the spaceship titanic dataset. What does your decision tree learn as the set of rules? In other words, draw the decision tree. You do not need to do this using code and can use any diagram software (e.g., PowerPoint, Google slides, Lucidchart, draw.io, mermaid, etc) to translate your tree to a flow diagram. For full credit, your diagram must be *drawn using software and not hand-drawn.*

   (f) (Written) What is the training accuracy and test accuracy on the Spaceship Titanic dataset for different values of max depth and minimum number of samples in a leaf? For full credit, you must:

      i. Plot the accuracy of train and test dataset as a function of these two parameters. You can either have 2 plots, one for each variable by fixing the other variable at a

default value; or have a 3D plot to understand the relationship of accuracy to both values.

ii. Comment (i.e., 1-2 sentences) based on the plot(s) from above what you would choose as the optimal max depth and min number of samples in a leaf.

(g) (Written) What is the computational complexity of the train and predict function you implemented in terms of the training size (n), the number of features (d), and the maximum depth (p)? You must justify your answer.

2. **(6+5+7+7+3+6=34 pts) Exploring Model Assessment Strategies**

We will be using the Spaceship Titanic dataset from homework 2 to explore the different model assessment strategies covered in class. The template code is found in `modelAssess.py`. You can use any of the `scikit-learn` modules in this problem. The 3 assessment function implementations (i.e., 2b, 2c, 2d) will share the following input parameter specifications:

- `model` is a `DecisionTreeClassifier` object from `sklearn`
- `xFeat` is a numpy 2D array of shape (n, d) where each row represents a sample and each column a feature,
- `y` is a numpy 1D array of shape $(n,)$ that contains which class (0 or 1)

The output of all the functions (i.e., 2a, 2b, 2c, 2d) should return a dictionary with the following keys, and value pairs (Note only 2a will not have the time elapsed key):

- `trainAUC` is the area under the ROC curve on the training dataset
- `testAUC` is the area under the ROC curve on the test dataset.
- `trainAUPR` is the area under the PRC curve on the training dataset.
- `testAUPR` is the area under the PRC curve on the test dataset.
- `trainF1` is the F1 score on the training dataset.
- `testF1` is the F1 score on the test dataset.
- `timeElapsed` is the time the function took.

Note that any sampling involved should not be deterministic and will likely vary each time you run it!

(a) (Code) Implement the function `eval_dt_perf` which given a Decision Tree classifier model instance, will train the model using the specified training data (i.e., `xTrain`, `yTrain`) and return the area under the ROC curve (AUC), area under the PRC curve (AUPRC) and the F1 score for training and test datasets.

(b) (Code) Implement the holdout technique to assess the model. The arguments to the function `holdout` are `model, xFeat, y, testSize`. The additional parameter specifications is `testSize` which is a `float` between 0 and 1 that represents the proportion of the dataset to include in the test split. For example, 0.6 means 60% of the dataset should be used in test. Hint: You might find 2a useful for your purposes.

(c) (Code) Implement the k-fold cross-validation approach for a model, where the performance is reported from the k-different validation. The arguments to the function are `model, xFeat, y, k`. See the beginning of this problem for the definitions of the first three parameters. k is the user-specified value for k-fold cross validation. For the output performance, the numbers should reflect the average across the k-different folds.

(d) (Code) Implement Monte Carlo Cross-validation approach with s samples. The arguments to the function are `model, xFeat, y, testSize, s`. See (a) for definitions of the first four parameters. s is the number of samples for performing the validation assessment. Similar to (b) the `trainAuc`, and `testAuc` should reflect the average across the s samples. Hint: You may find it useful to re-use code from (a).

(e) (Written) Run the `model_assess.py` script from the command line and **copy the table output from the terminal into your written document** to obtain full credit. Since the results are not deterministic, you can copy the successful first run (i.e., no need to copy the output from multiple runs).

(f) (Written) Given the results from 2e, comment on how the different model selection techniques compare with one another concerning the following dimensions. For full credit, you must *ground your comments with quantitative evidence from your table.*

i. Variability in performance (AUC, AUPRC, F1) and robustness of the performance estimates to the true test performance (last line in the table)

ii. Trade-offs between larger versus smaller-sized sized sets (e.g., what is the difference between 2-fold versus 10-fold or holdout using 0.5 versus holdout using 0.9?)

iii. Computational time of the three different hold-out techniques