6 Programming (70 points)

Your goal in this assignment is to implement a binary classifier, entirely from scratch–specifically a Decision Tree learner. In addition, we will ask you to run some end-to-end experiments on two tasks (predicting whether or not a patient has heart disease / predicting the final grade for high school students) and report your results. You will write two programs: `inspection.py` (Section 6.2) and `decision_tree.py` (Section 6.3). The programs you write will be automatically graded using Gradescope.

6.1 The Tasks and Datasets

Materials Download the zip file from the course website. The zip file will have a handout folder that contains all the data that you will need in order to complete this assignment.

Starter Code The handout will contain a preexisting `decision_tree.py` file that itself contains some starter code for the assignment. While we do not require that you use the starter code in your final submission, we *heavily* recommend building upon the structure laid out in the starter code.

Datasets The handout contains three datasets. Each one contains attributes and labels and is already split into training and testing data. The first line of each `.tsv` file contains the name of each attribute, and *the class label is always the last column*.

1. **heart:** The first task is to predict whether a patient has been (or will be) diagnosed with heart disease, based on available patient information. The attributes (aka. features) are:

   (a) `sex`: The sex of the patient—1 if the patient is male, and 0 if the patient is female.

   (b) `chest_pain`: 1 if the patient has chest pain, and 0 otherwise.

   (c) `high_blood_sugar`: 1 if the patient has high blood sugar (>120 mg/dl fasting), or 0 otherwise.

   (d) `abnormal_ecg`: 1 if exercise induced angina in the patient, and 0 otherwise. Angina is a type of severe chest pain.

   (e) `flat_ST`: 1 if the patient's ST segment (a section of an ECG) was flat during exercise, or 0 if it had some slope.

   (f) `fluoroscopy`: 1 if a physician used fluoroscopy, and 0 otherwise. Fluoroscopy is an imaging technique used to see the flow of blood through the heart.

   (g) `thalassemia`: 1 if the patient is known to have thalassemia, and 0 otherwise. Thalassemia is a blood disorder that may impair the oxygen-carrying capacity of the patient's red blood cells.

   (h) `heart_disease`: 1 if the patient was diagnosed with heart disease, and 0 otherwise. This is the class label you should predict.

   The training data is in `heart_train.tsv`, and the test data in `heart_test.tsv`.

2. **education:** The second task is to predict the final grade for high school students. The attributes (covariates, predictors) are student grades on 5 multiple choice assignments *M1* through *M5*, 4 programming assignments *P1* through *P4*, and the final exam *F*. Values of 1 indicate that a student received an A, and 0 indicates that the student did not receive an A. The training data is in `education_train.tsv`, and the test data in `education_test.tsv`.

3. **small:** We also include `small_train.tsv` and `small_test.tsv`—a small, purely for demonstration version of the **heart** dataset, with *only* attributes `chest_pain` and `thalassemia`. For

this small dataset, the handout folder also contains the predictions from a reference implementation of a Decision Tree with max-depth 3 (see `small_3_train.txt`, `small_3_test.txt`, `small_3_metrics.txt`). You can check your own output against these to see if your implementation is correct.

Note: For simplicity, all attributes are discretized into just two categories (i.e. each node will have at most two descendents). This applies to all the datasets in the handout, as well as the additional datasets on which we will evaluate your Decision Tree.

6.2 Program #1: Inspecting the Data (5 points)

Write a program `inspection.py` to calculate the label entropy at the root (i.e. the entropy of the labels before any splits) and the error rate (the percent of incorrectly classified instances) of classifying using a majority vote (picking the label with the most examples). You do not need to look at the values of any of the attributes to do these calculations; knowing the labels of each example is sufficient. **Entropy should be calculated in bits using log base 2.**

Command Line Arguments The autograder runs and evaluates the output from the files generated, using the following command:

```
$ python inspection.py <input> <output>
```

Your program should accept two command line arguments: an input file and an output file. It should read the `.tsv` input file (of the format described in Section 6.1), compute the quantities above, and write them to the output file so that it contains:

```
entropy: <entropy value>
error: <error value>
```

Example For example, suppose you wanted to inspect the file `small_train.tsv` and write out the results to `small_inspect.txt`. You would run the following command:

```
$ python inspection.py small_train.tsv small_inspect.txt
```

Afterwards, your output file `small_inspect.txt` should contain the following:

```
entropy: 1.000000
error: 0.500000
```

Our autograder will run your program on several input datasets to check that it correctly computes entropy and error, and will take minor differences due to rounding into account. You do not need to round your reported numbers! The autograder will automatically incorporate the right tolerance for float comparisons.

For your own records, run your program on each of the datasets provided in the handout—this error rate for a *majority vote* classifier is a baseline over which we would (ideally) like to improve.

6.3 Program #2: Decision Tree Learner (65 points)

In `decision_tree.py`, implement a Decision Tree learner. This file should learn a decision tree with a specified maximum depth, print the decision tree in a specified format, predict the labels of the training and testing examples, and calculate training and testing errors.

Your implementation must satisfy the following requirements:

- Use mutual information to determine which attribute to split on.
- Be sure you're correctly weighting your calculation of mutual information. For a split on attribute X, $I(Y;X) = H(Y) - H(Y|X) = H(Y) - P(X=0)H(Y|X=0) - P(X=1)H(Y|X=1)$.
- As a stopping rule, only split on an attribute if the mutual information is > 0.
- Do not grow the tree beyond a max-depth specified on the command line. For example, for a maximum depth of 3, split a node only if the mutual information is > 0 and the current level of the node is < 3.
- Use a majority vote of the labels at each leaf to make classification decisions. If the vote is tied, choose the label that is higher (i.e. 1 should be chosen before 0)
- It is possible for different columns to have equal values for mutual information. In this case, you should split on the ***first column to break ties*** (e.g. if column 0 and column 4 have the same mutual information, use column 0).
- Do not hard-code any aspects of the datasets into your code. We may autograde your programs on hidden datasets that include different attributes and output labels.

Careful planning will help you to correctly and concisely implement your Decision Tree learner. Here are a few *hints* to get you started:

- Write helper functions to calculate entropy and mutual information.
- It is best to think of a Decision Tree as a collection of nodes, where nodes are either leaf nodes (where final decisions are made) or interior nodes (where we split on attributes). It is helpful to design a function to train a single node (i.e. a depth-0 tree), and then recursively call that function to create sub-trees.
- In the recursion, keep track of the depth of the current tree so you can stop growing the tree beyond the max-depth.
- Implement a function that takes a learned decision tree and data as inputs, and generates predicted labels. You can write a separate function to calculate the error of the predicted labels with respect to the given (ground-truth) labels.
- Be sure to correctly handle the case where the specified maximum depth is greater than the total number of attributes.
- Be sure to handle the case where max-depth is zero (i.e. a majority vote classifier).
- Look under the FAQ post on Piazza for more useful clarifications about the assignment.

6.4 Command Line Arguments

The autograder runs and evaluates the output from the files generated, using the following command:

```
$ python decision_tree.py [args...]
```

Where above `[args...]` is a placeholder for six command-line arguments: `<train input> <test input> <max depth> <train out> <test out> <metrics out>`. These arguments are described in detail below:

1. `<train input>`: path to the training input `.tsv` file (see Section 6.1)
2. `<test input>`: path to the test input `.tsv` file (see Section 6.1)
3. `<max depth>`: maximum depth to which the tree should be built
4. `<train out>`: path of output `.txt` file to which the predictions on the *training* data should be written (see Section 6.5)
5. `<test out>`: path of output `.txt` file to which the predictions on the *test* data should be written (see Section 6.5)
6. `<metrics out>`: path of the output `.txt` file to which metrics such as train and test error should be written (see Section 6.6)

As an example, the following command line would run your program on the heart dataset and learn a tree with a max-depth of 2. The train predictions would be written to `heart_2_train.txt`, the test predictions to `heart_2_test.txt`, and the metrics to `heart_2_metrics.txt`.

```
$ python decision_tree.py heart_train.tsv heart_test.tsv 2 \
        heart_2_train.txt heart_2_test.txt heart_2_metrics.txt
```

The following example would run the same learning setup except with a max-depth of 3, and conveniently writing to analogously named output files, so you can can compare the two runs.

```
$ python decision_tree.py heart_train.tsv heart_test.tsv 3 \
        heart_3_train.txt heart_3_test.txt heart_3_metrics.txt
```

6.5 Output: Labels Files

Your program should write two output `.txt` files containing the predictions of your model on training data (`<train out>`) and test data (`<test out>`). Each should contain the predicted labels for each example printed on a new line. Use '`\n`' to create a new line.

Your labels should exactly match those of a reference decision tree implementation—this will be checked by the autograder by running your program and evaluating your output file against the reference solution.

The first few lines of an example output file is given below for the small dataset:

```
1
0
1
1
0
0
...
```

6.6 Output: Metrics File

Generate another file where you should report the training error and testing error. This file should be written to the path specified by the command line argument `<metrics out>`. Your reported numbers should be within 0.0001 of the reference solution. You do not need to round your reported numbers! The autograder will automatically incorporate the right tolerance for float comparisons. The file should be formatted as follows:

```
error(train): 0.214286
error(test): 0.285714
```

The values above correspond to the results from training a tree of depth 3 on `small_train.tsv` and testing on `small_test.tsv`. (Note that there is one space between the colon and value.)

6.7 Output: Printing the Tree

Finally, you should write a function to pretty-print your learned decision tree. **Your function should print your tree only *after* you are done generating the fully-trained tree.** Each row should correspond to a node in the tree. They should be printed using a *pre-order depth-first-search* traversal (but you may print left-to-right or right-to-left, i.e. your answer does not need to have exactly the same order as the reference below). Print the attribute of the node's parent and the attribute value corresponding to the node. Also include the sufficient statistics (i.e. count of positive / negative examples) for the data passed to that node. The row for the root should include *only* those sufficient statistics. A node at depth d, should be prefixed by d copies of the string '| '.

Below, we have provided the recommended format for printing the tree. You can print it directly rather than to a file. **This functionality of your program will NOT be autograded**.

```
$ python decision_tree.py small_train.tsv small_test.tsv 2 \
small_2_train.txt small_2_test.txt small_2_metrics.txt

[14 0/14 1]
| chest_pain = 0: [4 0/12 1]
| | thalassemia = 0: [3 0/4 1]
| | thalassemia = 1: [1 0/8 1]
| chest_pain = 1: [10 0/2 1]
| | thalassemia = 0: [7 0/0 1]
| | thalassemia = 1: [3 0/2 1]
```

However, you should be careful that the tree might not be full. For example, with a different subset of the small dataset, there may be no nodes under `chest_pain = 0` if all labels are the same.

The following pretty-print shows the education dataset with max-depth 3. Use this example to check your code before submitting your pretty-print of the heart dataset (asked in question 5 of the Empirical questions).

```
$ python decision_tree.py education_train.tsv education_test.tsv 3 \
edu_3_train.txt edu_3_test.txt edu_3_metrics.txt

[65 0/135 1]
| F = 0: [42 0/16 1]
| | M2 = 0: [27 0/3 1]
| | | M4 = 0: [22 0/0 1]
| | | M4 = 1: [5 0/3 1]
| | M2 = 1: [15 0/13 1]
| | | M4 = 0: [14 0/7 1]
| | | M4 = 1: [1 0/6 1]
| F = 1: [23 0/119 1]
| | M4 = 0: [21 0/63 1]
| | | M2 = 0: [18 0/26 1]
```

```
| | | M2 = 1: [3 0/37 1]
| | M4 = 1: [2 0/56 1]
| | | P1 = 0: [2 0/15 1]
| | | P1 = 1: [0 0/41 1]
```

The numbers in brackets give the number of positive and negative labels from the training data in that part of the tree.

At this point, you should be able to go back and answer questions 1-5 in the "Empirical Questions" section of this handout. Write your solutions in the template provided.

6.8 Submission Instructions

Programming Please ensure you have completed the following files for submission.

```
inspection.py
decision_tree.py
```

When submitting your solution, make sure to select and upload both files. Ensure the files have the exact same spelling and letter casing as above. You can either directly zip the two files (by selecting the two files and compressing them – do not compress the folder containing the files) or directly drag them to Gradescope for submission.

Written Questions Make sure you have completed all questions from Written component (including the collaboration policy questions) in the template provided. When you have done so, please submit your document in **PDF format** to the corresponding assignment slot on Gradescope.