HOMEWORK 2: DECISION TREES [*]

10-301 / 10-601 INTRODUCTION TO MACHINE LEARNING (SPRING 2025)
http://www.cs.cmu.edu/~mgormley/courses/10601/

OUT: Wednesday, January 22
DUE: Monday, February 3
TAs: Mihir, Rohini, Joaquin, Zhifei, Neural the Narwhal

Summary It's time to build your first end-to-end learning system! In this assignment, you will build a Decision Tree classifier and apply it to several binary classification problems. This assignment consists of several parts: In the Written component, you will work through some Information Theory basics in order to "learn" a Decision Tree on paper, and also work through some pseudocode that will help you algorithmically think through the programming assignment. Then in the Programming component, you will implement Decision Tree learning, prediction, and evaluation. Using that implementation, you will answer the empirical questions found at the end of the Written component.

START HERE: Instructions

- **Collaboration Policy**: Please read the collaboration policy here: http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html

- **Late Submission Policy: For this homework, you will only have 2 late days instead of the usual 3.** This allows us to provide feedback before the exam. See the late submission policy here: http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html

- **Submitting your work:** You will use Gradescope to submit answers to all questions and code. Please follow instructions at the end of this PDF to correctly submit all your code to Gradescope.

    - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, please use the provided template. Submissions can be handwritten onto the template, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. If your scanned submission misaligns the template, there will be a 2% penalty. Alternatively, submissions can be written in LaTeX. Each derivation/proof should be completed in the boxes provided. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader.

---

[*]Compiled on Wednesday 22nd January, 2025 at 15:28

Instructions for Specific Problem Types

For "Select One" questions, please fill in the appropriate bubble completely:

Select One: Who taught this course?

- ● Matt Gormley
- ○ Marie Curie
- ○ Noam Chomsky

If you need to change your answer, you may cross out the previous answer and bubble in the new answer:

Select One: Who taught this course?

- ● Henry Chai
- ○ Marie Curie
- ⊗ Noam Chomsky

For "Select all that apply" questions, please fill in all appropriate squares completely:

Select all that apply: Which are instructors for this course?

- ■ Matt Gormley
- ■ Henry Chai
- □ Noam Chomsky
- □ I don't know

Again, if you need to change your answer, you may cross out the previous answer(s) and bubble in the new answer(s):

Select all that apply: Which are the instructors for this course?

- ■ Matt Gormley
- ■ Henry Chai
- ⊠ Noam Chomsky
- ⊠ I don't know

For questions where you must fill in a blank, please make sure your final answer is fully included in the given space. You may cross out answers or parts of answers, but the final answer must still be within the given space.

Fill in the blank: What is the course number?

| 10-601 | 10-6̶301 |

Written Questions (37 points)

1 LaTeX Point and Template Alignment (1 points)

1. (1 point) **Select one:** Did you use LaTeX for the entire written portion of this homework?

   ○ Yes

   ○ No

2. (0 points) **Select one:** I have ensured that my final submission is aligned with the original template given to me in the handout file and that I haven't deleted or resized any items or made any other modifications which will result in a misaligned template. I understand that incorrectly responding yes to this question will result in a penalty equivalent to 2% of the points on this assignment.
   Note: Failing to answer this question will not exempt you from the 2% misalignment penalty.

   ○ Yes

2 Machine Learning as Function Approximation (8 points)

Imagine you are a young scientist on a mission to help space explorers navigate the galaxy! You've been tasked with predicting the movements of three mysterious celestial objects (let's call them Neural, Gradyant, and KeNNy) based on their positions and velocities at any given time. However, the actual equations that govern their motion are unknown to you because they exist in a chaotic system. A chaotic system is classically estimated using simulation or with a machine learning model. You decide to use a machine learning model as a function approximator to predict the celestial bodies' future positions.

The goal of your mission is to design a machine learning model that learns to approximate the motion of these celestial bodies from examples of past movements.

Note: Showing your work in these questions is optional, but it is recommended to help us understand where any misconceptions may occur. Only your numerical answer in the left box will be graded.

1. (1 point) Imagine you observe the celestial bodies' current positions and velocities in 3D space at a specific time. How many dimensions would this input space $\mathcal{X}$ have?

Dimensions	Work

2. (1 point) What is the set $\mathcal{X}$ of all possible inputs to your model?

Set	Work

3. (1 point) Note that the goal of the model is to predict the future positions of our celestial bodies after one time step. How many dimensions does this output space $\mathcal{Y}$ have?

Dimensions	Work

4. (1 point) What is the set $\mathcal{Y}$ of all possible outputs to your model?

Set	Work

5. (2 points) Broadly, what is the unknown target function c^*? Approximately how many celestial bodies would you need to know about in order to define c^* or a very good approximation to it?

Explanation

6. (2 points) Now, you decide to collect data on the past movements of Neural, Gradyant, and KeNNy. For each data point you record

   - Their positions and velocities at time t

   - Their positions at time $t + 1$

   If you trained a machine learning model on this dataset, what specific role does the model play in this problem?

   > **Explanation**
   >
   >
   >
   >
   >
   >
   >

3   Decision Tree Calculations (11 points)

First, let's think a little bit about decision trees. The following dataset D consists of 8 examples, each with 3 attributes, (A, B, C), and a label, Y.

A	B	C	Y
1	2	0	1
0	1	0	0
0	0	1	0
0	2	0	1
1	1	0	1
1	0	1	0
1	2	1	0
1	1	0	1

Use the data above to answer the following questions.

A few important notes:

- *All calculations should be done without rounding!* After you have finished all of your calculations, write your rounded solutions in the boxes below.

- Unless otherwise noted, numeric solutions should include 4 digits of precision (e.g. 0.1234).

- Note that, throughout this homework, we will use the convention that the leaves of the trees do not count as nodes, and as such are not included in calculations of depth and number of splits. (For example, a tree which classifies the data based on the value of a single attribute will have depth 1, and contain 1 split.)

- Note that the dataset contains duplicate rows; treat each of these as their own example, do not remove duplicate rows.

Note: Showing your work in these questions is optional, but it is recommended to help us understand where any misconceptions may occur. Only your numerical answer in the left box will be graded.

1. (1 point) What is the entropy of Y in bits, $H(Y)$? In this and subsequent questions, when we request the units in *bits*, this simply means that you need to use $\log$ base 2 in your calculations.[1] (Please include one number rounded to the fourth decimal place, e.g. 0.1234)

$H(Y)$	Work

2. (1 point) What is the mutual information of Y and A in bits, $I(Y;A)$? (Please include one number rounded to the fourth decimal place, e.g. 0.1234)

$I(Y;A)$	Work

3. (1 point) What is the mutual information of Y and B in bits, $I(Y;B)$? (Please include one number rounded to the fourth decimal place, e.g. 0.1234)

$I(Y;B)$	Work

---

[1] If instead you used $\log$ base e, the units would be *nats*; $\log$ base 10 gives *bats*.

4. (1 point) What is the mutual information of Y and C in bits, $I(Y;C)$? (Please include one number rounded to the fourth decimal place, e.g. 0.1234)

$I(Y;C)$	Work

5. (1 point) **Select one:** Consider the dataset given above. Which attribute (A, B, or C) would a decision tree algorithm pick first to branch on, if its splitting criterion is mutual information?

   ○ A

   ○ B

   ○ C

6. (1 point) **Select one:** Consider the dataset given above. After making the first split, which attribute would the algorithm pick to branch on next, if the splitting criterion is mutual information? (*Hint:* Notice that this question correctly presupposes that there is *exactly one* second attribute.)
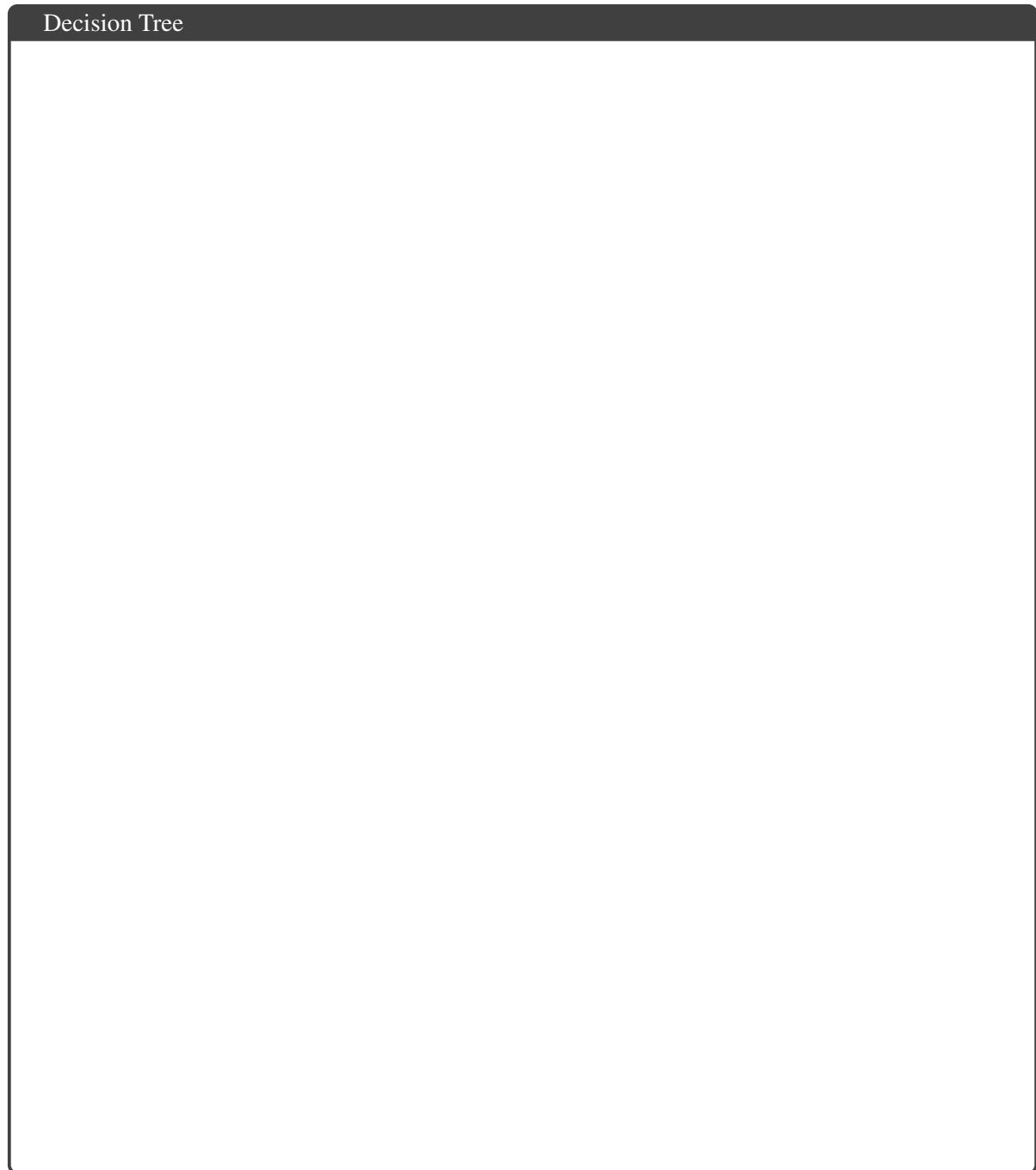
   ○ A

   ○ B

   ○ C

7. (1 point) If the same algorithm continues until the tree perfectly classifies the data, what would the depth of the tree be?

Depth

8. (4 points) Draw your completed Decision Tree. Label the non-leaf nodes with which attribute the tree will split on (e.g. B), the edges with the value of the attribute (e.g. 1 or 0), and the leaf nodes with the classification decision (e.g. $Y = 0$). You should include an image file below using the provided, commented out code in LaTeX, switching out *DecTree.png* to your file name as needed. The image may be hand-drawn.

Decision Tree

4   Pseudocode (6 points)

1. In the programming assignment, you will need to implement three main tasks: training a decision tree on an arbitrary training set, predicting new values with a trained tree given an arbitrary input dataset, and evaluating your predictions against an arbitrary dataset's true labels. For this problem, we will focus on thinking through the algorithm for the *second* task.

   Below, you will write pseudocode for the function `predict(node, example)`, which predicts the label of an `example` given a `node` of type `Node` representing the root of a *trained* tree. You must approach this problem recursively and use the `Node` class we have given to you.

```
class Node:
    def __init__(self, attr, v):
        self.attribute = attr
        self.left = None
        self.right = None
        self.vote = v

# (a) the left and right children of a node are denoted as
#     node.left and node.right respectively, each is of type Node
# (b) the attribute for a node is denoted as node.attribute and has
#     type str
# (c) if the node is a leaf, then node.vote of type str holds the
#     prediction from the majority vote; if node is an internal
#     node, then node.vote has value None
# (d) assume all attributes have values 0 and 1 only; further
#     assume that the left child corresponds to an attribute value
#     of 1, and the right child to a value of 0

def predict(node, example):
    # example is a dictionary which holds the attributes and the
    # values of the attribute (ex. example['X'] = 0)

    if ____Part (a)____:
        return ____Part (b)____
    else:
            ____Part (c)____
            _____
            _____
            _____
```

   (a) (2 points) Write the conditional statement of the base case of `predict(node, example)`.

   > Your Answer

(b) (1 point) Write the return statement of the base case of `predict(node, example)`.

> Your Answer

(c) (3 points) Write the recursive step of `predict(node, example)`. Limit your answer to 10 lines.
NOTE: This may be a multi line solution.

> Your Answer

5 Empirical Questions (11 points)

The following questions should be completed as you work through the programming portion of this assignment.

1. (4 points) Train and test your decision tree on the heart dataset and the purchase dataset with four different values of max-depth specified in the table. Report your findings in the HW2 solutions template provided. A Decision Tree with max-depth 0 is simply a *majority vote classifier*; a Decision Tree with max-depth 1 is called a *decision stump*. (Please round each number to the fourth decimal place, e.g. 0.1234)

Dataset	Max-Depth	Train Error	Test Error
heart	0		
heart	1		
heart	2		
heart	4		
purchase	0		
purchase	1		
purchase	2		
purchase	3		

2. (3 points) For the heart disease (`heart`) dataset, create a *computer-generated* plot showing error on the y-axis against depth of the tree on the x-axis. On a single plot, include *both* training error and testing error, clearly labeling which is which. That is, for each possible value of max-depth $(0, 1, 2, \ldots, m)$, where m = number of attributes in the dataset, you should train a decision tree and report train/test error of the model's predictions. You should include an image file below using the provided, commented out code in LATEX, switching out `heart.png` to your file name as needed.

Plot

3. (2 points) **Select one:** Suppose your research advisor asks you to run some model selection experiments and then report your results. You select the Decision Tree model's max-depth to be the one with lowest test error in metrics.txt and then report that model's test error as the performance of our classifier on held out test data. Is this a good experimental setup?

○ Yes, because we are using the test set in order to choose the best model.

○ No, because we should be using the *training* set to optimize the max-depth, not the test set.

○ Yes, because we are not using the training set to tune hyperparameters.

○ No, because we should be using a *validation* set to optimize the max-depth, not the test set.

4. (2 points) **Select one:** In this assignment, we used max-depth as our stopping criterion, and as a mechanism to prevent overfitting. Alternatively, we could stop splitting a node whenever the mutual information for the best attribute is lower than a threshold value. This threshold would be another hyperparameter. Theoretically, how would increasing this threshold value affect the number of nodes and depth of the learned trees?

○ The higher this threshold value is, the *smaller* depth and the *fewer* nodes the decision tree contains.

○ The higher this threshold value is, the *smaller* depth and the *more* nodes the decision tree contains.

○ The higher this threshold value is, the *higher* depth and the *fewer* nodes the decision tree contains.

○ The higher this threshold value is, the *higher* depth and the *more* nodes the decision tree contains.

○ The depth and number of nodes in the decision tree will not vary as the threshold value increases.

6   Collaboration Questions

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found here.

1. Did you receive any help whatsoever from anyone in solving this assignment? If so, include full details.

2. Did you give any help whatsoever to anyone in solving this assignment? If so, include full details.

3. Did you find or come across code that implements any part of this assignment? If so, include full details.

> Your Answer

7 Programming (79 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 whether a customer will make a purchase on an online store) and report your results. You will write two programs: `inspection.py` (Section 7.2) and `decision_tree.py` (Section 7.3). The programs you write will be automatically graded using Gradescope.

7.1 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), and 0 otherwise.

    (d) `abnormal_ecg`: 1 if the patient exhibits abnormal electrical heart activity by ECG, and 0 otherwise.

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

    (f) `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.

    (g) `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.

    (h) `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.

    (i) `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. **purchase:** The second task is to predict whether a customer will make a purchase based on their behavior on an e-commerce platform. The attributes are:

    (a) `Visited_Product_Page`: 1 if the customer visited a product page, and 0 otherwise.

    (b) `Added_to_Cart`: 1 if the customer added an item to the cart, and 0 otherwise.

(c) `Visited_Deals_Page`: 1 if the customer visited the deals or discounts page, and 0 otherwise.

(d) `Clicked_Ad`: 1 if the customer clicked on an advertisement, and 0 otherwise.

(e) `Viewed_Similar_Items`: 1 if the customer viewed similar items to what they were browsing, and 0 otherwise.

(f) `Abandoned_Cart`: 1 if the customer abandoned their shopping cart, and 0 otherwise.

(g) `Premium_User`: 1 if the customer is subscribed to the platform's premium services, and 0 otherwise.

(h) `Used_Search_Bar`: 1 if the customer used the search bar during their session, and 0 otherwise.

(i) `Clicked_Email_Link`: 1 if the customer clicked on a link in a marketing email, and 0 otherwise.

(j) `Viewed_Reviews`: 1 if the customer viewed product reviews, and 0 otherwise.

(k) `Purchase`: 1 if the customer made a purchase, and 0 otherwise.

The training data is in `purchase_train.tsv`, and the test data in `purchase_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.labels`, `small_3_test.labels`, `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.

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

Create and 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 7.1), 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.

7.3   Program #2: Decision Tree Learner (74 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.

7.3.1   Getting Started

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.

> Look out for the HW2 debugging checklist on Piazza!

7.3.2 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 seven command-line arguments: `<train input> <test input> <max depth> <train out> <test out> <metrics out> <print out>`. These arguments are described in detail below:

1. `<train input>`: path to the training input `.tsv` file (see Section 7.1)

2. `<test input>`: path to the test input `.tsv` file (see Section 7.1)

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 7.3.3)

5. `<test out>`: path of output `.txt` file to which the predictions on the *test* data should be written (see Section 7.3.3)

6. `<metrics out>`: path of the output `.txt` file to which metrics such as train and test error should be written (see Section 7.3.4)

7. `<print out>`: path of the output `.txt` file to which the printed tree should be written (see Section 7.3.5)

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`, the metrics to `heart_2_metrics.txt`, and the printed tree to `heart_2_print.txt`.

```
$ python decision_tree.py heart_train.tsv heart_test.tsv 2 \
        heart_2_train.txt heart_2_test.txt heart_2_metrics.txt
        \ heart_2_print.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
        \ heart_3_print.txt
```

7.3.3 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
...
```

7.3.4 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.)

7.3.5 Output: Printing the Tree

You must write a function to pretty-print your learned decision tree. (This will also help you debug!)
Generate another file where you report the printed tree. This file should be written to the path specified by
the command line argument `<print out>`. **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. **The branch
that corresponds to 0 should be printed before the branch that corresponds to 1.**

A simple breakdown of each row within the pretty-print tree is as follows:

1. d copies of the string '|' corresponding to the depth d of the current node

2. `<Feature most recently split on (if any)> = <0 or 1>:`

3. `[<#0s> 0/<#1s> 1]` (sufficient statistics of data at current node)

Here is a reference output for printing the decision tree.

```
$ python decision_tree.py small_train.tsv small_test.tsv 2 \
small_2_train.txt small_2_test.txt small_2_metrics.txt
\ small_2_print.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]
```

Note: Your format does not need to exactly match this output to receive full points as we give flexibility with spacing. However, **each node of the tree must be printed in a separate line**. Some examples of valid printing format include:

```
||thalassemia=0:[3 0/4 1]
||thalassemia=0:[3 0 / 4 1]
|  |  thalassemia = 0:[ 3 0/4 1 ]
...
```

You should also 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 purchase dataset with max-depth 4. Use this example to check your code before submitting to the autograder.

```
$ python decision_tree.py purchase_train.tsv purchase_test.tsv 4 \
purchase_4_train.txt purchase_4_test.txt purchase_4_metrics.txt \
 purchase_4_print.txt

[632 0/368 1]
| Added_to_Cart = 0: [433 0/83 1]
| | Clicked_Email_Link = 0: [371 0/0 1]
| | Clicked_Email_Link = 1: [62 0/83 1]
| | | Used_Search_Bar = 0: [62 0/0 1]
| | | Used_Search_Bar = 1: [0 0/83 1]
| Added_to_Cart = 1: [199 0/285 1]
| | Viewed_Reviews = 0: [199 0/48 1]
| | | Clicked_Email_Link = 0: [170 0/0 1]
| | | Clicked_Email_Link = 1: [29 0/48 1]
| | | | Used_Search_Bar = 0: [29 0/0 1]
| | | | Used_Search_Bar = 1: [0 0/48 1]
| | Viewed_Reviews = 1: [0 0/237 1]
```

The numbers in brackets give the number of positive and negative labels from the training data in that part of the tree. For the example above, the first line indicates that there at 632 zero-labels and 368 one-labels

in the entire tree, and the second line indcates that there are 433 zero-labels and 83 one-labels in the branch where `Added_To_Cart = 0`.

> At this point, you should be able to go back and answer questions 1-4 in the "Empirical Questions" section of this handout. Write your solutions in the template provided.

7.4   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. **Any other files will be deleted.** 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. **Do NOT submit a Python notebook file (*.ipynb Jupyter notebook or Google Colab notebook file).**

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.