HOMEWORK 2: DECISION TREES

10-601 Introduction to Machine Learning (Fall 2020)
https://www.cs.cmu.edu/~10601/
DUE: Monday, September 21, 2020 11:59 PM

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 Written component, you will work through some Information Theory basics in order to "learn" a Decision Tree on paper. Then in Programming component, you will implement Decision Tree learning, prediction, and evaluation. Using that implementation, you the empirical questions found at the end of the Written component.

START HERE: Instructions

- **Collaboration Policy**: Please read the collaboration policy here: https://www.cs.cmu.edu/~10601

- **Late Submission Policy:** See the late submission policy here: https://www.cs.cmu.edu/~10601

- **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, we will be using Gradescope (https://gradescope.com/). Please use the provided template. Submissions must be written in LaTeX. Regrade requests can be made, however this gives the staff the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted. Each derivation/proof should be completed in the boxes provided. For short answer questions you **should not** include your work in your solution. If you include your work in your solutions, your assignment may not be graded correctly by our AI assisted grader.

  - **Programming:** You will submit your code for programming questions on the homework to Gradescope (https://gradescope.com). After uploading your code, our grading scripts will autograde your assignment by running your program on a virtual machine (VM). When you are developing, check that the version number of the programming language environment (e.g. Python 3.6.9, OpenJDK 11.0.5, g++ 7.4.0) and versions of permitted libraries (e.g. `numpy` 1.17.0 and `scipy` 1.4.1) match those used on Gradescope. You have unlimited Gradescope programming submissions. However, we recommend debugging your implementation on your local machine (or the linux servers) and making sure your code is running correctly first before submitting you code to Gradescope.

- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on Piazza.

Written Questions (30 points)

1. Warm-Up

First, let's think a little bit about decision trees. The following dataset D consists of 7 examples, each with 3 attributes, (A, B, C), and a label, Y.

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

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.
> - 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: 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.

(a) (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)

$H(Y)$
0.9852

Work

$$Entropy = -\Sigma_{y \in Y} P(Y = y)log_2 P(Y = y)$$
$$= -[\frac{4}{7}log_2\frac{4}{7} + \frac{3}{7}log_2\frac{3}{7}]$$

---

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

(b) (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
0.5216	$H(Y) - H(Y|A) = 0.9852 - [\frac{3}{7}H(Y|A = 0) + \frac{4}{7}H(Y|A = 1)]$ $H(Y|A = 0) = -[1log_2 1 + 0log_2 0]$ $H(Y|A = 1) = -[\frac{1}{4}log_2\frac{1}{4} + \frac{1}{4}log_2\frac{1}{4}]$

(c) (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
0.0202	$H(Y) - H(Y|B) = 0.9852 - [\frac{3}{7}H(Y|B = 0) + \frac{4}{7}H(Y|B = 1)]$ $H(Y|B = 0) = -[\frac{2}{3}log_2\frac{2}{3} + \frac{1}{3}log_2\frac{1}{3}]$ $H(Y|B = 1) = -[\frac{2}{4}log_2\frac{2}{4} + \frac{2}{4}log_2\frac{2}{4}]$

(d) (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
0.1981	$H(Y) - H(Y|C) = 0.9852 - [\frac{3}{7}H(Y|C = 0) + \frac{1}{7}H(Y|C = 1) + \frac{3}{7}H(Y|C = 2)]$ $H(Y|C = 0) = -[\frac{2}{3}log_2\frac{2}{3} + \frac{1}{3}log_2\frac{1}{3}]$ $H(Y|C = 1) = -[1log_2 1 + 0log_2 0]$ $H(Y|C = 2) = -[\frac{1}{3}log_2\frac{1}{3} + \frac{2}{3}log_2\frac{2}{3}]$

(e) (1 point) 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?

Select one:

⬤ A

◯ B

◯ C

(f) (1 point) Consider the dataset given above. Which is the second attribute you would pick to branch on, if its splitting criterion is mutual information? (*Hint:* Notice that this question correctly presupposes that there is *exactly one* second attribute.)

Select one:

◯ A

◯ B

⬤ C

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

Depth
2

(h) (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.
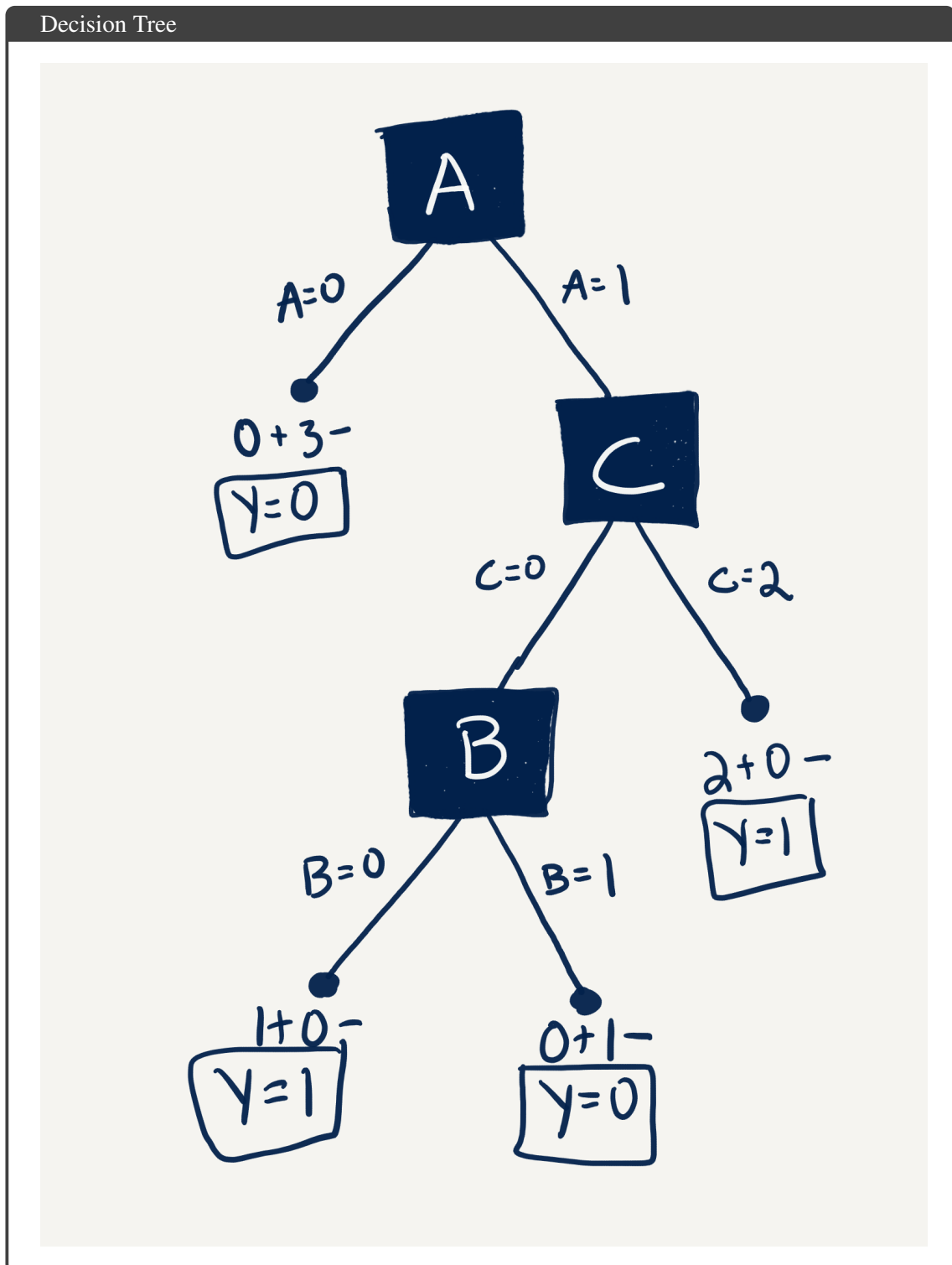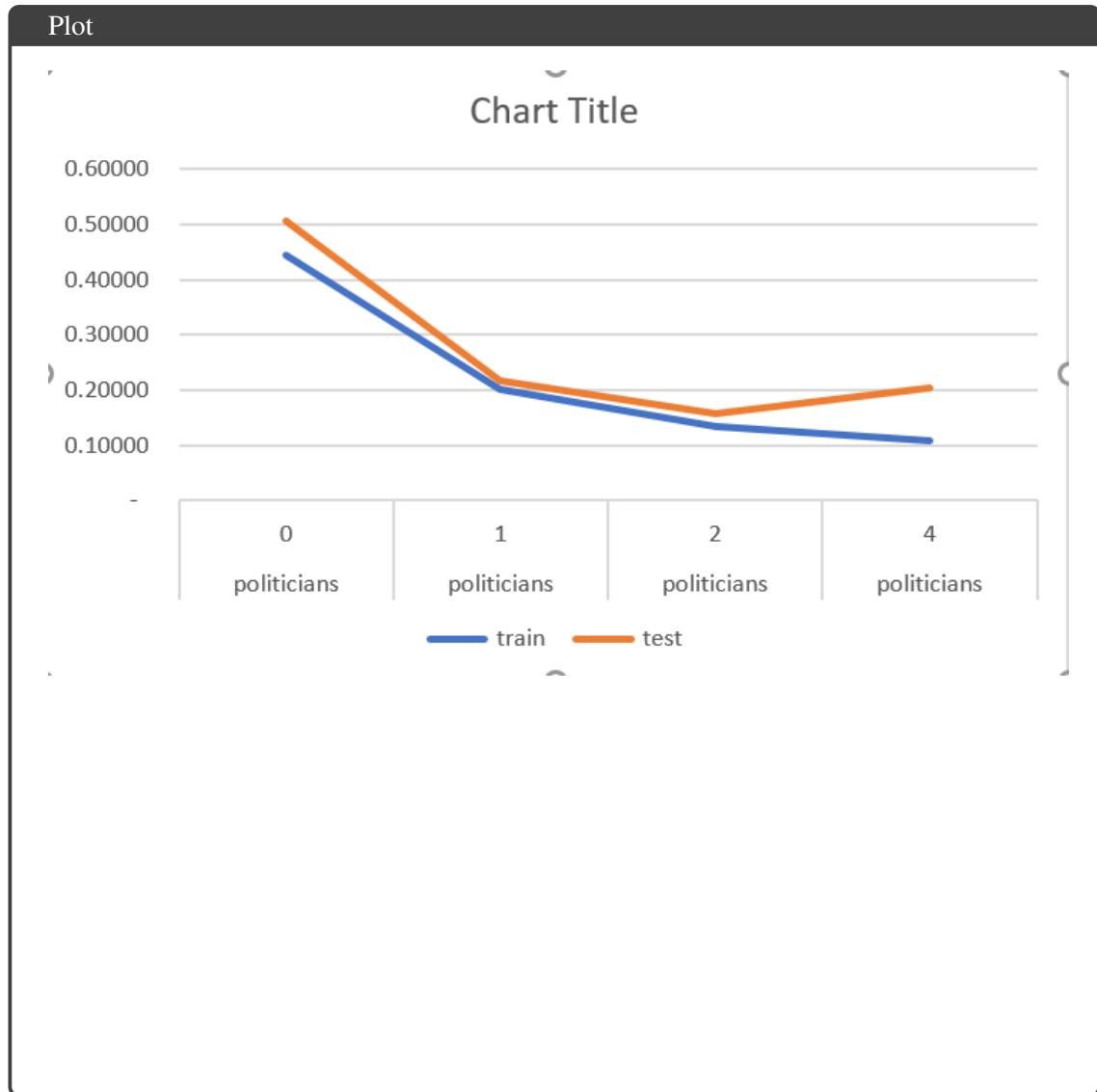
Decision Tree

2. Empirical Questions

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

(a) (3 points) Train and test your decision tree on the politician dataset and the education dataset with four different values of max-depth, $\{0, 1, 2, 4\}$. 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*. If desired, you could even check that your answers for these two simple cases are correct using your favorite spreadsheet application (e.g. Excel, Google Sheets). (Please round each number to the fourth decimal place, e.g. 0.1234)

Dataset	Max-Depth	Train Error	Test Error
politician	0	0.44295	0.50602
politician	1	0.20134	0.21687
politician	2	0.13423	0.15663
politician	4	0.10738	0.20482
education	0	0.32500	0.31000
education	1	0.19500	0.23000
education	2	0.19500	0.23000
education	4	0.13000	0.15500

(b) (3 points) For the politicians dataset, create a computer-generated plot showing error on the y-axis against depth of the tree on the x-axis. Plot *both* training error and testing error, clearly labeling which is which. That is, for each possible value of max-depth $(0, 1, 2, \ldots,$ up to the 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 *politician.png* to your file name as needed.

(c) (3 points) 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? If so, why? If not, why not?

> **Answer**
>
> No, this is not a good experimental setup because it's using the test error as a way to choose the model we use. If we do this, we're just using our test error as almost an extension of our training dataset, but that's not going to help us actually choose the best model for real new data.

(d) (3 points) 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?

> **Answer**
>
> In our code currently we used the threshold of 0. Increasing this threshold means that less and less splits will happen, since the only way we decide to split is if it matches that threshold, so it would lead to fewer nodes and a smaller depth.

(e) (3 points) From section 1.2 question 4, how would you set-up model training to choose the threshold value?

> **Answer**
>
> You would have to use some kind of holdout sample or validation data - this could include cross validation or leave one out validation. This way, you could still keep your test set to truly test your model rather than choose parameters.

(f) (4 points) Print (do not handwrite!) the decision tree which is produced by your algorithm for the politician data with max depth 3. Instructions on how to print the tree could be found in section 3.4.

> **Output**
>
> ```
> [15 democrat/13 republican]
> ||Anti_satellite_test_ban = n: [2 democrat/12 republican]
> |||Export_south_africa = n: [5 republican]
> |||Export_south_africa = y: [2 democrat/7 republican]
> ||Anti_satellite_test_ban = y: [13 democrat/1 republican]
> |||Export_south_africa = n: [1 republican]
> |||Export_south_africa = y: [13 democrat]
> ```

(g) 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? Is so, include full details. 2. Did you give any help whatsoever to anyone in solving this assignment? Is so, include full details. 3. Did you find or come across code that implements any part of this assignment ? If so, include full details.

Answer

Sinduja Sriskanda and I talked over some high level concepts with each other, such as object oriented programming and recursion.

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 the party of a politician / predicting final grade for high school students) and report your results. You will write two programs: `inspection.{py|java|cpp}` (Section 2) and `decisionTree.{py|java|cpp}` (Section 3). The programs you write will be automatically graded using the Gradescope system. You may write your programs in **Python**, **Java**, or **C++**. However, you should use the same language for all parts below. In general, **Python** is recommended.

1 The Tasks and Datasets

Materials Download the zip file from Piazza ("Resources" tab). The zip file will have a handout folder that contains all the data that you will need in order to complete this assignment.

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 is always the last column*.

1. **politician:** The first task is to predict whether a US politician is a member of the Democrat or Republican party, based on their past voting history. Attributes (aka. features) are short descriptions of bills that were voted on, such as *Aid_to_nicaraguan_contras* or *Duty_free_exports*. Values are given as *'y'* for yes votes and *'n'* for no votes. The training data is in `politicians_train.tsv`, and the test data in `politicians_test.tsv`.

2. **education:** The second task is to predict the final *grade* (A, not A) 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*. 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 politicians dataset, with *only* attributes *Anti_satellite_test_ban* and *Export_south_africa*. For this small dataset, the handout tar file 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.[2]

> **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.

---

[2]Yes, you read that correctly: we are giving you the correct answers.

2 Program #1: Inspecting the Data [5pts]

Write a program `inspection.{py|java|cpp}` 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:

For Python: `$ python inspection.py <input> <output>`

For Java: `$ javac inspection.java; java inspect <input> <output>`

For C++:   `$ g++ inspection.cpp; ./a.out <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 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`. For Python, you would run the command below:

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

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

```
entropy: 0.996316519559
error: 0.464285714286
```

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.

3 Program #2: Decision Tree Learner [65pts]

In decisionTree.{py | java | cpp}, 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.

- You should split with replacement. That is, when you split on a column, you should retain this column in child datasets.

- 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 attribute to split on that comes *last* in the lexicographical order (i.e. Republican should be chosen before Democrat)

- 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.

- Write a function to train a stump (tree with only one level). Then call that function recursively to create the 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's on Piazza for more useful clarifications about the assignment.

3.1 Command Line Arguments

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

For Python:	`$ `**`python`**` decisionTree.`**`py`**` [args...]`
For Java:	`$ `**`javac`**` decisionTree.`**`java`**`; `**`java`**` decisionTree [args...]`
For C++:	`$ `**`g++`**` -g decisionTree.`**`cpp`**` -o decisionTree; ./decisionTree [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 1)

2. `<test input>`: path to the test input `.tsv` file (see Section 1)

3. `<max depth>`: maximum depth to which the tree should be built

4. `<train out>`: path of output `.labels` file to which the predictions on the *training* data should be written (see Section 3.2)

5. `<test out>`: path of output `.labels` file to which the predictions on the *test* data should be written (see Section 3.2)

6. `<metrics out>`: path of the output `.txt` file to which metrics such as train and test error should be written (see Section 3.3)

As an example, if you implemented your program in Python, the following command line would run your program on the politicians dataset and learn a tree with max-depth of two. The train predictions would be written to `pol_2_train.labels`, the test predictions to `pol_2_test.labels`, and the metrics to `pol_2_metrics.txt`.

```
$ python decisionTree.py politicians_train.tsv politicians_test.tsv \
        2 pol_2_train.labels pol_2_test.labels pol_2_metrics.txt
```

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

```
$ python decisionTree.py politicians_train.tsv politicians_test.tsv \
        3 pol_3_train.labels pol_3_test.labels pol_3_metrics.txt
```

Linear Algebra Libraries When implementing machine learning algorithms, it is often convenient to have a linear algebra library at your disposal. In this assignment, Java users may use EJML[a] or ND4J[b] and C++ users Eigen[c]. Details below. (As usual, Python users have NumPy.)

EJML for Java EJML is a pure Java linear algebra package with three interfaces. We strongly recommend using the SimpleMatrix interface. The autograder will use EJML version 0.38. When compiling and running your code, we will add the additional command line argument `-cp "linalg_lib/ejml-v0.38-libs/*:linalg_lib/nd4j-v1.0.0-beta7-libs/*:./"` to ensure that all the EJML jars are on the classpath as well as your code.

ND4J for Java ND4J is a library for multidimensional tensors with an interface akin to Python's NumPy. The autograder will use ND4J version 1.0.0-beta7. When compiling and running your code, we will add the additional command line argument `-cp "linalg_lib/ejml-v0.38-libs/*:linalg_lib/nd4j-v1.0.0-beta7-libs/*:./"` to ensure that all the ND4J jars are on the classpath as well as your code.

Eigen for C++ Eigen is a header-only library, so there is no linking to worry about—just `#include` whatever components you need. The autograder will use Eigen version 3.3.7. The command line arguments above demonstrate how we will call you code. When compiling your code we will include, the argument `-I./linalg_lib` in order to include the `linalg_lib/Eigen` subdirectory, which contains all the headers.

We have included the correct versions of EJML/ND4J/Eigen in the `linalg_lib.zip` posted on the Piazza Resources page for your convenience. It contains the same `linalg_lib/` directory that we will include in the current working directory when running your tests. Do **not** include EJML, ND4J, or Eigen in your homework submission; the autograder will ensure that they are in place.

---

[a] https://ejml.org
[b] https://deeplearning4j.org/docs/latest/nd4j-overview
[c] http://eigen.tuxfamily.org/

3.2 Output: Labels Files

Your program should write two output `.labels` 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.

Note: You should output your predicted labels using the same string identifiers as the original training data: e.g., for the politicians dataset you should output democrat/republican and for the education dataset you should output A/notA. The first few lines of an example output file is given below for the politician dataset:

```
democrat
democrat
democrat
republican
democrat
...
```

3.3 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 0.01 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): 0.0714
error(test): 0.1429
```

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

3.4 Output: Printing the Tree

Finally, you should write a function to pretty-print your learned decision tree. (You may find it more convenient to print the tree *as* you are learning it.) Each row should correspond to a node in the tree. They should be printed in a *depth-first-search* order (but you may print left-to-right or right-to-left). 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 (example for python). You can print it directly to standard out rather than to a file. **This functionality of your program will not be autograded**.

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

[15 democrat/13 republican]
| Anti_satellite_test_ban = y: [13 democrat/1 republican]
| | Export_south_africa = y: [13 democrat/0 republican]
| | Export_south_africa = n: [0 democrat/1 republican]
| Anti_satellite_test_ban = n: [2 democrat/12 republican]
| | Export_south_africa = y: [2 democrat/7 republican]
| | Export_south_africa = n: [0 democrat/5 republican]
```

However, you should be careful that the tree might not be full. For example, after swapping the train/test files in the example above, you could end up with a tree like the following.

```
$ python decisionTree.py small_test.tsv small_train.tsv 2 \
swap_2_train.labels swap_2_test.labels swap_2_metrics.txt

[13 democrat/15 republican]
| Anti_satellite_test_ban = y: [9 democrat/0 republican]
| Anti_satellite_test_ban = n: [4 democrat/15 republican]
| | Export_south_africa = y: [4 democrat/10 republican]
| | Export_south_africa = n: [0 democrat/5 republican]
```

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 politics dataset (asked in question 14 of the Empirical questions).

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

[135 A/65 notA]
| F = A: [119 A/23 notA]
| | M4 = A: [56 A/2 notA]
```

```
| | | P1 = A: [41 A/0 notA]
| | | P1 = notA: [15 A/2 notA]
| | M4 = notA: [63 A/21 notA]
| | | M2 = A: [37 A/3 notA]
| | | M2 = notA: [26 A/18 notA]
| F = notA: [16 A/42 notA]
| | M2 = A: [13 A/15 notA]
| | | M4 = A: [6 A/1 notA]
| | | M4 = notA: [7 A/14 notA]
| | M2 = notA: [3 A/27 notA]
| | | M4 = A: [3 A/5 notA]
| | | M4 = notA: [0 A/22 notA]
```

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 1-7 in the "Empirical Questions" section of this handout. Write your solutions in the template provided.

4 Submission Instructions

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

```
inspection.{py|java|cpp}
decisionTree.{py|java|cpp}
```

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.

Note: Please make sure the programming language that you use is consistent within this assignment (e.g. don't use C++ for inspect and Python for decisionTree).

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.