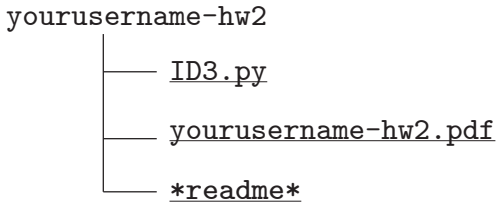
CS 373-Spring 2019
Instructor: Dan Goldwasser, Romila Pradhan
I collaborated with (...). I affirm that I wrote the solutions in my own words and that I understand the solutions I am submitting.

CS373 Homework 2

Due date: Monday, March 4, 11:59pm

Instructions for submission:
You need to implement this assignment from scratch in `Python`. Do not use any already implemented models like those in the `scikit-learn` library. Also, programs that print more than what is required will be penalized.
Your code needs to run in the server: `data.cs.purdue.edu`. Check that it works there by executing your script from the terminal. **For part 1, submit your code as a single file (ID3.py). For part 2, type your answers in a single PDF file. Compressed into zip file in the following format and upload it to the server**.



where you can include the `readme` if your code has special issues that we should be aware of. As usual, label the plots with the question number. Your homework **must** contain your **name** and **PUID** at the start of the file. If you omit your name or the question numbers for the plots, you will be penalized. We will also run through **code plagiarism checker** for your solutions, including all the previous years' codes
To submit your codes, log into `data.cs.purdue.edu` (physically go to the lab or use ssh remotely) and follow these steps:

1. Go to the directory containing `yourusername-hw2.zip` (e.g., if the files are in `/homes/dan/`, go to `/homes/dan`), and execute the following command:

   ```
   turnin -c cs373 -p hw2 yourusername-hw2.zip
   ```

   (e.g. Dan would use: `turnin -c cs373 -p hw2 dan-hw2.zip` to submit his work) Note that `cs373` is the course name for turnin.

2. To overwrite an old submission, simply execute this command again.

3. To verify the contents of your submission, execute the following command:

   ```
   turnin -v -c cs373 -p hw2
   ```

Part 0 Specification

Install Python3.7

You can install anaconda to configure the `Python` environment for Mac/Win/Linux at `https://www.anaconda.com/distribution/#download-section`. Make sure you install the right version, `Python-3.7-64-bit` installer is preferred. Then install the related packages, type this command in your terminal:

```
conda install pandas numpy scipy matplotlib
```

If you are not fully familiar with Python language, we recommend you to go though online tutorials before doing your homework. Recommended websites for you: `https://www.programiz.com/python-programming/tutorial` and `https://www.codecademy.com/learn/learn-python`.
You can also try out `Jupyter Notebook`[1] for real-time coding, which is quite similar to `R`.

Dataset Details

Find these files:

```
titanic-train.data, titanic-train.label, titanic-test.data, titanic-test.label
```

in the `HW2.zip` file. The overall attributes of dataset is shown in Table. 1. The definition of every feature is:

```
Survived: dead or survive (Bool);
Pclass: Class of Travel (Int);
Name: name of passenger(String);
Sex: Gender(Bool);
Age: Age of Passengers(Int);
Relatives: Number of relatives (Int);
IsAlone: If he/she has no relatives (Bool);
Ticket: ticket number (String);
Fare: Passenger fare (Float);
Embarked: Port of Embarkation (Int). C = Cherbourg (0), Q = Queenstown(0),
                                    S = Southampton(77);
```

Table 1: Samples in titanic raw data.

Survived	Pclass	Name	Sex	Age	Relatives	IsAlone	Ticket	Fare	Embarked
0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	1	Cumings, Mrs. John Bradley	female	38.0	1	0	PC 17599	71.2833	C
1	3	Heikkinen, Miss. Laina	female	26.0	0	1	STON/O2. 3101282	7.9250	S
1	1	Futrelle, Mrs. Jacques Heath	female	35.0	1	0	113803	53.1000	S
0	3	Allen, Mr. William Henry	male	35.0	0	1	373450	8.0500	S
0	3	Moran, Mr. James	male	35.0	0	1	330877	8.4583	Q

[1] `https://jupyter.org/install`

We consider 7 of the attributes as the input `features`: `Pclass, Sex, Age, Relatives, IsAlone, Fare, Embarked`. And consider the first attribute `Survived` as the `class label`. As shown in Table. 2, this is what you will read from the csv file.

Table 2: Sample data in the `titanic-train.data, titanic-train.label`.

Survived	Pclass	Sex	Age	Fare	Embarked	relatives	IsAlone
0	3	0	22	7	0	1	0
1	1	1	38	71	1	1	0
1	3	1	26	7	0	0	1
1	1	1	35	53	0	1	0
0	3	0	35	8	0	0	1
0	3	0	35	8	77	0	1

In order to read the CSV data and obtain our features and labels with the necessary attributes, you can use the following code:

```
import pandas as pd
X = pd.read_csv(data_file, delimiter = ',',  index_col=None, engine='python')
```

Input format

Your python script should take the following arguments:

1. `train-file`: path to the training set. (`train-file.data, train-file.label`)
2. `test-file`: path to the test set. (`test-file.data, test-file.label`)
3. `model`: model that you want to use. In this case, we will use:
   - `vanilla`: the full decision tree.
   - `depth`: the decision tree with static depth.
   - `min-split`: the decision tree with minimum samples to split on.
   - `prune`: the decision tree with post-pruning.
4. `train-set-size`: percentage of dataset used for training.

Each case may have some additional command-line arguments, which will be mentioned in its format section. Use following examples to get the list of arguments.

```
import sys
for x in sys.argv:
     print('arg: ', x)
```

Your code should read the training set from `train-file`, extract the required features, train your decision tree on the training set, and test it on the test set from `test-file`. Name your file `ID3.py`.
For debugging purposes, you can use a small fraction of the dataset, for example, by using `X[:100]` to work with the first `100` data points.

Model Details

Entropy

To help you build the model easier, we provide sample code for you to calculate the entropy:

```
import numpy as np
def entropy(freqs):
    """
    entropy(p) = -SUM (Pi * log(Pi))
    >>> entropy([10.,10.])
    1.0
    >>> entropy([10.,0.])
    0
    >>> entropy([9.,3.])
    0.811278
    """
    all_freq = sum(freqs)
    entropy = 0
    for fq in freqs:
        prob = ____ * 1.0 / ____
        if abs(prob) > 1e-8:
            entropy += -____ * np.log2(____)
    return entropy
```

where you need to pass the test case:

$$-1/2 * \log(1/2) - 1/2 * \log(1/2) = 1$$
$$-1 * \log(1) - 0 * \log(0) = 0$$
$$-3/4 * \log(3/4) - 1/4 * \log(1/4) = 0.811278$$

Information gain

we provide sample code for you to calculate the information gain:

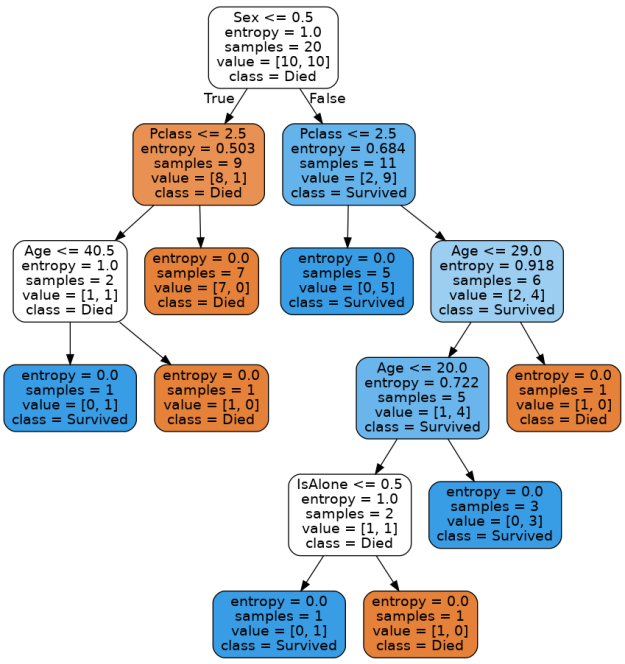
```
def infor_gain(before_split_freqs, after_split_freqs):
    """
    gain(D, A) = entropy(D) - SUM ( |Di| / |D| * entropy(Di) )
    >>> infor_gain([9,5], [[2,2],[4,2],[3,1]])
    0.02922
    """
    gain = entropy(____)
    overall_size = sum(____)
    for freq in after_split_freqs:
        ratio = sum(____) * 1.0 / ____
        gain -= ratio * entropy(___)
    return gain
```

where you need to pass the test case:

$$\begin{aligned}
\texttt{entropy(D)} &= 9/14 * \log(9/14) - 5/14 * \log(5/14) = 0.9402 \\
\texttt{entropy(Income=high)} &= -2/4 * \log(2/4) - 2/4 * \log(2/4) = 1 \\
\texttt{entropy(Income=med)} &= -4/6 * \log(4/6) - 2/6 * \log(2/6) = 0.91829 \\
\texttt{entropy(Income=low)} &= -3/4 * \log(3/4) - 1/4 * \log(1/4) = 0.81127 \\
\texttt{Gain(D,Income)} &= \texttt{entropy(D)} - (4/14 * 1 + 6/14 * 0.91929 + 4/14 * 0.81128) = 1
\end{aligned}$$

Sample decision tree

Here is the example decision tree by using the `sample.data, sample.label`:



Note: The tree building function and prune function should be implemented in a recursive manner, otherwise your solution score will be penalized by 80%.

Part 1 Decision Trees

Note: You need to finish only your code as a separate file (`ID3.py`) for this section.
Your algorithm should finish training and testing within 5 Minutes. We will verify your training and testing modules, and you should not include your pre-trained model.

1. (**20 points**) Implement a binary decision tree with no pruning using the ID3 (Iterative Dichotomiser 3) algorithm[2].

   Format of calling the function and accuracy you will get after training:

   ```
   $ python ID3.py ./path/to/train-file ./path/to/test-file vanilla 80
   Train set accuracy: 0.9123
   Test set accuracy: 0.8123
   ```

   The fourth argument (80) is the training set percentage. The above example command means we use only the first 80% of the training data from `train-file`. (We use all of the test data from `test-file`.)

2. (**20 points**) Implement a binary decision tree with a given maximum depth.
   Format of calling the function and accuracy you will get after training:

   ```
   $ python ID3.py ./path/to/train-file ./path/to/test-file depth 50 40 14
   Train set accuracy: 0.9123
   Validation set accuracy: 0.8523
   Test set accuracy: 0.8123
   ```

   The fourth argument (`50`) is the training set percentage and the fifth argument (`40`) is the validation set percentage. The sixth argument (`14`) is the value of maximum depth.
   So, for example, the above command would get a training set from the first 50% of `train-file` and get a validation set from the last 40% of `train-file` (the two numbers need not add up to 100% because we sometimes use less training data). Finally, we set the maximum depth of the decision tree as 14. As before, we get the full test set from `test-file`.

   Note: you have to print the validation set accuracy for this case.

3. (**20 points**) Implement a binary decision tree with a given minimum sample split size.
   Format of calling the function and accuracy you will get after training:

   ```
   $ python ID3.py ./path/to/train-file ./path/to/test-file min_split 50 40 2
   Train set accuracy: 0.9123
   Validation set accuracy: 0.8523
   Test set accuracy: 0.8123
   ```

[2] `https://en.wikipedia.org/wiki/ID3_algorithm`

The sixth argument (2) is the value of minimum samples to split on.
The above example command would get a training set from the first 50% of `train-file` and get a validation set from the last 40% of `train-file` (the two numbers need not add up to 100% because we sometimes use less training data). Finally, we set the minimum samples to split on of the decision tree as 2. As before, we get the full test set from `test-file`.

Note: you have to print the validation set accuracy for this case.

4. (**20 points**) Implement a binary decision tree with post-pruning using reduced error pruning.
Format of calling the function and accuracy you will get after training:

```
$ python ID3.py ./path/to/train-file ./path/to/test-file prune 50 40
Train set accuracy: 0.9123
Test set accuracy: 0.8123
```

The fourth argument (50) is the training set percentage and the fifth argument (40) is the validation set percentage.

So, for example, the above command would get a training set from the first 50% of `train-file` and get a validation set from the last 40% of `train-file`. As before, we get the full test set from `test-file`.

Part 2 Analysis

Note: You need to submit only your answers in the PDF for this section.
For the following questions, use `titanic-train.data, titanic-train.label` as the training file and `titanic-test.data, titanic-test.label` as the test file. You should use `numpy, matplotlib, seaborn` for plotting the graph, and include your code scripts. Make sure your `xaxis, yaxis, title` has proper name.

1. (**4 points**) For the full decision tree (`vanilla`), measure the impact of training set size on the accuracy and size of the tree.
   Consider training set percentages $\{40\%, 60\%, 80\%, 100\%\}$.
   Plot a graph of test set accuracy and training set accuracy against training set percentage on the same plot.
   Plot another graph of number of nodes vs training set percentage.

2. (**7 points**) Repeat the same analysis for the static-depth case (`depth`).
   Again, consider values of training set percentage: $\{40\%, 50\%, 60\%, 70\%, 80\%\}$. The validation set percentage will remain 20% for all the cases.
   Consider values of maximum depth from $\{5, 10, 15, 20\}$ and pick the best value using the validation set accuracy. The accuracies you report will be the ones for this value of maximum depth. So, for example, if the best value of maximum depth for training set 40% is 5, you will report accuracies for 40% using 5; if for 50% it is 10, you will report accuracies for 50% using 10.
   Plot a graph of test set accuracy and training set accuracy against training set percentage on the same plot. Plot another graph of number of nodes vs training set percentage.
   Finally, plot the optimal choice of depth against the training set percentage.

3. (**7 points**) Repeat the above analysis for the pruning case (`prune`).
   Again, consider values of training set percentage: $\{40\%, 50\%, 60\%, 70\%, 80\%\}$. The validation set percentage will remain 20% for all the cases. You will use the validation set when deciding to prune.
   Plot a graph of test set accuracy and training set accuracy against training set percentage on the same plot.
   Plot another graph of number of nodes vs training set percentage.

4. (**3 points**) Why don't we prune directly on the test set? Why do we use a separate validation set?

5. (**4 points**) How would you convert your decision tree (in the depth and prune cases) from a classification model to a ranking model?
   That is, how would you output a ranking over the possible class labels instead of a single class label?