Assignment 2
Using ID3 to Build a Decision Tree Model.

Late Submission Rules:

Up to 2 days late: 30% penalty.
More than 2 days late: not accepted

Objective

Using data from the census bureau database for the year 1994, use the ID3 algorithm to build a decision tree that classifies whether a person's yearly income is ≤ 50K or > 50K. The data was obtained from here (University of California – Irvine).

You must use the **ID3** algorithm (pseudocode on page 134 in Kelleher et al book) and **Entropy/Information Gain** as the statistical test to determine on what attributes to split data.

Training and Test Data Provided to You

The training and test data are found in the enclosed subfolder named "data". There are 4 files:

- ***census_training.csv***: use this dataset (30110 training examples) to build your decision tree model using ID3 and Info Gain.
- ***census_training_test.csv***: use this dataset (15028 examples) to test the accuracy of your decision tree model.

  The data in *census_training.csv* and *census_training_test.csv* consist of 11 attributes (all categorical) and 1 class label: *high_income*, which can be <=50K or >50K.

- *playtennis*.csv: a small dataset for which you know how the tree looks like. Use this small dataset to validate that your algorithm is somewhat working (tree shape is in Appendix A of this document)
- *emails*.csv: a small dataset for which you know how the tree looks like. Use this small dataset to validate that your tree is somewhat working (tree shape is in Appendix A of this document).

NOTE: I suggest you validate your algorithm on *playtennis* and *email* before running it on the census data.

The following pre-processing was done to *census_training.csv* and *census_training_test.csv*:

- 3 attributes that existed in the original dataset were removed. These are *fnlwgt*, *capital_gain*, and *capital_loss*.
- 3 attributes that are real-valued in the original dataset were converted to categorical attributes. These are *age, education_num*, and *hours_per_week*.
- Rows containing missing data or unknown data were removed.

Implementation Hints

- Build your code incrementally: create helper functions that you are likely to call (calculating entropy, information gain, best feature to choose for a split, etc.). **Test those functions on small data for which you know the results of the calculation** (e.g., you should already have Info Gain calculation code from Assignment 1 validated against PlayTennis Info Gain numbers that we know). **Ensure that these are working and doing correct calculations before proceeding**.

- Code your algorithm on small datasets that you know how their resulting decision trees look like: ***playtennis*.csv and *emails.csv* were provided to you for this purpose**. The shapes of the trees are in Figure 3 and 4 of Appendix A on the last page of this document.

- When building the decision tree, save it in memory as a dictionary of dictionaries (not a requirement, just a recommendation). For example, when printing the Email and PlayTennis trees (dictionaries), you should get something similar to:

```
{'SUSPICIOUS WORDS': {True: 'spam', False: 'ham'}}
```

```
{'Outlook': {'Rain': {'Wind': {'Weak': 'Yes', 'Strong': 'No'}}, 'Sunny': {'Humidity': {'Normal': 'Yes', 'High': 'No'}}, 'Overcast': 'Yes'}}
```

- After verifying that your algorithm is building the correct PlayTennis and Email trees, then you can test your algorithm on training data *census_training.csv*.

- Now use *census_training_test.csv* to test the accuracy of your Decision Tree model. Data in *census_training_test.csv* is data that your algorithm have never seen. So it is a good set to calculate how accurate your model prediction is. Count the number of accurate classifications and the number of inaccurate classifications.

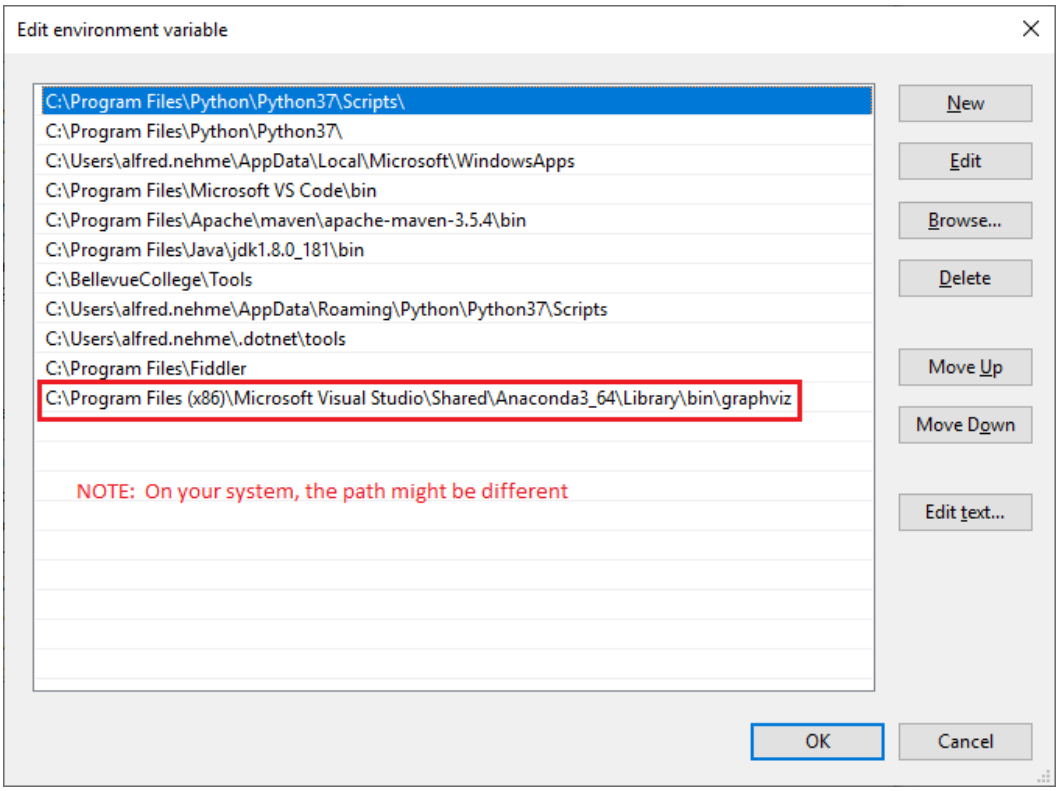
- If you represent your tree as a Python dictionary, you can use the function you used in Module 3, `draw_decision_tree_dictionary`, to get a visual of the tree. It is not a requirement to get a visual of the tree. However, after spending many hours you are probably curious to see how your tree looks like. In some cases, looking at the tree can reveal the existence of bugs.

  To render the tree to PDF, you can use code like this:

```
dot.render('censustree.gv', view=True)
```

where `dot` is the object returned by function `draw_decision_tree_dictionary`.

If you are coding in an IDE other than Jupyter Lab (example: VS Code), you may need to update your PATH environment variable and include the path where graphviz was installed as shown below. If you don't do that, you may get an error when trying to render the tree.



NOTE: when you want to test multiple times, make sure to close the PDF document that displays the tree. Otherwise you might get an error that the document is open and cannot be overwritten.

- My implementation takes no more than 2 minutes for training and testing to finish. If you see yourself waiting much longer, you might possibly have some problem in your recursion.

- You don't have to use Jupyter Lab. You can use it if you want to but it is not required. You can simply use any Python IDE that you like.

Output of Program

Show the output of your program **without** any tree pruning done. The output should print the accuracy against the test data (i.e., the data from ***census_training_test.csv***). Here is how mine looks like

```
================ Training started on 30110 examples
================ FINISHED training using training examples from: C:/BellevueCollege/C

================ TESTING STARTED

Number of testing examples = 15028
correct_classification_count = 12060
incorrect_classification_count = 2968
Accuracy = 80.25019962736226 %
================ TESTING ENDED
```

Figure 1: Output without pruning.

Now add tree pruning, and re-run your training/testing. You should see a slight improvement. Mine improved from 80.25 to 81.88 when I set my threshold to 30 (that is I stopped building the tree when the current partition size is <= 30).

```
================ Training started on 30110 examples
================ FINISHED training using training examples from: C:/Belle

================ TESTING STARTED

Number of testing examples = 15028
correct_classification_count = 12305
incorrect_classification_count = 2723
Accuracy = 81.88048975246207 %
================ TESTING ENDED
```

Figure 2: Output with pruning.

Grading

It is hard to run/grade a project like this by just looking at the code. I might ask students to meet with me for 5 min to demonstrate that their code works.

Your grade will be computed as follows:

1. 60% - Code runs without errors and builds a tree using ID3 and Info Gain. Your program outputs information somewhat similar to what I have above (you might get slightly different accuracies). Some of you might get better accuracies than my implementation and this is great. But accuracies below 75% might indicate a bug somewhere. You will get full credit for any accuracy above 80%.

2. 10% - You calculated the accuracy by checking your model against the test data. (***census_training_test.csv***).

3. 10% - You implemented tree pruning and showed a slight improvement in accuracy.

4. 20% - Code is well commented, and you can explain your code with confidence (if asked to do so).

    a. Algorithm steps are correct as explained in book and class.
    b. Each function should have a comment as to what it does and what it returns.
    c. There are comments along the steps your program is taking and I am able to easily follow/understand the steps.

What to Submit:

Do not include the data files (*.csv files) with your submissions.

a. Create a folder with your first name, last name, and assignment2 (e.g., john_smith_assignment2).

b. Put in this folder a PDF showing the outputs of your program (that is, similar to Figure 1 and 2 above).

c. Put in this folder your code - *.py file(s) or a JupyterLab folder (if you used JupyterLab) but without the *csv files.
d. Zip john_smith_assignment2 to generate john_smith_assignment2.zip.
e. Upload john_smith_assignment2.zip to Canvas before the due date.

Appendix A

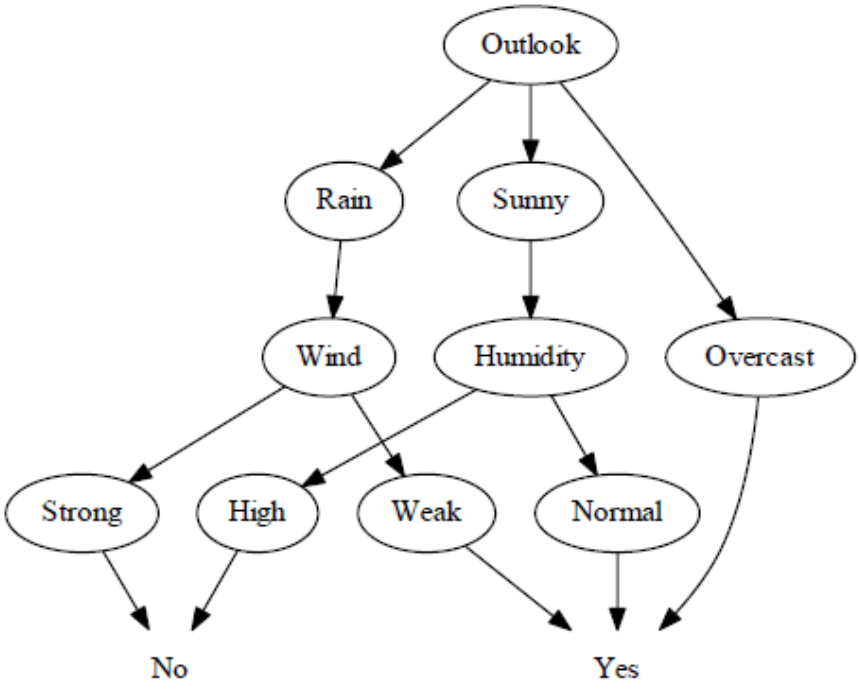


Figure 3: PlayTennis decision tree.

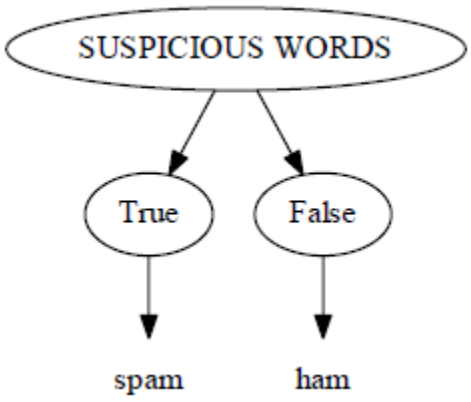


Figure 4: Spam/Ham emails decision tree.