Decision Tree Generation
Individually or in Pairs
See the associated Dropbox for due date.
Thomas B. Kinsman, PhD

Caution:

For the rest of the course, you have to submit projects which meets the following requirements.
Strict non-linear penalties will be imposed for violating them.

Violating any of the following during a technical interview has kept previous student from getting jobs. No kidding.

1. Your variable names will be at least marginally descriptive of the purpose they serve.
   Example: 50% penalty for using single letter variable names. You can easily think this is not important but it is important to Dr. Kinsman to make students well prepared for the professional world. The variables named i, j, and k… are from Fortran IV, around the time of 1974.

   Single letter variable names are great for mathematics, and theory, but not for communicating information in code.

2. Your code must be well documented so that the graders understand everything you do. The graders will not be experts in the package you are using. You will need to have comments to teach them how you solved the problem. If you use fancy python tricks, such as list comprehensions and zip, you need to explain what is happening. Your code should stand on its own.

   Example: 50% penalty if you use data tables in pandas, but do not explain what your code does.

3. You need to submit a PDF file which can be read, and which explains what you did.
   Your PDF file must stand-alone. It should describe what you did, your thought process, your results, and give a strong conclusion that demonstrates evidence of learning.

   Example: Just answering the questions from the homework, and not giving a conclusion will upset the graders. If the graders take more then 15 minutes to understand what you did, they can give you an arbitrary penalty, and Dr. Kinsman will stand behind them.

4. You should be able to describe your approach to solving the problem, and writing your code on a future quiz or exam.

The Decision Tree Project:
This homework assignment grade counts for two homework assignments. It will be entered twice in MyCourses twice.

Assume that the grader has no knowledge of the language or API calls, but can read comments.
Use prolific block comments before each section of code, or complicated function call, to explain what the code does, and why you are using it. Put your names and date in the comments at the heading of the program.

Hand in: One directory named HW_NN_LastName1_LastName2_dir
Zip up the entire directory such that when the zip file is unzipped, we see the directory HW_NN_LastName1_LastName2_dir on our end. Not several files in the current directory.

Test that this works before submission. If there are three on your team, make the obvious substitutions.

Inside that directory:
1. Your write-up, in either DOCX or PDF format,
2. Your mentor or training program code,
3. The resulting classifier, the program that you mentor program created,
4. The results your classification results of the provided **validation** data file.
   HW_NN_LastName_FirstName__MyClassifications.csv**.**

A csv file is a *comma separated value* file. It contains one record per line, with comas.

You are provided with a file of training data. This data has several attributes to select from.
They might include:
- The age of maturity the snowfolk (when their hair turns grey)
- Their height at maturity
- The length of their Bangs – the BangLn.
- The length of their tails, the TailLn
- Hair length …
- If their earlobes are attached.

Your goal is to classify the test data into Assam (-1) or Bhutan (+1).

Using the training data, your goal is to write one decision tree that classifies the results. Use only the attributes provided.
No feature generation is allowed.

The decision tree training program:
Name this program HW_NN_LastName_FirstName_Trainer…
You write a program that implements the generic decision tree creation process, recursively.
It creates a decision tree, using the decision tree algorithms we discussed in class.
The output of this program is another program, a trained classifier.

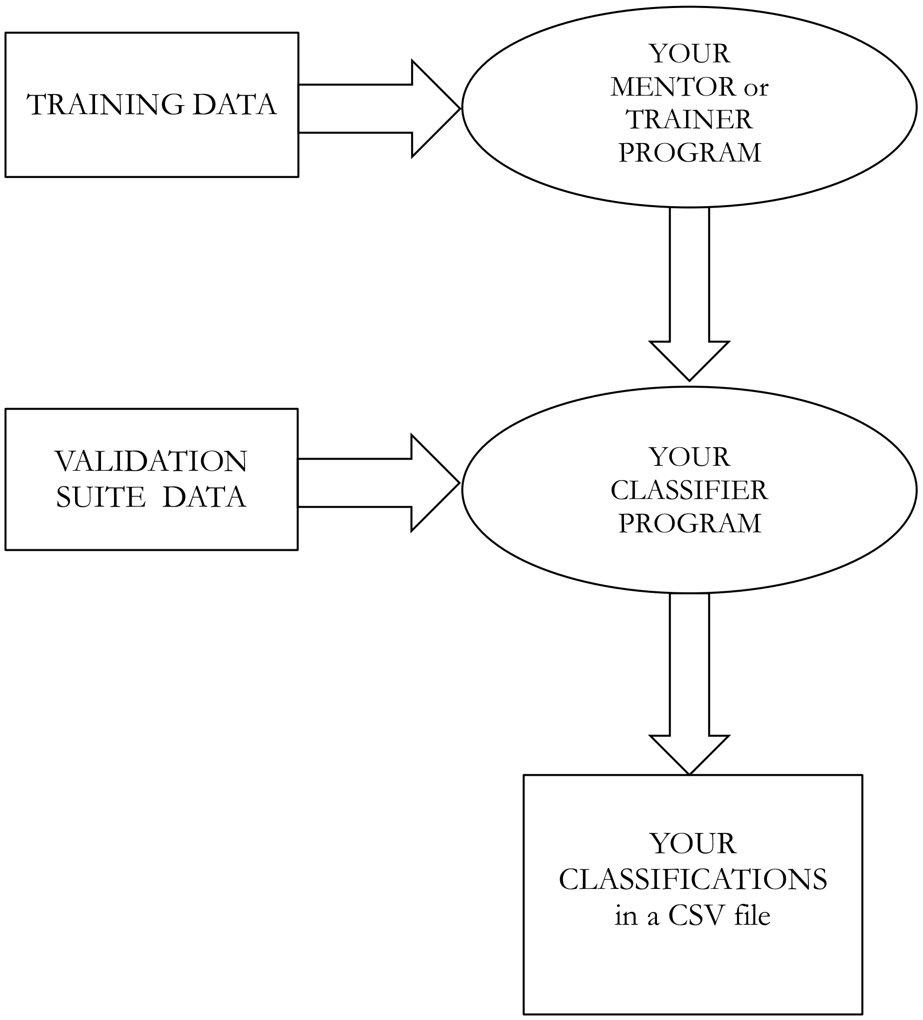
The trained classifier program must be able to read in the *.csv file that is used to train the decision tree.
So, the input to the decision tree training program is a flat file.

For simplicity's sake, you can write this program to make hard-coded assumptions. It can assume that there are only numeric attributes, and that there are only a certain number of attributes. So, don't feel like you need to write a generic decision tree training program.

The goal is to get the best accuracy on *the* validation data as possible.

Again, the output of this trainer program is *another program*, to be called **HW_NN_Classifier_LastName1_LastName2.py**

Your program should actually generate this classifier program. This takes time to do it correctly.
The following diagram shows the overall flow of the program generation.


TRAINING DATA
YOUR MENTOR or TRAINER PROGRAM
VALIDATION SUITE DATA
YOUR CLASSIFIER PROGRAM
YOUR CLASSIFICATIONS in a CSV file

The trained classifier program: HW_NN_Classifier_Lastname.py.

This resulting decision tree classifier program looks *something* like this.

```
…. header and prologue …

Given one parameter --
the string containing the filename to read in.

read in the input file
for each line in the input_test_data :
  if ( attribute_a >= threshold_a ) :
        if ( attribute_b >= threshold_b ) :
                class = -1 for Assam.
        else :
                class = +1 for Bhutan.
     else :
        if ( attribute_c >= threshold_c )  :
                class = -1 for Assam
        else :
                class = +1 for Bhutan
 else …

 print( class )

# print out the class value for each line of the test data file.
```

The goal of your decision tree mentor program is to select which attributes to use,
in which order, and the appropriate thresholds.

You will use the data in the *training_data* file to write this program.
Then you will run this program on the *validation_data* file, and guess the classification of each.

When you will run the classifier, have it print out one classification per line, so that the grader can quickly go down the list and compare to his or her answers.

The output should also be a file called **HW_NN_LastName1_MyClassifications.csv.** This means duplicating the print statement so that one copy goes to standard output, and one does into the classifications file.

Project plan:
Some students do not know where to begin, and are overwhelmed. Here is a possible plan of action:

1. Create a test suite. A small set of data that will let you test your code, with known results.
   Do not start with 16,000 lines of data. Start with 8. (Four of class Assam, and four of class Bhutan.)

   You will probably want multiple test suites, for which you know the answers. For example:
   a. Test_suite_A_height.csv, which has 8 records, and the height separates the two classes perfectly at 100 cm.
   b. Test_suite_B_age.csv, which has 8 records, and the age separates the two classes perfectly at 30 years.
   c. Test_suite_C_tail.csv, which has 8 records, and the tail length separates the two classes perfectly.
   Get your code working for the small test suites. Then work on bigger, more complicated data.

2. Write a mentor program that only looks at Attribute1, and does not use recursion.
   Check that your classifier program works.
   This should be a decision stub, similar to previous homework assignments.

3. Then change the program so that it tries all input attributes, and makes a decision based on the best one. This program should look like before, but it *might* not use Attribute1 for classification.

   a. Then add recursion, so that the program calls itself.
      You need to add a parameter which is the "call depth" of the call. This keeps the program from recursing forever. And, for python programs, you need to know how far to indent your "if" statements and the "else:" statements, which is a function of the call depth.

      This should generate a more complicated program.
      Again, make sure that your resulting classifier works.

The Design Decisions you have for this year are:

1. Quantize the age to the nearest 2 years, using rounding.
2. Quantize the height to the nearest 5 cm, using rounding.
3. Quantize anything else to the nearest 1 value, using rounding.

4. This semester your resulting classifier should use only binary splits.

5. Every leaf node should try to have at least 3 records in it.
   (Minimum leaf node size is 3 records.)
   Caution: Setting this to 3 might cause over-fitting.
   You might want more nodes as your minimum number of nodes in a leaf node.

6. The maximum depth of nested if-then statements should be 26 levels of depth.
   Again, stopping at 10 levels of call depth might lead to less over-fitting.

7. Use the Weighted Gini Index to decide which split is the best split.

8. Your code should stop recursing if:
   a. There are less then 3 data points in a node, OR
   b. The node is greater than or equal to 95% one class or the other, OR
   c. The tree depth has greater than or equal to 26 levels of decision nodes.

Commonly Seen Mistakes:

A. NOTE: your resulting classifier must round the input data the same way.
   Otherwise, it will not work the same way.

B. The resulting classifier must make the same <, <=, >, or >= decisions that the main training program did.

C. Some students forgot to quantize the data AT ALL.
   This is done to remove noise, and make the mentor program run faster.

   So, let's look at the heights, suppose that they range from [0 to 200], and are rounded to the nearest 2 values.

   With rounding, YOU ONLY NEED TO CHECK (200/2) ~= 100 different thresholds.
   IF you forgot to do this, and you have 16,000 Data points, then you are checking 16,000 possible thresholds.
   PRE-QUANTIZING the data is MUCH-MUCH Faster to do.

Write-Up Questions:

1. **Write-Up: HW_NN_LastName_FirstName.pdf**
   Create a significant write up that shows strong evidence of learning.

   a) What were your names?

   b) Who did what roles during the assignment?
   Was one person responsible for software quality assurance and documentation?
   Was one person responsible for coding? What was the division of labor?

   c) What was the maximum call depth of your final classifier?
   Did it actually use the maximum number of levels?

   d) Describe the decisions of your final trained classifier program (the resulting classifier).
   What were the most important attributes? Inspecting it, what does it tell you about the relative importance of the attributes? What is the most important attribute?

   b) Generate a confusion matrix for the original training data:
   How many Assam were classified as Assam?
   How many Assam were classified as Bhutan?
   How many Bhutan were classified as Assam?
   How many Bhutan were classified as Bhutan?

   c) What was the accuracy of your resulting classifier, on the training data?
   That is (# Correctly Classified) / (Number of Data Records)

   d) What was the hardest part of getting all this working?
   Did anything go wrong?

   e) Optional Bonus:
   Did you try anything different to get a bonus? For example, did you plot scatter plots to examine the data?
   Did you try implementing splitting using Entropy, or Misclassification Error, or another splitting criterion?

f) **Conclusions**
What did you discover?

Did you run into is the accuracy paradox?

Was the data completely separable?

Write full paragraphs, and full sentences. Show strong evidence of learning…