Fall 2023

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

The Decision Tree Project:
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_05_LastName_Firstname_dir, or HW_05_LastName1_LastName2_dir.
Zip up the entire directory such that when the zip file is unzipped, we see the directory on our end.
Not several files in the current directory. *Test that this works before submission.*

Hand in only one submission for both of you. Make the obvious changes to your filenames so we can see two unique names.

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**.**

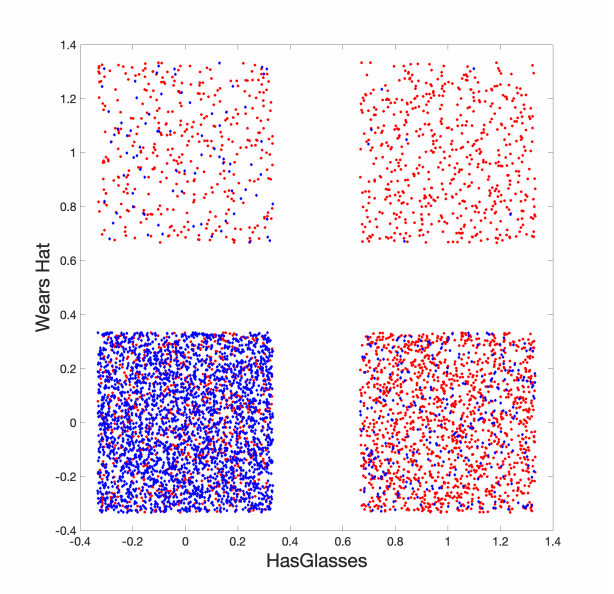
A csv file is a *comma separated value* file. It contains one record per line, with comas.

You are provided with a file of training data. This data has several attributes to select from.
Do not assume that the list is fixed. Other attributes might be added.
The last attribute is the target attribute. Your job is to write a program which will predict when the INTENT is 2 (aggressive).

Using the training data, your goal is to write one decision tree that classifies the results.

Exploratory Data Analysis (EDA):
Looking at the data, it is biased towards non-aggressive drivers. To fix that, I balanced the data by deleting enough non-aggressive drivers so that the number of non-aggressive and the number of aggressive matched.

One of the first steps when getting to know your data is to plot each attribute versus every other attribute. I wrote a program to do this, and in short order discovered the following relationship:



This shows that if the driver is wearing a hat, and the driver is wearing sunglasses, the intentions are likely to be aggressive and the driver is more likely to need to be pulled over.

You don't have to use scatter plots here. I could have created four sets of bar graphs. The advantage of scatter plots is that they work well with continuous variables (like the speed), whereas bar graphs only work for categorical data.

You can also use scatter plots for data cleaning. If the plots show just a few outliers in a group, you might want to identify them and delete them. That makes for a simpler classifier on your part.

The point here: creating scatter plots that plot each feature versus another only takes about an hour to code up, and it give you a visual way to quickly see differences in your data.

Here's a question with a caution – Hint:
What is the minimum number of scatter plots that I need?
Answer: If I have 15 attributes, then I only need (15*14)/2 plots. Why?

The 15*14 is because if the first attribute is number 1, you don't need to plot number 1 against number 1. There are not 15 times 15 combinations, there are only 15 times 14 combinations.

The division by two is because if you plot WearsHat vs HasGlasses, you do not need to also plot HasGlasses versus WearsHat. In other words, if you plot HasGlasses on the X axis and WearsHat on the Y axis, you don't need to plot the transpose. Only half of the plots are needed.

So, your loops should look something like this:

```
for first_attribute = 1 upto (number_of_attributes - 1)
      for second_attribute = (first_attribute+1) upto number_of_attributes
            plot( first_attribute versus second attribute
```

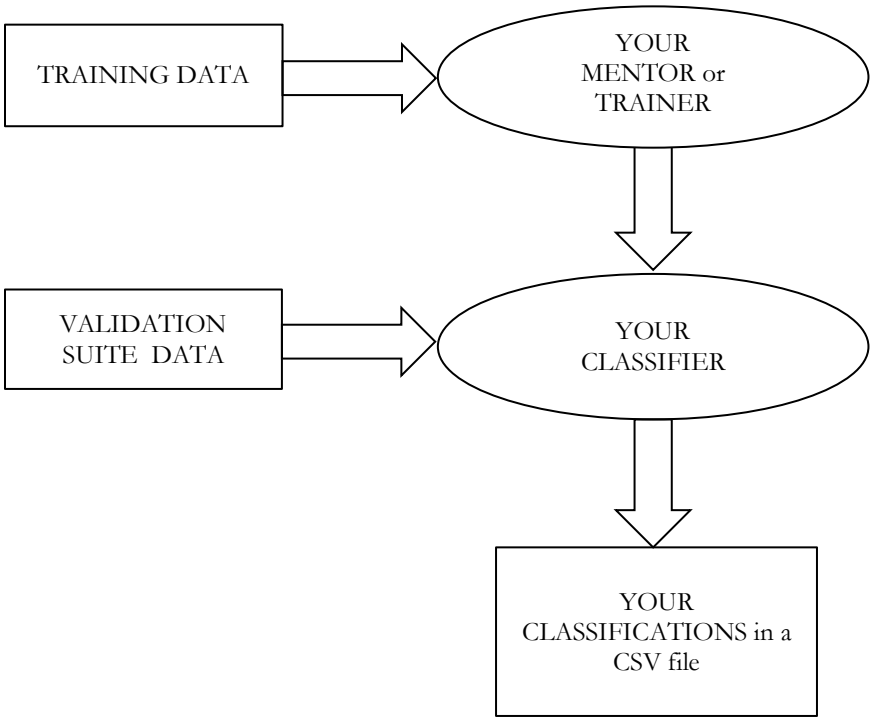
The decision tree training program:
Name this program HW_NN_LastName_FirstName_Trainer.
You write a program that implements the generic decision tree creation process, recursively.
It creates a decision tree, using the decision tree algorithms we discussed in class.
The output of this program is another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

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

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_LastName_Firstname.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.



The trained classifier program: HW_NN_Classifier_Lastname_Firstname.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 ) :
              intent = 2;                  # aggressive.
       else :
              intent = 1;                  # non-agressive.
    else :
       if ( attribute_c <= threshold_c )  :
              intent = 2;                  # aggressive.
       else :
              intent = 1;                  # non-agressive.
   else …

   print( intent ). # print out the class value for each line of the provided validation 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 might mean duplicating the final print statement so that one copy goes to standard output, and one does into the classifications file.

A Possible Project plan:

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

1. Re-balance the data. There are more non-aggressive drivers than aggressive drivers.
   This is data bias. You need to remove enough of the non-aggressive drivers so you have the same number of each target class.

2. Create a test suite. A small set of data that will let you test your code, with known results.
   Do not start with ten thousand lines of data. Start with eight. (Four aggressive, and four non-aggressive.) Set up each test suite so that you know what to expect the classifier to produce.

   You will probably want multiple test suites, for which you know the answers. For example:
   a. Test_suite_A_speed.csv, which has 8 records, and the speed separates the two classes perfectly.
   b. Test_suite_B_changes.csv, which has 8 records, and number of lane changes separates the two classes.
   c. Test_suite_C_tailgating.csv, which has 8 records, and the tailgating attribute separates the two classes perfectly.

   You get the idea. Make something up. Get your code working for the small test suites.
   Then work on bigger, more complicated data.

3. 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 assignment.

4. Then change the program so that it tries each input attributes.
   This might tell you which attributes work, and which should be ignored. (Good idea.)

5. Then write the program to try each input attribute, and only use the best one for this call.
   This program should be similar to HW_04 – it creates a one-rule.
   The resulting classifier uses only the best attribute.

6. Then add recursion, so that the program calls itself.
   You need to add a parameter which is the "call depth" of the call for two reasons:
   i. This keeps the program from recursing forever.
   ii. 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.

Design Decisions for this semester are:

1. **Round** all of the input values to the nearest integer.
   (54.4 → 54. 54.5 → 55.)

2. **Figure out which attributes to ignore.**
   Some of the attributes are complete garbage.
   If you can figure out which ones, you can have your training program ignore them.

3. **Feature Generation:**
   You might want to create a new feature or so.
   Your mentor program, the initial code can create features if it likes. For example, you might create a feature called "has_dents", which is true if either there are dents in the bumper or scratches on the sides of the car. Or you might create a feature called, "reckless" if the driver is distracted and has a speed above some threshold.

   However, your resulting classifier must create those same features.

4. Every leaf node should try to have greater than, or equal to, 5 records in it.
   Caution: Setting this to 5 might cause over-fitting. You might want to use a different value.
   You might want more nodes as your minimum number of nodes. That is your design decision.

5. The maximum depth of nested if-then statements should be 8 levels of depth.
   Again, you might want to stop sooner to avoid over-fitting.

6. Use the Information Gain Ratio to decide which split is the best split.
   Again, this is the entropy of the combined two nodes, minus the average entropy of the two nodes.

7. Your code should absolutely stop recursing if:
   a. There are less then 5 data points in a node, OR
   b. The node is greater than or equal to 90% one class or the other, OR
   c. The tree depth has greater than or equal to 8 levels of decision nodes.

Commonly Seen Mistakes:

A. NOTE: your resulting classifier must round and clean 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. The classifier program does not create the same features, in the same way that the mentor program did. If you create a feature named accident, (`accident = sideDents + bumperDents`),
   then your resulting classifier needs to create that same feature.

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

   YOU ONLY NEED TO CHECK a few different thresholds.
   IF you forgot to do this, and you have 10,000 Data points, then you are checking 10,000 possible thresholds. PRE-QUANTIZING the data is MUCH-MUCH Faster to do.

(continued)

Write-Up Questions:

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

   a) Your name, or names if you worked in pairs.

   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 (the ones at the top)?
   Inspecting it, what does it tell you about the relative importance of the attributes? What is the most important attribute?

   e) What was your choice of a minimum number of records in a node?

   f) What was your choice for a maximum splitting depth for the code?

   g) What was your choice for a maximum node purity to stop at?

   h) What features did you create, if any?

   i) Did you discover any features which could be ignored?

   j) Generate a **confusion matrix** for the *original* training data:
   How many aggressive were classified as aggressive?
   How many aggressive were classified as non-aggressive?
   How many non-aggressive were classified as non-aggressive?
   How many non-aggressive were classified as aggressive?

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

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

   m) **Conclusions**
   How did you clean your data? What pre-processing did you do?

   Did you create any new features to use?

   What did you discover? Anything else you discovered along the way?

   Did you run into the accuracy paradox?

   Was the data completely separable?

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

Cautions and Grading Rubrics:

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.

This assignment counts for two homework assignments. You get extra time for it, and you will need extra time for it.

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. The one exception is the variable named '_' in python, which is often used as a dummy variable.

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: if you use data tables in pandas, explain what your code does.
   Example: if you use R, explain how to run your program.

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.

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