HW Decision Tree Creation – Classifying the target variable muffins
Thomas B. Kinsman
CICS-420

The goal of this exercise is to write your own Decision Tree writing program. The idea of a program that writes a program has been around for decades. That is what a compiler does. Programs that create HTML do it. PDF files are programs that are created by programs. Other examples of programs that write programs abound.

It sounds more complicated then it is. However, it is important because you will be augmenting this program, and using it for a later homework.

As always, put your name and date in the comments at the heading of the programs, and use prolific block comments.

In addition to the standard rubric, 10% of your grade is based on getting a classification accuracy over 56% for the supplied testing data.

Hand in – <u>in a single ZIP file</u>:

1. Please remember to put your files into one directory named HWNN_First_Last_Name
2. Put all your files into this one directory:
3. Your write-up as a PDF file,
    a. Your well commented training code, (the "mentor" or learning program),
    b. The classifier (code) which your training program created,
    c. Your results your classifications run on the test data in a one column *.csv file.
    d. Your write-up with answers to questions, and a conclusion.

You are provided with a file of training data. The attributes are continuous numerical variables.

Using the training data, your goal is to write a program to create a decision tree. Each "if statement" can only test the attributes in the given data. (No feature generation allowed.)

The decision tree training program: Name this HW_NN_LastName_FirstName_Trainer.*
(with the obvious substitutions.) You write a program that creates a decision tree program, using the decision tree algorithms we discussed in class.

For consistency this semester, use the Weighted GINI index on a binary split. In theory we could use any of the common metrics for impurity: misclassification error, GINI index, Entropy, Information Gain, Information Gain Ratio, or mixed versions of these.

The output of your program is another program, a classifier program. It if perfectly legal to outline this program first, and then pull the constant parts of it into your training program, so that you're the training program has a function like "emit_prologue(file_pointer)" which emits most of the program. This "constant code" is sometimes referred to as "boiler plate" code. The resulting classifier reads in a similar *.csv file, and puts out a 1 for a muffin and a 0 for a cupcake.

The training program must be able to read in the training data, a *.csv file that is used to train the decision tree.

Do not feel like you need to write a completely generic decision tree training program. For simplicity's sake, you can write this program to make hard-coded assumptions. It might assume that the there are only numeric attributes (which is true) and that there are only a certain number of attributes.

The goal is to get the best accuracy on your *validation* data as possible. In this case you don't know the classifications (labels) of the test data.

The classifier program HW_NN_LastName_FirstName_Classifier.*

Your resulting output decision tree classifier program might look *something* like this.

```
…. header or prolog …

// Uses one parameter, which is the filename of the CSV file.

read in the validatioin file
for each line in the validation_data:

   if ( attribute_a >= threshold_a )
      if ( attribute_b >= threshold_b )
            class = [ 0 or 1], you choose;
      else
            class = [ 1 or 0], you choose;
   else
      if ( attribute_c >= threshold_c )
            class = [ 0 or 1], you choose;
      else
            class = [ 1 or 0], you choose;

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

The goal of your decision tree training program is to select attribute_a and threshold_a, then attribute_b and threshold_b, and then attribute_c and threshold_c, and so on. You might have a different structure than this, but the goal is to minimize the number of if statements (decision nodes) to 8 if statements.

Stages:

A. Use the data in the *training_data* file with your training program to write the classification program.
B. Run your classifier program on the *validation_data* file, and estimate (guess) the class.
   This prints out your answers in a 1 column (1 or 0) classification.
   Write out a *.csv file which is 1 in each line if you think the input data is class 1,
   and a 0 if the class is a zero.

   (Keep reading…)

1. **Files to Submit:**
   Turn in the following code items, and support files IN ONE ZIP FILE:
   a. **HW_NN_LastName_FirstName_Trainer…**

   b. **HW_NN_LastName_FirstName_Classifier…**

   c. **HW_NN_LastName_FirstName_MyClassifications.csv**
      This is your classifier run on the given Testing Data!

   d. **HW_NN_LastName_FirstName_Writeup.pdf**

   (continued)

2. **Work Breakdown Structure:**
   Think you are overwhelmed? Here are some implementation steps:
   a. **HW_NN_LastName_FirstName_Trainer**
      Write the procedure to read in the *.csv file that is given.
      There are read_table functions in R, Matlab, and python.
      Use one of them.

   b. **Add an "emit header" procedure.**
      Have HW_NN_LastName_FirstName_Trainer open up a file pointer to a file named HW_NN_LastName_FirstName_Classifier.

      Have your training program copy into this file a routine to read and go through every line of a given *.csv file. (Just like HW_NN_LastName_FirstName_Trainer does not.)

   c. **Add an "emit_classifier call" procedure.**
      Emit code for the classifier routine so that it will pass a line of data to a function for classification you name. Call it "my_classifier_function(data_record)".
      This assumes that there is a function somewhere in the training routine named my_classifier_function( ).

      Then the classifier program prints the result of the call, and continues to loop.

   d. **Add an "emit trailer" procedure.**
      Have the training procedure add to the classifier the end of the main routine onto the classifier program.

   e. **Add an "emit_decision_tree" procedure.**
      This has the training program create a procedure in the classifier that takes in a line of data, and classifies it. Eventually, this is the decision tree. For now, it is a stub.
      Have it always return 1 initially.

   f. **Test your classifier.**
      At this point you should be able to run your training program, produce a classifier program.
      Then you should be able to run your resulting classifier on your training data.
      It will always generate 1's, but it should be able to run.

   g. **Then change the "emit_decision_tree( )" procedure.**
      Then write your real code that reads in all the training data, and tries every single attribute to build the decision tree classifier routine. This will use recursive calls as outlined in the lecture notes.
      It might print the decision tree out directly, or build a tree structure and then print it out.
      It should not generate more then 8 "if statements". Use fewer if you decide.

   h. **Run your training program on the training data.**
      Iterate on this several times.

   i. **When the validation data is published,** run your classifier on the validation data to create **HW_NN_LastName_FirstName_MyClassifications.csv.** These are your results to hand in.

   j. **Write your write up.**
      Answer the following questions…

3. **Questions to answer in your write up. Copy the questions into the write up.**
   a. What stopping criteria did you use?

   b. Did you use any pruning or post-pruning?

   c. What splitting decision were you using?

   d. What structure did your final decision tree classifier have?
   What was the if-else tree you got? (Copy it into your write-up for completeness.)

   e. Run the original training data back through your classifier.
   What was the accuracy of your resulting classifier, on the training data?

   f. Did your program actually create the classifier program, or did it just generate the attribute list and thresholds for you to hand-code in.

   g. What else did you learn along the way here?

   h. What can you conclude?

4. **Accuracy:**
   Classification accuracy (TP+TN) of test data is counted toward part of your grade.
   (10%).