CS540 – Assignment 1

CS540 – Fall 2015 – Sections 2 & 4
University of Wisconsin, Madison

Submission and Deadlines:

You will submit two separate files through Moodle for this assignment: (1) a .pdf file with your written answers to the theory questions, and (2) a .java file with your code implementation for the practical problems. Both of these files shall be submitted no later than **Noon on Thursday, September 24th** to receive full credit. Late submissions will be penalized 10% per 24 hours that they are received late. After five days (120 hours), late submissions will no longer be accepted for credit.

Collaboration Policy:

This entire assignment shall be completed individually. While you are encouraged to discuss relevant concepts and algorithms with classmates and TAs, you shall only use different examples and problems from those assigned below. Academic misconduct on this assignment includes, but is not limited to: sharing, copying, and failing to protect the privacy of your answers and code fragments with respect to the assignment described below.

Decision Trees (Theory)

0. For this portion of the assignment, you will build and evaluate a decision tree that learns to predict whether a Kickstarter Campaign (on http://kickstarter.com) will succeed in funding. Kickstarter is a crowd funding site where campaign creators raise money to fund a creative project.

   The example data that you are learning from includes four attributes per campaign in addition to whether that campaign was successful in reaching its funding goal: the Goal Amount: [E]xpensive, or [C]heap, the Project Category: [G]ame, [T]echnology, [A]rt, the Campaign Duration: [S]hort, [L]ong, and the Social Media Presence: [F]amous, [P]opular, [U]nknown.

	Goal	Category	Duration	Social Media	Successful
Example 1	C	G	S	P	+
Example 2	E	G	S	P	-
Example 3	E	T	S	P	-
Example 4	E	T	L	U	+
Example 5	C	A	L	U	-
Example 6	E	T	S	F	+
Example 7	C	A	L	F	+
Example 8	E	A	L	U	-

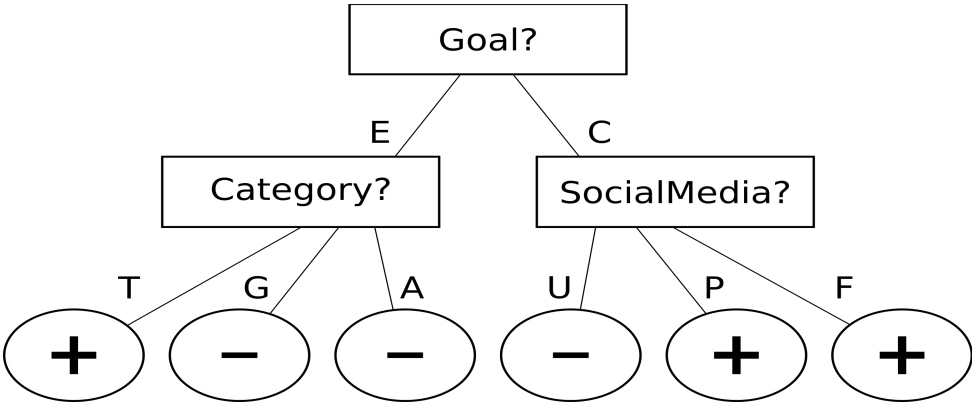
1. What is the Entropy of this initial set of eight examples, with respect to whether they successfully funded or not?

2. Based on this example data, calculate the Information Gain associated with splitting these eight examples by each of the four attributes (one at a time)?

3. Based on these calculations, which attribute should your decision tree split on first (as its root)?

4. Use this example data to build out the entire decision tree, following the algorithm in Figure 18.5 of your text, and using Information Gain as your measure of IMPORTANCE.

5. Suppose that one of your competitors built the following decision tree using an unknown method, and that you would like to test the accuracy of this tree using your own data (above). What percentage of your own examples are accurately predicted by this tree?



6. Using our original example data and the alternate decision tree provided in the last problem, we'd like to test the significance of the SocialMedia? branch. What is the total deviation Δ of this branch in relation to the null hypothesis?

7. Look-up this deviation in a chi-squared distribution table. Is this deviation statistically unlikely (<= 5% probability), or should it be pruned?

Java, Processing, and XML (Practical)

0. For this portion of the assignment, you will be writing a graphical application in Java. You will also get practice with the Processing library, which we will use throughout the course for drawing 2d shapes, and for loading and traversing trees of xml data.

   Download and install Processing from here: https://processing.org/download/ and then follow these instructions to link and get started using Eclipse: https://processing.org/tutorials/eclipse/ (running your program as a Java Application, instead of just an Applet). Play around with drawing a few rectangles: https://processing.org/reference/rect_.html, and also check out the XML documentation here: https://processing.org/reference/XML.html.

Once you are familiar with the structure of a simple Processing application, download *CSLOGIN_Assmt01.java* and replace CSLOGIN with your cs username: both the filename and the class name inside this file. Open this file in your IDE and link the *core.jar* library. You'll also need to ensure that there is a copy of *boxData.xml* in your working directory before you begin. Ensure that you can build and run this skeleton code before you start implementing any of the changes described in the following steps.

1. Look through the *boxData.xml* file to familiarize yourself with the its structure, including the <move> and <box> tags. Implement the provided method stub for:
   ```
   public XML loadBoxes(String filename) { }
   ```
   This method should read the xml data from a file named *filename* in from the working directory, and then return the root node of this tree as an instance of Processing's XML class type. This tree should not contain any of the extra whitespace nodes that Processing's loadXML() method preserves by default.

   To remove these extra whitespace nodes, you should traverse the xml tree searching for nodes with the name "#text" in which the content that is entirely whitespace. You can remove such nodes from the tree by removing these nodes from their parents' list of children.

2. The next stub to implement is for the recursive method:
   ```
   public void drawBoxes(XML xml, int x, int y) { }
   ```
   As this method recursively visits nodes in the xml tree, it will maintain a position (x,y) that the next box should be drawn at, or than the next move adjustment should be made relative to. For example, if a drawBoxes() call is passed the position (2,3) and a move node with an x attribute of 4, then all children of xml should receive the position (6,3) as they are visited. Whenever you encounter a box node through this traversal, you should draw a 20x20 box that is centered at the current (x,y) position. When the *boxData.xml* is drawn in this way, you should see a smiley face.

3. The next stub to implement is for the method:
   ```
   public void doubleMoves(XML xml) { }
   ```
   This method should double the x and y attributes of every move node in the xml tree.

4. And the final stub to implement is for the method:
   ```
   public void doubleBoxes(XML xml) { }
   ```
   This method should replace every box node in the tree with four move nodes that each have a single box node as a child. The four move nodes should position their child boxes +/- 10 units from the original box's position. Be careful to avoid further transforming the newly added boxes. Here is a diagram illustrating the transformation that this method should complete: