Machine Learning (CS 419/519)
Assignment 01 (50 points; 5 bonus/graduate)
Due 5:00 PM, Wednesday, 03 October 2018

In this assignment, you will implement the Decision Tree learning algorithm. You will use a data-set that includes mushroom records drawn from the *Audubon Society Field Guide to North American Mushrooms* (1981). The database describes samples from 23 species of gilled mushrooms in the families *Agaricus* and *Lepiota*. Each sample is described by a string of 23 characters, on a single line of the provided file `mushroom_data.txt`; each such string describes the values of 22 attributes for each sample (described below), and the last character corresponds to the correct classification of the mushroom into either edible (`e`) or poisonous (`p`) mushrooms. For example, the first two samples from the data-set are a poisonous and then an edible species, as follows:

```
x s n t p f c n k e e s s w w p w o p k s u p
x s y t a f c b k e c s s w w p w o p n n g e
```

The 22 attribute variables, and their values, are given in Table 1, at the end of this document. (and are also tabulated in the file `properties.txt`, for reference).
For full points, your program will work as follows:

1. (*5 pts.*) The program will execute from the command line, and handle all input/output using standard system IO (in Java, e.g., this is `System.in` and `System.out`, and in C++ you would use `cin` and `cout`, etc.).

   When the program begins, it should ask the user for two inputs:

   - A training set ***size***: this should be some integer value S that is a multiple of 250, within bounds $250 \leq S \leq 1000$.
   - A training set ***increment***: this should be some integer value $I \in \{10, 25, 50\}$ (that is, one of those three values only).

   Your program should indicate what sorts of values it expects, and validate the data; if the user does not enter an integer in the proper range, then the program should terminate with appropriate, meaningful in-program messages. (It should not simply crash, throw exception messages, or terminate with no explanation, for instance.)

   Note: The assignment folder contains a sample runs of the program so you can see what is expected in this step and later steps; your output should be much the same, if not identical.
2. (*40 pts.*) Once the user has made the necessary choices, the program will proceed automatically without any more user intervention. It will load the file describing the data-set (and the file of attributes and values, if necessary for your implementation), and then:
   (a) The program will choose a training set of the size S chosen by the user. It will do this by choosing S distinct examples at random from the entire data-set. These are removed from the data-set so that any testing of the tree will not be performed on the data instances used to train.
   (b) It will choose an increment I from the training set and build a tree using those I elements. For example, if the user chooses $S = 250$ and $I = 25$, it will first choose 250 examples

from the overall data and set them aside; it will then take 25 elements of *those*, and use them to build the tree.

(c) The tree will be built using the algorithm specified in Russell and Norvig, Figure 18.5, page 702; the choice of nodes for expansion in the IMPORTANCE() function will be done using the information gain heuristic covered in section 18.3.4, and covered in class.

(d) After building the tree, it will run through all the data in the original data-set (after the training set has been removed), and check the accuracy of the tree on that data. This will be reported in terms of the percentage of examples for which the tree is correct (to 4 decimal places, as seen in the sample output).

(e) The algorithm will repeat steps (b)–(d), increasing the increment each time on the size of the set used to build the tree. Again, if the user had chosen $S = 250$ and $I = 25$, this will mean the algorithm will repeat the tree-building and testing process 10 times, using 25, 50, 75, ..., 250 items of the available training data. (For any legal values chosen, the final iteration will use all of the elements of the training set.)

(f) Once the algorithm has finished generating and testing trees, it should summarize the testing results at the end of its print-out.

Again, see the sample runs included with this document to see the basic format expected. Those sample runs contain one where the user chose the values $S = 250$, $I = 10$ and one where they chose $S = 1000$, $I = 50$.*

3. (*5 pts.*) After completing and bug-checking your program, run it twice with different sets of parameters, and save the results to two separate text-files. Produce one graph for each run, using whatever software you like, showing the training set size as x-coordinate and the percentage correct on testing as y-coordinate. Image files of the two graphs should be saved in some standard image format (PDF, PNG, etc.); two examples have been included with this document, giving graphs for the two test-runs also included.

4. (*5 pts.*:) ***Required*** of students in a graduate (519) section; ***optional*** for other (419) students. Your program should provide the option to be run in *verbose* mode (via a command-line argument or the like). If this option is chosen, then after building the final tree, using all training data, but before the summary results, the program should print out the structure of the final tree that it built. This print-out should contain information about the attributes used for splitting at each internal node (referenced either by name of attribute or number in the attribute table), along with the value of the attribute used on each branch (referenced either by single-letter code or full name). Leaves of the tree should be labeled `Edible` or `Poison`, and the branching structure should be evident. A sample of what this might look like is included in the second of the two sample runs. Other formats, if readable, are permissible.

*In the latter case, there is some extra output; this is covered in the bonus/graduate portion of the assignment, described below. If you do not do that portion of the assignment, then you can ignore that part of that output file; the remainder of it is what is expected from any solution to the assignment, in any case.

You will hand in a single compressed archive containing the following:

1. Complete source-code for your program. This can be written in any language you choose, but should follow proper software engineering principles for said language, including meaningful comments and appropriate code style. Code should be in its own folder within the archive, to keep it organized and separate from the rest of the submission.

2. A `README` file that provides instructions for how the program can be run. It is expected that this will consist of the command-line instructions necessarily. If you have completed the final part of the assignment (the optional verbose mode), then your instructions should include the commands needed to activate that mode and see the tree print-out at the end.

3. Two sample runs, stored in text form, of the program output. Each program run should use different parameters for both S and I. If you do the verbose mode portion of the assignment, then one run should use that mode.

4. Two distinct graphs, one for each of the runs in your text-files. File-names or text/titles within the files should indicate which graph accompanies which run.

A table of attribute values for the mushroom data-set is on the next page.

#	**Property**	**Values**
1.	cap shape:	bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
2.	cap surface:	fibrous=f, grooves=g, scaly=y, smooth=s
3.	cap color:	brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
4.	bruises:	bruises=t, no=f
5.	odor:	almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
6.	gill attachment:	attached=a, descending=d, free=f, notched=n
7.	gill spacing:	close=c, crowded=w, distant=d
8.	gill size:	broad=b, narrow=n
9.	gill color:	black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y
10.	stalk shape:	enlarging=e, tapering=t
11.	stalk root:	bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r
12.	stalk surface above ring:	fibrous=f, scaly=y, silky=k, smooth=s
13.	stalk surface below ring:	fibrous=f, scaly=y, silky=k, smooth=s
14.	stalk color above ring:	brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
15.	stalk color below ring:	brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
16	veil type:	partial=p, universal=u
17.	veil color:	brown=n, orange=o, white=w, yellow=y
18.	ring number:	none=n, one=o, two=t
19.	ring type:	cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
20.	spore print color:	black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
21.	population:	abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
22.	habitat:	grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

Table 1: Mushroom attributes in data-set.