Homework #5

TAs' email: dsata AT csie DOT ntu DOT edu DOT tw

RELEASE DATE: 05/03/2013
DUE DATE: 05/16/2013, noon

As directed below, you need to submit your code to the designated place on the course website.

Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

Both English and Traditional Chinese are allowed for writing any part of your homework (if the compiler recognizes Traditional Chinese, of course). We do not accept any other languages. As for coding, either C or C++ or a mixture of them is allowed.

This homework set comes with 200 points and 40 bonus points. In general, every homework set of ours would come with a full credit of 200 points.

5.1 Heaps and Hash Tables

(1) (20%) Describe an efficient algorithm (with pseudo-code or C/C++) that changes the priority (key) of an arbitrary element in a max-heap. Briefly justify the time complexity of the algorithm.

(2) (20%) Hash function is everywhere. Use any search engine to study the "Bloom filter" and how the hash function is used within. Explain the data structure briefly to the grading TA in your own words. Also, cite the website that you learn the algorithm from.

(3) (20%) Do Exercise R-9.15 of the textbook.

(4) (15%) Do Exercise C-9.9(a) of the textbook.

(5) (15%) Do Exercise C-9.9(b) of the textbook.

(6) (Bonus 20%) Construct a perfect hash function that is efficiently computable for the following 32 programming/scripting languages. You need to explain why the hash function is perfect and why it is efficiently computable to get the full bonus.

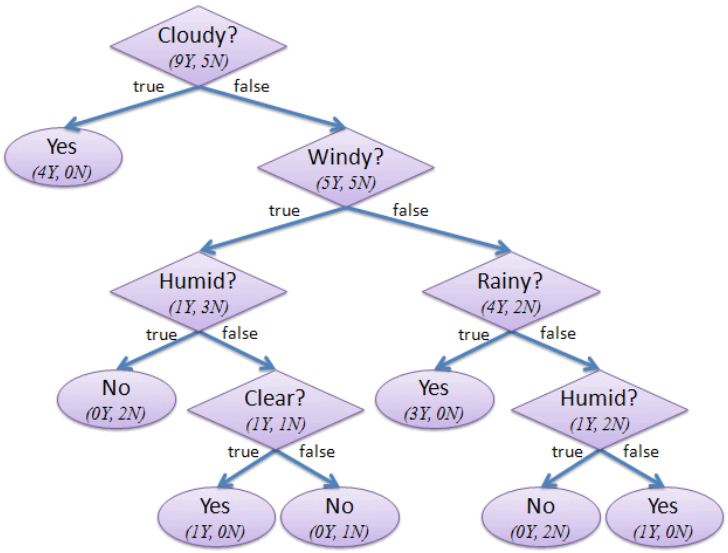
`C`, `Java`, `C++`, `Objective-C`, `C#`, `PHP`, `Visual Basic`, `Python`, `Perl`, `Ruby`, `JavaScript`, `Lisp`, `Pascal`, `Haskell`, `Scala`, `Fortran`, `Prolog`, `Assembly`, `Verilog`, `Erlang`, `MATLAB`, `Bash`, `SmallTalk`, `Caml`, `Scheme`, `Go`, `Ada`, `Cobol`, `Awk`, `Tcl/Tk`, `Delphi`, `Limbo`

5.2 Decision Tree

In this problem, we will explore an application of trees in the area of Artificial Intelligence and Machine Learning. Decision Tree is one of the earliest tool for Machine Learning. The non-leaf nodes of a decision tree represent choices (factors) considered, while the leaf nodes represent final decisions under the choices.

For simplicity, we will consider the trees with binary factors—i.e., binary decision trees. Such a tree is shown in Example 7.8 of the textbook.

For instance, the following tree[1] is a binary decision tree for deciding whether to play golf. If the sky is cloudy, we decide to play golf; if not, we check if it is windy—if so, we play golf only if it is not humid and the sky is clear. On the other hand, if it is not windy, we do not play golf only when the sky is not rainy but it is humid.



Such a decision tree is called a "classification tree." It classifies different $(sky, windy, humid)$ situations to decision categories $play? = \{yes, no\}$. The tree is not arbitrarily formed. In fact, it is automatically learned by a program from a bunch of given examples. In other words, you can "teach" the program with the examples. For instance, consider the following examples.

sky	windy	humid	play?
clear	true	true	no
clear	true	false	yes
rainy	true	false	no
rainy	false	true	yes
clear	false	true	no
clear	false	false	yes
cloudy	false	true	yes
clear	false	true	no
cloudy	true	false	yes
cloudy	true	true	yes
rainy	false	true	yes
cloudy	false	true	yes
rainy	true	true	no
rainy	false	true	yes

The decision tree can be taught with the examples in a top-down recursive way. First, we need to find the root branch. There are 9 yes and 5 no (9Y5N) in the examples above. If we consider a factor of "sky is clear or not", we can separate the 14 examples to two branches: 2Y3N for the clear branch and 7Y2N otherwise; if we consider a factor of "sky is cloudy or not", we can separate the 14 examples to two branches: 4Y0N for the cloudy branch and 5Y5N otherwise. We can continue checking possible branches.

One heuristic for making a good branching choice is to check the total confusion after branching. The confusion of a mixture of aYbN is defined as

$$confusion(a, b) = 1 - \left(\frac{a}{a+b}\right)^2 - \left(\frac{b}{a+b}\right)^2.$$

[1] Thanks to our previous TA Chun-Sung Ferng for drawing.

and the total confusion after branching from $(c+e)$Y$(d+f)$N to cYdN and eYfN is

$$total(c,d,e,f) = \frac{c+d}{c+d+e+f} confusion(c,d) + \frac{e+f}{c+d+e+f} confusion(e,f).$$

For instance, the total confusion after branching by "sky is clear or not" is

$$\frac{5}{14}\left(1-\left(\frac{2}{5}\right)^2-\left(\frac{3}{5}\right)^2\right)+\frac{9}{14}\left(1-\left(\frac{7}{9}\right)^2-\left(\frac{2}{9}\right)^2\right).$$

The heuristic tries to find a branch such that the total confusion is the smallest, with ties arbitrarily broken.

Now, after finding a good branch for the root, we separate the examples to two subsets: one for the left-child of the root, one for the right-child of the root. The same branching strategy can be applied to the two subsets to form the two sub-trees and the tree building continues recursively.

Recursively? What is the termination condition, then? Well, you do not need to branch if there is no confusion left—that is, when the examples considered belong to the same final decision like aY0N or 0YbN. In such a case we declare a leaf with the final decision. Also, if we have too few "examples" in the subset, a branching would be overly-explaining the data. So if the number of examples is no larger than a certain criterion θ, we can stop branching. For instance, if $\theta = 5$, then we would stop branching with 3Y1N in the subset. In that case, the decision of the subset is simply the majority vote (in this case, "Y"), with ties arbitrarily broken. The simple decision tree algorithm is listed as follows:

```
Decision-Tree(examples)
if no confusion in the examples or # examples ≤ θ then
    build and return a leaf node with the majority vote of the examples as the final decision
else
    find a branch such that the total confusion is smallest, store the branch in the root of the tree
    separate examples to two subsets, one for the left-child and one for the right-child
    set the left-subtree to be DecisionTree(example subset for the left-child)
    set the right-subtree to be DecisionTree(example subset for the right-child)
    return the tree
end if
```

The branch we discussed are on discrete factors. We can also branch on numerical (continuous) factors by setting a proper threshold. For instance, a numerical factor may be the *temperature* and a branch may be "is *temperature* greater than t?" The best threshold t can be found by "cleverly" searching all possible thresholds.

Consider branching on one numerical variable with M examples. Trivially, there are only $O(M)$ possible "regions" of thresholds t, where all the thresholds within the same region are equivalent—that is, they separate the M examples to two subsets in identical ways. For the $O(M)$ regions, consider choosing the median point of each region as the branching threshold. An intuitive way of evaluating the total confusion of a branch may take $O(M)$ per evaluation, and using that to pick the best branch would take $O(M^2)$.

In this problem, we ask you to implement such a program that can be taught with examples of numerical variables and produces the binary decision tree. One interesting thing about binary decision trees is that you can output the tree as some C code `if(...){...} else{...}`. That is, after you teach your program, it can automatically produce another program that can make future decisions.

(1) (Bonus 20%) Describe an $O(M \log M)$ algorithm for picking the best branching threshold from one numerical variable with M examples. (*Hint: Sorting can be done in $O(M \log M)$, and what next?*)

(2) (30%) Implement the decision tree algorithm with either the slow or the fast way of deciding the branching threshold. Your program should read the input examples that contain numerical values (format to be announced online), and print out a piece of C code representing the decision tree (format to be announced online).

(3) (20%) Illustrate the internal data structure you use to represent the decision tree. Please be as precise as possible.

(4) (20%) Teach your decision tree with the following examples to learn a function f. Draw the tree you get.

weight	height	age	f(weight, height, age)
68	154	32	false
74	165	22	true
80	187	36	false
83	173	18	true
69	152	28	false
52	144	24	true
43	180	33	true
57	177	23	true

(5) (20%) Construct your own data set with at least 2 numerical factors and at least 6 examples. Teach your program to make a decision tree of at least 2 levels with this data set. List the examples as well as draw the tree found. Briefly explain the tree.

(6) (20%) Let's set $\theta = 1$. If, instead of branching by minimizing the total confusion, you do a "random branching" by randomly pick one factor and branch with it. Due to randomness you may end up revisiting some factors you have considered in the top levels. You can also do "maximum total confusion branching" by branching with the maximum confusion factor instead of the minimum confusion one. Using the data set in Problem 5.2(4), compare the depth of the tree you get from "minimum total confusion branching" to the average depth of the trees you get from "random branching" (over 1000 random runs) to the depth of the tree you get from "maximum total confusion branching." Briefly state your findings.

Submission File (Program) and Written Copy

Please upload your program as a single ZIP compressed file to CEIBA before the deadline. The zip file should be like `b86506054.zip`, where the file name should be changed to your own school ID. The ZIP file should contain the following items:

- code such that `make tree` would generate a program named `tree`, which reads examples and outputs a piece of code for the decision tree according the format that will be announced online
- an optional `README`, anything you want the TAs to read before grading your code

For all the problems that require illustrations, please submit a written (or printed) copy in class or to CSIE R217 before the deadline.

MEDAL USAGE: If you want to use the gold medals for this homework, please write down the number of medals that you want on the first page of your printed copy (something like "use 2 medals"). Use your medals wisely—usage cannot be retracted.