Homework #4
RELEASE DATE: 04/22/2014
DUE DATE: 05/06/2014, **17:30** (after class) in CSIE R217

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 20 bonus points. In general, every homework set of ours would come with a full credit of 200 points.

4.1 Trees

(1) (20%) Do Exercise R-7.15 of the textbook.

(2) (20%) Describe an algorithm (with pseudo-code) that solves the problem in Exercise R-7.15 for arbitrary (correct) pairs of preorder/inorder traversal results.

(3) (20%) Do Exercise C-7.33 of the textbook.

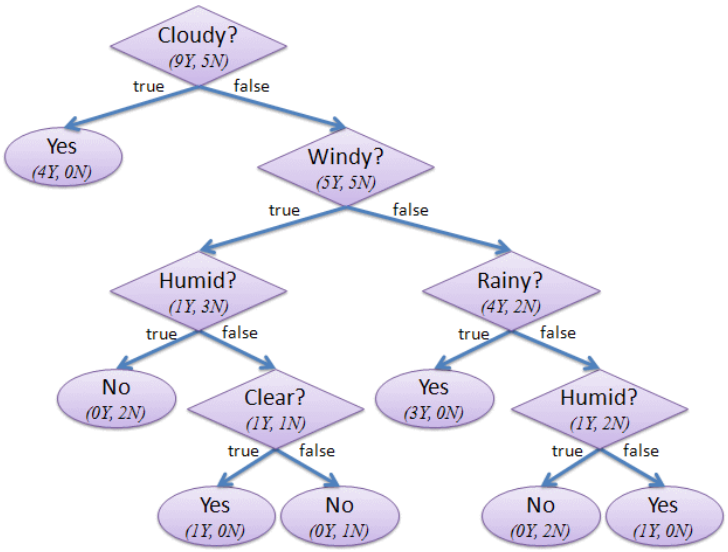
(4) (20%) Do Exercise C-7.34 of the textbook.

4.2 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.

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



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 being 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 being 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

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

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 total confusion after branching by "sky is cloudy or not" is

$$\frac{4}{14}\left(1-\left(\frac{4}{4}\right)^2-\left(\frac{0}{4}\right)^2\right)+\frac{10}{14}\left(1-\left(\frac{5}{10}\right)^2-\left(\frac{5}{10}\right)^2\right).$$

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, or if all the examples are the same and you cannot branch anymore. In such a case we declare a leaf with the final decision with the majority of the examples, with ties arbitrarily broken. The simple decision tree algorithm is listed as follows:

```
Decision-Tree(examples)
if no confusion in the examples or cannot branch anymore then
  build and return a leaf node with the associated 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.

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%) 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. Assuming that sorting can be done in $O(M \log M)$ (this will be taught later), describe an $O(M \log M)$ time algorithm for picking the best branching threshold and briefly justify that the algorithm achieves such a time complexity. (Note: 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 $O(M^2)$. Please think about how you can be faster than that.)

(2) (30%) Implement the decision tree algorithm. Your program should read the input examples that contain numerical values, and print out a piece of C code representing the decision tree. Your program should be named `tree`, and takes the data file as the first argument. For instance, `./tree heart` learns a decision tree from the data file `heart`.

The data file is assumed be of the famous LIBSVM sparse format.[2] Each line of the data file represents an example of the form

```
label sparse_array
```

where `label` is either `+1` (indicating YES) or `-1` (indicating NO), and `sparse_array` represents an array {`4, 0, 0, 3, 2`} by `1:4 4:3 5:2`

[2]LIBSVM is a world-famous machine learning algorithm developed by Prof. Chih-Jen Lin's lab in our department

You can read the sparse array and store it in whatever way (sparse or dense) you choose in your program. A friendly hint is that it might be simpler to internally use the dense array.

Please output your tree as a function in C/C++ language. The function must follow this interface:

```
int tree_predict(double *attr);
```

The only argument is a double array which contains the factors of one example in the same format as input. This function should return the label prediction of the example (1 or -1 for `heart`, for instance). Also, please name your output file as `tree_pred.h`. Then, you can compile and run the provided `tree_predictor.cpp` to check how good your decision tree is (see README). For example, your `tree_pred.h` should look like:

```
int tree_predict(double *attr){
  if(attr[0] > 5){
    return 1;
  }
  else{
    return -1;
  }
}
```

(3) (15%) Illustrate (ideally with drawing) the internal data structure you use to represent the decision tree in your memory. Please be as precise as possible.

(4) (15%) 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	153	32	-1 (NO)
74	167	22	+1 (YES)
90	182	26	-1 (NO)
83	179	18	+1 (YES)
59	152	28	-1 (NO)
52	144	24	+1 (YES)
43	170	33	+1 (YES)
47	171	23	+1 (YES)

(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) (30%) A popular algorithm in Machine Learning is called *Random Forest*. The random forest contains T decision trees. We will consider a simple case. Given N examples with F factors, each decision tree is learned from $\lfloor \frac{N}{2} \rfloor$ of randomly chosen examples and 6 different and randomly chosen factors. For making a decision, the random forest asks each tree to provide a decision (vote), and then takes the majority vote as the final decision, with ties arbitrarily broken. Implement the random forest algorithm. Your program should read the input examples that contain numerical values and T as input. Then, it prints out a piece of C code representing the random forest. Your program should be named `forest`, takes the data file as the first argument, and T as the second argument. For instance, `./forest heart.train 50` learns a 50-tree forest from the data file `heart.train`.

Similar with the decision tree function, you need to output your forest as a function in C/C++ language. The function must follow this prototype:

```
int forest_predict(double *attr);
```

The argument and return value specs are the same as the decision tree function. Also, please name your output file as `forest_pred.h`. Then, you can compile and run the provided `forest_predictor.cpp` to check how good your Random Forest is (see README). For example, your `forest_pred.h` should looks like:

```
int forest_predict(double *attr){
tree1_predict:
//do something
tree2_predict:
//do something
treeT_predict:
//do something
voting:
//do something
}
```

Submission File (Program) and Written Copy

Please push your program to your repository `<user_name>/dsa14hw4` (on GitHub) before the deadline at 5:30pm on Tuesday (05/06/2014). We will use the latest time that you pushed to the repository as your submission time.

10 of the total points will also depend on how you use *git*, such as whether the commit message is meaningful, whether each commit involves reasonable logical units of the source, etc. Please **DO NOT PUT BINARY FILES** in your repository.

Your repository should contain the following items:

- all the source code and your own `Makefile` (different from the TAs' `Makefile.ta`) such that `make tree` would generate a program named `tree`, which reads examples and outputs a piece of code for the decision tree, and `make forest` would generate a program named `forest`, which reads examples and output a piece of code for the random forest
- 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 visit `http://main.learner.csie.ntu.edu.tw/php/dsa14spring/login_medal.php` and submit the request **before the deadline + 4 days** (5/10).