CSI2110 Assignment 3 (Programming Assignment I ; weight 7.5%)

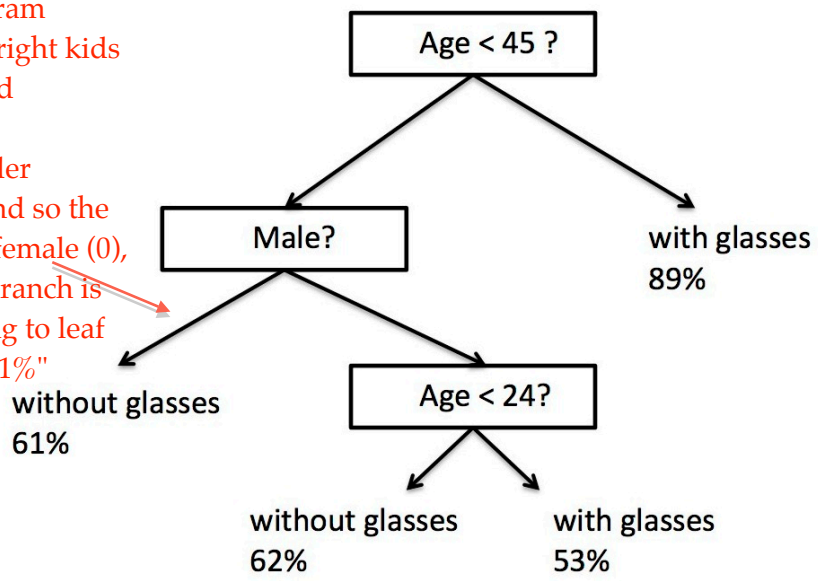
Decision Trees

Fall 2015

due date : November 25, 11 :55PM (1min-24hours late gets -30% late penalty)

In this assignment, you will explore decision trees. A binary decision tree is a full binary tree to identify the class to which observed data belongs. The following example illustrates this concept.

Correction:
to match the program example, left and right kids of this node should be switched.
The code has gender threshhold (0.5) and so the smaller branch is female (0), while the higher branch is for male (1) leading to leaf "without glasses 61%"



This example shows a decision tree to predict whether a given person wears glasses or not by simply using their age and gender. Obviously, using these two pieces of information is very limiting and certainly do not allow us to come to a reliable prediction. The percentages shown are an estimate of the accuracy of these predictions. They have been obtained based on a large number of samples (for which age, gender and presence of glasses were known). By applying the rules of the decision tree on a large number of observations, it is possible to obtain a good estimate of the accuracy of each of the classifications obtained in the leaves of the tree. Indeed, with these pre-classified samples, it is possible to count how many samples of each class reach a given leaf. For example, for the leaf corresponding to the decision [Age <45, Female, Age <24], it was observed that in 62% of cases the sample reaching this leaf was a person without glasses and in 38% of the cases it was a person without glasses. So if an observation to be classified enters the root and ends on this leaf, the class assigned will be 'without glasses' since there were a higher percentage of samples (62%) belonging to this class that arrived at this leaf. Obviously, a classification associated with a high percentage (say 97%) is far more reliable than a classification associated with a lower percentage, such as 62%.

To perform the classification, the decision tree uses input vectors that have a certain size (2 in our example, age and sex). The decision is to determine the membership class; in our example, there are 2 classes (with glasses or without glasses), but in general there may be more.

The decision tree consists of nodes (stumps) that use one of the entries of the decision vector and applies a threshold. Depending on the result of the comparison, we branch left or right. A vector enters the tree by the root. It follows a series of comparisons leading to a leaf of the tree.

The file `DecisionStump.java` provides an implementation for a decision node. Each node specifies an index (`featureIndex`) in the vector and a threshold (`threshold`) to be used in the comparison. The leaves of the tree are dummy nodes without comparison, which return the class with the highest probability for that node. These probabilities are obtained either by specifying them directly or by accumulating the results obtained by using pre-classified samples (`classFreq` and `total`).

The method `main` of this class builds the decision tree shown in the previous page, and tests its results with different inputs.

Part A (marks 50%)

Using the `DecisionStump` class, complete the `DecisionTree` class by adding the following operations :

`public void replace(DecisionStump leaf, int featureIndex, double threshold)` : that replaces a given leaf of the decision tree by a new stump that applies a decision threshold on one element of a feature vector. This operation also creates also dummy leaves. This operation will be used to grow the decision tree.

`public DecisionStump getSmallestMaxProb()`: that returns the leaf having the smallest maximum probability (i.e. the least reliable node). Look at the getMaxClass and getMaxProb methods of DecisionStump. This operation will be used to replace an unreliable leaf by a new decision stump.

`public void print()` : that prints all the nodes of a decision tree by calling the toString of the DecisionStump instances in a pre-order traversal.

`public int getDecision(double[] vector)`: that returns the class associated with a sample of unknown class. This operation will return the class number having the highest probability.

Part B (marks 25%)

File `iris.data.txt` contains a list of samples specifying the length and width of the petals and setals of 3 different species of iris. This is then a file made of features vector of dimension 4 with 3 possible classes.

1. Generate a decision tree made of one stump that uses element at index 0 with a threshold of 5.0.

```
// Create 1-node tree
DecisionTree dt= new DecisionTree(4, 3);
dt.replace(dt.getRoot(), 0, 5.0);
```

2. Call the `train` method of `DecisionTree` with each pre-classified samples in the given file.
3. Print all the nodes and their associated probabilities.
4. Find the leaf having the smallest maximal probability. Replace this one with a stump on index 2 with threshold 2.5. Train this new decision tree with the same dataset :

```
// train a 2-node decision tree
dt.replace(dt.getSmallestMaxProb(), 2, 2.5);
dt.resetAll();
// add training code…
```

5. Print all the nodes and their associated probabilities.
6. Add another node and re-train the tree as follows :

```
// train a 3-node decision tree
dt.replace(dt.getSmallestMaxProb(), 1, 3.0);
dt.resetAll();
// add training code…
```

Part C (marks 25%)

We now ask you to generalize the process described in Part B.

1. Create a 1-node decision tree by adding a stump with randomly chosen index and threshold.
2. Train this tree with the given dataset of pre-classified iris samples.
3. Find the leaf having the smallest maximal probability and replace it with a new decision stump with randomly chosen index and threshold.
4. Repeat steps 2 and 3 up to 50 times.
5. Do you see an improvement in the performance of the classifier ? Justify.
   Print the answer to this question from your code; you can print relevant information about the decision trees to justify your answer.

WHAT TO HAND IN :

- Your program: DecisionTree.java (with required modifications), DecisionStump.java (no need to modify)
- README.txt file (optional, explain any deviations (e.g. extra classes used) and any known bugs).
- Printout.txt: output of your program showing results for part B and C; this is expected to be identical to what we would obtain if we run the program you submit.
- Put the files specified above in a folder called u9999999 (where 9999999 is your student number); zip the folder and submit the zipped folder

DEVIATION TO FOLLOW THESE STANDARDS WILL GET 5-10% MARKS OFF.