CMPUT 466/566 — Programming Exercise #6

Instructor: R Greiner
Due Date: 5:00pm, Friday 15/Mar/2019
This exercise is intended to further your understanding of Decision Trees.

Relevant reading: [HTF: parts of Ch 10]
Everyone should answer Questions#1 – 3; Q#4 is optional, for extra credit.

Total points: 22 (+ 4 for extra credit)
Percentage of total course mark: Ugrad: 8% Grad: 6%

For the following questions, you have been provided with two datasets, some helper functions, and the skeletons of some other functions (that you will need to flesh out).

- `breast_cancer_dataset.mat`: The *breast cancer* dataset contains a `train_set`, `test_set`, and a vector of `attribute` descriptions. There are 21 attributes, each of which can take one of 10 values. The last column of the training and test sets is the class label, which takes the value 0 for `benign` or 1 for `malignant`.
- `planets.mat`: The *planets* dataset contains a `train_set`, `test_set`, and a vector of `attribute` descriptions. There are two attributes, each of which can take one of two values, {0, 1}. The last column of the training and test sets is the class label, which takes the value 0 meaning `the planet is not habitable` or 1 meaning `the planet is habitable`.
- `print_tree.m`: Helper code that prints a decision tree. There is no need to modify this code.
- `small_test.m`: Helper code that you can use to verify your code for `learnDecisionTree.m` and `classify.m`. It will print the decision tree trained on `train_set` in `planets.mat` and classify the first training instance. Feel free to modify this code however you like. It will not be marked.
- `learnDecisionTree.m`: Skeleton of the code that implements the algorithm for learning a decision tree.
- `classify.m`: Skeleton of the code that implements a function that takes a decision tree and an instance, and returns the class label predicted for that instance.
- `overfitting.m`: Helper code for the extra-credit portion of the assignment. Computes the training and test accuracies for decision trees trained to a specified depth. There is no need to modify this code.
- `learnDecisionTree_2.m`: Skeleton of the code that implements the algorithm for learning a depth-limited decision tree, for the extra credit portion of the assignment.

Some comments (reacting to problems with earlier years):

- Follow the instructions and read the comments in the functions!
- When using boolean values, make sure they are indeed boolean and not a string with the value 'true' or 'false'.
- Remember that we define $0 \log 0 = 0$ (by default matlab sets $\log 0 = -\infty$, so you need to handle that).
- Make sure you try out your code ... and test for the boundary conditions (*e.g.*, if no instance, if all instance match, ...)
- Make sure your code runs on Matlab (we will penalize if we need to debug the code to make it compatible with Matlab).

Question 1 *[14 points]* 2pts each for steps 1–4; 6pts for step 5 (see the skeleton code).
In `learnDecisionTree.m`, implement the algorithm from the slide numbered 29 (which is 23/82) of the decision-tree lecture notes (link can be found at the top of `learnDecisionTree.m` file).

The algorithm constructs a decision tree whose attributes are chosen based on minimizing the uncertainty. The algorithm continues until reaching "purity" (running out of all attributes, or all "discriminating" attributes). There is no pruning.

The equation for **Uncertainty** is:

$$\text{Uncert}(\, S,\, A\,) \quad \equiv \quad \sum_{v \in \text{Values}(A)} \frac{|S_{A=v}|}{|S|} \, \text{Entropy}(S_{A=v}) \tag{1}$$

where S is a collection of labeled examples, A is a particular attribute, and

$$S_{A=v} \;=\; \{(x, y) \in S \mid x_A = v\}$$

is the subset of S whose attribute A has the value v.

This uses:

$$\text{Entropy}(\, S\,) \quad \equiv \quad -\sum_{i=1}^{c} p_i \log_2 p_i$$

where there are c different class labels, and $p_i = \frac{|\{(x,y) \in S \;\mid\; y=i\}|}{|S|}$ is the proportion of S belonging to class label i.

Question 2 *[6 points]* 2 pts each for the leaf case and 4 pts for the non-leaf case.
In `classify.m`, implement the code that takes as inputs a decision tree (as outputted by `learnDecisionTree.m`) and a single instance (row) and returns the class label predicted by the decision tree for that instance.

Once you have implemented `learnDecisionTree.m` and `classify.m`, you may wish to do a quick sanity check: Execute the provided file `small_test.m`, which

1. learns a decision tree for `planets.mat`,
2. prints it out using `print_tree.m`, and
3. reports the class of the first instance of the training set, predicted by the decision tree you just learned.

You will not be marked on this, as the expected output is provided in `small_test.m`.

Question 3 *[2 points]*
Prepare a report, `report.txt`, which includes the following items:

- (1 pts) Please copy and paste the output of the following three lines of code, which prints the decision tree for the breast cancer dataset.

```
load breast_cancer_dataset
tree = learnDecisionTree(train_set, attribute, 0);
print_tree(tree)
```

- (1 pts) Please use `classify.m` to classify all the instances in `train_set` and `test_set` in `breast_cancer_dataset.mat`. Report the integer number of correctly classified instances for the training and test sets.

Question 4 *[4 points]* Extra Credit
One way to mitigate overfitting is to limit the depth of the decision tree. In `learnDecisionTree_2.m`, create a modified version of `learnDecisionTree.m` that takes a fourth input argument 'depth' and returns a decision tree with max-depth 'depth'.

Then, execute `overfitting.m` which calls `learnDecisionTree_2.m` with three different depth values and reports the training and test accuracies.

In **report.txt**, copy and paste the output of `overfitting.m`. Indicate which depth is the best and write one sentence to justify your answer.

Delieverables and Submission
Your final submission should be a single zip-file, called **PE_6.zip**, that includes all your files. You should only submit (upload into eClass) a zip-file.