IIT BOMBAY

Lab Assignment #5
Points: 5

Main TAs: *Ashish Agrawal, Ashwin Ramachandran, Mayank Jain*

Course: *CS 335* – Instructor: *Preethi Jyothi*
Due date: *September 4, 2023*

General Instructions

1. Download lab5.tgz from Moodle for CS 335, and extract the file to get `lab5`.
2. The folder `data/` within `lab5` contains all the data files required for this task. Make sure you update the path files in the template code to load these data files properly.
3. `q1_template.py` and `q2_template.py` contain template code for the two main questions in this lab.
4. For your final submission, submit the source files `q1_template.py` and `q2_template.py` with the TODOs filled in. This lab submission is due on Moodle on or before **11.59 pm on Sept 4, 2023**.
5. For your final submission, create [rollnumber].tgz with the following internal directory structure:

```
[rollnumber]/
  |
  +-Q1/
  |   |
  |   +- q1_template.py
  +-Q2/
  |   +- q2_template.py
  |   +- decision_tree.png
  |   +- regularized_decision_tree.png
  |   +- analysis.png
```

6. You will get 3 points if your test accuracy from Q1 matches our solution test accuracy 60.34% (within a tolerance limit). You will get 1 point for submitting the required png files for Q2, and 1 point if the regularized decision tree in Q2 yields a higher test accuracy compared to the unregularized decision tree (from Q1).

Q1: Implement a Decision Tree Classifier from Scratch

In this problem, you will implement a decision tree classifier from scratch without using any external libraries. Decision tree classifiers work by splitting the data into subsets using a splitting criterion (e.g., information gain) until each subset is mostly homogenous in a label. They offer interpretability and can work with both categorical and numerical data. The problem here is a binary classification problem based on breast cancer data where a particular data instance has to be classified as a cancer-recurrence-event or no-cancer-recurrence-event.

The **ID3** decision tree algorithm is as follows:

`ID3(Examples, Target_attribute, Attr_list):`
Input: `Examples` are training examples.
Input: `Target_attribute` is the class to be predicted by the tree.
Input: `Attr_list` is a list of attributes.
Output: Returns a trained decision tree.

```
Create a Root node for the tree
If all examples are +, return a single node Root with label +
If all examples are −, return a single node Root with label −
If Attr_list is empty, return a single node Root with label = most common value
   of Target_attribute in Examples
begin
    A ← the attribute from Attr_list with highest information gain
    for each possible value v of A do
        Add a new tree branch below Root, corresponding to A = v_i
        Let Examples_v be the subset of Examples with value v for A
        if Examples_v is empty then
            Below this new branch, add a leaf node with label = most common
   value of Target_attribute in Examples
        else
            Below this new branch, add the following subtree:
            ID3(Examples_v, Target_attribute, Attr_list-{A})
        end if
    end for
end
return Root
```

Q1: Implement a Decision Tree Classifier from Scratch (contd.)

Complete the following functions in `q1_template.py`:

- `entropy()`: Compute the entropy of a random variable. You can use `for` loops.
- `information_gain()`: Calculate the information gain given a dataset and an attribute. Recall from class, for a dataset S and attribute a, information gain can be written as:

$$Gain(S, a) = H(S) - \sum_{v \in \text{values}(a)} \frac{|S_v|}{|S|} H(S_v)$$

  where $H(S)$ is the entropy of the label distribution in S and S_v denotes the subset of S whose instances all have attribute a taking the value v.
- `construct_ID()` inside the class `IDTree`: This is the main function that recursively constructs an ID3 decision tree. Base cases have already been implemented. Implement the rest of the code. Comments are available in `q1_template.py` to guide you further.

Q2: `sklearn` for Decision Tree Classifiers

In this problem, you will use the sklearn library to train a decision tree classifier on the same data as above. This library will give you more control over the different hyperparameters via which the learned decision trees can be regularized (e.g., `min_samples_split`.

Complete the following functions in `q2_template.py`:

1. `train_model()`: Invoke the decision tree classifier from the `sklearn` library. Data files are within the `data` folder of the current directory. The function should return the train and test accuracies in the form of a dictionary. Also, the data to be passed to the function for model training should be the dataframe that is already available in `q2_templatey.py`; do not pass data in a numpy array or use any other format.

2. `train_model_regularized()`: In this function, learn a decision tree on the training data using different hyperparameters (for regularisation) to improve test accuracy. Examples of potentially useful hyperparameters appear in `q2_template.py`.

3. `plot_tree()`: Draw the learned trees from `train_model()` and `train_model_regularized()` and save them as `decision_tree.png` and `regularized_decision_tree.png`, respectively.

4. `train_diff_sizes()`: In this function, train the decision tree classifier using `train_model_regularized()` on different amounts of training data; the training ratios are already specified in `q2_template.py`. Plot the test accuracies as a function of varying train size and save the plot in `analysis.png`.