Coursework

Learning objective: Assess understanding of decision tree learning and pruning algorithms

You can complete the work in pairs but must make your own submission.

The code provided in `COMP2611_DT.py` is an example of a decision tree learning algorithm adapted from code provided with the course textbook. The code provides methods for defining a decision tree by hand, or learning a decision from data.
It also provides code for generating data from a known decision tree.

The tasks below require you to adapt the code provided in the COMP2611_DT.py file. The code has been commented to identify where the code should be modified for each task. For all the tasks you **must not** adapt any of the code apart from where indicated. In particular ensure you do not change the names of any of the methods provided, the variables they return or import any extra packages.

Your solution should be submitted via Gradescope. Please only submit your COMP2611_DT.py file and ensure its name is unchanged. When you submit your code it will be tested by the autograder using the files and data you have been provided with to ensure it returns the correct types/values etc.
For grading your solution will be tested on a different set of files and datasets.

Task 1 (Learning a tree from data)

For task 1 complete the learn_tennis_tree function (tasks are marked in the code)

Use the dataset in `tennis.csv` to learn a decision tree to predict the value of variable *Play* from all the other attributes provided in the file

To complete this you must create a Dataset object using the csv file.
You will see the parameter `filename` has been set in the main method
`filename = "./tennis"`

In the `learn_tennis_tree` method you can create the data set using the DataSet constructor e.g. **(1 mark)**

```
mydataset = DataSet(name=filename, target='Play')
```

You can then learn the decision tree using the dataset e.g.

```
mytree = DecisionTreeLearner(mydataset)
```

(1 mark)

If you wish to print the tree you can use
`print(mytree)` or if you prefer a more readable form to help you understand the tree you can temporarily use `mytree.display()`

Task 2 (Testing the tree)

For this task you must complete the `test_tennis_tree` function.

To evaluate the performance of the tree you can calculate the error ratio obtained using:

```
error = err_ratio(mytree, mydataset)
```

If the leaner has used the same data to learn the model as it has used to test it the error ratio should be zero! You should therefore wherever possible train and test your model on different data sets.

In this code you can split your data into a training and test set using

```
train,test = train_test_split(mydataset, test_split=0.5)
```

where the value of testSplit is the fraction of the dataset that will be used for **testing** and the rest is used for training

Adapt the `test_tennis_tree` function to train a model using 80% of the training data from `filename` and calculate the error ratio on the remaining 20%.

Ensure the function returns the training set, the test set, the learned decision tree and the error rate achieved.

(4 marks)

Task 3 (Generating data)

We can also generate data from an existing tree. You will see in the code there is a tree already defined that you may have seen parts of in lectures.

To generate 200 data examples from this tree you call the SyntheticRestaurant method

```
restaurant = SyntheticRestaurant(200)
```

You can generate datasets of different sizes and try and learn the tree from the examples

(Task 3a) Complete the `genSyntheticTrainSet` function to create a dataset of **200** examples using the `SyntheticRestaurant` method**.** **(1 mark)**

(Task 3b) Complete the `genSyntheticTestSe`t function to create a dataset a dataset of **100** examples using the `SyntheticRestaurantTest` method. **(1 mark)**

(Task 3c) Complete the `train_restaurant_tree` function to learn decision trees using different sizes of the training set you created in 3a from size 1 to N in increments of 1.
The function should also evaluate the performance of each tree as you did in task 2, using the error ratio against the whole test set you generated in task 3b. By displaying the error rates you can see what happens as the training set size increases.
The function should return the final tree obtained using all 200 samples, and the minimum size of the training set required (`samples_required`) to achieve the same error rate as is achieved using all 200 training samples. **(3 marks)**

(Task 3d) Complete the `train_tree`function to learn a decision tree using the whole training set you created in 3a passed as the parameter trainSet. The function should also

evaluate the performance of the tree using the error ratio against the whole test set you generated in task 3b (passed as parameter testSet).
The method should return the tree, and the error rate obtained. **(2 marks)**

Task 4 (Pruning the tree)

In order to improve the error rate of the tree generated in task 3d on test data we can prune the learnt tree to remove nodes that may be created due to overfitting.

(Task 4a) Complete the function `genPruneTestSet` to create a dataset of **1000** examples using the method `SyntheticRestaurantPruneTest`.

(Task 4b) To complete task 4 you will need to adapt three functions
`prune_tree, deviation` and `evaluate.`

`prune_tree` takes two parameters, the tree you wish to prune (result of task 3d) and the data set you wish to use to evaluate the tree (task 4a).
The function should call the evaluate method to prune the tree if necessary (p_value>0.05).
The evaluate function will only prune one node at a time and therefore it may be necessary to call evaluate several times until no further pruning is necessary e.g.

```
p_value,delta,error_rate = evaluate(tree,testSet)

clear_counts(tree)
p_value,delta,error_rate = evaluate(tree,testSet)
```

Between each evaluation you need to clear the counts of positive and negative examples at each node using the `clear_counts` function.

The `evaluate` function takes two parameters, the tree you wish to prune (predict) and the dataset you will use to evaluate the tree (testSet, created in task 4a).
It returns the p value, the value (delta) used to calculate the obtained p value and the error_rate produce by the pruned tree.

(Task 4c) In order to evaluate the tree you need to complete the `deviation` function. This calculates the difference between the actual counts and the expected counts at each node, these are then summed in the evaluation function to calculate delta (Δ in your slides).

deviation=(actual positives – expected positives)^2/expected positives +
(actual negatives – expected negatives)^2/expected negatives

Hint: Reference to our slides, expected positives in a child node is calculated as follows:

Expected positives =
parent positives*(child positives +child negatives)/ (parent positives +parent negatives)

It gives you the number of positives that will go to that child based on probability in an ideal situation and it does not need to be necessarily a whole number.

(Task 4d) Finally add a line to the `evaluate` function to calculate the p value for the value of DELTA. This requires the of the stats.chi2.cdf function. The degree of freedom (number of variables) is related to the number of branches at the parent.

(7 marks)
[Total 20 marks]