COMS 4701 – Artificial Intelligence – Fall 2014

Assignment 4: Decision Tree Learning

(version 7, Nov 11, 2014)

Due: Friday, December 5, 2014, 11:59 PM

The purpose of the assignment is to implement a Machine Learning algorithm in Python. The algorithm you will be implementing is the Decision Tree algorithm with pruning. Besides, you will write a program that outputs another program.

A Python Version of the Decision Tree algorithm from the AIMA book is available in ~cs4701/Project4/sample_code/learning.py. This code from the AIMA book is also your starting code for this assignment. Thus please read the code carefully and make sure that you understand how it works before you start.

I) Run Decision Tree Algorithm on Datasets (0 points)

Your first task is to run the Decision Tree Algorithm (your starting code) on the sample datasets.
a. First, copy everything except the solutions (you won't be able to) from ~cs4701/Project4 to your own directory, cd to sample_code in Project4 folder, and start the python console from there.
b. Then import the module by running "from learning import *". After this step, the decision tree code and the sample dataset should be loaded. You can check it by running:

```
>>> zoo
<DataSet(zoo): 101 examples, 18 attributes>
>>> restaurant
<DataSet(restaurant): 12 examples, 11 attributes>
```

Alternatively, you can load a dataset from a csv file
E.g. if you want to load the zoo dataset from "~cs4701/Project4/data/zoo.csv", you can run:

```
>>> zoo = DataSet(name='../data/zoo', target='type', exclude=['name'],
        attrnames="name hair feathers eggs milk airborne aquatic " +
        "predator toothed backbone breathes venomous fins legs tail " +
        "domestic catsize type")
```

c. The next step is to train the decision tree, which can be done by:

```
>>> learner = DecisionTreeLearner()
```

```
>>> learner.train(zoo)
```

If your training is successful, you should see the trained decision tree by doing:

```
>>> learner.dt.display()
Test legs
 legs = 0 ==> Test fins
    fins = 0 ==> Test toothed
       toothed = 0 ==> RESULT =  shellfish
       toothed = 1 ==> RESULT =  reptile
    fins = 1 ==> Test eggs
       eggs = 0 ==> RESULT =  mammal
       eggs = 1 ==> RESULT =  fish
 legs = 2 ==> Test hair
    hair = 0 ==> RESULT =  bird
    hair = 1 ==> RESULT =  mammal
 legs = 4 ==> Test hair
    hair = 0 ==> Test aquatic
       aquatic = 0 ==> RESULT =  reptile
       aquatic = 1 ==> Test toothed
          toothed = 0 ==> RESULT =  shellfish
          toothed = 1 ==> RESULT =  amphibian
    hair = 1 ==> RESULT =  mammal
 legs = 5 ==> RESULT =  shellfish
 legs = 6 ==> Test aquatic
    aquatic = 0 ==> RESULT =  insect
    aquatic = 1 ==> RESULT =  shellfish
 legs = 8 ==> RESULT =  shellfish
```

d. Then you can make a prediction using the trained tree:

```
>>> learner.predict(['shark',0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0])
 'fish'
```

II) Decision Tree Algorithm with Pruning (40 points)

For this part, you need to modify the decision tree learning code from the AIMA book and add Chi-Square pruning to it. **NOTE**: Feel free to add functions, but DO NOT change the interface of the train(), prune() and count_nodes() functions.

a) Add Pruning (30 points)

To add pruning, please add another member function to DecisionTreeLearner:

```
class DecisionTreeLearner(Learner):
   …
   def train(self, dataset):
      …
   def prune(self, dataset, maxDeviation):
```

```
    # Implement pruning here
    …
```

The prune function should use the validation dataset to perform Chi-Square pruning on the decision tree.
A small test dataset is provided for you to test your pruning. You can load and train the dataset by using

```
>>> test_set = DataSet(name='../data/test_data', attrnames='X1 X2 Y', target='Y')
>>> learner = DecisionTreeLearner()
>>> learner.train(test_set)
```

The decision tree you get should be:

```
Test X1
 X1 = T ==> RESULT =  Lost
 X1 = F ==> Test X2
   X2 = T ==> RESULT =  NotLost
   X2 = F ==> RESULT =  Lost
```

The deviation on node X2 is 6, so if you run learner.prune(test_set, 0) and learner.prune(test_set, 5), the node X2 should not be pruned and the decision tree learned should remain the same; If you run learner.prune(test_set, 6.5), the node X2 should be pruned and the decision tree should be:

```
Test X1
 X1 = T ==> RESULT =  Lost
 X1 = F ==> RESULT =  NotLost
```

b) Add node count (10 points)
Add a member function to the DecisionTree class to count the number of nodes in the decision tree.

```
class DecisionTree:
  …
  def count_nodes(self):
      # Implement nodes counting here
      …
```

A quick way to check if you have implemented this function correctly is that dt.count_nodes() should be equal to the number of appearances of the word "Test" in dt.display(). For the small test dataset, before pruning, the trained tree should have 2 nodes.

c) Test your code

A test script "testDecisionTree.py" is provided for you. It will generate the decision tree from the car dataset (see part IV for more). You can run the script by "python testDecisionTree.py".

III) Generate Program that Executes the Decision Tree Algorithm (40 points)

In the first part, you have implemented a Decision Tree Algorithm. After you train the program with some data, it should be able to make predictions on new data. In this part, after your program is trained, it should be able to output another program that uses the decision tree to make predictions.

NOTE: Feel free to add functions, but DO NOT change the interface of the outputDecisionTree() function.

The function you need to implement is:

```
def outputDecisionTree(self, path):
        …
```

It's a member function in DecisionTree class and it will output the tree as a program to a path.

E.g., given a sample dataset:

Name	Hair	Type*
Cat	1	Mammal
Eagle	0	Bird

*Type is the target

Or in python list format: [['cat', 1, 'mammal'], ['eagle', 0, 'bird']].

(Please read the sample code for more details on the format)

A simple trained decision tree dt will looks like this:

```
Test hair
        hair == 0 ➔ RESULT = bird
        hair == 1 ➔ RESULT = mammal
```

After dt.outputDecisionTree('./output.py') is called, your program should generate an output.py file with nested if-else statements similar to this:

```
def predict(example):
  if example[1] == 0:
    return 'bird'
```

```
    elif example[1] == 1:
        return 'mamal'
```

When grading, your output.py file will be imported and the predict function will get called with some examples, e.g.

```
>>> from output.py import *
>>> predict(['cat', 1])
'mammal'
>>> predict(['eagle', 0])
'bird'
```

Therefore please DO NOT change the interface of the predict() function.
A test script "testGenerateDecisionTree.py" is provided for you. It uses the zoo dataset, outputs the generated decision tree as python code and calls predict on that code. Before you implement anything, it will FAIL. It should output

```
example: ['shark',0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0]
output:  fish
```

if you implement this function correctly.

IV) Test Your Algorithm on a Real World Dataset (20 points)

A Car Evaluation dataset is installed on the CLIC machines for you to test your algorithms. In "~cs4701/Project4/data/car", there are several files "car_train.csv", "car_cv.csv" and "car_test.csv" containing the training data, validation data, and test data respectively. The examples in the files have already been shuffled randomly. In these csv files, the first six columns are attributes: "buying", "maint", "doors", "persons", "lug_boot" and "safety". The last column is the target: "values". You can find more details about the dataset from https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

A test file is provided here: "~cs4701/Project4/sample_code/ testDecisionTree.py".

Use this dataset to ANALYZE all the algorithms you have implemented, e.g. compare how maxDeviation affects the results and how the results change with the size of training dataset. Include your analysis in your report.

A few notes:

- Use the training data for training.
- Use the validation data for pruning.
- Using the validation data, try several different maxDeviation values and find the best maxDeviation for pruning.
- Use the test data for testing.

- With an appropriate parameter, on the validation set, your pruning version Decision Tree algorithm should always perform better than the non-pruning version.

Here is a list of metrics you need to include in your report:
- The macro F1 score of all algorithms on test data
- The number of nodes in the tree before and after pruning
- The learning curves of all algorithms, please use your favorite visualization tool to draw them in one plot

Some useful definitions:
- True Positive (tp), False Positive (fp), True Negative (tn), False Negative (fn):

Given a class i, its tp, fp, tn and fn are:

	$target_{actual} = i$	$target_{actual} \neq i$
$target_{predict} = i$	tp_i	fp_i
$target_{predict} \neq i$	fn_i	tn_i

E.g. $tp_i = the\ number\ of\ targets\ that\ target_{predict} = i\ \&\ target_{actual} = i$

- Precision:

The precision of a class i is defined as:

$$precision_i = \frac{tp_i}{tp_i + fp_i}$$

- Recall:

The recall of a class i is defined as:

$$recall_i = \frac{tp_i}{tp_i + fn_i}$$

- Macro precision

The average of precisions over all the classes

- Macro recall

The average of recalls over all the classes

- Macro F1 score

$$F1 = 2\frac{MacroPrecision * MacroRecall}{MacroPrecision + MacroRecall}$$

Submission

As in the previous assignments, you will create a Project4 directory in the folder named after your unique key. Put the following files in this directory directly (not in a subdir).
Please DO NOT change the interfaces of train, prune, count_nodes and outputDecisionTree functions in learning.py and the interface of predict function in outputted decision tree.
Remember to make sure that we have the right permission to access your files.

What to submit:

- “learning.py”, it has to run on the CLIC machines.
- A report in text format named *report_uni.pdf* that explains what you did and your analysis.