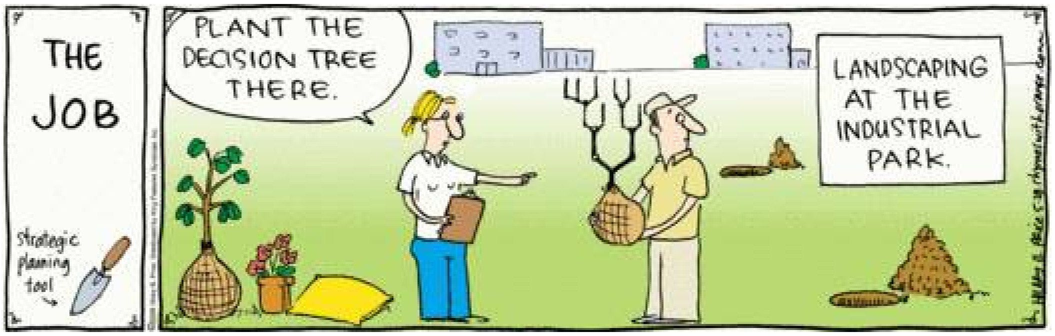

Project 4a:Decision Tree

Introduction

This project is intended to familiarize you with one of the standard approaches to classification problems, decision trees. You will code up decision tree learning and then apply it to several relatively simple problems. The hope is that, as you work on this project, you will come to understand the supervised learning process.

This project has two parts. In the first part, you will code up decision tree learning and test it on various data sets. In the second part, you will provide a writeup discussing the results.

This project will be done in Python. Your submission should consist of your code and your writeup. For the code, it should all be in DecisionTree.py; follow the standard submission convention you used in projects 1-3. For the writeup, publish it as a doc, rtf, or pdf document and include it with your submission as a separate file.

Files you will edit

DecisionTree.py	Your entire decision tree implementation will be within this file

Files you will not edit

DataInterface.py*	Functions for converting the datasets into python data structures
Testing.py	Helper functions for learning a tree and testing it on test examples
autograder.pyc	A custom autograder to check your code with

Evaluation: Your code will be graded for technical correctness and performance on the given datasets, primarily using the autograder. Your write up will be graded based on your interpretation of these

results.
Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; *please* don't let us down. If you do, we will pursue the strongest consequences available to us.

Getting Help: You are not alone! If you find yourself stuck on something, contact the course staff for help either during Office Hours or over email/piazza. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

***May be modified and submitted for extra credit**

Part 1: Decision Tree Learning Implementation (60%)

A classification problem is a problem where you classify instances into classes based on their features. For example, given the features *length_gt_5*, *has_teeth*, *big_and_scary*, and *flies*, we classify into *monster* and *not_monster*.
length_gt_5=y, has_teeth=y, big_and_scary=y, flies=n --> monster
length_gt_5=n, has_teeth=n, big_and_scary=y, flies=y --> monster
length_gt_5=n, has_teeth=n, big_and_scary=n, flies=n --> not_monster
length_gt_5=n, has_teeth=n, big_and_scary=y, flies=n --> not_monster
...
Other possible classification problems would include "Should I date this person or not?" or "Is this a good investment?" or "Animal or not?" or "Is this the picture of a man or a woman?"
A classification problem consists of four variables:

- *training-examples* - an ordered list of training examples, where each example is an *n*-dimensional vector
- *training-classes* - a corresponding ordered list of classification labels, where each label is t (in the class) or nil (not in the class). These should be in the exact same order as the training examples.
- *test-examples* - similar to *training-examples* in form, but used strictly for testing purposes.
- *test-classes* - similar to *training-classes* in form, but used to evaluate the results of running the algorithm on *test-examples*

In this project, we have decided to store sets of examples as lists of dictionaries. Each dictionary in a given list corresponds to an example vector in a given dataset and the class associated with it. For example, the monster dataset above will be stored as the following list:
[{‘length_gt_5’:’n’ , ’has_teeth’:’n’ , ’big_and_scary’:’y’ , ’flies’:’y’ , ‘monster’:’y’},
{‘length_gt_5’:’n’ , ’has_teeth’:’n’ , ’big_and_scary’:’n’ , ’flies’:’n’ , ‘monster’:’y’}
{‘length_gt_5’:’n’ , ’has_teeth’:’n’ , ’big_and_scary’:’y’ , ’flies’:’n’ , ‘monster’:’n’}
...]
Though it is typical to store examples as lists/vectors, this form of storage also retains the names of

all features. Note that this form of storage enables non-binary features and classes, which your algorithm will need to support. An example of a dataset with multi-valued features and classes is the cars dataset:
[{'maint': 'low', 'persons': 'more', 'lug_boot': 'small', 'safety': 'low', 'doors': '5more', 'buying': 'low', 'label':unacc',}
{'maint' : 'low', 'persons' : 'more', 'lug_boot' : 'small', 'safety': 'med', 'doors': '5more', 'buying': 'low', 'label': 'acc'}
{'maint' : 'low','persons' : 'more', 'lug_boot' : 'small', 'safety' : 'high', 'doors' : '5more', 'buying': 'low', 'label': 'good'}
{'maint' : 'low', 'persons': 'more', 'lug_boot' : 'med', 'safety' : 'low', 'doors' : '5more', 'buying' : 'low', 'label': 'unacc',}
…]
This, along with two with Connnect4 dataset and two small testing datasets, can all be obtained from DataInterface.py. The format and contents of the two real datasets can be found in the datasets folder.
To allow for autograding and ease the difficulty of implementing a full Decision Tree, we have prepared a framework of code in DecisionTree.py that has functions that you need to fill in. The functions have descriptive comments to specify what their functionality is. So for each question, reading those rather than guidelines here will let you know what to code. The questions and associated points are as follows:

Question 1 (2 points): Helper functions

Implement getMostCommonClass, getPertinentExamples, getClassCounts, and getAttributeCounts past the Node and Tree classes in DecisionTree.py in accordance with the comments. These will be useful later in implementing the information gain functions, and should get you familiar with the format of storing examples as dictionaries. As before, you can run the autograder with python autograder.pyc and can specify -q or -t/--test to check your solutions are correct.

Question 2 (2 points): Entropy functions

To start with, implement setEntropy, remainder, and infoGain in DecisionTree.py as covered in the book. These are covered in chapter 18.3.4 in the book, with setEntropy being analogous to H. However, note that the book specified H for a binary variable, so be sure to follow the instructions in the lecture and comments to implement setEntropy for non-binary variables.

Question 3 (2 points): gini functions

Now that you have completed the standard decision tree learning approach, you will implement another mechanism for splitting attributes, called the GINI index. The gini index function can be used to evaluate the goodness of all the potential split points along all attributes. Consider a dataset S consisting of n records, each belonging to one of c classes. The gini index for the set S is defined as:

$$gini(S) = 1 - \sum_{j=1}^{c} p_j^2$$

where p_j is the relative frequency of class j in S. If S is partitioned into two subsets S_1 and S_2, the index of the partitioned data can be obtained by:

$$gini^D(S, cs) = \frac{n_1}{n} gini(S_1) + \frac{n_2}{n} gini(S_2)$$

where n_1 and n_2 are the number of examples of S_1 and S_2, respectively, and cs is the splitting criterion. More generally, if S is divided into n subsets the equation above is a summation from S_1 to S_n. Here the attribute with the minimum gini index is chosen as the best attribute to split.

The functions to implement for this question are giniIndex and giniGain - be sure to read the comments.

Question 4 (4 points): Decision Tree Learning

With all the above functions in place, you are ready to implement the makeSubtrees method of DecisionTree.py. Note that the trivial makeTree method is provided to you and the implementation of the decision tree construction algorithm is left for you to do in the recursive makeSubtrees. This function should return a Node, which is the root node of the decision tree for the provided set of examples and attributes. You should follow the pseudocode in the book and slides, and have the same base cases. Additional means of testing are provided in Testing.py with methods such as testDummySet1.

Question 5 (2 points): Decision Tree Classification

With all the above functions in place, you are able to learn the structure of decision trees from data, and just need the function for using the learned tree. The classify function is in the Tree class, and should be implemented after makeSubtrees since it implements the logic for using a tree for classification. This should be a straighforward tree traversal implementation, so if you are not sure how to go about this be sure to conceptually review Decision Trees.

Part II: Analysis (40%+10% Extra Credit)

The analysis should be provided in 1-2 paragraphs. Please keep it short and concise - we primarily care about you displaying understanding of how Decision Trees fundamentally work.

Question 6 (4 points): Perfomance

Using the method of Testing.py, find and record classification rate and tree size statistics regarding the two fake data sets, as well as the Cars and Connect4 datasets.

For each data set, describe why you think the decision tree learning performed as it did in terms of accuracy statistics and tree size. Please be specific and justify your claims (ie: if you say there was not enough training data for data set X, or the nature of training data made Decision Tree learning less effective, what makes you think so?).

Question 7 (4 points): Applicability

For each of the two real datasets that come with this project, suggest some practical uses of learning and using a Decision Tree. For the cars dataset, you may suggest a similar dataset with the same or a different label and analyze how having a classifier such as a Decision Tree could be useful to consumers or websites selling products. For the Connect4 dataset, suggest some way in which the classifier could be incorporated with some of the past algorithms we have implemented for pacman to make a better Connect4 playing bot.

Question 8 (2 points Extra Credit): Novel Dataset

Explore the UCI Machine Learning Repository or other ML datasets found online, and select a new dataset to try your Decision Tree with. Write a method in DataInterface.py called getExtraCreditDataset that is akin to the other get methods we wrote. Answer the two questions above for this dataset, and submit your DataInterface.py in addition to DecisionTree.py for the option of extra credit. Additionally, submit the dataset files that you answered these questions for.

Submission

Zip only the files you altered for this assignment as a .zip or .tar.gz and submit it on T-Square before the due date. Include your analysis in the pdf. You have a strict one hour window after the due date to handle any last minute technical issues, after which you will not be able to submit and receive a 0 for the project.