Homework #3 - Machine Learning

COEN 266 - Artificial Intelligence

Spring 2017

Overview

For this homework assignment, you will be implementing a decision tree that uses the ID3 algorithm.

Part 1: Data Cleaning (20 points)

For this portion, you will need to develop a dataset comprised of at least 100 examples. If you would like, you can use one of the datasets that are publicly available at: https://archive.ics.uci.edu/ml/.

The data should be comprised of categorical attributes and classifier. If you find a data set you would like to use and it contains numerical values, you can massage it into categorical form by partitioning the values into ranges (extra credit can be had for implementing an algorithm that automatically performs this task, see later in the assignment).

You should also clean your data set by removing examples that have missing values (or figuring out a reasonable way to represent them, perhaps by adding a different value, such as "unspecified"). Next, you will need to create exactly three files: an attributes file, a training dataset file, and a testing dataset file.

Attributes File

This file contains the list of all of the attributes and their possible values (this will include the classifier and its values). Each line in the attributes file should be in the format "attribute: value0, value1, … valueN". For example:

```
size:small,medium,large
sharp teeth:yes,no
feet:yes,no
habitat:land,water
domesticated:yes,no
dangerous:yes,no
```

Your data set should be unique to you. Feel free to share it with other students for testing purposes, but they should not also submit the same data set.

Dataset Files

Each of the dataset files should have a list of examples, with one example per line. Each example is simply a comma-separated list of values, one for each attribute in the order provided in the attributes file. There may also be a "*name*:" at the beginning of a line to use as a label for the data. These labels are optional and simply for readability: they are thrown away by the parser. Three lines from a dataset file may look like this:

```
snake:small,yes,no,land,no,yes
bear:medium,yes,yes,land,no,yes
deer:medium,no,yes,land,no,no
```

Randomize the order of your examples (this is more important for some datasets than others, but it never hurts). This can be easily accomplished with the `shuf` utility on the DC workstations.

Next, parition all of your examples into two dataset files, one to use for training the decision tree algorithm (training set) and one for testing the effectiveness of your trained tree (testing set).

Part 2: ID3 Decision Tree Creation and Testing (70 points)

Implement an ID3-based decision tree creator and tester. It will accept a set of data in the format described in part 1, and from that data produce a decision tree. It will also allow testing the accuracy of the decision tree by running test cases against the tree and comparing the decision tree's performance against the known classifications.

In some cases, you will run out of attributes before uniformly classifying the examples (this can happen if more than one example has the same attributes, but different classifications). Also, you can have the case where there are values in a decision tree for which there are no examples (this should only happen if you have attributes with more than two values). The following guidelines describe appropriate handling of these cases:

1. If you run out of attributes and there are examples with different classifications, choose the classification for which there are the most examples. If there are a duplicate number of examples in more than one group, choose the classification with the earliest alphabetical value (using python's native string comparisons).

2. If you have a subpopulation that has no examples for an attribute value, choose the most prevalent example from the population that falls into the parent's domain, as per guideline 1.

Framework

You are provided the following source files to use in this assignment:

`main.py`: Provides a command-line interface to the decision tree. It takes positional parameters for the decision tree algorithm module name, and for the name of the classification attribute. Invoke with --help to see complete list of options.

> Example 1: Use the DTree implementation found in id3.py, and invoke on the dangerous-animals training data, classifying based on the 'dangerous' attribute:
>
> ```
> $ ./main.py id3 dangerous --attributes tests/dangerous-animals-attributes.txt \
>   --train tests/dangerous-animals-train.csv
> ```
>
> Example 2: Use the DTree implementation found in id3.py and invoke on the kr-vs-kp training data, classifying based on the 'white-can-win' attribute. After building the complete decision tree, run it against the test data and report its performance:
>
> ```
> $ ./main.py id3 white-can-win --attributes tests/kr-vs-kp-attributes.txt \
>   --train tests/kr-vs-kp-train.csv --test tests/kr-vs-kp-test.csv
> ```

`attributes.py`: Specifies an interface for storing information about attributes, including reading them from a file in the format described in section 1. Note that attributes are sometimes treated as ordered (when reading in examples and mapping the values to attributes), and sometimes treated as keyed (when accessing attributes to retrieve the possible values).

`dataset.py`: Specifies an interface for storing individual datapoints (class `Example`), and collections of datapoints (class `DataSet`). A `DataSet` can be initialized from a data file in the format described in section 1.

In general, the provided files should not be modified, with the exception of `dataset.py`, in which you should implement an entropy calculation (the `entropy` method is stubbed in for the framework).

A starting point for `id3.py` is also provided, but is mostly unimplemented.

API

You will need to implement the class `DTree` in the file `id3.py`. This class must implement at least the following methods:

Method	Arguments	Type	Meaning
`__init__` *Creates a new decision tree*	classifier	Attribute (see attributes.py)	Attribute that is being used for classification
	training data	DataSet (set dataset.py)	Set of training data
	attributes	Attributes (see attributes.py)	All attributes in this domain
`test` *Uses a decision tree to classify test examples*	classifier	Attribute	Attribute that is being used for classification
	testing data	DataSet	Set of testing data
	<return value>	int	Number of test examples that were correctly classified by the decision tree
`dump` *Prints out a visual representation of the decision tree*	None		

Feel free to add **optional** arguments to methods, but do not change the mandatory arguments -- these methods must continue to work with the provided `main.py` file.

The format of the dump output for a **non-terminal** node in the decision tree is:

attribute-name:*attribute-value-1*

attribute-name:*attribute-value-2*

…

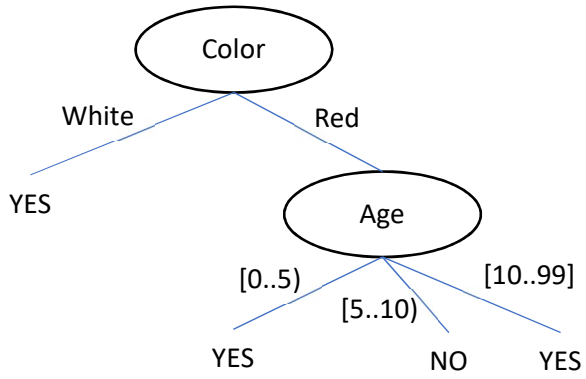
attribute-name:*attribute-value-n*

A **terminal** node should be represented as:

<classification>

Each line should be indented one space for each level it is below the first.

For example, consider the following decision tree for classifying wine:



It would be represented as:

```
color:white
 <YES>
color:red
 age:[0..5)
  <YES>
 age:[5..10)
  <NO>
 age:[10..99]
  <YES>
```

Testing

The shell script `run_tests.py` will be used to test your programs. Please use it to verify correctness before sumission. The script looks for all `.out` files in the specified test directory. For each found, it will look for the following files:

`testname``-attributes.txt` (required): Specifies all attributes and their values. The last attribute in the file is used as the classifier.

`testname``-train.csv` (required): The training data, with one example per line.

`testname``-test.csv` (optional): The testing data, with one example per line.

`run_tests.py` will run the decision tree algorithm with the training data, and print the resulting tree. If a testing data file is also present in the test directory, the script will also run those examples against the decision tree and report the accuracy of the tree in predicting a classification for them. Finally, all output will be compared against the known good results in the *`testname`*`.out` file.

Note

All code must run on the Engineering Design Center Linux machines (`linux.dc.engr.scu.edu`***) without additional packages installed. Note that these machines are running a fairly ancient (2.6.6.) version of Python, so if you do any development outside of the DC computers, make sure to test early and test often!***

Grading

You will be graded on correctness and coding style. The assignment will be graded out of 100 points, so anything above that is equivalent to extra credit.

Additional Credit

For additional credit, implement no more than **one** of the following:

- (15 points) Implement k-fold validation by running k iterations, each time using a different partition of the data for training and testing. Note that this will require changes to the main.py interface, so please create a separate `main-kfold.py`.
- (20 points) Implement a decision tree using random forests. It should have the same interface as the existing decision tree, but the tree dump should dump all of the generated trees in the forest. If you choose to implement this, create a `id3-random-forests.py` file for it.
- (30 points) Implement support for handling real-valued attributes, automatically partitioning at points where entropy is most greatly reduced. If you choose to implement this, create an `id3-real.py` file for it. Also, submit two data sets, one for the standard id3 algorithm consisting only of categorical data, and one with real-valued attributes for use with the extra credit assignment.

As always, the first person to identify each significant shortcoming in the assignment itself (this document) will receive a 5 point bonus.

Submission

The assignment is due Wednesday, June 7th at 8am. No assignment will be accepted after 8am on Monday, June 12th. All submissions should be made on Camino.