CSCI 373 Project #5
Playing Golf

Due April 18, 2017

—

Pacman will not know
to play golf on a clear day
but that you will learn

—

This week we're leaving the world of Pacman, as well as the scaffolding that the Pacman Python code has provided. For this assignment, you'll implement top down induction of decision trees in Python "from scratch". Your program will have to handle data sets with an arbitrary number of discrete-valued attributes, an arbitrary number of classes, and an arbitrary number of examples.

For part 1 of the assignment, you will need to read a data file where all attributes are nominal-valued and no data are missing, produce a human-readable version of the decision tree created from the data set, and use the decision tree to classify test examples. For part 2, you will choose an extension to implement, from a list that I provide. Note that if you took Machine Learning in the fall (or if you work with someone who took Machine Learning), you will need to complete two extensions.

Read this entire handout before you get started, as it isn't designed to take you through the process step-by-step in the same way as previous assignments this semester.

1 Part 1

1.1 Data Files

Download the data files in the zip archive available on the web page for this assignment. Skim the files to get a sense of their format.

Your program should begin by reading one of these files, as given by the user on the command line. Each file is called *domainName*.arff, where *domainName* is the name of the domain of application. The basic version of this assignment should be able to handle:

- weather.nominal.arff - the weather domain described in class, but with numeric features replaced by nominal ones;
- contact-lenses.arff - given a description of a patient, determine the type of contact lenses that should be prescribed;
- betterZoo.arff - given a description of an animal, determine its class. (The original data set contained each example animal's name as a feature. I have removed that feature but left the documentation about it.)

Note that these data sets contain only nominal-valued attributes, and they have no missing data. I will test your program on these as well as other similar data sets.

An arff file begins with a preamble that describes the domain. This is followed by a section that specifies the attributes and the possible values for each attribute. The final attribute listed is the class to be predicted. Each attribute is on a separate line and has the following form:

```
@attribute outlook {sunny, overcast, rainy}
```

It begins with the string "@attribute". This is followed by the attribute name. The possible attribute values are listed in curly braces, separated by commas. Please be careful when reading in this information. In particular, the word "attribute" is sometimes uppercase and at other times lowercase. Attribute values might be separated by commas only, or they may be separated by commas and blanks.

Each file also contains data – i.e., specific examples and their associated classifications. The data part of the file begins with the line

```
@data
```

Each line of data has the form:

```
sunny,hot,high,FALSE,no
```

This gives the values for each of the attributes defined earlier in the file. The final value is always the class value. Again, be careful about format. In an arbitrary arff file, examples' attribute values may be separated by commas only; they may also be separated by commas and blanks.

1.2 Top Down Induction of Decision Trees

Once you're able to read in a data set, you can begin building your decision tree. As part of this assignment, you'll need to perform leave-one-out cross-validation, but at the beginning, focus on learning a tree from the complete set of examples.

You will choose tests (i.e., attributes on which to split) using the information gain criterion. Recall that if T is the set of training data (containing $|T|$ instances) representing a concept with c classes, and $freq(C_j, T)$ is the frequency of class C_j occurring in that set, then the entropy of that set is:

$$entropy(T) = -\sum_{j=1}^{c} \frac{freq(C_j,T)}{|T|} \times \log_2 \left(\frac{freq(C_j,T)}{|T|} \right)$$

bits.

If an attribute X with v values is selected as a test attribute, then the expected entropy under that test is:

$$entropy_X(T) = \sum_{i=1}^{v} \frac{|T_i|}{|T|} \times entropy(T_i)$$

where T_i is the subset of T all of whose instances have value i for feature X.

The information gain, then, is simply the difference between the entropy with and without the test on attribute X:

$$gain = entropy(T) - entropy_X(T)$$

The attribute yielding the highest information gain is selected as the current test. Note that when splitting at a node of the tree other than the root, the set of examples under examination, T, will be only a subset of the original set of training examples (in particular, it is the subset of training examples that have propagated down through the nodes that are parents of the node under consideration).

1.3 Checking Your Work

You might be wondering how you'll know whether your learned decision tree is correct or not. The answer is that you'll be able to compare your tree against the one learned by Weka. Weka is software for data mining produced by a group at the University of Waikato in New Zealand.

Weka is installed on the machines in the unix lab. To run it, simply type

```
java -Xmx1g -jar /usr/share/java/weka.jar
```

This will start up a GUI. Click on the button that's labelled "Explorer". This will open another window that will allow you to select data sets and machine learning algorithms.

Begin by selecting a data set. You can do this by clicking on "Open file..." and then selecting an "arff" file of your choice.

Once you've selected a data file, click on "Classify" and then "Choose". In the "Trees" directory, click on Id3. Before starting the training process, select "Use training set" in the "Test options" panel. This will force the algorithm to use all of the available examples for training.

Then click on "Start" at the left side of the window below "Test options". (You may need to resize the window to see the start button.) Scroll through the "Classifier output" panel to find the decision tree. It's a purely textual representation of a tree, but it should be quite easy to read and will serve as a model for your own output.

When you run Id3 on the zoo data set, notice that some of the leaves are null. Think about why that can happen. Also consider what you might do if you wanted your tree to produce a class at all leaves (i.e., replace all "nulls" with class names).

If you have any trouble working with Weka, let me know.

1.4 Creating Training and Test Sets

Once you feel confident that you're inducing decision trees correctly, you can move on to evaluation. Because the data sets I've provided are quite small, you'll perform leave-one-out cross-validation. This is essentially n-fold cross-validation, where n is the number of examples in the entire training set.

First you'll need to write code that will allow a learned decision tree to classify a new example. Then you can write the leave-one-out cross-validation code, which (at least at a high level) will look something like this:

```
for each example in the data set
  remove that example and reserve it for testing
  learn a decision tree with all examples except the one just removed
  test the learned tree on the reserved example
  keep track of whether it was classified correctly or not
report the accuracy (as percent correct)
```

1.5 Part 1: Summary of Requirements and Deliverables

It is up to you to determine how to design the program, with just a couple of constraints. I will expect to be able to run it from the command line with:

```
python tdidtPart1.py Data/weather.nominal.arff
```

So `tdidt.py` should be the name of the file containing the `main` method that will start everything off, and it should expect a file name to be included as an argument.

As always, please turn in all relevant files. Also, please send me an email telling me the names of the partners working on the project, and who turned it in.

2 Part 2

You've just finished implementing ID3, which makes some strong assumptions about the data available for learning. In this part of the assignment, you will relax some of those assumptions. If you took Machine Learning in the fall (or if you are working with someone who took Machine Learning), select two of the extensions below. Otherwise select one extension to implement.

The possible extensions are as follows:

- Handle continuous-valued attributes.
- Handle missing attribute values.
- Implement reduced-error pruning.

Before doing any implementation, think carefully about how you will separate the functionality required for Parts 1 and 2, as I will test those separately. One option is to have two parallel copies of the code – one that does everything required for Part 1, and an expanded version that does everything for Part 2. (Part 2 still requires being able to handle everything above.) Or you can have a single version that allows the user (i.e., me) to specify, say "no pruning" vs "pruning".

2.1 Handling continuous-valued attributes

If you choose to do this, you will need to start by modifying the code that reads and parses an arff file. In the Data zip archive you'll find two data sets that contain continuous-valued attributes on which you can test your work. They are betterZoo.numeric.arff and weather.arff. You cannot use Id3 to check your trees. Instead, use Weka's J48 implementation. You will need to modify a couple of parameters to ensure that J48 doesn't do any pruning. To do so, click on the "J48" text to the right of the "Classifier" panel's "Choose" button. Set `minNumObj` to 1 and `unpruned` to True.

2.2 Handling missing attribute values

Missing attribute values are indicated in an arff file by the question mark symbol. In the Data zip archive you'll find two data sets that contain missing attributes on which you can test your work. They are vote.arff and weather.nominal.unknowns.arff. As with continuous-valued attributes, you'll need to use J48 rather than Id3 to check your work. You will need to modify a couple of parameters to ensure that J48 doesn't do any pruning. To do so, click on the "J48" text to the right of the "Classifier" panel's "Choose" button. Set `minNumObj` to 1 and `unpruned` to True.

There are several different ways to handle missing values. C4.5 (of which J48 is an implementation) does so as follows. When computing information gain for an attribute, it considers only those examples that have a value for that attribute. (Remember that when this happens, the size of the data set at that node will temporarily look smaller than it actually is.) When splitting a training set at a node, if the splitting attribute's value is unknown for an example, C4.5 sends a fraction of that example to each child. So if an example gets split over two children, each child gets half an example; if there are three children, each one gets a third of an example. This isn't a problem when computing information gain. Instead of counting a weight of "1" per example, you count whatever weight the example currently has.

When C4.5 tests an example and gets to a node where it needs an attribute value that is missing, it again takes the example and sends a piece of it in each direction. This might mean that you eventually get several possible classifications that you need to reconcile (say, by voting).

An alternate – but simpler – way to handle missing values is to replace them with means (in the real-valued case) or modes (in the nominal case).

If you choose this extension and test against J48, you might not induce the exact same tree, but it will provide at least a sanity check.

2.3 Reduced-error pruning

Reduced-error pruning replaces a subtree with a single leaf when that leaf represents an error rate no worse than the subtree. The pruning procedure works roughly as follows:

```
Remove a random subset of examples from the training set (already separate from the test
set) and reserve them for pruning.

Construct the full unpruned tree with the training set (minus the pruning set).

In a postorder fashion, traverse the decision tree, replacing a non-leaf node with a leaf
when the error rate of the leaf does not exceed the error rate of the subtree.  The error
rate should be measured on the pruning set.
```

As a sanity check on your output, you can compare your results to those of J48. Set the parameter for `reducedErrorPruning` to True and the parameter for `subtreeRaising` to False. The `numFolds` parameter in the J48 panel specifies how many training examples are held out for pruning. If the number of folds is 3 (the default), then one third of the examples are held out. You might find it especially fun to try out your code on weather.nominal.arff with just one example held out for pruning. The contact-lenses.arff data set is another one where you should get some dramatic results.

2.4 Part 2: Summary of Requirements and Deliverables

As for Part 1, I will expect to run your code from the command line. Here, however, you will need to tell me how to run your program. Specify the basic command as well as any parameters that I can select. Include this information, along with anything else I should know, in a README.txt file. Submit this with your code.