Massachusetts Institute of Technology

6.038 Artificial Intelligence in Practice

Assignment 2: Tuesday, February 11, 1997

1 Objective

The objective of this assignment is to give you experience in using two powerful programs in machine learning – one focused around neural nets and the other around classification trees.

2 Files

You can find the software required for this assignment in `/mit/6.038/bin`. SNNS and C4.5 have been compiled for Sun and SGI workstations. Let me know if you have any problems running the code.

Make sure that you `add 6.038` when starting to work on this assignment. This will put the correct bin directory on your PATH.

The other files mentioned below can be found in `/mit/6.038/assign-2/`.

3 Part 1: Neural Nets with SNNS

In this part, we will go through the process of building and training nets for a couple of simple problems. We will consider two data sets: `vote` and `sin`. The `vote` data set is a classification problem: predicting democrat or republican based on a pattern of votes. It was derived from `vote.data` and `vote.names`). The `sin` data set comes from some cycles of a sin wave with uniformly distributed noise. It was generated by the code in the file `snns.lisp`.

For each data set, you will find two data files, e.g. `vote-train.pat`, `vote-test.pat`. One data set is for training and the other is to be used for testing the networks. Note that you can have both the training and data set loaded into SNNS at the same time – you can set VALID in the Control Panel to, e.g. 2, to see the test set error as well as the training set error during training. The USE buttons can be used to select among the loaded data sets for training and validation.

In constructing a Neural Net, one usually has to experiment a bit to find a good architecture, in particular, the number of layers and units. For each data set, you should build three feedforward networks:

1. A network with no hidden units

2. A network with one layer of hidden units (try 10 units)

3. A network with two layers of hidden units (try 10 units in each layer)

One crucial issue that you will have to decide on is the "activation function" for the output unit of the network for each kind of problem. The default activation function for feedforward networks is the sigmoid but this is not appropriate for every problem (Why?). To change the activation function for the output unit, you need to use the INFO panel (see the manual p. 45-46 and remember to hit SET after you've chosen the function you want). Note that you can select which unit is being displayed in the INFO panel by middle clicking on the unit in the DISPLAY panel.

For each architecture one tries, one has to find a good method for training the net without overfitting. For each network, try the following three "learning" functions:

- `Std_Backpropagation`
- `BackpropMomentum`
- `Rprop`

For each learning function, you should keep training until you're sure the training error (SSE) stops changing (10000 cycles or more in some cases). Try each learning function three times, hit RESET and INIT between each try so you can see the impact of a different random set of initial weights on the training. Warning: SNNS requires you to type meaningful numbers for the parameters of the learning functions; see page 29 of the manual for suggested ranges. You might need to play with the parameters a bit to get good performance.

You should use the GRAPH window to monitor performance during learning. You also get a printed display on the log file (stdout as a default). I've noticed that there appears to be a bug in graphing, you don't always get values consistent with the printing and sometimes the graph does not appear at all. Nevertheless it is usually a useful indicator of how the training is going.

Generate a result (RES) file (that includes the training patterns as well as the output patterns) for each problem. You can generate this file from the FILE menu. Note that the results file is generated for the data set that is currently chosen as the trainging set, not the validation set! You can usually tell which set has been chosen by hitting the DEFAULT button on the result file generation window and looking at the number of patterns in the data set. Note that the default is to NOT include the desired output for each pattern in the results file, but you do want to include it for comparison.

You can generate an informative plot on the absolute value of the error for each pattern in the results file as follows:

```
add lucidcl
cd /mit/6.038/assign-2/
lisp
(load "snns.lisp")
(generate-error-plot-file-from-result-file "vote.res" "vote.plot"
(quit)
gnuplot
plot "vote.plot"
```

That is, start lisp and run a function that processes the results file into a plot file (which is simply an entry, for each patern, corresponding to the abs of the difference of the desired value and the computed value for each pattern). This can be plotted using gnuplot.

Hand in a summary of your training results, indicating for each problem the impact of the network architecture, output activation function and training method. This can simply be a table indicating performance (SSE on the test data and, for `vote`, the number of classification errors); avoid reams of output. Is the best final trained performance on the `sin` data reasonable? Explain. Also, explain the performance on the training data vs the test data during training (in both cases).

Look at the weights for the neural net without any hidden units for the `vote` data set. What is the net doing?

What lessons do you draw about the architecture and training of neural nets?

4 Part 2: Classification Trees with C4.5

Luckily, using the classification tree builder is a bit easier. You will find two data sets in the directory: `vote` and `soybean`. Each data set is represented by two files, e.g. `vote.data` and `vote.names`.

Run the tree and rule generators individually on both data sets and look at the trees and rules produced. Can you find intuitive interpretations of the results?

Repeat the analysis in Table 9.1 (p 93) in your handout on both of these data sets. Explain the results.

Compare the performance of the neural net and the classification tree on the `vote` data set. Can you explain the behavior?

What lessons do you draw about classification trees and rules?

5 Part 3: Applications

Can you think where either or both of these programs may be useful in building Harvey? Give concrete examples.