Non-Symbolic AI lecture 9

Data-mining – using techniques such as Genetic Algorithms and Neural Networks.

Looking for statistically significant patterns hidden within loads of data – picking out the jewels from the rubbish.

Potentially valuable in searching for gold, searching for new drugs, searching for patterns in the stockmarket …

… etc etc etc.

Simple example

A survey of 20 CSAI graduates 5 years after graduating shows that 10 of them are dot.com millionaires and 11 of them are in jail.

There appears to be no particular correlation with their overall degree results … but perhaps there is a correlation with how well they did in one or more particular courses.

Maybe all the millionaires did well at Non-Symbolic AI **AND** badly at Intro to Logic Programming.

How can we find out if there are any such patterns?

The Outer loop

Split data into 50% train/ 50% test

Initialise at random a GA popn ofgenotypes

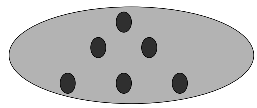
Run a GA for many generations, using mutation and recombination, to reduce errors on generalisation.

evaluate

Best trained ANN

Middle loop

Specification of 3 inputs (3 CSAI courses) to take notice of



Build 3-2-1 ANN

Initialise random wts

Train weights on train data

Calculate error on test data

Inner loop

Repeat over many presentations of training data

Present one set of inputs to the ANN

Pass forward thru layers to produce output

Error = difference between Output and Target

Back-propagate errors, and then adjust the weights

Inputs and Outputs

OK, lets take as inputs the scores on each of 30 courses taken over 3 years, each translated into a number between 0.0 and 1.0.

Lets take as the single output a number 1.0 for = millionaire, a number 0.0 for in jail.

So we are looking for a Black Box of some kind that will take 30 separate inputs, and output a single number that is its guess at how likely that person is to finish up a millionaire/in jail

Only 20 examples of data in our training set – is that a problem?

Overfitting ?

The small number of data points **could** be a problem.

If there is a clear and consistent simple rule that works well for the training set –

e.g. "All and only those who got over 60% on Non-Sym AI become millionaires, forget all the other courses as they are irrelevant"

-- then it might be reasonable to trust this.

But if the Black Box generates really complex rules involving scores on many courses, this may well be over-fitting the data.

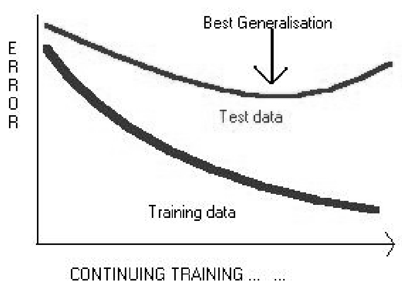
Training and Test data

The standard way to check for overfitting is to split the data into a training set and a test set.

Train the Black Box (… whatever method this may be …) on the training set, and then see how well it generalises to the unseen test data.

If there is a risk of overfitting, then the longer you train on the training data, the better the fit *on the training data* will be – but very likely at some stage the generalisation to test data will get worse.

Stop before you overfit



Calculating the errors

Errors, on either training or test data, depend on the difference between predictions (the actual output of the Black Box prediction machine, the ANN) and te known Target values.

Guessing 1.2 too high, or 1.2 too low usually counts as equally bad.

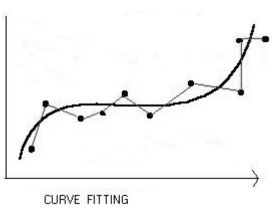
So the usual measure of errors is **Sum Squared Errors**

So over all the M members of a test set, the error measure will be

$$SSE = \sum_{i=1}^{i=M} \left(T_i - O_i\right)^2$$

Smoothing

Generally, smoother curves are more likely to generalise to new data than the (over-fitted) curves that go exactly through every training point.



Regularisation theory discusses topics such as how to get such smooth curves.

Regularisation

One method of 'encouraging' an ANN to produce smoother curves is to avoid lots of big positive or big negative weights – they are the ones that tend to produce jagged, non-smooth curves.

A good trick for doing this is **weight-decay**.

Introduce an extra routine into your program code, so that after every set of weight changes (through eg backprop) **all** the weights are decreased in (absolute) value by eg just 1%.

Then if the algorithm 'has a choice' between producing jagged curves or smooth ones, it will inevitably tend to opt for the smoother curves.

Too little data?

Back to the train/test approach for avoiding over-fitting.

That requires you to keep some part of the dataset back for testing, to check for generalisation.

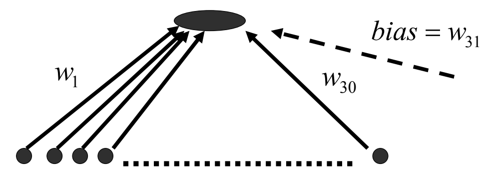
But if you only have 20 items of data anyway, dividing these into 2 parts makes the training data set even smaller.

This is often a serious problem.

Linear Black Boxes

If the problem is fairly simple, then there may be a simple linear formula that gives a good prediction.

$$out = w_1 In_1 + w_2 In_2 + \ldots w_{30} In_{30}$$



Linear

So a single-layer perceptron could handle this.

Note that if you have 30 weights to discover (or 31 if you add a bias term), and only 20 data points, then there will be many different possible **exactly accurate** formulae (or Black Boxes).

$$eqn1 \ldots 1.0 = 0.6w_1 + 0.3w_2 + \ldots + 0.7w_{30} + w_{31}$$

$$\ldots\ldots\ldots$$

$$\ldots\ldots\ldots$$

$$eqn20 \ldots 0.0 = 0.5w_1 + 0.55w_2 + \ldots + 0.65w_{30} + w_{31}$$

20 simultaneous equations in 31 variables, to be solved.

Linear (ctd)

Basically, if you have **N**=31 parameters to find (weights + bias in a feedforward ANN, or formula as before), then

- If you have less than **N** equations (less than **N** examples in your training set) there are loads of possible ‘exact’ solutions (… many of which will not generalise …)
- If you have exactly **N** eqns there is 1 ‘exact’ solution (and math techniques can find it, you don’t need ANNs)
- If there are more than **N** eqns (datapoints – CSAI students) then probably there is not an exact linear solution.

Non-linear Black Boxes

A linear Black Box is something relatively simple like:

“Chance of becoming a millionaire = 2.0 * Non-Sym AI score minus 3.2 * Logic Prog score +0.5”

Otherwise rephrased as: $out = 2.0 In_{12} - 3.2 In_{17} + 0.5$

Non-linear means potentially much more complex formulae, like:-

$$y = 0.2x_1 - 3x_1^2 + 2x_1x_2 - x_3^{4.5} + \frac{x_6}{\sqrt{x_7}} - \log\left[2x_5 - \frac{x_4^3}{x_5}\right] + \ldots$$

But don’t worry …

… because a multi-layer perceptron or ANN – and normally adding just 1 hidden layer will do --- with non-linear transfer functions (the sigmoids) can do a pretty good job of making a Black Box that reproduces **any** such complex formula.

Provided that you have enough hidden nodes, and can find the right weights through a learning algorithm such as backprop.

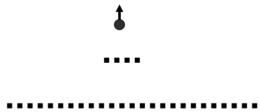
Still a few problems, though.

How many parameters to find ?

Suppose you now have 1output,

4 hidden,

and 30 inputs.



Counting biases as weights, you now have

(30 +1) * 4 + (4 + 1) * 1 = 129 weights or parameters to find

But you still only have 20 datapoints, 20 examples of CSAI students with known input ==> output results.

That simply isn't really enough. Suggestions ?

Suggestions

Well, one suggestion is to get a whole lot more data. Rather than 20 students from 1 year, get 1000 students from many years, then you will have enough to split into training and test sets, and train the ANN up nicely.

But this isn't always possible.

Maybe you have to hope that there is a simpler model, that only uses a much smaller number of the inputs.

For instance, maybe you can predict millionaire/in jail from results on only 3 CSAI courses rather than 30.

The Outer loop

Split data into 50% train/ 50% test

Initialise at random a GA popn ofgenotypes

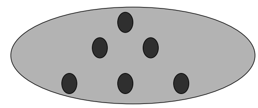
Run a GA for many generations, using mutation and recombination, to reduce errors on generalisation.

evaluate

Best trained ANN

Middle loop

Specification of 3 inputs (3 CSAI courses) to take notice of



Build 3-2-1 ANN

Initialise random wts

Train weights on train data

Calculate error on test data

Inner loop

Repeat over many presentations of training data

Present one set of inputs to the ANN

Pass forward thru layers to produce output

Error = difference between Output and Target

Back-propagate errors, and then adjust the weights

Common data-mining problem

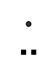
You have lots of *possibly* relevant input data, but it seems likely that most of it is irrelevant.

"90% of advertising is wasted – except we don't know *which* 90%"

Well, one suggestion is to try in turn **all** possible subsets of 10% of the inputs; each time generate a Black Box using just those, and see how well it generalises.

With the CSAI problem, try in turn **all** possible subsets of 3 courses out of 30.

Use a 3-input, 2-hidden, 1-output layer ANN.



Smaller ANN

This smaller ANN now has only (3+1)*2 +(2+1)*1 = 10 weights to learn, so we might have a chance even with only 20 or so training data.

We could split this into 10 training data, 10 test data.

Then for each selection of 3 courses (from 30):

- Build a 3-2-1 ANN
- Train on the training data
- Test on the test data
- The Error on the test data shows how good a generaliser it is

Compare all the possibilities

So now we will find that for many subsets-of-3 courses, we get lousy predictions of millionaire/in jail (for the previously unseen test data).

But hopefully some of them will generalise well – and we can pick out the best generaliser.

Then we will have two successes:

- The most relevant 3 courses to consider (ignore the rest !)
- And the non-linear combination of the scores on those 3 courses, that combine to give a good prediction.

A problem

There is at least one problem with all this method – anyone spot it?

We are going to have to do lots of choices of 3 subsets out of 30, and then do a whole neural network training routine each time.

How many of these will there be?

30*29*28/6 = 4,060 = rather a lot

Can we cut this down? -- Yes!

Use a Genetic Algorithm

There is a search space of thousands of subsets-of-3, and a ready-made **fitness function** for any one of them –

--- namely the Error on trying to generalise to test data.

(This is the kind of fitness-function one tries to make as **small** as possible).

So why not use a Genetic Algorithm to search through various possible sets-of-3 – and hopefully we can find the ideal set-of-3 with much less than exhaustive search of all the possibilities.

Genetic encoding

There are several different possible ways to genetically encode possible solutions.

A genotype must specify 3 out of the 30 possibilities. One simple way would be this:

Genotype[0] = 5 13 27

Genotype[1] = 11 17 29 etc etc

Then a mutation would change any selection to a new number in range 1-30

But problems …

Encoding problems

Firstly, a mutation might produce a genotype 5 13 13

Secondly, recombination between 2 parents might also produce 5 13 13

Where this is now a selection of 2 different, rather than 3 different numbers. Solutions?

One solution is to put special code in the program, so as to prohibit mutations or crossovers that produce such results.

Another solution

Another, easier, solution is just to ignore the problem!

If an input number is duplicated, then just carry on, and build the ANN with 2 inputs having he same vale. If the results are less fit (the trained ANN generalises badly to test data) then it will probably not have any offspring in the next generation of the GA.

GAs are forgiving

This is one example of how flexible and forgiving GAs are.

Often, rather than forcing the algorithm to generate and test only 'legal' possible solutions through using hard constraints, you can relax and allow all possibilities.

If the illegal ones don't work, then they will be eliminated naturally.

After all, that is what natural evolution is all about.

Another encoding

You could have a different genetic encoding, binary genotypes of length 30, where 0s mean 'ignore this factor (this CSAI course)' and 1s mean 'pick on this one'.

Then if genotypes have three 1s and 27 0s, they will effectively be selecting 3 out of 30.

Similar problems with mutations and recombination – but similar solutions to these problems can be found.

The Outer loop

Split data into 50% train/ 50% test

↓

Initialise at random a GA popn ofgenotypes

↓

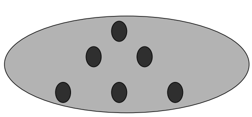
Run a GA for many generations, using mutation and recombination, to reduce errors on generalisation. → **evaluate** ←

↓

Best trained ANN

Middle loop

→ Specification of 3 inputs (3 CSAI courses) to take notice of



Build 3-2-1 ANN

Initialise random wts

Train weights on train data ⇄

← **Calculate error on test data**

Inner loop

→ **Repeat over many presentations of training data**

Present one set of inputs to the ANN

Pass forward thru layers to produce output

Error = difference between Output and Target

← **Back-propagate errors, and then adjust the weights**