GP for Symbolic Regression

Maarten Keijzer

Chordiant Software Inc.

Overview

- What is Symbolic Regression?
- What techniques are in use?
- Issues with SR
  - overfitting
  - parameters
  - size
  - numerics
- Formal description of regression
  - maximum likelihood/maximum posterior
  - The role for priors
- Interpretability

The Regression Problem

- Given a set of input data *x* and a set of desired outputs *t*, find a function *f* such that:

$$t = f(x_1, \cdots, x_n) + \epsilon$$

x1	x2	x3	t
0.26	0.51	0.91	1.13
0.11	0.1	0.44	0.43
0.2	0.54	0.66	0.98
0.55	0.16	0.16	-0.24
0.05	0.99	0.05	0.84
0.95	0.18	0.12	-0.66
0.1	0.37	0.94	1.2

(Non-)Linear Regression

Linear:

$$t = w_0 + w_1 x_1 + \cdots + w_n x_n + \epsilon$$

Generalized Linear:

$$t = w_0 + w_1 f_1(x_1 \cdots x_n) + \cdots + w_n f_n(x_1 \cdots x_n) + \epsilon$$

Feedforward ANN:

$$t = w_0 + \sum_i w_i g(h_{0i} + \sum_j h_{ji} x_j) + \epsilon$$

What's this epsilon thing for?

$$t = f(x) + \epsilon$$

Denotes the *noise* in the measurements t

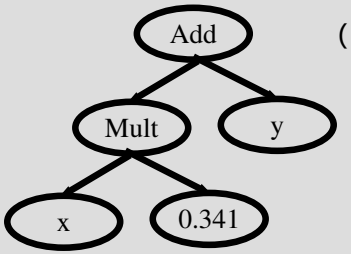
The nature of this noise guides the choice of objective function

We seldomly (read: **never**) expect that we can obtain a perfect fit

Corollary: perfect fit of the 'sextic' polynomial is of no interest

Symbolic Regression

- Instead of finding coefficients for functions
- Find function structure (+ coefficients)




```
(ADD (MULT X 0.341) Y)
```

$$0.341\,x + y$$

What does this buy?

- Automatic variable selection
- Explicit symbolic results
  - Interpretation (grey box?)
  - Easy implementation of resulting expressions
- Freedom to implement non-continuous cost functions
- Multi-Objective

Applications

- Physics
  - empirical equations/differential equations
- Econometrics
  - empirical relations
- Finance
  - trading rules
- Industry
  - process control/identification
- ...

Setting up a run

- Define:
  - Terminal set (*independent variables*)
  - Scheme for handling constants (ERC, mutation)
  - Desired output (*dependent variable*)
  - Function set
    - simple arithmetic
    - sqrt, pow
    - trig
  - Error measure
    - Sum squared
    - Sum absolute
  - Usual parameters (population size, elitism, etc.)

The Standard Representation

- Individuals represented as trees
  - Possibly implemented as pre/postfix strings
  - Subtree crossover
  - Subtree (branch) mutation
  - Node (point) mutation
- Strong elitism
- Largish populations (500-10000)

Machine Language Regression (Nordin & Banzhaf)

- Init registers with variables
- Execute register assembler program

```
x += x;
y *= x;
x /= y;
  ...
return x;
```

Pretty Fast!

Regression through Assembler

- FAST
- Commercially available (Discipulus)
- Potential difficulties with interpretation

Stroganoff (Iba & Nikolaev)

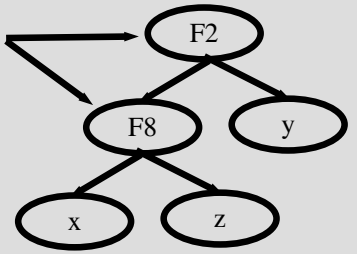
- Use set of polynomial basis functions
- Put them in a tree
- Optimize constants for each level
  - minimize error for subtrees

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2$$

weights are set to zero to create different functions

Stroganoff (ctd.)

coefficients optimized w.r.t. target



GP used to find shape and size

F2 $w_0 + w_1 x_1 + w_2 x_1^2 + w_3 x_1 x_2$

F8 $w_0 + w_1 x_2 + w_2 x_1^2$

Stroganoff (ctd)

- Inspired by GMDH
- Handles constants
- Handles smoothness
- Restricted (though complete) functionality
  - More in line with other methods of regression.
- Does variable selection

Issues

Evaluation

Most 'off the shelf' GP toolboxes iterate over all cases

```
for each case:
   for each node:
       eval_node
```

implies overhead of interpreter incurred for each case and each node

For regression, we often have:
- Fixed set of cases
- No interdependencies between cases

Vectorized Evaluation

So we can do:

```
for each node:
   interpret_node
    eval_func_for_all_cases
```

Overhead of interpreter now independent of number of cases
Additional benefits: pipelining +,*,-

Standard stuff in Matlab/Octave/Mathematica
Usually absent in general GP toolboxes

Overfitting

- Getting a good fit on the training data is *not* good enough:
  - Need testing data
  - Possibly need crossvalidation
- Time-honoured solution
  - Split data in training/validation/testing set
- Other solutions:
  - ensemble modelling
  - add noise (both input and output)
  - use smoothness (Stroganoff)

Using Constants

- Many approaches
  - Koza: ephemeral random constants (ERC)
  - Angeline, Schoenaur, many others: special constant mutation methods:
    - Gaussian, Cauchy, uniform in small range, etc.
  - ERC + Linear Scaling:
    - *f(x) -> a + b f(x)* to minize *squared* error
    - Multiple linear regression: many functions, combined
  - Iba, Nikolaev (Stroganoff): Optimize all constants
    - minimize subtree error
  - Topzy & Punch: gradient descent on (symbolically) differentiated functions

Ephemeral Random Constants

- Initialize with randomly chosen constants
  - from range, or gaussian, or ....
- Don't allow changes to constants (ephemeral)
  - New values can be synthesized by crossover
    - (0.4 + 0.1*2.3)
- Makes constant mutation a structural operator

Mutable constants

- Initialize constants
- Add mutation operators to:
  - change constants using fixed distribution(s)
  - change constant through adaptive distribution
- Idea: small local changes help in finding proper value
- Problem: two optimizations going on at the same time
  - Search for proper structure
  - Search for proper values inside structure
- Difficult to find proper balance

Linear Scaling

Very simple method to improve the fitting behaviour on squared error considerably.

Find a and b, such that:

$$\sum (t - a - b\,f(x))^2$$

is minimal.

$$b = \frac{cov(t, f(x))}{var(f(x))}$$

$$a = \bar{t} - b\,\bar{f(x)}$$

Why Linear Scaling?

- Deterministic & Fast
- Gives upper bound on squared error *var(t)*
- Does not have problems with colinearity
  - (multiple regression does)
- No extra parameters
- Appears to work good (Keijzer), but maybe a bit too good
- Does not tackle 'non-linear' constants
  - Does not seem to matter much (?)

Multiple Regression

- Evolve a number of trees:
  - Find weights using multiple regression techniques
- How many trees?
- Won't we overfit (even more)?

Gradient Descent

- Calculate (symbolic) derivative of error w.r.t. the constant values
- Move constants in direction of gradient
- Assumes relationship between error and constants is locally linear
  - True for ANN: not necessarily true for GP
- Very expensive:
  - Doing a few gradient steps for an individual that gets thrown out later on seems wasteful
  - Balance between effort of finding optimal structure and optimal constants within structure

Size Control

- Many methods to control size have been developed for genetic programming
  - penalty based
  - multi-objective based
  - lexicographic
- Most (if not all) are directly applicable to regression type of problems

Reasons for Size Control

- Cull bloat
  - shorter solutions evaluate quicker
- It is thought that smaller solutions generalize better.
  - Appeal to Occam's razor

Occams Razor

- ***Objects should not be multiplied beyond necessity***
  - What is necessity?
  - If any improvement in error is a *good thing*, Occam does not lead to penalty based parsimony pressure:
    - does lead to lexicographical parsimony pressure

Occams razor as such does not provide justification for balancing size and error

Minimum Description Length (Rissanen)

- Minimize the total length in bits to transmit:
  - The model
  - The exceptions

tree_coding_length + exception_coding_length

tree size

error

Problem: coding bias. True MDL is undecidable

Penalty Functions

MDL-Fitness = $\sum_i (t_i - f(x_i))^2 + \gamma |f|$

How to set the parameter?

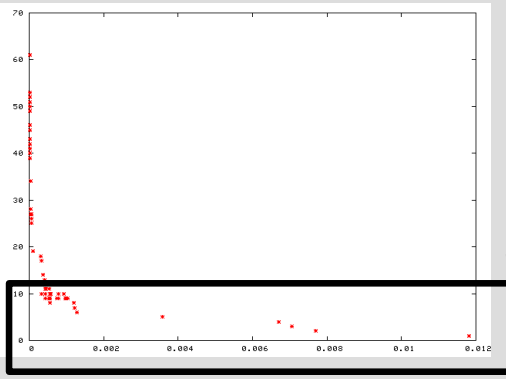
Fixed (trial and error)

low initially, stronger later (Zhang et. al. 1995)

Reports that penalty functions on size often work very well, yet do lead to quite a few 'failed` runs. (Soule & Foster)

Multi Objective

Evolve a front of individuals that uniquely balance size and performance



Choose which one?

Small size, bad performance

Lexicographic Parsimony Pressure

Only prefer shorter solutions when they're equal in fitness (or equal in rank)

True implementation of Occam's Razor: if there's no other reason (necessity) to prefer one model over the other, prefer the shorter one.

Luke & Panait (2002) report that this doesn't work well for symbolic regression type of problems.

Numerics

Floating point arithmetic is imprecise

3.14 =
3.1400000000000001 (as a double)
3.1400001049041748 (as a float)

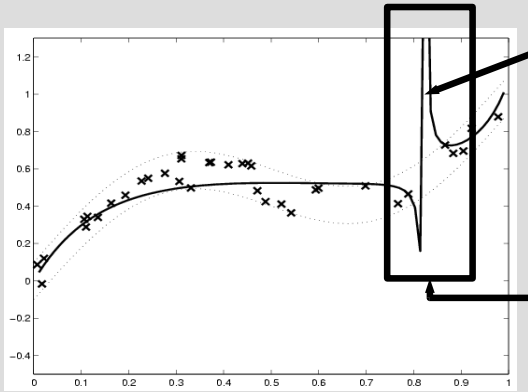
In large expressions, round-off errors can accumulate

```
float b = sin(a); // truncate!
float c = log(b);
```
is **different** from
```
float c = log(sin(a));
```

tip: always check the results produced in another environment:
(for instance: check C/Java code with Matlab/Octave)

More Numeric Issues

Protected operators are a BAD idea!



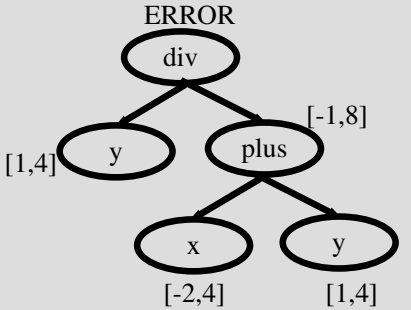
Interval Arithmetic

Solution: Use Interval Arithmetic

```
x   = [-2,4]
y   = [1,4]
x+x = [0,8]
x*y = [-2,16]
x/y = [-2,4]
y/x = error
```

however:

```
x*x = [-8,16]
sqr(x) = [0,16]
```



Measuring the Worth of Expressions

Formal description of relationship between probability, error measures, likelihood, posterior distributions and prior distributions

Likelihood

The Likelihood of our gp-function *f* is the probability that we will have observed targets *t*, **given** our estimation of *f*.

$$p(t|f)$$

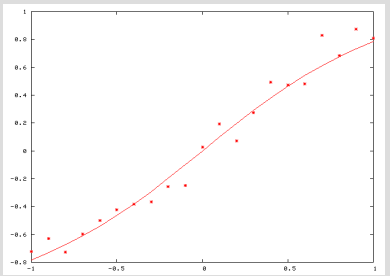
If there is no noise in the problem (i.e., measurements are perfect), the likelihood of *f* is necessarily 1 **iff** $t = f(x)$, and zero otherwise.

In other words. If we know that *t* is noise-free, we will never settle for anything less than a perfect fit.

Likelihood and Noise

If however we have reason to assume that the noise is normally distributed (and this is the maximum – least knowledge -- entropy distribution), our likelihood function will become:

$$p(t|f) = \prod_i \frac{1}{\sigma\sqrt{2\pi}} \exp[-(t_i - f(x_i))^2 / 2\sigma^2]$$



noise was 0.1
p(t|f(x)) = 4%

Log-Likelihood / Squared Error

Maximizing Likelihood is equivalent with minimizing negative log-likelihood. After taking the logarithm and deleting constants, we end up with

$$argmax_f\, p(t|f) = argmin_f \frac{1}{2} \sum_i (t_i - f(x_i))^2$$

$$\prod_i \exp[-(t_i - f(x_i))^2]$$

Thus: family of squared error functions assumes **noise** is distributed normally.

Other Error Measures

Sometimes you do have an idea of the nature of the noise.
If you expect outliers (possibly caused by human intervention), you might want to consider one of these **robust** measures

absolute error $|t-f(x)|$ --- double exponential distribution

Lorentzian $log(1+(t-f(x))^2)$– Lorentzian (Cauchy) distribution

Pearson limit VII $log(sqrt(1+(t-f(x))^2)$ --- Pearson limit distr.

For 1/0 Classification:

cross entropy $t\ log(s(f(x))) + (1-t)log(1-s(f(x)))$ --- Binomial Distribution
s is the sigmoid function

However

In general we are not interested in the probability of observing the data given the (correctness of the) function.

We want the probability (correctness) of the function given the data itself.

Maximizing this probability is called *maximizing the posterior.*

Bayes Rule

posterior likelihood prior

$$p(f|t) = \frac{p(t|f)\,p(f)}{p(t)}$$

normalizer, generally unknown/uncomputable

Maximize Posterior

maximizing posterior equals maximizing likelihood times prior

$$argmax_f\, p(f|t) = argmax_f\, p(t|f)\,p(f)$$

The method of maximum likelihood assumes that the *prior* is uniform, i.e., all functions are equally likely

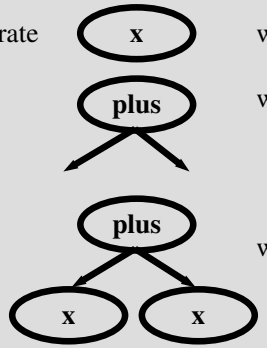
But how likely is a given GP function a priori?????

A Prior for GP

Suggestion: take the probability of generating the function at random as the prior. For instance under *grow* initialization

T = {x} and F = {+}

generate



with probability 0.5

with probability 0.5

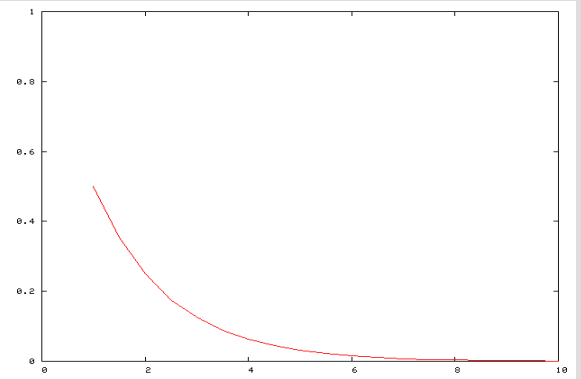
with probability 0.125

$$p(f) = 2^{-|f|}$$

A Prior for GP

$$p(f) = c^{-|f|}$$

c depends on terminal/function set



Maximum Posterior for GP

$$argmax_f\, p(f|t) = argmax_f\, p(t|f)\, p(f)$$

$$= argmax_f \prod_i \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(t_i - f(x_i))^2}{2\sigma^2}} \cdot c^{-|f|}$$

$$= argmin_f \sum_i \frac{(t_i - f(x_i))^2}{2\sigma^2} + |f| \log(c)$$

$$= argmin_f \sum_i (t_i - f(x_i))^2 + \gamma |f| \qquad \gamma = \log(c)\, 2n\sigma^2$$

Exponential Prior on size = Minimum Description Length

$$argmin_f \sum_i (t_i - f(x_i))^2 + \gamma |f|$$

$$\gamma = \log(c)\, 2n\sigma^2$$

Coding bias induced by primitive set (can be more complex, for instance 'speed')

Intrinsic problem noise (generally unkown, can also be more complex; per case uncertainty, weights)

Another Prior

Instead of incorporating *structural* information (size) in the prior, it might be possible to incorporate more 'functional' information

Tikhonov Regularization:
Weighted sum between:
Error (likelihood)
Smoothness (prior)

Smoothness defined as sum of squared derivatives.

Usually only second derivative is considered

Smoothness Prior

$$argmin_f \sum_k (t - f(x))^2 + \gamma \sum_k \sum_i \sum_j \left(\frac{\partial^2 f}{\partial x_j x_i}\right)^2$$

Prefers 'smooth' changes over wildly varying values

Guiding principle in neural network optimization (weight prior)

For General GP, very expensive; feasible with polynomials

Currently only used in Stroganoff (Nikolaev 2000)

Variance Reduction Ensemble Methods

- Bias/Variance tradeoff:
  - Inflexible methods will not have good error (high bias)
  - Flexible methods will not have good error (overfitting, high variance)
- Flexible methods can be made more reliable by:
  - training many models on different parts of the data
    - bootstrapping/crossvalidation
  - add the predictions together (bagging)
- Now we have many trees instead of one!

Interpretation

```
f(x0) = -0.00363108+0.262845*(sin(sqrt(x0)) *
((sin(sin((cos(x0) * sqrt((5.160156 +
sqrt(sqrt((x0 * 7.222656)))))))) * sin((x0 +
x0))) + ((sin(sqrt(x0)) * (-sin((x0 + x0)))) *
sqrt((sqrt((exp((((-sin((x0 + x0))) *
(sin(sqrt(x0)) * (sin(7.652344) *
(sin(7.652344) * (sin(sqrt(x0)) * x0))))) *
(sin(sqrt(x0)) * (sin(sqrt(x0)) *
(sin(sqrt(x0)) * (-sin((x0 + (sin(sqrt(x0)) *
x0)))))))) + x0)) + x0)))))
```

Interpretation

The manual approach

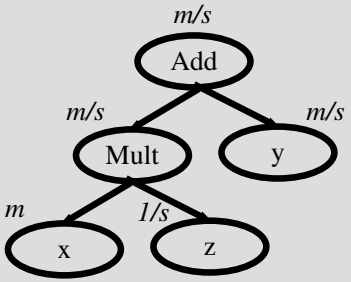
Pick formula apart, simplify
analyze pieces,
throw away pieces that are not necessary
put pieces back together again

Often works surprisingly well
Yet many people are lazy enough to use black boxes

Can GP make life a bit easier?

Units of Measurement

- Physical Measurements often have Units
- Units form a more-or-less formal system
- Can GP use this?



Two Approaches (Keijzer 2002)

- Strongly typed:
  - Search only dimensionally correct expressions
    - Needs new GP system to handle these constraints (ALP)
  - Fairly involved
- Coercion
  - Normal GP
  - Calculate amount of dimension error
  - Use this as a second objective in a MOO search

Coercion approach also works when units are sparse and incomplete

Units of Measurement example result

GP-induced sediment transport formula:

$$c_b = 10^{-5} \frac{(u_f' - w_s)}{u_f + u_f'} \left(1 + 100 \frac{u_f' w_s}{g\, d_{50}}\right)$$

Correct balance of units,
competitive with existing equation,
simpler

Outlook: Matrix Arithmetic

In some fields large quantities of data can be transparently manipulated with matrix expressions

Image processing: `convolve(mask, picture)`

Time series: `max(high[1:2:20])`

Bio-informatics: `dist(protein, protein)`

Is there space for strongly typed/coercion Symbolic Regression in such domains?

Conclusions

- Symbolic Regression is an interesting approach
  - Still needs significant work
  - Is productive though
- Interpretability allows for 'grey-box' model development
  - This does need human intervention
  - Not 'off the shelf'
- Fitting behaviour still ill-understood
  - More informed priors?
  - Smoothness?