Breeding Expression Trees for Symbolic Regression

Ben Wiley and **Jackson Spell**
{bewiley,jaspell}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

The abstract should be between four and seven sentences long. Introduce the problem you are studying. Describe what you did. Summarize your results — what did you discover, what is the main take-away message?) Basically, you're trying to sell your paper to the reader, so be brief and to the point. **Do *not* include any citations in the abstract.**

1 Introduction

Linear, polynomial, and other forms of regression analysis are commonly used in the field of statistics in order to fit a function to a data set. Produced functions can be used to predict behavior at arbitrary input values. Typically, regression analysis is used upon data sets for which the approximate function *type* (e.g. linear, quadratic, etc.) is known. Traditional regression analysis is not well-suited, however, for data sets which approximate an unknown function type.

One tried solution for such scenarios is symbolic regression by way of a genetic algorithm — an algorithm that generates sets of candidate solutions, randomly splices together traits from pairs of fit "parents" into "children" over successive generations, and occasionally mutates pieces of those candidates, until a strongly fit solution is found.

In our paper we introduce expression trees as a means of representing functional expressions in such a way that they can be easily spliced by a genetic algorithm. We discuss our own implementation of symbolic regression for two known data sets representing unknown functions, inspired by the work of John Koza. (**?**)

2 Background

Describe any background information that the reader would need to know to understand your work. You do not have to explain algorithms or ideas that we have seen in class. Rather, use this section to describe techniques that you found elsewhere in the course of your research, that you have decided to bring to bear on the problem at hand. Don't go overboard here — if what you're doing is quite detailed, it's often more helpful to give a sketch of the big ideas of the approaches that you will be using. You can then say something like "the reader is referred to X for a more in-depth description of...", and include a citation.

Alternately, you may have designed a novel approach for the problem — your own algorithm or heuristic, say. A description of these would also be placed in this section (use subsections to better organize the content in this case).

Enumerating

Create bulleted lists by using the `itemize` command (see source code):

- Item 1
- Item 2
- Item 3

Create numbered lists by using the `enumerate` command (see source code):

1. Item 1
2. Item 2
   (a) Sub-item 2a
   (b) Sub-item 2b
3. Item 3

Formatting Mathematics

Entire books have been written about typesetting mathematics in LaTeX , so this guide will barely scratch the surface of what's possible. But it contains enough information to get you started, with pointers to resources where you can learn more. First, the basics: all mathematical content needs to be written in "math-mode" — this is done by enclosing the content within $ symbols. For example, the code to produce $6x + 2 = 8$ is `$6x + 2 = 8$`. Note that this is only good for in-line math; if you would like some stand-alone math on

a separate line, use *two* $ symbols. For example, `$$6x + 2 = 8$$` produces:

$$6x + 2 = 8$$

Here are various other useful mathematical symbols and notations — see the source code to see how to produce them.

- Sub- and super-scripts: $e^x, a_n, e^{2x+1}, a_{n+2}, f^i_{n+1}$
- Common functions: $\log x, \sin x$
- Greek symbols: $\epsilon, \phi, \pi, \Pi, \Phi$
- Summations: $\sum_{i=0}^{i=100} i^2$
- Products: $\prod_{i=0}^{\infty} 2^{-i}$
- Fractions: $3/2$

$$\frac{x+5}{2 \cdot \pi}$$

Other useful resources:

- Find the LaTeX command you're looking for by drawing what you want to produce[1]:`http://detexify.kirelabs.org/classify.html`
- Ask others: `http://tex.stackexchange.com/`
- Every LaTeX symbol ever: `http://tinyurl.com/6s85po`

3 Experiments

This experiment was designed to use a genetic algorithm to adapt a population of expression trees to correctly identify an unknown function. A population of 1000 expression trees were adapted through 50 generations, with the goal of matching the goal function.

Genetic crossover was achieved by probabilistically combining expression trees based on their fitness functions. The fitness score of each tree is calculated as the reciprocal of the total error between the expression tree and goal function, evaluated at a large number of points. The reciprocal is taken in order to give more accurate trees, with lower sum errors, higher fitness scores. Trees were then chosen randomly according to a probabilistic distribution weighted by the fitness scores, with trees replaced back into the population after each crossover.

One of the main concepts behind genetic algorithms is that parents in the algorithm pass on traits recognizably to children. By choosing the most fit expression trees to reproduce, the algorithm should, after a number of generations, converge on a locally optimal solution. In order for preferable characteristics from parent expression trees to be transferred to child trees, the crossover operation chose a random node from each tree, then swapped the subtrees extending from the chosen nodes. Child trees therefore were composed of partial sections of each of the parent trees. Because trees were chosen with replacement, and combine randomly during each crossover, populations may have multiple identical trees or trees made by combining the same two parent trees at different nodes.

After each crossover operation, a portion of the population was mutated in order to introduce random variation. For a given expression tree, the mutation operation randomly selected a node and altered the value or mathematical operation at that node based on the original value. To avoid damaging the behavior of the tree too much, thereby potentially lowering the accuracy of the algorithm, the mutated operation or value was chosen to structurally match the original node. Leaf nodes were only replaced with terminals: randomly chosen constants or appropriate variables. Nodes originally containing binary operations (+, −, ×, /) were only replaced with other operations of that nature.

It is of note that none of the nodes of the expression trees contain the power operation, e.g. x^2. This was a code optimization choice made because power operations can be created through multiplication of appropriate variables. Removing power operations decreased the chance that an expression tree would become arbitrarily complicated through high powers, instead favoring low integer powers – x, x^2, x^3.

4 Results

In order to help the algorithm converge on the correct solutions, the parameters guiding the behavior of the algorithm were tailored to increase population diversity

5 Conclusions

In this section, briefly summarize your paper — what problem did you start out to study, and what did you find? What is the key result / take-away message? It's also traditional to suggest one or two avenues for further work, but this is optional.

6 Acknowledgements

This section is optional. But if there are people you'd like to thank for their help with the project — a person who contributed some insight, friends who volunteered to help out with data collection, etc. — then this is the place to thank them. Keep it short!

[1]Thanks to Dr. Kate Thompson for pointing me to this resource. Also, this is how you create a footnote. Also, don't overuse them — prefer citations and use the acknowledgements section when possible. I usually only use footnotes when I want to link to include a pointer to a web site.