CS492B
Search Based Software Engineering, Autumn 2016

Coursework #2: Genetic Programming
Due by 23:59, 03 November 2016

1 Aim

To implement genetic programming and to solve a benchmark symbolic regression problem. Training data will be publicly available; grading will be based on a hidden set of test data.

2 Symbolic Regression

The goal is to find a expression that fits the given dataset with as small error as possible. For example, suppose your training data set includes tuples (x, y), from which you want to learn the best model f such that $y = f(x)$ explains the given data set, as well as unseen test data set, as precisely as possible. If the training data consist of $X = \{1.2, 2, 3\}$ and $Y = \{3.1, 4.6, 6.8\}$, and the test data set consist of $X' = \{6, 5\}$ and $Y' = \{13, 10.5\}$, one possible symbolic regression model would be $y = 2x + 1$. Figure 1 shows the result of symbolic regression.

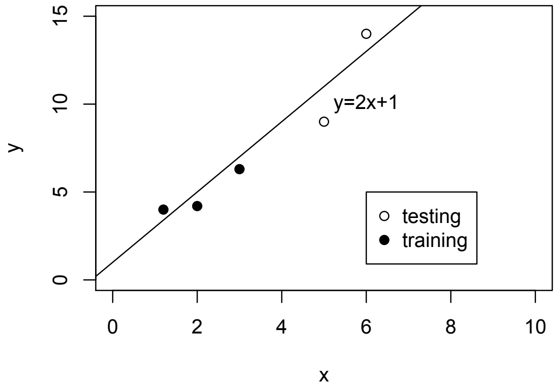

Figure 1: Symbolic regression example

With Genetic Programming, you can build candidate expressions for f, and evaluate them using Mean Square Error, which is calculated as follows:

$$MSE(f, X, Y) = \frac{1}{|X|} \sum_{x_i \in X, y_i \in Y} (f(x_i) - y_i)^2$$

3 Dataset

The symbolic regression dataset we are using contains 57 input variables $(x_1, \dots, x_{57})$, and 1 output variables, y. The training dataset contains 747 rows, and is available from . Using this dataset, evolve a symbolic regression model.

Implement two programs. The first program (e.g. `train.py` or `train`) should be an implementation of GP that takes a `.csv` file containing the training data as input, and prints out the evolved expression using Reverse Polish Notation. The second program (e.g. `test.py` or `test`) should take two inputs: the evolved RPN expression in one string, and a `.csv` file containing the test data. Subsequently, it should evaluate the given RPN expression on the test data and print out the MSE.

You can use the following ternimal and non-terminal nodes for GP, and corresponding symbols when printing out the evolved expression in RPN:

- **Terminals:** `x1`,. . ., `x57`, as well as any floating point constant numbers
- **Unary Operators:** ∼ (unary minus), abs, sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, exp, sqrt, log
- **Binary Operators:** +, -, *, /, ˆ (power)

For example, $x_{24}^2 - sin(-2x_3)$ would be represented by `x24 2 ˆ 2 ∼ x3 * sin -`.

```
> python training.py train.csv
...
x24 2 ˆ 2 ˜ x3 ∗ sin −
> python test.py ”x24 2 ˆ 2 ˜ x3 ∗ sin −” test.csv
7.27557544149e+79
```

4 Competition

Similarly to TSP, there will be a an online leaderboard (`http://coinse.kaist.ac.kr:8000`). You can submit a text file (`.txt`) that contains the evolved RPN expression in a single line, and the leaderboard will calculate the MSE based on the hiddn test dataset. The person who has submitted the smallest MSE by the due date will win a **prize**. Later, we as a group will have a chance to go through interesting solutions. Note that **the competition result is not directly linked to the grades**, as marking will consider other aspects, such as report/documentation, implementation quality, and the novelty of the optimisation idea.

5 Deliverables

Each person should submit the following deliverables by the submission deadline:

- **Implementation**: source code of your symbolic regression solver, self-contained in the code directory (see below).
- **Report**: include a written report that contains detailed descriptions of how you approached the problems. Describe the optimisation you have implemented in as much detail as possible. There is no page limit.

For ease of marking, follow the following directory structure, and submit a zip file containing the top level directory, through KLMS.

```
[your student number]
├── report.pdf ........................... Your report documenting your implementation
└── symbreg.............................GP-based symbolic regression and dependencies
```

6 Guidelines

- Examples are given with `python` only for illustrative purposes; you are free to choose any programming language.
- **BUT** you do have to implement the optimisation algorithm yourself; do not use pre-developed frameworks.
- Make sure your submission is self-contained. It should not depend on any file outside the submitted directory, such as files on your own hard drive or online. We expect the solvers simply to work out of the box. If you use Windows machines (or, in fact, any machine of your own), we strongly recommend that you test the submission on a separate machine, in order to test whether it is relaly self-contained.