Project II: Symbolic Regresion

This document describes the second project for the course, in which you will apply genetic programming methods for symbolic regression of a function.

Make sure to **read all of the information** present in this document.

1 Introduction

The goal of this project is to design a program which is given a set of data points of the form (x, y), and finds a function that such that $y \approx f(x)$ for all of the given data. You are expected to implement the whole project yourselves, as well as any mutation and crossover operators that might help to obtain the best results possible.

For this project, **no experimental analysis is needed**: you must only provide an explanation of your approach in a **short technical report**, as well as the **results for some test data** which will be provided at a later date, as well as **some information about the execution of the tests**, such as the number of tries, population size, and number of generations.

You will work in **groups of 3 students**, to implement the algorithms and write the technical report. The project deliverables include the **technical report** detailing your work, as well as the **code** for the entire project. The evaluation will be based on the **quality of the approaches**, and will be judged mainly according to the report. The contribution of each member of the group will be **individually tested during the exam** by a few questions on the project details.

The tentative deadline is **April 13**.

This document contains information about the project and what is expected from you, including:

Section 2: information about the problem to be studied;

Section 3: advice on the methodology and how you should conduct your work, including some ideas to get you started;

Section 4: how to structure the report and what to submit at the end of the project;

Section 5: how the work will be evaluated.

2 Symbolic Regression

Symbolic Regression involves finding a mathematical expression that best fits a given set of data points, without prior knowledge of the form of the expression. Formally, we are given a set of data points $\{(x_1, y_2), (x_2, y_2), \ldots, (x_n, y_n)\}$, and we want to find a function such that $f(x_i)$ is close to y_i for every $i \in [n]$.

For this project, we will consider only functions that can be obtained using the basic operations $\{+, -, *, /\}$, as well as constants and constant powers of the variable x ($x^2, x^3, \ldots$), represented as `x**a` where `a` is an integer. **No other operations are allowed.**

For instance, we might approximate the data points $(0, 1), (0.2, 1.15), (0.5, 1.41), (0.8, 1.74)$ as $f(x) = 1 + 16x/23 + 10x^2/37$, which produces $f(0) = 1$, $f(0.2) \approx 1.150$, $f(0.5) \approx 1.415$, $f(0.8) \approx 1.729$.

Our goal is to find a function that approximates the data points well, but is not too convoluted, that is, does not use an unnecessary number of operations.

2.1 Data

Further instances will be provided on the Blackboard webpage. The filename will follow the structure `sr_poly_4.txt`, where:

- `sr` simply identifies the problem;
- `poly` is a keyword describing the type of instance, in this case a polynomial; types are described in a `README` file inside the archive containing the files
- `4` is an identifying number of the instance, in this case instance number 4.

Each instance follows the following format: in the first line, a single integer n specifies the number of data points to follow; the following n lines contain two real numbers separated by a space, corresponding to x and y respectively. The instance above would thus be represented as:

```
4
0 1
0.2 1.15
0.5 1.41
0.8 1.74
```

The output should be as a single string that represents the function as a Python string and using its priority of operation. Order of elements does not matter unless when relevant for computation, and white space is ignored. Each of the lines below represents an equivalent possible output for the example function given above:

```
1 + 16*x/23 + 10x**2/37
1 + 10/37*x**2 + 16/23*x
1 + (16/23)*x + (10/37)*(x**2)
1 + 2*x*(8/23 + x*5/37)
```

3 Methodology

The objective of this work is to create an algorithm to do symbolic regression as described above. This algorithm must use genetic programming techniques to design a function to approximate the input data, and produce output as described. You are encouraged to experiment with different mutation, recombination, and selection methods to obtain the best possible results.

Your implementation should allow for easy selection of different operators and selection methods, as well as parameters such as population size and number of iterations. For that purpose, we recommend the library argparse, though you are free to use whatever method you prefer.

3.1 Some Tips and Ideas

- Although you will use strings to represent the phenotype of the solution (since the output is a string representing a Python expression), the best genotype for this project is not a string;
- A good fitness function for comparing sets of data points is the *mean squared error* function:

$$\mathrm{mse}(f) := \sum_{i=1}^{n} \big(f(x_i) - y_i\big)^2;$$

- Think about what makes good mutation and recombination operators for this problem;
- To reduce bloat and make your functions more compact, try adding a term to the fitness function that penalizes long functions.

4 Report and Deliverables

The project submission consists of two parts:

1. A written report, describing your approach and the results for the test instances;
2. The code you wrote along with instructions on how to run it.

4.1 Report

For the report, you are expected to write a structured document, 5–10 pages long, describing the entirety of your approach. It should contain the following sections:

- Introduction: describing the problem and a high-level overview of your approach.
- Methodology: describing in detail the approaches used, how they were modified for the problem, and in general everything that bring value to your work;
- Results and discussion: the results obtained for the test instances, as well as all of the relevant information about the execution (repetitions, parameters, etc), as well as any comments you might want to add about your results.

Formatting guidelines: Use headings and subheadings to organize information logically; ensure consistent formatting, such as font size, spacing, and margins; and incorporate visual elements like charts, tables, or bullet points to enhance readability.

Before submitting, make sure to proofread the report carefully to maintain clarity, coherence, and accuracy throughout.

4.2 Deliverables

At the end of project, you must submit two files: the report in pdf format, and a zip archive containing your code and all other useful data. These files must be submitted on Moodle before the deadline.

Your code should be well-documented (using comments) and all of the required files must be present in the zip archive, along with instructions on how to execute it (in a file named `README`).

5 Evaluation

The projects will be evaluated based on the following criteria:

- Implementation of the algorithm;
- Choice of operators, selection, and parameters;
- Description of the approach in the report;

- Results on the test inputs;

And to a lesser extent:

- Writing quality and structure of the report;
- Code quality and documentation.

Each member of the group must be capable of explaining any part of the code, discuss the experiments that were performed, and be aware of the content of the report.

Individual competence will be tested during the exam with a few simple questions about the functioning of your project. To be clear, these questions do not test the course material, but rather if you fully understand your own project, and serve solely to check that all members have good working knowledge of the project.