GeneticDF
A genetic programming-driven data fitting method

Matteo Scucchia

matteo.scucchia@studio.unibo.it

October 2021

GeneticDF is a personal implementation of a data fitting method using genetic programming, inspired by the paper *A Genetic Programming-Driven Data Fitting Method* [2] by Hao Chen *et al.* It is realized for the exam of the course of Intelligent Robotic Systems.

Contents

List of Figures

1 Analysis

Since the project is inspired by this *paper*, in this section only the topics useful for understanding the implementation part are explained. For a complete analysis of the problem, please refer to the paper.

1.1 Functions and Terminals

Functions and Terminals are the nodes of the IE-Trees.

Functions are binary operators, and they corresponds to the nodes of the trees, so they have two children: a left and a right sub-tree. In the project, three types of nodes are used:

- **AddNode**, that performs the addition operation between the two children;
- **SubNode**, that performs the subtraction operation between the two children;
- **MulNode**, that performs the multiplication operation between the two children;

Terminals are the leaves of the tree, so they don't have children, they only return a value. In the project, three types of terminals are used:

- **ConstNode**, that returns a constant value;
- **InputNode**, that is evaluated with an input value;
- **GaussianNode**, that is evaluated with an input value, and it returns the value of the gaussian function with at the given the input, according to its mean and a its variance;

1.2 Choice of technology

To implement the project, the Scala language [3] was chosen, because it smartly combines object-oriented programming and functional programming. All the project was developed with a particular attention to the use of a declarative and functional programming style, exploiting recursion and avoiding side effects, as a personal challenge. Furthermore, recursion is very effective when working with data structure such as trees.

2 Design

In the following section the design of the system is described. For a greater clarity, UML diagrams are shown.

2.1 Design of the IE-Trees

Figure 1 shows the UML class diagram of the IE-Tree structure, with the nodes identified in 1.1.

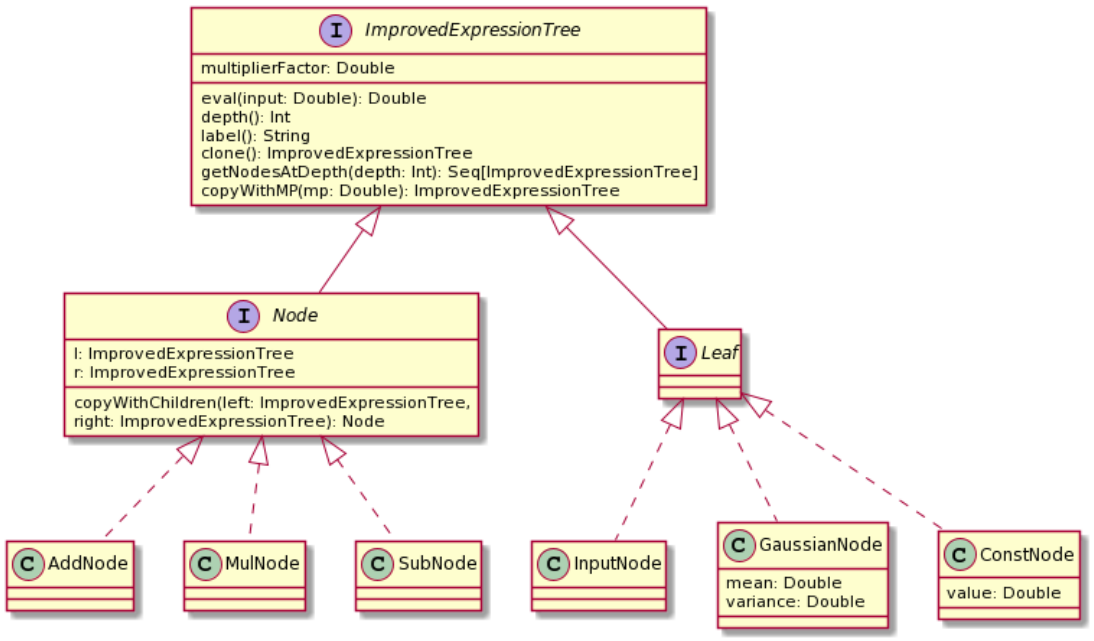

Figure 1: Class diagram for the IE-Tree.

2.2 Design of the genetic programming cycle

Figure 2 shows the UML class diagram of the genetic programming components. These entities are modeled as singletons and they implements pure functions: they don't have an inner state.

In the GPManager object, the genetic data fitting cycle is implemented. In the computation it obviously uses the MutationManager object, that implements a function to carrying out a mutation in a given tree and returns the offspring, and the CrossoverManager object, that implements a function to perform crossover between two given trees, and returns the obtained offsprings.

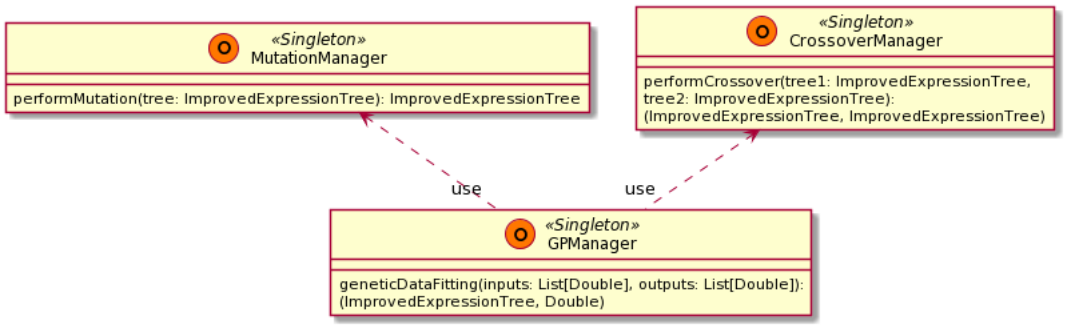

Figure 2: Class diagram for the genetic programming objects.

3 Implementation

This section provides a brief explanation of the implementation. The implementation was made following the indications of the reference paper, but some changes have been introduced. All the code is well documented, so just salient implementation choices are explained here.

The things implemented in a coherent way with the paper are:

- the creation of the trees;
- mutations, differentiating between GaussianNode and other nodes;
- crossover, using the two strategies explained in the paper;

3.1 Multiplier factor

Each IE-Tree (nodes and leaves) has a multiplier factor that is generated randomly. In the most programming languages, and in Scala too, the random double values are generated between 0 and 1, that is obviously a very small range. In the project the values generated by the random function are multiplied by 10, in order to bring the range between 0 and 10, so all the multiplier factors of the nodes are between this range. It is possible to modify this value and observe the results.

3.2 Competition and selection

The competition between new offsprings and old specimens is not implemented as explained in the paper. Contrary to the paper, offsprings does not compete directly with the parents by which they are generated, but with the entire population. Between the

total population, composed by new offsprings and old speciments, only the best *POPULATION_SIZE* elements are selected. An IE-Tree is considered better than another if it has a lower weighted error, that is the weighted sum between the error and the complexity of the tree

$$weighted\_error(e, c) = \alpha \cdot e + (1 - \alpha) \cdot c. \tag{1}$$

where e is the absolute error, c is the complexity computed as the number of nodes of the tree and α is an hyperparameter to weigh error and complexity in a different way and it is set to 0.99. It is possible to modify this value and observe the results.

4 Results and evaluation

The project respects the functional requirements of genetic programming. To verify such thing, in the Main object of the project it is possible to set a list of inputs and a list of outputs and launch the program in order to regress the function and observe the results. Initially, to verify the correct behaviour of the trees' functionalities, even a few formal test using the ScalaTest suite [1] were implemented.

5 Conclusions

In this project, genetic programming was explored an applied to the problem of data fitting. I found it a very interesting argument and i really had fun in the implementation of the project. I'm really satisfied of the final result, not only because it works, but because I think it is designed in a way that is easily expandable and implemented in an elegant way, exploiting the power of functional programming to make it succinct.

5.1 Critical issues

There aren't critical issues in the project. One thing worth noting is that a GaussianNode, as explained in the paper as well as in the implementation, cannot mutate into another type of node, while other types of node can mutate into a GaussianNode. For this reason, an overabundance of GaussianNodes often occurs in trees.

5.2 Future Works

There are some features and functionalities that could be implemented as future works. First, new nodes can be added, such as LogNode and ExpNode, that are the logarithmic

and the exponential node respectively. Different functions to compute the error could be introduced, for now only the absolute error is provided, but also the squared error and the Huber Kernel can be easily added to observe the results. Finally the selection as explained in the paper can be implemented and one could choose between this strategy and the one already available.

References

[1] Inc. Artima. Scalatest.

[2] Hao Chen, Zi Yuan Guo, Hong Bai Duan, and Duo Ban. A genetic programming-driven data fitting method. *IEEE Access*, 8:111448–111459, 2020.

[3] Switzerland École Polytechnique Fédérale Lausanne (EPFL) Lausanne. The scala programming language.