Predicting the execution time for compilation flag configurations using supervised learning algorithms

1 Description

The compiler flag selection problem refers to the task of selecting an appropriate combination of the compilation flags for a program in such a way that some measure of efficiency, e.g., the running time of the program or its size, are maximized [1, 2, 3, 4, 5, 6]. A possible approach to this problem is to learn a regression model that, without the actual compilation of the program, allows to predict how long time will take executing the compiled code.

2 Objectives

The goal of the project is to learn a set of regression ML algorithms to predict, using different combinations of flags, what will be the expected running time of a program for a fixed set of test cases (inputs to the program). The students can decide whether to create their own dataset with configurations of the compilation flags for a given program (or set of programs) or, may use C++ code already available for publication [4][1]. The C++ code will only be used to generate the training and test sets, i.e., possible configurations of the flags and the actual running times of the executions for these sets.

The student should: 1) Generate the training and test sets. 2) Define and learn the regressors (at least three regressors). 3) Design the validation method to evaluate the accuracy of the proposed classification approaches.

As in other projects, a report should describe the characteristics of the design, implementation, and results. A Jupyter notebook should include calls to the implemented function that illustrate the way it works.

3 Suggestions

- Check the description of the flag optimization problem as presented in [4].
- See example notebook for work projects.
- Choose as a measure of accuracy the least square difference between the actual time and the time predicted by the regressors.
- See notebook with example of linear regression algorithm presented in the first laboratory.
- Implementations can use any Python library.
- If classes are not well balanced you may use performance measures different to the accuracy.

[1] This code can be requested from the authors

References

[1] Amir Hossein Ashouri, Giovanni Mariani, Gianluca Palermo, and Cristina Silvano. A Bayesian network approach for compiler auto-tuning for embedded processors. In *Embedded Systems for Real-time Multimedia (ESTIMedia), 2014 IEEE 12th Symposium on*, pages 90–97. IEEE, 2014.

[2] Prathibha A Ballal, H Sarojadevi, and PS Harsha. Compiler optimization: A genetic algorithm approach. *International Journal of Computer Applications*, 112(10), 2015.

[3] David Branco and Pedro Rangel Henriques. Impact of GCC optimization levels in energy consumption during C/C++ program execution. In *Scientific Conference on Informatics, 2015 IEEE 13th International*, pages 52–56. IEEE, 2015.

[4] Unai Garciarena and Roberto Santana. Evolutionary optimization of compiler flag selection by learning and exploiting flags interactions. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, GECCO '16 Companion, pages 1159–1166, New York, NY, USA, 2016. ACM.

[5] PJ Joseph, MT Jacob, YN Srikant, and K Vaswani. Statistical and machine learning techniques in compiler design. *The compiler design handbook, optimization and machine code generation. CRC Press, Boca Raton*, 2008.

[6] Eun Jung Park. *Automatic selection of compiler optimizations using program characterization and machine learning*. PhD thesis, University of Delaware, 2015.