Comparison of performances of ML-Algorithms in the estimation of the execution time of non-parallel Java programs

I. M. Mihindu Pramantha De Ranasinghe[1]* and L. Munasinghe[2]

[1]Sysco LABS Technologies (Pvt) Ltd, Colombo 03, Sri Lanka.
[2]Software Engineering Teaching Unit, University of Kelaniya, Sri Lanka

*Corresponding author: mihinduranasinghe@ieee.org

https://orcid.org/0000-0002-1855-0305

Received: 25.08.2022 Revised: 02.02.2023 Accepted: 15.02.2023 Online: 15.05.2023

Abstract Performance enhancement of a computer program is an important aspect of today's world. Developers produce programs, and there is a lack of accurate methods for predicting the execution time of a computer program prior to its execution in an executable environment. Predicting the execution time of a particular program before execution would be great for developing the program with the highest performance efficiency and with the lowest execution latency. This research introduces a Machine Learning based solution to predict an execution-time-based label for a given computer program. There are three main types of parameters in a computer program that affect the execution time, such as Static Code Features, HTTP Calls, and the Hardware Performance of the execution environment. In this research, the Machine Learning model was trained for the parameters of the above types (Programs with Static Programs & HTTP calls) by executing them on a fixed hardware infrastructure execution condition. We analyzed the number of if conditions, methods, breaks, switches, loops, nested-loop-depth, frequencies, and the behaviour of HTTP calls, kind of features of a computer program in order to generate an accurate execution time complexity prediction label of a computer program. Further, in the collected data set, the most prominent feature which affects the complexity among the features that we considered is the number of HTTP calls and nested loop depth, followed by loops. Accuracy Score, Precision, Recall, and F1 Score values of the ML model were generated for the traditional classification algorithms such as Decision Tree Classifier, K Nearest Neighbor Classifier, Random Forest Classifier, Naive Bayes Classifier, Support Vector Classifier, and MLP Classifiers in order to verify the effectiveness of the model. The best accuracy score was achieved with an overall 88% by using the approach of Random Forest. The findings of this research can be optimized for implementing an IDE plugin or a developer tool that can predict the exact execution time of a given computer program live by integrating the specifications of the execution device. It will help developers in terms of optimizing a particular computer program and developing it for a minimum execution latency and enhancing the performance of the program.

Keywords: Complexity, Execution, Machine Learning, Performance, Prediction

Introduction

Optimizing the Performance of computer programming is one of the modern trends in the world. Every research and development team is working on the performance improvement of their own programs and languages, especially when it comes to cloud-native software development. This is where our intention comes into the picture to introduce a Machine Learning based model to predict the execution time of a computer program. This prediction makes it easier for developers to optimize the program to meet the best performance of the program to be executed.

Theoretically, there are a number of ways of calculating the complexity of a computer program. For example, the master theorem is effective for calculating the run-time complexity of divide-and-conquer type problems. It is constrained to that type of problem, and there are several constraints on automating the process. Mathematically it is not practical to find a universal function to compute the complexity of all types of programs (Park, 1993). Therefore, we wanted to come up with a Machine Learning based solution in order to learn the internal code structure effectively. The ultimate goal is to save time and resources and make the most effective and efficient programs which meet the best

performance by training ML algorithms by feeding historical execution data.

The deep learning community has recently done a lot of research on programming codes. Some of them, along with their techniques, the dataset used and their results have been listed as follows.

The study Priya et al. (2011), focuses on addressing how long a Machine Learning Application would take to execute. Here it introduced an approach for predicting processing time specifically for static ML tasks. The study was conducted using 78 publicly available data sets, 6 ML algorithms, and 4 meta-regressors. In this research, they have assumed that the Machine Learning Tasks do not contain any HTTP Calls. There are several independent variables that have been considered against the execution time in different hardware performance conditions, such as the Log of the number of instances, attributes, classes, Proportion of continuous attributes, Normalized class entropy, and average absolute between target, attributes, and correlation of all pairs of attributes. In addition to those parameters, hardware resource usages such as Memory and CPU.

Predicting workflow execution times is relevant, and the approach can be used as a learning tool in our research. The study Nadeem & Fahringer (2009) experiments with the workflow estimation time prediction in a cloud environment. These studies benefit workflow users in making different decisions on workflow execution time prediction.

There is a mathematical model found which describes a way to predict the execution time of a computer program which can help us to explain and validate our findings in a mathematical view. The study Huang et al. (2010) proposes the Sparse Polynomial Regression methodology for making predictions based on feature data collected by executing sample programs. In this paper, two SPORE algorithms are able to build relationships between responses (e.g., the execution time of a computer program) and features, and select features for constructing a non-linear and explicitly sparse model for generating predictions.

Methodology

Multiple factors affect the overall execution time of a computer program that is written in a specific programming language. The overall execution time of a computer program is mainly a combination of Static Codes Complexity, External HTTP calls (I/O), and the Hardware Performance of the execution environment (Figure 1).

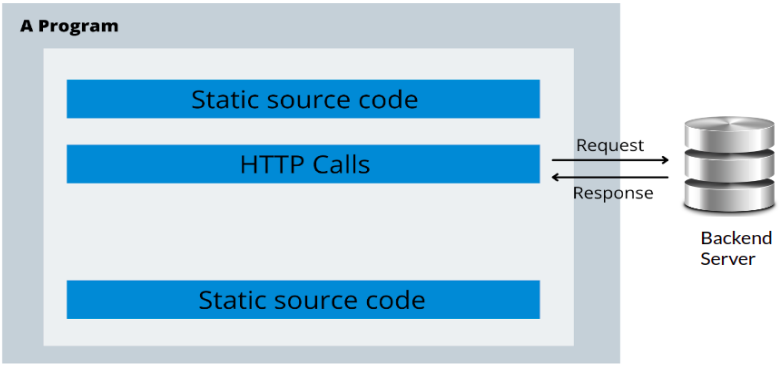

Figure 1: Structure of a computer program

Three main types of parameters affect the execution time **:**

- Static code complexity (If conditions / Loops / Classes … etc) => X
- Http calls (I/O) => Y
- Hardware performance and usage => Q

Execution time = ΣX + ΣY + ΣQ

There are a certain number of HTTP calls in a computer program, and there will be a considerable effect on the execution time of a computer program. In order to address this, we considered the behavior of HTTP calls of the data set as a feature. The hardware performance of the execution environment is dependent on a computer-to computer. Therefore, the same computer program can be executed several execution times based on the hardware performance of the device (Shah et al., 2019). Because of that, predicting the exact execution time of a computer program regressively is impractical and useless. In this research, we propose a solution to predict the Execution Time Complexity Level of a computer program based on the Static code complexity and HTTP calls (I/O) of a computer program while keeping the hardware performance constant.

Sample data set

In order to come up with a data set, the Codeforces platform was used, where developers regularly host programming contests. A wide variety of source codes for programming problems in many languages are submitted there. In this research, we collected 933 Java source codes. This code base sampling was done on the basis of data tags associated with the problem statement (structure/algorithm/binary search/sorting etc) to make sure that the code samples contain data belonging to a variety of complexity classes, $O(1)$, $O(logn)$, $O(n2)$, $O(n)$ and $O(nlogn)$.

Also, we included multiple source codes for the same problem. Different programs have various complexities. There are two approaches used when training the classification model such as extracting features from the code using hand-engineered-static analysis and automating with code embedding. There can be some unreachable codes that are never called from the main function. This was the main challenge when it came to preparing the data set. Removing such unreachable code manually is difficult. There are JDT plugins to identify the reachable methods in order to extract the listed features. The same technique was used here. We have considered the features as stated in Table 1 in this research.

Table 1: Features of the model

Number of methods	Number of breaks
Number of switches	Number of loops
Conditional-Loop frequency	Loop-conditional frequency
Loop-Loop frequency	Conditional-conditional frequency
Nested loop depth	Number of ifs
Number of variables	Number of jumps
Number of statements	Number of HTTP Calls

Big O notation of a computer program is the most general metric to calculate time complexity. The static code Time Complexity of a computer program proportionately varies theoretically as

80% and 20% training set and the test set data, respectively. A numerical representation was given in order to train the model. Complexity class-wise data set distribution is given in Table 2.

Table 2: Complexity class wise data set distribution

Complexity Class	Count
O(n)d	385
O(n2)	200
O(*nlogn*)	150
O(1)	143
O(*logn*)	55

Dataset contains code samples with 1-8 HTTP Calls / API GET Requests from the same server. Final Execution Time Prediction Label is based on five overall execution time complexity levels based on all the static code features including the behavior of HTTP calls.

Since total execution time varies between 0.280937598 and 7.135457685 Seconds, in our code samples, the complexity levels were defined as given in Table 3.

Table 3: Complexity levels

Total Time Class	Representation	Distribution
Above 6 Seconds	Execution Time is Higher	15
Between 4 - 6 Seconds	Execution Time is High	161
Between 2 - 4 Seconds	Execution Time is Average	457
Between 1 - 2 Seconds	Execution Time is Low	221
Between 0 - 1 Seconds	Execution Time is Lower	79

The distribution of the dataset based on the number http calls are tabulated in Table 4.

Table 4: Distribution of dataset based on http requests

Number of http calls	Samples Count
1	124
2	124
3	108
4	137
5	109
6	108
7	113
8	113

When the ML model was trained, each piece of the program was executed in a fixed hardware performance condition in order to make sure the constant of the hardware resources such as CPU, RAM, OS...etc as given in Table 5.

Table 5: System specifications

Specification	Value
CPU	11th Gen Intel core i5 -1135G7
GHz	2.40GHz
Memory	16 Gig Ram
OS	Ubuntu 20.04.4. LTS

Results

Table 6 depicts the accuracy score, precision, recall, and F measure values for these classifications using 6 different classification algorithms, with the best accuracy score achieved by using the approach of Random Forest.

Table 6: Results comparison for accuracy score, precision, recall, and F measure values

ML Model	% Accuracy	% Precision	% Recall	% F1 Score
Decision Tree Classifier	86.096	87.096	84.045	85.545
K Nearest Neighbor	58.824	58.884	52.625	55.579
Random Forest Classifier	88.235	88.958	88.268	88.611
Naive Bias Classifier	22.995	23.975	22.995	23.475
Support Vector Classifier	50.802	54.802	55.802	55.297
MLP Classifier	54.011	56.141	57.811	56.964

Since Random Forest gives the highest accuracy score out of all the models we trained, the Precision Recall Curve was generated (Figure 2) based on the Random Forest Classifier. A precision-recall curve shows the relationship between precision and recall for every possible cut-off.

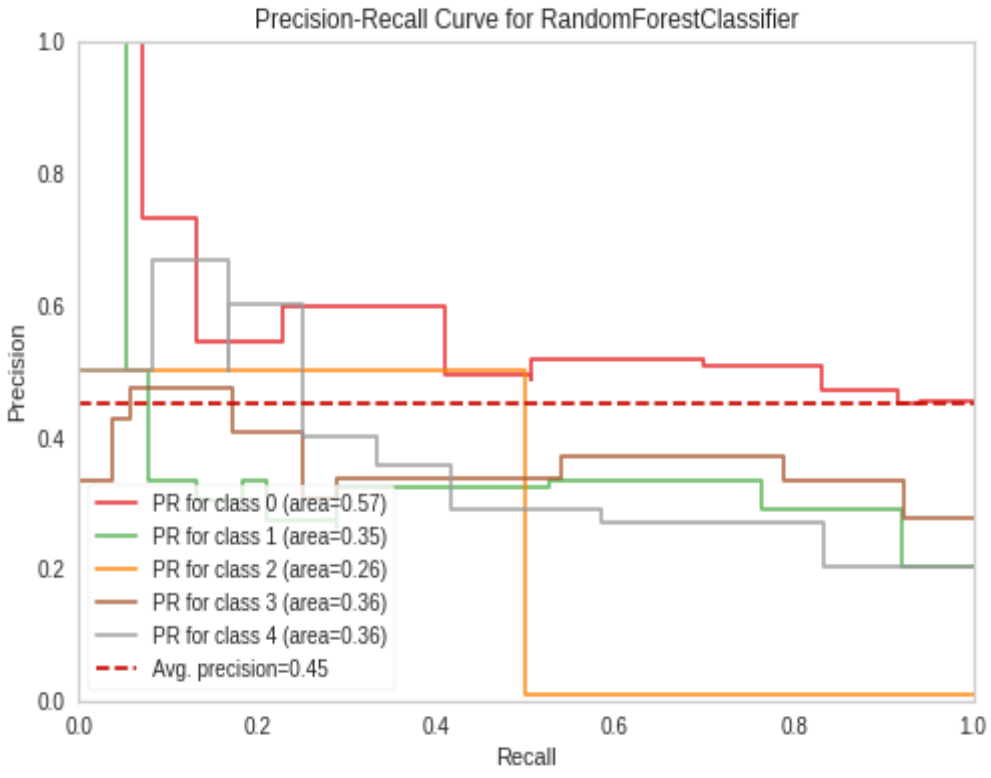

Figure 2: Precision-recall curve

The GINI index chart (Figure 3) was generated for the Random Forest Model and in case it needs to be magnified, it is accessible / downloadable via the link below. Link : graphviz-gini-index-chart

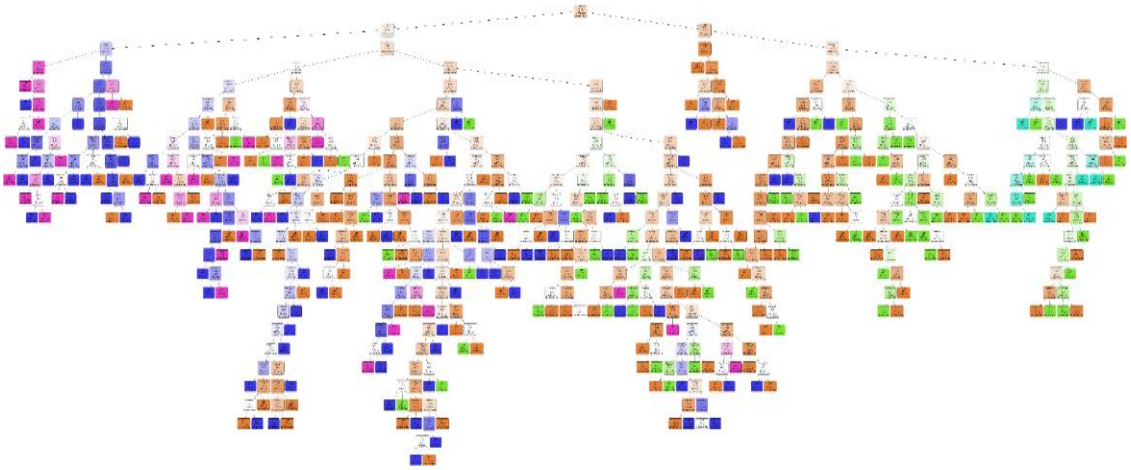

Figure 3: Graphviz Gini Index Chart

Discussion

More than 900 code samples (933) were analyzed that contain features against the execution time complexity classes. We chose traditional ML classification algorithms such as Decision Tree Classifier, K Nearest Neighbor Classifier, Random Forest Classifier, Naive Bayes Classifier, Support Vector Classifier, and MLP Classifiers to verify the impact of how the features affect computer programs on their runtime complexities.

Moreover, export_graphviz was used to generate the graph based on the Gini index to determine which features are mostly affecting the execution time of a computer program. The most prominent features among the considered features were the number of HTTP calls and the nested loop depth, followed by loops. An important finding of this study is that since the execution time depends on Static Code Complexity, API Calls, and, most importantly Hardware performance, Predicting the execution time in a regression manner is impractical. As a solution, we came up with this research study to predict The Level of Execution Time Complexity as a classification that determines some useful findings for the people who are working on developing some ML-based IDE plugins in order to predict the exact execution time of a given programming language. As the next major step, we will utilize the findings of this research in order to develop a powerful tool or an IDE plugin that can utilize the hardware performance of the running computer and predict the exact execution time of the computer program while the developer is coding his program live.

Conclusions

Execution time complexity prediction is an important aspect of well-structured algorithms and computer programs. It will definitely be a measure of the performance of a computer program for a given problem where developers struggle to produce the best performing code. We identified that the execution time depends on a number of variable factors such as the Static code complexities, in and out HTTP calls in the code as well as the operating system, RAM, processors, etc…and the hardware specifications. Since it is machine-dependent, giving the prediction as an exact execution time prediction cannot be used as a universal measure of analyzing the efficiency of algorithms. As a solution for that, in this research, we used some complexity classes (Labels) as the prediction, which showcases the picture of how efficient it will be in execution. Further, in this research, we identified that in the collected data-set, the most prominent feature which affects the complexity among the features that we considered are the number of HTTP calls, and nested loop depth, followed by loops. In the future, the size of the data set can be increased, and the accuracy of the findings will be further increased. The findings of this research can be optimized for implementing an IDE plugin or a developer tool that can predict the exact execution time of a given computer

program lively by considering the specifications of the execution device as well, and also it will help developers in terms of optimizing a particular computer program and develop it for a minimum execution latency and enhancing the performance of the program.

Acknowledgements

The authors acknowledge the support given by the research & development division at WSO2, Sri Lanka, by providing insights and expertise support (Non-Monetary).

Conflicts of interest

The authors declare no conflicts of interest.

Author contribution

IMMPDeR: Conceptualization, Data collection, Writing the manuscript; LM: Supervision, analyzing the data set, methodology, implementation, editing the manuscript.

References

Priya, R., de Souza, B. F., Rossi, A. L., & de Carvalho, A. C. (2011). Predicting execution time of machine learning tasks using metalearning. *Proceedings of World Congress on Information and Communication Technologies*, 1193-1198. IEEE. doi: 10.1109/WICT.2011.6141418

Park, C. Y. (1993). Predicting program execution times by analyzing static and dynamic program paths. *Real-Time Systems, 5(1)*, 31-62. doi: 10.1007/bf01088696

Nadeem, F., & Fahringer, T. (2009). Predicting the execution time of grid workflow applications through local learning. *In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* pp. 1-12. IEEE. doi:10.1145/1654059.1654093

Huang, L., Jia, J., Yu, B., Chun, B. G., Maniatis, P., & Naik, M. (2010). Predicting execution time of computer programs using sparse polynomial regression. *Advances in Neural Information Processing Systems*, 23. doi:10.1145/1654059.1654093

Shah, S., Amannejad, Y., Krishnamurthy, D., & Wang, M. (2019). Quick execution time predictions for spark applications. *Proceedings of the 15th International Conference on Network and Service Management (CNSM)* 1-9. IEEE. doi:10.23919/cnsm46954.2019.90127