## 4.1.3 Predictive Models

In this study, the Python library sklearn was utilized to construct various machine learning models, including linear regression and random tree regression. To evaluate the performance of each model, various metrics were applied including Mean Squared Error, R2 score, and cross-validation scores. For cross validation, the function cross\_val\_score was used to estimate model's accuracy in a more robust way than with just the typical train-test split. All the data were fed inside it, and the function made the necessary train-test splits, compared with using train\_test\_split.

Part of model validation for linear regression model was applied. The linear relationship was checked by scatter plotting Actual value Vs Predicted value and multivariate normality was best check with Q-Q plot. Also, residual Vs fitted value scatter plot was checked to make sure heteroscedastic plot exhibited a funnel shape pattern.

The random tree regression model was specifically set with 1000 decision trees running in the model and 42 random states, and several parameters were adjusted to determine the criterion used to determine model outcomes, the maximum possible depth of each tree, and the maximum number of features the model considered when determining a split.

The model's consistency was further evaluated using Mean Squared Error, R2 score, and cross-validation scores. During cross-validation, RepeatedKFolder function from sklearn was utilized to determine the number of splits to use. This cross-validator repeats the K-Fold n times with different randomization in each repetition. The robustness of results was ensured by splitting the data into 5 folds and repeating the process 10 times, each with different random splits. This configuration could evaluate the performance of machine learning models on a limited sample of data in a more robust manner.

## 4.2. Experiment Result

Apple

rfr

-259107704843.12335

lr

Linux

rfr

-287562454327615.9

lr

# 7. Reference

[[] Machine Learning Approach to Predict Computer Operating Systems Vulnerabilities](https://ieeexplore.ieee.org/document/9096731/authors" \l "authors)

[Freeh Alenezi; Chris P. Tsokos](https://ieeexplore.ieee.org/document/9096731/authors" \l "authors)

[Published in: 2020 3rd International Conference on Computer Applications & Information Security (ICCAIS)](https://ieeexplore.ieee.org/document/9096731/authors" \l "authors)

[Date of Conference: 19-21 March 2020](https://ieeexplore.ieee.org/document/9096731/authors" \l "authors)

[Date Added to IEEE Xplore: 20 May 2020](https://ieeexplore.ieee.org/document/9096731/authors" \l "authors)

https://dev.to/arepp23/how-to-write-to-a-csv-file-in-c-1l5b

https://towardsdatascience.com/how-to-build-your-first-machine-learning-model-in-python-e70fd1907cdd

https://towardsdatascience.com/how-to-use-random-seeds-effectively-54a4cd855a79

https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/

https://www.linuxjournal.com/article/8110

# Topic:

performance. I can also use classification algorithms such as decision trees or neural networks to classify the performance as good or bad based on the input data. So sounds straightforward.

3. Predictive Models

In addition, I have also encountered some challenges for this project.

Thus it is Not efficient enough to collect linux data in this way. Compared with the data collecting approach I applied on macbook, it still has some advantage that the data is recorded constantly on virtual file. I can still choose to export large gigabytes of data without missing consistent data info since the kernel module is working.

~~In our experimentation we modify the Linux Kernel scheduler to allow scheduling with customized time slices. The "Waikato Environment for Knowledge Analysis" (Weka), an open source machine-learning tool is used to find the most suitable ML method to characterize our programs. We experimentally fined that the C4.5 Decision Tree algorithm most effectively solved the problem. We fined that predictive scheduling could reduce TaT in the range of 1.4% to 5.8%. This was due to a reduction in the number of context switches needed to complete the process execution. We find our result interesting in the context that generally operating systems presently never make use of a program's previous execution history in their scheduling behavior.~~