# Learning based Methods for Code Runtime Complexity Prediction

Jagriti Sikka, Kushal Satya, Yaman Kumar, Shagun Uppal, Rajiv Ratn Shah, Roger Zimmermann

## Objective

Automatic prediction of runtime complexity of the source codes collected from Online Judges using traditional ML classification models and neural code embeddings.

## Motivation

Time Complexity computation is a crucial aspect in the study and design of well-structured and computationally efficient algorithms. Code runtime complexity prediction can have applications in:

- Automatic grading of coding assignments.
- Fast performance software development, by integrating complexity prediction models with IDEs.

## Dataset Curation

The data collection and preparation involved the following phases:

- **Data Collection**: Source codes of different runtime complexities were collected using Codeforces API. Multiple topics of Data Structures And Algorithm were considered while collecting the dataset.
- **Data Annotation**: A dataset of 932 codes [1] across 5 complexity classes, namely, $O(1)$, $O(logn)$, $O(n)$, $O(nlogn)$ and $O(n^2)$, was curated and manually annotated.

## Contributions

- Releasing a novel annotated dataset of program codes with their runtime complexities.
- Proposing baselines of ML models with hand-engineered features and study of how these features affect the computational efficiency of the codes.
- Proposing another baseline, the generation of code embeddings from Abstract Syntax Tree(AST) of source codes, to perform classification.

## Feature Engineering

For the first baseline, we trained traditional ML models on key features extracted from code.

- **Feature Identification:** We identified 14 key features that represent the fundamental coding constructs of any programming language, for e.g., number of loops, nested loop depth, use of sort calls etc.
- **Feature Extraction:** We extracted these features from the Abstract Syntax Tree(AST) of source codes. We used Eclipse JDT plugin [2] to extract the ASTs of codes and traversed the extracted ASTs to compute the values of the features.

We found a strong correlation between some features and the complexity classes. For example, depth of nested loops was mostly 1 for complexity class $O(n)$ whereas it was mostly 2 for complexity class $O(n^2)$.

## Code Embeddings

AST of a program captures comprehensive information regarding a program's structure, the syntactic and semantic relationships between variables and methods. Since an AST is infact a graph, we used graph2vec , a neural embedding framework [3], to compute code embeddings. Graph2vec learns task agnostic embeddings in an unsupervised manner by identifying non-linear graph constructs, and does not require a large corpus of data, making it apt for our problem.

## Baselines Comparison

- For first baseline, we trained 6 classification models on hand-engineered features. SVM and Random Forest classifiers achieved highest accuracy.
- For second baseline, we computed 1024-dimensional code embeddings from ASTs using graph2vec and trained an SVM classifier on these embeddings.
- The SVM accuracy of both baseline models was comparable; code embeddings achieved slightly higher precision and recall.

| Complexity class | Number of samples |
|---|---|
| $O(n)$ | 385 |
| $O(n^2)$ | 200 |
| $O(nlogn)$ | 150 |
| $O(1)$ | 143 |
| $O(logn)$ | 55 |

Table 1:Classwise data distribution

| Baseline | Accuracy | Precision | Recall |
|---|---|---|---|
| Feature Engineering | 72.96 | 69.43 | 70.58 |
| Code Embeddings | 73.86 | 74 | 73 |

Table 2:Accuracy, Precision, Recall values for SVM based classifier for the two baselines

| Ablation Technique | Accuracy | |
|---|---|---|
| | Feature Engineering | Code Embeddings |
| Label Shuffling | 48.29 | 36.78 |
| Method/Variable Name Alteration | NA | 84.21 |
| Replacing Input Variables with Constant Literals | NA | 16.66 |
| Removing Graph Substructures | 66.92 | 87.56 |

Table 3:Data Ablation Tests Accuracy of feature engineering and code embeddings baselines

## Data Ablation Experiments

We designed four different data ablation experiments to get insight into the predictions of the baselines. These experiments were designed to study the robustness of the baselines to accommodate changes in the dataset and its accuracy of prediction of unseen data points and outliers.
Code embedding approach performed significantly better in these experiments and thus is a better baseline.

## Conclusion

Automatic prediction of runtime complexity of codes is a novel problem with interesting potential applications. We believe a public code-complexity dataset will enable further research in this domain.

## References

[1] Our Dataset Link.
https://github.com/midas-research/corcod-dataset.

[2] Eclipse JDT.
https://projects.eclipse.org/projects/eclipse.jdt.

[3] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal.
graph2vec: Learning distributed representations of graphs.
*CoRR*, abs/1707.05005, 2017.

## Contact Information

- Jagriti Sikka, Adobe, Noida: jsikka@adobe.com
- Kushal Satya, Adobe, Noida: satya@adobe.com
- Yaman Kumar, Adobe, Noida: ykumar@adobe.com
- Shagun Uppal, MIDAS, IIIT Delhi: shagun16088@iiitd.ac.in
- Rajiv Ratn Shah, MIDAS, IIIT Delhi: rajivratn@iiitd.ac.in
- Roger Zimmermann, NUS, Singapore: rogerz@comp.nus.edu.sg

MIDAS@IIITD Multimodal Digital Media Analysis Lab

ASU Arizona State University

Bloomberg