Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 159 (2019) 582–589

Procedia Computer Science

www.elsevier.com/locate/procedia

23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Visualization of Programming Skill Structure by Log-Data Analysis with Decision Tree

Shinichi Oeda[a*], Mutsumi Chieda[b]

[a]*Department of Information and Computer Engineering, National Institute of Technology, Kisarazu College, 11-1, Kiyomidaihigashi 2-chome Kisarazu City, Chiba, Japan*

[b]*Advanced Course of Control and Information Engineering, National Institute of Technology, Kisarazu College, 11-1, Kiyomidaihigashi 2-chome Kisarazu City, Chiba, Japan*

Abstract

To evaluate the ability of a programmer in universities or staff agencies, evaluators conduct examination in which applicants solve problems, or they refer to the applicants' GitHub and review their code. However, the collected source code is evaluated by a human under present conditions. This leads to two problems: many source codes cannot be evaluated simultaneously, and it is difficult to maintain consistency in the evaluation criteria among evaluators. We propose methods to estimate the current skill of the programmer, which can be used by the evaluators to understand an applicant's skill by analyzing the collected source code automatically. In particular, this study visualizes the feature of the students' source code in term of quarters using a decision tree to estimate the programming skill.

Keywords: Programming lesson; Beginner Programmer; Decision Tree; Random Forest; Latent Skill; EDM(Educational Data Mining)

1. Introduction

Intelligent tutoring systems (ITSs) and learning management systems (LMSs) have been widely used in the field of education. They allow the collection of log-data from learners, and especially, students. Educational data mining (EDM) aims to obtain useful information from massive amount of electronic data collected by these educational systems. EDM is an emerging multi-disciplinary research area; several methods and techniques are being developed to explore data from various educational information systems [1].

* Corresponding author. Tel.: +81-438-30-4144 ; fax: +81-438-98-5717.
E-mail address: oeda@j.kisarazu.ac.jp

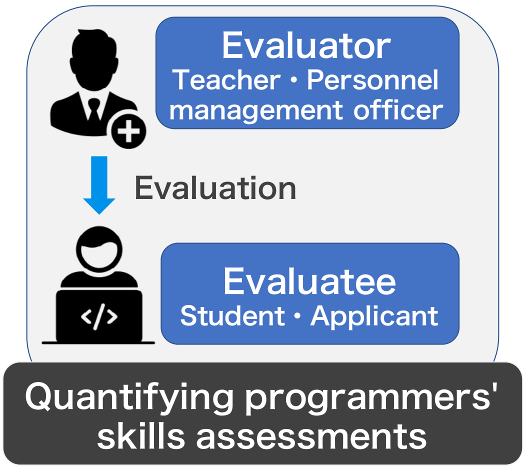
Evaluator
Teacher・Personnel management officer
Evaluation
Evaluatee
Student・Applicant
Quantifying programmers' skills assessments

Fig. 1: Background of our research

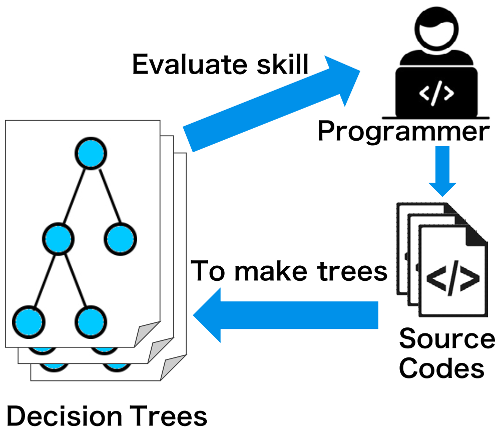

Fig. 2: Overview of the proposed method

We focus on actual school educational systems. In most schools, one teacher teaches many students in a class at once. Thus, it is difficult for teachers to provide detailed guidance to each student. A teacher needs to know the students' level of skill to provide high-quality education. Hence, periodic examinations are used to confirm whether students have acquired the necessary skills. However, it is difficult to know the status of the student from each lesson because examinations are time-consuming and burdensome to grade. In contrast, it is easy to automatically record behavior in programming classes, because the history of each student's UNIX commands and source-code editing can be stored as log-data.

Places where programmers' abilities are evaluated, such as schools and employment examinations, examinations may be conducted, and the written source code may be used as a reference. At present, those evaluations are currently conducted manually; hence, it is difficult to maintain the consistency in the evaluation criteria. Fig. 1 shows the background to evaluate programmers' skill.

To solve this problem, we aim to analyze source code automatically and extract programmers' skill. In this study, we visualized the feature of the source code of each student in examinations conducted every several months and the behavior of the programmer based on the command log using the decision tree model. The structure of the programming skills was also visualized. Fig. 2 indicates the overview of the proposed method.

2. Problem Setting and Related works

Monitoring and supporting students is considered very important in many educational institutions. If a teacher can identify a weak student at an early stage, the can take measures to ensure that the student will not drop out of the class. Therefore, it is necessary to identify these students during each lesson, so that the teacher can pay special attention to them. In programming classes, the behavior of students can be confirmed using log-data. We developed a logging system that can acquire the input history of UNIX commands and the editing history of the source code. We obtained several datasets from students in our school. Our dataset from programming lessons consists of data from 39 students. When predicting potential dropouts, it is common to create evaluators with supervised learning[2, 3, 4, 5]. However, such data are difficult to learn if the size of the dataset is small. Moreover, the features in the log-data that depend on the content of each lesson; for example, whether there are multiple exercises or sufficient explanations. Therefore, we apply outlier detection with unsupervised learning. We assume that students who belong to an outlier cluster can be compared with either the excellent students or inferior ones. We performed k-means clustering according to the Euclidean distance, to compare the timing of the active behavior and clustering using dynamic time warping. Thus, we can compare the flow of activity to the exclusion of time-series deviations.

Massive open online courses (MOOCs) offer a new and increasingly popular model for delivering educational content online to anyone who wants to take a course, with no limit on attendance. MOOCs, first introduced in 2006, are a recent and widely studied development in distance education. By 2012, MOOCs had already become a widely popular mode of learning [6, 7]. However, because the number of students in our target classes is fewer than 50, a new model must be developed. In Japan, several studies have been conducted to improve classes using log-data analysis. Most of these studies developed original tools to obtain log-data based on the student behavior, and analyzed this data by writing programs using the tools[8, 9]. In contrast, we aim to analyze log-data that can be acquired without affecting the coding environment of the students and without educational constraints.

First, we investigated if there were any similar studies or methods to analyze and evaluate the source code. Some existing papers and services have obtained results closely related to our study. In the paper referred to from this point of view, in the version control system "git," data to be used for learning are acquired using "blame," which is a function for acquiring information on specified lines of source code. There was something [10]. For the collected data, abstract syntax trees were created using "joern," which can even analyze fragmented source code. Moreover, the data to be learned treated a few lines of code as one sample, and extracted approximately 60,000 features for a dataset of 15 people. However, the "joern" parser has not been updated since it was released in 2014, and the processing environment has not been developed because of dependency problems. In addition, in the study [11], to evaluate programmers, a general survey was carried out and a feature was suggested to add the scores of other subjects. The questionnaire collected detailed information such as family structure and class method from general information such as name, age and gender. Although some of them were referred to, it is necessary to do a questionnaire at present stage was not possible.

One of the existing employment and career change services "paiza[12]" and "moffers[13]" are the result of a job and career changer solving a programming task on the Web. The result and process are verified, and the skills of the career and career change candidate are It is a service guaranteed for the company that uses it. In this service, to evaluate the ability of the programmer, " the contents of the code itself are not evaluated, instead the correct answer rate of multiple test cases, code description time, execution speed, and memory consumption are evaluated." The source code itself is evaluated at the time of company interview, and the judgment criteria depends on the company's viewpoint. Therefore, the automatic evaluation of the source code itself is not considered to be sufficiently reliable.

3. Proposed Methods

3.1. Decision Tree

In this study, it is assumed that it is used in the field of education, and we visualize the features so that they are easily understood by those not familiar with machine learning. Decision trees [14] are prediction models based on tree structures, and they are suitable for such problems because the prediction results are output in a form that is easy for people to understand. Therefore, in this study, we used decision trees to visualize the results for regression applications. The procedure for the generation of a decision tree is as follows.

1. Input all data into the root node.
2. Divide data into nodes so that the Gini coefficient is minimized.
3. Repeat steps 1 and 2 at each split node until the mean squared error in the node falls below a certain value or the tree reaches a certain depth.

The Gini coefficient is defined as follows. For a node t with a decision tree, consider the case where there are n samples in the node and c classes in the node. In node t, assuming that the number of samples belonging to class i is n_i, the ratio $p(i|t)$ of samples belonging to class i is given by

$$p(i|t) = \frac{n_i}{n}. \qquad (1)$$

Now, the Gini coefficient I_G is defined as

$$I_G(t) = 1 - \sum_{i=1}^{N} p(i|t)^2. \tag{2}$$

3.2. Random Forest

Random forest[15] is a prime example of the ensemble machine learning method. An ensemble method aggregates less predictive base models to produce a better predictive model. Random forests, as one can intuitively guess, ensembles various decision trees to produce a more generalized model by reducing the over-fitting tendency of decision trees.

In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e., they have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees that are trained on different parts of the same training set, to reduce the variance. However, this results in a a small increase in the bias and some loss of interpretability, but the overall performance of the final model is improved considerably.

3.3. Feature vector

In this study, the command log and source code collected every minute are converted into multiple features for the periodic examination of the subject of "Programming language" in the second year of Kisarazu College of Technology. The extracted features are organized into a CSV file and used to generate a decision tree model. Some of the features used are shown in the following Table1.

Table 1: Feqture vector

Features	Abbreviation
Frequency of appearance of self-made function declaration	ln_functions
String length of self-made function name	mean_function_name
Number of function arguments	n_function_args
String length of variable name	mean_variable_name
Indentation method (BSD type / kernel type)	indentation_style
Frequency of appearance of blank line	ln_blank_lines
Number of characters per line	mean_letters_of_line
Distribution of the number of characters per line	std_letters_of_line
Indent character (tab / space)	tab_space_style
Frequency of appearance of "do"	do
Frequency of appearance of "while"	while
Frequency of appearance of "for"	for
Frequency of appearance of "if"	if
Frequency of appearance of "else if"	elseif
Frequency of appearance of "else"	else
Frequency of appearance of "switch"	switch
Number of lines in self-made function	mean_function_lines
Frequency of appearance of single-line comment	oneline_comment
Frequency of appearance of multi-line comments	multiline_comment

4. Experimental Results

The preprocessed data are used to train a decision tree model to visualize the skill structures that affect student scores. Moreover, regression prediction is performed from the generated decision tree model, and the model is validated by finding the decision index R^2 between the prediction and the actual score. The decision index is an index of accuracy; the closer the index is to 1, the more accurate the model.

$$R^2 = 1 - \frac{\sum_{i=0}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n}(y_i - \bar{y})^2} \tag{3}$$

We applied the random forest to the source code collected from each student. The number of decision trees is five in the random forest. The decision trees output from the random forest is shown in Fig. 3,4,5,6,7. We used several the evaluation index. "Precision" and "Recall" are two ways to assess the model accuracy. Because this is a multiclass classification, it is calculated by combining micro averages. However, for the precision, the model is considered good if all predicted values are 0.0. For recall, it is considered good if the values are all 1.0. Therefore, the F-measure obtained by taking both harmonics is used as the evaluation index. $TP(i)$ means "True Positive." Similarly, $FP(i)$ means "False Positive" and $FN(i)$ means "False Negative," The precision that estimated rating using the generated model was 0.714 in F-measure.

$$Precision = \frac{\sum TP(i)}{\sum TP(i) + \sum FP(i)} \tag{4}$$

$$Recall = \frac{\sum TP(i)}{\sum TP(i) + \sum FN(i)} \tag{5}$$

$$F - measure = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision} \tag{6}$$

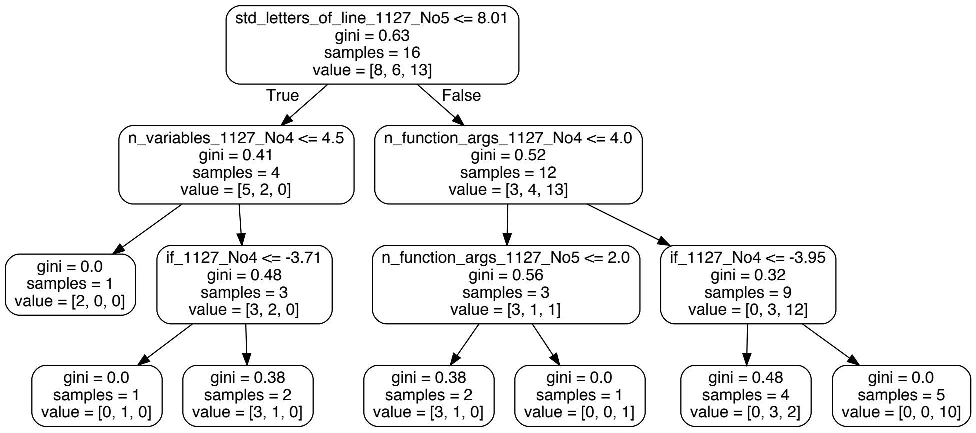

Fig. 3: Generated decision tree (No.1)

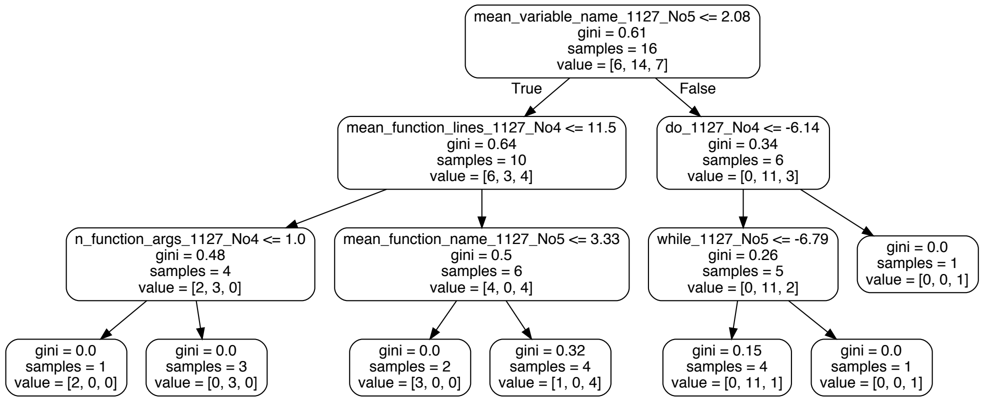

Fig. 4: Generated decision tree (No.2)

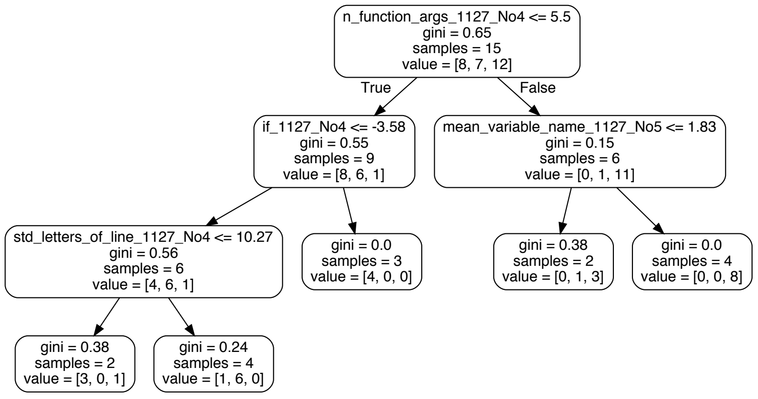

Fig. 5: Generated decision tree (No.3)

5. Discussion

From the experimental results, it can be seen that the decision index takes negative values, and the generated regression tree could not predict the test data. This might be because of the following factors.

- There are only 42 samples available for machine learning at this stage. At least 50 samples are required to perform machine learning.
- The extraction of features, especially analysis of log data, was insufficient.
- Decision tree regression is considered to be inferior to methods such as stochastic gradient descent and logistic regression in pure regression prediction.
- The regression tree was not adjusted.

In addition, the importance of prediction for each feature was the highest at the end of the test, and the next was the number of lines per code. This is because the decision tree used the amount of program as a judgment criterion because the number of features given to the decision tree was small, and that the students who could not solve the task were not able to touch the coding.

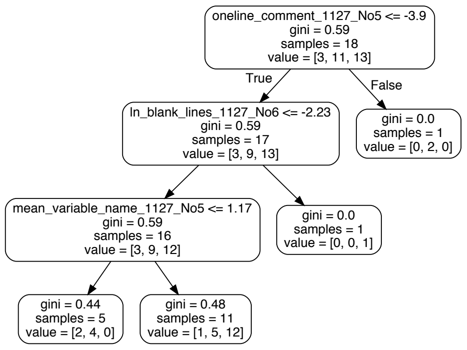

Fig. 6: Generated decision tree (No.4)

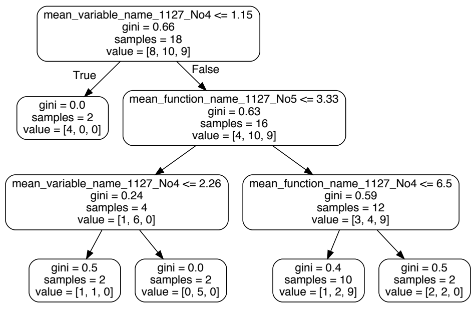

Fig. 7: Generated decision tree (No.5)

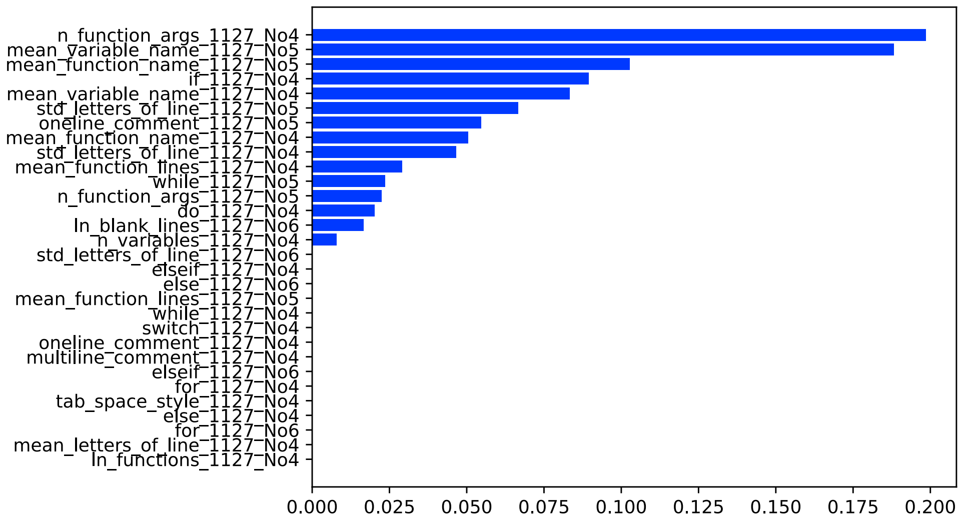

Fig. 8: Significant features (sorted)

6. Conclusion and Future works

In this study, we proposed using decision tree regression to visualize the prediction of the skill of students and factors that are the basis of the prediction. However, the prediction accuracy in this experiment was not practical. Therefore, in the future, it is particularly important to improve the feature extraction stage in machine learning. Two issues must be addressed in the future. First, because the accuracy of the decision tree is not yet practical, we will attempt the following to improve the accuracy.

- Securing additional features by log data analysis.
- Use another regression method.
- Search for hyperparameters.

Second, although the results of the regular tests were used for the teacher data in this study, the scoring criteria depends on the grader; therefore, teacher data excluding the personality of the grader is generated. A previous study [16] attempted to recommend the target part of the refactoring using machine learning for the source code written in

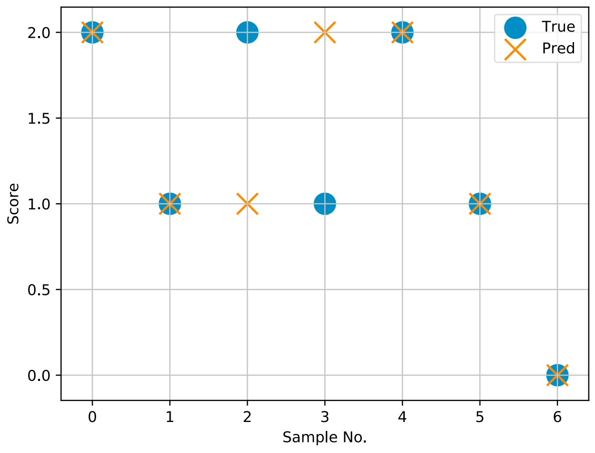

Fig. 9: F-measure

Java. Therefore, a method can be developed in which the sample to be refactored is given a high score for a small sample relative to the size of the code.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 19H01728.

References

[1] Toon Calders, Mykola Pechenizkiy. "Introduction to The Special Section on Educational Data Mining", SIGKDD, Vol. 13, No. 2, pp. 3–5, 2011.
[2] Gerben Dekker, Mykola Pechenizkiy, Jan Vleeshouwers. "Predicting students drop out: a case study", Proceedings of the 2nd International Conference on Educational Data Mining (EDM2009), pp. 41–50, 2009.
[3] Wentao Li, Min Gao, Hua Li, Qingyu Xiong, Junhao Wen, Zhongfu Wu. "Dropout prediction in MOOCs using behavior features and multi-view semi-supervised learning", Neural Networks (IJCNN) 2016 International Joint Conference on, pp. 3130–3137, 2016.
[4] Diyi Yang, Tanmay Sinha, David Adamson, Carolyn Penstein Rose. "Turn on, tune in, drop out: anticipating student dropouts in massive open online courses", Proceedings of the 2013 NIPS Data-driven Education Workshop. Vol. 11, 2013.
[5] Sherif Halawa, Daniel Greene, John Mitchell. "Dropout prediction in MOOCs using learner activity features", Experiences and Best Practices In and Around MOOCs, Vol. 7, 2014.
[6] Laura Pappano. "The Year of the MOOC", The New York Times. Retrieved 18 April 2014.
[7] Ezekiel J. Emanuel. "Online education: MOOCs taken by educated few", Nature 503, 342, 2013.
[8] H. Igaki, S. Saito, A. Inoue, R. Nakamura, S. Kusumoto. "Programming Proces Visualization for Supporting Students in Programming Exercise", Journal of Information Processing Society of Japan, Vol. 54, No. 1, pp. 330–339, 2013, (in Japanese).
[9] T. Kuchiki, K. Yamada, J. Sasaki. "A Study on Evaluation Methods of programming Skill Level", Proceedings of the 72nd National Convention of IPSJ. pp. 521–522, 2010, (in Japanese).
[10] E. Dauber, A. Caliskan, R. Harang, R. Greenstadt, "Git Blame Who?: Stylistic Authorship Attribution of Small, Incomplete Source Code Fragments", Cornell University Library, arXiv:1701.05681, Jan 2017.
[11] A. Pradeep, J. Thomas, "Predicting College Students Dropout using EDM Techniques", International Journal of Computer Applications (0975-8887), Vol. 123, No. 5, pp. 26–34, August 2015.
[12] paiza, `https://paiza.jp/`
[13] moffers, `https://moffers.jp/`
[14] Quinlan, J. Ross, "Induction of decision trees", Machine learning, Springer, Vol. 1, No. 1, pp. 81–106, 1986.
[15] Breiman, Leo (2001), "Random Forests", Machine Learning, Springer, Vol. 45, No. 1, pp. 5–32, 2001.
[16] Akira Goto, Norihiro Yoshida, Kenji Fujiwara, Eunjong Choi, Katsuro Inoue, "Recommending Extract Method Opportunities Using Machine Learning", Information Processing Society of Japan, Vol. 56, No. 2, pp. 627–636, 2015, (in Japanese).