Machine Learning Model for Analyzing Learning Situations in Programming Learning

Shota Kawaguchi
Graduate School of Education
Tokyo Gakugei University
4-1-1 Nukuikita, Koganei, Tokyo,
184-8501 Japan
m183305w@st.u-gakugei.ac.jp

Yoshiki Sato †
Facul. Education
Tokyo Gakugei University
4-1-1 Nukuikita, Koganei, Tokyo,
184-8501 Japan
satoyo@u-gakugei.ac.jp

Hiroki Nakayama
Global Education Center
Waseda University
1-104 Totsuka, Shinjuku, Tokyo,
169-8050 Japan
nakayama@aoni.waseda.jp

Ryo Onuma
Dept. Computer Science & Mathematics
Fukushima University
1 Kanayagawa, Fukushima,
960-1248 Japan
onuma@sss.fukushima-u.ac.jp

Shoichi Nakamura
Dept. Computer Science & Mathematics
Fukushima University
1 Kanayagawa, Fukushima,
960-1248 Japan
nakamura@sss.fukushima-u.ac.jp

Youzou Miyadera
Div. Natural Science
Tokyo Gakugei University
4-1-1 Nukuikita, Koganei, Tokyo,
184-8501 Japan
miyadera@u-gakugei.ac.jp

***Abstract* — In programming learning, students have individual difficulties, and teachers need to grasp those difficulties and provide appropriate support for the students. However, since it is a heavy burden for teachers, a method to automatically estimate the learning situations of students is required. In this research, we developed a method that adopts the development of a machine learning model as an approach to achieve this purpose. This machine learning model outputs the estimated learning situation when the source code editing history of new students is input. As a result of evaluating the developed method, it was possible to estimate the correct learning situations with high accuracy of 98%. The applicability of this learning situation estimation method in practical lessons was shown.**

Keywords — Programming Learning, Learning Situations Estimation, Machine Learning, Source Code Editing History, Education Support

I. INTRODUCTION

In recent years, students face various challenges in classes that teach programming in educational institutions such as universities. However, some students struggle to solve the problems in their exercises and often give up. For such students, teachers need to grasp detailed learning situations, such as the cause of their pitfalls and their weak points, and to provide appropriate support. However, it is difficult to grasp the learning situations of individual students. Therefore, it is desirable to develop a system to automatically estimate the learning situations of students.

Some studies have attempted to develop an automatic estimation method of learning situations. Reference [8] attempted to estimate detailed learning situations from the source code editing history to reflect a student's trial and error process. Since the analysis in this research uses the parse tree of the source code, it is impossible to estimate the learning situation when compile errors occur. However, it is considered that student encounters pitfalls when a compile error occurs. Therefore, in order for teachers to provide more meaningful support, learning situations need to be estimated even if a compile error occurs.

Therefore, in this research, we aim to develop a method to estimate learning situations regardless of success or failure of compilation. As an approach, the source code editing history of the student, including the times when compile errors occur, is analyzed. This method is based on the development of a learning situation estimation engine that outputs the learning situation of a student at a specific time from the source code editing history that reflect the student's trial and error process. In addition, the learning situation estimation engine is applied a machine learning model from existing methods. Those methods are multiclass classification methods which are one of supervised learning

As the evaluation results are based on the existing evaluation method, it was suggested that the learning situations could be estimated with high accuracy of 98%.

In this research, we target programming exercise classes in the form of making programs that satisfy the task specifications imposed by teachers in C language. Furthermore, it is assumed that all the tasks given to the student. will be answered with only one source code file.

II. RESEARCH SUMMARY

A. Overview of learning situations estimation method

Many studies aimed at developing a learning situation estimation method in programming learning have been conducted. However, as can be seen from reference [1] of data mining and learning analytics in programming learning, many researchers [2-6] use students' data, including error duration, number of errors, error messages output by the compiler, and the progress of tasks. Thus, estimation of the detailed learning situations, i.e. a point that is difficult and a point that is not understood, is not sufficiently made.

Therefore, several studies have attempted to estimate detailed learning situations on the basis of the source code editing history that reflects the student's trial and error process. Reference [7] developed the pitfall estimation method for each student. In this research, it is possible to estimate pitfalls by comparing student's source code to that of snapshots. Furthermore, reference [8] conducted an analysis using parse trees, and it is possible to estimate the learning situations on the basis of arbitrary granularity of the teacher.

However, in Fujiwara et al.'s research, it is impossible to estimate the factor of a student's pitfalls. In reference [8], because they analyze using a parse tree, learning situations cannot be estimated when compile errors occur. In a situation

† Info screw inc., 13-6 Sakae, Itabashi, Tokyo, 173-0015, Japan

where a compile error occurs, it is assumed that some kind of pitfall has occurred. Even if a compile error occurs, it is required to estimate the learning situations.

Therefore, from the source code editing history, development of a method to estimate the detailed learning situations regardless of success or failure of compilation is required, but no such research is currently being done.

The learning situations estimation method outlined in this research outputs the learning situation of a student at a specific time from their the source code editing history. Moreover, this method is achieved by the development of a learning situation estimation engine that utilizes the relationship between the learning situation and editing history. Because this learning situation estimation engine has already collected a large amount of learning history data at our university, we adopt a machine learning model obtained by applying from an existing machine learning method. A multiclass classification method is used, which outputs a predetermined concept known as a class that corresponds to an input. This method is a type of supervised learning that requires prior data known as training data. A method of outputting corresponding classes from an input of compilation is required, but no such research is currently being done.

B. Research Procedure

The procedure of this research is shown below.

- Definition of learning situations (Section III)
- Definition of input / output to the learning situations estimation engine (Section IV)
- Development of the support tool to create training data (Section V)
- Development and Evaluation of the learning situations estimation engine (Section VI)

In Section III, "learning situations in C language programming" is defined to clarify the learning situation handled in this research.

In Section IV, input / output to the learning situation estimation engine is defined to conform the existing machine learning method.

In Section V, the support tool to create training data necessary to create a machine learning model is developed.

In Section VI, training data is created on the basis of the support tool developed in Section V. Using various machine learning algorithms, we then develop learning situation estimation engines and evaluate each engine at the same time. In addition, we consider the possibility of learning situation estimation when this method is applied to the source code editing history of new students.

III. Definition of Learning Situations

To concretely define learning situations to estimate, "learning situations in C language programming" is defined. These learning situations should be as detailed as possible and not overlap in consideration of actual estimation.

Therefore, in this research, we set the learning situations in C language programming as "items to be learned divided as much as possible in C language programming" and "unable to understanding an item." In this section, we will consider specific division methods and describe the defined learning situations.

A. Items to be Learned Divided

The method of dividing items to be learned in C language programming has been developed and studied for a long time [9]. However, since the items to be learned by each educational institution vary, the items to be learned with finer granularity are not actually divided.

Therefore, we divided in accordance with the following ontology design algorithm proposed by reference [9].

1. Glossary development
2. Laddering
3. Disintegration
4. Categorization
5. Refinement

In Glossary development, we select and verbalize all essential objects and concepts. Here, concepts are "items to be learned" in this research, and these were extracted from reference [10]. In Laddering and Disintegration, we divide these concepts into smaller concepts by using top-down strategy. Finally, in Categorization and Refinement, similar concepts are grouped by using bottom-up strategy, eliminating synonyms and contradictions.

As a result of these algorithms, we could divide the items to be learned and maintain independence in every detail.

B. Definition of The Learning Situations

The learning situations in this research are defined as situations in which a student does not understand the items to be learned in detail. As a result, we defined 63 learning situations in this research.

Table I shows some of the learning situations. These learning situations are independent of each other.

TABLE I. PART OF THE LEARNING SITUATION

ID	*Learning Situations*
1	• Not understanding the data that can be handled with char type.
2	• Not understanding the data that can be handled with int type.
3	• Not understanding the data that can be handled with float type.
9	• Not understanding how to declare variables.
10	• Not understanding how to use arithmetic operators.
11	• Not understanding how to use comparison operators.
12	• Not understanding how to use logical operators.
16	• Not understanding how to use assignment operators.
18	• Not understanding the priority of evaluation.
19	• Not understanding how to use blocks.
23	• Not understanding the grammar of the for statement.
38	• Not understanding how to use a one-dimensional array.
55	• Not understanding how to use the printf statement.
56	• Not understanding how to use the scanf statement.
63	• Logical error.

IV. Definition of I/O to Learning Situations Estimation Engine

In developing the learning situations estimation engine, we adopted a multiclass classification method as described

above. Multi-class classification is a method that inputs a multidimensional vector and outputs a classifying element called a class as a result. In this section, the multidimensional vector to be input and the result of output are defined specifically.

A. Definition of Output

The output is determined as the probability corresponding to each of the 63 learning situations defined in Section III. As a condition, the result of the sum of these 63 probabilities must be 100%. In other words, the multidimensional vector to be input must be related to just one learning situation.

B. Definition of Input

As described earlier, source code editing history at each compile time is used as input regardless of success or failure of compilation to estimate a student's detailed learning situation. The history is a set of two or more entries of a student's source code data in chronological order.

In this section, we describe the method to express the source code editing history as a multidimensional vector. At this time, we have to satisfy the following three conditions.

(i) One multidimensional vector is related to one learning situation.

(ii) To be able to express the source code editing history with the finest granularity as possible.

(iii) To be able to handle the source code editing history after each compilation regardless of success or failure.

In previous research [8], the source code editing history was the main focus after each compilation. In order to model history entries, we also view the source code editing history as a set of differences in each compilation section and collect and analyze the transformation data of the parse tree in each section. This transformation refers to the operations of adding / deleting sentences and replacing tokens in the parse tree. In addition, the robust tree edit distance (RTED) algorithm [11] is used to collect the number of operations.

We define the source code editing difference of each compilation section as one multidimensional vector. Since some results have been obtained in previous research, it seems that this modeling will be related to one learning situation. Therefore, condition (i) can be satisfied.

Next, we consider a method of modeling source code editing differences as multidimensional vectors. In this research, we aim to estimate learning situations regardless of the success or failure of compilation, so we cannot adopt the modeling of previous research using parse trees.

To satisfy condition (ii), we first focused on the "token, " which is the minimum unit in the source code. We also developed a method to consider source code editing difference as a transformation of each token and express them as a multidimensional vector. This vector has a number of additions, deletions, and replacements of each token as elements. In addition, this method can handle even if the source code has compilation errors. Therefore, conditions (ii) and (iii) can be satisfied. The detailed algorithm creating source code editing difference vectors will be described in Section V-C.

Each element of the source code editing difference vector and the number of dimensions is considered again. In this research, "tokens" of the C language are divided into 84 types including reserved words, delimiters, identifiers, numbers, characters, and strings. To store the number of additions, deletions, and replacements of each token, it is necessary to secure a number of dimensions that can store the number of addition / deletion of each token (84 * 2 = 168) and the number of an arbitrary token replaced with another arbitrary token (about 84 * 83 = 6972). However, since the number of dimensions may be related to the accuracy of estimation, we want to make it as small as possible. Therefore, to reduce the dimensions of the replacement operations, the replacement operations were replacing tokens with those within the same defined group. We divided the 84 types into a total of 20 defined groups. Some of the groups classified in Table II are shown.

TABLE II. EXAMPLES OF TOKEN GROUPS

Token groups
• Type declaration • Brackets • Arithmetic operators • ID • Number

To summarize the above, the source code editing difference vector that is input to the learning situations estimation engine handled in this research is shown in Table III.

TABLE III. CONTENTS OF SOURCE CODE EDITING DIFFERENCE VECTOR

Source code editing difference vector	
Overview	Source code editing difference data of compilation section of student.
Storage element	• The number of additions of each tokens.(= 84) • The number of deletions of each tokens.(= 84) • The number of tokens of a group (= 20) replaced by those of the same group.
Dimensions	188 (= 84 + 84 + 20)

V. DEVELOPMENT OF TRAINING DATA CREATION SUPPORT TOOL

In this research, we need to collect training data to implement supervised learning. We developed a support tool to reduce the burden of creating data.

The main functions of this tool are as follows.

(1) Learning history collection function

(2) Source code editing difference vector creation function

(3) Support function for associating source code with learning situation

Function (1) is achieved by using a learning platform environment [12] currently being used in programming learning lessons. Functions (2) and (3) are implemented as a

Web application. Specific functions will be described in the following sections.

A. Learning History Collection Function

In this function, the learning history data shown in Table IV is collected. By using this function, it is possible to collect the source code editing history without burdening the students. The specific collection method is as follows.

1. The teachers distribute the development environment to each student. In this environment, wrapper commands of "gcc: compile command" and "a.out: execution command" exist.

2. Confirming whether the environment is connected to the network.

3. If it is connected to the network, each time the students execute wrapper commands, learning history data and the log file are sent to the Web server and stored in the database.

TABLE IV. COLLECTING LEARNING HISTORY DATA

Collecting Timing	***Collecting Data***
Each compilation	student ID, task ID, error message, time, source code, parse tree
Each execution	input data, output data
Each program submission	free description quesrtionnaire about difficult places and reasons

B. Source Code Editing Difference Vector Creation Function

In this function, it is possible to view a bar chart showing the total number of compilation data entries for each task and all task ids and task contents. Fig. 1 shows an example of a bar chart of task data displayed by this function.

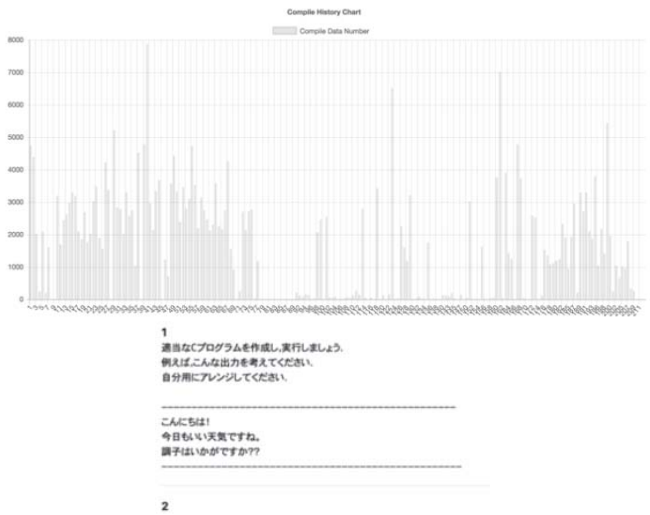

Fig. 1. Bar Chart of Compilation Data and Tasks

From this information, the task id is specified, and the source code editing difference vectors are created from a student's source code editing history of the task. Specifically, according to the following algorithm, source code editing difference vectors are created from each source code editing difference in the source code editing history.

I. Source code editing difference vectors that store the number of additions, deletions, and replacements of each token defined in Section IV-C is prepared.

II. Each of the two consecutive source code entries in a compilation time series is regarded as a sequence of tokens, and those tokens are determined by line breaks.

III. The token differences in each compilation section are collected by using the UNIX diff command.

IV. The obtained token differences are analyzed, and the number of addition, deletion, and replacement of tokens is collected and stored in each element of the prepared source code editing difference vector.

C. Support Function for Associating Source Code with Learning Situation

This function supports teachers when they associate the learning situations with each source code editing difference vector created in Section V-B. It is finely classified into the following three sub-functions.

a) Learning history viewing sub-function

b) Source code editing differences viewing sub-function

c) Learning situation association sub-function

a) *Learning History Viewing Sub-Function*

In this sub-function, it is possible to view the learning history for a certain task of any student.

Fig. 2 shows an example of a screen displayed by this function. The learning history is displayed in chronological order from the left. The source code, error messages, execution history (input / output result), and the compilation date and time are displayed in order from the top. These data are collected at the time of compilation.

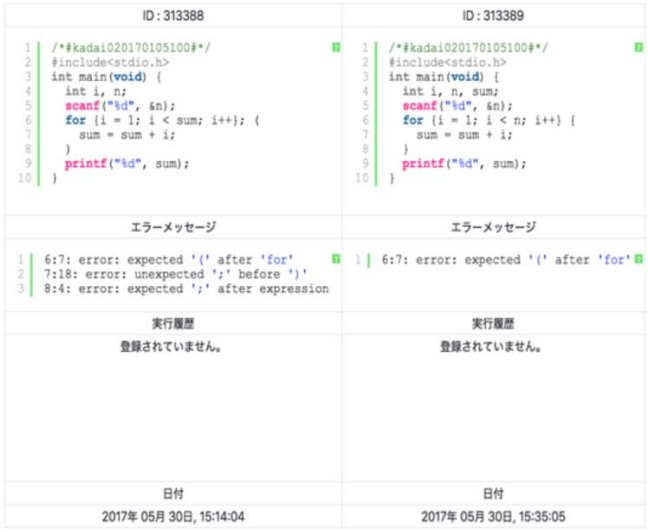

Fig. 2. Screenshot of stduent's learning history

Fig. 3. Screenshot of source code editing differences

b) *Source Code Editing Differences Viewing Sub-Function*

In this sub-function, it is possible to view the source code editing differences before and after each compilation for a student's task.

Fig. 3 shows an example of a screen displayed by this sub-function. On the upper side of the screen, the kind of editing specifically done is displayed in token units. In this case, "i, " "=, " "0", and ";" are added, and the "comparing operator" is replaced. In addition, on the lower side of the screen, the kind of editing specifically done is displayed in source code units.

c) *Learning Situation Association Sub-Function*

In this sub-function, it is possible to correlate the source code editing differences with the learning situations on the basis of the data that can be described in subsections V-C-a and V-C-b. One learning situation is selected for one source code editing difference and associated with it. Fig. 4 shows an example of a screen displayed by this sub-function.

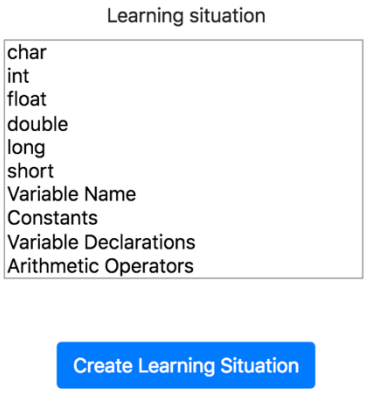

Fig. 4. Screenshot of learning situation association form

By utilizing these functions, it is possible to create training data that can be handled in this research.

VI. Development and Evaluation of the Learning Situation Estimation Engine

We first analyzed past source code editing histories of students to create training data. The histories of 1215 students belonging to the Department of Information Studies at our university were analyzed on the basis of the following task: create a void function that receives a decimal number and prints the number as a string.

Table V shows the outline of the learning situations in the created 1026 training data sets. The learning situation IDs in this table corresponds to the IDs shown in Table I.

TABLE V. Overview of Training Data

Learning situation ID	*Number*	*Learning situation ID*	*Number*
1	50	18	18
2	61	19	75
3	24	23	2
9	84	38	41
10	169	55	2
11	131	56	3
12	19	63	332
16	15	**Total**	**1026**

Using these training data, we developed the learning situations estimation engine using Azure Machine Learning Studio. The multiclass classification algorithms shown in Table VI were used for the following analysis.

In this section, we consider the possibility of learning situations estimation by applying algorithms considered to be optimal on the basis of the evaluation results for each algorithm.

A. Evaluation of Each Algorithm

The machine learning model obtained by each algorithm is evaluated using K-fold cross-validation and multiclass log loss (MCLL). The results are shown in Table VI.

TABLE VI. RESULTS OF EVALUATION

Algorithm Name	*Mean MCLL*
Multiclass Decision Forest	0.044
Two-Class Decision Forest	0.057
Two-Class Locally-Deep SVM	0.063
Two-Class Decision Jungle	0.072
Two-Class Averaged Perceptron	0.085
Two-Class Boosted Decision Tree	0.116
Two-Class SVM	0.170
Two-Class Bayes Point Machine	0.199
Multiclass Logistic Regression	0.459
Two-Class Logistic Regression	0.534
Multiclass Decision Jungle	0.620

The mean MCLL represents the average value of each MCLL in K times (10 times in this research) when K-fold cross-validation is used in each algorithm.

From the evaluation results, it is suggested that by using the Multiclass Decision Forest algorithm, estimation of a student's learning situations can be performed from source code editing differences with high accuracy.

B. Inspection

We consider the possibility of the learning situations estimation when using the Multiclass Decision Forest algorithm. By using K-fold cross-validation, we could evaluate 1026 training data sets.

Fig. 5 shows the results of summarizing the estimated number of true classes in each training data set.

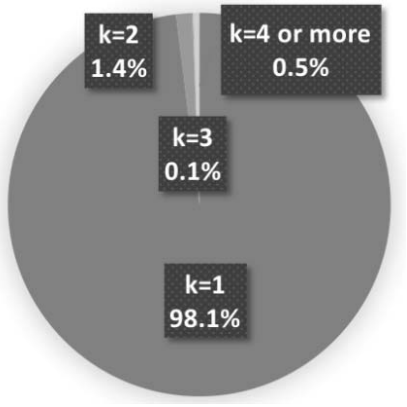

Fig. 5. Source Code Editing Difference Browsing Screen

From these results, it can be considered that a class can be classified into one corresponding learning situation, even if the input is the source code editing history of new students.

Therefore, the applicability of this learning situation estimation method in actual practice lessons was shown.

VII. CONCLUSION

In this research, we developed a method to estimate learning situations from the source code editing history of students including those with compilation errors. As a result, a learning situations estimation engine was introduced by this method, which is based on an existing machine learning evaluation method. It was suggested that a learning situation can be estimated from the source code editing history of a new student with a high degree of accuracy.

In the future, we will examine the learning situation class and reduce a teacher's burden when preparing training data. And this time, we targeted procedural languages, however will We aim to further improve the estimation accuracy and to apply this method in programming learning lessons and evaluate it for new students.

REFERENCES

[1] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers, M. Á. Rubio, J. Sheard, B. Skupas, J. Spacco, C. Szabo and D. Toll, "Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies", Proceedings of the 2015 ITiCSE on Working Group Reports, pp.41-63, 2015.

[2] T. Kato and T. Ishikawa, "Realization of Functions of Assessing Learning Conditions in Learning Management Systems for Programming Practicum [In Japanese]", J.IPS Japan, vol.55, no.8, pp.1918-1930, 2014.

[3] K. Nozaki, Y. Morimoto, S. Nakamura and Y. Miyadera, "Development of a Sequential Pattern Mining Method for Analysis of Programming Learning History [In Japanese]", IEICE Technical Report, vol.112, no.300, pp.49-54, 2012.

[4] T. Kato, Y. Kambayashi, Y. Terawaki and Y. Kodama, "Analysis of Students' Behaviors in Programming Exercises Using Deep Learning", SEEL 2017, pp.38-47, 2017.

[5] A. Ahadi, R. Lister, H. Haapala and A. Vihavainen, "Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance", ACM International Computing Education Research, pp.121-130, 2015.

[6] A. F. ElGamal, "An Educational Data Mining Model for Predicting Student Performance in Programming Course", International Journal of Computer Applications, vol.70, no.17, 2013.

[7] K. Fujiwara, K. Uemura, H. Igaki, K. Fushida, H. Tamada, S. Kusumoto and H. Iida, "An Approach to Identify Pitfalls in Programming Exercise Using Snapshots [In Japanese]", JSSST, vol.35, no.1, pp.3-13, 2018.

[8] K. Ishiwada, Y. Morimoto, S. Nakamura, Y. Nakayama and Y. Miyadera, "Development of a Method for Analyzing Source Code Editing Processes to Estimate Students' Learning Situations [In Japanese]", IEICE Technical Report, vol.116, no.438, pp.75-80, 2017.

[9] S. Sosnovsky, T. Gavrilova, "Development of educational ontology for C-programming", International Journal Information Theories & Applications, Vol.13, No.4, pp.303-308, 2006.

[10] B. W. Kernighan, D. M. Ritchie, "The C programming language", 2006.

[11] P. Mateusz, and N. Augsten, "RTED: a robust algorithm for the tree edit distance", Proceedings of the VLDB Endowment 5.4, pp.334-345, 2011

[12] A. Ohashi, K. Nozaki, Y. Ito, Y. Morimoto, S. Nakamura and Y. Miyadera, "A Programming Learning Base System for Support Tools Utillzing Learning History [In Japanese]", IEICE Techinical Report, vol.113, no.316, pp.15-20, 2013.