Chapter 3

Dataset Preparation

Thirty students worked on a hands-on activity for data preparation during three long lab assignments for the Introduction to Computer Programming (ICP) (01CT0101) course in the first year of the B. Tech in Information and Communication Technology. All topics were covered in class before the assignment was given out. None of the students have programming background.

3.1 Programming Assignment Designing

Programming problem set divided into three different topics. Conditional statement, Looping statement, and Functions. Each topic further divided into five subtypes of it like conditional statement divided into IF conditions, IF-ELSE with Logical Operator, IF-ELSE Ladder, Switch Statement and Nested IF-ELSE with Logical Operator. Each subtype is provided with five different level of complexity problem type like Beginner, Easy, Medium, Advanced, and Expert.

Complexity Type	**Problem Definition**
Beginner	Write a C program to check if a number entered by the user is positive.
Easy	Write a C program to check if a number entered by the user is even.
Medium	Write a C program to check which provided number is bigger out of two integers.
Advanced	Write a C-Program which accepts two numbers and check whether given first number is divisible by second or not.
Expert	Write a C Program which accepts weekly_hours that worker worked and hourly_rate. Based on input, calculate the earnings by workers who are paid an hourly wage, with weekly hours greater than 40 being paid time and a half.

Table 3.1: IF-ELSE statement different complexity level Problem Set

Total 75 problem set are prepared for data collection. On every long hour day students were provided total 25 set of problems with specific test cases to solve using C programming. The problem is solved once its able to pass all test cases provided for it. Based on complexity of each problem in different topic and sub category number of trials and number of errors learner encounter were stored in dataset.

3.2 Programming Environment Setup

For executing program one computer in lab was designated as a server. To be accessible in lab Wamp server which is freeware was installed in server [64]. Open-source web application was installed in Wamp server to allow learner to execute their code. Setup was created just to provide access to same C programming compiler and environment for all participants.

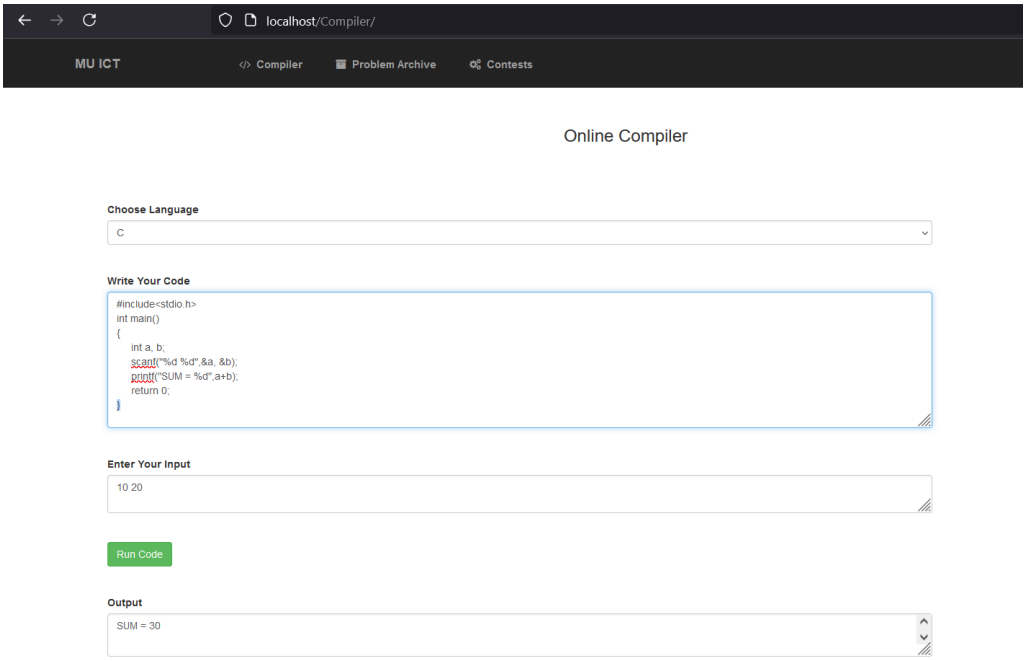

Figure 3.1: Programming environment setup using Wamp Server

3.3 Dataset Field Finalization and Data Collection

Dataset were stored in separated excelsheet of each learner. Field of dataset was selected with long vision of to prepare relational database. Followings are the field of dataset.

Data Field	Type	Content
studentId	Numerical	Unique Id given to each student from range 1 to 30
studentName	Characters	Name of student
topicId	Numerical	Unique Id given to each topic from range 1 to 3
topicName	Characters	Name of topic e.g. Conditional Statement
problemId	Numerical	Unique Id given to each sub topic from range 1 to 5
problemType	Characters	Problem type e.g. IF Condition
problemDefinition	Characters	Problem statement e.g. Write a C program to check if a number entered by the user is positive.
levelOfComplexity	Numerical	Unique Id given to complexity level from range 1 to 5. 1 – 5, low-high
complexityType	Characters	Five Type of complexity e.g. Beginner, Easy etc.
maxNoOfTrial	Numerical	Maximum number of trials allowed to solve the specific problem. Any positive integer
maxErrorAllow	Numerical	Maximum number of cumulative errors are allowed to solve the specific problem Any positive integer

Table 3.2: Initial dataset fields

Once the problem set is provided to students, they need to submit their data of performance. They have been asked to submit data like code successfully executed with all test case. They also need to submit number of trials they have taken to successfully execute code with number of errors they encountered during this process. These data will be added in to new field which will further used for performance predictions.

3.4 Features Engineering

Feature engineering is the preprocessing stage of supervised machine learning and statistical modeling, transforming raw data into a more effective set of inputs. An input will contain several features. This makes relevant information provided to models very potent for increasing their predictiveness and the accuracy in decisions made.

Our research identifies several relevant factors that influence the selection of the programming problem to be done next by the students.

- The count of failed trails
- The quantity of error detected in the problem solution

The learner is provided with a predefined number of trials for each task, which increases as the complexity level of the code progresses. The model suggests a new problem to the learner in two different scenarios. First, when the learner successfully executes the code, indicating a sufficient level of knowledge. Second, when the remaining number of trials for a particular problem reaches zero, encourages the learner to move on to a new challenge.

By incorporating these factors and implementing a dynamic recommendation system, we aim to provide learners with a well-rounded and advanced learning experience in programming.

The count of failed trails: The model considers the number of unsuccessful attempts made by the learner while solving a particular problem. This helps gauge the learner's progress and determine if they need to continue with the same problem or move on to a new one.

To determine the correctness of a solution S, we rely on the outcome of its test cases. A solution is considered correct only if all the test cases pass. To represent this, we define a correctness score, denoted as $f_c(S)$. If all the test cases pass, the maximum correctness score can be 1, indicating a correct solution. However, if even a single test case fails, the correctness score is set to 0, indicating an incorrect

solution. By looking at the complexity of the task, some numbers of trials are provided to the learner, t_{max}. Every task is set with a maximum number of trials that the learner can take, t_{max}. Leaner can try to code the perfect solution for t_{max} times. As the value of actual trial t_{act} will increase, it will decrease the correctness score $f_c(S)$ from 1 to 0.

Calculation of the correctness score, $f_c(S)$ will be calculated as follows.

$$f_c(S) = \frac{t_{max}}{(t_{max} + t_{act})} \text{ , if } t_r >= 0,\ t_r = t_{max} - t_{act} \qquad \text{Eq. (1)}$$

The quantity of error detected in the solution: The model analyzes the types of errors encountered in the learner's code. By understanding the specific areas where the learner is struggling, the model can suggest programming problems that target those areas for improvement.

$$f_e(S) = \frac{e_{max}}{(e_s + e_{max})} \qquad \text{Eq. (2)}$$

where e_S represents the number of compilation errors encountered for program S, and e_{max} is the maximum number of compilation errors under consideration [42].

The error score is a measure of how well a program performs in terms of compilation errors. It is derived by dividing the maximum numbers, e_{max}, of allowed errors by sum of maximum number of error and actual error, e_S, come across while reaching to solution of provided task. This normalization ensures that the error score falls within the range of 0 to 1, with higher scores indicating fewer compilation errors and better performance.

By using this error score metric, we can assess the quality and reliability of a program based on the number of compilation errors it exhibits, relative to the maximum number of errors observed among all programs [42].

Final Performance Calculation: The final performance of learner for specific task from successful submission can be determined using a linear function that incorporates three types of scores mentioned earlier:

$$p(S) = w_c * f_c(S) + w_e * f_e(S) \quad \text{Eq. (3)}$$

In this equation, the weights (w's) are assigned based on the importance of each score type. These weights are learned through a linear regression model. To ensure that the weights meet a constraint, their sum is set to 1, i.e., $w_c + w_e = 1$ and it is important to note that the performance p(S) falls within the range of 0 and 1 [42]. To suggest next programming task, the performance will be compared with different levels of performance value.

3.4.1 Mathematical model-based evaluation

Evaluation can be done based on task submission results. Here we are discussing each test case of programming task submission evaluation.

Code did not pass all the test cases: This situation can occur when there is an error in the code or when the program's output does not match the expected result. In such cases, the learner is typically given a set number of attempts to successfully complete the code. If the remaining trials reach zero, the learner will receive a code complexity level that is less challenging for the same task. However, if the learner has already reached the lowest complexity level, c1, for the task, it becomes mandatory to complete that task and achieve an acceptable level of complexity c_i. Here, i is an acceptable level of complexity set by the instructor, which is ≥ 1 and $\leq n$, where n is the highest level of complexity same for each topic which is 5 in our study.

Code passed all the test case: If code gets passed from a predefined set of test cases successfully, then the final performance calculation will start. For measuring the performance of the learner, the instructor can set a benchmark value between 0 to 1, based on which proper decision can be made. These benchmarks are P_{accept}, P_{min}, and P_{max}. P_{accept} is the minimum acceptance level,

if the performance score, p(S), is less than it, then the learner will be provided a lower complexity task on the same topic. P_{min} is the next acceptance level, if the performance score, p(S), is more or equal to P_{min}, then only the learner will be able to move to n.

Here, if the performance value is more than or equal to P_{min}, less than P_{max} and the current complexity level is greater than or equal to c_i, where then the next topic task with the same complexity can be given to the learner. If the performance value is more than P_{max} then the next topic task with a higher complexity level will be assigned to the learner.

Code Submission Status	**Complexity Level, c_i**	**Number of remaining trials t_r**	**Model Decision**
Did not pass all test cases	$c_i, i \neq 1$	$t_r > 0$	Same topic, Same task with the same Complexity level c_i
	$c_i, i > 1$	$t_r = 0$	Same topic, New task with Complexity level c_{i-1}
	$c_i, i = 1$	$t_r = 0$	It mandatory to complete the task or another task for the same topic with the same complexity level c_i may be assigned.

Figure 3.2: Code did not pass all test cases

Code Submission Status	**Task Level, T_k**	**Complexity Level, c_i**	**Performance Score, p(S)**	**Model Decision**
Passed all test cases	T_k, $1 \leq k < n$	$c_i, i = 1$	$p(S) < P_{accept}$	Same task with the same Complexity level c_i
		$c_i, 1 < i \leq n$	$p(S) < P_{accept}$	Next task with Same Complexity level c_i
		$c_i, 1 \leq i \leq n$	$P_{accept} \leq p(S) < P_{min}$	Same task with Higher Complexity level c_{i+1}
		$c_i, 1 \leq i < n$	$P_{min} \leq p(S) < P_{max}$	Next task with the same Complexity level c_i
			$p(S) \geq P_{max}$	Next task with Higher Complexity level c_{i+1}
	$T_k, k = n$	$c_i, 1 \leq i < n$	$P_{min} \leq p(S) < P_{max}$	Next Topic, New task with Complexity level c_i
			$p(S) \geq P_{max}$	Next Topic, Next task with Higher Complexity level c_{i+1}
		$c_i, i = n$	$p(S) \geq P_{max}$	Next Topic, Next task With the same Complexity level c_i

Figure 3.3: Code passed all test cases

The model decision is based on various factors, including task level (T_k), complexity level (c_i), performance score (p(S)), and model decision. When all test cases are passed, indicating successful completion of a task (T_k) with complexity level c_i and a performance score below P_{accept}, the student proceeds to the same task with the same complexity level. If the performance score remains below P_{accept} for subsequent tasks with the same complexity level (c_i) but different levels (i), the student progresses to the next task with a lower complexity level. Conversely, if the performance score falls between P_{accept} and P_{min}, the student continues with the same task but at a higher complexity level (c_{i+1}). If the score exceeds P_{max}, the student advances to the next task, either with the same or higher complexity level, depending on the task level and completion status. Finally, if the performance score surpasses P_{max} at the final task level (k=n), the learner moves on to the next topic, tackling a new task with the corresponding complexity level (c_i).

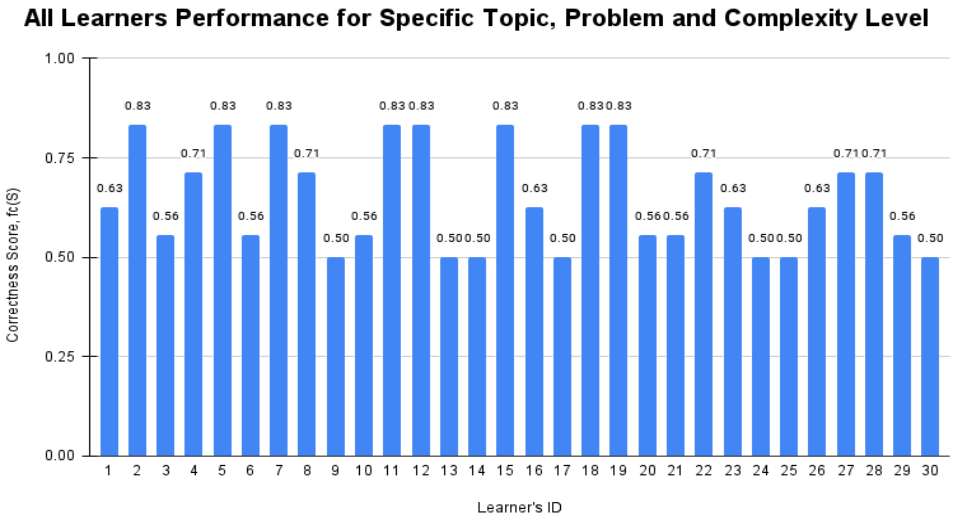

Figure 3.4: Correctness Score, $f_c(S)$

In Fig. 3.4, we can see the correctness score, $f_c(S)$, of all learners for the task with the same topic and complexity level. Instructors can get valuable insights from Fig 3.4., like the number of unsuccessful attempts to complete the task. This information becomes more useful for the instructor in teaching learners how to overcome logical errors.

In Fig. 3.5, we can see all learners' error scores, $f_e(S)$, for the task with the same topic and complexity level. Instructors can learn about the types and number of errors, specifically syntactical errors, that learners face during solving tasks. This information helps the instructor in teaching programming more effectively.

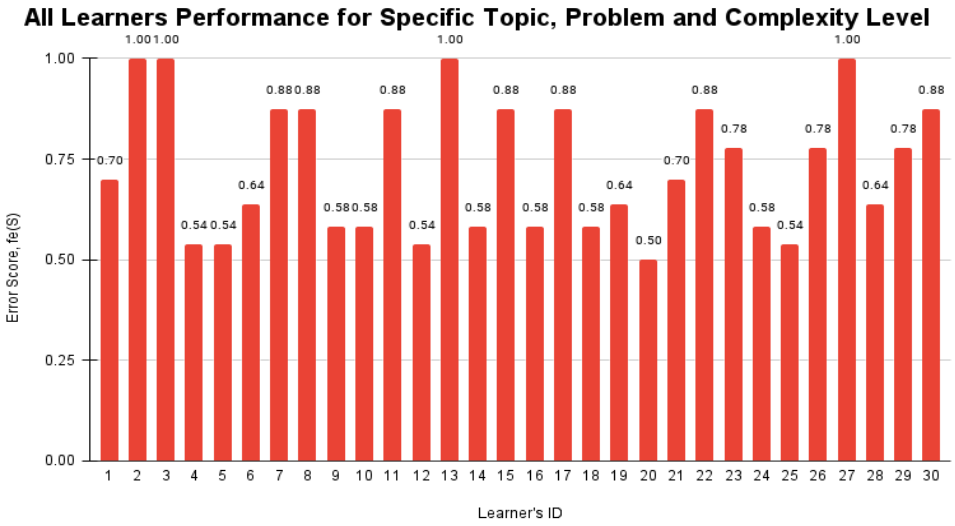

Figure 3.5: Error Score, $f_e(S)$

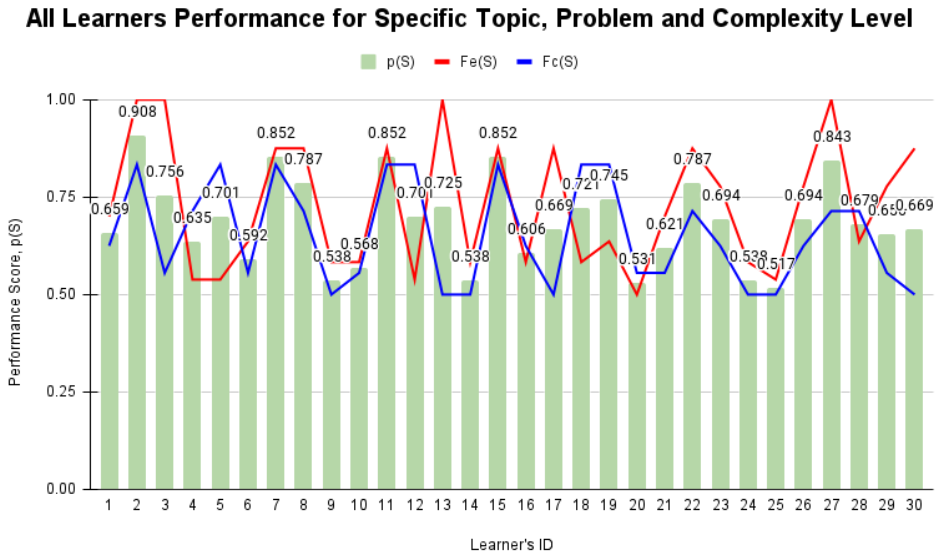

Figure 3.6: Performance Score, p(S), for all 30 learners

Fig. 3.6 shows the performance score, p(S), calculated using equation (3), relation with Correctness Score, $f_c(S)$, and Error Score, $f_e(S)$.

In Fig. 3.7, we can see the performance score, p(S), of one specific learner that shows how the learner performed in the course. It depicts the performance of learners in specific topics at different complexity levels.

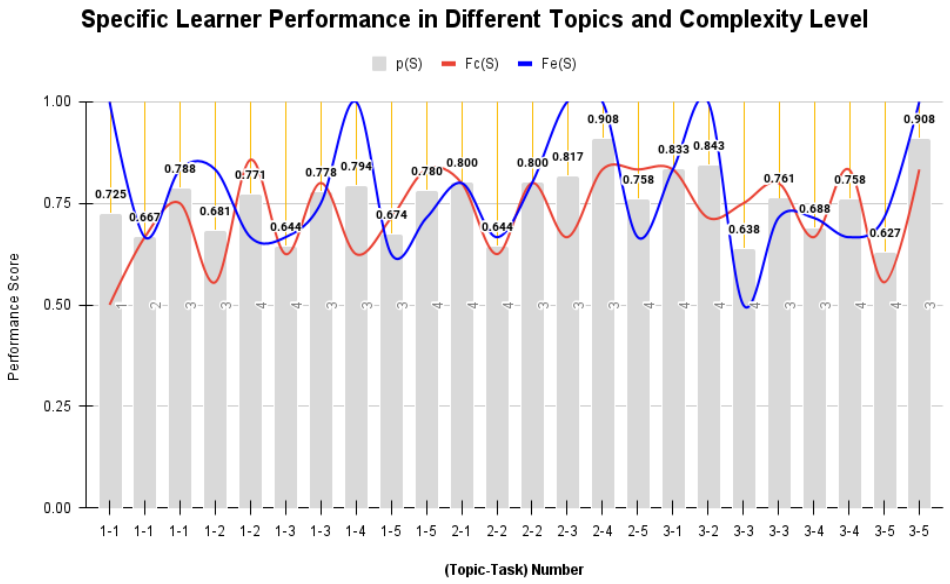

Figure 3.7: Specific Student's Performance in all topics