ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

DISCOVERING PATTERNS IN PISA

SEMESTER PROJECT IN DATA SCIENCE

Ajkuna Seipi
Supervisors: Jade-Maï Cock, Dr. Laila El-Hamamsy
Professors: Prof. Tanja Käser, Prof. Engin Bumbacher

8th June 2024

Abstract

Computational thinking is a vital skill for critical and logical reasoning, applicable beyond computing to various life and work areas. Recognizing this, PISA - *Program for International Student Assessment* - introduced block-based programming challenges.
At the ML4ED laboratory at EPFL, in collaboration with the HEP of Lausanne, we analyzed students' interactions with the RobotArm programming game from PISA 2025 to explore their reasoning patterns in three learning tasks.
Our study emphasized analyzing the process of block-based programming rather than just the outcomes, as focusing solely on output covers different reasoning and learning trajectories. We introduced a distance metric for the game's world space to understand spatial relationships and movements, comparing it to the code space to link programming actions with in-game effects. Through Sequential Pattern Mining and Correlation Prediction experiments, we identified distinct patterns between successful and unsuccessful students, revealing different strategies and reasoning processes. Based on these findings, we developed a success predictor model using identified patterns and metrics. However, the model's reliance on final attempts limits its ability to predict failures early, reducing its effectiveness in providing timely feedback.

Chapter 1

Introduction

Computational thinking is a crucial skill in today's world, encouraging young learners to think critically and logically. This skill is not only essential in computing but is also easily transferable to many areas of life and work. Recognizing its importance, PISA - *Program for International Student Assessment* - has introduced block-based programming challenges in its program.

Computational thinkinkg skills in PISA 2025 are evaluated through 7 programming exercises, including "RobotArm", "NaturePod" and "DigitalArt". Within those exercises, students engage in programming exercises and should attain a goal state within a grid-like world. Students can iteratively construct solutions, test their code, and receive a feedback in case of error.
Each programming exercise is composed of three learning tasks, and a challenging task which is more difficult. Students also take a pre-test quiz before taking part of the learning tasks, to assess their knowledge and critical thinking skills in programming.

The challenge in interactive teaching lies in the evaluation of the student's work. In fact, the output fails to provide sufficient information to understand the issues faced by students. In general, the tasks have multiple solutions, therefore relying uniquely on the output is not enough to evaluate the student's reasoning and computational thinking.
Furthermore, having multiple solutions also means that some of them are more optimal than others. This indicates the level of comprehension of a student regarding the task, and more specifically its ability in computational thinking. This approach would give a more fine grained clustering of the students.

In this project, we aim to analyze students' reasoning by discovering patterns in their block-code programming. We approach this by examining their progress through the code space and the world space separately, focusing on the distances in each space. Our goal is to identify successful students by recognizing optimal reasoning through their code space distances and the resulting world space distances. Analyzing both spaces provides deeper insights into students' reasoning, allowing us to identify patterns that lead to successful or unsuccessful outcomes.

The project is divided into three parts:

1. How do we measure progress in code and world space?
2. Is this progress consistent in both spaces?
3. Can we predict success using these spaces?

Several research have focused on analysing the student's behavior in block-based programming languages. One relevant work is the one from (Jiang and al. 2022). The work focused on identifying the most common mistakes made by students on the path to the optimal solution, based on the code trajectories

of students' attempts. But, the study focus solely on the evolution of student code attempts towards the optimal solution, without exploring alternative solutions that students might have found, which is limiting and does not consider the teaching aspect of the task.
Another notable study is the one from (Emerson and al. 2019). It focuses on predicting the student's success with four categories of features, which are the following: prior performance, hint usage, activity progress and interface interaction. However, the study does not focus on the analysis of the choice of block-code. Indeed, as highlighted in the discussion part of the study, "*A final limitation of this work that should be investigated in future work is level of granularity at which analyses of student code is conducted*". (Emerson and al. 2019)

The contribution of this work is an intersection of both studies, which is a prediction of the success of a student in a task by analysing thoroughly the block-code. The approach is innovative, because we not only focus on the block-code distances, but on the resulted outcome in the grid-like interface. Combining the two distances will surely highlight new patterns related to success or failure of students.

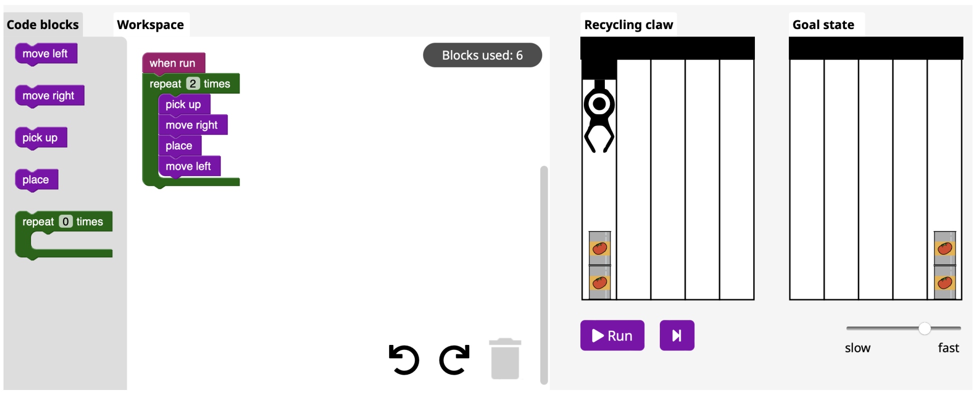
CHAPTER 2

METHODOLOGY

To evaluate progress, we first developed an appropriate distance metric for the world space. We then analyzed the correlation patterns between distances in both spaces using Sequential Pattern Mining and predictive models.

2.1 ROBOTARM

In this project, we focus on the three learning tasks of "RobotArm" programming exercise. The students need to instruct the arm of the robot to pick up and dispose the elements of the grid-like world, such as their spatial disposition is similar to the goal state represented. The figure 2.1 is an example of the interface of the RobotArm exercise. On the left, we observe the block-code environment, that we will refer to as the *code space*, and on the right, we observe the grid-like environment representing the goal state, that we will denote as the *world space*.



```
when run
repeat 2 times
    pick up
    move right
    place
    move left
```

FIGURE 2.1
Interface of the Task 1 from RobotArm;
Left: Block-code environment. Right: Grid-like environment.

There are in total five blocks of code:

1. `move left` : The robot arm moves to the left by one step.
2. `move right` : The robot arm moves to the right by one step.
3. `pick up` : The robot arm picks up an object.

4. `place` : The robot arm places an object.
5. `repeat x times` : The for loop to repeat the actions.

There are two distinct objects; a can and a glass. The number of objects varies depending on the task configuration. Additionally, there are 4 types of programming errors:

1. `move1` : The claw hit the wall.
2. `move2` : The claw had a collision with one of the objects. The objects are stacked too high.
3. `pickup1` : The claw is already holding an object, so it cannot pick up one.
4. `place1` : The claw is not holding any object, so it cannot place one.

Each time a student runs their code or adds a block to it, an XML log file is generated by the application. This file contains the entire block-based code in the student's workspace. All the XML log files are then stored in a unique JSON file, which includes the student's identifiers, scores, and other relevant data. This preprocessing step was completed by Joanna Wolski in a previous semester project. (Joanna 2023)

The world space is represented as a string vector. We observed that certain types of logical errors could be easily identified through the world space, specifically when a `place` block is missing at the end of an attempt.

To enhance further analysis, we consider the **case space** which includes $logical$ and $programming$ errors. More specifically:

1. **Case 1** : The case where a $programming$ and $logical$ error took place.
2. **Case 2** : The case where a $programming$ error took place.
3. **Case 3** : The case where a $logical$ error took place.

2.2 STUDENTS' DATA

All students taking part of the activities are around 15 years old. We also assume that none of them did the tasks before.

As observed in Figure 2.2, the proportion of successful students and students that fail is rather imbalanced across the tasks. In fact, a huge proportion of the students that participated in the learning task 1 succeeded, because of its low level of difficulty. Thus, analysing the task 1 is not relevant to the project, since the patterns observed could be misleading because of the little proportion of the failing group. Therefore, learning task 2 and 3 should be more relevant to our analysis. Still, since the dataset size is quite small, we continue to analyze the three tasks, keeping in mind the group imbalance.

On average, students make around 5 attempts in total. Successful students tend to make more attempts. Conversely, there is a significant portion of students who fail with only one attempt. For this project, we opted to exclude students who made only one attempt as it would not provide substantial insight into their reasoning process.

2.3 METRICS ANALYSIS

We tried to find suitable metrics to analyze and evaluate the changes in the block-code and the resulting changes in the world space.

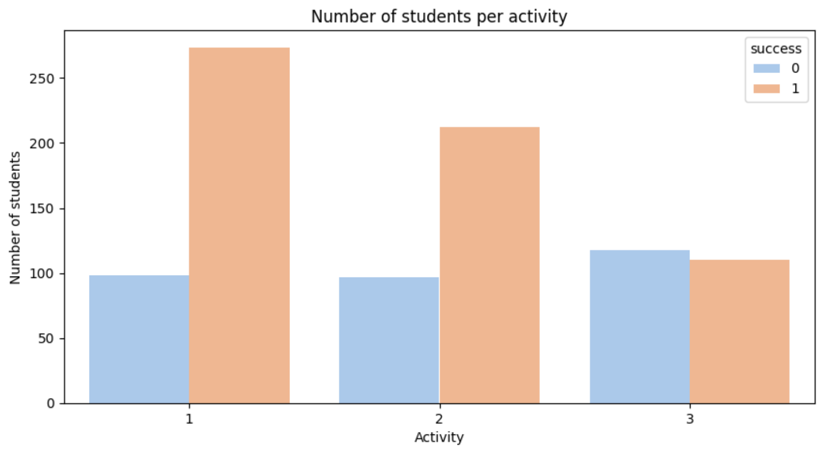


FIGURE 2.2
Number of students per Task

2.3.1 CODE SPACE

For the code space, we build on the work done in (Joanna 2023). The XML structures are used to create Abstract Syntax Trees (ASTs).

To compute the distances between the ASTs, the Tree Edit Distance algorithm was employed. As noted in (Sager and al. 2006), the authors concluded that the Tree Edit Distance algorithm performs the best for this purpose.

2.3.2 WORLD SPACE

We focused on implementing an appropriate metric to compute the distances in the world space. Four types of metrics were considered (Slimani 2013):

1. **Euclidean distance** : A metric used to compute the distance between two points.
2. **Levenshtein** : An edit-based algorithms, which measures the minimum number of single character operations required to transform one string into another.
3. **Jaro-Winkler** : An edit-based algorithms, which considers the number of matching characters and the position of those characters.
4. **Ratcliff-Obershelp** : A sequence-based algorithm, which focuses on finding the similarity between two strings based on their common substrings.

First, we compare these metrics by simulating experiments in a grid-like environment with two different elements. We test various configurations and evaluate the outcome of each metric. The metric that produces the best results should be selected for further analysis.

Additionally, to justify the choice of metrics, we compute the Pearson correlation coefficient between the world space distances and the code space distances.

We also add a **penalty** to the distance when a $logical$ or $programming$ error happens:

1. **Logical errors**: The penalty corresponds to the distance between the stuck object in the claw and the position of the missing object in the goal state vector.
2. **Programming errors**: The penalty used is 0.1.

Adding a penalty brings more information on the block-code state, thus increasing the correlation between the distances in both spaces.

Finally, for each student in a specific task, we compute two types of distances: 1) the distance between the world vector at timestamp t and the goal state vector, referred to as "world distance", and 2) the distance between the world vectors at timestamps $t-1$ and t, referred to as "successive world distance".

2.4 PATTERN MINING

To identify patterns that could differentiate the trajectories of students who successfully completed a task from those who failed, we can use **Sequential Pattern Mining (SPM)** to analyze input sequences.

The SPM algorithm used is the one described in (Kinnebrew and al. 2013). The advantage of this SPM algorithm is its ability to handle sequential features effectively. The SPM algorithm consists of two main steps:

1. Computes the student support for each possible pattern.
2. Computes the interaction support for each pattern for each student.

The *student support*, also known as the *s-support*, is the proportion of the students which exhibits a particular pattern. Hence, we computed the s-support for each pattern and filtered the ones that appear in more than 50% of the students, by setting a threshold of 0.5. In fact, we want to observe patterns that appear in the majority of the students, thus by choosing a threshold of 0.5, we ensure that the patterns frequently appear in the group.
The *interaction support*, *i-support*, is computed per student per pattern and corresponds the number of times the interaction pattern appears without overlap in each student's sequence.

2.4.1 SEQUENCES PROCESSING

To identify a maximum number of patterns that could indicate success prediction in a task, we apply sequential pattern mining (SPM) on four sequences: **error sequences**, **case sequences**, **world distance sequences**, and **code distance sequences**.

Applying sequential pattern mining to the distances can help identify similarities and differences between the block-code distance and the world space distance. The pattern mining is applied to successive distances.

As a first step, we need to pre-process the distances into categorical values to apply sequential pattern mining. We decide to split the distances into two buckets: *high* and *low*. *High* indicates larger changes between a distance at $t-1$ and a distance at t, while *low* indicates smaller changes.

To split the continuous distances, we consider two methods:

1. **Distribution split** : We split the distances based on a threshold determined by the distribution. In our case, the distribution of the successive distances forms an elbow-like shape. Thus, the split point is at the sharp value of the elbow.
2. **Median split** : We split the distances according to the median of the successive distances.

To note: The thresholds are computed separately for each task and are applied to the successive distances of that task. This is necessary because the world environments differ significantly across tasks, requiring distinct threshold splits for each one.

Four experiments are conducted between different groups of students: 1) **success and failure groups**, 2) **across tasks groups**, 3) **error and non-error groups** and 4) **first and last attempts groups**.

2.5 PREDICTIVE MODELS

The last phase of the project involves predictive models. We use them for two analysis: **metrics correlation** and **success prediction**.

2.5.1 METRICS CORRELATION

We aim to predict the successive world distance at attempt n by considering as features the successive code distance, the error or the case space, and the successive world distance until a maximum of $n-1$.

We considered as features the sequences generated for the pattern mining analysis. This means that the distances are sequences of *low* and *high*, errors are sequences of *error* and *none*, and cases are sequences of *case* and *none*. The distance sequences are processed into categorical values with a distribution split-like.

The inputs of the model are all the sequences of a student until their n_{th} attempt. The different sequence types are concatenated together.

We consider as **Baseline** the model which takes as features the successive world distances until attempt $n-1$, and which takes as label the n_{th} attempt of the successive world distance.

To note :

- Successive world distance sequences are represented by w.
- Successive code distance sequences are represented by cod.
- Error sequences are represented by e.
- Case sequences are represented by cas.

We represent the sequence up to attempt n as $w_{:n}$, which corresponds to the successive world sequence up to attempt n. The sequence at attempt n is represented as cod_n, corresponding to the successive code sequence at attempt n. Concatenation of sequences is denoted by $+$. For each experiment, the target is the successive world distance sequence at attempt n, e.g., w_n.

We compared three classification models:

- **Logistic Regression** with the following hyper-parameters: $epochs = 200$, $learning\_rate = 0.01$, $weight\_decay = 0$, and $dropout\_rate = 0$.
- **Random Forest** with the default hyper-parameters of the Scikit-Learn library.
- **LSTM** with the following hyper-parameters $epochs = 150$, $learning\_rate = 0.05$, $weight\_decay = 0$, $dropout\_rate = 0.1$ and $hidden\_dimension = 100$.

Several experiments are conducted where the choice of features varies considerably. The objective is to identify the sequences that are most relevant for the predictive model, i.e., which give the best results.

Finally, the model performance is evaluated using 5-fold cross-validation by computing the mean accuracy and mean AUC. In our case, AUC is more relevant since it provides a better measure of the model's ability to distinguish between the different classes, particularly in imbalanced datasets. This helps in understanding how well the model can predict the successive world distance, considering both true positive and false positive rates.

2.5.2 SUCCESS PREDICTION

Last but not least, we experiment with different architectures to evaluate the model's ability to predict students' success in a particular task. The problem is framed as a classification task.

The features are the successive code distances, the successive world distances, and the error space or the case space until the n_{th} student's attempt. The model should predict, based on these sequences, whether a student will succeed or fail.

Once again, we consider training and evaluating the following three models with the same hyperparameters as explained in 2.5.1.

SEQUENCE TRUNCATION

To identify which attempt or which sub-sequence is most relevant for accurately predicting a student's success, we truncate the sequences step-by-step using a parameter n.

Two methods are used:

1. n_{th} **truncation method**: If the mode is *first*, the method removes the first n attempts from a student's sequences. If the mode is *last*, it removes the last n attempts from a student's sequences.
2. **sub-sequence truncation method**: If the mode is *first*, the method keeps only the first n sub-sequences of the student's sequence, e.g., with $n = 3$, only the first 3 attempts are kept. If the mode is *last*, the method keeps only the last n sub-sequences of the student's sequence, e.g., with $n = 3$, only the last 3 attempts are kept.

Both methods are evaluated iteratively by modifying the n parameter and the $mode$ parameter with similar evaluation as in 2.5.1.

As a baseline performance, we consider the results found in 2.5.2.

Chapter 3

Results

The analysis was conducted on the three learning tasks of the RobotArm programming game. In this section, we outline the analysis and results obtained.

3.1 World distance metric

The selection of the distance metric relies on numerous test cases. Distances are calculated between the vector's state at timestamp t and the goal state vector.

However, as illustrated in Figure 3.1, it's evident that the Euclidean distance aligns more closely with the desired curve. In Figure 3.2, which depicts two cans, one atop the other, each timestamp (e.g., (1)) represents the vector's state at that specific moment. Clearly, the arrangement of the cans at each timestamp gradually approaches the configuration of the goal state. Consequently, the distance curve should exhibit a linear decrease over time.
Indeed, the layout of the grid-like interface appears to be better suited for spatial distances, such as the Euclidean distance, rather than edit-based sequences or sequence-based distances. Hence, we opted to maintain the Euclidean distance as our chosen world-distance metric.

3.2 Error Analysis

Figure 3.3 illustrates the significantly relevant patterns observed across different student groups. The y axis displays these patterns, while the x axis categorizes the groups; for example, *l1_success* represents students who succeeded in task 1, and *l1_failure* denotes those who failed task 1.

The heatmap bar indicates the *s-support*. The *i-support* is shown as text within the corresponding heatmap box.

From the figure, we notice that students generally make clustered errors rather than successive ones, as evidenced by the $none \rightarrow case$ pattern. Additionally, the pattern $case \rightarrow case$ is prevalent among students who failed both tasks 1 and 2. For error patterns, $error \rightarrow error$ is unique to task 1 and may be an indicator of failure. However, since this pattern does not appear in task 3, it might not be generalizable to all configurations.
The distinction between case patterns and error patterns lies in the inclusion of $logical$ errors in the case set, highlighting their significance in revealing patterns beyond programming errors.

Another noteworthy finding is the pattern $case \rightarrow none$, which appears exclusively among failing students across all tasks, making it a crucial predictor of success. A similar pattern, $error \rightarrow none$, is

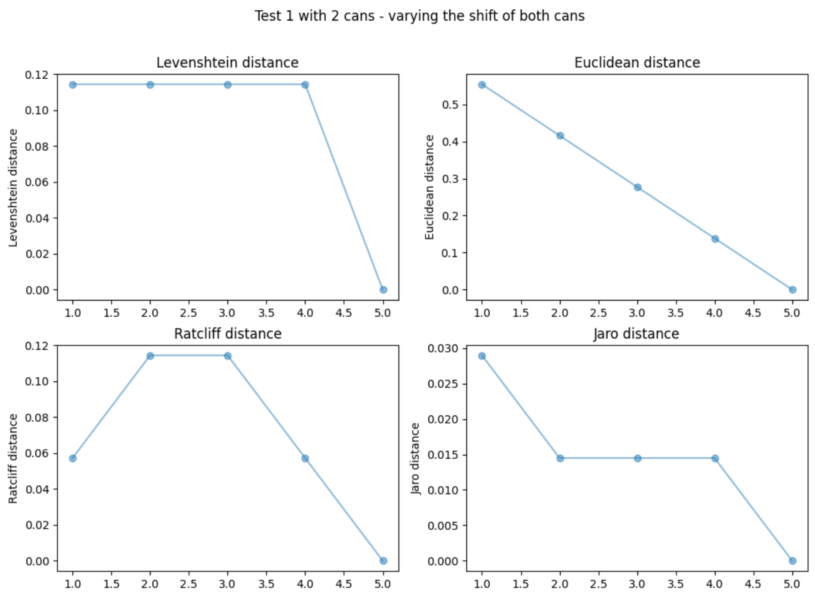


FIGURE 3.1
Metrics comparison - Test 1

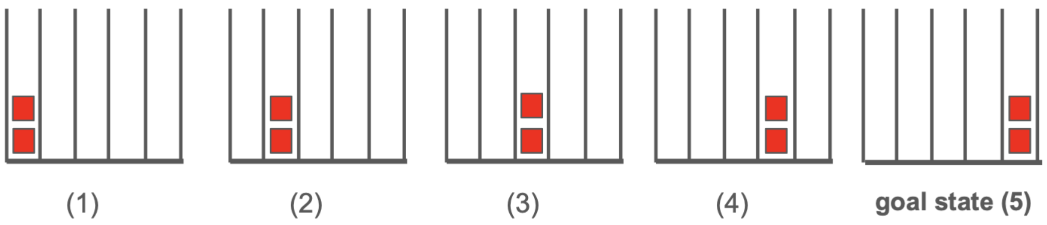


FIGURE 3.2
Grid-like world space interface - Test 1

also observed.

Further investigation reveals that these patterns are more significant in the latter half of student attempts. This suggests that while successful students also make errors, they distinguish themselves by overcoming these errors, unlike the failing students.

3.3 DISTANCES CORRELATION

Now that we have established a suitable world space metric, we can proceed with the analysis by comparing the trajectories of the code space and world space metrics.

Figure 3.4 illustrates the curves for the successive world space distances (top) and code distances (bottom) for Task 2. The left plot shows the evolution of these successive distances. Generally, the curves exhibit similar patterns of change, though there are notable differences. Specifically, the world space distances tend to vary more than the code space distances.

The right plot in Figure 3.4 presents the successive distances for students who made a total of 5 attempts

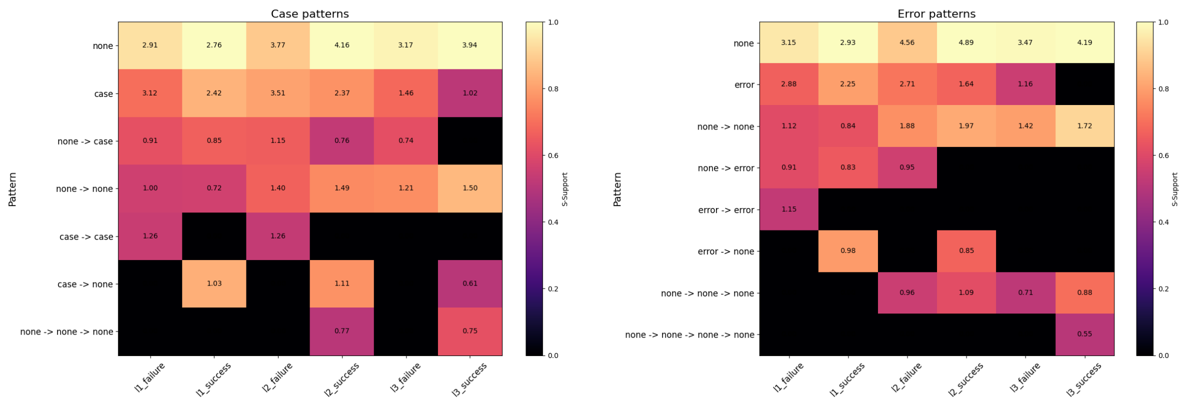


FIGURE 3.3
Heatmaps of Case and Error patterns.

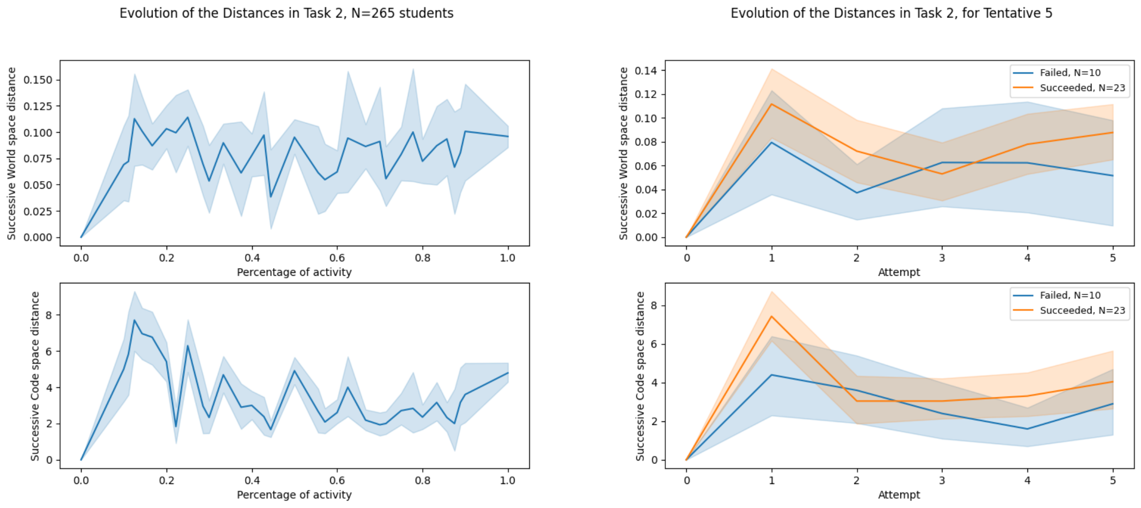


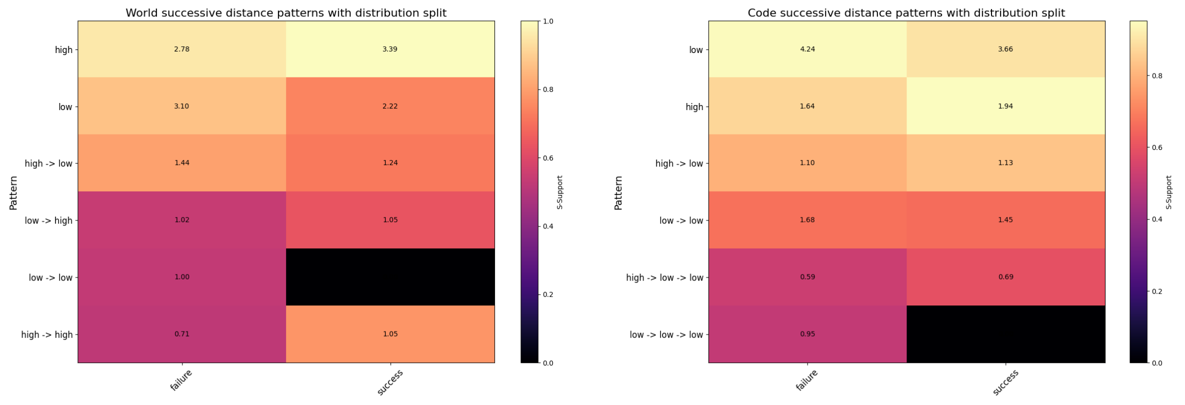
FIGURE 3.4
Code and World space distances

in Task 2. Here, we observe significant differences toward the end of the attempts for failing students. The world space distances tend to decrease toward the end, while the code space distances appear to increase.

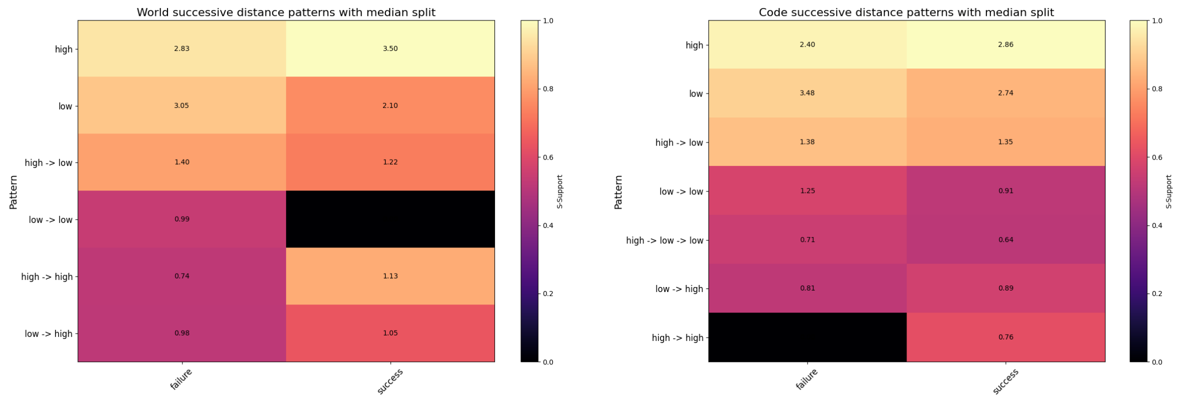
We performed a Pearson's statistical test to compute the correlation coefficient between the successive world distances and successive code distances. In Table 3.1, the p values for each task are very small, indicating a highly statistically significant correlation between the metrics. Task 3 shows the highest correlation coefficient. This could be attributed to Task 3 having a more balanced set of students, potentially resulting in less noisy behavior compared to the other two tasks, leading to more accurate results in the world distance metric.

	Task 1	Task 2	Task 3
p value	1.9×10^{-119}	8.6×10^{-119}	4.6×10^{-93}
Pearson's coeff.	0.478	0.475	0.541

TABLE 3.1
Pearson's coefficient for each task



(A) Distribution split



(B) Median split

Figure 3.5
Heatmaps of the pattern mining on the successive distances.

3.3.1 Pattern Mining

As a first experiment, we apply the sequence pattern mining on both distances. We compare the patterns with the two different split-type, namely median split and distribution split.

Figure 3.5 depicts a heatmap of significant patterns appearing in the groups of successful and failing students for successive distances of the world and code space.

When examining the patterns with a distribution split, the successive world distances (left) show a significantly higher frequency of $high$ patterns. Patterns such as $high \to high$ appear quite prominently among successful students, while the pattern $low \to low$ seems to indicate failure.

In contrast, when observing successive code distances with a distribution split (top right), the patterns differ notably. Despite the correlation observed between world and code distances, the patterns do not align as expected. Interestingly, low patterns are more predominant than $high$ patterns in both student groups. In the world distances, low patterns, such as those with low as a subsequent keyword, are more associated with failing students. However, in the code space, low patterns are also related to successful students, such as the pattern $high \to low \to low$.

This finding reveals an important fact about the relationship between code and world spaces: significant changes in the world space can be related to very small changes in the code space.

The bottom part of Figure 3.5 uncovers patterns of both distances when a median split is applied.

Interestingly, the patterns appearing in the successive code distances and successive world distances are more similar than those seen with a distribution split.

With a median split, we observe a greater number of $high$ patterns in the code space. This shift can be attributed to the threshold created by a median split, which is much smaller than that of a distribution split, resulting in more sequences with a large number of $high$ patterns.

3.3.2 PREDICTIVE MODEL

To evaluate the correlation between different spaces, we predict the successive world distance at attempt n by using other spaces as features.

The importance of this step lies in identifying which sub-sequence or space is most decisive for accurately predicting the successive world distance at attempt n.

To this end, we experimented with different feature sequences. Table 3.2 summarizes the results of these experiments.

	Features	**LR**	**RF**	**LSTM**
Baseline	$w_{:n-1}$	0.55 ±0.01	0.52 ±0.02	0.50 ±0.02
Experiment 1	$cod_{:n} + e_{:n}$	0.58 ±0.02	0.64 ±0.00	0.58 ±0.05
Experiment 2	$cod_{:n-1} + e_{:n-1}$	0.55 ±0.01	0.54 ±0.01	0.55 ±0.06
Experiment 3	$cod_{:n}$	0.64 ±0.01	0.62 ±0.01	0.63 ±0.01
Experiment 4	$cod_{:n} + e_n$	0.64 ±0.01	0.61 ±0.01	0.62 ±0.01
Experiment 5	$cod_{:n} + e_{:n} + w_{:n-1}$	0.59 ±0.01	0.63 ±0.01	0.55 ±0.04
Experiment 6	$cod_{:n} + e_{:n} + w_{n-1}$	0.58 ±0.02	0.65 ±0.01	0.58 ±0.07
Experiment 7	$cod_{:n} + cas_{:n}$	0.60 ±0.01	**0.66** ±0.02	0.61 ±0.04

TABLE 3.2
Mean AUC scores with standard deviation

The above notation is explained in 2.5.1.

Table 3.2 reveals that adding the n_{th} attempt of the code distance sequence significantly enhances the models' performance. However, adding $e_{:n}$ to the features seems to introduce more noise than useful information, while e_n positively impacts performance.

Surprisingly, successive world distances do not provide additional information. Similar to the error sequence, $w_{:n-1}$ introduces noise rather than improving performance. However, w_{n-1} appears to add useful information, as seen in the improved performance of the **RF** model in **Experiment 6**.

As expected, $cas_{:n}$ provides valuable insights and significantly boosts the performance of the **RF** model. This improvement is related to the patterns identified using the *Sequential Pattern Mining* method, which highlighted decisive patterns indicating student success or failure.

3.4 SUCCESS PREDICTION

The last step of the analysis is to build a predictive model for success prediction. The choice of the models remains the same.

Table 3.3 shows the performance of the models with distribution and median split applied to the successive world and code distances.

To note :

- **Error** refers to the following features: $w_{:n} + cod_{:n} + e_{:n}$.
- **Case** refers to the following features: $w_{:n} + cod_{:n} + cas_{:n}$.

The above notation is explained in 2.5.1.

	Distribution		*Median*	
	Error	**Case**	**Error**	**Case**
LR	0.82 ±0.02	0.82 ±0.03	0.83 ±0.03	0.84 ±0.03
RF	0.83 ±0.02	0.84 ±0.03	0.85 ±0.02	**0.87** ±0.04
LSTM	0.79 ±0.04	0.80 ±0.04	0.79 ±0.03	0.82 ±0.04

TABLE 3.3
Mean AUC scores with standard deviation

We notice that the best performance is attained using a Random Forest (RF) with median-split type distances and case sequences. Median split appears to yield better performance when used with both RF and LR. This could be explained by the higher correlation between the successive code and world distances that the median split highlights.

Additionally, features that include case sequences seem to be slightly more informative overall than those with error sequences.

3.4.1 MINIMAL SEQUENCE

The goal of the project is to identify students' success as early as possible to provide them with appropriate feedback at the right time.

Figure 3.6 depicts two plots of the mean AUC of the Random Forest (RF) model on **Case** features with a distribution split. As indicated, the model performance with the entire sequences has a mean AUC of 0.84.

The left plot shows the mean and standard deviation (std) of the AUC for the model when we apply the n_{th} *truncation method*. The x-axis corresponds to the n parameter of the method. As we observe, with $mode = first$, removing the first and second attempts of each sequence does not significantly impact the model's performance. However, from the third attempt onward, we observe a drastic decrease in the mean AUC curve.

A probable reason for this behavior is that truncating sequences simultaneously removes data. Specifically, truncating attempts up to n means that we completely remove the sequences of students with n total attempts. From the preprocessing of the data, we know that most students make an average of 5 total attempts per task; the average is even lower for failing students. Hence, by removing 3 attempts, we eliminate considerable data and information necessary for the model to predict accurately.

With $mode = last$, the mean AUC drastically decreases with $n = 1$ and continues to decrease quite linearly until $n = 3$. Beyond this point, the model performs similarly to a random one. This clearly indicates that the last attempts of the sequences have a significant impact on the success prediction of the model.

The right side of Figure 3.6 shows the mean and standard deviation (std) of the AUC for the model when we apply the *sub-sequence truncation method*. What is interesting to notice is the curve with

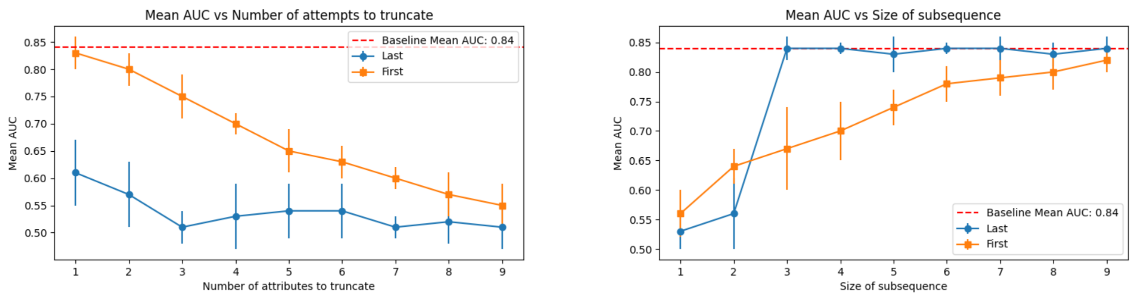


FIGURE 3.6
Mean AUC Variation with truncated sequences

$mode = last$, which depicts the weight of the n_{th} last sub-sequences of each sequence space. In fact, the curve drastically increases when $n = 3$, reflecting the significant importance of the last 3 attempts of a student.

This concludes that with the current features combination, we still lack enough information to have earlier signals to predict students' success accurately.

Chapter 4

Discussion

4.1 Limitations

4.1.1 Dataset Size

The rather limited number of students imposes a challenge on success prediction. First, the patterns identified might not generalize well. Second, since the predictive models are fed with sequences per student, we get a relatively small training set, which limits the models' capacity to learn sufficiently.

4.1.2 Tasks completion

We were curious about how students' behaviors varied across different tasks. Initially, we anticipated that if a student failed in one task, they would likely fail in subsequent ones as well. Surprisingly, we found that a small percentage of students who failed in earlier tasks managed to succeed in more challenging ones. Additionally, there were students who did not attempt the first task at all but tackled the subsequent tasks.

As a result, our sample of students is quite diverse, making it noisy and challenging to analyze their behaviors comprehensively.

One solution would be to concentrate on the samples of students who completed all three tasks. However, this would mean discarding a non-negligible amount of data. Having tested some of the models' performance with this subset of students, we did not achieve better results. In fact, their performance slightly decreased, highlighting the issue of insufficient data size.

4.2 Future Work

4.2.1 Predictive model enhancement

The predictive model performance could be investigated more thoroughly. First, some hyper-parameters tuning could be analyzed and applied, and which might positively impact the model performance. In fact, Random Forest classifier is used with the default parameters of the Scikit-Learn library.

Another analysis that could be performed is to transform the task of correlation prediction into a regression one. In fact, even though two different methods were used, splitting the successive distances into two distinct groups might have cover some significant patterns.

Therefore, maintaining the distances as continuous values and transforming the classification task into a regression one might help first to increase the models' performance, but might also help the model learn insightful patterns.

4.2.2 DISTANCE SPLITS ANALYSIS

As stated in 4.2.1, splitting the distances into two distinct groups might negatively impact the models' performance, especially for success prediction.

The two-split action might obscure more nuanced distances, potentially causing us to miss earlier signals for success prediction.

An alternative would be to optimize the split method. This could involve increasing the number of groups by splitting into **tertiles** or **quartiles**, or investigating different split methods to divide the data in another manner.

4.2.3 NEW FEATURES CONSIDERATION

With the current features and analysis, we conclude that the last attempts of a student are the most decisive for success prediction. This hinders our goal to identify failing students as early as possible, so that we can provide them with relevant feedback in time.

As observed through the analysis, students' sequences are quite attempt-dependent, meaning their behaviors vary significantly depending on the number of attempts they take to complete the task, whether successfully or not. The current models do not consider this dependency, making it challenging to generalize correctly.

We could overcome this challenge by considering another feature: the timestamps between each attempt. We could create a new sequence with the duration of time between each attempt. Previous analyses of the code space have shown that timestamp durations differ significantly between successful and failing students. Indeed, successful students tend to take more time between submissions.

This new feature could help discover success or failure patterns earlier, thus enabling the model to predict a student's success earlier.

CHAPTER 5

CONCLUSION

In this work, we proposed an analysis pipeline for uncovering patterns based solely on the block-code implementation to predict a student's success during a task as early as possible.

The project involved analyzing different spaces, namely the world space and code space of the student. Throughout the analysis, additional spaces proved to be significantly insightful for uncovering interesting patterns. Specifically, the analysis of error space and case space revealed distinct indicators between successful and failing students.

The world space was analyzed by first creating a suitable metric to compute the distances between each submission. To assess the choice of metric, we analyzed the correlation between the world space metric and the code space metric through various methods, including Sequential Pattern Mining, Pearson's correlation tests, and correlation prediction.

In parallel, several analyses were conducted on the different spaces to uncover significant patterns specific to the success or failure of the task.

Finally, we conducted various experiments to identify which sequences were the most informative and decisive for success prediction. We found that the last three attempts of a student's sequence are critical in predicting their success. However, this makes it difficult to predict a student's failure early enough to provide timely and relevant feedback.

There is still future work to be done, and we are confident that with deeper analysis, we may discover other surprising patterns that could help us build a more effective success predictor.

Bibliography

Jiang, Bo and al. (2022). *Programming Trajectories Analytics in Block-Based Programming Language Learning*. URL: `https://eric.ed.gov/?id=EJ1326859`. (accessed: 03.06.2024).

Emerson, Andrew and al. (2019). *Predicting Early and Often: Predictive Student Modeling for Block-Based Programming Environments*. URL: `https://eric.ed.gov/?id=EJ1326859`. (accessed: 03.06.2024).

Joanna, Wolski (2023). *Unveiling Insights into Student Behavior using Abstract Syntax Trees and Machine Learning*.

Sager, Tobias and al. (2006). *Detecting similar Java classes using tree algorithms*. URL: `https://dl.acm.org/doi/10.1145/1137983.1138000`. (accessed: 03.06.2024).

Slimani, Thabet (2013). *Description and Evaluation of Semantic Similarity Measures Approaches*. URL: `https://arxiv.org/abs/1310.8059`. (accessed: 03.06.2024).

Kinnebrew, J. S. and al. (2013). *A contextualized, differential sequence mining method to derive students' learning behavior patterns*. URL: `https://files.eric.ed.gov/fulltext/EJ1115377.pdf`. (accessed: 03.06.2024).