A Machine Learning Approach to Detect Student Success in Pair Programming

Gabriela C. B. Bezerra
Telecommunications Engineering Department
Federal Institute of Ceará
Fortaleza, Brazil
gabriela.carvalho.barros00@aluno.ifce.edu.br

Felipe Pinto Marinho
Teleinformatics Engineering Department
Federal University of Ceará
Fortaleza, Brazil
fpmarinho@alu.ufc.br

Ajalmar R. da Rocha Neto
Computer Engineering Department
Federal Institute of Ceará
Fortaleza, Brazil
ajalmar@ifce.edu.br

***Abstract*—Learning software development remains a significant challenge for students, even when supported by active learning methodologies such as Challenge-Based Learning (CBL). In CBL, students often work in pairs or groups to develop applications that address real-world problems. However, interpersonal issues and varying collaboration styles can negatively affect performance. This study proposes a ML approach to predicting which student pairs are likely to struggle, based on social styles (using the Merrill-Reid model) and project complexity measurements. A dataset of 38 paired programming projects was collected from a real classroom setting. Supervised machine learning models — Support Vector Machine (SVM), k-Nearest Neighbors (KNN), and Decision Tree — were trained and evaluated using the Leave-One-Out methodology, with model tuning done via grid search and cross-validation. The best-performing model (KNN) achieved an accuracy of 78.94% in identifying pairs that would meet at least 70% of the project requirements. The results demonstrate the potential of data-efficient models to support timely instructional interventions in collaborative learning environments.**

***Index Terms*—education, software development, pair programming, challenge based learning, machine learning, data-efficient**

I. Introduction

Artificial intelligence (AI) aims to create intelligent systems through logical/mathematical models that describe or imitate human intelligence and can learn to perform tasks that humans perform. Recent advancements, particularly in Artificial Neural Networks, have significantly contributed to the field [1]. A subset of AI, machine learning (ML) focuses on developing systems that learn to recognize patterns from data to solve specific problems [2]. These models process data, adapt based on successes and errors, and improve over time, with three main types: supervised, unsupervised, and reinforcement learning. In supervised ML, datasets include attributes and an expected output, addressing two primary tasks: classification, where the output is discrete and categorical, and regression, where the output is continuous and numeric [2]. Common classification models like Support Vector Machines (SVM), k-Nearest Neighbors (KNN), and Decision Trees are widely used for tasks such as facial recognition, disease diagnosis, and sentiment analysis [3]. Machine learning also offers solutions to challenges in computing education [4], with related work discussed in section II. These models are discussed in more depth in section III, along with techniques for training and testing them with a data-efficient approach.

Teaching programming in higher education is still one of the greatest pedagogical challenges of our time [5]–[7]. Software development and programming logic are subjects rarely covered in schools and are usually only seen in professional or higher education courses such as computer science, computer engineering, and others. That said, it is common for many students to have difficulties when they try to program for the first time, even with the best teachers [8]. Typically, in a traditional programming course, the student enters with great curiosity, feeling engaged and optimistic, believing that he will be able to face all the technical challenges and be successful [5]. However, after a few classes and exercises, some of them tend to become increasingly frustrated with their own mistakes until they give up on the exercise they are doing, lose motivation, and, finally, drop out of the course itself [9]–[11].

The use of active teaching methodologies such as Challenge Based Learning (CBL) alleviates this problem [12], but other issues may happen. In CBL, students are encouraged to "learn how to learn" by working in groups, engaging in a real-world challenge, idealizing, prototyping, and implementing solutions as software applications. This process occurs in cycles, changing groups and challenges every couple of weeks. Every cycle, the students must engage in a new challenge, write questions about it, do some research or investigation, synthesize the answers, think of solutions (software) that help solve the challenge they have chosen, choose the best solution possible, and build it. The main role of the instructors is to support the students in both learning hard skills and soft skills during the cycles. The instructors are also responsible for designing each cycle by defining its duration and technical requirements to be reached by the students while they build their apps.

Although this format reduces student technical frustration, group work requires decision-making and collaboration, where potential conflicts between colleagues can arise, sometimes big enough to disrupt the progress of the project and be more detrimental than beneficial to the team's technical learning. It is necessary to develop tools that somehow prevent these big conflicts from occurring, or at least alert the team of

instructors when and where they are happening, thus enabling the conflict/friction to be resolved as soon as possible and avoiding damage to the class's learning.

The objective of this paper is: given a pair of students in a course that uses CBL as a teaching methodology, build a classifier that is capable of predicting the chance that this pair will meet at least 70% of the requirements requested by the instructors at the beginning of a project cycle based on information about the social styles of each member of the pair (Merrill-Reid social style) and the complexity of the project designed by the pair.

With this classifier, the team of instructors will have a tool to help identify as soon as possible which pairs will potentially need closer technical-pedagogical support throughout a given CBL cycle. To build the classifier, data were collected from 38 projects carried out by the students applying the pair programming technique.

More details about the problem addressed, the format of the dataset, and the way the data were collected are described in the IV section. The methodology used to develop the classifier and the computational simulations performed were described in section V. The results obtained, and final conclusions are in the following sections VI and VII.

II. Related Works

Machine learning models have been used to solve problems in several areas, including many relevant applications in the field of education. Educational data mining (EDM) is a recurring theme among many researchers, who have even produced works related to predicting students' academic performance using machine learning.

A recent systematic review analyzed works in this area that have been developed from 2009 to 2021 [13] and found that most research addresses the problem addressed with binary classification algorithms, mostly trying to predict student dropout. It was also concluded that there is little attention paid to detailing the attributes of the datasets built for research. Most of them focus on student demographic data, academic data, and records of use of *e-learning* systems, without considering the dynamic nature of learning, which is a process that involves, among other things, the interaction between students, the personal interests of each one, and the challenges faced by the class.

There are also several studies focused on predicting student performance in programming courses [14]–[16] but none of them are focused on classes in courses that use active teaching methodologies to teach programming.

There are also studies analyzing how much personality influences the collaboration of two programmers working using the pair programming technique. One of those was inconclusive and based its analysis on the five-factor model of personality [17]. Another one found greater compatibility between people with heterogeneous personality traits [18]. A third very interesting study found that students [19] who have similar learning styles (*learning styles*) and technical levels tend to make better use of pair programming activities during the course they are taking. In addition, a study attempted to predict the compatibility between two programmers through machine learning, predicting the success of the pair's collaboration using the participants' eye-tracking records as the main data [20].

A systematic review of 74 papers on the subject [21] concluded that pair programming is an effective practice for improving overall class performance and that the most impactful factor for the success of students who study using pair programming is the previous experience of at least one of the members of the pair. The review also identified the need for studies that focused on analyzing the use of pair programming in conjunction with modeling and design activities. Finally, the review highlighted the need for more studies aimed at developing pedagogical tools that assist in the effective use of pair programming in the classroom.

There are no studies that aim to build a classifier capable of predicting the success of a pair of students practicing pair programming within the context of a course that uses active learning methodology, using as part of the database their social styles (based on the Merrill-Reid model), and the time and complexity of the projects to be carried out.

III. Theoretical Framework

As previously mentioned, machine learning is a sub-area studied within artificial intelligence, which seeks to develop computational techniques for learning and build systems capable of learning automatically [2], generally with the aim of finding solutions to specific problems. To learn to solve problems, a computational model must first process a set of data with information related to the problem in question. During its learning phase, this model acquires "experience" with each new samples of data to which it is exposed.

A dataset is formed by a set of samples of information about a given problem or context. For example, a dataset in an educational context may have a set of examples of students in a class, including specific information about each student such as age, number of absences, tasks completed, grades, among others. In machine learning, the specific information of the samples in a dataset are called attributes.

A. Supervised Learning

A dataset may include an output value for each sample, which may represent important information for the context analyzed. This output value can often be used to group different data examples into a single class or label, differentiating them from the others. In the example of the educational base mentioned above, there could be a new attribute called *failed*, in which the possible values would be *Yes* or *No*, that is, these would be the classes or labels of this dataset. The example covered is described in the Table I.

A supervised learning model can extract hidden patterns from labeled data through a training process based on historical data, allowing it to make accurate predictions at the inference stage.

TABLE I
EXAMPLE OF A DATASET FOR SUPERVISED LEARNING

ID	Age	Absences	Grades	Assignments	Failed	
0	13	2	[7.0, 9.6, 6.0]	5	No	
1	14	7	[3.0, 6.5, 6.9]	2	Yes	
...	...	...	...	...	...	...

TABLE II
PREDICTION OF A NEW DATA POINT

ID	Age	Absences	Grades	Assignments	Failed
108	13	4	[6.0, 7.5]	5	?

The initial processing phase of the dataset is called training. During this phase, the computational model processes the attributes of each example in the dataset and adjusts its internal parameters, trying to construct heuristics, that justify the class/label defined for that pattern. After training, the trained model can receive an entirely new pattern as input, process its attributes, and return the prediction of which class it belongs to. A new pattern is described as an example in the Table II.

Some examples of consolidated and widely used supervised learning models are SVM, KNN, Decision Tree, Logistic Regression, Naive Bayes, ANN, and Random Forest [22].

B. Classification x Regression

The attributes of a dataset can generally be of two types:

1) **Continuous**: numerical values, with many variations that do not frequently repeat and can be arranged in a linearly.
2) **Discrete**: categorical values, which repeat for several examples in the same dataset and cannot be ordered in a linearly.

In a dataset, when the output value that is intended to be predicted (class/label) is of a discrete type, the problem at hand falls into the classification type. If it is continuous, the problem addressed is referred to as regression [2]. Additionally, a classification problem may be binary when it classifies new data into one of two distinct classes or multi-class when it classifies new data into one of more than two distinct classes.

In classification, to measure the success of a model, metrics are derived from a confusion matrix, which outlines the number of correct and incorrect predictions for each class predicted by the classifier. The most commonly used metric to measure the effectiveness of a model is accuracy [23], which is calculated by dividing the number of correct predictions by the total number of tests conducted. However, other metrics such as precision and recall are also utilized.

In regression, the most commonly used metrics to measure how effective a model is are Mean Square Error (MSE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) [24].

C. SVM

Support Vector Machine (SVM) is a type of supervised learning algorithm that can be used for both classification and regression. It operates by finding a hyperplane in a high-dimensional space that maximally separates patterns of different classes [25].

For example, in a two-dimensional space, the hyperplane would be a line that divides the space into two regions, one for each class. To find the ideal hyperplane that separates the regions between two classes in an n-dimensional space, the SVM employs a technique known as the kernel trick, which allows for the transformation of patterns to a higher dimension in the space, where it is easier to find a hyperplane that separates them. Subsequently, this hyperplane is transformed back to the original dimension of the dataset, making it possible to classify new patterns.

There are various types of kernel functions: RBF, polynomial, sigmoid, among others. Particularly in SVM, the choice of kernel has a significant impact on the effectiveness of the desired classification/regression. It is essential to correctly select the type of kernel function to be used for each dataset, and it is possible to refine and modify it so that it fits the problem at hand [3]. The radial basis function (RBF) is one of the most commonly used functions to build classifiers even when the dataset is small [26] and there is no clear relationship between the attributes (input values) and the output value; therefore it was the chosen kernel for this research.

D. KNN

K-nearest neighbors (KNN) is a machine learning algorithm that can be used for both classification and regression [27]. When applied to classification, KNN stores all available training data and classifies new patterns based on a measure of similarity.

To classify a new pattern, the KNN algorithm first calculates the similarity between this new pattern and all the patterns in the training dataset. The similarity measure is typically a distance metric, such as Euclidean distance, Manhattan distance, or Minkowski distance. Once the similarities are calculated, the KNN algorithm selects the K patterns in the training set that are most similar to the pattern being analyzed and then assigns the most common label among these K patterns to that pattern. This is known as majority voting.

When used for regression, the KNN algorithm simply takes the average of the K patterns in the training set that are closest to the new pattern to make a prediction.

E. Decision Tree

The decision tree is a model used for both classification and regression. It operates by creating a decision model that resembles a tree structure, where each internal node represents a 'test' on an attribute (for example, whether a coin flip results in heads or tails), each branch represents the outcome of the test, and each leaf node represents a class. The algorithm then uses the tree to make predictions based on unseen data, traversing from the root node to a leaf node, utilizing the attribute values of the processed data to determine which branch to follow at each internal node [28].

In classification, a decision tree begins by dividing the input space into regions (leaves) where each leaf represents a class or a predicted value. The decision tree model learns by dividing the training data into subsets based on the values of the input attributes and then recursively applying the same process to each subset until each leaf of the tree contains only patterns of a single class or predicted value.

To make a prediction, the decision tree starts at the root node, which represents the entire input space. It then follows a series of decisions, moving from one node to another, until it reaches a leaf node. The class or predicted value associated with the leaf node is then returned as the prediction for the new pattern.

An advantage of decision trees is that they can handle continuous, categorical data. In addition to being able to deal with attributes on different scales, there is no need to perform dataset normalization in the preprocessing stage. They are also easy to interpret and visualize, which makes them a popular choice for many applications. However, they can be sensitive to changes in the training data and can overfit the data if the tree becomes too deep. To avoid overfitting, the tree can be ”pruned” by removing unnecessary nodes or limiting the maximum depth of the tree [29]. Resampling methods, such as bootstrap aggregation (BAGGING) [30] can also be useful to reduce the variability of the decision tree model, helping to avoid the occurrence of overfitting.

F. Overfitting

Overfitting occurs when a model performs well on training data but fails to maintain good performance on new data. This can happen because the model has learned patterns in the training data that do not generalize to a broader population. In other words, the model has learned to make predictions based on random noise or outliers in the training data, rather than the underlying relationship between the input and output variables [31].

This problem can occur when the model has few attributes or when the training data is noisy and contains a lot of irrelevant information. Overfitting can also occur when a model is trained for too long, causing it to 'memorize' the training data instead of learning to generalize outputs for new data, or when the dataset used for training is small and unbalanced [32].

G. Leave One Out

The Leave One Out technique is one of the strategies for separating the dataset into a training set and a testing set. In this approach, a single pattern is used as the testing set while the remaining patterns are utilized as the training set. This process is repeated for each pattern, ensuring that each one is used as the testing set exactly once. Each repetition is referred to as a realization. An example can be visualized in the Fig. 2. The model's performance is then computed by calculating the average performance across all test sets from each realization.

This technique is useful when the dataset is small, making it important to utilize as much data as possible for training. Likewise, it is beneficial when the patterns are not independent, such as when they come from a time series, because it ensures that each pattern is tested on a different set. However, it may be computationally expensive, as it requires training the model once for each row in the dataset. For large datasets, it may be more efficient to use a different method.

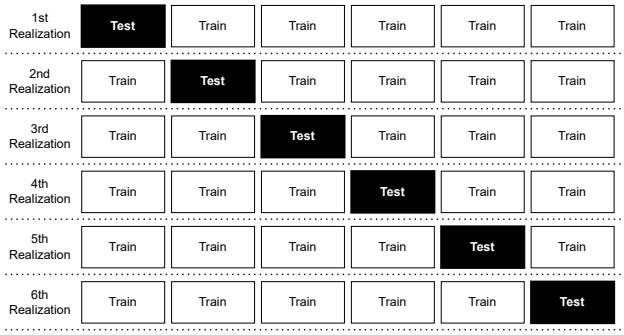

Fig. 1. Applying Leave One Out on a sample dataset with 6 rows.

H. Cross Validation

Cross-validation is a statistical method used to evaluate the performance of machine learning models. It is commonly employed in the field of data mining, being a resampling procedure used to assess a model by dividing the data into a training set and a test set.

First, the dataset must be divided into k fixed-size subsets, referred to as *folds*. With all the *folds* divided, k-1 of them are used for training, with the remaining one used as the test set. This process is repeated k times, with the model being trained on the training set, used to make predictions on the test set, and then having its performance evaluated. After all validations, an average performance score is calculated.

To split *folds*, you can implement specific rules to choose which types of data are selected in each of them. After splitting, if the ratio between the number of classes in the dataset has been preserved in each *fold*, it is said that the cross-validation was done in a stratified manner [33]. Otherwise, it's considered unstratified.

Cross-validation is useful because it helps to avoid overfitting, which, as mentioned earlier, occurs when a model is excessively complex and performs well on training data but does not maintain good performance on new data.

I. Grid Search

Grid search is a technique used to tune the hyperparameters of a machine learning model. Hyperparameters are values that are not learned from the data and are defined before the model training. Examples of hyperparameters include the learning rate, the kernel function, and the regularization coefficient.

In grid search, a grid of possible values for the hyperparameters is established, and the model is then trained and evaluated using each combination of hyperparameter values. The model's performance is subsequently compared across the different combinations, and the hyperparameter values that

yield the best performance are selected. Grid search is an efficient method for finding the ideal hyperparameters for a machine learning model and is often used in conjunction with cross-validation.

IV. Problem

One of the main hypotheses to explain students' frustration in programming courses is that, generally, such students are more concerned with passing tests than with learning how to solve problems and build applications using programming logic [9]. Thus, learning methodologies that are not centered on tests and assessments, but that encourage students to learn how to program by engaging in a challenge or problem that interests them and then trying to build their own applications from scratch (*e.g.*, Challenge Based Learning (CBL), Problem Based Learning (PBL) and others) can be a way to address this problem, changing the focus of assessments to the acquisition of the knowledge necessary to solve a real-world challenge/problem.

CBL in particular helps students feel more motivated by making them learn technical subjects while creating a solution to a real-world problem [12]. In contrast to more traditional teaching, which consists of classes where students are invited to learn passively, it is possible to teach programming in an active way, promoting the teaching of theoretical concepts aligned with practice and the constant sharing of knowledge gained with other students. But even though it is very beneficial, there is room for improvement.

On one hand, the active approach may work better than the traditional one for teaching programming, making the students feel more engaged and constantly practice what is studied. On the other hand, there are aspects of this teaching format that require caution and attention. One of these points is that the greater autonomy and independence of students to break down and solve open-ended problems can make them feel more stressed, which may lead to conflicts within development teams. To mitigate this problem, it is essential to consider the students individually and try to place them in teams where all members support each other during the most challenging moments of the practical project they will undertake. To this end, within the learning methodology addressed in this research, before actively beginning to form groups and develop projects, the students participated in a workshop with an expert and were invited to reflect on themselves, discovering which type of social style they align with most, based on the classic model of Merrill-Reid [34].

The four Merrill-Reid social styles are:

1) **Amiable**: generally more social and conciliatory in a group of people. They value relationships and individuals.
2) **Analytical**: they tend to be the quietest and most timid in the group, focused on completing the tasks assigned to them. They value thoughtfulness and introspection.
3) **Driver**: They tend to be the leaders of the group, often being the ones who make decisions and demand results. They value reaching deadlines and competition.
4) **Expressive**: They are typically quite social and spontaneous and need to feel recognized in what they do. They value recognition and approval.

After reflecting, students identified their primary and secondary social styles and assessed their flexibility in adapting to other styles. They were then encouraged to form diverse teams, promoting interaction with peers of different styles.

After two cycles conducted by the students in pairs, totaling 38 implemented projects through paired programming, a tabular dataset was created with one row for each project carried out by a pair, including columns representing:

1) **Social Styles**: primary, secondary, and flexibility rate of the social styles for each member of the pair. The value of the social style columns ranges from 0 (Amiable), 1 (Analytical), 2 (Driver), and 3 (Expressive), while the flexibility rate goes from 1 to 5 (little to very flexible). Collected through a form completed by the students.
2) **Project features**: the duration of the cycle, the amount of easy, medium, and hard functionalities envisioned by the pair as part of their project. All these values are integers, and were provided by an instructor of the course.
3) **Requirements fulfillment**: the target feature, filled with the number 0, representing failure if the pair has met less than 70% of the requirements by the end of the cycle, or with the number 1, representing success if they have met 70% or more. This data was provided by an instructor.

The amount of patterns for each of the possible classes (0 for failure and 1 for success) are described in the Table III.

TABLE III
Amount of patterns for each class

Class	Percentage	Pattern
failure	55.3%	21
success	44.7%	17

The criteria used to define the number of easy, medium, and hard functionalities for each project, as well as the list of requirements, were established for each cycle individually prior to the construction of the dataset. Thus, an objective assessment of each project implemented by each pair was sought, without any changes in the criteria during the evaluation. The complete dataset can be found on the following gist webpage https://gist.github.com/GabrielaBezerra/87f07ec50d283869eed6c82bd34307df.

V. Experiments

With the dataset in hand, computational experiments were carried out with the objective of building multiple classifiers with different algorithms capable of successfully predicting the value of the column *fulfilled_requirements* (target value) using all other columns as input (features).

Due to the small size of the dataset (38 instances), we adopted a Leave-One-Out (LOO) methodology as the main experimental approach for trying to reach model generalization. In this approach, each of the 38 instances was held out

once as a test sample, while the remaining 37 were used for training. This process was repeated 38 times, ensuring that each data point was used exactly once for testing and 37 times for training. The final performance of each model was computed as the average of all 38 individual predictions.

Using the scikit-learn library in Python [35], a code was written to execute the flow explained in the Fig. 2 for the construction and performance evaluation of multiple classifiers (SVM, KNN, Decision Tree) with the following steps:

1) **Leave One Out**: A part of the dataset was selected for training (37 lines) and another for testing (1 line) using the Leave One Out technique. This was done interactively 38 times, once for each pattern in the dataset.
2) **Grid Search with Cross Validation**: Using the 37 patterns from the training set selected in step 1, experiments were conducted with 6 different grid search strategies using stratified and unstratified cross-validation, both with 6, 10, and 12 folds. After each execution, a different classifier model was generated, configured with the best hyperparameters found in this stage of the process.
3) **Train**: Each classifier generated in the previous step was then trained with the training set (37 lines) initially separated in the Leave One Out interaction.
4) **Test**: Each classifier trained in the previous step was used to predict the test set (1 line), which had never been processed before. The result was processed by the function *classification_report* from the scikit-learn library and stored for future evaluation. While there are still Leave One Out realizations left, this step leads again to the first step.
5) **Metrics**: Finally, after all the realizations, the arithmetic mean of the accuracy rate, the standard deviation (std), and the confusion matrix of all the stored results are calculated for each classifier and printed to the console.

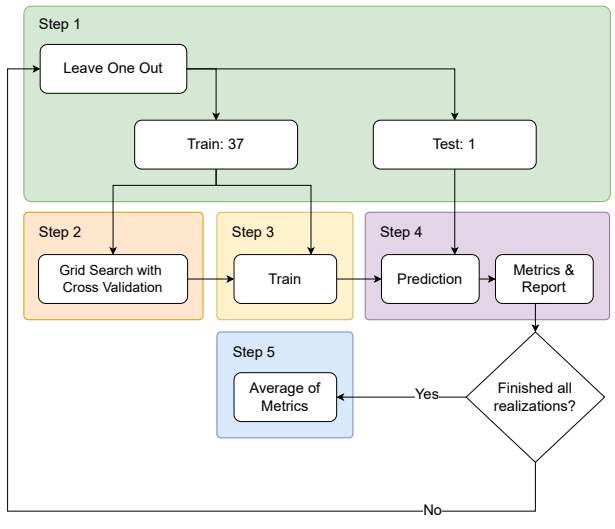

Fig. 2. Flowchart of the experiment steps

A. Justification of the Selected Models

It is not possible to assert that there is a single classification model that will always be superior for small datasets, as the ideal model must always be selected on a case-by-case basis and will depend on the specific characteristics of its attributes, their correlations, and the classification task at hand. However, some models are used more frequently than others for classifying small datasets. Among these are KNN, SVM, and decision trees. These models can be generally effective for small datasets because they can handle high-dimensional data and can be regularized to prevent overfitting, one of the most common issues that arise during the modeling of small datasets.

B. SVM

Using SVM, a grid search was conducted with a fixed RBF (Radial Basis Function) kernel, combining hyperparameter values within the ranges described in the Table IV.

TABLE IV
SVM: GRID SEARCH HYPERPARAMETERS

Hyperparam	Values
C	10^0 to 10^{15}
γ	10^{-10} to 10^{-5}

The results obtained during the experiments conducted with different techniques applied in cross-validation are detailed in Table V.

TABLE V
SVM: METRICS OBTAINED FROM GRID SEARCH WITH CROSS-VALIDATION METHODS.

Cross Validation	Folds	Hyperparam	Accuracy	Std
Unstratified	6	C: 10^2 to 10^7 γ: 10^{-9} to 10^{-5}	**73.68%**	0.440
Unstratified	10	C: 10^2 to 10^7 γ: 10^{-9} to 10^{-5}	**73.68%**	0.440
Unstratified	12	C: 10^2 to 10^9 γ: 10^{-9} to 10^{-5}	**73.68%**	0.440
Stratified	6	C: 10^2 to 10^{13} γ: 10^{-9} to 10^{-5}	60.52%	0.488
Stratified	10	C: 10^2 to 10^8 γ: 10^{-8} to 10^{-5}	**73.68%**	0.440
Stratified	12	C: 10^2 to 10^7 γ: 10^{-8} to 10^{-5}	71.05%	0.453

C. KNN

In the KNN model, grid search was performed by combining the hyperparameter values within the ranges described in Table VI, using distance as weight.

TABLE VI
KNN: GRID SEARCH HYPERPARAMETERS.

Hyperparam	Values
K-neighbors	1 to 13

The results obtained during the experiments carried out with different techniques applied in the cross-validation are

described in Table VII, using the Euclidean metric [36] to calculate the distance between the patterns.

TABLE VII
KNN: METRICS OBTAINED FROM GRID SEARCH WITH CROSS-VALIDATION METHODS, EMPLOYING THE EUCLIDEAN DISTANCE.

Cross Validation	Folds	Hyperparam	Accuracy	Std
Unstratified	6	K-neighbors: 3 to 4	73.68%	0.440
Unstratified	10	K-neighbors: 3 to 5	76.31%	0.425
Unstratified	12	K-neighbors: 3 to 7	73.68%	0.440
Stratified	6	K-neighbors: 3 to 7	71.05%	0.453
Stratified	10	K-neighbors: 3 to 6	**78.94%**	0.407
Stratified	12	K-neighbors: 3 to 7	73.68%	0.440

The results obtained during the experiments made with different techniques applied in the cross-validation are described in the Table VIII, using the Minkowski [36] metric to calculate the distance between the patterns.

TABLE VIII
KNN: METRICS OBTAINED FROM GRID SEARCH WITH CROSS-VALIDATION METHODS, EMPLOYING THE MINKOWSKI DISTANCE.

Cross Validation	Folds	Hyperparam	Accuracy	Std
Unstratified	6	K-neighbors: 3 to 4	73.68%	0.440
Unstratified	10	K-neighbors: 3 to 7	65.38%	0.425
Unstratified	12	K-neighbors: 3 to 7	76.31%	0.425
Stratified	6	K-neighbors: 3 to 7	71.05%	0.453
Stratified	10	K-neighbors: 3 to 6	**78.94%**	0.407
Stratified	12	K-neighbors: 3 to 7	73.68%	0.440

D. Decision Tree

With decision tree, the grid search method was conducted by combining the values of hyperparameters described in the Table IX.

TABLE IX
DECISION TREE: GRID SEARCH HYPERPARAMETERS

Hyperparam	Values
Max Leaf Nodes	50 to 100
Min Samples Split	2, 3, 4

The results obtained during the experiments conducted with different techniques applied in cross-validation are described in the Table X.

VI. RESULTS

After completing all the experiments for each algorithm, we computed the overall accuracy, standard deviation, and confusion matrix for the best model configurations identified via grid search. Table XI summarizes the best results for each model across all experiments. The highest performance was achieved by the KNN classifier, which reached an average accuracy of 78.94% using stratified 10-fold cross-validation within each training fold to optimize hyperparameters.

Fig. 3 present confusion matrices and Fig. 4 shows hit/miss comparisons for each model, illustrating their classification behavior. KNN showed the highest accuracy for predicting failure, which suggests that its instance-based learning was able to model small similarities in student social profiles that correlate with this outcome. Decision Tree performed well, with highest accuracy for predicting successes. SVM performed worse, with over 20% errors in both classes, suggesting difficulty in generalizing with a clear separating hyperplane.

Despite these results, all models exhibited a modest standard deviation across realizations, reflecting some variability that is expected given the small and imbalanced dataset. Further investigation is needed to evaluate these models, and to explore other evaluation metrics (e.g., F1-score, ROC AUC) that will bring additional insight.

TABLE X
DECISION TREE: METRICS OBTAINED FROM GRID SEARCH WITH CROSS-VALIDATION METHODS.

Cross Validation	Folds	Hyperparam	Accuracy	Std
Unstratified	6	Max Leaf: 50 to 92 Min Samples: 2 to 4	71.05%	0.453
Unstratified	10	Max Leaf: 50 to 98 Min Samples: 2 to 4	**76.31%**	0.425
Unstratified	12	Max Leaf: 50 to 99 Min Samples: 2 to 4	73.68%	0.440
Stratified	6	Max Leaf: 50 to 92 Min Samples: 2 to 4	**76.31%**	0.425
Stratified	10	Max Leaf: 50 to 98 Min Samples: 2 to 4	68.42%	0.464
Stratified	12	Max Leaf: 50 to 83 Min Samples: 2 to 4	68.42%	0.464

TABLE XI
COMPARISON BETWEEN THE MODELS WITH THE BEST RESULTS

Model	Hyperparameters	Global Accuracy	Std
KNN	K-neighbors: 3 to 6	**78.94%**	0.407
Decision Tree	Max Leaf: 50 to 98 Min Samples: 2 to 4	76.31%	0.425
SVM	C: 10^2 to 10^9 γ: 10^{-9} to 10^{-5}	73.68%	0.440

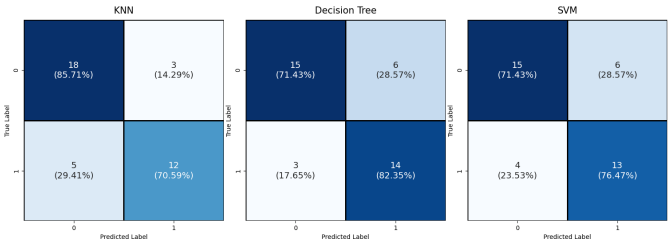

Fig. 3. Confusion matrices for each model

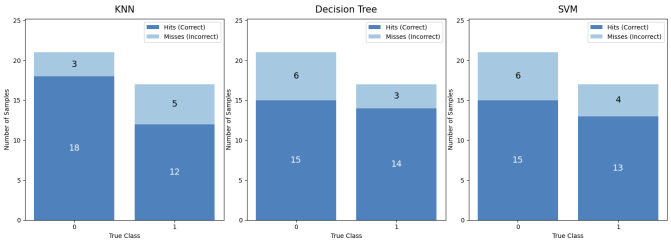

Fig. 4. Stacked bar plots illustrating hits and misses for each model

VII. Conclusion

This work presented a data-efficient machine learning approach to predict if student pairs in a Challenge-Based Learning (CBL) course would meet at least 70% of the project requirements. A small dataset was used, combining info about students' social styles and project complexity, and three classification models (KNN, Decision Tree and SVM) were tested using Leave-One-Out as evaluation method.

The KNN model reached the best accuracy (78.94%), slightly higher than the other models. These results suggest that simple models, trained with the right features, can help instructors to detect struggling teams and offer early support. This approach can maximize group learning efficiency and can be a critical tool in learning environments where team work is highly required.

Future works would benefit from expanding the database with more samples to further improve the effectiveness of the analyzed models. Eventually, it would also be useful to make an exploratory analysis of the dataset to understand which attributes have greater impact on the classes and experiment with other algorithms, supervised or not, to study this problem further.

References

[1] H. L. van der Maas, L. Snoek, and C. E. Stevenson, "How much intelligence is there in artificial intelligence? a 2020 update," *Intelligence*, vol. 87, p. 101548, 2021.

[2] M. C. Monard and J. A. Baranauskas, "Conceitos sobre aprendizado de máquina," *Sistemas inteligentes-Fundamentos e aplicações*, vol. 1, no. 1, p. 32, 2003.

[3] D. M. Abdullah and A. M. Abdulazeez, "Machine learning applications based on svm classification a review," *Qubahan Academic Journal*, vol. 1, no. 2, pp. 81–90, 2021.

[4] J. W. P. Luz, M. J. H. Rehfeldt, and M. C. Schorr, "Revisão sistemática da literatura sobre o uso de learning analytics no ensino de programação," *RENOTE*, vol. 19, no. 2, pp. 203–212, 2021.

[5] R. P. Medeiros, G. L. Ramalho, and T. P. Falcão, "A systematic literature review on teaching and learning introductory programming in higher education," *IEEE Transactions on Education*, vol. 62, no. 2, pp. 77–90, 2018.

[6] C. S. Cheah, "Factors contributing to the difficulties in teaching and learning of computer programming: A literature review," *Contemporary Educational Technology*, vol. 12, no. 2, p. ep272, 2020.

[7] S.-C. Kong, M. Lai, and D. Sun, "Teacher development in computational thinking: Design and learning outcomes of programming concepts, practices and pedagogy," *Computers & Education*, vol. 151, p. 103872, 2020.

[8] J. H. Berssanette and A. C. de Francisco, "Active learning in the context of the teaching/learning of computer programming: A systematic review," *Journal of Information Technology Education. Research*, vol. 20, p. 201, 2021.

[9] S. Popat and L. Starkey, "Learning to code or coding to learn? a systematic review," *Computers & Education*, vol. 128, pp. 365–376, 2019.

[10] J. Bennedsen and M. Caspersen, "Failure rates in introductory programming," *SIGCSE Bulletin*, vol. 39, pp. 32–36, 06 2007.

[11] Y. Bosse and M. A. Gerosa, "Why is programming so difficult to learn?: Patterns of difficulties related to programming learning mid-stage," *ACM SIGSOFT Software Engineering Notes*, vol. 41, pp. 1–6, 01 2017.

[12] F. Binder, M. Nichols, S. Reinehr, and A. Malucelli, "Challenge based learning applied to mobile software development teaching," 11 2017, pp. 57–64.

[13] B. Albreiki, N. Zaki, and H. Alashwal, "A systematic literature review of student' performance prediction using machine learning techniques," *Education Sciences*, vol. 11, no. 9, 2021. [Online]. Available: https://www.mdpi.com/2227-7102/11/9/552

[14] T. T. Mai, M. Bezbradica, and M. Crane, "Learning behaviours data in programming education: Community analysis and outcome prediction with cleaned data," *Future Generation Computer Systems*, vol. 127, pp. 42–55, 2022.

[15] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer science education*, vol. 13, no. 2, pp. 137–172, 2003.

[16] K. Quille and S. Bergin, "Programming: predicting student success early in cs1. a re-validation and replication study," in *Proceedings of the 23rd annual ACM conference on innovation and technology in computer science education*, 2018, pp. 15–20.

[17] J. E. Hannay, E. Arisholm, H. Engvik, and D. I. Sjoberg, "Effects of personality on pair programming," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 61–80, 2009.

[18] P. Sfetsos, I. Stamelos, L. Angelis, and I. Deligiannis, "An experimental investigation of personality types impact on pair effectiveness in pair programming," *Empirical Software Engineering*, vol. 14, no. 2, pp. 187–226, 2009.

[19] L. Williams, L. Layman, J. Osborne, and N. Katira, "Examining the compatibility of student pair programmers," in *AGILE 2006 (AGILE'06)*. IEEE, 2006, pp. 10–pp.

[20] M. Villamor and M. M. Rodrigo, "Predicting successful collaboration in a pair programming eye tracking experiment," in *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization*, 2018, pp. 263–268.

[21] N. Salleh, E. Mendes, and J. Grundy, "Empirical studies of pair programming for cs/se teaching in higher education: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 37, no. 4, pp. 509–525, 2011.

[22] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 161–168.

[23] I. Muhammad and Z. Yan, "Supervised machine learning approaches: A survey." *ICTACT Journal on Soft Computing*, vol. 5, no. 3, 2015.

[24] A. Botchkarev, "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology," *arXiv preprint arXiv:1809.03006*, 2018.

[25] M. I. Hossain, "Support vector machine," 2022.

[26] R. Andonie, "Extreme data mining: Inference from small datasets," *International Journal of Computers Communications & Control*, vol. 5, no. 3, pp. 280–291, 2010.

[27] D. Muhajir, M. Akbar, A. Bagaskara, and R. Vinarti, "Improving classification algorithm on education dataset using hyperparameter tuning," *Procedia Computer Science*, vol. 197, pp. 538–544, 2022.

[28] M. Bansal, A. Goyal, and A. Choudhary, "A comparative analysis of k-nearest neighbour, genetic, support vector machine, decision tree, and long short term memory algorithms in machine learning," *Decision Analytics Journal*, p. 100071, 2022.

[29] M. Al-Akhras, K. El Hindi, M. Habib, B. A. Shawar *et al.*, "Instance reduction for avoiding overfitting in decision trees," *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 438–459, 2021.

[30] G. James, D. Witten, T. Hastie, R. Tibshirani *et al.*, *An introduction to statistical learning*. Springer, 2013, vol. 112, no. 1.

[31] X. Ying, "An overview of overfitting and its solutions," in *Journal of physics: Conference series*, vol. 1168. IOP Publishing, 2019, p. 022022.

[32] J. W. Rocks and P. Mehta, "Memorizing without overfitting: Bias, variance, and interpolation in overparameterized models," *Physical Review Research*, vol. 4, no. 1, p. 013201, 2022.

[33] J. Wieczorek, C. Guerin, and T. McMahon, "K-fold cross-validation for complex sample surveys," *Stat*, p. e454, 2022.

[34] R. H. Merrill, David W. Reid, *Personal Styles & Effective Performance*. CRC Press, 1981.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[36] K. Chomboon, P. Chujai, P. Teerarassamee, K. Kerdprasop, and N. Kerdprasop, "An empirical study of distance metrics for k-nearest neighbor algorithm," in *Proceedings of the 3rd international conference on industrial application engineering*, 2015, pp. 280–285.