FEDERAL UNIVERSITY OF AMAZONAS - UFAM

INSTITUTE OF COMPUTING - ICOMP

GRADUATE AND POSTDOCTORAL STUDIES

Towards AI-human hybrid online judges to support decision making for CS1 instructors and students

Filipe Dwan Pereira

Manaus - AM

July 2022

Filipe Dwan Pereira

Towards AI-human hybrid online judges to support decision making for CS1 instructors and students

Doctoral dissertation submitted to the Institute of Computing – IComp-UFAM, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science.

Supervisor

Elaine Harada Teixeira de Oliveira, Dr.

Co-supervisors

David Braga Fernandes de Oliveira, Dr., Alexandra Iona Cristea, Dr.

Federal University of Amazonas - UFAM

Institute of Computing - IComp

Manaus - AM

July 2022

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Pereira, Filipe Dwan
P436t Towards AI-human hybrid online judges to support decision making for CS1 instructors and students / Filipe Dwan Pereira . 2022
265 f.: 31 cm.

Orientadora: Elaine Harada Teixeira de Oliveira
Coorientador: David Braga Fernandes de Oliveira
Coorientadora: Alexandra Ioana Cristea
Tese (Doutorado em Informática) - Universidade Federal do Amazonas.

1. Hybrid intelligence. 2. Programming online judges. 3. Introductory programming. 4. Machine learning. 5. Prescriptive analytics. I. Oliveira, Elaine Harada Teixeira de. II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Towards AI-human hybrid online judge to support decision making for CS1 instructors and students"

FILIPE DWAN PEREIRA

Tese de Doutorado defendida e aprovada pela banca examinadora constituída pelos Professores:

Profa. Elaine Harada Teixeira de Oliveira - PRESIDENTE

Prof. Marco Antônio Pinheiro de Cristo - MEMBRO INTERNO

gov.br Documento assinado digitalmente
DIEGO DERMEVAL MEDEIROS DA CUNHA MATOS
Data: 08/07/2022 09:15:17-0300
Verifique em https://verificador.iti.br

Prof. Diego Dermeval Medeiros da Cunha Matos - MEMBRO EXTERNO

Prof. Rafael Dias Araújo - MEMBRO EXTERNO

Prof. Rafael Ferreira Leite de Mello - MEMBRO EXTERNO

Manaus, 07 de Julho de 2022

Doctoral dissertation entitled *Towards AI-human hybrid online judge to support decision making for CS1 instructors and students* presented by Filipe Dwan Pereira and submitted to the evaluation of the following members:

Dr. Elaine Harada Teixeira de Oliveira
Supervisor
Institute of Computing
Federal University of Amazonas

Dr. Rafael Dias Araújo
Faculty of Computing
Federal University of Uberlândia

Dr. Diego Dermeval Medeiros da Cunha Matos
Institute of Computing
Federal University of Alagoas

Dr. Rafael Ferreira Leite de Mello
Department of Computer Science
Rural Federal University of Pernambuco

Dr. Marco Antônio Pinheiro de Cristo
Institute of Computing
Federal University of Amazonas

Manaus - AM, 7th July 2022

To my daughters Beatriz and Isabela, my wife Anne,
my mother Regina and my grandma Rosa

Acknowledgements

First, I would like to thank God, because all the good things that I have come from God. Indeed, Your endless wonders shine everywhere. My God, You are good for us. I see your glory, love, grace, and mercy in our lives.

I'm very thankful to my mom, Francisca Regina da Silva Pereira, for always supporting me and giving me precious advice. I love you mom. You are my hero. Thank you very much, my beloved wife, Anne Ely Peres Pereira, for giving me the best conditions to study day after day. You are such an incredible person! I'm really luck for having you as my wife. I'm very thankful to my two beautiful daughters who make my life much happier. Dad loves you, my girls. I'm thankful to my gramma Rosa, because what she did for me in my childhood has an impact on who I am today. I love you all, you are the women of my life.

My thanks to my little brother André. Your companionship was crucial in this process.

I would like to thank to all my close friends Adenilson, Anderson, Berg, Elano, and Rodrigo for your companionship and friendship of always. You are awesome, guys.

I would like to thank to the best supervisors: professor Elaine H. T. Oliveira, professor David B. F. Oliveira, and professor Alexandra I. Cristea. I have learned a lot from you. I will be forever grateful for everything you've done on this challenging journey!

I want to thank to all researchers that collaborate with me in this study: Armando Toda, Craig Stewart, Henrik Bellhäuser, Hermino Freitas, Leandro Galvão, Lei Shi, Luiz Rodrigues, Marcelo Henklain, Marcos Lima, Samuel Fonseca, Seiji Isotani. I've learned

a lot from you. This work would not be the same without your support and advice.

My thanks also to the friends I made during my sandwich doctorate internship at Durham University: Ahmed Alamri, Yona Falinie, Neelanjan Bhowmik, Mohammad Alshehri, Tahani Aljohani, Jialin Yu, Khulood Alharbi, Laila Alrajhi, Zhongtian Sun, Anoushka Harit, Jack Barker, Brian Kostadinov, Abril Corona.

I would also like to thank all the IComp professors who supported me to conduct this research by sharing their immeasurable knowledge.

Finally, this research, carried out within the scope of the Samsung-UFAM Project for Education and Research (SUPER), according to Article 48 of Decree nº 6.008/2006 (SUFRAMA), was partially funded by Samsung Electronics of Amazonia Ltda., under the terms of Federal Law nº 8.387/1991, through agreements 001/2020 and 003/2019, signed with Federal University of Amazonas and FAEPI, Brazil. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Therefore, as God's chosen people, holy and dearly loved, clothe yourselves with compassion, kindness, humility, gentleness and patience. Bear with each other and forgive one another if any of you has a grievance against someone. Forgive as the Lord forgave you. ***And over all these virtues put on love****, which binds them all together in perfect unity.*

Colossians 3:12-14

Towards AI-human hybrid online judges to support decision making for CS1 instructors and students

Author: Filipe Dwan Pereira

Supervisor: Elaine Harada Teixeira de Oliveira, Dr.

Co-supervisors: David Braga Fernandes de Oliveira, Dr., Alexandra Iona Cristea, Dr.

Abstract

Introductory programming (also known as CS1 - Computer Science 1) may be complex for many students. Moreover, there are a high failure in these courses. A common agreement from computing education research is that programming students need practice and quick feedback on the correctness of their code. Nonetheless, CS1 classes are usually large with high heterogeneity of students which make individual/group personalised support almost impractical. As an alternative to improve and optimise the learning process, researchers indicate a system that automatically evaluates students' codes, also called online judge. These systems provide assignments created by instructors and an integrated development environment, where the student can develop and submit the solutions to problems and receive immediate feedback about the code correctness. Additionally, these online judge systems have opened up new research opportunities since it is possible to embed software components capable of monitoring and recording fine-grained actions performed by students during their attempts to solve the programming assignments. Research in the areas of Intelligent Tutoring Systems, Adaptive Educational Hypermedia and AI in Education have shown that personalisation using data-driven analysis is essential to improve the teaching and learning process and can be useful to provide individualised/group support for stakeholders (instructors and students). In this sense, in this work we collected students' interaction logs within an

online judge, recording very fine-grained data, such as keystroke, number of commands typed, number of submissions, etc., making it possible to do research of great precision into the exact triggers for students' progress. From these logs, we extract students' programming behaviours to compose what we call programming profiles. Furthermore, we extract useful information from the program statements using Natural Language Processing (NLP). Using such programming profiles and NLP extracted information, we propose and validate descriptive, predictive, and prescriptive AI methods that combine the large-scale approach formula for generalities with the flexibility given by an in-house online judge system, allowing unprecedented research depth and amenability to provide personalised individualised/group support for stakeholders. Indeed, our AI methods have the potential of improving the CS1 students learning by stimulating effective practice at the same time that reducing the instructors' workload. Our results include: i) a cutting-edge interpretable machine learning method that predicts the learners' performance and explains individually and collectively factors that lead to failure or success; (ii) a method that, for the first time, to the best of our knowledge, detects early effective programming behaviours and indicates how those positive behaviours can be used to guide students with ineffective behaviours; iii) a novel prescriptive model that automatically detects the topic of problems achieving state-of-the-art results and makes problems' recommendations based on that and the students' programming profiles. Finally, we also explored how our AI methods could be used in collaboration with the instructors' intelligence, giving thus a move towards novel human/AI online judge architecture to support the decision making of CS1 instructors and students. To do so, the results of our methods are represented in the format of hybrid human/AI concept designs, which are validated consistently and systematically by CS1 instructors, who are responsible for deciding which concept designs should be available to the students.

Keywords: CS1, machine learning, human/AI intelligence, hybrid intelligence, predictive analytics, prescriptive analytics, recommendation systems, topic extraction, NLP.

List of Figures

Figura 1 – Typical architecture of an OJ system 28
Figura 2 – Extension of a typical Human/AI OJ architecture to support CS1 teaching and learning. 34
Figura 3 – Evolution of pass rate in CS1 classes. 53
Figura 4 – CS1 course configuration 53
Figura 5 – Non-CS undergraduate courses at the Federal University of the Amazonas 54
Figura 6 – Screenshot of the student interface of CodeBench. 55
Figura 7 – Logs collected from CodeBench when the learner was writing a *print* command. 56
Figura 8 – Pairwise Spearman's rho correlation between behavioural features . 59
Figura 9 – Pairwise Spearman's rho correlation between evaluative factors. . . . 61
Figura 10 – Evaluative factors distribution of each cluster. 64
Figura 11 – Programming profile of students in each cluster 65
Figura 12 – Association Rules representation. 67
Figura 13 – Operators for building the ML pipelines using TPOT method. 87
Figura 14 – Comparing performance of the predictive models - deep Learning vs. evolutionary algorithm for optimisation 92
Figura 15 – Precision and recall curve and ROC curve of the deep learning model 94
Figura 16 – Choosing the optimal alpha for our Ridge Regression Model 96
Figura 17 – Prediction error of our regression model 98
Figura 18 – Residuals of our regression model. 98

Figura 19 – Prediction error (residuals) of our regression model in another perspective. 99
Figura 20 – Pareto plot of the feature's relevance based on their coefficients. . . . 100
Figura 21 – Workflow of the SHAP method. 113
Figura 22 – Learning Curve of our interpretable predictive model 118
Figura 23 – Precision and recall curve and ROC curve of our interpretable predictive model 119
Figura 24 – Decision plots to explain the potential leading factors for passing or failing 120
Figura 25 – Force plot of a given student. 122
Figura 26 – Sample of students' prediction paths 124
Figura 27 – Cluster centroids of prediction paths 125
Figura 28 – Prediction forceplots rotated 90 degrees, stacked horizontally. Here we highlight the 896th instance, showing her/his feature values and contributions. 129
Figura 29 – Summary of features' importance for the model's decision. 131
Figura 30 – Workflow of Behavioural Based Recommender System 147
Figura 31 – Representation of a recommendation list 150
Figura 32 – Affective states classified based on learners' comments. 152
Figura 33 – Analysis of the recommendations submitted to the OJ. 154
Figura 34 – Analysis of *logs* obtained in the evaluation of programming students. 156
Figura 35 – Limitation of the recommender system presented in this Chapter. . . 160
Figura 36 – Our front-heavy pipeline to automatically categorise programming problems based on their statements. 175
Figura 37 – Methods comparison. Measure: interchangeable. 184
Figura 38 – Methods comparison. Measure: coding effort. 185
Figura 39 – Methods comparison. Measure: resolution time. 185
Figura 40 – Methods comparison. Measure: hit rate. 185
Figura 41 – Methods comparison. Measure: topic. 186

Figura 42 – Extension of a typical Human/AI OJ architecture to support CS1 teaching and learning. 196
Figura 43 – Quantitative evaluation of concepts performed by the CS1 instructors. 206
Figura 44 – Results of the XGboost model for all sessions. 237
Figura 45 – Programming profile of students in each cluster for the remaining sessions (s2-s7). 239
Figura 46 – Distributions of programming behaviours (features) and distribution of our dependent variable (final grade) before discretisation. 241

LIST OF TABLES

Tabela 1 – Number of instances in CodeBench's data set (by term) 57

Tabela 2 – Features (programming behaviour) used in the machine learning models 58

Tabela 3 – Pairwise cluster comparisons of evaluative factors. 66

Tabela 4 – Statistics of the predictive models outcomes (better results are bolded). C.I. stands for Confidence Interval, L.B. stands for Lower Bound, and U.B. stands for Upper Bound. 92

Tabela 5 – Wilcoxon W test to compare the predictive models. Effect size (η^2) is considered small if η^2<0.01, it is medium if $.01 < \eta^2 \leq .06$, and high if $\eta^2 > 0.06$ (COHEN, 2013) 93

Tabela 6 – Regression Results. Df stands for degrees of freedom, MAE stands for mean absolute error 96

Tabela 7 – Coefficients of our regression model. Statitiscally significan coefficients bolded ($p - value < 0.05$). 101

Tabela 8 – Comparison of our prediction model and our baselines. C.I. stands for Confidence Interval, L.B. stands for Lower Bound, and U.B stands for Upper Bound. 117

Tabela 9 – Data-driven features to represent student's effort to solve programming problems. 145

Tabela 10 – Affective states used to evaluate learner's comments 148

Tabela 11 – Results of the Mann-Whitney test in the distribution of the characteristics of the recommendation systems. 156

Tabela 12 – Description of Topics based on CS1 Syllabus on CodeBench. 172
Tabela 13 – Unified topics for the experimental evaluation 179
Tabela 14 – List A: recommended-test list; List B = baseline-recommended list; Items = interchangeable, topic, resolution time, coding effort, hit rate. 180
Tabela 15 – Comparison of number of problems evaluated as Equivalent (Eq) and Non-Equivalent (N-Eq) for each measure, where List A was created using our method, whereas List B used our baseline. 184

CONTENTS

1 INTRODUCTION 20

1.1 General Objective 24

1.2 Specific Objectives 24

1.3 Thesis Overview 25

1.4 Programming Online Judges and CS1 26

1.5 Using AI over data collected from Online Judges 28

1.6 Hybrid human/AI systems 30

1.7 A novel hybrid human/AI Programming Online Judge architecture 32

1.8 Scope and Generalities 36

1.9 List of Publications and Awards 37

1.10 Thesis Guideline 44

2 PROGRAMMING PROFILE WITH EFFECTIVE AND INEFFECTIVE PROGRAMMING BEHAVIOURS 47

2.1 Overview of the Chapter 47

2.2 Practitioner Notes 48

2.3 Research Questions Addressed in this Chapter 49

2.4 Definition of effectiveness, ineffectiveness and resilience . 50

2.5 Effective and Ineffective Behaviours in Programming - State of the art 50

2.6 Educational Context 52

2.7 Method 55

2.7.1 Instrument 55

2.7.2 Data Collection 56

2.7.3 Features Extraction and Selection . . . 56
2.7.4 Evaluative Factors . . . 59
2.7.5 Clustering and Association Rule algorithms . . . 62
2.8 Results and Discussion . . . 63
2.8.1 Analysing Consolidation Data . . . 63
2.8.2 Comparing Student Clusters . . . 64
2.8.3 Association rules analysis . . . 66
2.9 Pedagogical Implications . . . 68
2.10 Chapter Conclusions . . . 70

3 EARLY PERFORMANCE PREDICTION . . . 72
3.1 Overview of the Chapter . . . 72
3.1.1 Practitioner Notes . . . 73
3.2 Research Questions Addressed in this Chapter . . . 75
3.3 Early Performance Prediction - State of the art . . . 77
3.4 Method . . . 82
3.4.1 Instrument . . . 82
3.4.2 Data Collection and Programming Behaviours . . . 82
3.4.3 Student Programming Profile . . . 83
3.4.4 Cleaning the features and dealing with imbalanced dataset . . . 84
3.4.5 Prediction and Validation . . . 85
3.4.5.1 Genetic Algorithms and Automatic Machine Learning . . . 86
3.4.5.2 Deep Learning Model . . . 87
3.4.5.3 Regression Model . . . 88
3.4.5.4 Validation . . . 89
3.5 Results and Discussion . . . 90
3.5.1 Deep Learning versus Evolutionary Algorithm . . . 90
3.5.2 Performance Analysis Per class . . . 94
3.5.3 Regression Analysis . . . 95
3.5.4 Analysis of Feature Effects on the Regression Outcomes . . . 98
3.6 Pedagogical Implication of Early Prediction . . . 102

3.7 Chapter Conclusions 104

4 EXPLAINING INDIVIDUAL AND COLLECTIVE EFFECTIVE AND INEFFECTIVE PROGRAMMING STUDENTS' BEHAVIOUR 106

4.1 Overview of the Chapter 106

4.1.1 Practitioner Notes 107

4.2 Research Questions Addressed in this Chapter 109

4.3 Explainable Machine Learning 111

4.3.1 Machine Learning Models 114

4.4 Method 115

4.4.1 Prediction and Validation 116

4.5 Results and Discussions 116

4.5.1 Reliability of the XGBoost model 117

4.5.2 Interpretation of the predictive model 119

4.5.3 Individual analysis 119

4.5.3.1 Explaining individual programming behaviours of a learner with a high chance of passing 121

4.5.3.2 Explaining individual programming behaviours of a learner with a high chance of failing 122

4.5.4 Small group analysis (Passed versus Failed) 123

4.5.5 Prediction paths 124

4.5.6 Global Analysis 130

4.6 Pedagogical Implications 131

4.7 Chapter Conclusions 134

5 RECOMMENDING PROBLEMS FOR STUDENTS BASED ON EFFECTIVE AND INEFFECTIVE BEHAVIOURS 136

5.1 Overview of the Chapter 136

5.1.1 Practitioner Notes 137

5.2 Research Question Addressed in this Chapter 138

5.3 Affective states whilst learning to program 141

5.4 Recommender Systems in Online Judges - State-of-the-art 142
5.5 Methods 144
5.5.1 Feature Extraction to Represent Learner's Effort 144
5.5.2 Behaviour-based Recommender System 146
5.5.3 Evaluation of the Recommender Model 148
5.5.3.1 Participants 148
5.5.3.2 Measures 148
5.5.3.3 Experimental Manipulation 149
5.6 Results and Discussions 151
5.7 Pedagogical Implications 157
5.8 Chapter Conclusions 159

6 RECOMMENDING PROBLEMS FOR INSTRUCTORS TO COMPOSE ASSIGNMENTS AND EXAMS 162
6.1 Overview of the Chapter 162
6.1.1 Practitioner Notes 163
6.2 Research Question Addressed in this Chapter 164
6.3 State of the art 166
6.3.1 Classification of Topics from Programming Online Judges problems 166
6.3.2 Problem Recommendation 168
6.4 Data 171
6.5 Effort-topic based Recommender 171
6.5.1 Recommendation mechanism 173
6.5.1.1 Topic Detection 175
6.5.1.2 Finding the nearest neighbour 176
6.6 Evaluation 177
6.6.1 Procedure 179
6.6.2 Measures 179
6.6.3 Pilot 181
6.6.4 Participants 181

6.6.4.1 Data filtering . . . 182
6.6.5 Instructors answer analysis . . . 182
6.7 Results . . . 183
6.8 Discussion . . . 186
6.8.1 General . . . 186
6.8.2 Human/AI collaboration . . . 188
6.8.3 Pedagogical Implications . . . 190
6.9 Chapter Conclusion . . . 191

7 VALIDATION OF THE HYBRID AI/HUMAN ONLINE JUDGE ARCHITECTURE . . . 193
7.1 Overview of the Chapter . . . 193
7.2 Research Question Addressed in this Chapter . . . 195
7.2.1 Hybrid human/AI systems . . . 197
7.2.1.1 Dimensions for hybrid Human-AI systems . . . 198
7.3 Method . . . 199
7.3.1 Speed Dating Design Method . . . 199
7.3.2 Pilot and Concepts Definition . . . 201
7.3.3 Descriptions of the Validated Design Concepts . . . 202
7.4 Results and Discussion . . . 206
7.4.1 Concepts Classification . . . 206
7.4.2 Concepts Sumarisation in Instructors' Perspectives . . . 207
7.4.3 General Analysis . . . 220
7.4.4 Hybrid Human/AI Collaboration . . . 222
7.5 Chapter Conclusions . . . 223

8 CONCLUSIONS . . . 225
8.1 Findings, Applications and Implications . . . 226
8.2 Limitations . . . 232
8.3 Future Work . . . 235

APPENDIX A APPENDIX I . . . 237

APPENDIX B APPENDIX II 238

APPENDIX C APPENDIX III 240

C.0.1 Characterisation of Students' Programming Behaviours 240

Bibliography 246

1

INTRODUCTION

> In God we trust; all others must bring data.
>
> - William Edwards Deming

Introductory programming (also known as CS1) , one of the basic topics in Computer Science and STEM programmes, might be considered one of the most complex courses for many learners (ULLAH *et al.*, 2018; ARAUJO *et al.*, 2021; LIMA *et al.*, 2021b; BRAZ *et al.*, 2021b; MENDONÇA *et al.*, 2021). Previous studies report high failure rates in CS1 (CARVALHO *et al.*, 2016; ROBINS, 2019; LUXTON-REILLY *et al.*, 2018), posing significant challenges for instructors, who naturally wish their learners to progress effectively. Additionally, instructors may face institutional pressures if students do not perform well (ROBINS, 2019).

In face of those challenges, a common agreement from computing education research is that programming students need practice and quick feedback on the correctness of their codes (IHANTOLA *et al.*, 2015; CARVALHO *et al.*, 2016; LUXTON-REILLY *et al.*, 2018; ROBINS, 2019; OLIVEIRA *et al.*, 2020; BRAZ *et al.*, 2021a). However, instructors usually face large and heterogeneous classes, which makes well-crafted assessment and individualised/group/personalised support almost impractical (ANDERSEN *et al.*, 2016; CARVALHO *et al.*, 2016; PEREIRA *et al.*, 2020; COSTA *et al.*, 2021). For instructors, the typical challenges confronted are the difficulty and endeavour needed to assess students' codes (SOUZA *et al.*, 2016) and to deliver individualised, timely feedback based on that assessment (IHANTOLA *et al.*, 2015; ULLAH *et al.*, 2018). For students,

it is also essential to receive precise and quick feedback, so that they can correct their mistakes in a timely fashion (IHANTOLA *et al.*, 2015; WASIK *et al.*, 2018; ROBINS, 2019; LUXTON-REILLY *et al.*, 2018).

In this sense, Programming Online Judges (OJs) are learning environments targeted for programming learning that can bring many advantages (BEZ *et al.*, 2014; WASIK *et al.*, 2018; LIMA *et al.*, 2020; PESSOA *et al.*, 2021). Recently, these systems are increasingly being used to support CS1 classes, as they reduce instructors' workload in correcting the learners' programming tasks and provide instantaneous and accurate feedback to students about the correctness of their solutions (WASIK *et al.*, 2018; PEREIRA *et al.*, 2020). The typical architecture of an OJ provides an Integrated Development Environment (IDE) where learners can develop, test, compile, run, and submit solutions to programming assignments and exams (IHANTOLA *et al.*, 2015; CARTER *et al.*, 2019; PEREIRA *et al.*, 2020a). The instructors' role is generally to select problems to compose assignments lists and exams on different topics covered in the CS1 course (KURNIA *et al.*, 2001; BEZ *et al.*, 2014; IHANTOLA *et al.*, 2015; WASIK *et al.*, 2018; PEREIRA *et al.*, 2020a). In addition, the instructors can visualise the students' progress in these assessments.

Nonetheless, ordinary OJ systems do not provide any learning analytical tool or adaptive approach to provide support for instructors to teach in such numerous and heterogeneous classes as required for CS1 (CARVALHO *et al.*, 2016; WASIK *et al.*, 2018; PEREIRA *et al.*, 2020a). It is important to note that the use of AI to design adaptive approaches is claimed to be more powerful for instruction, due to students' variance in abilities and performance (WOOLF, 2010; ANDERSEN *et al.*, 2016; TENÓRIO *et al.*, 2021). In this sense, studies (IHANTOLA *et al.*, 2015; SOUZA *et al.*, 2016; WASIK *et al.*, 2018; PEREIRA *et al.*, 2020a; BILEGJARGAL; HSUEH, 2021) indicate that the typical architecture of OJ systems needs to be extended to leverage CS1 teaching and learning. To illustrate, Pereira *et al.* (2021a), Pereira *et al.* (2021b) propose an AI-agent to support the instructors in the selection of problems to compose assignments in CS1 courses. This process is known to be cumbersome, due to there being many problems to search manually in OJs. Instead, using an AI agent, data can be collected and processed automatically.

Moreover, these data can lead to a better understanding of learners' behaviours and help in enhancing the instructors' methodology, decision-making process, and allowing prompt interventions (IHANTOLA *et al.*, 2015; WASIK *et al.*, 2018; CARTER *et al.*, 2019; PEREIRA *et al.*, 2020a).

Nevertheless, recent literature claims that AI alone is not enough and must not disregard the instructors' expertise from their daily classroom experience (DELLERMANN *et al.*, 2019; AKATA *et al.*, 2020; HOLSTEIN *et al.*, 2020; TENÓRIO *et al.*, 2021). Dellermann *et al.* (2019), Akata *et al.* (2020) point out that the combination of humans and AI can provide better results than AI and humans alone. Indeed, instructors might have essential information and knowledge that the Artificial Intelligence Educational (AIEd) system has no access to (HOLSTEIN *et al.*, 2020). Thus, hybrid systems where AI and instructors collaborate are required. In such systems, instructors might help the AI with cases where it misinterprets the data; or where it needs creativity, intuition, flexibility, or empathy. Similarly, instructors can benefit from the information that the AIEd system has exclusive access to.

Despite their potential, there is a lack of studies on proposing designing and validating these hybrid systems or methods. Many works (RODRÍGUEZ-TRIANA *et al.*, 2018; CHEN *et al.*, 2018; DE-ARTEAGA *et al.*, 2020; TENÓRIO *et al.*, 2021) point to the need for studies that explore the ways humans can augment the AIEd interpretation and understanding of the data, and vice-versa. Specifically, Holstein *et al.* (2020) claim a need for research to propose domain-specific hybrid systems and explore concept designs for these agents (human and AI) to augment their intelligence. To the best of our knowledge, there are no studies available addressing opportunities to propose, design, and validate hybrid methods and architectures in the online judges' field.

In light of this, this work proposes, designs and validates descriptive, predictive and prescriptive AI methods to compose a *novel hybrid human/AI online judge architecture* to support the decision-making process of instructors and students in CS1 classes. Our novel architecture is designed to extend the commonly used OJ architecture. To do so, our AI methods encompass descriptive, predictive and prescriptive methods that use a programming profile from the CS1 students to support the CS1 instructors

and students. The methods employ fine-grained data collected from the OJ systems when students are solving programming problems. In other words, using OJ data, our methods can perform: i) descriptive analysis of students' behaviours by structuring their data, ii) predictive analysis by estimating the students' outcomes, and iii) prescriptive analysis, for example, by recommending pedagogical items of providing feedback about important patterns found from the eXplainable Artificial Intelligence (XAI) approach.

Finally, we further validate our methods in format of concept designs to support instructors to enhance the students leaning in CS1 classes. Our validation involves the integration of those concept designs in a novel hybrid human/AI architecture that could potentially extends to typical OJ architecture. We opted to validate the concept-designs with instructors because all the potential application of our methods are mediated by them.

Moreover, we validate the concepts systematically and consistently by using the speed dating method (DAVIDOFF *et al.*, 2007), which is a user experience approach to explore the application of concepts in new systems without requiring any technology implementation. This is important in our case, because although our descriptive, predictive and prescriptive methods have been implemented and validated in our experimental settings, they have not yet been employed in real scenarios with CS1 students.

Briefly, the main contributions of this work are as follows:

- Proposing and validating cutting-edge AI descriptive, predictive and prescriptive methods to support CS1 teaching and learning.
- Combining those methods in a novel architecture for a hybrid human/AI OJ system to support CS1 teaching and, hence, the learning process in CS1.
- The validation of this novel architecture from the perspective of CS1 instructors' with OJ experience.

1.1 General Objective

This study aims to propose, design and validate a hybrid human/AI OJ architecture to create mechanisms to support the decision-making of CS1 instructors to enhance students' learning. For instructors, these mechanisms must be carried out in order to help them improve their pedagogical and methodological practices, enabling early and long-term interventions that minimise the chances of at-risk students to end up failing, at the same time that enhance the chances of students with a high probability of passing. For students, support is mediated by the instructors aiming at description, prediction and prescriptions/recommendations of programming behaviours that can increase the student's chances of being approved in the course.

Finally, we also have the goal of creating a mechanism to reduce the instructors workload in composing assignments and exams, given to them more time for other tasks, such as improving their methodology, reflecting on potential interventions, and adapting their pedagogy.

1.2 Specific Objectives

The following are the specific objectives of this study:

- Collect fine-grained data from students in CS1 classes as they solve coding problems in an IDE built into the OJ.
- Compose a programming profile based on features that represent programming behaviours that can increase or decrease the chances of CS1 students being approved.
- Propose and validate descriptive and predictive methods that apply AI techniques (e.g., clustering and classification) that detect which programming behaviours can increase or decrease CS1 students' chances of passing.
- Based on the methods generated from the aforementioned objectives, propose and validate prescriptive methods (recommendation) that provide support to CS1

instructors and students[1] decision making.

- Validate the methods generated consistently and systematically with CS1 instructors.

1.3 Thesis Overview

In this work we present how helpful typical OJ are to introductory programming (also known as CS1) classes. Furthermore, we show the gaps left by those systems that can be potentially fulfilled by using AI methods combined with the CS1 instructors' intelligence (hybrid human/AI intelligence). More specifically, we demonstrated how typical OJ architectures can be extended using human/AI hybrid intelligence, to support the decision making of CS1 instructors to enhance the CS1 learning. We do that by first proposing and validating AI methods to support instructors, in which they are responsible for mediating which applications from our methods should be exposed to the students. Moreover, we explore in which extent our methods might enhance the instructors' knowledge about the students and minimise instructors' workload. Those potential application of our methods are then represented in format of hybrid human/AI concept designs, and validated consistently and systematically by our target users - CS1 instructors. Finally, we combined and integrated those concept designs in our novel human/AI OJ architecture. In other words, we propose and validate AI methods to extend the OJ typical architecture, and then validate the hybrid human/AI concept designs, derived from our AI methods, with CS1 instructors.

Our results are important for the field of CS1 learning as a step towards enhancing instructors decision making power and students learning. Moreover, our results can also be helpful to optimise the instructors work time, freeing their time in repetitive tasks such as selecting problems to compose assignments and exams.

It is worth noting that this thesis is a portfolio of articles that converge to our validated hybrid human/AI OJ architecture to support CS1 classes. In this sense, the chapters of this

[1] In this work, all methods presented to the students should be mediated by the instructors first. So that, first the instructors decide which methods should be exposed to the students to improve their learning.

thesis are an aggregation of our publications, whilst this chapter is an overview of them and, hence, an overview of the thesis. Thus, we present the novelty of each of our AI methods in chapters 2-6 and the integration of those in Chapter 7. Moreover, our research questions is presented in each Chapter separately since we address specific problems in each chapter. By addressing these problems, we move towards achieving our general objective (Section 1.1).

In the next sections, we explore and contextualise the use of OJ systems in CS1, explain how data is collected from these systems, show how hybrid intelligence (human/AI) can be employed using such data, and then we present our hybrid human/AI architecture to extend typical OJ architecture.

1.4 Programming Online Judges and CS1

Many studies (BEZ *et al.*, 2014; SOUZA *et al.*, 2016; CARVALHO *et al.*, 2016; WASIK *et al.*, 2018; ZHOU *et al.*, 2018; PEREIRA *et al.*, 2020a; BILEGJARGAL; HSUEH, 2021) showed that OJs could bring benefits for CS1 students and instructors. This is due to these systems providing instantaneous and precise feedback to students and reducing the instructors' workload in code evaluation. For instance, with manual code evaluation, in a class with 100 students, if an instructor creates an assignment with 10 coding exercises, they need to evaluate 1000 pieces of code ($100*10$). Note that in CS1, instructors typically employ a 'Many Small Programs' (MSP) approach (ALLEN *et al.*, 2018; ALLEN; VAHID, 2021), which requires learners to solve many small problems weekly during the course. Accordingly, evaluating all students' codes precisely and fairly is unrealistic.

In a pioneering study about OJ, Kurnia *et al.* (2001) compared human code evaluation versus the OJ mechanism of automatic code correction. As a result, the authors showed that, when implemented carefully, OJs are more secure and convenient. In addition, manual evaluation carries human subjectivity, which may bias the code evaluation. Thus, OJ correction brings more fairness to the evaluation process (KURNIA *et al.*, 2001; ZHOU *et al.*, 2018). Furthermore, Choy *et al.* (2005) demonstrated that OJ systems might be beneficial for student motivation. They observed that learners were

encouraged to practice more, without fear of being judged for their programming mistakes, which piqued students' interest in developing their programming skills.

Despite these benefits, recent works state that the OJ architecture needs to be extended, to better support the instructors and enhance learning (IHANTOLA *et al.*, 2015; SOUZA *et al.*, 2016; WASIK *et al.*, 2018; PEREIRA *et al.*, 2020a; BILEGJARGAL; HSUEH, 2021). Ihantola *et al.* (2015), Carter *et al.* (2019) claim that it is important to understand how students can improve their codes and learning process to provide refined information for instructors. Luxton-Reilly *et al.* (2018), Tenório *et al.* (2021) argue that CS1 instructors need tools to improve their pedagogy/methodology. Still, Zhou *et al.* (2018), Pereira *et al.* (2021a), Oliveira *et al.* (2021b), Lima *et al.* (2021a), Fowler & Zilles (2021) explain that it is challenging for the instructors to select problems to create the assignments and exams using the questions available on OJ systems, because there are typically thousands of problems available in these systems. Whilst high variation of questions is good for diversity, it often consumes too much time for the instructors to select the questions to compose assignments and exams.

Moreover, instructors usually face large and heterogeneous classes, which makes individualised/personalised or even group support almost impractical (ANDERSEN *et al.*, 2016; PRICE *et al.*, 2020). However, because the OJs feedback is only based on code correction, more research is needed to improve the OJ mechanisms to provide individualised and group support for heterogeneous CS1 classes and to encourage effective problem-solving practice and behaviours (KEUNING *et al.*, 2018; CARTER *et al.*, 2019; KARVELAS *et al.*, 2020; OLIVEIRA *et al.*, 2020; PEREIRA *et al.*, 2020). In addition, early individualised and group support is vital in CS1 classes, if behavioural changes are desired. Importantly, this support needs to be scalable for large and heterogeneous classes with an increasing number of students (PRICE *et al.*, 2016; RIVERS; KOEDINGER, 2017; MARWAN *et al.*, 2019a; MARWAN *et al.*, 2019b; PRICE *et al.*, 2020).

In this sense, AIEd research has shown, albeit mainly on small-scale teaching experiments, that analysing data from students is essential to improve the learning process and can be helpful to provide support for instructors and leverage the learning process (KULIK; FLETCHER, 2016; HOLSTEIN *et al.*, 2019; PEREIRA *et al.*, 2020a).

Furthermore, Holstein *et al.* (2020) explain that researchers should also use AIEd systems to help instructors in interpreting and drawing inferences from what they perceived and how to guide instructors towards interpretations of data. Moreover, it is essential to scaffold learners in more productive ways using the data available. In light of this, the following sections show how the data collected from OJ systems and AI techniques can be employed to support CS1 teaching and learning.

1.5 Using AI over data collected from Online Judges

Overall, OJ architectures provide students with an environment in which they can develop, compile, run and debug their code in an IDE, which is typically embedded in the OJ (IHANTOLA *et al.*, 2015; CARTER *et al.*, 2019; PEREIRA *et al.*, 2020a). After developing the code, students submit it and then receive feedback on the correctness of their solution. Figure 1 illustrates the typical architecture of an OJ system. Notice that the data can be collected at different granularities when students solve the problems (VIHAVAINEN *et al.*, 2014; IHANTOLA *et al.*, 2015; CARTER *et al.*, 2019; QUILLE; BERGIN, 2019; PEREIRA *et al.*, 2020a). That is, in addition to the code submitted by the student, it is possible to evaluate how students developed their code (e.g., the time spent to solve the problem, number, type of errors made, etc.).

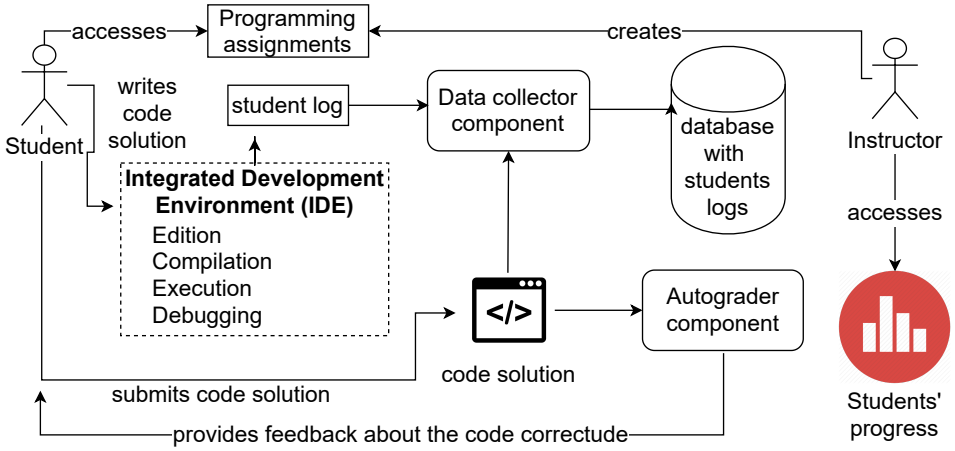

Figure 1 – Typical architecture of an OJ system used to support programming courses (PEREIRA *et al.*, 2020a).

Note that students' code and the process behind the code development play

a critical role in the students' assessment (BLIKSTEIN *et al.*, 2014; ZHOU *et al.*, 2018; EDWARDS *et al.*, 2020; FILHO *et al.*, 2020). Indeed, what improves students' abilities in programming is the experience they gather when solving problems, instead of just the outcome – whether the code is correct or not. As such, it is crucial to evaluate the code's correctness and the data-log collected when the students are solving the problems (WATSON *et al.*, 2013; VIHAVAINEN *et al.*, 2014; IHANTOLA *et al.*, 2015; CARTER *et al.*, 2019; QUILLE; BERGIN, 2019; PEREIRA *et al.*, 2020a; PEREIRA *et al.*, 2021).

In this direction, researchers in the Computer Education field (AHADI *et al.*, 2015; AHADI *et al.*, 2016; ESTEY; COADY, 2016; LEINONEN *et al.*, 2016; CASTRO-WUNSCH *et al.*, 2017; COSTA *et al.*, 2017; Abu Amra; Maghari, 2017; PEREIRA *et al.*, 2019b; FONSECA *et al.*, 2019; BOCKMON *et al.*, 2020; Kumar Veerasamy *et al.*, 2020; PEREIRA *et al.*, 2021) have used OJ data to model students' behaviour and quantify aspects of students' outcomes. Typically, these studies have in common that they perform a data-driven analysis, extract useful information, and use them as features in data mining, machine learning, and inferential statistical techniques to predict the CS1 students' performance, potentially, at an early stage in the course (CARTER *et al.*, 2019; AHADI *et al.*, 2016; CASTRO-WUNSCH *et al.*, 2017; ROBINS, 2019; PEREIRA *et al.*, 2021). However, few studies have used fine-grained OJ interaction data (e.g., keys pressed when coding, copy and paste patterns, etc.) to better understand students' performance and behaviours, often due to lack of data availability or inadequate granularity (PEREIRA *et al.*, 2020a).

Another limitation of these methods is that they were not employed in any educational setting and, hence, their effects were not measured in a real-life scenario (QUILLE; BERGIN, 2019; PEREIRA *et al.*, 2020a). Indeed, the methods proposed by the literature are evaluated in terms of accuracy, f1-score, precision, and other machine learning metrics, but not by target users. These methods remain in laboratory settings rather than being employed in CS1 classes supported by OJ systems. Also, note that one crucial step before implementation is the validation of design concepts of a potential novel functionality or method (HOLSTEIN *et al.*, 2020; TENÓRIO *et al.*, 2021).

Furthermore, for the adoption of these machine learning methods, Agrawal *et al.* (2018), Dellermann *et al.* (2019), Holstein *et al.* (2019), Pereira *et al.* (2021) explain that

they should also ensure interpretability and transparency on the predictive methods' decision. Besides the prediction provided by the ML methods, it is crucial to understand what leads the predictive method to make the decision. EXplainble Artificial Intelligence (XAI) appears as a viable solution, with methods (e.g., SHAP Lundberg & Lee (2017), LIME Ribeiro *et al.* (2016)) that can be used to explain why the model[2] estimated that a student has high/low chances of passing.

Thus, using OJ data, AI methods could perform: i) descriptive analysis of students' behaviours by structuring their data, ii) predictive analysis by estimating the students' outcomes, and iii) prescriptive analysis, for example, by recommending pedagogical items of providing feedback about important patterns found from the XAI approach.

Additionally, Luxton-Reilly *et al.* (2018) highlight that many works focus on the students' deficits, which is considered of high importance. However, it is also crucial to create mechanisms to improve pedagogical practice, since teaching CS1 students to program is complex and laborious. Moreover, when we enhance the teaching process, then we potentially would leverage the learning as well (LUXTON-REILLY *et al.*, 2018; PEREIRA *et al.*, 2020a; TENÓRIO *et al.*, 2021).

Finally, although AI applications are potentially helpful, in practice, AIEd or adaptive systems in education are generally implemented in the form of an AI agent, working with a mediator, such as an instructor. In this sense, some of the recent AIEd literature (VANLEHN, 2016; GERRITSEN *et al.*, 2018; MOLENAAR *et al.*, 2019; HOLSTEIN *et al.*, 2020; PEREIRA *et al.*, 2021) highlights a need for both AI that help humans and humans that supports AI, to support effective human decisions and teachable machines in a move towards hybrid human/AI systems. We explore this in the following sections.

1.6 Hybrid human/AI systems

Instead of combining homogeneous agents (humans, animals), hybrid intelligence combines the complementary strengths of heterogeneous intelligent agents (e.g., instructors

[2] Generated using the ML method

and AI). Although AI is being adopted much less than ideal in educational practice (HOLSTEIN *et al.*, 2020), it could be applied to support instructors in structuring learners' data, recognising hidden patterns, making forecasts, and so forth. Notice that humans often act non-Bayesian making inconsistent decisions and violating probabilistic rules. In light of this, the analytical power of AI grounded by the statistical and probabilistic foundation can empower instructors in making more effective, consistent, and accurate decisions (AGRAWAL *et al.*, 2018; DELLERMANN *et al.*, 2019).

Moreover, humans' intuition can empower the choices from the AI predictions or prescriptions. Indeed, on the opposite path (humans helping AI), humans provide domain knowledge and instructions to teach machines in many tasks that they cannot do by themselves. A typical example is when humans provide labels for supervised machine learning approaches to train the models or to make sense of unsupervised approaches. Also, humans are notorious better to perform tasks that require intuition, flexibility, creativity, empathy, and common sense (DELLERMANN *et al.*, 2019).

As such, well-designed AIEd systems may allow both agents (AI and human) to augment the intelligence of each other (HOLSTEIN *et al.*, 2020; DE-ARTEAGA *et al.*, 2020). In other words, AIEd systems will potentially work more effectively by combining the complementary strengths of humans and vice-versa (MOLENAAR *et al.*, 2019; VANLEHN *et al.*, 2021); such a combination is called hybrid intelligence. Many authors claim that hybrid systems will be a dominant method in many fields (DELLERMANN *et al.*, 2019; HOLSTEIN *et al.*, 2019; MOLENAAR *et al.*, 2019; DE-ARTEAGA *et al.*, 2020). However, there is a lack of studies about proposing, designing and validating human/AI hybrid methods and systems. For instance, Dellermann *et al.* (2019) claim that more research is necessary to build domain-specific human/AI educational systems and to explore interface designs that allow users helpers to teach an AI system and vice-versa.

In light of this, Holstein *et al.* (2020) proposed a conceptual framework to map distinct ways of human/AI collaboration. The framework shows dimensions that capture crucial components of an AIEd system. These dimensions are based on frameworks available in the literature (NEWELL, 1994; RUMMEL, 2018; ALEVEN *et al.*, 2016; VAN-

LEHN *et al.*, 2021), which were captured and adapted in a more general conceptual framework for human/AI hybrid systems. Due to the generality, we opted to use the dimensions presented in Holstein *et al.* (2020) to capture how humans and AI can augment each other's abilities in our hybrid human/AI OJ architecture. The dimensions are: goal augmentation, perceptual augmentation, action augmentation, and decision augmentation. These dimensions are helpful to provide guidance on the design of hybrid systems and they will be discussed in depth in Chapter 7.

1.7 A novel hybrid human/AI Programming Online Judge architecture

Based on the dimensions pointed out by Holstein *et al.* (2020) and the AI methods we propose and validate in this research, we proposed our hybrid AI/Human OJ architecture.

Importantly, this research adopted as a premise that the AI cannot make the final decision unassisted in our architecture of human/AI hybrid OJ system. The instructor mediate all the final decisions since they are responsible for using software in their classes. As such, although the instructor could augment the AIEd decision-making process, the instructor must be the terminal node of the pipeline and control of what is available for the students. We believe this is important to consider how effective the AI decisions are and to calibrate the AI during a cycle of human validation of the AI decisions.

Moreover, our OJ architecture requires collecting fine-grained data from students whilst solving problems in the IDE embedded in the OJ. Fine-grained data provide refined information about the students learning process that can be translated to meaningful feedback (IHANTOLA *et al.*, 2015; WASIK *et al.*, 2018; CARTER *et al.*, 2019; PRICE *et al.*, 2020; PEREIRA *et al.*, 2020a).

More specifically, logs, codes, and interaction data with the OJ system are needed to represent students' behaviours. Figure 7 (Chapter 2) shows an example of log data extracted from an open dataset of fine-grained data collected from an OJ system.

We employ this data to extract features that compose the programming profile from the learners. Such programming profile represents their effective and ineffective programming behaviours. Effective means behaviours that increase the students' chances of passing, whereas ineffective is the opposite (ROBINS, 2019; PEREIRA *et al.*, 2020). These programming profiles, in turn, can be used to produce descriptive, predictive, and prescriptive methods based on AI techniques. Table 2 (Chapter 2) presents the features we propose to represent the programming profiles of the students. The following is an illustration of three features used in our programming profile:

- procrastination: time between first code edit and assignment deadline;
- events: number of log lines;
- ideUsage: total time to solve problems, disregarding inactive use of the IDE;

We chose the features based on a systematic mapping of the literature that we carried out (PEREIRA *et al.*, 2020a) and discussions between the authors about possible useful information for describing metrics that measure programming practices that can lead to pass or fail. Additionally, for a specific task related to one of our methods we use text-data from problems' statements.

Figure 2 presents the architecture of our novel human/AI OJ system proposed in this study. Instructors might create programming assignments for students supported by the AI. Moreover, descriptive, predictive and prescriptive methods provide meaningful feedback for the instructors based on how students solve the programming assignments. More specifically, descriptive methods will be used for instructors to gain insights from the fine-grained data collected and through visualisation and statistical analysis of the relationships of programming profile attributes with student performance. Predictive methods can estimate student behaviours and performance to allow effective intervention. Finally, prescriptive methods can be used so that instructors better understand the possible factors that are leading students to success or fail. Moreover, they support the instructors in their more recurrent role in OJs, selecting problems to compose assignments and exams. All those applications of descriptive, predictive, and prescriptive methods are examples of how the AI could enhance the instructors'

instructional goals, perception, actions, and decision-making (the dimensions defined by Holstein *et al.* (2020)).

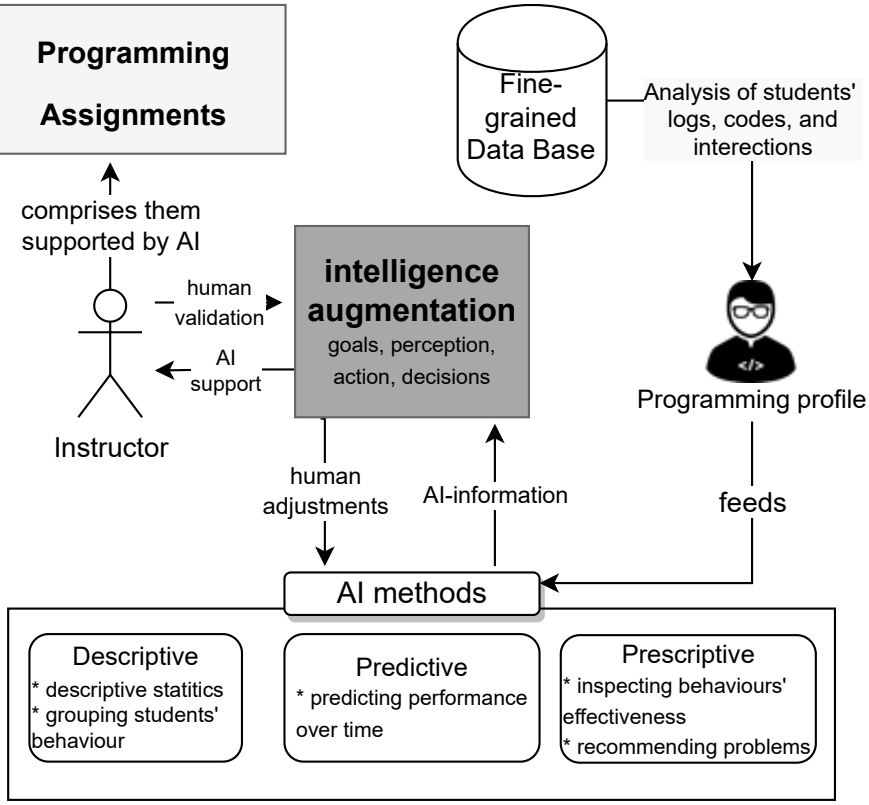

Figure 2 – Extension of a typical Human/AI OJ architecture to support CS1 teaching and learning.

On the other hand, the instructor can expand the AI capabilities mainly by relabelling descriptions, prediction, prescription and inserting domain knowledge based on their daily experience to the AI algorithms. To illustrate, if the instructors perceive that the predictive model has misclassified a student, they can adjust it to improve the system. In this way, the model can be periodically be retrained to learn from the instructor's input.

Finally, it is worth noting that the results of all methods proposed in this study were compared with methods presented in the literature. In addition, the recommendation method was evaluated in a double-blind controlled study, in which instructors were exposed to our method and a baseline method. In this way, all methods (descriptive, predictive and prescriptive) were validated in an experimental scenario before being exposed to instructors in the format of design concepts. The design concepts are potential

applications of the methods that can be incorporated into our novel human/AI hybrid OJ architecture. Finally, we validated the design concepts with CS1 teachers using the speed dating technique (DAVIDOFF *et al.*, 2007), which is a technique recommended for this purpose (HOLSTEIN *et al.*, 2019; HOLSTEIN *et al.*, 2020).

Briefly, the main steps (in chronological order) that were carried out in this study are:

- Collection and pre-processing of data used in our methods.
- Compose student programming profile.
- Descriptive data analysis through clustering of student behaviours for the construction and validation of a method that describes effective and ineffective programming behaviours.
- Construction of an interpretable predictive method for early prediction of student performance using data from the first two weeks of course[3].
- Local and global interpretation of predictive method decisions to identify which behaviours can potentially lead to pass/fail.
- Construction of an automatic problem recommender based on the programming profile features, software engineering code metrics and textual analysis (natural language processing) of problem statements.
- Comparison of methods with others presented in the literature.
- Implementation of design concepts with the descriptive, predictive and prescriptive methods in a move towards a hybrid human/AI OJ system.
- Use of speed dating to validate the design concepts with CS1 instructors.

[3] We also show that the predictive method can be accurately used not only in an early stage.

1.8 Scope and Generalities

The scope of this thesis is restricted to online judges focused on supporting introductory programming classes and not programming competitions. In addition, our target audience and stakeholders are CS1 instructors and students. We focus on introductory programming classes since this is the root of the problem: high failure rate, numerous and heterogeneous classes, many students struggling to learn to program, and the need of reducing instructors' workload. In addition, these introductory programming courses might be offered to non-CS courses and, hence, we might face the lack of motivation problem from non-CS students in learning to program.

Finally, to create our programming profile, we use fine-grained behavioural data collected from one online judge called CodeBench, due to convenience, since CodeBench is developed and maintained by one of the supervisors of this work (professor David Oliveira), what facilitates the process of data collection and sharing. Notice that fine-grained public data collected from online judges is scarce. For the text mining and NLP techniques, however, we use data collected from 3 different online judges (URI Online Judge, A2 Online Judge, and CodeBench) because such data is typically public in many OJs. Despite that, we strongly believe that the methods we built can be generalised to other educational contexts that use OJ's (or any variation of automatic assessment systems), or even Learning Management Systems (LMS) or Massive Open Online Course (MOOC) that allow granular collection of student data whilst they are solving programming problems. We claim that as the features we use were generally extracted from different works that were replicated in the literature. Moreover, our self-devised features are easily replicated since we will make available our extractor on Github[4] and our data[5]. Additionally, for the methods that use NLP, problem's statements have a similar structure, regardless of the OJ.

[4] github.com/filipedwan/CodeBench-Features-Extractor
[5] codebench.icomp.ufam.edu.br/dataset/

1.9 List of Publications and Awards

Following is the list of papers published during the doctorate. Each paper has an identifier that will be used in the next section to show how they relate to the chapter of this thesis. Moreover, we highlighted (italic) and provided the DOI of all papers directly related to this thesis.

Other articles which were not directly related to the chapters were not incorporated to this thesis as chapters. In those articles the author of this thesis worked on the conceptualisation, methodology, validation, writing - review and editing - of the studies.

Journal Papers:

J01 Júnior, H. B. F., **Pereira, F. D.**, da Silva Pereira, A. L., Silva, L. F. (2019). Reconhecimento de emoções em imagens utilizando técnicas de construção e otimização em métodos ensembles baseados em árvores de decisão. RCT-Revista de Ciência e Tecnologia, 5(8).

J02 ***Pereira, F. D.**, Oliveira, E. H., Oliveira, D. B., Cristea, A. I., Carvalho, L. S., Fonseca, S. C., Toda, A. Isotani, S. (2020). Using learning analytics in the Amazonas: understanding students' behaviour in introductory programming. British Journal of Educational Technology. doi: doi.org/10.1111/bjet.12953*

J03 ***Pereira, F. D.**, Fonseca, S. C., Oliveira, E. H., Oliveira, D. B., Cristea, A. I., Carvalho, L. S. (2020). Deep learning for early performance prediction of introductory programming students: a comparative and explanatory study. Brazilian journal of computers in education., 28, 723-749. doi: 10.5753/RBIE.2020.28.0.723*

J04 ***Pereira, F. D.**, Fonseca, S. C., Oliveira, E. H., Cristea, A. I., Bellhauser, H., Rodrigues, L., Oliveira D., Isotani, S. Carvalho, L. S. (2021). Explaining Individual and Collective Programming Students' Behavior by Interpreting a Black-Box Predictive Model. IEEE Access, 9, 117097-117119. doi: 10.1109/ACCESS.2021.3105956*

J05 Lima, M. , Carvalho, L. , Oliveira, E. H. T., Oliveira, D. , **Pereira, F. D.** (2021). Uso de atributos de código para classificar a dificuldade de questões de programação

em juízes online. Brazilian Journal of Computers in Education., 29, 1137-1157.

J06 de Sousa Freitas, F. D., **Pereira, F. D.** (2021). Mapeamento automático de rotas para vias expressas de ônibus: uma abordagem com algoritmos genéticos com ênfase na cidade de Boa Vista-RR. RCT-Revista de Ciência e Tecnologia, 7.

J07 Rodrigues, L., **Pereira, F. D.**, Toda, A., Palomino, P., Oliveira, W., Pessoa, M., Carvalho, L. S., Oliveira D., Oliveira, E. H., Cristea, A. I., Isotani, S. (2022). Are they learning or playing? Moderator conditions of gamification's success in programming classrooms. ACM Transactions on Computing Education.

J08 Rodrigues, L., **Pereira, F. D.**, Toda, A., Palomino, P., Pessoa, M., Carvalho, L. S., Oliveira D., Oliveira, E. H., Cristea, A. I., Isotani, S. (2022). Gamification suffers from the novelty effect but benefits from the familiarization effect: Findings from a longitudinal study. International Journal of Educational Technology in Higher Education.

Papers in International Conference and Workshops (with referee):

C01 ***Pereira, F. D.***, *Oliveira, E., Cristea, A., Fernandes, D., Silva, L., Aguiar, G., Alamri, A. Alshehri, M. (2019, June). Early dropout prediction for programming courses supported by online judges. In International Conference on Artificial Intelligence in Education (pp. 67-72). Springer, Cham. doi: doi: doi.org/10.1007/978-3-030-23207-8_13*

C02 Alamri, A., Alshehri, M., Cristea, A., **Pereira, F. D.**, Oliveira, E., Shi, L. Stewart, C. (2019, June). Predicting MOOCs dropout using only two easily obtainable features from the first week's activities. In International Conference on Intelligent Tutoring Systems (pp. 163-173). Springer, Cham.

C03 ***Pereira, F. D.***, *Oliveira, E. H., Fernandes, D. Cristea, A. (2019, July). Early performance prediction for CS1 course students using a combination of machine learning and an evolutionary algorithm. In 2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT) (Vol. 2161, pp. 183-184). IEEE. doi: 10.1109/ICALT.2019.00066*

C04 ***Pereira, F. D.***, *Oliveira, E., Fernandes, D., de Carvalho, L. S. G. Junior, H. (2019, November). Otimização e automação da predição precoce do desempenho de alunos que*

utilizam juízes online: uma abordagem com algoritmo genético. In Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE) (Vol. 30, No. 1, p. 1451). doi: 10.5753/cbie.sbie.2019.1451

C05 Fonseca, S., Oliveira, E., **Pereira, F. D.,** Fernandes, D. de Carvalho, L. S. G. (2019, November). Adaptação de um método preditivo para inferir o desempenho de alunos de programação. In Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE) (Vol. 30, No. 1, p. 1651).

C06 ***Pereira, F. D.**, Toda, A., Oliveira, E. H., Cristea, A. I., Isotani, S., Laranjeira, D., Almeida, A. Mendonça, J. (2020, June). Can We Use Gamification to Predict Students' Performance? A Case Study Supported by an Online Judge. In International Conference on Intelligent Tutoring Systems (pp. 259-269). Springer, Cham. doi: doi.org/10.1007/978-981-32-9335-9_6*

C07 Aljohani, T., **Pereira, F. D.,** Cristea, A. I. Oliveira, E. (2020, June). Prediction of Users' Professional Profile in MOOCs Only by Utilising Learners' Written Texts. In International Conference on Intelligent Tutoring Systems (pp. 163-173). Springer, Cham.

C08 Fonseca, S. C., **Pereira, F. D.**, Oliveira, E. H., Oliveira, D. B., Carvalho, L. S. Cristea, A. I. (2020, july). Automatic subject-based contextualisation of programming assignment lists. International Conference on Educational Data Mining (EDM 2020).

C09 ***Pereira, F. D.**, Souza, L., Oliveira, E., Oliveira, D., Carvalho, L. (2020, November). Predição de desempenho em ambientes computacionais para turmas de programação: um Mapeamento Sistemático da Literatura. In Anais do XXXI Simpósio Brasileiro de Informática na Educação, (pp. 1673-1682). Porto Alegre: SBC. doi: 10.5753/cbie.sbie.2020.1673*

C10 Toda, A., **Pereira, F. D.**, Klock, A., Rodrigues, L., Palomino, P., Oliveira, W., Oliveira, E., Gasparini, I., Cristea, A., Isotani, S. (2020, November). For whom should we gamify? Insights on the users intentions and context towards gamification in education. In Anais do XXXI Simpósio Brasileiro de Informática na Educação, (pp. 471-480). Porto Alegre: SBC.

C11 *Júnior, H., **Pereira, F. D.**, Oliveira, E., Oliveira, D., Carvalho, L. (2020, November). Recomendação Automática de Problemas em Juízes Online Usando Processamento de Linguagem Natural e Análise Dirigida aos Dados. In Anais do XXXI Simpósio Brasileiro de Informática na Educação, (pp. 1152-1161). Porto Alegre: SBC. doi: 10.5753/cbie.sbie.2020.1152.*

C12 Santos, I., Oliveira, D., Carvalho, L., **Pereira, F. D.**, Oliveira, E. (2020, November). Tempos de Transição em Estados de Corretude e Erro como Indicadores de Desempenho em Juízes Online. In Anais do XXXI Simpósio Brasileiro de Informática na Educação, (pp. 1283-1292). Porto Alegre: SBC.

C13 Lima, M., Carvalho, L., Oliveira, E., Oliveira, D., **Pereira, F. D.** (2020). Classificação de dificuldade de questões de programação com base em métricas de código. In Anais do XXXI Simpósio Brasileiro de Informática na Educação, (pp. 1323-1332). Porto Alegre: SBC.

C14 Filho, D., Oliveira, E., Carvalho, L., Pessoa, M., **Pereira, F. D.**, Oliveira, D. (2020, November). Uma análise orientada a dados para avaliar o impacto da gamificação de um juiz on-line no desempenho de estudantes. In Anais do XXXI Simpósio Brasileiro de Informática na Educação, (pp. 491-500). Porto Alegre: SBC.

C15 Oliveira, J., Salem, F., Oliveira, E., Oliveira, D., Carvalho, L., **Pereira, F. D.** (2020, November). Os estudantes leem as mensagens de feedback estendido exibidas em juízes online?. In Anais do XXXI Simpósio Brasileiro de Informática na Educação, (pp. 1723-1732). Porto Alegre: SBC.

C16 ***Pereira, F. D.**, Pires, F., Fonseca, S. C., Oliveira, E. H. T., Carvalho, L. S. G., Oliveira, D. B. F. Cristea, A. I. (2021, march). Towards a Human-AI Hybrid System for Categorising Programming Problems. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432422*

C17 de Lima Lima, M. A. P., de Carvalho, L. S. G., de Oliveira, E. H. T., de Oliveira, D. B. F., **Pereira, F. D.** (2021, April). Uso de atributos de código para classificação da

facilidade de questões de codificação. In Anais do Simpósio Brasileiro de Educação em Computação (pp. 113-122). SBC.

C18 Araujo, A., Zordan Filho, D. L., de Oliveira, E. H. T., de Carvalho, L. S. G., **Pereira, F. D.**, de Oliveira, D. B. F. (2021, April). Mapeamento e análise empírica de misconceptions comuns em avaliações de introdução à programação. In Anais do Simpósio Brasileiro de Educação em Computação (pp. 123-131). SBC.

C19 Alrajhi, L., Alamri, A., **Pereira, F. D.**, Cristea, A. I. (2021, June). Urgency Analysis of Learners' Comments: An Automated Intervention Priority Model for MOOC. In International Conference on Intelligent Tutoring Systems (pp. 148-160). Springer, Cham.

C20 ***Pereira, F. D.***, *Junior, H. B., Rodriguez, L., Toda, A., Oliveira, E. H., Cristea, A. I. Isotani, S. (2021, June). A recommender system based on effort: Towards minimising negative affects and maximising achievement in cs1 learning. In International Conference on Intelligent Tutoring Systems (pp. 466-480). Springer, Cham. doi: doi.org/10.1007/978-3-030-80421-3_51*

C21 Alamri, A., Sun, Z., Cristea, A. I., Stewart, C., **Pereira, F. D.** (2021, June). MOOC next week dropout prediction: weekly assessing time and learning patterns. In International Conference on Intelligent Tutoring Systems (pp. 119-130). Springer, Cham.

C22 Mendonça, E., Carvalho, L. S., dos Santos, A. L. M., Oliveira, E. H., Oliveira, D. B., **Pereira, F. D.** (2021, July). Vivência acadêmica e desempenho acadêmico de ingressantes em cursos de computação. In Anais do XXIX Workshop sobre Educação em Computação (pp. 458-467). SBC.

C23 Oliveira, J. C., Carvalho, L. S., Oliveira, E. H., Oliveira, D. B., **Pereira, F. D.** (2021, November). Correlação entre habilidade de resolução de problemas e desempenho em disciplina introdutória de programação. In Anais Estendidos do XXXII Simpósio Brasileiro de Informática na Educação (pp. 15-20). SBC.

C24 de Oliveira, D. B., Lavareda Filho, R. M., Oliveira, E. H., Carvalho, L. S., **Pereira, F. D.**, Colonna, J. G., Menezes, A. (2021, November). Um Método de Detecção de Plágio para Sistemas Juiz On-line baseado no Comportamento dos Alunos. In Anais do XXXII Simpósio Brasileiro de Informática na Educação (pp. 836-848). SBC.

C25 Pessoa, M., Melo, R., Haydar, G., de Oliveira, D. B., Carvalho, L. S., de Oliveira, E. H., **Pereira, F. D.**, Rodrigues, L. Isotani, S. (2021, November). Uma análise dos tipos de jogadores em uma plataforma de gamificação incorporada a um sistema juiz on-line. In Anais do XXXII Simpósio Brasileiro de Informática na Educação (pp. 474-486). SBC.

C26 Braz, A. C. R., Carvalho, L. S., Oliveira, E. H., Oliveira, D. B., Bittencourt, R. A., Santana, B. L., **Pereira, F. D.** (2021, November). Tradução e validação de um inventário de conceitos sobre programação introdutória. In Anais do XXXII Simpósio Brasileiro de Informática na Educação (pp. 1253-1264). SBC.

C27 Al-Rajhi, L., **Pereira, F. D.**, Cristea, A., Aljohani, T. (2022, July). A Good Classifier is Not Enough: a XAI Approach for Urgent Instructor-Intervention Models in MOOCs. In International Conference on Artificial Intelligence in Education. Springer, Cham.

C28 Toda, A., **Pereira, F. D.**, Palomino, P., Rodrigues, L., Klock, A., Borge, S., Oliveira, E., Gasparini, I., Isotani, S., Cristea, A. (2022, July). Gamification through the looking glass - perceived biases and ethical concerns of Brazilian teachers. In International Conference on Artificial Intelligence in Education. Springer, Cham.

C29 Rodrigues, L., Toda, A., **Pereira, F. D.**, Palomino, P., Klock, A., Pessoa, M., Oliveira, D., Gasparini, I., Oliveira, E., Cristea, A., Isotani, S. (2022, July). GARFIELD: A Recommender System to Personalize Gamified Learning. In International Conference on Artificial Intelligence in Education. Springer, Cham.

C30 Toda, A., Klock, A., **Pereira, F. D.**, Rodrigues, L., Palomino, P., Lopes, V., Stewart, C., Oliveira, E., Gasparini, I., Isotani, S., Cristea, A. (2022, July). Gamification

through the looking glass - perceived biases and ethical concerns of Brazilian teachers. In International Conference on Educational Data Mining (EDM 2022).

Abstracts in International Conference and Workshops (with referee):

A01 ***Pereira, F. D.**, de Freitas Júnior, H. B., de Oliveira, E. H. T., de Carvalho, L. S. G., de Oliveira, D. B. F., Benedict, A., Cristea, A. I. (2021, April). Automatic creating variation of CS1 assignments and exams. In Anais Estendidos do I Simpósio Brasileiro de Educação em Computação (pp. 21-22). SBC. doi: doi.org/10.5753/educomp_estendido.2021.14856*

A02 ***Pereira, F. D.**, de Oliveira, E. H. T., de Oliveira, D. B. F., de Carvalho, L. S. G., Cristea, A. I. (2021, April). Interpretable AI to Understand Early Effective and Ineffective Programming Behaviours from CS1 Learners. In Anais Estendidos do I Simpósio Brasileiro de Educação em Computação (pp. 16-17). SBC. doi: doi.org/10.5753/educomp_estendido.2021.14853*

A03 Oliveira, J. C., de Carvalho, L. S. G., de Oliveira, E. H. T., de Oliveira, D. B. F., **Pereira, F. D.** (2021, April). Análise de correlação entre habilidade de Resolução de Problemas e desempenho em disciplinas de programação. In Anais Estendidos do I Simpósio Brasileiro de Educação em Computação (pp. 43-44). SBC.

A04 Melo, R., Pessoa, M., Haydar, G., de Oliveira, D. B. F., de Oliveira, E. H. T., de Carvalho, L. S. G., **Pereira, F. D.** (2021, April). Um estudo sobre a relação entre os elementos de jogos e os tipos de usuários de sistemas gamificados. In Anais Estendidos do I Simpósio Brasileiro de Educação em Computação (pp. 35-36). SBC.

A05 da Rocha Braz, A. C., de Carvalho, L. S. G., de Oliveira, E. H. T., de Oliveira, D. B. F., **Pereira, F. D.**, Bittencourt, R. A., Santana, B. L. (2021, April). Validação e análise de um inventário de conceitos sobre programação introdutória. In Anais Estendidos do I Simpósio Brasileiro de Educação em Computação (pp. 27-28). SBC.

A06 Costa, T. L., de Oliveira, E. H. T., Passito, A., de Souza Pinto, M. A., de Carvalho, L. S. G., de Oliveira, D. B. F., **Pereira, F. D.** (2021, April). Material Didático Interativo para a disciplina de Introdução à Programação de Computadores. In Anais

Estendidos do I Simpósio Brasileiro de Educação em Computação (pp. 41-42). SBC.

Awards:

- Best paper Award on the Intelligent Systems for the Promotion of Teaching and Learning track - Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2019.
- Second best paper Award on the Intelligent and Adaptive Systems track - Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2020.
- Second best paper Award on the Technologies in Computer Education track - Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2020.
- Second best paper Award on the Educational Games and Innovative Technology for Education track - Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2020.
- Wiley award for **top cited article in 2020-2021**, related to the article "Using learning analytics in the Amazonas: understanding students' behaviour in introductory programming", published in British Journal of Educational Technology.
- Award for being the supervisor of the Best undergraduate thesis in the Thesis and Dissertation Contest Distinction - Brazilian Congress of Computer in Education (Congresso Brasileiro de Informática na Educação). 2020.
- Samsung Ph.D. Scholarship Award.

1.10 Thesis Guideline

In Chapter 2, we show how we collected the data used in this thesis. Additionally, we present how we extracted the features to create the students' programming profiles.

We also show our descriptive method that provides potential effective and ineffective programming behaviours that can be used as references for instructors or learners as a move towards improve learning. Moreover, we present how the early detection of effective and ineffective behaviours can be useful to guide students with ineffective behaviours. The publications related to this chapter are: J02 and C09.

In Chapter 3, we present how the features from the programming profile can be used in ML learning to achieve cutting-edge results for early performance prediction. Moreover, we demonstrate that the features can accurately be employed not only on the beginning of the course to predict the performance of students, but also during the rest of the course. We also point to potential applications and implications for early and non-early prediction. The publications related to this chapter are: J03, C01, C03, C04, and C06.

In Chapter4, we employ interpretable AI to inspect the decisions of our most accurate predictive method. To do so, we perform an analysis of non-linear relationship between the effective and ineffective students' behaviours and the final grade to interpret the features effects in the method's prediction individually and collectively. We also explore ways of using these results in potential applications to extend the OJ system. The publications related to this chapter are: J04, A02.

In the Chapter 5 we used the effective and ineffective behaviours from the programming profile to perform recommendation of problems for learners. To evaluate our recommendations, we compare our Behavioural-based Recommender System (BRS) with a Random Recommender System (RRS), which simulates typical human selection of problems in online judges. Such comparison was conducted through a double-blind control experiment to verify the impact on student's programming achievement, motivational affect and effort employed to solve the recommended problems. The publications related to this chapter are: C11, C20.

In the Chapter 6 we enhanced the recommender proposed in Chapter 6 to help instructors in selecting problems to create programming assignments and exams. The publications related to this chapter are: C16, C20, A01. Another article related to this chapter is being evaluated by the reviewers from the IEEE Transactions on Learning

Technologies (IEEE TLT). Moreover, for the topic prediction, we have also another paper being evaluated by the reviewers from the IEEE Access journal, and published as a pre-print (PEREIRA *et al.*, 2022).

For all the mentioned chapters, we also explore ways of using these results in potential applications to extend the typical OJ system in a move towards our goal, that is, to propose and validate a human/AI OJ architecture to support CS1 instructors. Notice that it is important to validate such architecture in the perspective of the instructors. Thus, in Chapter 7, we propose and validate concept designs based on our methods (presented in the previous chapters) to validate a hybrid human/AI architecture that can be used to extended the typical OJ architecture. The article related to this chapter is being evaluated by the reviews from the International Journal of Artificial Intelligence in Education (IJAIED).

Conclusions, limitations, future work, as well as the research next steps are presented in Chapter 8.

2

Programming Profile with effective and Ineffective Programming Behaviours

Learning is a treasure will
follow its owner everywhere.

- Chinese Proverb

2.1 Overview of the Chapter

As we show in the previous Chapter, tools for automatic grading programming assignments, also known as Programming Online Judges, have been widely used to support CS courses. Nevertheless, a limited number of studies have used these tools to acquire and analyse interaction data to better understand students' performance and behaviours, often due to data availability or inadequate granularity. To address this problem, in this study we use an Online Judge called CodeBench, which allows for fine-grained data collection of student interactions, at the level of, e.g., keystrokes, number of submissions, and grades. We deployed CodeBench for three years (2016-2018) and collected data from 2058 students from 16 introductory computer courses, on which we have carried out fine-grained descriptive, predictive, and prescriptive analy-

sis. In this Chapter, specifically, we aim at detecting effective/ineffective behaviours regarding learning CS1 concepts that can be used to represent a programming profile of the students. Results extract clear behavioural classes of CS1 students, significantly differentiated both semantically and statistically, enabling us to better explain how student behaviours during programming have influenced learning outcomes. Finally, we also identify behaviours that can guide novice students to improve their learning performance, which can be used for interventions. We believe this the findings from this Chapter is a step forward towards enhancing OJ systems and helping instructors and students improve their CS1 teaching/learning practices.

2.2 Practitioner Notes

What is already known about this topic:

- Studies suggest a high failure rate in CS1 courses (CARVALHO *et al.*, 2016; ROBINS, 2019; LUXTON-REILLY *et al.*, 2018).
- Learning to program takes a lot of practice and feedback is highly desirable (IHANTOLA *et al.*, 2015; LUXTON-REILLY *et al.*, 2018; CARVALHO *et al.*, 2016).
- Student activity in Online Judges can be used to predict their outcome (PEREIRA *et al.*, 2020a), but studies are few.
- Data for such studies is often proprietary or of inadequate granularity (IHANTOLA *et al.*, 2015; PEREIRA *et al.*, 2020a).

What this Chapter adds:

- Creating a *programming profile* using features collected from a new Online Judge system, CodeBench, which allows for fine-grained descriptive, predictive and prescriptive analysis of student behaviour for CS1.
- Employing descriptive and predictive analytics to identify early effective behaviours for novice students and, for the first time in our knowledge, how these behaviours can be useful for ineffective students.

- A novel classification of students into effective, average and ineffective, based on their behaviour, which shows both semantic and significant statistical differences.
- A clear indication that student behaviour during programming influences learning outcomes for CS1.
- A proposal, design, and implementation of a large scale, longitudinal study of student behaviour in CS1.

Implications for practice and/or policy:

- Students may need to reflect on their behaviour and self-regulate.
- Instructors may propose specific strategies and guidance based on early ineffective behaviours.
- Instructors have a powerful descriptive and predictive analytic method to understand student behaviour and patterns.
- Instructors can combine their knowledge with the descriptive and predictive analytic method to perform interventions.

2.3 Research Questions Addressed in this Chapter

To improve the way students learn to program and to tackle the high dropout rate, instructors from the Federal University of Amazonas, Brazil, have developed an Online Judge from scratch, called CodeBench, which automatically evaluates and feeds back on CS1 assignments. Such home-made system combines the large-scale approach of the popular MOOC formula with the flexibility given by an in-house system, allowing unprecedented research depth and amenability. Running since 2016, CodeBench has collected interaction data from 2058 non-CS major students, across six semesters, from 16 different classes every year. It collects fine granularity data, whilst students attempt to solve assignments and exams, such as keystrokes in the embedded Integrated Development Environment (IDE), submissions, etc.

In this work, we model this fine-grained data using descriptive and predictive analytics methods, to identify early effective programming behaviours and how they could be useful to guide ineffective students. We represent those programming behaviours in form of ML-features. *Those features together compose what we call the students' programming profile*.

Thus, as a step towards understanding CS1 student behaviours, our goal is to answer the following two research questions:

- **RQ1-1**: How can effective and ineffective behaviours of CS1 students be detected early, using data from an Online Judge system?
- **RQ1-2**: Which effective behaviours of novice students can be useful to guide students with ineffective behaviours and which ineffective behaviors should be avoided by learners?

2.4 Definition of effectiveness, ineffectiveness and resilience

In this work, we call "effective students" those who make progress in learning to program, typically leading to successful outcomes (ROBINS, 2019). Conversely, "ineffective students" are those who do not make progress or require excessive effort, typically leading to unsuccessful outcomes (ROBINS, 2019). Finally, we use the term "resilience" to refer to the students who struggle to edit and submit code more than the median of attempts, employing then more effort in problem-solving.

2.5 Effective and Ineffective Behaviours in Programming - State of the art

Despite the considerable number of studies about novice and expert learners from introductory programming classes, Robins (2019) explains that we know very little about effective and ineffective behaviours of novice students in CS1 classes, especially

how effective behaviours can be identified and whether they can be used to improve the learning process of the ineffective learners.

In this sense, Edwards *et al.* (2009) conducted an initial study, involving 1,101 students from three different computer science courses (CS1, CS2, and CS3). The authors inspected students' time and code-size data and found that students who achieve better grades start and finish assignments earlier than students with worse grades. In addition, successful novice programmers moderately wrote more program code. Although this study has analysed an extensive sample, they inspected just a few features (procrastination and changes in the codes). In our work, we employ hence a much larger set of 16 features (Table 2), as well as a larger number of students (Table 1) with a new method to detect early effective behaviours.

Data collected from Online Judges can be useful for formative assessment, prediction of student outcomes and early identification of students at risk (HERODOTOU *et al.*, 2019; CARTER *et al.*, 2019; QUILLE; BERGIN, 2019; OLIVEIRA *et al.*, 2021a). Recently, these kinds of analyses have been arousing the interest of researchers. For example, Ahadi *et al.* (2016), Otero *et al.* (2016), Dwan *et al.* (2017), Castro-Wunsch *et al.* (2017), Quille & Bergin (2019), Leeuwen *et al.* (2019) use code metrics, such as number of submissions, time spent programming, temporal patterns, number of syntactic errors, a.o., to estimate students outcomes, using varied machine learning and data mining techniques.

Still, Hosseini *et al.* (2014) performed a formative assessment of programming students by evaluating how 101 learners develop their code over time in order to explore problem solving paths. Rivers & Koedinger (2017) investigated a small dataset of 15 programming students collected from a what they called intelligent teaching assistant for programming with the aim of generating automatic hints. Both works are much smaller scale than ours; they support, like our work, a fine-grained data analysis, as crucial for a better understanding of programming behaviours.

Jadud (2006) proposed an algorithm called Error Quotient (EQ), which uses snapshots of compilation to quantify the student errors whilst they are programming. The EQ algorithm receives as input a pair of compilation events and assigns to them a

penalty, if both events ended with error. The penalty could vary, e.g., whether or not the error of both compilation events was the same. Watson *et al.* (2013) extended EQ with an algorithm to compute the Watwin Score (WS), which scores compilation pairs, by additionally considering the problem-solving time.

Considering descriptive and predictive analytics in the context of CS1 classes to model students' behaviour, studies have explored the compilation errors (JADUD, 2006; WATSON *et al.*, 2013; SANTOS *et al.*, 2020), how students deal with deadlines (EDWARDS *et al.*, 2009; SPACCO *et al.*, 2013; VIHAVAINEN, 2013; AUVINEN, 2015; CARTER *et al.*, 2019), how many attempts they need to solve the problems (AHADI *et al.*, 2016; CASTRO-WUNSCH *et al.*, 2017; ESTEY; COADY, 2016; FONSECA *et al.*, 2019), how they use hints in a web-based system (ESTEY; COADY, 2016; OLIVEIRA *et al.*, 2020), which are their frequencies of submission and unique attempts (MUNSON; ZITOVSKY, 2018), how much effort they put into code (static analysis) writing (OTERO *et al.*, 2016), and which is their typing pattern and keystroke latency when programming (LEINONEN *et al.*, 2016). All these studies analyse the relationship between students' code metrics and performance.

Here, instead, we start with all fine-grained code metrics (features) in CodeBench, proposed our self-devised ones, as a basis for selecting the best. Hence, one important contribution of this work is to show, through our case study, how important it is to measure, collect, analyse and report the fine-grained data proposed by the CodeBench, and how this helps stakeholders to have an early understanding of students' behaviours in programming classes.

2.6 Educational Context

At the Federal University of Amazonas, about 640 STEM undergraduate students enrol every year in CS1, distributed over sixteen classes, 11 held in the first term and 5 in the second. Figure 3 shows the evolution of pass rates from 2010 to 2018.

In public Brazilian universities, students must go through a new selection process if they want to move from one undergraduate program to another. Thus, the dropout

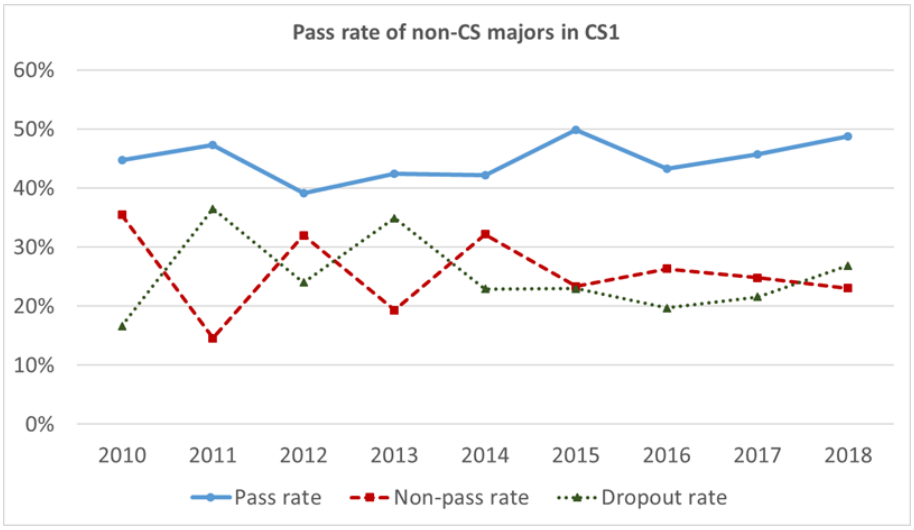

Figure 3 – Evolution of pass rate in CS1 classes.

rate of CS1 is partially explained by the dissatisfaction of some students with the program itself.

Since 2015, the content is divided into seven modules, each consisting of four two-hour sessions: (1) variables; (2) conditionals; (3) nested conditionals; (4) while-loops; (5) vectors; (6) for-loops; and (7) matrices. Each module follows this sequence of activities: a lecture class, two practical classes, and a partial exam. Python is adopted as the base programming language. Figure 4 illustrates , during the CS1 course, students in our environment typically solve 7 assignments lists, whereas which assignment precede a test on the same programming topic. Each list has an average of 10 questions, and the tests have 2 questions.

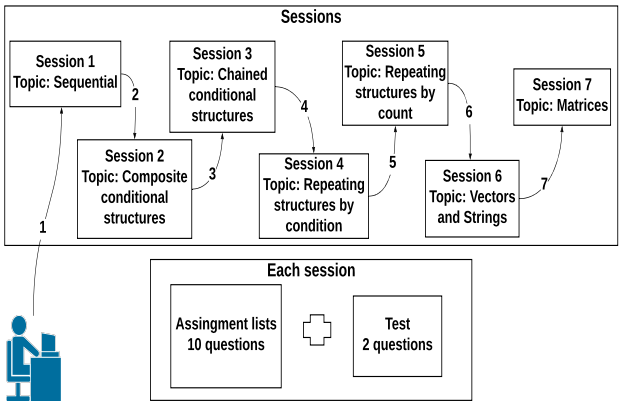

Figure 4 – CS1 course configuration

Since CS1 content gradually increases in complexity, the final grade is a weighted

average of the partial exams, where more advanced modules have higher weights. The final grade is determined based on 7 partial exams (summative assessments), 7 mandatory programming assignments (formative assessments), and one final exam. Partial exams contribute with increasing weights (6.1% to 18.2%) to the final grade, whereas all assignments have the same weight ($\approx 1.3\%$). For this study, we normalised the final grades in the range from 0 to 1.

Thus, we analyse here running the Introductory Programming (CS1) course at the Federal University of the Amazonas, via this self-designed OJ, which is delivered to 15 non-CS undergraduate degrees across the university. These courses are divided into 5 major areas: Mathematics, Physics, Engineering, Statistic and Geology. Three of the degrees belong to Mathematics, 2 to Physics, 8 to Engineering, 1 to Statistics and 1 to Geology. Figure 5 illustrates this configuration.

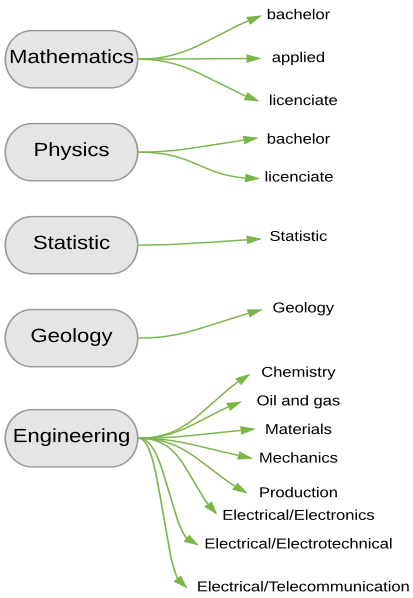

Figure 5 – Non-CS undergraduate courses at the Federal University of the Amazonas

2.7 Method

2.7.1 Instrument

In this work, we use as instrument the CodeBench Online Judge environment, which is self-designed and implemented, as it allows us the freedom to add the changes inspired by our research results. CodeBench[1] (Figure 6) is developed from scratch by one of the authors[2]. It evaluates students' codes using a "dynamic analysis approach", as defined by the taxonomy proposed by Ullah *et al.* (2018), i.e. students submit their program to the platform, and the output is compared against a set of test cases, manually provided by the instructor.

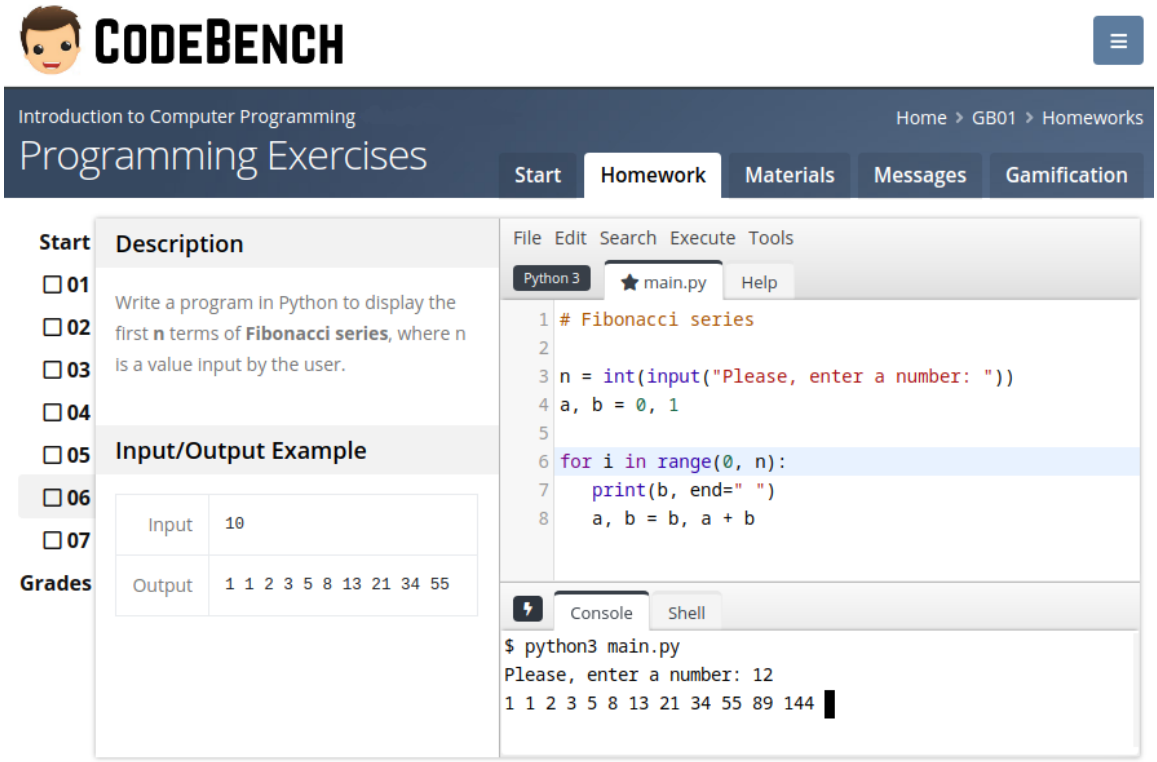

Figure 6 – Screenshot of the student interface of CodeBench, showing a programming assignment composed of 7 questions (left), the description of question #6 (middle), the built-in IDE (right), and the Python console (right bottom).

[1] codebench.icomp.ufam.edu.br

[2] The author who developed CodeBench is professor David Braga Fernandes de Oliveira, who is the co-supervisor of this work.

2.7.2 Data Collection

CodeBench provides its own code editor (IDE), designed to be simple and novice-friendly, as shown in Figure 7. All actions performed by students in the IDE (e.g., keystrokes, submissions, code pasting, mouse clicks, tab or window transitions, etc.) are timestamped and recorded in a log file on the server side. Figure 7 shows an example of the log, from where we can extract variables to measure learner behaviour.

```
1  27/5/2016@8:34:42:871:focus
2  27/5/2016@8:34:43:475:change:{"from":{"line":0,"ch":0},"to":{"line":0,"ch":0},"text":["p"],"removed":[""],"origin":"+input"}
3  27/5/2016@8:34:43:615:change:{"from":{"line":0,"ch":1},"to":{"line":0,"ch":1},"text":["r"],"removed":[""],"origin":"+input"}
4  27/5/2016@8:34:43:677:change:{"from":{"line":0,"ch":2},"to":{"line":0,"ch":2},"text":["i"],"removed":[""],"origin":"+input"}
5  27/5/2016@8:34:43:811:change:{"from":{"line":0,"ch":3},"to":{"line":0,"ch":3},"text":["n"],"removed":[""],"origin":"+input"}
6  27/5/2016@8:34:43:884:change:{"from":{"line":0,"ch":4},"to":{"line":0,"ch":4},"text":["t"],"removed":[""],"origin":"+input"}
7  27/5/2016@8:34:43:884:change:{"from":{"line":0,"ch":5},"to":{"line":0,"ch":5},"text":["()"],"removed":[""],"origin":"+input"}
```

Figure 7 – Logs collected from CodeBench when the learner was writing a *print* command.

We have analysed data from 2016-1 to 2018-2 (2016-1 means the first term of 2016), with emphasis on the first four weeks, since our goal is to detect early effective and ineffective behaviour. After gathering data from six academic terms (semesters), we had 2058 students in total, which we call consolidation data[3]. As it is difficult to compare data from different runs, we applied a z-score (z_i) to each feature (briefly explained in the next section), i.e., assigning zero to the mean ($\bar{x}$) and replacing values (x_i) with the amount of standard deviations ($\sigma(x_i)$) from this mean for any value other than the mean, as in equation 2.1.

$$z_i = \frac{x_i - \bar{x}}{\sigma(x)} \tag{2.1}$$

Table 1 presents descriptive statistics, differentiated by term, on the number of students and submission attempts for the programming assignments and exam exercises.

2.7.3 Features Extraction and Selection

To create the programming profiles, we extracted useful information from two sources in CodeBench: IDE raw logs and students' source codes. We defined which features

[3] Our dataset can be found on codebench.icomp.ufam.edu.br/dataset/

Table 1 – Number of instances in CodeBench's data set (by term)

	2016-1	2016-2	2017-1	2017-2	2018-1	2018-2	Total
Students	535	176	481	190	486	190	2058
Programming assignments	675	447	1278	556	1550	893	5399
Exam exercises	128	110	153	93	180	107	771
Submission attempts	154163	38933	119370	27613	148775	46765	535619

to be extracted, firstly based on previous work (PEREIRA *et al.*, 2020a), to which we added self-devised features, based on discussions with three of the instructors (who also authored this work). The features suggested by the instructors were mainly related to coding activity, time spent in the IDE and inappropriate use of copy&paste.

We computed several features, such as the proportion of copy&paste events, number of executions between submissions, number of deleted characters, program metrics (number of cycles, cyclomatic complexity, number of variables, number of comments, number of non-comment lines, etc.), among others. In order to reduce the feature space, we analysed the pairwise Spearman's rho correlation among all the features, since they were not normally distributed. Hinkle *et al.* (2003) claims that a correlation ≥ 0.9 (absolute value) is strong. As such, we removed features with correlation bellow -0.9 and above 0.9 with other features. In the case of a high correlation between a pair of features, we opted to remove the one with a lower correlation with the final grade, as we intend to find features related to effective and ineffective behaviour.

Table 2 describes each remaining feature along with its categorisation, according to a taxonomy of 'useful information' derivable from IDE data, proposed by Carter *et al.* (2019), where Count represents features that can be extracted by counting events in raw log files or source codes, Math represents features that need a mathematical formula to be computed, and Algo represents those features that need an algorithm applied in the raw data to extract useful information. *These features (programming behaviours) together compose what we call the students' programming profile*. In other words, the programming profile from CS1 students is a set of features that represent the learners' effective and ineffective programming behaviours. We also provide descriptive statistics of our

features in Appendix C.

Table 2 – Features (programming behaviour) used in the machine learning models

Features (Programming behaviour)	Description	Type
procrastination	Root square[4] of time in minutes between first code edit and programming assignment deadline (EDWARDS *et al.*, 2009; CARTER *et al.*, 2019) multiplied by -1 after the z-score transformation[5];	Count
amountOfChange	Amount of code added/changed between submissions (EDWARDS *et al.*, 2009; CARTER *et al.*, 2019);	Count
attempts	Average number of submission attempts (regardless whether correct or not) for each problem (CASTRO-WUNSCH *et al.*, 2017);	Math
lloc	Total number of logical lines for each submitted code (OTERO *et al.*, 2016). Imports, comments, and blank lines were not counted;	Math
systemAccess	Total number of student logins between the beginning and the end of the forth week;	Count
firstExamGrade	Student grades for the first exam taken at the end of the fourth week of the course;	Count
events[6]	Number of log lines on attempt to solve problems. To illustrate, each time the student presses a button in the embedded IDE of CodeBench, this event is stored as a line in a log file (adapted from (LEINONEN *et al.*, 2016; CASTRO-WUNSCH *et al.*, 2017))	Count
eventActivity	A binary self-devised feature, where 1 is assigned when the student solves a problem with less than an amount [7] of *events*[8]. To aggregate the feature, we calculate the probability of a student having a value 1;	Algo
correctness	Number of problems solved correctly from the programming assignment realised in the first four weeks (CASTRO-WUNSCH *et al.*, 2017);	Count
copyPaste	A self-devised feature, the proportion between pasted characters ('ctrl+V') and characters typed;	Math
syntaxError	A self-devised feature, ratio between the number of submissions with syntax error[9] and the number of attempts (ESTEY; COADY, 2016);	Math
ideUsage	Total time spent, in minutes, by the students solving problems in the embedded IDE (counted only when students were typing - we removed downtime);	Algo
keystrokeLatency	Keystroke average latency of the students when typing in the embedded IDE (we also removed downtime) - adapted from (LEINONEN *et al.*, 2016);	Algo
errorQuotient	Compute a score based on the number of code errors and repeated errors (JADUD, 2006);	Algo
watWinScore	An extension of errorQuotient taking into account the problem solving time (WATSON *et al.*, 2013) (more explanation in related works - Section 2.5);	Algo
countVar	Total number of variables in the source codes (CARTER *et al.*, 2019);	Count

Figure 8 presents pairwise Spearman's rho correlations among the features in Table 2. It shows that watWinScore, procrastination, copyPaste, syntaxError, and errorQuotient are positively correlated. However, they are negatively correlated with other features: amountOfChange, attempts, lloc, systemAccess, firstExamGrade, events,

[4] We apply square root to smooth the data, since time is measured in minutes.

[5] As an example, after the transformation, a z-score = 2 for procrastination means that the student solved the problem with an antecedence of 2 standard deviation below the mean. Thus, a value above zero would mean low procrastination, which is not reasonable in terms of semantics. As such, we addressed it by multiplying the feature value by -1 after the transformation and, after that, a high value means high procrastination and low value means low procrastination.

[6] (CASTRO-WUNSCH *et al.*, 2017) called this feature 'number of steps'. However, unlike them, we averaged it.

[7] One standard deviation minus the median of the numbers of *logRows/events* for a problem

[8] Castro-Wunsch et al. (2017) called this feature 'number of steps'. However, unlike them, we averaged it.

[9] SyntaxError is a common and generic exception in python.

correctness, ideUsage, eventActivity, and keystrokeLatency. This suggests that high values of features of the first group might correspond to negative/undesirable behaviour (e.g., high syntaxError is an undesirable behaviour, which should be improved). This observation is further used in the clustering analysis.

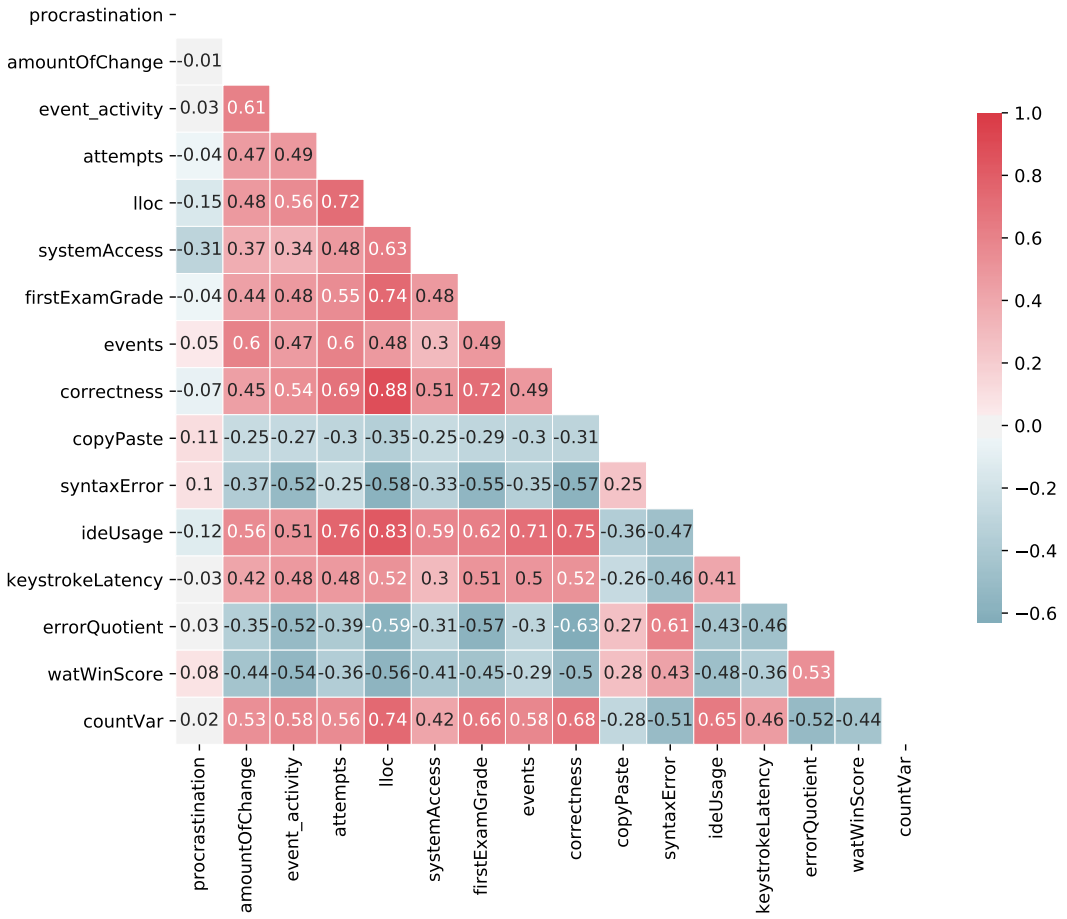

Figure 8 – Pairwise Spearman's rho correlation between remained features.

2.7.4 Evaluative Factors

As effectiveness and ineffectiveness are psychological constructs, we derived a set of evaluative factors to perform an operational definition of these concepts. For each student, we determined the number of solved and unsolved questions, considering the

number of attempts of code submission to CodeBench correction. At the same time, for each programming question, we calculated the median of the number of attempts. We state here that students made few attempts if they made less than the median of attempts, and many attempts, otherwise. Moreover, if students perform many attempts, they are called resilient. From these assumptions, we defined five evaluative factors to measure effective or ineffective behaviours, as follows, which were inspired in Yera & Martínez (2017):

- Non-attempt ratio: $noAttempts = \frac{\#non-attempted-questions}{\#questions}$
- Unsuccessful without resilience ratio: $unsucNoRes = \frac{\#unsolved-questions-after-few-attempts}{\#questions}$
- Successful without resilience ratio: $sucNoRes = \frac{\#solved-questions-after-few-attempts}{\#questions}$
- Unsuccessful with resilience ratio: $unsucRes = \frac{\#unsolved-questions-after-many-attempts}{\#questions}$
- Successful with resilience ratio: $sucRes = \frac{\#solved-questions-after-many-attempts}{\#questions}$

Note that the sum of these five factors is equal to 1. In addition, we defined another two evaluative factors, based on how successfully learners solve programming assignments, as follows:

- Effective attempt ratio: $effectAttRate = \frac{\#solved-questions}{\#attempted-questions}$
- Effective general ratio: $effectGenRate = \frac{\#solved-questions}{\#questions}$

Here we hypothesised that an effective behaviour is related to low values of noAttempt and unsucNoRes. Similarly, it is related to high values of sucNoRes, sucRes, effectAttRate, effectGenRate. On the other hand, ineffectiveness is the opposite. To check if our hypotheses are reasonable, we measured the correlation of the evaluative factors with the final grades. Figure 9 shows the pairwise Spearman rho correlation among evaluative factors and final grade.

Regarding unsucRes, a high value may look like a sign of ineffectiveness. When students try a lot, even not achieving a correct solution, they show resilience, which is an important characteristic for programmers (Pereira et al., 2019c). In addition, Online Judges based on dynamic analysis have limitations and sometimes they may be unfair,

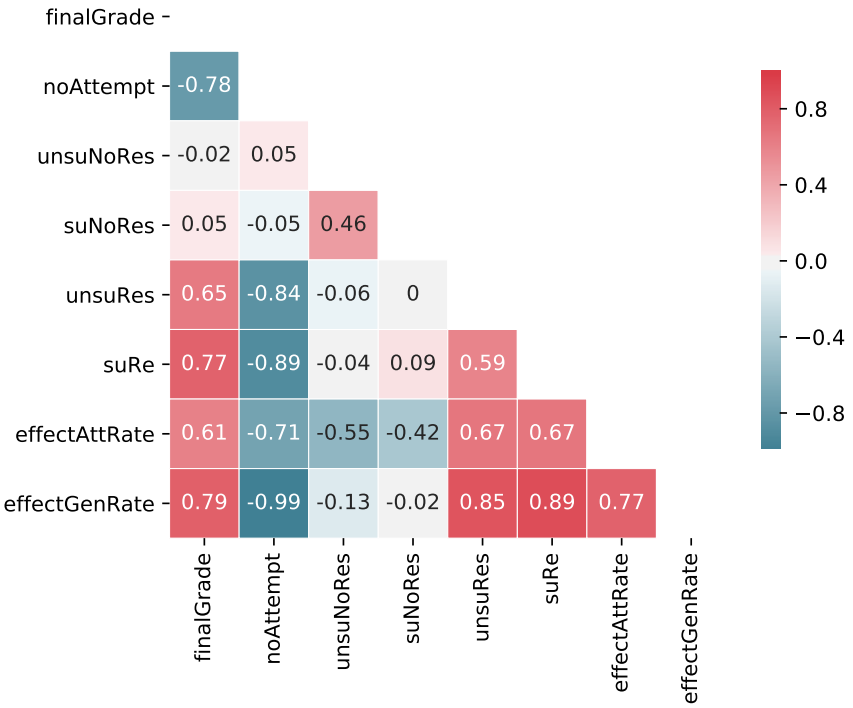

Figure 9 – Pairwise Spearman's rho correlation between evaluative factors.

since they perform a string comparison between student solution output and the expected output. Consequently, if the student misses a space or a break line in the output (presentation error), the solution will be considered as wrong, even if it is logically correct. Thus, to confirm this assumption, we have analysed the correlation between unsucRes and finalGrade. We found a positive value ($r_s = .65$) and, hence, we state that a high unsucRes is related to effectiveness.

Notice that sucNoRes has a weak positive correlation with finalGrade ($r_s = 0.05$). Although this outcome also sounds unexpected, Ahadi *et al.* (2016) showed that there are cases in which solving correctly the programming problem is irrelevant, provided that the learner achieves a threshold of attempts (resilience). Still, there are also problems that are important to solve correctly (success), but students are expected to have done so with more than a specific number of attempts (with resilience). Moreover, at the beginning of the course, students are learning how to deal with the nuances of CodeBench and it is common to make naive mistakes (as previously explained), provoking an increase on

the number of attempts even for students who end up passing.

Thus, different from our prior assumption, effectiveness and ineffectiveness without resilience are not correlated with the final grade (Figure 6). Hence, we kept as evaluative factors only the features with Spearman's rho correlation with the final grade above .6, which were: noAttempt, unsucRes, sucRes, effectAttRate, effectGenRate, and obviously the finalGrade.

2.7.5 Clustering and Association Rule algorithms

Clustering algorithms can uncover hidden patterns in a complex dataset and many works (ANTONENKO *et al.*, 2012; SHI; CRISTEA, 2018; SHI *et al.*, 2020; DUTT *et al.*, 2015) used these unsupervised learning methods to analyse new relationships on educational data. Still, students' behaviour is heterogenous and, as effective and ineffective programming indicators need to be found, these would be expected to have different values for different student subpopulations. Hence, we cluster students based on the students' logs, in order to inspect the patterns of programming behaviours in each student cluster and how these behaviours reflect on the evaluative factors. To do so, we used the well-known k-means algorithm (MACQUEEN *et al.*, 1967). This algorithm clusters n observations within a predetermined number of k clusters, where each observation belongs to the nearest group mean. Thus, each observation is closer to its own cluster centroid[10]. Henceforth, we have used the mean silhouette coefficient (ROUSSEEUW, 1987) of observations to choose the most appropriate number of clusters for our data, as this method can be applied to analyse the distance between every pair of clusters. We opted for k-means after empirically trying other techniques such as Gaussian Mixture Models, Spectral clustering, and DBSCAN. Indeed, k-means performed a better separation in subgroups (taking in consideration the mean silhouette coefficient) at the same that has polynomial smoothed running time.

Furthermore, to strengthen and validate our conclusions and triangulate results, Association Rule Mining (ARM) is used to identify groups within a given dataset, based

[10] Represented by the mean of the observations within the cluster

on the support (frequency) of items (AGRAWAL *et al.*, 1993). The effectiveness of ARM can be evaluated through different measures; in this work, we opted to use confidence and lift, since those are some of the most used in the literature (HUANG *et al.*, 2017).

2.8 Results and Discussion

2.8.1 Analysing Consolidation Data

To tackle the research questions, we modelled students' programming behaviour using the features presented in Table 2. These served as observations of the k-means clustering method. From the complete three-year data set, we have used data from the first four weeks[11] of the course, as we aimed to detect effective behaviour early on. Previous studies support that it is possible to draw patterns using fine-grained data from the first weeks of introductory programming courses (ESTEY; COADY, 2016; CASTRO-WUNSCH *et al.*, 2017; COSTA *et al.*, 2017; MUNSON; ZITOVSKY, 2018; ROMERO; VENTURA, 2019; BOCKMON *et al.*, 2020; PEREIRA *et al.*, 2020a).

We inspected the relationship between the k-means clusters and the effectiveness or ineffectiveness (based on evaluative factors). The convergence of k-means was achieved in the 10th iteration with k=3 as the best value, and 44.94% of the students were assigned to Cluster A, 29.93% to Cluster B, and 25.12% to Cluster C. We applied the non-parametric Mann-Whitney U test for pairwise cluster comparison of features, to inspect the impact of each feature, individually, in the cluster formation. The results indicate a statistical difference (even with Bonferroni correction: $p \ll 0.05/3$) between features in different clusters, except for the errorQuotient between Clusters A and B. We also observed high effect sizes[12] ($r > 0.5$).

Furthermore, in terms of evaluative factors, Figure 10 shows that Cluster A performs better than Cluster B, which performs better than Cluster C, and, by transitivity, Cluster A performs better than Cluster C. Indeed, this pattern is kept in terms of all

[11] We performed the clustering using data from the first two, four, and six weeks. Results in the fourth week were significantly better (better effect sizes) than in the first two weeks. However, the difference between the fourth and sixth weeks was not significant, thus we opted for the fourth.

[12] Mangiafico (2016) states that r in between 0.10 and 0.30 is considered small, whilst r greater than 0.30 and lower than 0.50 is considered medium, and r > 0.5 is a large effect.

evaluative factors presented in the methods section. To check the statistical significance between these evaluative factors, we conducted a pairwise comparison between each cluster, as shown in Table 3. Apart from unsucRes for Cluster A vs Cluster B only, Cluster A outperforms Cluster B, which outperforms Cluster C for all evaluative factors, even with Bonferroni correction ($p \ll 0.05/3$).

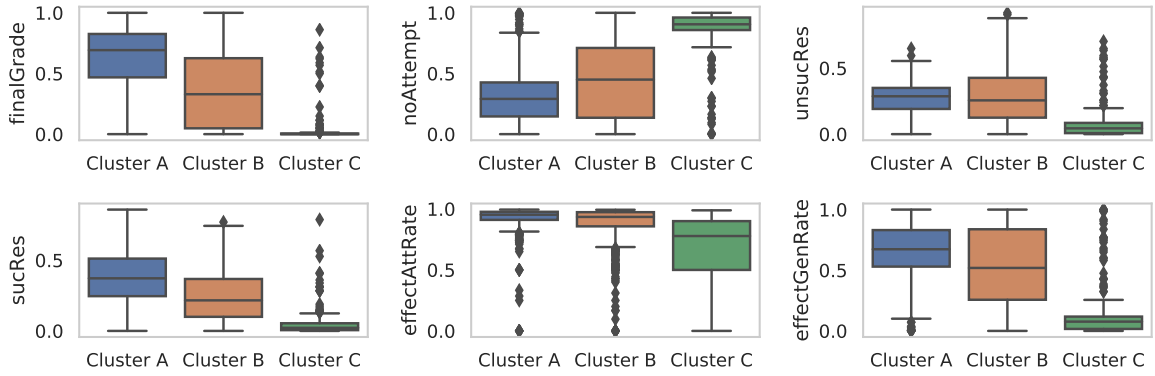

Figure 10 – Evaluative factors distribution of each cluster.

Based on Figure 10 and pairwise Mann-Whitney tests of the evaluative factors (Table 3), we can label each cluster, as follows: (i) Cluster A contains the effective students, (ii) Cluster B comprises the average students, and (iii) Cluster C has the ineffective ones. Moreover, Figure 11 presents the centroids of each cluster, showing the general programming behaviour of each group. From Figure 8, we can see which early programming behaviours (represented by the k-means centroids) are leading factors for effectiveness or ineffectiveness by adopted clusters.

2.8.2 Comparing Student Clusters

First, comparing effective with ineffective students, we can observe that the effective ones deal with errors better, since they have a lower errorQuotient, and they take less time to correct mistakes, as they have a lower watWinScore and, hence, they tend to not get stuck too much time with, for example, the same errors. Besides, effective learners have a higher amountOfChange between submissions, which might explain why these learners can fix errors faster. Coupled with that, ineffective learners are more affected with syntaxError, a generic and recurring exception in Python that may be

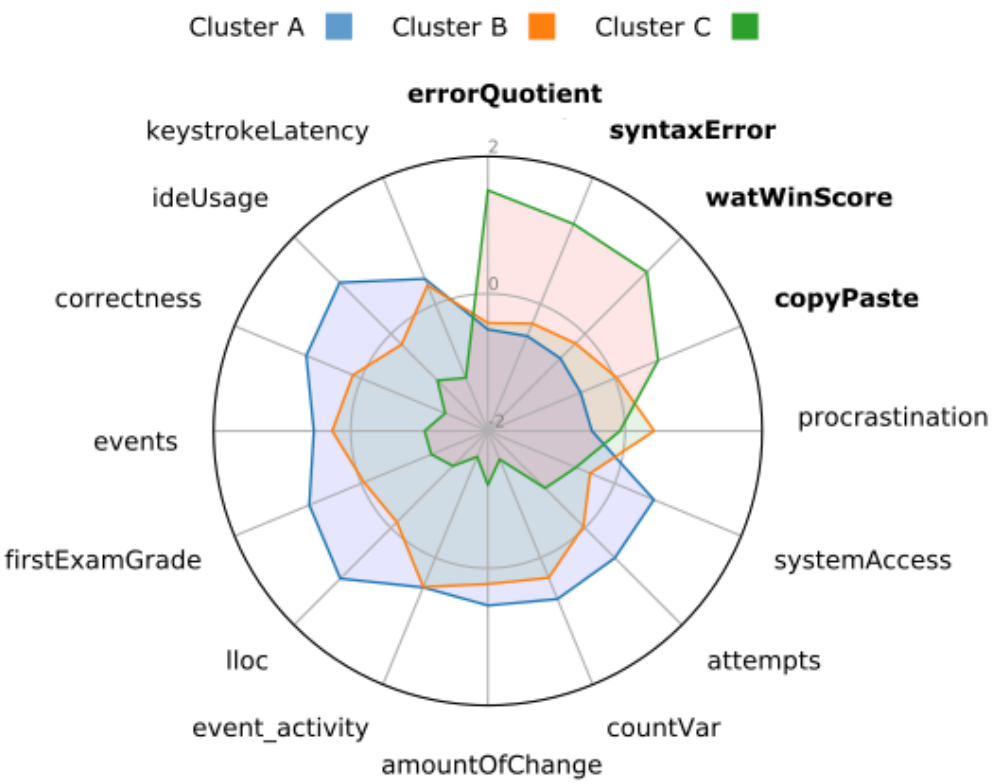

Figure 11 – Programming profile of students in each cluster. We have marked with bold and grouped together the features with a reverse scale for a better visualisation.

difficult to fix for CS1 students. With that in mind, instructors might be notified, or even the learners can be made aware of the risk of having many erroneous submission pairs, especially if the error is the generic Python SyntaxError. Moreover, a clear sign of concern is represented by a student with a huge difference in timestamps between a pair of submissions with errors (mainly with the same errors), coupled with a small amount of change.

Equally important, effective students have more attempts, lloc, systemAccess, events, ideUsage, and countVar, as they spend more time programming, and they submit more problems to the Online Judge. This clearly shows that these learners are more engaged with the course. Furthermore, effective students tend to code faster (higher keystrokeLatency). They also solve more problems (higher correctness) which is one possible reason why they achieve a better grade in the first exam (higher firstExam-

Grade). Finally, these students tend to manage their time better, as they procrastinate less. Still, when they solve problems, they do so with more events. If students solve a problem with too few events (lower eventActivity) they are copying and pasting code already created, which may or may not have been developed by themselves. In this sense, we can also see from Figure 11 that ineffective students tend to copy and paste more, which is not a desirable behaviour for introductory students at the very beginning of the course (first four weeks in our case).

To compare the average students with the two other groups, we should consider the effect sizes (COHEN, 2013) from Tables 3. We can see for all features a medium to large ($r > 0.4$) degree to which a sample from Cluster A (effective students) has stochastic dominance compared with the other sample from Cluster C (ineffective students). However, we can see a lower degree ($r < 0.3$) to which the null hypothesis (sample from the same group) is false comparing Cluster A versus Cluster B and comparing Cluster B versus Cluster C, which shows that Cluster B has nuances from both groups, i.e., students from Cluster B might have effective and ineffective behaviours.

Table 3 – Pairwise cluster comparisons of evaluative factors.

		finalGrade	noAttempt	unsuRes	suRe	effectAttRate	effectGenRate
Cluster A	W	427702.5	335088.5	523403.0	426444.0	490898.5	478139.5
vs	Z	-13.7420	-6.5230	-0.1230	-13.9100	-4.7460	-6.5610
Cluster B	Effect size (r)	0.1405	0.0317	0.0000	0.1440	0.0168	0.0320
	Asy. Sig. (2)	0.0000	0.0000	0.9020	0.0000	0.0000	0.0000
Cluster A	W	55826.5	161972.5	53377.0	49302.0	68775.5	49876.5
vs	Z	-22.5410	-24.0780	-23.0630	-24.2020	-18.7570	-24.0350
Cluster C	Effect size (r)	0.5807	0.6626	0.6079	0.6694	0.4021	0.6602
	Asy. Sig. (2)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Cluster B	W	427702.5	335088.5	523403.0	426444.0	490898.5	478139.5
vs	Z	-13.7420	-6.5230	-0.1230	-13.9100	-4.7460	-6.5610
Cluster C	Effect size (r)	0.1405	0.0317	0.0000	0.1440	0.0168	0.0320
	Asy. Sig. (2)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

2.8.3 Association rules analysis

A total of 83 rules[13] were found (0.1 < support < 0.6, confidence > 0.1 and lift = 6.0850). Figure 12 presents the stronger rules, sorted by their confidence (strongest rules have a higher contrast). The figure shows how different clusters (A, B, C) are more influenced by some features than by others.

[13] The full set of association rules can be accessed on shorturl.at/czEOX

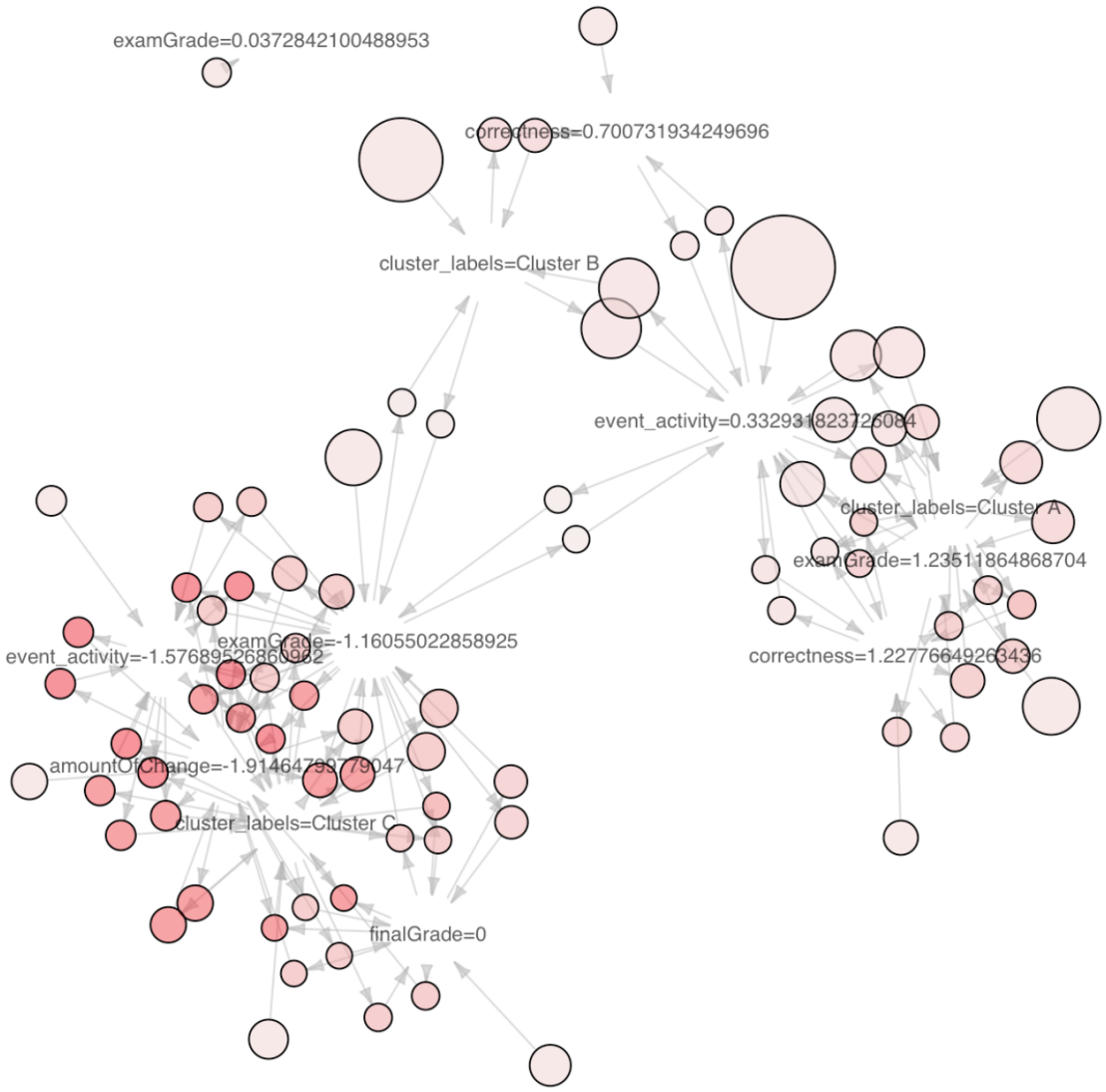

Figure 12 – Association Rules representation.

In general, the rules confirmed what we found for effective and ineffective behaviour using clustering, e.g., the amountOfChange, eventActivity and examGrade are highly associated (confidence >= 0.8, lift > 3) with the behaviour of non-passing students. As a new finding, we observed that low eventActivity with a small amountOfChange is related to ineffective students (lift = 6.1, confidence = 0.8, rule 55). On the other hand, effective learners have solved more problems (higher correctness) with higher eventActivity (lift = 3.59, confidence = 0.56, rule 64). By finding these new associations, alongside the similarities within our clusters, we can infer that these rules might be useful (due to the high lift) *as predictors to identify at-risk students*.

Finally, knowing which behaviour can be effective for CS1 learners may help

towards self-regulation and awareness of what kind of attitude can be dangerous and detrimental to their performance. An easily implementable option is to share this information as a report to instructors, to provide them with tools to motivate students. For instance, we have shown that resilient students (higher sucRes and unsucRes – Cluster A), who do not easily give up trying to solve a problem, usually perform better and end up passing. Instructors can then encourage weaker students to be resilient, as a path towards better performance. However, resilience alone is not enough. There are more factors involved in effectiveness, such as knowing how to fix, analyse or debug compilation errors, managing well their time to solve the assignments, and practising to develop skills - such as a higher accuracy of problem-solving, and so forth.

2.9 Pedagogical Implications

The origins of Online Judges trace back to programming competitions, whose intent was to test programming ability, instead of building it (KURNIA *et al.*, 2001; WASIK *et al.*, 2018). More recently, Online Judges are used as a self-learning tool, in parallel to regular courses (WASIK *et al.*, 2018; IHANTOLA *et al.*, 2015; PEREIRA *et al.*, 2020a). Thus, this work impacts on a wide variety of stakeholders, such as developers, teachers, students. Programming is a hands-on activity, which, however, requires a great amount of feedback. Such feedback is precious in educational terms, and early feedback is vital, if behavioural changes are desired. However, it is not scalable to large class sizes and growing number of students, as in the case studied here, of the Amazonas. Instead, with an approach such as ours, effective and ineffective behaviours in introductory programming can be automatically identified early on, and encouraged or discouraged, respectively. On the other hand, some measured behaviours may not be straightforward or wise to automatically action upon. In such situations, instructors can receive alerts of which students are at risk of low performance and why, so they can reflect on possible causes of the observed ineffective behaviours. For example, explicitly inefficient student behaviours (such as procrastination, copying/pasting of big chunks of code, and low IDE usage time) can be automatically tracked by an Online Judge. Students could

themselves be directly alerted, as a first port of call, on how far their behaviour is from the higher-performance students within their group, as well as in average, and as a result be invited to change their study strategy.

As another example, copy&paste behaviour may be due to plagiarism, or simply due to writing code separately and only pasting it in the system when “ready". Nevertheless, writing code directly in the system is very informative in the analysis and can lead to much more refined feedback to students. Thus some initial notification on undesired behaviour can be useful for students to be given a chance to change their own behaviour to better reflect their knowledge status. However, notice that such intervention (the alert, in this case) should be mediated by the instructors, as this is a premise of our work. Indeed, instructors can combine the information provided from this clusters to reinforce, offline, the need for changing study behaviour.

Other behaviours, such as a high keystroke latency and a small amount of change in code may be an observed consequence of non-observed actions. In this case, this information should be reported only to instructors, who, in turn, should plan activities in order to address the cause of such behaviours. For example, they can assign extra exercises that target debugging, misconceptions, or code patterns.

Furthermore, identified effective behaviours can be brought to the attention of instructors, effective, and also ineffective students – with instructors mediation and care about non-disclosure of personal information. For example, late students can be warned when a certain number of students complete assignments or spend more time coding in the IDE than they do. Here, again, group membership can inform the feedback, so that low-performance students have the opportunity to compare themselves with the best amongst their group, as opposed to the best in class, which may be nonviable for them to “beat". Additionally, the Online Judge can notify instructors about which students need extra help, in good time before any deadlines, allowing for proactive instead of reactive pedagogical interventions.

Briefly, we believe that our descriptive method provided in this chapter through our clustering and associate rules analysis can be used as reference values for instructors and in some cases for students. Such reference values (effective, ineffective, average (or

intermediate) - see Figure 11) can be provided to instructors (or even other stakeholders such as administrators), who in turn might make them available to students. Moreover, we performed the analysis only on early data, however the method can be generalised for the other sessions. Such analysis during the course can also be helpful for instructors and students. Indeed, a bit out of the scope of this Chapter, we also carry out experiments using non-early data. We observed that the hidden patterns found though the clustering descriptive method become even more explicit throughout the sessions (S1 - S7). The results can be seen in the Appendix B.

Finally, it is worth noting that, whilst correlation or even association rules per se do not imply causation, the two-pronged triangulation approach used is providing more evidence towards prediction power. Moreover, undesirable behaviours may need to be addressed in some cases, even if they may not directly cause ineffectiveness – as in the example of the copy&paste case.

2.10 Chapter Conclusions

CS1 classes usually have high heterogeneity among students. This was clear in this work, showing variations in the patterns of student behaviours, resulting in three different clusters. We further supported the findings via statistical differences in learning outcomes, programming behaviour and evaluative factors between these clusters. Importantly, our analyses showed which early (based on only the first four weeks) programming behaviours potentially indicate effectiveness or ineffectiveness in learning. We believe that these programming effective, ineffective, and average/intermediate behaviours can be used as reference values for instructors, who in turn can decide which reference value should be available to the students.

Thus, this result can support decision making of students as well as instructor intervention, such as designing specific guidance for a struggling group of students, proposing new and challenging exercises for effective students, and personalising exercises, according to different student needs. Furthermore, knowing which behaviour can be effective might help students to improve their self-regulation and awareness of what

kind of programming behaviour can be dangerous or beneficial to their performance. In the next Chapter, we will show how the programming profile used in this Chapter can be also useful for early performance prediction, thus giving another move towards human/AI hybrid systems in which the instructors will have early information about the final grade and situation (passed or failed) of each student. Also, we will go deeper into the analysis of the effective and ineffective behaviours by interpreting the predictive model since it is critical for adoption and prescription. Notice that the analysis of how the instructors might help the AI is provided only on Chapter 7, when we integrate the our AI methods.

3

EARLY PERFORMANCE PREDICTION

The journey of a thousand miles
begins with one step.

- Lao Tsu.

3.1 Overview of the Chapter

We showed through the use of unsupervised machine learning algorithms (clustering and association rules analysis) a clear indication that early student behaviour during programming influences learning outcomes for programming classes. In this Chapter, we will demonstrate thought supervised machine learning algorithms that these early behaviours can also be employed as features to predict the students' performance in an early stage of the course.

As previous explained, introductory programming may be complex for many students. Moreover, there is a high failure and dropout rate in these courses. A potential way to tackle this problem is to predict student performance at an early stage, as it also facilitates human/AI collaboration towards prescriptive analytics, where the instructors/monitors will be told how to intervene and support students - where early intervention is crucial. However, the literature states that there is no reliable predictor yet for programming students' performance, since even large-scale analysis of multiple features have resulted in only limited predictive power. Notice that Deep Learning (DL)

can provide high-quality results for huge amount of data and complex problems. In this sense, we employed DL for early prediction of students' performance using data collected in the very first two weeks[1] from introductory programming courses offered for a total of 2058 students during 6 semesters[2]. We compared our results with the state-of-the-art, an Evolutionary Algorithm (EA) that automatic creates and optimises shallow machine learning pipelines. Our DL model achieved an average accuracy of 82.5%, which is statistically superior to the model constructed and optimised by the EA ($p - value \ll 0.05$ even with Bonferroni correction). In addition, we also adapted the DL model in a stacking ensemble for continuous prediction purposes. As a result, our regression model explained 62% of the final grade variance. In closing, we also provide results on the interpretation of our regression model as a first step to understand the leading factors of success and failure in introductory programming, demonstrating in another perspective early effective and ineffective behaviours.

3.1.1 Practitioner Notes

What is already known about this topic:

- Researchers have argued that early performance prediction is vital (IHANTOLA *et al.*, 2015; HELLAS *et al.*, 2018; PEREIRA *et al.*, 2020a; QUILLE; BERGIN, 2019).
- One potential way to perform such early prediction is by extracting useful information from students' log-data and use this information as features in machine-learning algorithms (CARTER *et al.*, 2019; PEREIRA *et al.*, 2020a).
- There is no reliable early predictor of programming student performance (ROBINS, 2019).
- Besides the prediction, it is also important to interpret the predictive model to facilitate adoption (PEREIRA *et al.*, 2021; BERENDT *et al.*, 2020; TENÓRIO *et al.*, 2021).

[1] We opt to use data from the first 2 weeks instead of 4 weeks since there are other works which use this minimum period to construct a predictive model capable of predicting the students' outcomes at the end of the course.

[2] detail of the data is presented in the previous chapter

- Deep learning and optimisation of machine learning pipelines using evolutionary algorithms are achieving state-of-the-art results for other prediction tasks (OLSON *et al.*, 2016; GÉRON, 2019).

What this Chapter adds:

- A new combination of features, compared to the state-of-the-art, in order to explore potential avenues for enhancing students' performance.
- Showing how a deep learning pipeline can surpass state-of-the-art evolutionary algorithm for construction and optimisation of shallow machine learning models.
- Cutting-edge classification performance for early performance prediction using a large scale, longitudinal data from introductory programming students.
- Going one step further than binary classification and constructing an interpretable stacking method that combines deep learning and easily explainable regularised linear regression model.
- Showing in another perspective (by a regression model) how student behaviour during programming can be effective and ineffective, leading students to success and failure, respectively.

Implications for practice and/or policy:

- With early prediction, in a standard course, teachers could provide extra assignments for the high-achieving group and personalised support to those who are struggling.
- effective and ineffective behaviours can be automatically identified early on, and encouraged or discouraged, respectively.
- Such process of early intervention can be performed using dashboards, e-mails, etc. In other words, as 'prevention is better than a cure', likewise, it is better to prevent students from failure as soon as possible, instead of finding out students are struggling when their poor marks come in.

- Applying a linear regression model is a step towards Interpretable Machine Learning by understanding effective and ineffective programming behaviours, which facilitates the adoption of early interventions thought learning analytics tools to monitor the students' performance.
- Predicting student performance at an early stage might facilitates human/AI collaboration towards prescriptive analytics.

3.2 Research Questions Addressed in this Chapter

Many works have tried to find ways to mitigate the problem of high failure rate (IHANTOLA *et al.*, 2015; DWAN *et al.*, 2017; HELLAS *et al.*, 2018; LUXTON-REILLY *et al.*, 2018; ROBINS, 2019; FONSECA *et al.*, 2019) in programming courses. In this sense, researchers (Abu Amra; Maghari, 2017; COSTA *et al.*, 2017; AGUIAR; PEREIRA, 2018; HELLAS *et al.*, 2018; QUILLE; BERGIN, 2018; PEREIRA *et al.*, 2019a; PEREIRA *et al.*, 2019b; ROMERO; VENTURA, 2019; PEREIRA *et al.*, 2020a) have argued that early performance prediction is vital. This can facilitate human–AI collaboration towards prescriptive analytics, where the instructors/monitors will be told how to intervene and support students - where early intervention is crucial (HELLAS *et al.*, 2018; ALAMRI *et al.*, 2019; ROMERO; VENTURA, 2019; PEREIRA *et al.*, 2020). Furthermore, if the predictive model is interpretable, it can allow understanding the model's decisions (MOLNAR, 2020), for a better analysis of which factors can lead to success or failure of learners (PEREIRA *et al.*, 2020a). With such understanding, we can offer valuable information to instructors and learners. However, Robins (2019) explains that currently there is no reliable predictor yet for programming students' performance, since even large-scale analysis of multiple features have resulted in only limited predictive power. Still, Ihantola *et al.* (2015), Quille & Bergin (2018), Hellas *et al.* (2018), Robins (2019) state there is a need to advance further in this field.

One potential way to perform such early prediction is by extracting useful information from students' log-data and use this information as features in machine-learning algorithms (CARTER *et al.*, 2019). To illustrate, there are studies which perform predic-

tion using log-data, to represent how students deal with errors (JADUD, 2006; WATSON *et al.*, 2013), deadlines (CARTER *et al.*, 2019; EDWARDS *et al.*, 2009), attempts and correctness (CASTRO-WUNSCH *et al.*, 2017; AHADI *et al.*, 2016; ESTEY; COADY, 2016; FONSECA *et al.*, 2019; PEREIRA *et al.*, 2020b); static analysis of code (OTERO *et al.*, 2016), typing patterns (LEINONEN *et al.*, 2016), etc. In this sense, after demonstrating in the previous Chapter a clear indication that student behaviour during programming influences learning outcomes for programming classes, here we compiled these set of predictive factors (described in Chapter 2) based on the recent literature to be used as features in supervised Machine Learning (ML) models, to predict students' performance at the very beginning of introductory programming courses. We collected log-data from a home-made online judge, and employed a method using an evolutionary algorithm, to build and automatically optimise the machine learning pipeline, i.e., without the need of an expert in data science. The evolutionary algorithm explored many combinations of feature selection techniques, machine learning algorithms, combination of hyperparameters and their tuning, to the best ones thus representing a quite competitive technique.

Despite that, the evolutionary algorithm does not explore any deep learning technique for the prediction. However, many researchers (HINTON *et al.*, 2012; KINGMA; BA, 2014; SRIVASTAVA *et al.*, 2014; GÉRON, 2019; ALJOHANI *et al.*, 2020) have shown that deep learning can provide high-quality results for huge amount of data and complex problems. With this in mind, it is worth noting that early performance prediction is a complex problem (ROBINS, 2019) and we have a reasonable amount of data (N = 2058) (HERNÁNDEZ-BLANCO *et al.*, 2019) to allow for deep learning. Thus, we raised our first research question of this Chapter: *(RQ2-1) Would a deep learning model surpass an state-of-the-art evolutionary algorithm for early prediction of students' performance?*

Moreover, many works (ESTEY; COADY, 2016; COSTA *et al.*, 2017; CASTRO-WUNSCH *et al.*, 2017; MUNSON; ZITOVSKY, 2018; PEREIRA *et al.*, 2020a) defined the problem of prediction as binary classification, in which a student must earn a final grade of at least 5 (on a scale 0 – 10) in order to pass the course. Yadin (2013), Elarde (2016), Robins (2019) used the term *bimodal outcomes* related to programming students' grades,

which means that, in general, programming classes have mainly two groups: high achievers and failure students. For this, a binary classification make sense, reflecting the bimodal nature of programming students' outcomes. Nonetheless, this approach does not consider properly students in the mid-range. As such, we raised a second research question of this Chapter, in order to go a step further than the binary classification: *(RQ2-2) How we can effectively use the same data as for student result classification in a regression model, to obtain early prediction of the students' actual final grades?*

Additionally, for prescriptive analysis, education applications must go a step further than prediction itself and analyse the relevance of features (programming behaviours), with the aim of understanding the major factors that can lead to success or failure for a given cohort (HELLAS *et al.*, 2018; ROBINS, 2019; QUILLE; BERGIN, 2019; PEREIRA *et al.*, 2020a). In other words, a better understanding of what programming behaviours might negatively or positively influence the students' grades could lead to a better analysis of which strategies we might use to teach and how our students would like to learn. With a different perspective than when we are addressing RQ1-1 and RQ1-2 (Chapter 2), we also want to interpret the regression model used to address RQ2-2 and, thus, detect effective and ineffective behaviors (factors that can lead to success or failure, respectively) in a new angle , that is, based on the analysis of the coefficients of a regression model instead of clustering and association rules analysis. Thus, we raised our third (and last) research question of this Chapter: *(RQ2-3) How can we interpret the results of the regression model to better understand effective and ineffective behaviours?*

3.3 Early Performance Prediction - State of the art

Research on how to improve the teaching and learning process in introductory programming courses has been a major focus in computing education research (ROBINS, 2019). For example, there are many relevant works pointing out the advantages of using e-learning systems, such as online judges (WASIK *et al.*, 2018). Some propose methods of exploring the large-scale data that come from the interaction of students

with these systems (LUXTON-REILLY *et al.*, 2018; WASIK *et al.*, 2018). This kind of data collection enables a *data-driven analysis of student behaviours* (IHANTOLA *et al.*, 2015; PEREIRA *et al.*, 2018; AGUIAR; PEREIRA, 2018; CARTER *et al.*, 2019; LIMA *et al.*, 2020), important to encapsulate the learners' progress during the programming courses (WATSON *et al.*, 2013; HELLAS *et al.*, 2018; CARTER *et al.*, 2019; PEREIRA *et al.*, 2020a). In this context, *early performance prediction* is gaining increasing attention, as it creates the possibilities to leverage the educational outcomes via early interventions (HELLAS *et al.*, 2018; ROMERO; VENTURA, 2019; PEREIRA *et al.*, 2020a). In general, researchers in this field investigate *features* that can be used in machine learning algorithms to make predictions (HELLAS *et al.*, 2018; PEREIRA *et al.*, 2020a). This section presents some relevant work that can contribute in the search of solutions to the problem of early prediction of programming student's performance, using data collected from e-learning systems employed in programming classes. Moreover, we draw parallels to our work to show both the need for it and its additional contributions.

Many works (JADUD, 2006; AHADI *et al.*, 2016; ESTEY; COADY, 2016; LEINONEN *et al.*, 2016; CASTRO-WUNSCH *et al.*, 2017; COSTA *et al.*, 2017; Abu Amra; Maghari, 2017; PEREIRA *et al.*, 2019b; PEREIRA *et al.*, 2019) have been collecting fine-grained data in the context of programming classes, to model students' behaviour and to quantify aspects of students' performance. All these studies have in common that they analyse the data-driven student behaviours, extract useful information, and use them as features in machine learning and inferential statistics techniques, with the goal of predicting the programming students' performance at an early stage in the course.

Costa *et al.* (2017) conducted an important work in predicting students' outcomes, comparing different machine learning algorithms and the impact of data pre-processing and fine-tuning of hyperparameters. The authors analysed data from 262 distance education students and 161 on-campus learners. They used demographic features, such as age, gender, civil status, and other data-driven features, like number of accesses to the system and participation in discussion forums. Their best prediction results have an f-score of 82% for on-campus students and an f-score of 80% for distance education ones, both results using data from the *first two weeks of the course*. Although the outcomes

of this work are promising, their database is small. Moreover, their use of demographic data is static in nature and, hence, fails to encapsulate changes in students' learning progress during the course. Still, their results missed out on continuous prediction (regression). They also did not explain their machine learning model's decisions. In this study, we also use their early, first-two-weeks-based prediction idea. However, we present a different approach, using non-static, data-driven features for *predicting student's grades both for discrete and continuous outcome*. Moreover, we explore the use of deep learning, instead of only shallow models and we interpret the model's results.

In a pioneering and highly cited study, Jadud (2006) conducted a data-driven analysis to propose an algorithm called *Error Quotient* (EQ), which uses snapshots of compilation, to quantify the errors from the students, whilst they are programming. In essence, the EQ algorithm received as input a pair of compilation events and assigned to them a penalty, if errors were found. The penalty could vary; for example, if the error of both compilation events were the same, then the penalty would be greater, as the student would be showing the same misunderstandings repeatedly. Watson *et al.* (2013) extended EQ with an algorithm computing the *watWinScore*, which scores compilation pairs, by additionally taking into account the problem resolution time. However, the results of these algorithms were modest in predicting students' outcomes. The algorithm proposed by Watson *et al.* (2013) obtained an average accuracy of 68.8% during the course, whereas the EQ results had an average accuracy of 55.8%, both for performance prediction in an experiment reported by Watson *et al.* (2013) with 45 programming students. In addition, for a regression task, the EQ explained $\approx 30\%$ ($r^2 = 0.3005$) of the students' grades, whilst *watWinScore* explained somewhat more, but still only $\approx 42\%$ ($r^2 = 0.4249$), both using data from the entire course. Using data from the beginning of the course (first 3 weeks), EQ and *watWinScore* had a poorly explained variance ranging from 5% to 10% ($r^2 = [0.5\tilde{}0.1]$). Despite the poor results for early prediction, they provided some interesting ideas, useful to explore in another education context. Thus, we considered using the EQ and *watWinScore* together, as features in a machine learning model, to investigate their predictive power in conjunction.

Leinonen *et al.* (2016) used Machine Learning models to detect students with

prior programming experience and to infer which students will pass or fail in CS1 classes. In order to do so, they used features based on typing patterns (e.g., number of pairs of alphabetic or numeric characters) and keystroke latency. For the problem of detecting whether the students would pass or not, they also achieved a modest accuracy of 65.8%, using data from 226 students in their first two weeks of course. Similar to EQ and *watWinScore*, more studies were considered to be required to be conducted, to check whether some combination of the features presented by Leinonen *et al.* (2016), together with others, would achieve higher outcomes.

Castro-Wunsch *et al.* (2017) adopted deep learning and traditional ML models to identify students (n=897) in need of assistance. The authors found that the number of attempts for each code exercise and the proportion of test cases accepted (correctness) in problems from online judges are effective predictors to be used as features in machine learning models. As a result, they achieved the best accuracy of 71.81%, using data from the fourth week of the course. Still, Castro-Wunsch *et al.* (2017) suggested in their conclusions that deep learning would likely achieve better results, with new combination of features and more data. Following this, and using similar features as Castro-Wunsch *et al.* (2017), Estey & Coady (2016) revealed that students at risk not only have fewer submission attempts, but also a lower compilation frequency, and different patterns in relation to the repeated consumption of hints generated in an online environment. Using data from 652 students for early prediction (first two weeks), Estey & Coady (2016) achieved an accuracy for the failing students of 30%. This results were also found in a replication of Estey & Coady (2016)'s work conducted by Fonseca *et al.* (2019). In brief, Estey & Coady (2016), Castro-Wunsch *et al.* (2017), Fonseca *et al.* (2019) demonstrated the predictive power of simple metrics (e.g., attempts and correctness), however their results were still modest. As such, there is a need of more improvements and new experiments with other features.

In an exploratory study, Auvinen (2015) analysed deeply the programming behaviours of 1777 students in an online judge and found that studying close to the deadline and trial and error behaviour is related to poor performance. Kazerouni *et al.* (2017) tracked and assessed procrastination behaviours and the amount of change of

370 students whilst solving programming problems. To check the positive and negative effect of such behaviours, they interviewed the students. As a result, they confirmed what Auvinen (2015) has found that procrastination behaviours can be dangerous for programming students. Following, Otero *et al.* (2016) analysed code metrics of programming students' codes, such as number of lines, number of conditions and so forth. As a result, the authors suggested that these code metrics were related to the student's performance. Again, all these works are relevant, however there is a need to understand their predictive power together in machine learning models, using data collected from different educational institutions.

In this sense, in an effort to compile these data-driven programming behaviours (features) exposed previously in this section, we have used some adaptations of code metrics proposed by the literature (JADUD, 2006; WATSON *et al.*, 2013; AHADI *et al.*, 2015; ESTEY; COADY, 2016; AHADI *et al.*, 2016; LEINONEN *et al.*, 2016; OTERO *et al.*, 2016; CASTRO-WUNSCH *et al.*, 2017) in Chapter 2 to early identify effective and ineffective behaviours. Now we are interested in testing those features for early prediction. To do so, we will use traditional ML model optimised by an evolutionary algorithm. To train and validate the model, we will employ the same data explained in previous Chapter, that is, data collected from 2058 learners, however, in the first two weeks of introductory programming courses.

Meanwhile, deep learning revolutionised the field of machine learning, by obtaining state-of-the-art results (MIIKKULAINEN *et al.*, 2019). Many areas have benefited from applying deep learning, and education is no exception (HERNÁNDEZ-BLANCO *et al.*, 2019). Moreover, as stated by Robins (2019) in a recent book, predicting programming students' performance is a complex problem. Géron (2019) explains that deep learning is suitable for large amounts of data (as for our dataset) and complex problems (such as the early prediction performance of programming students).

As such, we will compare SoA techniques of evolutionary algorithms for optimisation of ML pipeline with SoA deep learning architecture using fine-grained behavioural longitudinal data. We will use such cutting-edge techniques with the features combined in the previous Chapter, which comes from works presented in this

section. As such, we are addressing a claim from the literature (IHANTOLA *et al.*, 2015; QUILLE; BERGIN, 2018; HELLAS *et al.*, 2018; ROBINS, 2019) stating that more studies need to be replicated and new methods need to be proposed to advance in this field.

3.4 Method

In this work, we validate and compare machine learning pipelines with the goal of: i) estimating whether students will pass or not in a CS1 course using classification models; ii) predicting the students' final grades using regression models; and iii) interpreting what programming behaviours are leading factors for success and failure. To do so, we used data collected from a home-made online judge that will be described next.

3.4.1 Instrument

As in the previous Chapter, we also used an online judge called CodeBench, developed from scratch by one of the authors for automatic evaluation of students' solutions.

3.4.2 Data Collection and Programming Behaviours

We used the programming behaviours (features) defined in Table 2, explained in Chapter 2, extracted from students' interactions with CodeBench during their attempts to solve the programming problems. However, we added two features to try to perform a deeper analysis of correctness taking into consideration the events (logs) and whether students are redoing/rewriting their codes analysing how they delete their code. In addition, we added these two features because both deep learning techniques and the evolutionary algorithm perform feature selection and, hence, if these two features are not necessary they will be automatic excluded. The features that takes into consideration are described as follows:

- *correctnessCodeAct*: Represents the same as correctness, but in this case, we consider

'correct' only student's solutions with more than 50 events. To illustrate, if a student submits a correct solution by copying and pasting (only 1 event), for this problem it will be assigned 0 to the feature correctnessCodeAct, however for the feature correctness will be assigned 1.

- *deleteAvg*: average of deleted characters for each problem.

All the features were extracted from students logs when solving the programming problems from assignment and exams of the programming courses. The programming assignment has from 10 to 12 programming problems. These activities are followed by an exam, conducted in the same web-based system. There were 2 programming problems in each exam. The problems from the exams were on the same topics as the assignment beforehand. The exams were shorter because they were face-to-face in a lab. In this work, we call each pair formed of an assignment and an exam a 'session'. In total, 7 sessions were held throughout the course, lasting a little over 2 weeks each.

For the training of machine learning algorithms, *we use only the data from the first session (first two weeks)*, since the purpose of this method is to investigate early predictors. The content of the seven sessions were: variables and sequential structure (S1), conditionals (S2), nested conditionals (S3), 'while' repetition structures (S4), vectors and strings (S5), 'for' repetition structures (S6), and bi-dimensional matrix (S7). Thus, the sessions were structured so that problems became gradually harder.

3.4.3 Student Programming Profile

A 'programming profile' for each student in the first session (S1) was constructed by using 18 code features (programming behaviours) which represented metrics proposed by state-of-the-art studies. This 'student programming profile' was then digitally represented as a feature matrix, i.e., allocating each student a row and each feature a column. The features and corresponding description are listed on Table 2 and subsection 3.4.2, and sources are given.

3.4.4 Cleaning the features and dealing with imbalanced dataset

First, for classification, we normalise our features using MinMax (equation 3.1) to produce maximum absolute value of each feature scaled in unit size. We used this scaling technique due to its robustness to features with small standard deviation as ours and because it preserves zero entries. Moreover, such scaling technique is important for neural networks because if a feature has a variance that is orders of magnitude larger than others, the estimator might be unable to learn from other features correctly as expected (GÉRON, 2019; PEDREGOSA *et al.*, 2011).

$$X_{std} = \frac{X - min(X)}{max(X) - min(X)} \tag{3.1}$$

Moreover, the data generated by students that did not attend the course were removed from this analysis, since they did not have any interaction with the online judge. In addition, the database is very slightly unbalanced, since, unlike in other educational environments there are somewhat more students who passed ($\approx 56.7\%$) than failed ($\approx 43.3\%$). As we are dealing with features extracted from a very fine-grained log-data collected from students' interactions with an online system, some packets of data may be lost before reaching the server side. To illustrate, if a student loses internet connection whilst solving a problem on the IDE, then his/her logs will not be sent properly to the server. Still, students can start the course with a desirable behaviour, engaged, solving many problems from the assignments and performing well on the first exam, but change throughout the course, possibly due to external factors, ending with a low grade (and vice-versa). As such, our database might have some outliers. Thus, aiming to decrease the biases of the classifiers due to the unbalanced nature of the database and the presence of outliers, we used a statistical technique called Tomek Links (TOMEK, 1976) to remove noise. When a Tomek's link is formed on unbalanced datasets (as ours), Batista *et al.* (2004) recommended removing one instance from the majority class, in order to decrease the unbalance of the database, whilst also removing noise. Using this approach, we removed 100 instances from the majority class (students who passed) that form Tomek's links on our database.

Notice that a Tomek's link (L_T) is defined between two samples x_i and x_j of

different classes c_1 and c_2, respectively, for any sample y (equation 3.2), where $d(.)$ is the Euclidean distance between the two samples. In other words, a Tomek's link is represented by two samples from different classes that are the nearest neighbours of each other, which might confuse the ML model, when creating the decision boundaries to separate the instances of each class (BATISTA *et al.*, 2004).

$$L\_t = d(x_1, x_2) < d(x_1, y)\ and\ d(x_1, x_2) < d(x_2, y) \tag{3.2}$$

To further deal with our imbalanced dataset, we divided the data into homogeneous subgroups called stratum (stratified sampling), so that the right number of instances is sampled from each stratum in order to keep the same class proportion in the training and validation sets (GÉRON, 2019). To do so, we used the library *StratifiedKFold* from *scikit-learn* (PEDREGOSA *et al.*, 2011), using a total of 10 folds.

For regression, we simply divided the data into training (70%) and testing (30%) as the outcome is continuous and, hence, there is no problem of unbalancing. Moreover, instead of MinMax normalisation, we used standardisation with *z-score* (equation 2.1) as it measures values in terms of standard deviation, making it easy to interpret the coefficients and features values of the regression model.

3.4.5 Prediction and Validation

First, it is important to explain how the task of classification is performed, when the target is a continuous variable. We will employ the same approach used by the SoA work presented in section 3.3, which represented the problem as a binary classification, where students that achieve a final grade greater or equal to 5 are represented as the class 'passed', the rest being 'failed'. We use the grade 5 as threshold since this grade is used by the university to check whether the student passed or failed. Next we will show how we configurate and validate our classifiers and regressor.

3.4.5.1 Genetic Algorithms and Automatic Machine Learning

There is a research field that studies the automation of the machine learning process. This field is known as *Automated Machine Learning* (AutoML). Initially, the researchers investigated techniques for optimizing subsets of ML *pipeline*, such as automating the hyperparameter setting or selecting attributes (HUTTER *et al.*, 2015).

With the evolution of AI, sophisticated *AutoML* methods have been proposed. To illustrate, Feurer *et al.* (2015) presented *auto-sklearn* which is a tool that uses bayesiana optimization for the entire production of ML pipelines, that is, the pre-processing of attributes, selection of the ML algorithm and adjustment of hyperparameters. However, *auto-sklearn* is not able to produce a large number of pipelines. Indeed, *auto-sklearn* exploits a fixed number of *pipelines* steps that include only a pre-processing algorithm data, one attribute pre-processing algorithm and one ML algorithm on the *pipeline*.

On the other hand, Zutty *et al.* (2015) demonstrated that optimization with genetic programming can surpass humans in the search for a ML pipeline that best fits in a supervised learning task. In this way, Olson *et al.* (2016) proposed a method of construction and optimization of ML pipelines trees using genetic algorithms. The method created by Olson *et al.* (2016) is called *Tree-based Pipeline Optimization Tool* (TPOT) and the main idea is to initially create an entire population of random shallow ML pipelines and evolve them with mutations and crossover operators over the generations. To build the pipelines, Olson *et al.* (2016) employed several algorithms for selecting, constructing and transforming features and ML algorithms together with fine-tuning hyperparameters. Olson *et al.* (2016) explain that the tree is created using many operators.

To illustrate, Figure 13 presents an example of the process of building a pipeline tree, where each circle represents an operator. Note that copies of the training base are created so that operators can combine the modified attributes and data. Subsequently, there may be a process of selecting attributes so that the predictive model can finally be built. Also note that the pipeline shown in Figure 13 is an individual from the population that will evolve over the generations. Note that to carry out the evolution an elective selection is used, evaluating the individuals of the population with a *fitness function* that measures the performance of the pipeline in the training base. The fitness function in

this context can be accuracy or another performance metric.

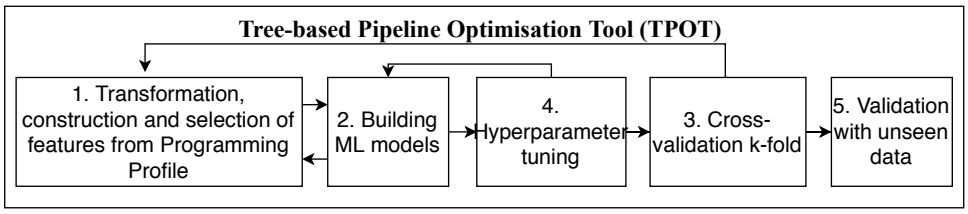

Figure 13 – Operators for building the ML pipelines using TPOT method. Adapted from Olson *et al.* (2016).

Finally, TPOT employs the NSGA-II (DEB *et al.*, 2002) algorithm, which is a multiobjective genetic algorithm. The objective functions used in the NSGA-II were to maximize the accuracy of the models and minimize the pipelines' complexity, in which complexity is measured by the number of operators in the pipeline.

Therefor, using TPOT, we found as the best model a regularised *Random Forest* (RF) with 100 constituent decision trees that considers 30% of the features when splitting the constituent decision trees using bootstrap for resampling. Moreover, the trees must have at least 8 instances to create a new branch. To perform the classification, the RF used *hard-voting*.

3.4.5.2 Deep Learning Model

To validate our hypotheses that a deep learning model will outperform the evolutionary algorithm that optimeses shallow ML models, we adopted the popular and widely used Multilayer Perceptron (MLP). We use MLP, as this is one of the most effective neural network techniques for modelling and prediction (GÉRON, 2019; MAHAJAN; SAINI, 2020). In addition, Géron (2019) has claimed that MLP can be effectively employed for modelling nonlinear and complex processes of the real world, and education is not an exception (HERNÁNDEZ-BLANCO *et al.*, 2019). In brief, MLP is a neural network fully connected that comprises an input layer followed by one or more hidden layers, and a final output layer. MLP is a feed forward Neural Network (NN) as the data flows only from the input to the output. MLP is considered a deep learning model as many authors use this term whenever neural network is involved (GÉRON, 2019), even for shallow NN.

In general, deep learning models are suitable for huge amount of data and complex problems (GÉRON, 2019). Here, we are tackling a complex problem (HELLAS *et al.*, 2018; QUILLE; BERGIN, 2019; ROBINS, 2019), however with not that much data. In this scenario, a highly deep MLP will likely perform well on the training, but not on the testing set (overfitting). As such, we configured our MLP with only two hidden dense[3] layers. Each dense layer has 64 nodes[4] and we use the state-of-the-art Rectified Linear Unit function (RELU) as activation function. We initialised the weights and biases of our NN randomly, following a normal distribution as widely recommended (GÉRON, 2019). As a way of regularising our NN, we added the widely used dropout technique (HINTON *et al.*, 2012; SRIVASTAVA *et al.*, 2014) followed by each hidden dense layer. We configured the dropout[5] with p=0.5, which means that 50% of the neurons of a hidden layer are ignored during training. Still, we used adaptive moment estimation (adam) technique (KINGMA; BA, 2014) for optimisation of the gradient descent, as this method tends to accelerate the convergence of the model and reduces the fast decay of the learning rates (GÉRON, 2019). In addition, we used the popular binary cross entropy as loss function for our classification problem. The output layer has two nodes, one for each class, and uses the sigmoid activation function, which gives a probability of a student passing or failing.

3.4.5.3 Regression Model

Moreover, a continuous estimation of students' grades can be more useful for instructors, as it is more detailed and thus more powerful and information rich than a binary classification (passed or failed). In this sense, we go a step further by adopting, besides classification, a regression model, to perform *continuous estimation*. To do so, we adapted the ideas behind stacking ensemble (WOLPERT, 1992). We configured the stacking ensemble using the outcome (predicted probabilities) from the MLP model as input on a meta-classifier (blender). In other words, as we performed 10-fold stratified cross-

[3] It is called a dense layer when all the neurons in a layer are connected to every neuron in the previous layer (fully connected layer) (GÉRON, 2019). Moreover, we tested different configuration for the model, varying the number of hidden layers of the NN from 2 to 8 on the training set.

[4] This value was found empirically, after testing 16, 32 and 64 on the training set.

[5] We experimented with different values for p (from 0.2 to 0.8), and 0.5 renders the best results

validation for the classification problem with our DL model, we have a probability prediction for each student based on the sigmoid activation function, which is estimated on each test fold of the cross validation. As such, these probabilities as one of the features (dlProbs) in our Ridge Regression model. Thus, we used this 'meta-feature' concatenated with the features from the programming profile (presented on Table 2) and used as input on an easily interpretable Ridge Regression algorithm, that worked as a meta-classifier (blender) on the stacking ensemble. This allows us to have the predictive power of our deep learning model (by using its outcome) along with the interpretability power of a Ridge regression linear model, which is important to understand the early programming behaviours that might be positive or negative in terms of success and failure in CS1 courses. We opted for Ridge Regression, as this regularised version of a linear regression is a good default in cases where we suspect that almost all features will likely be useful (GÉRON, 2019).

As extra information, we state that we also performed tests with other machine learning algorithms in the regression task (RandomForest, Extra Trees, XGboost and others), including with optimisation of hyperparameters; however, the results were not significantly superior to those found with our linear model of regression. Therefore, we chose to use Ridge Regression, which gives us, in addition to the predictive power, an easy interpretation of the confidence intervals of the regression coefficients.

3.4.5.4 Validation

For classification, we compared both models, using the following statistical metrics suitable for unbalanced dataset such as ours: *recall* (equation 4), *precision* (equation 5), *F1-score* (equation 6), and *accuracy* (equation 7), where TP stands for true positive, TN stands for true negative, FP stands for false positive, and FN stands for false negative. For regression, we used the widely recommended (MONTGOMERY *et al.*, 2012) MAE (equation 8) and the coefficient of determination r^2 (equation 9), where n is the number of samples, $\hat{y}_i$ is the predicted value of the *i-th* sample, y_i he corresponding true value,

and $\overline{y}$ is the average of all true values.

$$recall = \frac{TP}{TP + FN} \tag{3.3}$$

$$recall = \frac{TP}{TP + FP} \tag{3.4}$$

$$F1 = 2.\frac{precision.recall}{precision + recall} \tag{3.5}$$

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.6}$$

$$MAE = \frac{1}{n}\sum_{i}^{n} |y_i - \hat{y}_i| \tag{3.7}$$

$$r^2 = 1 - \frac{\sum_i^n (y_i - \hat{y}_i)^2}{\sum_i^n (y_i - \overline{y})^2} \tag{3.8}$$

3.5 Results and Discussion

We organised the results in three subsections, in order to answer our three research questions. In subsection 3.5.1, we compare our deep learning model with the best previous model found evolutionary algorithm, answering RQ2-1. In subsection 3.5.3, we show the results of our regression model, in order to respond to RQ2-2. Furthermore, in education, it is also important to understand which behaviours mostly affect student performance. As such, answering RQ2-3, in subsection 3.5.4, we analyse how each feature affected the regression model outcome in a move towards understanding each positive and negative programming behaviour from the students' programming profile.

3.5.1 Deep Learning versus Evolutionary Algorithm

In Chapter 2 we have showed a novel classification of students into effective, average and ineffective, based on their programming behaviour (features), which shows both

semantic and significant statistical differences. Here we also check whether the features are relevant to distinguish programming behaviours from students who passed and failed and, hence, whether they would be useful as features for the ML model and for our explanation of the model decision. For this, we first applied the non-parametric Mann-Whitney U test to compare the features of students who passed or failed. Results indicate a statistical difference (even with Bonferroni correction: $p \ll 0.05/18$) between all features. Hence, we opted not to perform feature selection and use all features as input to construct our predictive model. Another reason for not performing feature selection is that it is intrinsic to deep learning models and the evolutionary algorithm to automatically extract the best.

Given that, in this subsection we answer *RQ2-1*: *Would a deep learning model surpass an state-of-the-art evolutionary algorithm for early prediction of students' performance?'*. To answer this question, we ran the model using shallow classification settings constructed and optimised by an evolutionary algorithm, using TPOT. Subsequently, we compared the results with our deep learning predictive model (both predicting models are explained in subsection 4.4.1). For each predictive model, we ran the stratified cross-validation 20 times with 10 folds, varying the seed over a range from 1 to 20, in order to shuffle the database in different ways and, hence, explore the range of possible outcomes. Hence, we obtained 200 results for each metric (10 x 20), one outcome for each fold.

Figure 14 shows the performance of the models, in which we can observe a superiority of the deep learning model. For a deeper analysis, Table 4 shows the descriptive statistics of each model outcome, in which Random Forest (RF) depicts the model constructed and optimised by the evolutionary algorithm. Although the RF has less variance (measured here by the standard deviation and Interquartile Range), our deep learning model achieved a higher performance in all metrics (accuracy, F1-score, recall, and precision). Indeed, our accuracy ranging from $\approx 81.9\%$ to $\approx 82.7\%$ (C.L. 95%) using data from the first two weeks of the course for training, whereas the RF model found by our baseline achieved an accuracy ranging from $\approx 77.8\%$ to $\approx 78.6\%$ (C.L. 95%).

To check for statistical significance following we define null hypotheses and,

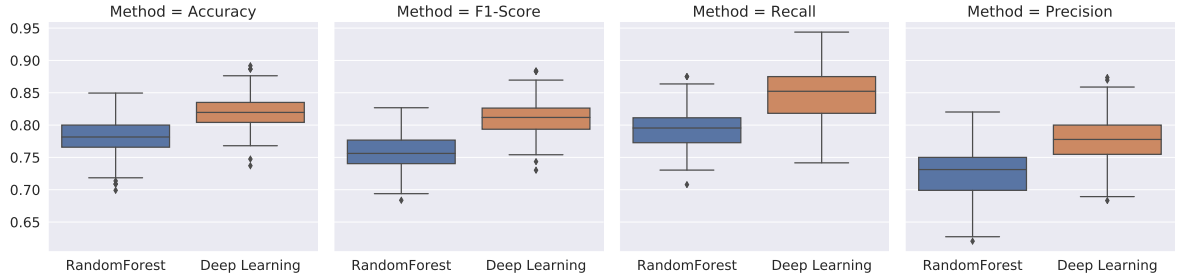

Figure 14 – Comparing performance of the predictive models - deep Learning vs. evolutionary algorithm for optimisation.

Table 4 – Statistics of the predictive models outcomes (better results are bolded). C.I. stands for Confidence Interval, L.B. stands for Lower Bound, and U.B. stands for Upper Bound.

		Accuracy		F1-Score		Recall		Precision	
		DL	RF	DL	RF	DL	RF	DL	RF
Mean		**0.823**	0.797	**0.818**	0.792	**0.860**	0.838	**0.782**	0.751
95% C.I. for Mean	L.B.	**0.819**	0.794	**0.814**	0.789	**0.854**	0.833	**0.777**	0.747
	U.B.	**0.827**	0.800	**0.822**	0.795	**0.865**	0.843	**0.787**	0.756
Median		**0.827**	0.798	**0.821**	0.792	**0.864**	0.835	**0.781**	0.747
Std. Deviation		0.027	**0.022**	0.027	**0.022**	0.039	**0.034**	0.037	**0.032**
Minimum		**0.754**	0.742	**0.746**	0.746	0.739	**0.766**	**0.692**	0.684
Maximum		**0.895**	0.855	**0.888**	0.847	**0.966**	0.915	**0.886**	0.830
Interquartile Range		0.037	**0.030**	0.037	**0.028**	0.053	**0.047**	0.052	**0.038**

in Table 5, we show the pairwise Wilcoxon test results to check these null hypotheses. Indeed, Table 5 shows that our model (DL) statistically surpasses our baseline (RF), even after Bonferroni correction ($p - value \ll 0.05/4$) in all model evaluation metrics (accuracy, F1-score, precision and recall), which shows that we can better recognise (higher recall) students who pass and fail with a higher precision. Thus, we can refute all the null hypotheses $H_0 1$, $H_0 2$, $H_0 3$, $H_0 4$. The effect size (η^2) is considered medium (COHEN, 2013) for all metrics (accuracy, recall, f1-score, and precision).

- $H_0 1$: The distribution of Accuracy is the same between the models Deep Learning and Random Forest.
- $H_0 2$: The distribution of F1-score is the same between the models Deep Learning and Random Forest
- $H_0 3$: The distribution of Recall is the same between the models Deep Learning and Random Forest.

- H_04: The distribution of Precision is the same between the models Deep Learning and Random Forest

Table 5 – Wilcoxon W test to compare the predictive models. Effect size (η^2) is considered small if η^2<0.01, it is medium if $.01 < \eta^2 \leq .06$, and high if $\eta^2 > 0.06$ (COHEN, 2013)

	Accuracy	Recall	F1-Score	Precision
Wilcoxon W	29028.00	33297.00	28971.00	30700.00
Z	-9.58	-5.89	-9.63	-8.13
Effect size (η^2)	.05	.02	.05	.04
Asymp. Sig. (2-tailed)	.00	.00	.00	.00

.

Moreover, giving a parallel with other relevant works (presented in section 3.3) that also have the goal of early performance prediction in introductory programming, Leinonen *et al.* (2016) achieved an accuracy of 65.8%, using data from 226 students in their first two weeks of the course and in extension work, using more data Edwards *et al.* (2020) achieved 72% of accuracy. Using data of 897 learners in their first four weeks[6], Castro-Wunsch *et al.* (2017) achieved an accuracy of 71.81%, Quille & Bergin (2019) achieved an average accuracy of 71% using early data from 692 students, Tomasevic *et al.* (2020) achieved 78% of accuracy. Indeed, our result is superior to all related works that performed early performance prediction (section 3.3). Other works such as Jadud (2006), Watson *et al.* (2013) that used only one feature achieved lower accuracy (<66%) using data from the whole course instead. Only Costa *et al.* (2017) achieved a similar result (80%), however, using demographic features. Notice that demographic attributes might be good predictors, however, the stakeholders have no control on the demographics (e.g. age, gender, etc.) of students and, hence, such static features are not useful in this work since we aim at predicting and prescription and, for the later, behavioural attributes are needed as stakeholders might stimulate the effective behaviours whereas guiding the ineffective ones.

In addition, although all these works were conducted in different education scenarios, their performance gives us the intuitions that the problem of early prediction is complex and our DL model achieved cutting-edge performance. Moreover, we are the

[6] The performance in the first two weeks is not reported in Castro-Wunsch *et al.* (2017)

first, to the best of our knowledge, to apply explainable artificial intelligence to interpret the predictions of the predictive model's decision, as it will be explained in section 3.5.4.

3.5.2 Performance Analysis Per class

Therefore, we now will further explore the results of our deep learning model, analysing the performance per class (failed or passed). In Figure 15 (left), we show the precision/recall curve of our model for different thresholds, whilst in Figure 15 (right) we show the ROC curves, i.e., the false positive rate (x-axis) and true positive rate (y-axis) also for different thresholds, where class 0 depicts the students who fail and class 1 represents the students who passed. Before the explanation about Figure 15 (left and right), it is worth illustrating what means threshold in the context of these figures. In a binary classification task as ours, the DL algorithm uses probability to classify a student as passed or failed. Thus, to perform the classification, the model uses a standard threshold of 0.5 (50%). As such, if the model estimates that the probability of a student passing is 60% and the threshold is equal to 50%, then the student will be classified as passed. On the other hand, the same student would be classified as failed if the threshold is equal to 70%. Note that increasing the threshold increases the confidence level to classify a sample as passed, however the algorithm's hit rate decreases, as it tends to estimate an observation as pass, only when it has a higher level of confidence.

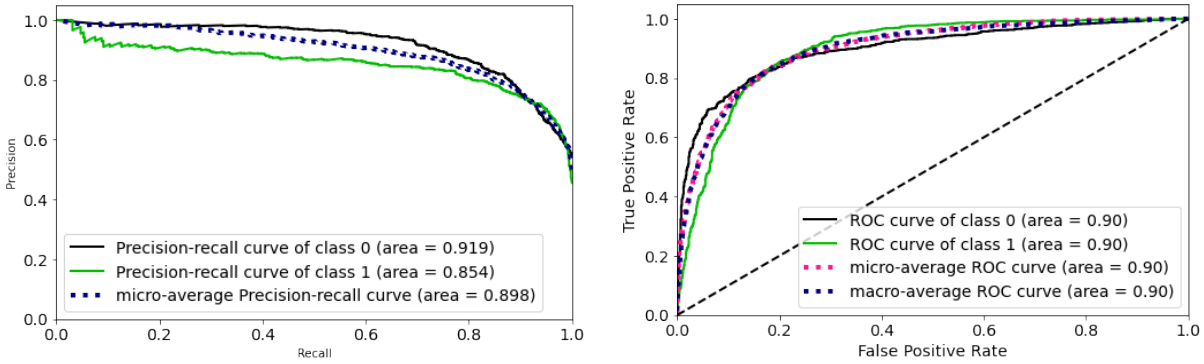

Figure 15 – Precision and recall curve (left) and ROC curve (right) of students who passed (class 1) and students who failed (class 0). The micro-average takes into consideration the class proportion of students who passed and failed, whereas the macro-average threats each class independently.

Given that, from a visual inspection of Figure 15, it is possible to see that our model indeed performed well on both classes. With regards to the precision/recall curve, we achieved an area under the curve of 0.854 for students who passed and ≈ 0.92 for learners who failed. Still, with regards to the ROC curves, we achieved an area under the curve of 0.90 in both classes. This shows that our model distinguished students who passed and failed, even when the threshold was different from the central value (0.5). This can be seen by analysing the continuous lines (green and black) of Figure 15 (right) (left and right) that are close together, sometimes overlapping.

3.5.3 Regression Analysis

In this subsection we answer *RQ2-2*: *'How we can effectively use the same data as for student result classification in a regression model, to obtain early prediction of the students' actual final grades?'*.

As we are using a linear model with regularisation, Ridge Regression, we first adjust the alpha value, which controls the amount of regularisation, i.e., the higher the alpha, the less complex (and higher bias) the model, whereas the lower the alpha, the more complex (and higher variance) the model. Notice that too complex models can lead to overfitting, whilst a too simple model can cause underfitting. Thus, we tested different values of alpha to find a balance between this bias-variance trade-off, as can be seen in Figure 16. As a result, the best value found is alpha=2.474 (see the vertical dotted line). Using this alpha, we achieved a coefficient of determination (r^2) of approximately 0.62 (see Table 6). Thus, overall, our model can explain 62% ($r^2 \approx 0.62$) of the variance on students' final grades, by using a relatively high degree of freedom for the residuals (Df residuals) and a low degree of freedom for the model (Df model). This is important in linear regression, as r^2 only increases when the Df model (number of features + 1) increases (MONTGOMERY *et al.*, 2012). Indeed, the predictions of our regression model is statistically significant ($p - value < 0.05$ and $f - statistics = 166.4$).

In addition, for a better understanding of our regression's performance, we evaluated our MAE, which is 1.6 [+/- 0.11, CL of 95%] (Table 16), what means that in

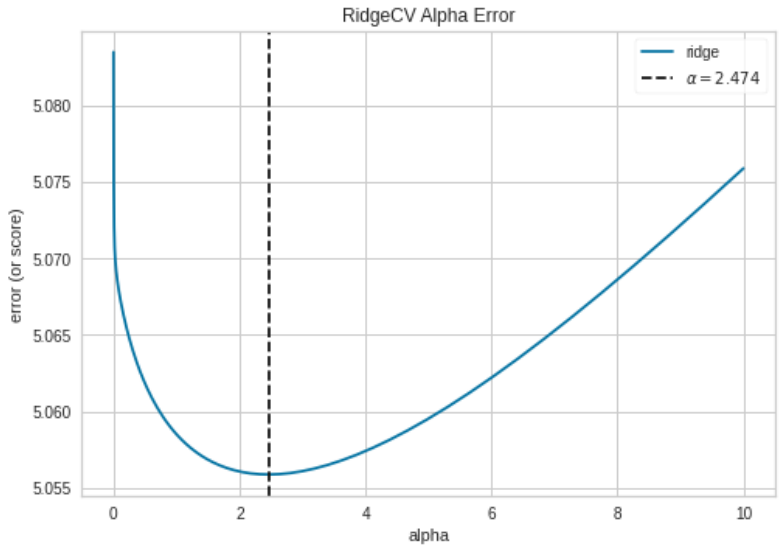

Figure 16 – Choosing the optimal alpha for our Ridge Regression Model. We use as error measure the Mean Squared Error (y-axis) as this statistic metric gives a higher penalty to errors than Mean Absolute Error and Coefficient of Determination.

Table 6 – Regression Results. Df stands for degrees of freedom, MAE stands for mean absolute error

Df residuals	1936
Df model	19
MAE	1.6
R2	0.62
F-statistic	166.4
p (f-statistic)	0.00

some cases the model's residual might be high, as these predictions are on a scale of [0-10] (students' grades). Nonetheless, notice that MAE is the mean of the residuals[7] absolute values (equation 8) and the mean is quite sensitive to outliers.

In this sense, in Figure 17, we plot the actual grade (y) of each student on the x-axis, whilst on y-axis we plot the grades predicted ($\hat{y}$) by our model. Moreover, we show a dashed 45-degree line (identity) to compare with our model's best fit prediction (best fit). Using this we can inspect the amount of variance in our regression model, i.e., where the model makes bigger mistakes and where the model hits. To illustrate, there are many points close to the identity, which means that the residual of the prediction is close to zero. However, we can observe a few outliers along a range of the target domain, which might explain the high value of our MAE. To illustrate, notice the density of

[7] n this case, residual is the difference between the actual student's grade and the predicted grade

errors on the point zero of the x-axis forming a kind of a 'vertical line', which means that the model predicted higher grades for some students who achieved an actual grade close to zero. A possible explanation for that is that we are using data from the very beginning of the course (first two weeks) and, hence, some students might begin the course with effective behaviours that probably would lead to success. However, during the course they might give up because of personal reasons or even become less engaged and dropout. As instructors, we know empirically that students can change their posture in terms of engagement during the course, i.e., they can begin well but end up failing. We can also observe in this figure a few cases of the opposite phenomenon, i.e., students who achieved higher grades but the model predict low grades to them, which likely means that these students begin with bad habits but end up passing with higher grades because of possible changes on habits. As such, for regression, we believe that our model predictions are not highly accurate for extreme values (0's and 10's) because of those uncontrollable and unknown factors influencing the students' grades, typical in human sciences. However, we believe that when using our regression model combined with teachers, these estimator errors for extreme values (0's and 10's) will be detected by the teachers 'intuition and experience with students, thus lessening the limitations of our regression model.

In Figure 18 we can see the residuals are proportionally distributed along the zero axis, which indicates there is no heteroskedasticity. To confirm that, we applied *Goldfeld-Quandt test*, and we did find homoscedasticity ($p-value = 0.3471$). Besides that, Figure 19 shows the residuals of our model on y-axis and the actual students' grades on the x-axis. With this in hand, we can perform a visual inspection of the regions within more or less errors regarding to the students' grades prediction, confirming what we explained previously about our prediction on extreme grades (0's and 10's). Nonetheless, we can see that in overall the points are close to the horizontal black line (point zero on y-axis), which means lower errors. Moreover, we claim that the regularization was done properly as the predictions on the training and testing set have almost the same spectrum and residual distribution (see the histogram), demonstrating that our model generalises well. Still, the histogram of the residuals shows a distribution centred on

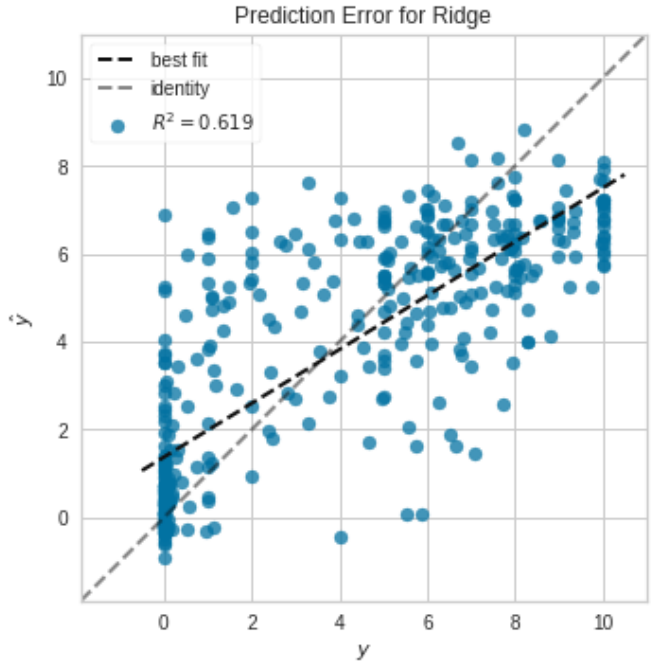

Figure 17 – Prediction error of our regression model. On the x-axis we show the actual values and on y-axis we show the predicted values for each student on the testing set.

zero and with a bell shape, which suggests a linear underlying nature of the data.

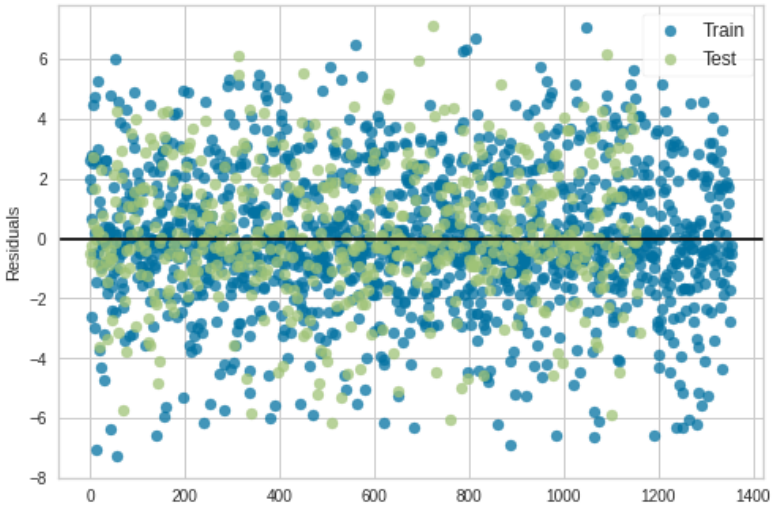

Figure 18 – Residuals of our regression model.

3.5.4 Analysis of Feature Effects on the Regression Outcomes

In this subsection, we answer *RQ2-3: 'How can we interpret the results of the regression model to better understand effective and ineffective behaviours?'*.

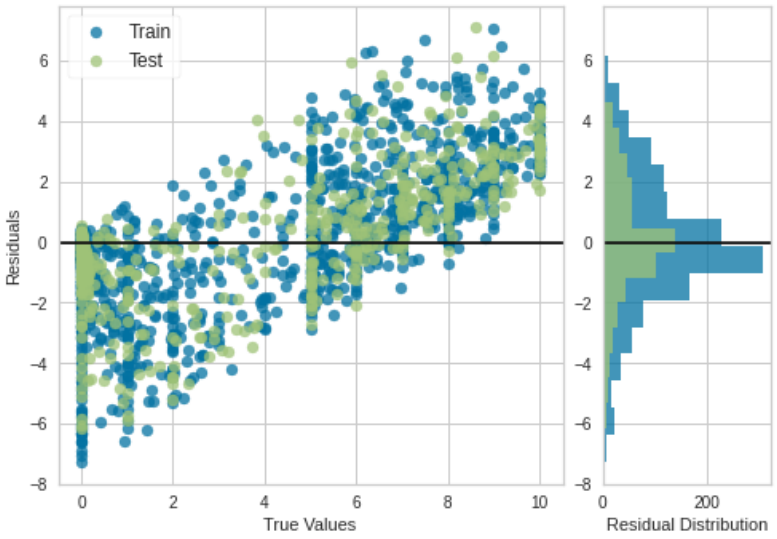

Figure 19 – Prediction error (residuals) of our regression model in another perspective.

A better understanding of what students programming behaviours might be effective or ineffective and what goes behind the students' grades could lead to a better analysis of which strategies we might use to teach and how our students would like to learn. As such, in this subsection, we go beyond prediction and move towards analysing the effects of features in our regression model outcome with the aim at understanding the major factors that can lead to success and failure in our cohort. To do so, we will analyse the coefficients of our regression model. Figure 20 shows a *Pareto* plot with the cumulative relevance of each feature, measured by the absolute value of each coefficient. We can see that the *dlProbs* (predicted probability from the DL model), is the most influential factor, being responsible for 47% of the prediction's weight, which shows the technique of stacking works as we expected. In addition, the features *dlProbs*, *firstExamGrade*, *systemAccess*, *correcteness* and *events* accomplish for 80% of the predictive power of the model, which follows the *Pareto's Principal*, or the 80/20 Pareto's rule, which states that, in general, 80% of the effects come from 20% of the causes (NEWMAN, 2005), in our case 20% of the features.

Moreover, in Table 7, we show the coefficients values, their confidence interval, statistics, and their statistical significance. A simple way to interpret these coefficients is the higher the absolute value the higher effect on the model outcome. To illustrate, for every additional degree (increase by one std as we used *z-score* standardisation) of *ideUsage*, the expected increase on final grade is something between [0.09-0.49] (95%

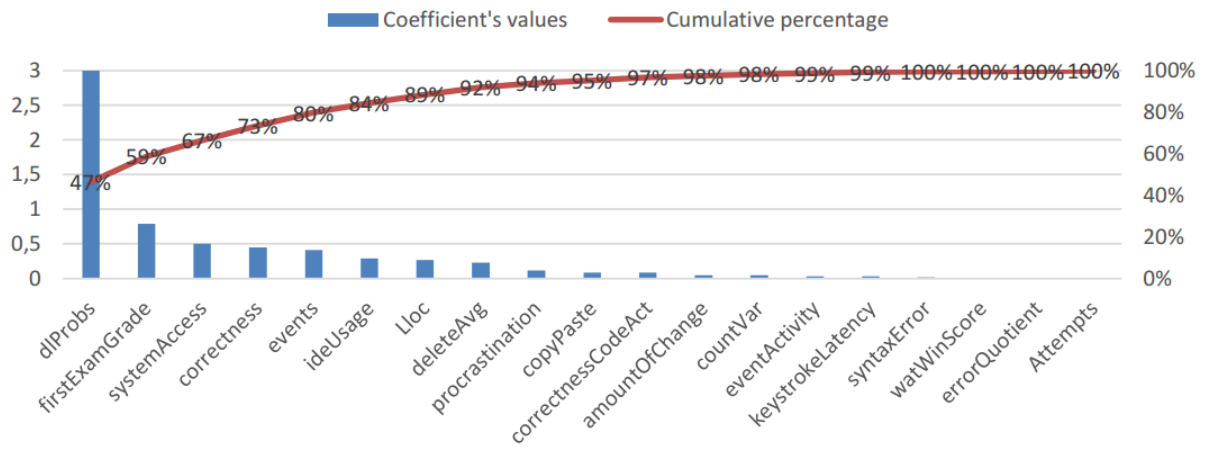

Figure 20 – Pareto plot of the feature's relevance based on their coefficients.

C.L.), or 0.29 on average, keeping all other variables constant. With this in mind, the top features with higher impact are, respectively, *dlProbs* (predicted probability from the DL model), *firstExamGrade*, *systemAccess*, *correctness*, *events*, *ideUsage*, and *lloc*. First, dlProbs is the most influential factor which shows the technique of stacking works as we expected. Following, all the other top features have a positive magnitude which means that the greater the value the higher the positive effect regarding to the final grade of students. About *firstExamGrade* and *correctness* appearing the top features was only a confirmation, as we expected the students who do well on the first exam and assignment tend to perform better. However, as a more hidden pattern found, we can see that coding activity also plays an important role (*systemAccess*, *events*, *ideUsage*, and *lloc*) for student success. In other words, it suggests that it is important for learners to access (*systemAccess*) regularly the online judge actively (high *events*, *ideUsage*, and *lloc*), that is, solving the problems from the assignments with a reasonable number of logical lines of code (*lloc*), and spending time on the IDE with a considerable number of events generated (*events*), which means that the student is typing something on the IDE whilst he/she is accessing the online judge. This support our findings in our previous Chapter 2

The other features that follow are, respectively, *deleteAvg*, *procrastination* and *copyPaste*. These three features have a negative coefficient, which means that lower values are better. As such, these coefficients might suggest that procrastination and a higher number of *copyPaste* on the first code solutions of an introductory programming

Table 7 – Coefficients of our regression model. Statitiscally significan coefficients bolded ($p - value < 0.05$).

	coef.	**std err**	**t**	**p> | t |**	**[0.025**	**0.095]**
Intercept	3.0	0.14	21.43	**0.00**	2.72	3.28
dlProbs	3.0	0.28	10.71	**0.00**	2.44	3.56
firstExamGrade	0.79	0.08	9.88	**0.00**	0.63	0.95
systemAccess	0.5	0.07	7.14	**0.00**	0.36	0.64
correctness	0.45	0.14	3.21	**0.00**	0.17	0.73
events	0.41	0.14	2.93	**0.01**	0.13	0.69
ideUsage	0.29	0.1	2.90	**0.01**	0.09	0.49
Lloc	0.27	0.06	4.50	**0.00**	0.15	0.39
deleteAvg	-0.23	0.11	-2.09	**0.04**	-0.45	-0.01
procrastination	-0.12	0.05	-2.40	**0.02**	-0.22	-0.02
copyPaste	-0.09	0.04	-2.25	**0.03**	-0.17	-0.01
correctnessCodeAct	0.09	0.09	1.00	0.24	-0.09	0.27
amountOfChange	0.05	0.06	0.83	0.28	-0.07	0.17
countVar	-0.05	0.07	-0.71	0.31	-0.19	0.09
eventActivity	-0.03	0.06	-0.50	0.35	-0.15	0.09
keystrokeLatency	-0.03	0.07	-0.43	0.36	-0.17	0.11
syntaxError	0.02	0.06	0.33	0.38	-0.1	0.14
watWinScore	0.01	0.05	0.20	0.39	-0.09	0.11
errorQuotient	0.01	0.06	0.17	0.39	-0.11	0.13
Attempts	0.01	0.08	0.13	0.40	-0.15	0.17

course are not desirable programming behaviours, similar to what we pointed out in the previous Chapter 2. Furthermore, a high value of *deleteAvg* might indicate that students are struggling, by deleting the code and redoing it repeatedly.

Finally, differently from Chapter 2, the following features have less impact (no statistical significance, $p - value \gg 0.05$) on the model outcome: *correctnessCodeAct*, *amountOfChange countVar*, *keystrokeLatency*, *eventActivity*, *syntaxError*, *watWinScore*, *attempts*, and *errorQuotient*. This is interesting, since the literature (JADUD, 2006; EDWARDS *et al.*, 2009; WATSON *et al.*, 2013; LEINONEN *et al.*, 2016; CARTER *et al.*, 2019) report that these features are generally useful to predict student's performance. However, for this cohort and for our regression problem, this is not the case. A possible reason is that, again, we are dealing with data from the first two weeks of the course and, hence, some patterns are still not evident yet. For example, repeating the same errors (*errorQuotient*, *SyntaxErrors*, *watWinScore*) could not be a common behaviour in the beginning of the course because the problems are very easy. However, when the problems become more challenging, these students would need to use debugging tools,

otherwise they might be stuck with the same errors increasing the value of metrics that compute errors. Notice that in the previous Chapter we used data from the first fourth weeks, whilst in this Chapter we are using data from the very first two weeks.

In closing, these features may not have a high relevance in isolation; however, together they have a reasonable predictive power for classification and regression. We believe that these top features might be generalised for other contexts, as they do not depend on nuances of a given compilation message or a specific programming language or web-based system. Moreover, some of our features were not statistically significant for our regression model, which indicates there is no linear relationship with the students' final grades and these two features.

3.6 Pedagogical Implication of Early Prediction

Ihantola *et al.* (2015), Yera & Martínez (2017), Luxton-Reilly *et al.* (2018), Robins (2019), Júnior & Pereira (2020) advocate about the importance of students solving many problems to improve their *programming skills*. However, this leads to a significant increase on the workload of instructors, as they need to evaluate many codes from many students. In this sense, the use of online judges in CS1 courses are gaining momentum (IHANTOLA *et al.*, 2015; CARVALHO *et al.*, 2016; WASIK *et al.*, 2018), since they bring benefits in decreasing instructors/monitors workload and enable a data-driven analysis of CS1 students' behaviours. Blikstein (2011), Carter *et al.* (2019) explain that such analysis allows a formative assessment, as not only the code submitted by the learners can be evaluated, but also the process behind it, when students are building their code.

Moreover, in programming classes, we teach our students to be excellent problem solvers. According to the SOLO taxonomy, this represents the highest level of abstraction. SOLO (BIGGS; COLLIS, 2014), which stands for Structure of the Observed Learning Outcome, is a general education framework that describes levels of increasing complexity in a learner's understanding of a subject. It ranges from pre-structural (no understanding), to unistructural, multistructural, relational, and extended abstract (understanding at a high level that enables generalisation to a new topic or area). Although

instructors of CS1 classes aim to bring all students to the extended abstract level, many students will probably remain at the lower levels.

In our work, we aim at high SOLO levels by using a methodology based on practice and formative assessment. As explained by Robins (2019), *hands-on experience* is the most effective form of learning for students. Thus, in the 7 sessions of the CS1 program, we propose many assignments and exams, always with a close tutoring, allowing a formative, beside the summative assessment. In other words, the fine-grained data collect from our online judge allows for a formative assessment, as not only the code submitted by the students can be evaluated, but also the process behind it. This kind of assessment provides a comprehensive view upon the student's learning and can guide the teacher towards truly effective interventions.

Additionally, with early prediction, in a standard course, teachers could provide extra assignments for the high-achieving group and personalised support to those who are struggling. If the strategies of effective novices can be identified, it may be possible to promote effective strategies to all groups. Such early prediction allows personalised feedback, but this is not scalable to large classes without proper technological support. With an approach such as ours, effective and ineffective behaviours can be automatically identified early on, and encouraged or discouraged, respectively. Such process of early intervention can be performed using dashboards, e-mails, etc. In other words, as 'prevention is better than a cure', likewise, it is better to prevent students from failure as soon as possible, instead of finding out students are struggling when their poor marks come in.

Moreover, this approach prevents *fragile learning* (ROBINS, 2019). This occurs when students pass by learning in a short period something that they should have learn over the entire course (by 'cramming' at the end). As such, it is crucial to decrease the time lag between detecting at-risk students and acting to provide early help to the learner.

Importantly, according to Ihantola *et al.* (2015), this kind of approach presents essential benefits for the educational context since learning analytics promise better understanding of student behavior and knowledge, as well as new useful information

on the hidden factors that contribute to learners' actions. This knowledge can be used to inform decisions related to course and tool design and pedagogy, and to further engage students and guide those at risk of failure.

3.7 Chapter Conclusions

In this work, we developed a deep learning pipeline that statistically outperforms a predictive model optimised by an evolutionary algorithm. These results were achieved using features extracted from data collected in the very first two weeks of introductory programming courses, allowing early intervention. We achieved a competitive performance in both tasks of classification and regression. Such high performance from our predictive models can facilitate human–AI collaboration towards prescriptive analysis, where the instructors/monitors can take advantage of the predictions to implement different and personalised pedagogic approaches targeting at-risk students and higher achievers, leading to profound effects for students' learning and experience. On the student side, such prediction can promote self-regulation and awareness of their strengths and weakness, showing that there is room for improvement.

In brief, we can claim there are significant benefits in using data-driven code metrics presented in our *programming profile* to design LA tools. Importantly, as the data refers to the student's learning process and not only to the final product generated and submitted (code with the solution), our study is a move towards formative evaluation. That is, our model is not inspecting only the feedback from 'online judges' which judge only the program sent, but also taking into account all the paths taken (challenges and difficulties faced) by analysing a fine-grained log-data of students during the construction of the code produced and submitted. Still, we interpreted our regression model outcomes, showing the effect of programming behaviours related to how students solve problems, how active they are whilst solving these problems, how they are managing deadlines, and so forth. We believe that the features of the programming profile are not sensitive to the educational context, since they are not tied to the nuances of the programming language and, hence, are sufficiently generic.

Nonetheless, some of our features were not statistically significant for our regression model, which indicates there is no linear relationship with the students' final grades and these two features. However, our cutting-edge results using DL for classification task were achieved using non-linear algorithms, which may suggest a non-linear relationship between the features and the final grade. Additionally, we can observe from the Pareto plot (Figure 20) that the embedded probabilities (dlProbs) from the DL model (non-linear) comprise almost for the half (47%) of the predictive power of the linear regression model, which, hence, enforces the indication of a non-linearity relation among the features and students performance, that is, our data is likely more complex than a "straight line" (actually a hyper-plane) from a linear regression. Thus, our regression model is only the first step in the direction of explainable machine learning. Indeed, more research needs to be conducted to interpret this potential non-linear relationship, what we will do in the next Chapter.

Notice that we could have used polynomial transformation on the features to perform polynomial regression, in an attempt to cover such mentioned non-linear relationship. In other words, we could have added powers of each feature as new features and have used them in our regularised regression model. However, using polynomial transformation we would need to add all combinations of features up to the given degree[8], which is not feasible for interpretation purposes due to the combinatorial explosion. To illustrate, we have 18 features and, using a polynomial degree of 5, for example, there will be *33649 new features*[9] to use in the regression model, which is prohibitive for interpretation. As such, in the next Chapter, we opt to use a game theory-based framework, that covers linearity and non-linearity, to analyse how the effective and ineffective behaviours (represented by our features) effect the students' performance.

[8] For example, 2 features x, z, and a polynomial degree of 2, we would have x^2, z^2, xz, x^2z, xz^2, and x^2z^2

[9] $\frac{(n+d)!}{d!n!}$, where n is the number of features and d is the polynomial degree.

4

Explaining individual and collective effective and ineffective programming students' behaviour

> So teach us to number our days
> that we may get a heart of
> wisdom.
>
> - Psalm 90:12

4.1 Overview of the Chapter

Predicting students' performance as rapidly and early as possible and analysing to which extent initial students' behaviour could lead to failure or success is critical in introductory programming courses, for allowing prompt intervention in a move towards alleviating their high failure rate. As CS1 is often the first point of contact with programming (and possibly last) for many students, it is vital to efficiently understand their needs. Nevertheless, as shown in the previous Chapter, the highest precision predictive models are typically black box models, and don't provide any interpretation. Lately,

the research community has started understanding the importance of working towards explainable Artificial Intelligence (AI). However, in CS1 performance prediction, there is a serious lack of studies that interpret the predictive model's decisions, to understand why the student is classified as failed or passed. Thus, in our current Chapter, we tackle both fronts: i) constructing an accurate early predictor and ii) interpreting its predictions. To do so, we use the same data from the previous Chapter, but now exploring deeply the relationship between features and students' performance. To do so, we construct a new non-linear model based on gradient boosting and decision trees to predict learners' performance that has the same (statistically saying) predictive power of our DL model, however, with the advantage of individual and collective interpretation of the model's decision. More specifically, to allow an effective intervention and to facilitate human/AI collaboration towards prescriptive analytics, we, for the first time, to the best of our knowledge, go a step further than the prediction itself and leverage this field by proposing an approach to explaining our predictive model decisions individually and collectively using a game-theory based framework (SHAP), as recently recommended in the Nature Journal (LUNDBERG *et al.*, 2020), that allows interpreting our black box non-linear model linearly. In other words, we explain the feature effects clearly by visualising and analysing individual predictions, the overall importance of features, and identification of typical prediction paths. This method can be further applied to other emerging competitive models, as the CS1 prediction field progresses, ensuring transparency of the process for key stakeholders: administrators, instructors, and learners.

4.1.1 Practitioner Notes

What is already known about this topic:

- Knowing about student performance in advance can be useful for many reasons (PEREIRA *et al.*, 2020a; ROMERO; VENTURA, 2019).
- The highest performance for prediction tasks is accomplished by 'black box' ML algorithms (GÉRON, 2019; LUNDBERG *et al.*, 2020).

- Besides the early prediction, it is also important to interpret the model's decisions to analyse behaviours that could lead to failure or success (ROBINS, 2019; BERENDT *et al.*, 2020).

- Stakeholders need not only a collective and overall explanation of models' decision (e.g. using only the feature importance of the predictive models), but also an individual analysis of the features for each instance (here, student), as the learners' behaviours are heterogeneous (ROBINS, 2019; BERENDT *et al.*, 2020). Thus, a general effective or ineffective behavioural pattern might not be applicable to all learners.

What this Chapter adds:

- A new interpretable predictive model that achieved cutting-edge results for early prediction.

- An important move towards Explainable, Transparent AI in Education (BERENDT *et al.*, 2020), by demonstrating how to explain the predictive model's decision (individually and collectively), to better support students and instructors (and other stakeholders).

- Collective and individual explanation of the model's decisions by the identification and analysis of typical prediction paths and analysis of feature effect for each instance.

- Identification and analysis of typical prediction paths for general behaviours.

Implications for practice and/or policy:

- Early prediction empowered by its explanation might potentially allow an effective early intervention by the stakeholders.

- A visualisation dashboard including prediction and interpretations might contribute for a more formative assessment.

- Reflection and diagnosing potential causes of the students' lack of success for stakeholders.

- Metacognitive strategies which get the students to think about their own learning.
- Predicting and explaining the student performance at an early stage might facilitates human/AI collaboration towards prescriptive analytics.

4.2 Research Questions Addressed in this Chapter

Typically, educational data-driven researches identify patterns of behaviour based on data collected from the students learning process (BAKER; INVENTADO, 2014). In general, there are many works (DWAN *et al.*, 2017; CASTRO-WUNSCH *et al.*, 2017; QUILLE; BERGIN, 2018; CARTER *et al.*, 2019; QUILLE; BERGIN, 2019; ROMERO; VENTURA, 2019; FWA, 2019; PEREIRA *et al.*, 2021a) in this field that use machine learning to construct models to predict the students performance in programming classes. However, as explained in the previous Chapter, a book about introductory programming research (ROBINS, 2019) claims there is no reliable method in the literature to predict CS1 students' performance. In this sense, we advanced in this field by constructing a deep learning model with cutting-edge results. Moreover, we adapted the deep learning model probabilities in a linear regression model to interpret the leading factors of effectiveness and ineffectiveness. However, our results using DL for classification task were achieved using non-linear algorithms, which may suggest a non-linear relationship between the features and the final grade. Notice that linear regression does not uncover such kind of non-linear relationship[1]. Thus, more research needs to be conducted to interpret this potential non-linear relationship.

The highest performance for prediction tasks is accomplished by 'black box' ML algorithms, which even professional data scientists struggle to interpret. Thus, the next challenge is to extract the explainable, transparent model for the AI in Education for CS1 (BERENDT *et al.*, 2020). Some previous attempts (ALTMANN *et al.*, 2010; LIU *et al.*, 2012; QIU *et al.*, 2018) to explain predictive ML models in other fields proposed the analysis of the overall importance of general and specific features to the model. However, in

[1] We could apply manually polynomial transformation to deal with non-linearity, but it would make difficult to interpret the model.

reality, more than a collective explanation of features is needed, as learners' behaviours are heterogeneous and, thus, a generic average pattern might not match individual learners. In other words, it is also important to make a deep individual analysis of the model's decision for each student.

Carter *et al.* (2019), Robins (2019), Quille & Bergin (2019) claim there is a lack of studies that use such prediction analyses to improve instruction and pedagogy. Indeed, there is a need for a descriptive, predictive and prescriptive analytics infrastructure that provides information to support instructors and students. Recently, Carter *et al.* (2019) state that there are relevant open questions concerning what learning data should be collected within an e-learning environment for programming courses in order to provide a foundation for improving student learning and how the learning data should be analysed to provide useful information on student learning in individual and collective level.

Thus, in this Chapter, our main focus is on understanding which students' early programming behaviours are related to the learner's success or failure and, hence, with effectiveness and ineffectiveness. Moreover, we aim at analysing students' behaviours generally, to give stakeholders a big picture of early programming behaviours, and individually, to provide an analysis of students' specificities, allowing self-regulation and higher self-knowledge for the learner. To achieve this goal, we constructed a non-linear predictive model using the features of our previous Chapters and we applied a game-theory based framework (SHAP method) (LUNDBERG; LEE, 2017; LUNDBERG *et al.*, 2020) that allows interpreting our black box non-linear model linearly. The features depict useful information from fine-grained log-data collected from a home-made online judge system (PEREIRA *et al.*, 2020) used in our programming classes. We believe that our findings are important to enrich the research on programming learning with findings of effective and ineffective early students behaviours (currently considered an open question (ROBINS, 2019), and the educational data mining field with an accurate and explainable machine learning pipeline that can be useful for early intervention and student self-regulation. In brief, in this Chapter we will respond the following research questions:

RQ3-1) How to interpret the features effects in the black-box non-linear model's prediction for each student (individual analysis)?

RQ3-2) How to interpret the features effects in the black-box non-linear model's prediction for all students (collective analysis)?

4.3 Explainable Machine Learning

Nowadays, ML is mainstream, with great potential to improve education. However, predictive models often do not explain their decisions, which might be a barrier to adoption (MOLNAR, 2020). There are some simple ML methods, such as decision trees, linear regressions, decision rules, which are easily explainable (MURDOCH *et al.*, 2019; MOLNAR, 2020). However, they often lack predictive power, possibly because higher accuracy for complex datasets is commonly achieved by non-linear black-box models (LUNDBERG; LEE, 2017), such as deep learning and ensembles (GÉRON, 2019; CHOLLET *et al.*, 2018). Consequently, a trade-off appears between performance and interpretability.

In this sense, the literature has been proposing new methods for explaining complex ML models at breakneck speed and it is often unclear how these methods are related and which one to choose (MOLNAR, 2020; RAI, 2020; LUNDBERG; LEE, 2017). As a response to this, (LUNDBERG; LEE, 2017) proposed a unified framework for interpreting predictions, SHAP (SHapley Additive exPlanations). This state-of-the-art method unifies in a single framework prestigious additive feature attribution methods such as LIME(RIBEIRO *et al.*, 2016), DeepLIFT(SHRIKUMAR *et al.*, 2017), classic Shapley value estimation(LIPOVETSKY; CONKLIN, 2001), layer-wise relevance propagation (BACH *et al.*, 2015) and others. Still, (BOULANGER; KUMAR, 2020) note that, no matter the SHAP implementation used, Shapley values are challenging to interpret. Thus, as one of the contributions of our paper, we, for the first time, to the best of our knowledge, are interpreting a black-box model to better understand effective and ineffective programming student behaviours. This allows trust in early performance predicting, through transparency, as recommendations based on machines that may

impact on human life need to be tractable and explicit.

SHAP is a method with foundations in game theory (SHAPLEY, 1953), where the features divide rewards in a way which reflects each of their contributions to the model's prediction (LUNDBERG; LEE, 2017; MOLNAR, 2020). The SHAP interpretation method calculates the Shapley values for each feature at instance-level.

In practice, using SHAP we can compute the magnitude of positive or negative effects for each feature on individual predictions. To do so, the method tests how the prediction changes when feature j is withheld from the model (LUNDBERG; LEE, 2017). In other words, SHAP calculates the feature importance of a feature $j \in F$, for a given local instance *x*, in a given predictive model *f*, by evaluating the marginal contribution of that feature *j* for all subsets $S \subseteq F$, where *F* is the set of all features. Thus, the marginal contribution of *j* (Shapley value) is calculated by the weighted average of $f_{S\cup\{j\}}(x_{S\cup\{j\}}) - f_S(x_S)$ for all subsets $S \subseteq F$, where x_S depicts the values of the input features in the subset *S*. Formally, the *contribution* of a given feature *j* is measured by the following equation, that is the weighted average for all possible differences, computed as the combination function:

$$\phi_j = \sum_{S\subseteq F\setminus\{j\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!}[f_{S\cup\{j\}}(x_{S\cup\{j\}}) - f_S(x_S)]$$

where ϕ_j is the marginal contribution of feature *j* on the model output $f_{S\cup\{j\}}(x_{S\cup\{j\}})$. To calculate the feature contributions in a fair way, SHAP keeps fairness properties called additivity, missingness, and consistency (SHAPLEY, 1953; LUNDBERG; LEE, 2017; MOLNAR, 2020).

Additivity means that the sum of the feature contributions together should match the output of *f* for the simplified input x' (which corresponds to the original input x). More formally, SHAP keeps the additivity property as:

$$f(x) = g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j.x'_j \tag{4.1}$$

where g is the explanation model, $x' \in \{0,1\}^M$ is the simplified feature vector, M is the maximum simplified features vector size, and $\phi_j \in R$ is the feature contribution, for a feature *j*, of Shapley values (LUNDBERG; LEE, 2017; MOLNAR, 2020). Here, ϕ_0

represents the expected value with no prior information about the features (similar to the intercept in a regression model). In practice, ϕ_0 is the average of predictions in the training set.

The second fairness property is *missingness*. This is a trivial property, defined as:

$$x'_j = 0 \implies \phi_j = 0 \tag{4.2}$$

This trivial property requires features missing in the original input to have no impact (LUNDBERG; LEE, 2017).

Finally, *consistency* means that if one feature contributes more to the model output, it cannot get a lower Shapley value. It is worth noting the feature contribution calculated by SHAP is the only possible explanation model that satisfies these 3 fairness properties (see the theorem proof in (LUNDBERG; LEE, 2017)).

In more recent work, (LUNDBERG *et al.*, 2020) show that combining many local explanations allows capturing global patterns from the representation of the predictive model whilst retaining local faithfulness to the original model, which can be used for detailed and accurate representations of model behaviour. Figure 21 illustrates the workflow of the SHAP method, which can be used to analyse local feature effects and to combine local explanations of individual predictions, in order to generate global explanations (data insights, model summarisation, collective feature effects).

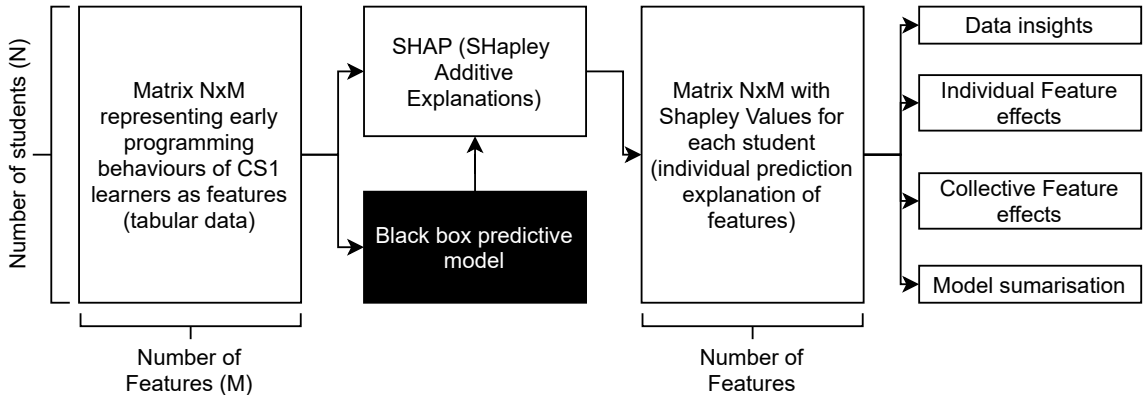

Figure 21 – Workflow of how we used the SHAP method (LUNDBERG *et al.*, 2020) for computing local predictions to create individual and collective explanations of the predictions from our tree-based black-box model.

4.3.1 Machine Learning Models

To develop our predictive model we used the popular eXtreme Gradient Boosting method (XGBoost) (CHEN; GUESTRIN, 2016). XGBoost is an optimised implementation of the Gradient Tree Boosting (GTB) ML algorithm, an ensemble method based on decision trees. Specifically, XGBoost utilises the boosting principle in an iterative way, wherein each iteration the algorithm attempts to correct the errors of the previous iteration, by optimising specific loss functions as well as applying several regularisation techniques. We opted for XGBoost because this model might outperform standard deep learning models on tabular-style databases as ours, in which the attributes are meaningful and they lack strong multiscale temporal or spatial structure (LUNDBERG *et al.*, 2020; CHEN; GUESTRIN, 2016). However, it is important to experiment more machine learning techniques (No-Free-Lunch Theorem - Machine Learning (WOLPERT, 2002)).

In previous chapter, we have composed a set of data-driven features that, in conjunction, have a high predictive power to infer the students' performance when using data from the very first two weeks of course. The model optimised by the EA achieved an average accuracy of 78.2% using data collected in the first two weeks of course, which outperformed cutting edge results for this task (PEREIRA *et al.*, 2019). As an extension, we surpassed the EA (PEREIRA *et al.*, 2019) with an average accuracy of 82.2% using deep learning architecture, as we shown in the previous Chapter. Between the model presented in this current Chapter using XGBoost and our previous best result using deep learning, we did not find statistical significance (p-value>0.05), as we achieved an average accuracy of 81.3%, using XGBoost (the comparison of the ML models is provided in section 4.5). Thus, here we can state that there are no performance drawbacks or advantages in our choice for XGBoost instead of Deep Learning model.

Notice that tree-based ensembles and deep neural network are non-linear techniques that construct complex models, that is, typical black-box models. As our goal is mainly interpretation, we need to 'open' such a black-box to explain the model's decision. To do so, as mentioned in the previous subsection, we used a state-of-the-art unified approach to interpret model predictions, SHAP method (LUNDBERG; LEE,

2017). There are several implementations of SHAP, such as TreeSHAP, which is designed for tree-based models, and KernelSHAP, which is a model-agnostic designed for a variety of ML pipelines, such as deep neural networks. Nonetheless, (BOULANGER; KUMAR, 2020) explain there are several caveats of KernelSHAP such as: i) KernelSHAP requires access to the entire dataset to calculate the Shapley values; ii) KernelSHAP is procedurally slower when calculating Shapley values of large datasets; iii) KernelSHAP ignores feature dependency; iv) using KernelSHAP, the Shapley values are not exactly computed, instead they are only estimated. Indeed, KernelSHAP performs a sampling of features when evaluating the possible subsets $S \subseteq F$. The TreeSHAP implementation solves all of these issues, by calculating the exact Shapley values in polynomial time (see (LUNDBERG *et al.*, 2020)). Thus, as a final justification for our choice for XGBoost instead of deep learning, we used this tree-based model with the TreeSHAP implementation, because it gives us more interpretative power of the models' decision, with no drawbacks with regards to the predictive model performance.

Additionally, please note that to calculate the Shapley values we need to run the predictive model many times with missing features (LUNDBERG; LEE, 2017; LUNDBERG *et al.*, 2020). Thus, there is a need to supply a background dataset (LUNDBERG *et al.*, 2020). In our case, we use the training set as user-supplied background dataset, by relying only on the path coverage information stored in the tree-models, as recommended by the authors of the method (LUNDBERG *et al.*, 2020).

4.4 Method

In this Chapter we applied the same experiment design presented in Chapter 3, using the same instrument, data collection and programming behaviours (features), and data cleaning process. The unique difference is about the machine learning algorithm in which we also used the XGboost ensemble method. More details in the next section.

4.4.1 Prediction and Validation

In this work, as said, we used XGBoost, which has parameters to control the ensemble training, such as the number of trees (n_estimators), as well as parameters to control the growth of trees (e.g., max_depth, min_samples_leaf, etc.). Moreover, an important parameter is the learning rate, which scales the contribution of each tree. To train our model, we used a popular regularisation technique called shrinkage (Géron, 2019), in which we set a low value to the learning rate (e.g. 0.05), and a high number of decision trees (100 estimators). Finally, we used the early stopping technique with at most 100 rounds, meaning that we stopped training when the validation error stops decreasing to avoid overfitting.

As a baseline for our model, we used the deep learning model and the classifier found more promising by a genetic algorithm presented in Chapter 3.

4.5 Results and Discussions

It is intrinsic for XGBoost to automatically select the best feature as the root of the constituting tree of the predictive model (and, similarly, for sub-trees), using this algorithm performs an automatic feature importance analysis. As such, we opted for not using any dimensionality reduction technique at this point.

For each predictive model, we ran the stratified cross-validation 20 times with 10 folds (as recommended in other study (EDWARDS *et al.*, 2020; PEREIRA *et al.*, 2020)), varying the seed in a range from 1 to 20, in order to shuffle the database in different ways, to ensure reliable results. Hence, we report outcomes from the 200 results for each metric (20 times 100, one outcome for each fold). All models were trained using data from the very first two weeks of the course for early prediction.

The results of each method are presented in Table 8, where Random Forest (RF) is the model found by the genetic algorithm (PEREIRA *et al.*, 2019) and DL is the deep learning model (PEREIRA *et al.*, 2020), both explained in the previous Chapter. Our predictive model (XGB) achieved an accuracy ranging from 81.1% to 81.6% (C.L. 95%). Our current model statistically surpasses the RF even with Bonferroni correction

($p - value < 0.05/3$) in all evaluation metrics (accuracy, F1-score, precision and recall). Moreover, although we can see a difference between the performance of our model and the deep learning model, this difference is not statistically significant for the F1-score ($p - value = 0.625$) and recall ($p - value = 0.05$), whilst there are statistical difference for accuracy ($p - value < 0.05/3$) and precision ($p - value < 0.05/3$). Thus, our XGB model surpassed the DL model in terms of precision, whilst was surpassed by the DL model in terms of accuracy, and there is a draw for the other metrics (F1-score and recall). Notice that we are dealing with a database that is a bit unbalanced and, hence, our XGBoost have some advantage since the XGB model achieved higher results for the precision, and accuracy might be misleading even for such subtly unbalanced databases. Moreover, we can also see that our XGBoost model is more stable in terms of accuracy since the standard deviation and Interquartile Range of accuracy are lower than in the DL model. In addition, as explained in section 4.3.1, we opted for the XGBoost because TreeSHAP, which is designed for tree-based models, solved several issues of KernelSHAP, which is designed for kernel models such as deep neural networks.

Table 8 – Comparison of our prediction model and our baselines. C.I. stands for Confidence Interval, L.B. stands for Lower Bound, and U.B stands for Upper Bound.

		Accuracy			Recall			F1-Score			Precision		
		XGB	DL	RF	XGB	DL	RF	XGB	DL	RF	XGB	DL	RF
Mean		0,813	**0,823**	0,797	0,850	**0,860**	0,838	**0,821**	0,818	0,792	**0,794**	0,782	0,751
95% C.I. for Mean	L.B.	0,811	**0,819**	0,794	0,844	**0,854**	0,833	**0,816**	0,814	0,789	**0,791**	0,777	0,747
	U.B.	0,816	**0,827**	0,800	0,857	**0,865**	0,843	**0,826**	0,822	0,795	**0,797**	0,787	0,756
Median		0,809	**0,827**	0,798	0,841	**0,864**	0,835	0,809	**0,821**	0,792	0,797	0,781	0,747
Std. Deviation		**0,018**	0,027	0,022	0,049	0,039	**0,034**	0,033	0,027	**0,022**	0,023	0,037	0,032
Minimum		**0,774**	0,754	0,742	**0,787**	0,739	0,766	**0,775**	0,746	0,746	0,750	0,692	0,684
Maximum		0,839	**0,895**	0,855	0,936	**0,966**	0,915	0,882	**0,888**	0,847	0,835	0,886	0,830
Interquartile Range		**0,031**	0,037	0,030	0,079	0,053	**0,047**	0,047	0,037	**0,028**	0,022	0,052	0,038

4.5.1 Reliability of the XGBoost model

To demonstrate the reliability of our XGBoost model, we analysed its learning curves for an increasing number of instances (students) using 10 fold cross-validation as recommended in a prestigious machine learning book (GÉRON, 2019). We plotted the average cross-validation performance (accuracy, F1-score, recall, and precision) and the standard deviation in the shaded areas of Figure 22. We started with 180

instances and then incremented 400 instances for the cross-validation of our XGBoost. We stopped this process with 1760 as one more increment would exceed our total number of instances. Notice that from 580 instances on, the predictive model achieves a score close to 80% in all performance measures. As such, 580 instances is potentially the number of students needed for convergence, whichendorses that our model can be used even in a lightweight database. Moreover, from a visual inspection of the plots, we can observe that our model generalises well on the validation set, as the continuous lines (red and green) are close to each other, which indicates that our model did not overfit the training set.

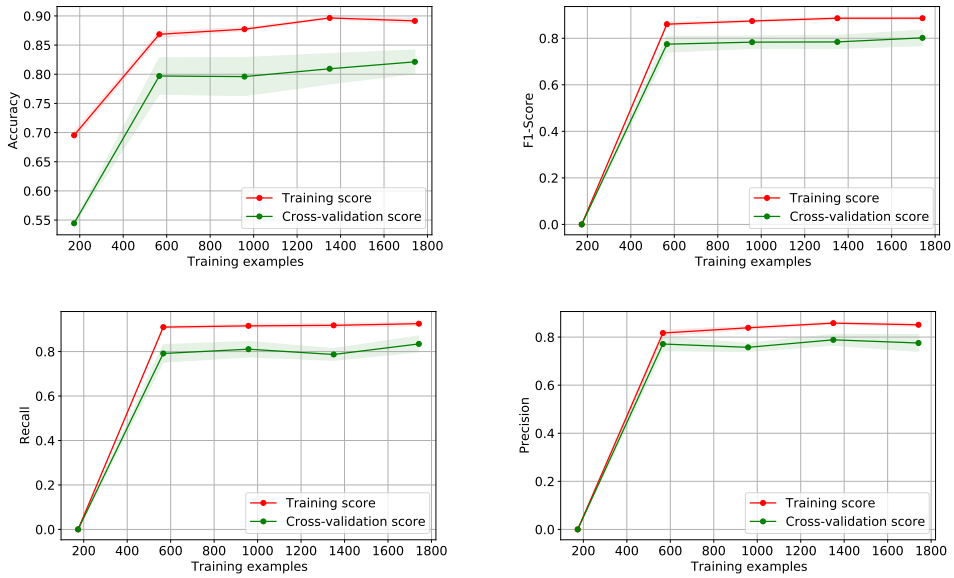

Figure 22 – Learning Curve of our interpretable predictive model for different metrics (accuracy, F1-score, precision and recall) taking into consideration varied number of instances for training.

Moreover, analysing the trade-off between bias and variance we can state that our model potentially found a balance between these errors since the variability around the training score and cross-validation score curves are almost stable from the convergence point and the curves are similar (see Géron (2019)).

Finally, a bit out of the scope of this Chapter, we also carry out experiments using non-early data. We observed that the predictive model become even more accurate throughout the sessions (S1 - S7). The results of the models can be seen in the Appendix A.

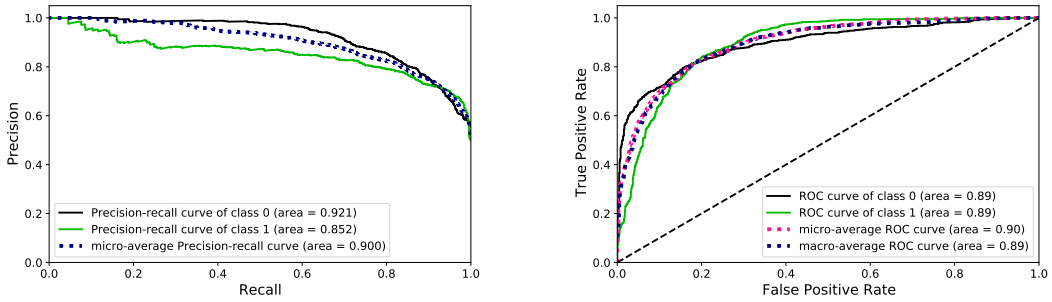

Figure 23 – Precision and recall curve (left) and ROC curve (right) of students who passed (class 1) and students who failed (class 0). The micro-average takes into consideration the class proportion of students who passed and failed, whereas the macro-average threats each class independently.

4.5.2 Interpretation of the predictive model

As previously argued, obtaining a good prediction model is important, but not enough, if its decisions are obscure - especially when working with a vulnerable population such as that of learners. Thus, after ensuring a competitive performance of our model, we show here how to interpret its decisions. The performance of our model ensures we are interpreting a reliable model. Next, we use the SHAP method (as explained above) to explain individual predictions, and what we call *predictions paths* to explain collective behaviours.

4.5.3 Individual analysis

To provide a general idea of how we can evaluate which learner programming behaviours are effective and ineffective, we can inspect graphically the Shapley values of each learner, individually. Figure 24 shows decision plots with coloured lines (vertical), where a light brown line represents an individual prediction of a student that failed in CS1, whilst a dark purple line depicts a student who passed in the course. The students were chosen at random. The x-axis represents the model's output: in this case, the probability of a student passing[2]. The students' coloured line cross the (top) x-axis at our model's predicted probability value. To classify the students, we used as threshold the

[2] In order to calculate the probability of failing we just subtracted: $(1 - p_p)$, with p_p probability of passing.

base value[3], which is approximately 0.46. Hence, if the probability of students passing is higher than this threshold, then they will be classified as passed, otherwise as failed. To illustrate, in Figure 24 (a), our model predicted that the probability of the highlighted student passing is close to 0.05 (5%) and, hence, the student is classified as failed, whilst in Figure 24 (b), the probability of the highlighted learner passing is close to 0.85 (85%), thus classified as passed.

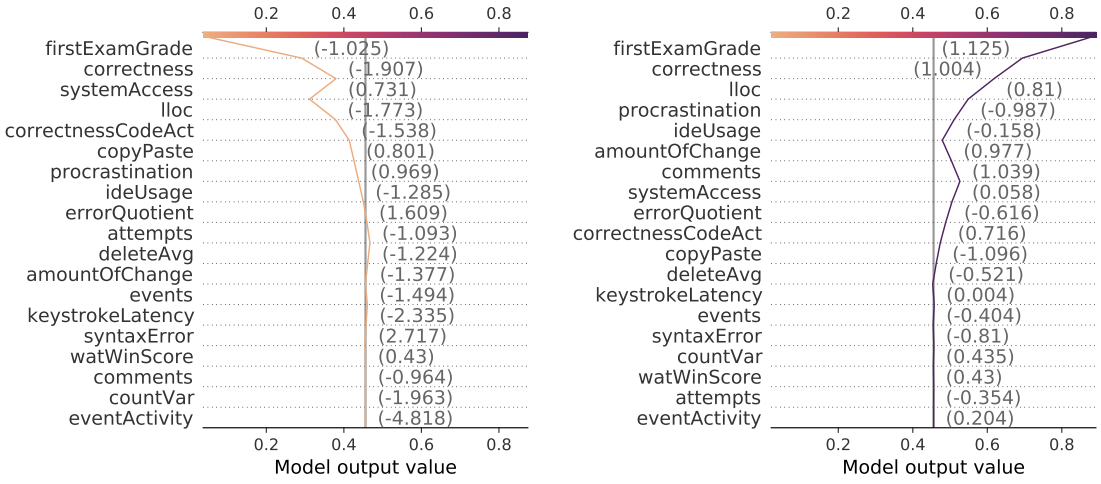

Figure 24 – Decision plots to explain the potential leading factors (early programming behaviours) for passing or failing.

The y-axis of the decision plot lists our features in descending order of importance. Each feature's importance is specific for the student plotted in that particular decision plot. Moreover, the straight vertical grey line marks the model's base value. Finally, from the bottom to the top of the plot, the decision plot shows cumulative Shapley values (feature effects - see section 4.3.1) for each student's programming behaviour, i.e., for a given prediction we show how each student's programming behaviour (feature value) contributes to the overall prediction over the model's base value. We also show the feature values next to the coloured student prediction line, for reference. Remember that the feature values are standardised with the z-score, representing, for each feature, how far a student is from its mean value.

[3] The base value is is the average prediction over the training set. This value represents the overall value that would be predicted if we did not know any features of the current output(LUNDBERG; LEE, 2017).

4.5.3.1 Explaining individual programming behaviours of a learner with a high chance of passing

With information as above, we can perform a deep explanation of individual predictions, i.e., we are able to uncover notable patterns of programming behaviours that can be useful for a better understanding of what might lead to success or failure. In Figure 24 (b), we can see that this student has a high probability of passing ($\approx 85\%$).

Observing the decision plot, we can notice that the features *eventActivity*, *attempts*, *watWinScore*, *countVar*, *syntaxError*, *events*, and *keystrokeLatency* had no effect for this learner. Indeed, feature values that push the prediction higher (effective behaviours) are the learner's moderately low *deleteAvg* (-0.5), low *copyPaste* (-1.1), moderately high *correctnessCodeAct* (0.72), moderately low *errorQuotient* (-0.6), average *systemAccess* (0.1), average *ideUsage* (-0.2), low *procrastination* (-1.0), moderate high *lloc* (0.81), high *correctness* (1.0) and high *firstExamGrade* (1.1). Considering the effective behaviours of this student, we notice that the student deleted parts of their code less frequently than her/his peers, which might indicate that this student is not struggling, or rewriting the code many times. This can also be seen by observing that the negative *errorQuotient* increases the student's chances of passing. Moreover, s/he makes low use of *copyPaste* and, hence, s/he is potentially writing the code from scratch. Finally, s/he achieved a high grade in the first assignment list and exam. On the other hand, the features that push the prediction lower (ineffective behaviours) are the high value of *comments* (1.0) and *amountOfChange* (1.0), which is somewhat unexpected. A high value of *amountOfChange* as ineffective might be explained by the fact that this learner has a moderately low *errorQuotient* and, thus, theoretically, would not need to make many changes between submissions. About the *comments*, it seems to be a hidden pattern that the predictive model uncovers for this learner. That is, a high number of comments in the beginning of the course is not increasing the learner's chances of passing. This may potentially be because the (Python) code required is too easy and brief at this point of the CS1 course (first two weeks), without great need of documentation. Mnemonic variable names might be enough to explain such code.

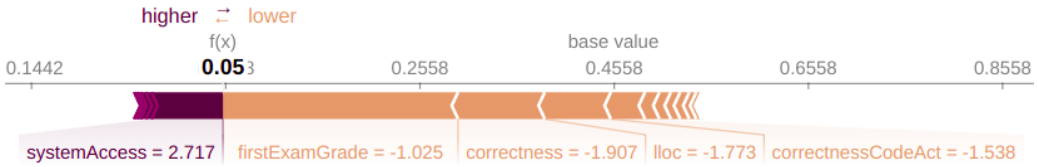

Figure 25 – Force plot of a given student.

4.5.3.2 Explaining individual programming behaviours of a learner with a high chance of failing

Conversely, in Figure 24 (a) we show an example of a student who has a high chance of ending up failing. Overall, this learner has ineffective behaviours, such as a high *errorQuotient* (1.6), low *ideUsage* (-1.3), high *procrastination* (1.0), and moderately high *copyPaste* (0.80). Moreover, s/he has low correctness (-1.9), *correctnessCodeAct* (-1.5), and *firstExamGrade* (-1.0). As an effective behaviour, s/he accesses the system more than the average: *systemAccess* = *0.7*. Thus, we can assume that this learner expends little effort in trying to solve the problems from the assignment. Some indicators of that are the low *ideUsage*, high *procrastination*, high *copyPaste*, and low *correctnessCodeAct*.

We can also visualise an individual explanation of the model prediction as a force plot (LUNDBERG *et al.*, 2018), presented in Figure 25. Similarly to the decision plot, the force plot presents a prediction for a student (here, chosen at random). The $f(x)$ function is the model output (the predicted probability for that student), and the base value follows the same reasoning of the decision plot (average of model predictions). The features that push the prediction higher are shown in dark purple, whilst the ones which push the prediction lower are in light brown. To be more meaningful, the dark purple features are right arrows, whereas the light brown ones are left arrows. The arrow's size represents the effect of that feature. Given that, from a visual inspection of Figure 25, we can observe that the leading factors that are pushing the prediction lower are that s/he has a low *firstExamGrade* (-1.02), *correctness* (-1.9) , *lloc* (-1.7), and *correctnessCodeAct* (-1.5). However, similar to the learner from Figure 24 (a), s/he has a high *systemAccess* (2.7) value, which is slightly increasing the learners' chance of passing.

Based on such individual explanations, we can generate automatic, customised, fine-grained suggestions to a student; or provide this detailed information to the teacher, who can use it in talking (face to face) with the student, to encourage the learner to better

use her/his potential; e.g., guiding them towards being more hardworking - by solving more problems from scratch, and not too close to the deadline. As another example where there is room for improvement, some learners are copying and pasting more than 1 standard deviation above the average, which might not be a desirable behaviour for a novice student in the first two weeks of the course. Instead, it is expected that students solve problems from scratch, to practice more, as recommended by the testing effect theory (ROWLAND, 2014), which explains the role of effortful processing as a contributor to the achievement.

Notice that although the force plot might seems more intuitive for interpretation, it is useful only for a few features, while the decision plot can present a large number of features effects clearly. Moreover, in a decision plot, we can visualise multi-output predictions, as we show in the next subsection, which allows detecting some prediction paths.

4.5.4 Small group analysis (Passed versus Failed)

After an individual analysis of student behaviours, we join 10 low-achieving students who failed, in Figure 26 (a), and 10 high-achievers students, who passed, in Figure 26 (b), to inspect patterns related to the predictions. All students were chosen at random. Such local explanations can be useful, as building-blocks for global insights. Here we notice that the failing students have a similar trajectory (prediction paths), that is, their learning lines are relatively close for many features, which shows a similarity in their programming behaviours. However, we can see some exceptions. To illustrate, we can observe a student who crossed the margin line, which suggests that this learner was performing well towards passing the CS1 course, e.g., s/he did not procrastinate too much, accessed the online judge (*systemAccess*) regularly, with a medium number of events and *eventActivity*. Nonetheless, s/he made many mistakes while submitting the code (see *errorQuotient*, *watWinScore* and *syntaxError*) and solved a lower number of problems from the assignments (lower *correctness*), and then, perhaps for some an unknown reason (extraneous variables), ended up failing.

Another observation for prediction paths of the successful students (Figure 26 (b)) is that we note two divisions in the plot: (i) the first is for students who did not struggle too much, which is illustrated by the lines which have often been above the vertical line margin; (ii) the other students have encountered higher difficulty, but they were still successful. The lines of these students are on both sides of the margin line.

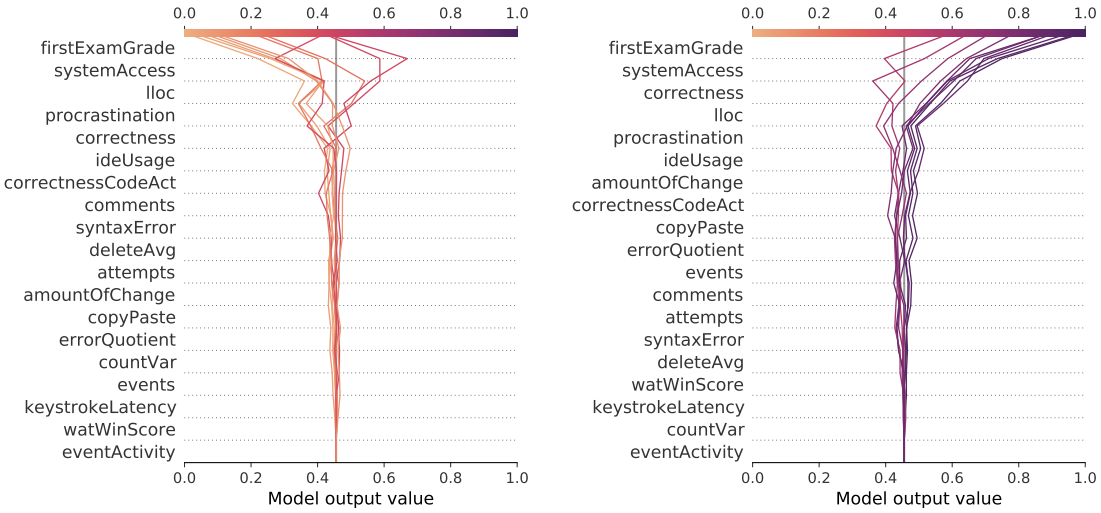

Figure 26 – Prediction paths of 10 learners who failed (left) and 10 learners who passed (right).

From this small sample of learners, although we can observe similarity in prediction paths of the successful and unsuccessful learners, there are some nuances in the behaviours that might lead to success or failure in this cohort. In the next subsection we will evaluate these nuances in the prediction paths more holistically, taking into consideration almost the entire dataset, instead of a small sample of learners.

4.5.5 Prediction paths

To evaluate possible prediction paths, we cluster the Shapley values of all learners using the well-known k-means algorithm. We use the knee point detection algorithm (SATOPAA *et al.*, 2011) to automatically find the potential optimal number of clusters. The metrics used to evaluate the maximum curvature point (knee point) (SATOPAA *et al.*, 2011) were the *mean silhouette score* and *inertia*, as recommended in (GÉRON, 2019). After running the k-means algorithm with k ranging from 2 to 10, we found that 5 is

the most suitable number of clusters. In other words, we found five different prediction paths, represented by five behavioural patterns that might lead to success and failure, which are shown in Figure 27. These decision plots show the centroids of each cluster. Notice that the centroids in k-means are the averages of the instances inside a cluster. As such, the centroids in this case depict the overall Shapley values (*feature effect*) of the learners in each cluster. In Figure 27, we keep the same feature order in the decision plots, to make it easier to compare the different prediction paths.

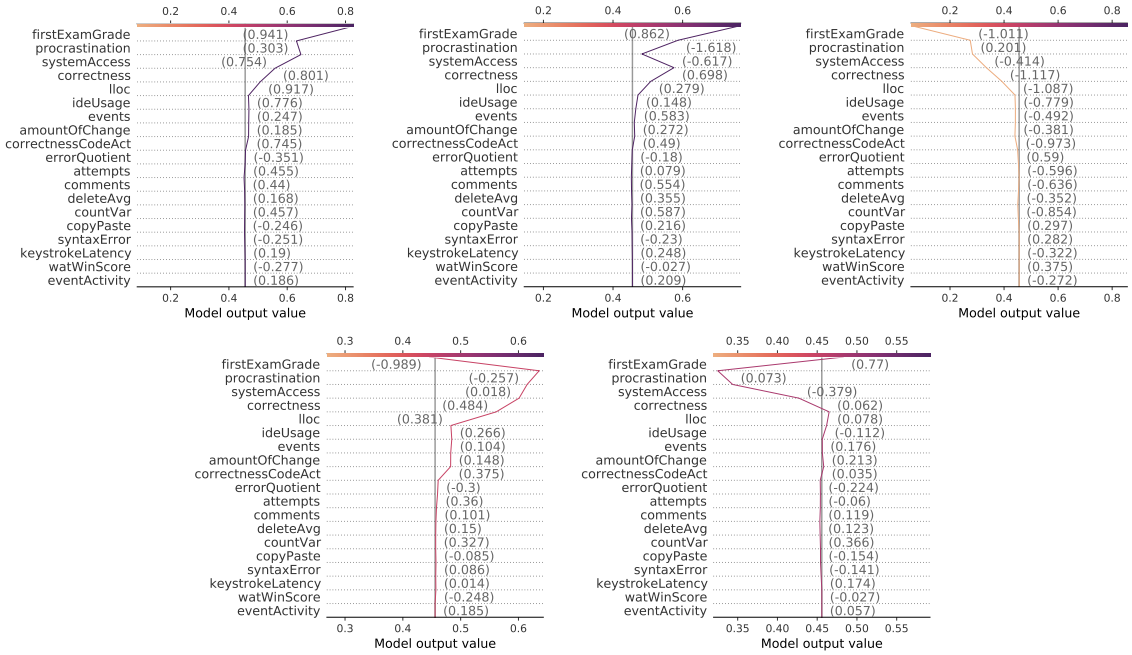

Figure 27 – Cluster centroids of prediction paths of our model prediction for a better understanding of effective and ineffective behaviours. Here the subfigures' labels are counted from left to the right, up to down, beginning from a) until e).

Following, we give a brief description of each prediction path that we found (see Figure 27):

- Prediction path 1: students with high chances of passing and who have mostly effective behaviours. They may also have some minor ineffective behaviours.
- Prediction path 2: students with moderate to high chances of passing, who have mostly effective behaviours, but with a different pattern than prediction path 1.
- Prediction path 3: students with a high chance of failing and who have mostly ineffective behaviours. They may also have some slightly effective behaviours.

- Prediction path 4: students whose chances of passing are uncertain. In general, their chances are a bit lower then the base value and, hence, they are borderline cases, potentially unsuccessful. Indeed, these students have moderately effective behaviours; however, they achieved a low grade in the first exam.

- Prediction path 5: students whose chances of passing are uncertain. Their chances are generally a little higher than the base value, and hence, similar to the prediction path 4. They are borderline cases; however, potentially successful ones. Indeed, whilst these learners have some moderate effective and ineffective behaviours, they have a high first exam grade.

Approximately 30.02% of the students follow the prediction path 1, 8.32% follow the prediction path 2, 34.85% follow the prediction path 3, 13.32% follow the prediction path 4, and 13.49% follow the prediction path 5. In other words, 38.34% (30.02% + 8.32%) of the learners have high chances of passing, 34.85% have high chances of failing, and 26.81% (13.32% + 13.49%) are borderline cases, for which the prediction model predicts with moderate to high level of uncertainty.

For a better understanding of the prediction paths, we analyse the effective and ineffective behaviours present in each plot from Figure 27. In Figure 27 (a), we can inspect that the learners from this cluster likely made less common errors (e.g., *syntaxError* = -0.25), dealt with the errors better (*errorQuotient* = -0.35) and tended to spend less time to fix errors (*watWinScore* = -0.28), which is a sign that these learners were not struggling to solve the problems. Moreover, they used *copyPaste* (*copyPaste* = -0.25) moderately, which is important for a novice learner. In spite of the importance of knowing that, these behaviours from this cluster have low effect (low Shapley value) in the model's decision. Indeed, the programming behaviours that have highest impact for this prediction path are the fact that the learners from this cluster solved most of the problems from the assignment (*correctness* = 0.80 and *correctnessCodeAct* = 0.76), and spent more than the average time coding in the IDE (*ideUsage* = 0.78). Moreover, the students accessed the system regularly and achieved a moderate to high grade in the first exam (*firstExamGrade* = 0.94). These effective behaviours are the potential explanation of why such learners had a probability of passing (close to 80%). The programming

behaviours *events* (0.25) and *amountOfChange* (0.19) have also some minor impact in the model's decision. The average value of these features is somewhat expected, as these effective learners do not make many errors, even having a moderate to high number of attempts (0.46) and correctness (0.80). As a counterexample, a learner who had a moderate to high number of attempts, who solved many problems, and who submitted many code snippets with errors, should have changed her/his code a lot to fix problems, which would have generated many log events. Finally, it is expected that these learners have a low value of *procrastination*, so that there is still room for improvement for the students from this cluster.

In Figure 27 (b) we can observe a similar prediction path (see the trajectory of the coloured line) showed in Figure 27 (a). That is, the learners from this cluster have also high chances of passing, potentially because of similar reasons to the learners who follow prediction path 1. The main difference is that these learners (that follow prediction path 2) have a lower value of *systemAccess* (-0.62) and procrastination (-1.62). However, such a moderately low value of *systemAccess* is likely a positive indicator for this prediction path. Indeed, as the learners solved most of the problems ($correctness = 0.70$), with a low value of *procrastination*, this suggests them solving problems from the assignment as soon as the instructors made them available. Hence, after that, they did not keep accessing the online judge, as they had already finished their assignment.

On the other hand, Figure 27 (c) shows the students who follow the third prediction path. The students from this cluster have more than the average number of code errors (*errorQuotient* = 0.59), and they were not dealing well with the errors (*watWinScore* = 0.38). That is, they potentially were not trying to fix the problems (see the low *amountOfChange* = -0.38), which might explain why they achieved low grades in the first assignment (*correctness* = -1.12) and exam (*firstExamGrade* = -1.01). Additionally, based on the number of attempts (*attempts* = -0.6) and time spent to solve problems (*ideUsage* = -0.78), we can deduce that these learners are neither effective nor resilient in trying to fix code errors. These ineffective behaviours are potential explanations of why the students from this cluster have their average probabilities of passing close to 10%.

Figure 27 (d), the forth prediction path, are learners with a similar trajectory to

those following prediction path 1. The main differences are twofold. Firstly, the feature values from this cluster are almost half of those that follow the first prediction path. Overall, they solved half of the questions from the assignments, they have half of the *lloc* value, they spent half of the time that learners from the first cluster spent in solving problems. Secondly, these learners might have some moderate effective behaviours, but achieved a low grade in the first exam (*firstExamGrade* = -0.99). This discrepancy is resulting in the uncertainty of the model for these cases. Indeed, the second reason (low *firstExamGrade*) has a high impact on the model's decision and changed the direction of the prediction to the left, decreasing the learners' chances of passing.

The learners that follow the prediction path 5 (Figure 27 (e)) have average values for almost all programming behaviours, which makes the trajectory of the prediction (looking from the bottom to the top of the plot) close to the grey vertical line (base value). Indeed, the direction starts to change from programming behaviours *ideUsage* and *lloc*, which increase somewhat the chances of passing. This indicates that average values of these 2 features might be effective behaviours for this prediction path. Still, an average *correctness* associated with a moderate to low *systemAccess* and an average *procrastination* is decreasing the learners' chances of passing. A possible reason why an average *correctness* is potentially an ineffective behaviour is that the first assignment has only easy problems and, thus, many learners solved all the questions (for more details, see the *correctness* statistical analysis in Appendix A). Moreover, a moderate/average *procrastination*[4], without a high correctness, might not seem as an effective behaviour. However, unexpectedly, as an inflection point, a moderate to high *firstExamGrade* changed the direction of this prediction path to the right, raising the overall chances of these learners above the base value. A possible explanation is that these students did not access the IDE regularly (*systemAccess* = -0.38) and may not have solved all the exercises from the programming assignments, not because of lack of knowledge, but because they may already have known programming, i.e., they might have had contact with programming before the CS1 course. Another possibility is simply because of plagiarism in the exam. Notice that this kind of behaviour might confuse

[4] Notice that a moderate procrastination means that the learner started solving the problems between 4 or 5 days before the deadline - as better explained in our Appendix II.

the predictive model and bring about false negatives. Such outliers are interesting in themselves to find, to analyse separately (ideally, by an instructor) as they may have quite distinct needs from the rest of the cohort.

Finally, for a deeper analysis of each feature importance based on our model prediction, we make available a link[5] with interactive plots. The shared folder has 10 HTML files with plots for each fold tested in this study. The plots are a combination of individual force plots, rotated 90 degrees and stacked horizontally, and ordered by similarity of SHAP explanation, using the cluster analyses. To illustrate, in Figure 28 we show the first 1000 instances of the first fold (cross-validation). The bold value on the y-axis shows the probability to pass of the student in position 896. Similarly to the explained force plot, the feature values in purple represent a positive effect and the light brown ones a negative effect for this individual student. With such an interactive plot on-hand, the stakeholders (instructors, monitors, coordinators, etc.) can preventively evaluate which behaviour should be stimulated and which should be improved upon, for each student and for groups of learners, since the plot is sorted by similar Shapley values.

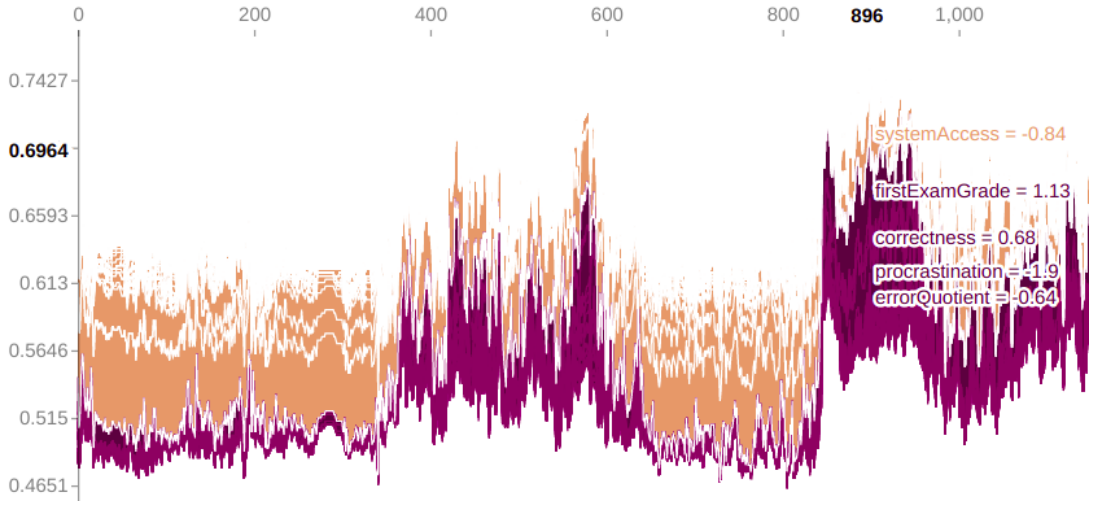

Figure 28 – Prediction forceplots rotated 90 degrees, stacked horizontally. Here we highlight the 896th instance, showing her/his feature values and contributions.

[5] bit.ly/2PVCCaP

4.5.6 Global Analysis

Regarding the importance of the features, Figure 29 (a) presents a bar chart with the average impact (mean of Shapley values) for each feature, in terms of model output magnitude. As arguably expected, the three most relevant are *firstExamGrade*, *systemAccess*, and *correctness*. This translates into the conclusion that if the learner performs badly in the first exam and in our programming assignment lists, a 'red flag' needs raised. Still, it is important to monitor how regularly students are accessing the online judge, as the number of accesses (*systemAccess*) plays an important role at the beginning of the course. Moreover, the number of logical lines of code (*lloc*) matters in the solution submitted, as *lloc* is the forth most important feature. A potential reason is that the total *lloc* of the solutions sent by the learners for all problems of the first assignment might have an expected value, and the predictive model potentially uncovered the likelihood of the expected value that might be effective or ineffective. Still, we can see in the plot that *procrastination* might be an undesirable behaviour for some students and can influence their performance negatively. Finally, as found by Pereira et al. (PEREIRA *et al.*, 2020), spending more time solving problems (*ideUsage*) and being resilient are positive behaviours. Here, resilience might be measured by the association of *attempts*, *ideUsage*, *lloc*, *errorQuotient*, *syntaxError*, *amountOfChange*, and *watWinScore*. That is, even when the solutions are not correct at first (*errorQuotient*, *syntaxError*), it is important to spend qualitative time (*ideUsage*) trying to fix the error (*attempts*, *amountOfChange*, *watWinScore*) more than once. Notice that such attempts will increase the *lloc* and *countVar*, as these features compute the total number of logical lines of code and variables (respectively) in all submissions, regardless whether accepted or not.

Additionally, it is important to note that the feature effects might be different for different students. To illustrate, whilst general procrastination is associated negatively with performance (STEEL, 2007), this effect might be less pronounced or even reversed for some students. In this sense, Figure 29 (b) presents the direction and the distribution of the feature effect. For some features, there are some medium to long tails, meaning that those features might have low global importance, but a high relevance for specific instances. To illustrate, *systemAccess* has a higher total model impact than *procrastination*.

Nonetheless, for the instances in which *procrastination* plays an important role (long tail), it has more impact than systemAccess. Thus, *procrastination* impacts a few predictions, by a large amount; whilst *systemAccess* affects almost all predictions, by a smaller amount.

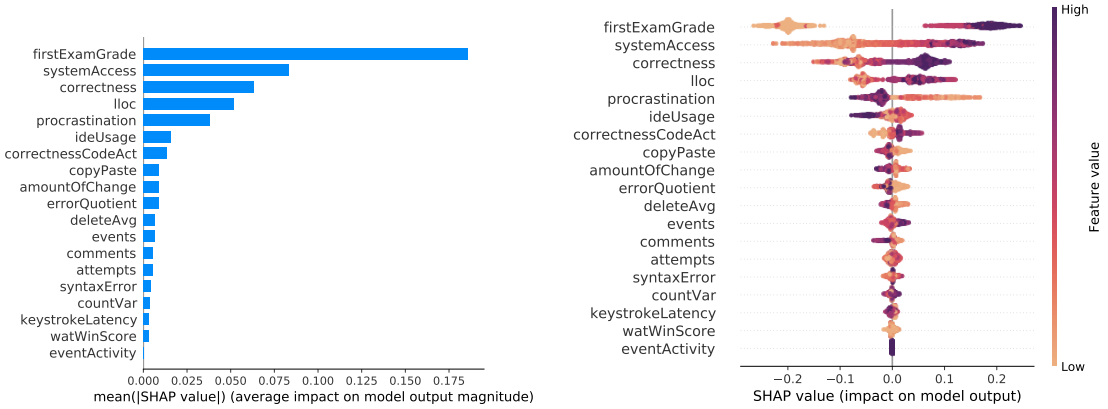

Figure 29 – Summary of features' importance for the model's decision.

4.6 Pedagogical Implications

Our work enriches the research on programming learning with findings of effective and ineffective early students behaviours (currently considered an open question (ROBINS, 2019; CARTER *et al.*, 2019; QUILLE; BERGIN, 2019; PEREIRA *et al.*, 2020a)), and the educational data mining field, with an accurate and explainable ML pipeline that can be useful for early intervention and student self-regulation.

An important finding from our approach is the notion that for different learners, a different set of predictors seem to have an impact on successful learning. As we demonstrated above, even generally undesirable behaviours, like *procrastination* (STEEL, 2007) might be more-, or less-harmful, for a particular person. As psychological and educational research typically applies linear modelling (GUASTELLO *et al.*, 2008), such a complex nonlinear interplay has remained undiscovered by prior research applying traditional methods.

Regarding the different features used for prediction in our analysis, we need to emphasise that there are some behaviours that are easier to modify than others.

Whilst it is possible to instruct students to avoid procrastination and increase total time investment (BELLHäUSER *et al.*, 2016), keystroke latency or the number of deleted characters are less suitable for interventions.

For a more generalist analysis for adaptation of instructional decisions, we presented the power of global explanation by the identification and analysis of typical prediction paths. Moreover, our focus not only on global behaviours, but also on individual ones, enabled by visualising and analysing feature effects at single-student granularity level, can be used in an unprecedented variety of pedagogical applications. Indeed, this early prediction, empowered by its explanation, might potentially allow an effective early intervention by stakeholders. To illustrate, our interactive force plots (Figure 28) of each student might be shown to the instructors at the end of the second week of the course, who in turn might create some proactive way of minimising the chances of at-risk students ending up failing. What is more, as the plot is sorted by similar Shapley values, student behaviours might be grouped, for recommendation purposes.

Notice that, in CS1 classes, each student may have a different timing to learn to program. However, in traditional non-personalised classes, all students are treated in the same way. Ideally, students should be challenged to learn as much as they can, taking into consideration their individual learning weaknesses and strengths. For example, a student that solves tasks fast and effortlessly, may be bored and potentially frustrated. One possible solution for that is creating more challenging tasks for the students with high probability of passing. More specifically, more challenging problems may be recommended for students who have low procrastination, solve all the exercises on the assignment, and access the system regularly. Hence, traditional Intelligent Tutoring Systems or Adaptive Educational Hypermedia rule-based approaches (STASH *et al.*, 2004; BROWN *et al.*, 2005) can be combined with modern educational data mining and SHAP-based processing for large-scale personalised education.

In addition, for instructors, managers, and educators, a visualisation dashboard, including our force plots, decision plots (and so forth) might contribute for a more formative assessment. As such, not only the learner product is evaluated, but also

the process behind, that is, not only their codes are evaluated, but also their learning paths and effort to produce their codes. Formative feedback might be sent to students, to improve student attainment, in response to a request from the literature for such works (KEEFER *et al.*, 2014; CRIŞAN, 2017; MENÉNDEZ *et al.*, 2019). For example, an automatic notification might be sent to each learner, showing them their own force plots with their most important behaviours that should be encouraged, and the ones which need improved upon. An individual decision plot might be also sent to students for self-reflection of all analysed behaviours. This would empower the students to better guide their own study. Such metacognitive strategies, which get the students to think about their own learning, have been proven efficient in many areas of education. Indeed, (ROWLAND, 2014) showed that metacognitive strategies may be worth the equivalent of an additional 7x times greater progress than that used in a traditional environment. The study explains that the major reason for such progress is that the learners were aware of their strengths and weaknesses, which motivated them to engage in and improve their learning. Such metacognitive or self-regulatory strategies can also be trained via web-based training (BELLHäUSER *et al.*, 2016).

Furthermore, this dashboard can increase the chances of the instructor reflecting and diagnosing potential causes of the students' lack of success. For example, an instructor might explore in the dashboard a plot like in Figure 28, where forceplots are clustered. Using that AI-based information combined with classroom experience, the instructor might schedule a meeting with specific groups, to discuss how certain programming behaviours they are having are potentially jeopardising their learning. In other words, this could amplify the instructor's ability to implement effective interventions.

Indeed, based on such a dashboard we can intervene on many design dimensions (CARTER *et al.*, 2019), such as providing to the learners AI-based information, critique, suggestions, and encouragement. Such intervention content might be shown in our dashboard visually, or through text notification, with the intention of positively affecting their programming behaviours, learning process and outcomes. Moreover, students might also explore visually and interactively their learning process and progress. A dashboard might allow triggered intervention as well, in which a notification or plot

might to be shown to the learner in response to her/his actions. Another option would be performing intervention on demand, in which the learner must explicitly require some feedback or suggestion on effective and ineffective behaviours.

Finally, another possibility for interventions is to use the log file data for group formation. As heterogeneity has been proven as beneficial for collaborative learning in many scenarios (MÜLLER *et al.*, 2021), one pedagogical approach would be to form groups of learners with force plots differing from each other.

4.7 Chapter Conclusions

In this Chapter, we developed an explainable ML pipeline that competes in performance with current state-of-the-art (inexplicable) black-box models. We have also shown that there are significant benefits in using fine-grained data-driven code metrics to extract features using insightful algorithms, since this allows, besides predicting student performance early, to analyse behaviours that are related to struggling and successful students. Moreover, we trained our model using data from the first two weeks of classes, allowing early intervention.

For replication purposes, we provide our fine-grained dataset[6]. Moreover, for works that want to replicate our work but use only globally relevant features, the most important features for *early prediction* were *firstExamGrade*, *systemAccess*, *correctness*, *lloc*, *procrastination*, *ideUsage*, *correctnessCodeAct*, and *copyPaste* (see Figure 29 (a)). This translates into: if some students perform badly in the first exam and in the early programming assignments, by procrastinating, do not spend appropriate time solving the problems, then a 'red flag' needs raised, as it has likely negative consequences for the students' final performance. Furthermore, we have shown also the local impact of features for each individual student, where less important features could have high relevance for some learners (Figure 29 (b)). As such, researchers that want to replicate this work could consider this local importance of features, additionally to the global one.

[6] Our dataset can be found on codebench.icomp.ufam.edu.br/dataset/

Additionally, our high-performance predictive model is explainable, which can facilitate human/AI collaboration towards prescriptive analysis, where the instructors/monitors will have access to individual and collective analysis on which student behaviours should be encouraged and which ones should be inhibited. On the student side, such analysis can promote self-regulation and awareness of their strengths and their chances for improvement. To illustrate the usefulness of the approach from a student's point of view, they may trust more on a recommendation if they understand why they received it. From the instructors' and coordinators' side, understanding why students are failing or passing would allow them to apply effective efforts to tailor pedagogical material, instructions and interventions for future classes.

Nonetheless, besides that, more prescription needs to be done in order to support students and instructors. For the instructors side, although the benefits that our method can bring online judges to support CS1 classes, these systems also bring about a workload increase for instructors, since there are many repetitive and laborious tasks to feed the system for different classes and terms. To illustrate, following we point out some: a) selecting problems to create assignments lists and exams; b) creating different problems assignments for distinct classes during terms to avoid plagiarism; c) categorizing these problems in topics presented on the methodological CS1 curriculum. All these tasks are important to allow the methodology and resources adaptation and dynamism in order to personalise the system for different knowledge domain during the courses and terms.

Moreover, some students like solving problems outside the assignments created by the instructors. That is, such students like performing a self-direct learning (ZHAO *et al.*, 2018). However, due to the overload of problems available in online judge systems, learners might feel frustrated in searching problems adequate to their knowledge. As such, in the next Chapter we will go a step further in terms of prescription by proposing and validating a recommender system to support CS1 classes using the knowledge we obtained so far in this Chapter and in previous. Indeed, we will use the programming profile based on effective and ineffective behaviours to help learners to solve adequate problems and to help instructors selecting problems to compose assignments and exams.

5

Recommending Problems for Students based on Effective and Ineffective Behaviours

> This I call to mind and therefore
> I have hope.
>
> - Lamentations 3:21

5.1 Overview of the Chapter

We showed in previous Chapters how fine-grained data collected from online judges can be used to depict effective and ineffective behaviours and how those behaviours can be used as predictors of students performance in programming courses, enabling proactive interventions from stakeholders and potential prescriptions. Here we use this knowledge to deal with a particular problem in online judges: learners in general, and novices in particular, typically struggle to find problems in online judges that are adequate for their capabilities and programming skills. A potential reason is that online judges present problems with varied categories and difficulty levels, which may cause a cognitive overload due to the large amount of information (and choice) presented to the student. Thus, students can often feel less capable, which may result in affective effects,

such as frustration and demotivation, decreasing their performance and potentially leading to increasing dropout rates. This problem occurs specifically when students are performing self-direct learning, what is quite common in online judges. On the other hand, when students are guided in the learning process by the instructors, these educators need to select adequate problems to compose the assignments and exams, what takes time, since typically those instructors needs to create variation of the assignments for different classes and terms to avoid plagiarism. Recently, the literature has presented systems to recommend problems in online judges, however, using a non-granular analysis of student data, that do not take into consideration the students' effective and ineffective behaviours, achievement. In this sense, *in this Chapter we proposes for the first time, to the best of our knowledge, a prescriptive analytics solution for students' programming behaviour* by constructing and evaluating an automatic recommender module based on students' effective and ineffective behaviours to personalise the problems presented to the learner in online judges in order to improve the learners achievement, whilst minimising negative affective states, and to help instructors in selecting problems to comprise the assignments and exams. To evaluate our recommendations, we compare our Behavioural-based Recommender System (BRS) with a Random Recommender System (RRS), which simulates typical human selection of problems in online judges. Such comparison was conducted through a double-blind control experiment to verify the impact on student's programming achievement, motivational affect and effort employed to solve the recommended problems. Results showed that our method significantly maximised positive affective states, whereas minimising the negatives ones. Moreover, BRS significantly outperformed typical human selection (simulated via RRS; $p < 0.05$) in terms of effort and achievement (correct solutions) and reduced dropout and failure in the problem-solving process.

5.1.1 Practitioner Notes

What is already known about this topic:

- Fine-grained data collected from online judge can be used to depict effective and

ineffective behaviours and as early predictors of student performance (PEREIRA *et al.*, 2020; PEREIRA *et al.*, 2020; PEREIRA *et al.*, 2021).

- Few methods to recommend problems in online judge available (TOLEDO *et al.*, 2018; FANTOZZI; LAURA, 2020).
- Studies have shown recommender systems for online judges using only non-granular features (attempts and correctness) (YERA; MARTÍNEZ, 2017; PEREIRA *et al.*, 2021a; SAITO; WATANOBE, 2020).

What this Chapter adds:

- Showing how our fine-grained features can represent the effort expected to solve programming problems.
- A novel behavioral recommender system based on students' expected effort to solve a given problem.
- Showing, through a double-blind controlled experiment, empirical evidence on how personalised recommendations based on effort influence achievement and affective states.
- Showing potential implication and application of our recommender system for learners and instructors.

Implications for practice and/or policy:

- Potential application for self-direct learners who will easily find problems more adequate to their knowledge level and skills.
- reducing negative affective states in the task of searching for adequate questions and in trying to solve inappropriate problems.

5.2 Research Question Addressed in this Chapter

The adoption of OJ environments by instructors and institutions has increased in the last few years (YU *et al.*, 2015; WASIK *et al.*, 2018; ZHAO *et al.*, 2018; SAITO; WATANOBE,

2020). Despite the notorious benefits of OJs in education, these systems are not able to recommend the appropriate problems for the students, which may impact affective perception, leading to affective states such as frustration, over time (RODRIGO; BAKER, 2009; YU *et al.*, 2015; TOLEDO *et al.*, 2018; CHAU *et al.*, 2017; SETTLE *et al.*, 2015; LUXTON-REILLY *et al.*, 2018). Frustration has been shown to be directly related to the amount of effort a student needs to spend to solve a problem and may even lead to dropout (RODRIGO; BAKER, 2009; NGAI *et al.*, 2010; LEE *et al.*, 2011; FORD; PARNIN, 2015). This happens due to effort being intrinsically related to the students' confidence, competence and consequently affecting their motivation (KELLER, 2009; DECI; RYAN, 2010).

The amount of effort a student puts into a task is also tied to their motivational experience (e.g., reaching the flow state proposed by (CSIKSZENTMIHALYI; CSIKSZENTMIHALYI, 1992)) where their effort is appropriate to the task these students are doing. Indeed, a good balance of effort required to solve tasks is related to an increase in achievement (DUCKWORTH *et al.*, 2015). In this sense, it is important to measure the students' effort in those environments and show how this effort is related to their personalised programming tasks, affective states, and achievement.

To adapt the programming problems to the students' effort, recommender systems appear as a viable solution (MANOUSELIS *et al.*, 2011; KULKARNI *et al.*, 2020; JÚNIOR; PEREIRA, 2020; JÚNIOR *et al.*, 2020). Recommender systems (RS) are environments used to identify and provide content based on rules that use user data (RICCI *et al.*, 2011). These systems have been widely used in educational scenarios; however, few studies have tackled ways to provide recommendations based on a deep analysis of user behaviours (RIVERA *et al.*, 2018; SAITO; WATANOBE, 2020; KULKARNI *et al.*, 2020; JÚNIOR *et al.*, 2020). More specifically in the scope of programming learning, there are only a few studies available in the literature proposing methods to recommend problems in OJs, and such studies make the recommendations only based on students' attempts and results from the submissions to the OJ. Notice that a deep behavioural analysis of fine grained data is crucial to make appropriate recommendations (CUI *et al.*, 2016; KULKARNI *et al.*, 2020). Moreover, effort and its implications for affective states

and achievement should be taken into consideration in programming learning (RODRIGO; BAKER, 2009; KINNUNEN; SIMON, 2010; EDWARDS; LI, 2016; UMAPATHY; RITZHAUPT, 2017; FWA, 2018; IM; KANG, 2019; RANGEL *et al.*, 2020).

We showed in the previous Chapters that our fine-grained data is useful to depict effective and ineffective behaviours and that those behaviours are related to resilience and achievement (e.g. *finalGrade*). Notice that resilience and effort are intrinsically related. Proag (2014) explains that resilience may be measured as the effort required to do something. Indeed, we showed in previous Chapters (in Chapter 2, specifically) that effective students are those who employ a good balance of effort in problem-solving and make progress in learning to program, typically leading to successful outcomes. We showed that thorough an analysis of a set of features that we called *programming profile*.

As such, in this work, besides the variables previous used in previous works (attempts and results from submissions), we also employed our programming profile to represent the effort expected to solve a given problem. Using these features, we make a recommendation based on the following hypothesis: if a student s solves a given problem p, and our method recommends a problem p' that requires an effort to be solved similar to that of p, then the student s is able to solve the problem p'. Using that, we believe that the recommendations will minimise students' negative affective factors, whilst maximising the positive ones, as the problems recommended will not require a disproportionate effort from the learners. In addition, as aforementioned, effort is related to the students achievement (DUCKWORTH *et al.*, 2015; PEREIRA *et al.*, 2020). Hence, our second hypothesis is that our recommendation based on expected effort will increase the student achievement and decrease dropout and failure rate in problem-solving. Thus, in this Chapter we aim at solving the following research question:

- *RQ4-1) How to recommend adequate problems for students and instructors based on effective and ineffective behaviours?*

5.3 Affective states whilst learning to program

The several programming problems available in OJ allow students to test themselves (e.g., writes and submits a code to receive feedback on its correctness) multiple times. This practice can be seen as self-testing, an approach beneficial to learning (e.g., improving long-term knowledge retention) known as testing effect (ROWLAND, 2014; GOLDSTEIN, 2014; ROEDIGER-III; KARPICKE, 2006). This context demonstrates the value that the OJ has to those who are learning to program, indicating the importance of keeping learners engaged with these systems.

However, according to (KINNUNEN; SIMON, 2010; LUXTON-REILLY *et al.*, 2018), introductory programming students present high levels of anxiety and frustration among any other disciplines in computer science courses. Frustration[1] likely leads to many negative outcomes, such as deterioration of students' self-esteem, disengagement, poor learning outcomes and retention, and may even lead to dropout (RODRIGO; BAKER, 2009; KINNUNEN; SIMON, 2010). Consequently, minimising learners' frustration from interacting with OJ is necessary to prevent that they quit using these systems.

Whilst frustration is a negative affective state for learning, having students satisfied with their performance, as well as pleased with the material, likely works in the opposed way. (D'MELLO *et al.*, 2010) consider feeling this way as happiness, and reasons for this affective state contributing to learning might lie in attending basic human needs. That is, as learners feel satisfied with their performance, they likely have their competence need satisfied (DECI; RYAN, 2010), which contributes to high-quality learning experiences (DICHEV *et al.*, 2014; TODA *et al.*, 2020). Therefore, working towards maximising students' happiness whilst using OJs is another approach relevant to improve learning.

Based on this context, OJs should strive to prevent learners from feeling frustrated, at the same time they should seek to enhance their happiness. Frustration can

[1] In the scope of this work, we consider frustration when an effort disproportionate to what the student is used to is required for him/her to solve a certain problem. In other words, the student needs to solve a problem that is too difficult or too easy for his/her level of knowledge, that is, his/her learning expectations have not been met (D'MELLO *et al.*, 2010).

be mitigated through the students' effort[2] (KELLER, 2009). When applying the tailored amount of effort to a certain activity, students may become engaged and motivated by achieving what they wanted, which leads to a better learning experience (LEE *et al.*, 2011). Hence, mitigating frustration and the negative outcomes that derives from it may help with satisfaction with their performance (i.e., happiness).

As such, recommender systems present a viable solution to handle students' feelings of frustration and happiness through effort. These systems provide contents based on given inputs, such as students' skill level. Henceforth, by understanding and mapping students' efforts in programming courses, RS can suggest the best tasks that are tailored to students' level of knowledge, which will likely lead them to better learning experiences and achievement. Consequently, this will contribute i) to improving students' happiness, as they will feel satisfied with their performance by achieving what they expected and by receiving learning materials aligned to their level of knowledge, and ii) to mitigating frustration by providing assignments tailored to the amount of effort needed. Finally, those systems can help instructors in selecting adequate problems to create assignments and exams.

5.4 Recommender Systems in Online Judges - State-of-the-art

In recent years, coding automatic feedback has been used to aid in programming teaching and competitions (TOLEDO; MOTA, 2014; ANSARI *et al.*, 2016; YERA; MARTÍNEZ, 2017; DWAN *et al.*, 2017; CHAU *et al.*, 2017; PEREIRA *et al.*, 2019a; PEREIRA *et al.*, 2019b; PEREIRA *et al.*, 2019; SAITO; WATANOBE, 2020; FANTOZZI; LAURA, 2020; OLIVEIRA *et al.*, 2020; PEREIRA *et al.*, 2020a; FILHO *et al.*, 2020; LIMA *et al.*, 2021a). Since then, students have embraced the advantages of self-learning through online judges, which led to the emergence of a new research direction related to better understanding students' needs and interaction as well as designing and building better OJ systems

[2] In the scope of this work, effort is defined as the learner's work employed to try to solve a given problem (DUCKWORTH *et al.*, 2015).

(CHEN, 2009; WASIK *et al.*, 2018; PEREIRA *et al.*, 2020b; SAITO; WATANOBE, 2020; FANTOZZI; LAURA, 2020; MELO *et al.*, 2021; PESSOA *et al.*, 2021). However, there is a lack of studies exploring the role of effort and how personalised recommendations may have impact on effective states and achievement.

Hosseini & Brusilovsky (2017), Chau *et al.* (2017) proposed to extract information from codes to choose the suggestions for students learning programming. Hosseini & Brusilovsky (2017) created an RS that provided hints during the solving process, using techniques as *term frequency–inverse document frequency* to represent similarities. Chau *et al.* (2017) created an RS that suggested learning materials to help teachers in designing courses. Different from Hosseini & Brusilovsky (2017) and Chau *et al.* (2017) who focused on tips about learning materials, we used data-driven behaviours analysis to infer the knowledge and effort of the students based on the data logs generated through real time execution.

Indeed, data-driven behaviour analysis have been a trend in the past few years. Many studies analysed a set of metrics that can estimate the students' behavioural patterns in OJs, such as: number of attempts, IDE usage, number of errors, average lines of codes, etc. (JADUD, 2006; WATSON *et al.*, 2013; ESTEY; COADY, 2016; AHADI *et al.*, 2016; OTERO *et al.*, 2016; LEINONEN *et al.*, 2016; CASTRO-WUNSCH *et al.*, 2017; DWAN *et al.*, 2017; LEEUWEN *et al.*, 2019; CARTER *et al.*, 2019; QUILLE; BERGIN, 2019; LOPES *et al.*, 2019; MARGULIEUX *et al.*, 2020). All these works showed that these fine grained behavioural features are related to students' achievement/performance. Nonetheless, none of these studies analysed the features' in conjunction neither how they can be used to automatic recommending problems. Notice that you do both in this work since we analysed the features in conjunction in previous Chapter and now we are focusing on recommendations.

Moreover, in the direction of recommendations, Toledo & Mota (2014),Yera & Martínez (2017),Toledo *et al.* (2018) and Saito & Watanobe (2020) used learner behavioural data for automatic recommendation of problems in online judges. Toledo & Mota (2014),Yera & Martínez (2017) and Toledo *et al.* (2018) proposed RS that recommend problems based on a collaborative approach. They used a binary matrix as a

basis for the recommender. Saito & Watanobe (2020) proposed a learning path recommendation system based on learner's submission history in an online judge. However, these authors Toledo & Mota (2014), Yera & Martínez (2017), Toledo *et al.* (2018), Saito & Watanobe (2020) consider only the number of attempts and results from the submissions as features. In this work, we extend the metrics that are used in previous works to the ones shown in our programming profile (Table 2) and others (it will be shown next in Table 9) that were extracted from the codes submitted and log data collected from CodeBench. Moreover, different from all of them, we evaluated our method with real learners and checked their affective factors, achievement rates, and effort employed when solving the problems.

In brief, none of the previous studies performed an analyses of effort considering such fine-grained set of features to design a behavioural recommender model, as a way to personalise the recommendations in online judges. In this study, our RS provides problems adapted to the students' skills, and we measured the resulting achievement rate, effort employed, and affective states (frustration, happiness and confusion) when solving our recommended problems.

5.5 Methods

As in previous Chapters we use real interaction fine-grained data from students in CodeBench. Thus, the instruments and data collection process are the same as explained in the previous Chapters, except that we now use the data from the entire courses, instead of the first two weeks, as we are interested in recommending problems instead of early predictions.

5.5.1 Feature Extraction to Represent Learner's Effort

Effort is a psychological construct and, therefore, there is no standard way to measure it. In other words, there is no pattern scale to put students that will give a direct and precise measure of how effortful they are in solving problems. (HADEN, 2019) explains that in

this cases, there is a need of features that indirectly measure the construct. That is, we need to define some observable, recordable measure that we believe accurately reflects the construct, what is called the operational definition of the construct. As such, we have established an operational definition to compute the expected effort to solve a given problem, using features that have already proved to be efficient in the literature and code metrics established in software engineering to measure the effort of programmers in the development process, as well as the features we showed in previous Chapters to be good predictors of achievement and relevant measures of efficient behaviours.

Features	***Description***
countCicle	Average number of loops from submitted students' codes for a given problem;
countCondition	Average number of conditional structures from submitted students' codes for a given problem;
cyclomaticComplexity	Average cyclomatic Complexity from submitted students' codes for a given problem, where cyclomatic Complexity represent the source code as a control flow graph, corresponding to the number of independent paths of this graph (OTERO *et al.*, 2016);
nDistinctOperands	Average arithmetic operands in the source codes;
nDistinctOperators	Average arithmetic operators in the source codes;
test	Average number of times the student tested the source code (DWAN *et al.*, 2017);
totalOperands	Total of operands in the source codes (LEINONEN *et al.*, 2016; OTERO *et al.*, 2016);
totalOperators	Total operators present in the source codes (LEINONEN *et al.*, 2016; OTERO *et al.*, 2016);

Table 9 – Data-driven features to represent student's effort to solve programming problems.

As explained in previous Chapters, we extracted these features from the students' logs and further processed them, e.g., extracting the average number of students' attempts for each problem, average number of lines of codes for each problem, etc. Thus, our observable, recordable measure to represent the effort is presented in Table 2 (Chapter 2). Moreover, we extended this set of features to represent also behaviours that are important after the beginning of the course, such as the number of cycles which are

not covered in the first weeks of CS1 courses. Thus, we also added features presented in Table 9, which are inspired by previous works (OTERO *et al.*, 2016; LEINONEN *et al.*, 2016; DWAN *et al.*, 2017) in the field, as they have been proven to be tied with students' achievement and in the software metrics (FENTON; BIEMAN, 2014) from software engineering field. Still, notice that achievement is completely related to effort, as explained by (DUCKWORTH *et al.*, 2015). Therefore, here we are using these features to measure the expected effort the students need to employ to solve a given problem to improve learners' achievement rate, in a move towards minimising students' frustration and maximising satisfaction (i.e. happiness) in the learning process.

We also adapted other variables presented in Chapter 2, where we propose the concept of resilience, which is related to the number of attempts a student takes to solve a given problem. We added all these variables to the analysis to measure the student effort in-depth. For a better understanding, following is the list of features used in our BRS: *procrastination*, *amountOfChange*, *attempts*, *lloc*, *systemAccess*, *events*, *correctness*, *syntaxError*, *ideUsage*, *keystrokeLatency*, *errorQuotient*, *watWinScore*, *countVar*, *contCicle*, *contCondition*, *cyclomaticComplexity*, *nDistinctOperands*, *test*, *totalOperands*, *totalOperators*, *noAttempts*, *unsucNoRes*, *sucNoRes*, *unsucRes*, *sucRes*, *effectAttRate*, *effectGenRate*. Notice that now we aggregate these features per problem and use data from the entire curse instead of first weeks.

Concerning the data transformation, we used a widely established statistic measure, the z-score, to standardise the features in each problem (MANSOURY *et al.*, 2019).

5.5.2 Behaviour-based Recommender System

The architecture of the BRS is presented in Figure 30, in which the system recommends problems that contain similarities to the current problem solved by the student. The features indicate the behaviour each student presents when using the IDE in the OJ. This set of features forms the input data-driven behaviour model, which can be used to represent the students' effort. The similarity between the recommended problem and

the target problem is made through nearest neighbour analysis, using cosine similarity as distance metric. We use this technique to support our first hypothesis that is if a student *s* solves a given problem *p* (which we call a target problem), and our method recommends a problem *p'* that requires an effort to be solved similar to that of *p*, then the student *s* is able to solve the problem *p'*, and, hence, *p'* is adequate to the learner's level of knowledge. As such, the nearest neighbour analysis is playing a role of matching the target and recommended problem by analysing the problems' similarities.

With nearest neighbour technique, each group of questions (solved by the student and contained in the system) is represented through an array and the similarity is the calculation of the correlations between these arrays (the angle between them, always ranging from 0 to 180). Furthermore, cosine arc is used to transform the result into a value between 0 and 180 degrees. In this sense, given two arrays P and E, the cosine between them is given by the product of $P \times E$ divided by the product of l_2-$norm$ of P and E, given by $\sqrt{\sum_{k=1}^{n} {X_k}^2}$, where $X = \{P, E\}$. This way, we guarantee that our results generated through the similarity of the cosine are always normalised, with values ranging from 0 to 1. The similarity formula is thus:

$$S_{PE} = \frac{P \cdot E}{\|P\| \, \|E\|} = \frac{\sum_{i=1}^{n} P_i \times E_i}{\sqrt{\sum_{i=1}^{n} P_i^2} \times \sqrt{\sum_{i=1}^{n} E_i^2}} \tag{5.1}$$

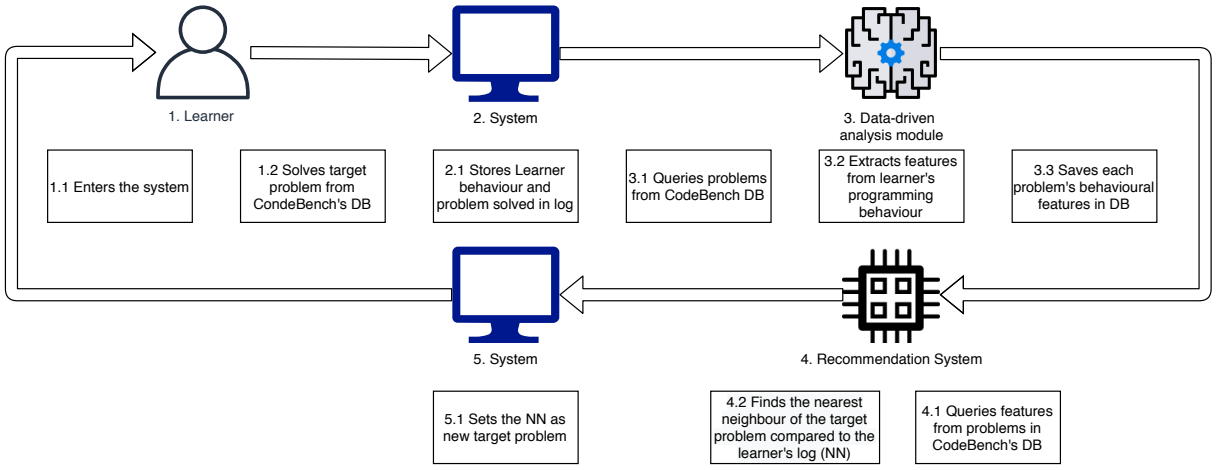

Figure 30 – Workflow of Behavioural Based Recommender System

5.5.3 Evaluation of the Recommender Model

5.5.3.1 Participants

As a proof of concept we evaluated our method in a double-blind controlled experiment. For the evaluation, we recruited students who had already done introductory programming, from the Federal University of Roraima (UFRR), Brazil, due to convenience sampling. We have written a message to all mailing computer science students from UFRR, explaining our research goals, asking for volunteers to participate in a 10-minute phone call, scheduling calls for all who replied.

Before starting the evaluation, we have explained the study to the learners and obtained their consent to participate. In total, 15 students agreed to participate and each of them solved 16 problems, 12 recommendations and 4 target problems[3], totalling 180 recommendations (12x15) to be evaluated.

Affective State	**Description**
Boredom	Uninterested in the current recommended problem.
Confusion	Poor comprehension of the problem, attempts to resolve erroneous belief.
Engagement	Student motivated to solve the current problem recommended.
Neutral	No visible affect, at a state of homeostasis.
Frustration	Problem recommended was not as expected, that is, more difficult or easier.
Happiness	Satisfaction with the recommendation, feelings of pleasure about the problem.

Table 10 – Affective states used to evaluate learner's comments

5.5.3.2 Measures

For each recommended problem, we asked the participants to make a comment about the effort required to solve the target problem and the recommended problem. Thus, we evaluate the affective states of the comments based on the most frequent affective states when solving problems (D'MELLO; CALVO, 2013), which are boredom, confusion,

[3] Target problems do not count as a recommendation

engagement, neutral, and frustration. Besides that, in our context it is also crucial to evaluate when the learner is satisfied with the recommendation. Therefore, we included happiness (EKMAN, 1992; JÚNIOR *et al.*, 2019), as (D'MELLO *et al.*, 2010) explain that happiness is a typical affective state presented when students are satisfied when solving problems. Thus, we showed a description of each used affective state in Table 10.

We further use a data-driven approach to evaluate the recommendations in terms of effort employed and achievement, following the procedures:

- For achievement, we perform an analysis of correct and incorrect submitted answers, and problems which the students did not try to solve;
- For effort employed, we perform a feature analysis of the data log from the submitted recommendations' solutions. In this case, we adopted the metrics presented in Table 9, which we found that better represent the effort required by the student to develop a solution for the problem. These metrics were: number of attempts (attempts), average usage time in the IDE (IDEusage), average of successful submissions (successAverage), average log rows (events), and length of codes (lloc). We opted to use only those five instead of all features for a simpler analysis (Occam's razor) after some discussion between authors.

5.5.3.3 Experimental Manipulation

To apply our RS, we elaborated lists of recommendations, which means, two lists with eight problems totalling 16 problems per student. Each student has his/her own personalised list divided in groups, as shown in Figure 31. Each group contains four problems, first group is composed of easy problems and second one of intermediate problems. First question of each group is a target problem (TP_1 and TP_2) that was selected by the authors of this study, in collaboration with lecturers and professors of programming. These target problems act as a starting point to balance the RS to generate the recommendations. After the target problems, we have sequenced 3 recommendations, as shown in Figure 31. Thus, we construct the list of problems to the participants using the following sequence: $TP_1 \rightarrow R_1, R_2, R_3$ and $TP_2 \rightarrow R_4, R_5, R_6$.

It is worth to mention that the target problems were not included as part of the recommendation. For the easy target problems we chose sequential and conditional problems (if...then...else), whereas for the intermediate problems we selected problems that use repetition structure (loops), vectors, strings and matrices. To personalise the recommendation of each student, we selected five different target problems for TP_1 and TP_2. Moreover, we calculated the 10 nearest neighbours of each target problem. After that, we randomly assigned 3 out of 10 nearest neighbours of a given target problem to compose its recommendations (Figure 31).

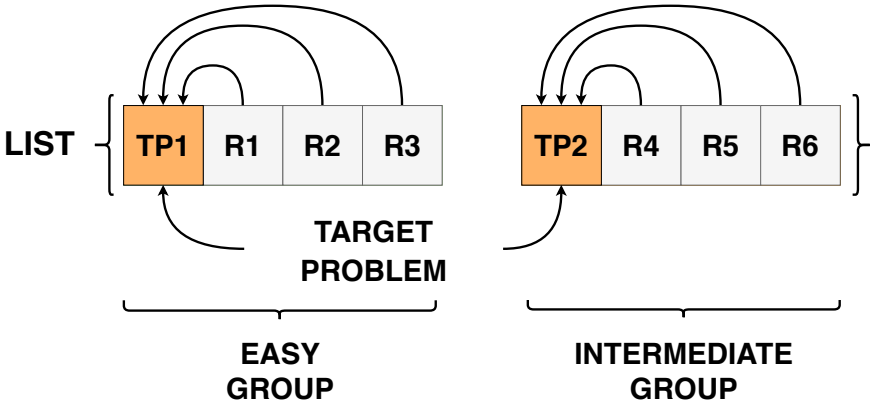

Figure 31 – Representation of a recommendation list

To evaluate the BRS itself (experimental treatment) we compared the personalised recommendation with a random recommender (control treatment). We called Random Recommender System (RRS) because the input is known but not the output, which means that given a target problem, the RRS recommends the next problem(s) by performing a random selection of questions from pre-determined lists of problems selected by instructors. These pre-determined lists of problems might vary in terms of topic, that is, there are assignments lists that comprise problems of variables, conditionals, loops, vectors, and simple matrices operation. As such, the random method works as a baseline (or a placebo) to check the recommendation impact of our BRS. (ZHAO *et al.*, 2018) explain that, in OJs, learners need to find problems spread across multiple volumes of questions with varying topics and difficulty levels. Thus, the RRS simulates the way self-direct learners select problem in OJs, as these systems do not provide automatic recommendation of problems. In other words, the learner needs to

find the next problem to solve randomly searching in the volume of problems until finding a suitable one.

The comparison between the BRS and RRS was conducted through a within-subject experiment double-blind controlled, where students did not know which treatment (BRS or RRS) they were using neither the authors knew which student was receiving which treatment. We first applied the BRS and after the RRS, so that, the participants would have the recommendation from the BRS as reference to classify the recommendation for the RRS. However, as the evaluation was double-blinded and, hence, the participants have not known about that.

5.6 Results and Discussions

In this section, we present our results regarding to the affective states, effort employed, achievement, failure and dropout rate of the students when they were solving the problems that we recommended on our BRS and our baseline, the RRS.

First, we performed a qualitative analysis of the students' comments[4] over the recommended problems to identify the affective states. To perform that analysis, two authors independently classified[5] each comment based on the affective states presented in Table 10. Subsequently, we performed Kappa Cohen Test (COHEN, 1960) to check the agreement level and, as a result, we achieved 0.83, which is considered a high level of agreement (ARTSTEIN; POESIO, 2008). For the cases of disagreement, another author acted as the third judge. Using this classification, Figure 32 shows the affective states presented in the comments about the recommendation of each method. Comparing the methods, we can see a clear difference in terms of happiness and frustration. Indeed, the difference is statistically significant ($p - value < 0.05$, χ_2 - even after Bonferroni correction), which reveals that our method maximise the positive affective state (happiness which are related with satisfaction), whist minimising

[4] The original comments are in portuguese. We, thus, translated it using google translate API for python. The comments can fe found on https://www.dropbox.com/s/gbi37va72lqgvub/comments_english.xlsx?dl=0

[5] Remembering that the evaluation was double-blinded and, hence, authors have not known if the comments were from BRS nor from RRS.

frustration.

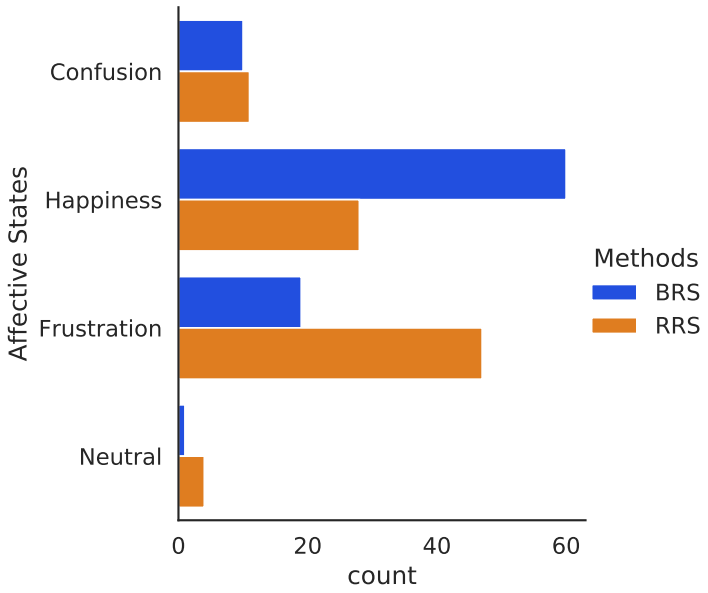

Figure 32 – Affective states classified based on learners' comments.

Analysing each affective state in isolation, we observe only few neutral comments, which makes sense, since the students' comments about the recommendations tend to be pragmatic, that is, they usually stated that they were satisfied with the recommendation (happiness) or that the recommendations did not require the same effort as the target problems (frustration). Indeed, neutral comments only occurred in cases where the students did not show any visible affective state. Moreover, we can observe that boredom and engagement was not assigned for any comment. A possible reason is that the students may not have experienced an aversive state to the activity nor have felt sufficiently engaged as they treated the recommendations as an experiment and not as an usual learning activity.

Another affective state that occurred with a relatively low frequency (N = 21) was confusion. In total, there were 11 cases of confusion in the RRS and 10 cases in the BRS, which reveals a balance in relation to this state. This affective state occurred when students did not understand the problem statement or the way in which the outputs of their codes should be presented in order to pass in all test cases. To illustrate, in some comments, students reported that their codes were correct, but the OJ did not judge them right because, apparently, the test cases were pointing out an error that they could

not find. Wasik *et al.* (2018) state that this phenomenon can happen, as test cases are analysed by comparing strings, so if the student forgets a line break in a *print* command, then the problem can be assessed as wrong, even if logic being right. Nonetheless, notice that this is a limitation of the way in which OJs assess exercises and not a limitation of our recommender method. Similarly, a poorly designed question (which can cause confusion) is out of the scope of our method. Still, these cases of confusion may have slightly influenced the failure rate and dropout in both methods, however, as there are almost the same number of confusion cases in both methods, such influence potentially weighed equally.

In terms of happiness, there are 28 cases in our baseline, whereas 60 in our method, more than double. In addition, there were 47 cases of frustration in the RRS, whilst 19 in our method. This is a first evidence that our method is mitigating frustration, whilst maximising the students' satisfaction (i.e. happiness), what support our belief about our first hypothesis that the recommendations will minimise students' negative affective factors, whilst maximising the positive ones, as the problems recommended will not require a disproportionate effort from the learners.

After this qualitative analysis of the student affective states, we analyzed the effort employed by students using the following metrics: i) achievement rate, which is the proportion of correct submissions over all submissions; ii) failure rate, which is the proportion of incorrect over all submissions; and iii) dropout rate, which is the proportion of recommended problems that were not attempted by the students over all problems. It is worth mentioning that students were free to execute and submit solutions to the recommended problems as they wished, i.e., there was no limit of attempts to solve the recommended problems.

Figure 33 shows the results for each method in terms of three rates: achievement, failure and dropout. When the students solved the problems recommended by the BRS, 60% of their submissions were assessed as correct (achievement rate), whereas using the RRS, the achievement rate was of only 25%. In terms of failure rate, only 14% of students' solutions were not accepted within the BRS, against 24% within the RRS. Indeed, these differences are statistically significant ($p - value < 0.05$ - $\chi^2 test$, even after

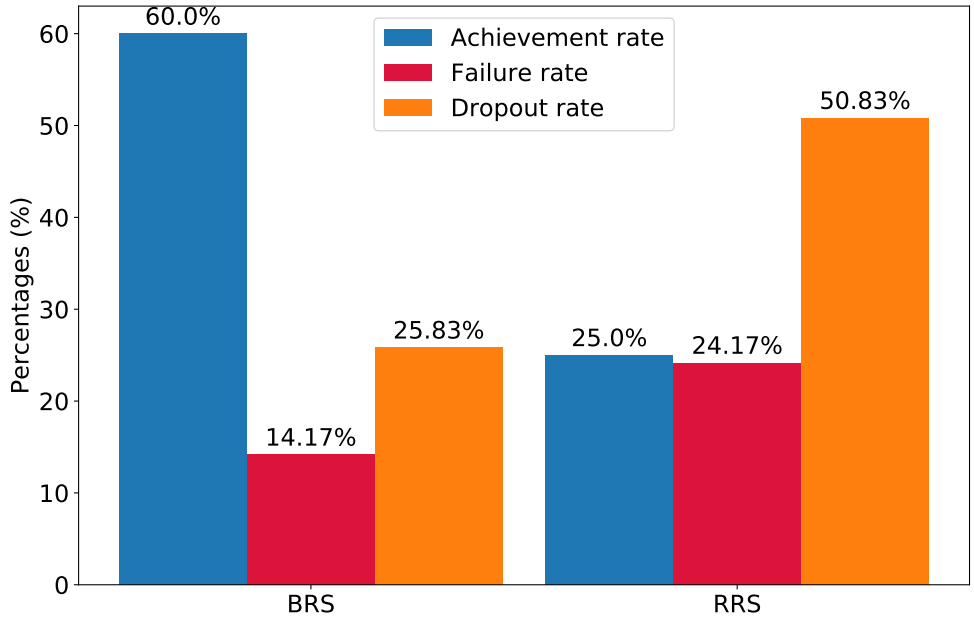

Figure 33 – Analysis of the recommendations submitted to the OJ.

Bonferroni correction). Thus, these results may reveal that in the RRS, the effort required to solve these problems is much higher than they used to employ in the target problems. Furthermore, this is an evidence found that suggests the importance of recommending problems that are more appropriate to the students' efforts so as not to lead them to a low achievement rate and, hence, to frustration. Still, something worth to note is that in the RRS, students have a high rate of untried problems (51% of dropout rate), whereas in the BRS the learners had only 26%. These findings support our second hypothesis that recommendation based on effort expected will increase the student achievement and decrease dropout and failure rate.

About the difference in terms of dropout and failure rate, we can state that this is another confirmation that the problems recommended by the RRS either required more effort from learners or were more complex to the point where the students did not even try to solve them. Such high dropout rate from the RRS is a clear evidence of students' frustration in trying to solve problems not adequate to the effort they apply in the target problems. Other reasons that may have led the student to not try may be either the lack of understanding of the problem or the lack of skills (confusion). Nonetheless, the target problems of each method are, respectively, an easy and intermediate problem. As such, if the recommender system works well, the first group of recommendation should

comprise only easy problems, and the second group of recommendations should contain only intermediate problems (see Figure 31). Consequently, the lack of skills to solve the problem should not be presented as all the students who solved the recommended problems (in both methods) had already done introductory programming and are able to solve easy and intermediate problems. So, what likely happened was some bad recommendations in both systems, proportional to the students' dropout rate and failure rate. Notice that, BRS was statistically superior in terms of achievement rate, failure rate and dropout rate, which likely means that the recommendation of the BRS were more suitable to students effort.

In addition, it is also important to analyse the recommendations from other perspectives. As such, we also examined the students codes submitted for each recommendation and the learners' logs when they were solving the recommended problems in the embedded IDE of CodeBench. This approach makes it possible to investigate students' effort in a more holistically way to build solutions with a formative evaluation in which we are not only inspecting the code submitted by the student but also the process behind this code, which is when the learner is building the solution. More specifically, we conducted such data-driven approach by checking the number of *attempts*, *IDEUsage*, *events*, *sloc* and *successAverage* on both recommenders.

After comparing all these features for each recommender, we found statistical significance even after Bonferroni correction ($p - value < 0.05$ - Mann Whitney Test), as presented in Table 11. Furthermore, for a better visualisation of this difference we plotted the mean of each one in the radar plot presented on Figure 33. Before plotting, we normalised the features using *MinMaxScaler* technique to better see the differences among all features.

Through a visual inspection in the Figure 34 regarding to the random recommender, we can observe that the students spent more time programming in the IDE (greater solution time, $p - value < 0.05$) at the same time that more line of logs were generated (greater *events* - $p - value < 0.05$) whilst they were solving the problems. This might suggest that learners were 'rewriting' the code more on the RRS than on our BRS. Moreover, the codes submitted by the students in the recommendations of the RRS were

smaller (lower value of *sloc* - $p - value < 0.05$), which also indicates that students had to employ much more effort (greater *IDEUsage*, *events*, and *attempts* - $p - value < 0.05$) to submit solutions with fewer lines of code.

	attempts	IDEUsage	events	sloc	sucess_average
Mann-Whitney U	5232.000	5339.000	5528.500	5696.000	4666.500
Z	-3.991	-3.494	-3.135	-2.835	-5.446
Asymp. Sig. (2-tailed)	.000	.000	.002	.005	.000

Table 11 – Results of the Mann-Whitney test in the distribution of the characteristics of the recommendation systems.

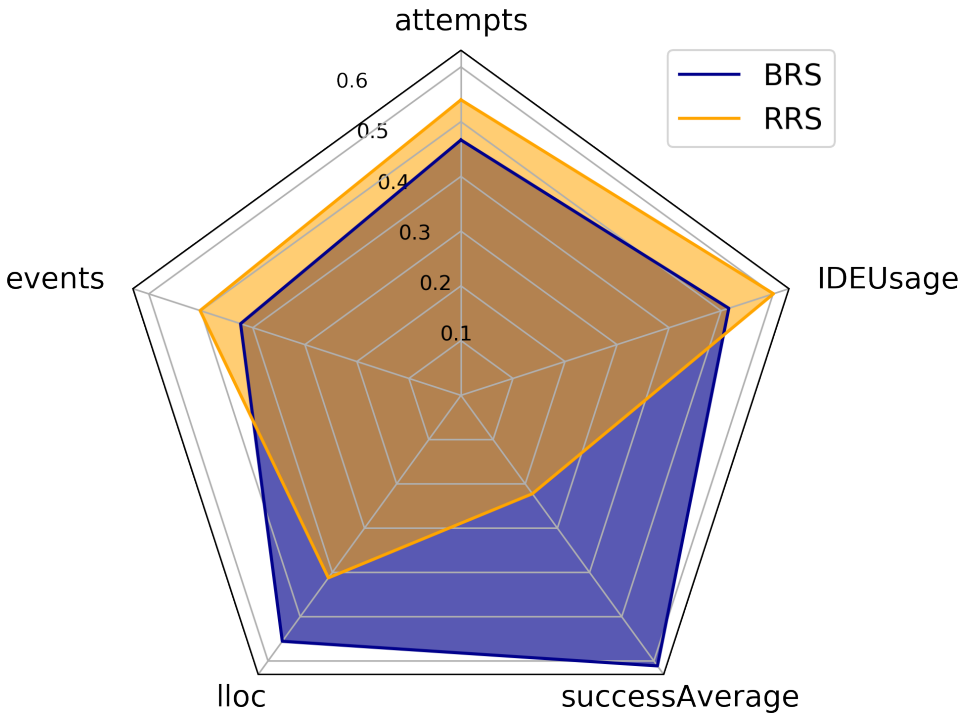

Figure 34 – Analysis of *logs* obtained in the evaluation of programming students.

Furthermore, in the RRS the students had more submissions (greater number of *attempts* - $p - value < 0.05$) at the same time they had a lower success rate when solving the recommended problems (lower successAverage - $p - value < 0.05$). This is another evidence that students tried harder to solve the problems recommended by the RRS.

In summary, we showed in Figure 33 and Figure 34 and through our analysis, that our BRS is superior to the RRS. Indeed in our BRS, students had a lower dropout rate

when trying to solve problems. In addition, the learners' logs suggest that they employed a more proportional effort to solve recommended problems and that the problems were more adequate to their programming skills. Providing adequate problems contributes to maintaining them testing their solutions in the OJ, which likely enhances their learning based on the testing effect (see (ROWLAND, 2014)). Notice that such adaptation and personalisation of learning might lead to minising the chances of students becoming frustrated and even dropping out of the course.

The evidences we found in our qualitative analysis associated with what we found analysing the achievement rate and students logs supports our hypotheses that, in general, our recommendations require similar level of effort regarding to the target problem. Indeed, the higher level of frustration in our baseline is potentially the driven factor that lead to such a high dropout and failure rate, whereas the higher rate of happiness might be related to the high achievement rate in our method, supporting, thus, our second hypothesis that the affective states influences achievement, defined here in terms of lower failure and dropout and higher number of problems solved.

With regards to the evaluation of the recommended problems, it should be taken in consideration that human responses may be subject to bias, as it is difficult to control human attitudes and behaviour, even in a controlled experiment. Thus, consider that the way we evaluated our recommendation method was designed to reduce potential biases. That is, besides the comments analysis, we also evaluated the students' interaction with the OJ and the problem solving process.

5.7 Pedagogical Implications

There are five main implications from our findings. First, students felt less frustration and more happiness when completing assignments recommended by the behavioural RS. Compared to when using the RRS, students' feedback concerning the BRS demonstrate they largely perceived the assignments with more feelings of happiness and less feelings of frustration. This finding suggests using our proposed RS contributed to minimising students' frustration whilst maximising their satisfaction with the mate-

rial (happiness). As mitigating and improving frustration and happiness, respectively, is important to improve learning outcomes, this finding implies the use of our proposed recommendation approach holds the potential to enhance learning experiences in solving programming assignments.

Second, we found that when provided with programming assignments recommended based on their behaviours, students showed higher achievements rates and lower failing and dropout rates compared to when receiving random recommendations. This finding suggests that our recommendation approach contributed to maintaining students trying to solve the assignments (less dropouts), preventing them to fail, and enhancing their achievements. Therefore, this finding implies the need to provide adequate recommendations for programming students to practice, instead of relying on their own non-guided choices, which are likely to act as the random approach that would be harmful to their learning.

Third, we found students were more successful within less tries when completing assignments recommended based on their behaviours rather than randomly selected. That is, when completing assignments suggested by the BRS, learners' correctly completed way more assignments, with few attempts, and within a smaller time period than when completing assignments recommended by the RRS. This finding implies that using our approach for assignment recommendation improves programming practice, as learners will likely need less time to successfully complete more assignments that if they would practice with assignments recommended with no consideration for their behavioural data.

Fourth, our results indicate the relationship between affective states and achievement. On one hand, learners' felt less frustration and more happiness from the assignments provided by the BRS than from the RRS. On the other hand, students were more successful (e.g., higher achievement, less dropout) when completing assignments recommended by the BRS compared to when completing those suggested by the RRS. This finding suggests an association between affective states and achievement, that is, frustration and low success and happiness and high success. The implication from this finding is that designers and instructors should strive to ensure learners are not

frustrated but feeling happiness concerning the learning activities, as this will likely enhance their learning experiences.

In closing, it is worth noting that our recommender system have a potential application for instructors. Typically instructors need to create variations of programming assignments lists for different classes, in order to avoid plagiarism, for example. Using our method, consider each problem in a list of exercises already created by a instructor as a target problem. By generating N recommendations for each of these problems, we can compose N new lists of exercises that require effort and knowledge similar to those required to solve the original. Thus, the instructor's workload to compose new programming assignment lists is significantly reduced.

5.8 Chapter Conclusions

This work proposed and validated a method for recommending programming problems in an OJ based on the expected effort extracted from the student behaviour. We compared our method with an random method that simulates the way learners search problems in OJs. To compare the performance of both methods, five descriptive metrics based on data-driven behaviour analysis were collected from the students' logs. Such metrics indicate the effort made by the student when they were developing the solutions. Additionally, we compared both methods in terms of achievement, failure, dropout and affective states trigered when learners were solving the recommended problems.

During the evaluation of the methods, the behaviour-based method obtained a statistically significant higher rate of achievement, and lower rate of failure and dropout. Also, after analysing the logs of the submitted recommendations, we confirm that the behaviour-based recommender suggests problems that are more appropriate to the students' effort based on previous submissions. In addition, our method triggered more positive affective states and less negatives ones in students. Furthermore, we show that our method has potential benefits for learners and instructors, in a way to improve self-regulation learning for the further and reducing the workload in creating assignment tasks in programming classes supported by OJs for the later, towards minimising

plagiarism threats.

It is worth noting that the effort required to solve a given problem depends on the previous knowledge acquired by the learner about the programming topic of that problem. To illustrate, if the student already known how to manipulate matrices using python, it is easier for them to code a matrix sum, as the *numpy* module allows summing up matrices as scalars. However, for a student who has no prior knowledge of matrix manipulation with *numPy*, the effort to learn will be greater. The way in which the level of effort required to solve a problem in the RS based on behaviour was modelled does not take into account this prior knowledge that the student has about the CS1 topic. Figure 35 exemplifies better such limitation of our recommender. Question 1 is a given target problem. The students need to read two variables and print the sum of these variables. A potential recommendation for such target problem would be the question 2, which asks the student to read two matrices (same shape) and print the sum of these matrices. The reason for such bad recommendation is that the effort required to solve both questions are quite similar, and our BRS recommends problems only based on the effort.

```
### Question 1 -- adding two scalars

a = float(input('reading first scalar:'))
b = float(input('reading second scalar:'))

print(a+b)

### Question 2 -- adding two quadratic matrices

import numpy as np

A = np.array(eval(input('reading first quadratic matrix:')))
B = np.array(eval(input('reading second quadratic matrix:')))

print(A+B)
```

Figure 35 – Limitation of the recommender system presented in this Chapter.

Thus, a potential limitation of our behavioural method is recommending a matrix sum problem for a student who is learning how to sum variables. As a way to solve that problem, in the next Chapter we take into consideration topics of problems needed

to be specified on each problems. Notice that a challenging of performing such analysis is that problems in general are not annotated with this information in OJs.

6

Recommending problems for Instructors to compose assignments and exams

> The roots of education are bitter,
> but the fruit is sweet.
>
> - Aristotle

6.1 Overview of the Chapter

For instructors, OJ is a useful tool for creating assignments and exams. However, the task of selecting problems in OJs is time-consuming. First, problems are generally not organised based on topics covered in the CS1 syllabus. Second, assessing whether problems require similar effort and topic to be completed is a subjective and expert-dependent task. It becomes even more difficult if the instructor must create variations of these assessments to avoid plagiarism. Thus, in this Chapter, we propose a method as an extension of the method proposed in the previous Chapter. The method makes intelligent recommendation that works in collaboration with the instructors, helping them in this task of selecting problems to compose one-size-fits-all or personalised assignments/exams. That is, recommendations are made based on the same mechanism,

a fine-grained data-driven analysis of the students' effort on solving problems in the IDE of an OJ system, plus an automatic detection of topics for CS1 problems, based on problem descriptions. We evaluated our method against the state-of-the-art, in a simple blind experiment with CS1 instructors (N = 35). The results show that our recommendations are 88% more accurate as evaluated by instructors in assignments/exams ($p < 0.05$).

6.1.1 Practitioner Notes

What is already known about this topic:

- Fine-grained data collected can also be used for problem recommendation. However, the topic related to the problems should consider when performing recommendation (PEREIRA *et al.*, 2021a).
- Detecting problem topics is challenging since problems are generally not annotated with this information (ZHAO *et al.*, 2018; INTISAR *et al.*, 2019).

What this Chapter adds:

- Demonstrating how our fine-grained features can be combined with a topic detector to perform problem recomendation for instructors.
- Showing, through a single-blind controlled experiment, empirical evidence on how our recommender can be used to support instructors to select problems to compose assignments and exams.

Implications for practice and/or policy:

- For instructors of programming classes supported by OJs (OJ is a trend in education nowadays), the recommender can be used for semi-automatic creation of assignment tasks, that is, based on a single assignment task previous created, *N* assignment tasks can be automatically constructed by recommending N problems for each constituent question of the original assignment task.

- Different programming assignments is something that can be used to avoid plagiarism, which is critical in programming learning.
- Recommending adequate problems might facilitates human/AI collaboration towards prescriptive analytics.
- Presenting how our method can also be used to create a pool of equivalent questions that can be used for many applications.
- Showing a way to expose students' *Fragile learning* (ROBINS, 2019; LEHTINEN *et al.*, 2021).

6.2 Research Question Addressed in this Chapter

Despite the known (IHANTOLA *et al.*, 2015; WASIK *et al.*, 2018; PEREIRA *et al.*, 2020a; ZHAO *et al.*, 2018) benefits OJs bring for CS1 instructors, by reducing their workload via automatic students' code correction, some challenges remain. First, instructors need to select problems to compose assignments and exams. This is a high cognitive load task, due to the many problems available to select from. Second, those questions are generally not hierarchically arranged, nor structured according to CS1 topics, which makes the search processing exhaustive. Third, as an aggravating factor, researchers claim that plagiarism is a common practice in CS1 for many reasons (see (ALBLUWI, 2019; FOWLER; ZILLES, 2021; VICIAN *et al.*, 2006)). Thus, instructors typically create multiple versions of one assignment/exam, to proactively avoid plagiarism. Authors (WASIK *et al.*, 2018; ALBLUWI, 2019; PEREIRA *et al.*, 2020a; FOWLER; ZILLES, 2021) state that one of the most recurrent and repetitive task of a CS1 instructor, when using OJ, is to select the problems to create these varied assignments and exams.

As an *illustrative scenario* about the effort needed for selecting problems in OJs, assume that an instructor uses an OJ to choose problems to create variations of an exam (already used in previous classes) to test students on conditional structures (if-then-else problems). This task seems elementary - simply find equivalent problems to each one on the original exam. However, this exam should be fair to all students, regardless of

the version they take (DORODCHI *et al.*, 2017). This task can then be broken down into two sequential parts: (i) looking through the whole set of OJ problems for those about conditional structures[1], and ii) among the subset of problems on conditional structures, selecting problems that require a similar effort to be solved (equivalent problems), compared to the problems from the original assignment/exam.

In this sense, a recommender system might be used to map an equivalent problem for each problem in the original exam in our illustrative scenario. Given an assignment/exam used as reference by the instructor, the recommender might create one or more variations of it through this equivalence mapping. Note that, for a problem to be equivalent to another, both should share the same CS1 topic, and the effort required to solve each should be similar. Thus, our mechanism presented in the previous Chapter could be used to make recommendation if we enhance it with a method to detect the topic of problems.

Thus, to fill the gap of OJs on lack of problem organisation (ZHAO *et al.*, 2018; INTISAR *et al.*, 2019; WASIK *et al.*, 2018), we use a Deep Learning (DL) architecture and Natural Language Processing (NLP) techniques, adopted from (PEREIRA *et al.*, 2021b), to detect the topic of questions related to the CS1 syllabus. Thus, the research question addressed in this Chapter is related to our previous research question, which is:

RQ6-1) How to support instructors in selecting problems to compose assignments and exams using an enhanced version of our behavioural recommender?

Notice that by detecting the topic and effort required to solve a question, we, for the first time, to the best of our knowledge, propose and validate a method addressing this cumbersome task of selecting problems to compose variations of assignments and exams.

Previous studies on recommender systems for OJ targeted mainly expert learners, preparing for competitive programming competitions. Thus, neither novices nor CS1 instructors were considered. Moreover, the methods did not target problem topics, which were seen as crucial by previous works (ZHAO *et al.*, 2018; INTISAR *et al.*, 2019; PEREIRA *et al.*, 2021b). Differently, the forefront work of (YERA; MARTÍNEZ, 2017)

[1] Some OJs have a category called 'beginning' which facilitates this process somewhat, but it is still far from ideal, since problems from different CS1-related topics are just pilled in this broad category.

makes recommendation also suitable for novice learners and the features employed might implicitly detect the problems topics.

Thus, we compared our method to that of Yera & Martínez (2017) on different CS1 topics, based on different accuracy measures for recommendations to replace an original problem with a new variation. Since (YERA; MARTÍNEZ, 2017)'s work is suitable to suggest problems for students (not for instructors), we performed then an adaptation to conduct our comparison. As a result, we statistically ($p < 0.05$) surpassed our baseline in all measures tested. Our outcomes point to implications that go beyond our primary objective of assisting students in composing one-size-fits-all or personalised (variations) assignments/exams, including creating a pool of equivalent questions to detect *fragile learning* (ROBINS, 2019; LEHTINEN *et al.*, 2021), diminish plagiarism proactively, and provide an inventory of pre- and post-tests for computer education researchers.

6.3 State of the art

In this section, we will explore works that provide solutions for the topic detection and for problem recommendation[2]. We will also discuss our contribution in unifying topic detection and effort measurement to propose and validate a recommender system to support CS1 instructors in composing assignments and exams.

6.3.1 Classification of Topics from Programming Online Judges problems

OJ problems span over different computer science contents. (WASIK *et al.*, 2018; ZHAO *et al.*, 2018; INTISAR *et al.*, 2019; PEREIRA *et al.*, 2021b) explain that users can find problems from beginner topics (e.g., conditionals, loops) to more advanced ones (e.g., dynamic programming, computational geometry) in these systems. Nonetheless, only a few OJs categorise problems based on topics to facilitate the searching process for

[2] For problem recommendation, we extend the discussion we provide in the section 5.4, giving more focus on how a topic detector would benefit the recommender system.

users. For instance, URI online judge (BEZ *et al.*, 2014) organise problems in nine main categories: (1) beginner, (2) ad-hoc, (3) strings, (4) data structures and libraries, (5) mathematics, (6) paradigms, (7) graph, (8) computational geometry, and (9) SQL. A few OJs organise problems in other macro categories targeted for expert users (ZHAO *et al.*, 2018; INTISAR *et al.*, 2019; ATHAVALE *et al.*, 2019). However, many do not provide any hierarchical classification of problems per topic, therefore piling problems into new volumes without any annotation. The main reason is that manually categorising OJ questions is laborious and tough to scale. In addition to the wide variety of problems available, new problems are regularly registered in these systems (WASIK *et al.*, 2018; ZHAO *et al.*, 2018). In this sense, some recent works (INTISAR *et al.*, 2019; ATHAVALE *et al.*, 2019; PEREIRA *et al.*, 2021b; FONSECA *et al.*, 2020) proposed ways to mitigate this issue, by using machine learning and natural language processing techniques over the statements of the problems, to automate/semi-automate the categorisation process.

In a pioneering work, (ZHAO *et al.*, 2018) employed problem description to detect 12 general topics (Dynamic programming, Palindromes, Geometry, Tricky problem, Hardest problem, Basic problems, etc.) using ML techniques. However, they achieved poor results ($\approx 40\%$ accuracy) using Latent Dirichlet Allocation (LDA). Despite that, (ZHAO *et al.*, 2018) recognised the potential of using text mining on problem descriptions and that other ML methods might achieve better results, such as neural networks. (ATHAVALE *et al.*, 2019) employed shallow ML methods with NLP techniques to also categorise OJ problems into five categories (data structures, dynamic programming, greedy algorithms, implementation, and mathematics). Their best predictive models achieved an F1-score of 62.2%. (INTISAR *et al.*, 2019), using techniques similar to (ATHAVALE *et al.*, 2019), achieved a high F1-score of 92% to detect six different topics (Geometry, Number Theory, Game Theory, Dynamic Programming, Graph, Data Structures). However, (INTISAR *et al.*, 2019) used only a small dataset, with only 200 problems and, hence, the sample might not be representative enough for a complex multi-classification task (GÉRON, 2019). Finally, (PEREIRA *et al.*, 2021b) combined many NLP techniques with shallow and deep learning models, constructing a cutting-edge pipeline that achieved a high F1-score ranging from 86% to 96%, to classify eight dif-

ferent topics (Beginner, Graphs, Data Structure, Geometry, Mathematics, Paradigms, Strings) relating to the computer science subject. (FONSECA *et al.*, 2020) used similar techniques and achieved high accuracy (ranging from 92% to 97%) for detecting 23 contexts (e.g., Computational, Games, Mathematics, Sports, etc.) of problems rather than computer science topics. Both of these works, (PEREIRA *et al.*, 2021b; FONSECA *et al.*, 2020) demonstrated that deep learning models are promising for this multi-classification task.

Despite the usefulness of this accurate topic classification for expert users, novice students may still struggle to find problems relating to their skills. For example, a novice student who wishes to practice with simple problems about vectors may still have to struggle across many volumes to find one in that category. In addition, CS1 instructors may also face difficulty finding problems relating specifically to categories in their syllabus. Thus, there is a need for the categories to be more granular, by, for example, segregating problems based on the CS1 syllabus.

In our prior work (PEREIRA *et al.*, 2021b; FONSECA *et al.*, 2020), we achieved high accuracy using an elevated number of categories for the multi-classification task. As such, we hypothesise that our ML pipeline can also be adapted to detect problem topics based on the CS1 syllabus. Thus, we employed the same NLP preprocessing steps proposed in our prior work (PEREIRA *et al.*, 2021b), with a deep learning architecture, to detect eight CS1-related topics.

6.3.2 Problem Recommendation

Despite the calls from the literature (WASIK *et al.*, 2018; SAITO; WATANOBE, 2020; PEREIRA *et al.*, 2020a; IHANTOLA *et al.*, 2015; LUXTON-REILLY *et al.*, 2018; ZHAO *et al.*, 2018; COSTA *et al.*, 2021) for research about recommending problems in OJs, only a few works have proposed methods for this task. Additionally, as mentioned in the previous Chapter, these methods are typically suitable for expert students, but not for novices. For example, (FANTOZZI; LAURA, 2020) used an auto-encoder neural network to recommend problems for expert users who were training for the Interna-

tional Collegiate Programming Contest (ICPC). Similarly, (SAITO; WATANOBE, 2020) employed the learner's submission history to construct the student learning path and perform recommendations based on that and student objectives. Both works (FANTOZZI; LAURA, 2020; SAITO; WATANOBE, 2020) make recommendations based on students' attempts and correctness.

(YERA; MARTÍNEZ, 2017) proposed an approach to recommend OJ problems based on the number of attempts and prior correctness. The authors explained that these features could be useful to measure the students' effort necessary to solve problems. This work is comparable to ours, since it can be used to recommend problems for novice students, and, hence, we used it as our baseline. Their method received a user-problem matrix as input, which covers a set of triples <a, p, j>, where *a* represents the number of attempts a user *u* needed in trying to solve a problem *p*, resulting in the OJ judgement *j*. The judgement was a binary variable, with value *accepted*, if the student solved the problem p, or *not accepted* if they did not. Thus, after processing this input, the method returned a list of problems for a given user, after sequentially building and pre-processing the user-problem matrix, M, based on the following rules:

- M[u, p] = 0, user u has never attempted to solve the problem p;
- M[u, p]= 1, user u has not solved the problem p, and has tried it just few times;
- M[u, p] = 2, user u has not solved the problem p, and has quite a few previous failed attempts;
- M[u, p] = 3, user u solved the problem p, having failed many times beforehand;
- M[u, p] = 4, user u solved the problem p quickly.

To define what 'few' and 'many' mean, they used the average number of problem attempts as the threshold. Exemplifying, assume that the average number of attempts for a given problem p_1 is γ. Then, if a given user u_1 has more than γ attempts and she/he do not solved the problem, then $M[u_1, p_1] = 2$. We adapt the (YERA; MARTÍNEZ, 2017) rules in our set of features (more details in Chapter 2). Hence, our feature space extends

the one proposed by (YERA; MARTÍNEZ, 2017). Our extension considers fine-grained features instead of only rules based on the number of attempts and correctness.

Such fine-grained features (detailed in Chapter 2) could represent behaviours that measure the effort needed to solve the problems, as we demonstrated in the previous Chapter. Indeed, data-driven behaviour analysis has been a trend in the past few years. Many studies analysed metrics that can estimate the students' behavioural patterns in OJs, such as IDE usage, number of errors, average lines of code, etc. Jadud (2006), Watson *et al.* (2013), Dwan *et al.* (2017), Leeuwen *et al.* (2019), Carter *et al.* (2019), Quille & Bergin (2019), Margulieux *et al.* (2020), Pereira *et al.* (2021), Pereira *et al.* (2020). All these works showed that these fine-grained behavioural features are related to students' effort and achievement/performance - as we demonstrated in the previous Chapter. Nonetheless, none of these studies analysed the features in conjunction to automatically recommend problems.

Additionally, none of the presented related work that recommend problems in the OJ considers the problem topics. Thus, uniquely differing from these works, we employed and validated a set of fine-grained features representing the students' effort necessary to solve problems. Moreover, we combined the effort required to solve problems with the topics that we automatically detected to make recommendations, filling a gap from previous works (YERA; MARTÍNEZ, 2017; SAITO; WATANOBE, 2020; FANTOZZI; LAURA, 2020; PEREIRA *et al.*, 2021a). Additionally, despite the importance of these works that target learners, the literature (LUXTON-REILLY *et al.*, 2018; HOLSTEIN *et al.*, 2020) points out the lack of works viable to support CS1 instructors - which ours addresses. Furthermore, the methods presented in this section were not evaluated in the perspective of their target users. Differently, we propose and compare our method with a baseline and evaluate them with CS1 instructors with experience with OJs.

6.4 Data

The educational context and instruments are the same we use in the previous Chapter. The main difference is that now we use not only the students' behavioural data, but also the description of problems available in the Codebench system. We used the problem's description to feed a ML pipeline that automatically detects the topic of the problems based on the CS1 syllabus. The total number of distinct problems from CodeBench we used in this work is 1,026. Table 12 shows the descriptions for each topic (where *a0* depicts the period when students get used with the CodeBench), and the number of problems per topic. Moreover, to clarify the concepts covered in each topic, in Table 12 we provide an example of code, illustrating a typical answer to a question on that topic.

6.5 Effort-topic based Recommender

In this Chapter our goal is to propose and validate a method to recommend questions to instructors to support them in the creation of assignments/exams. By achieving our goal we are then able to respond our research question. Our method uses the concept of hybrid intelligence, in which AI provides the recommendation to the instructors to empower their decision-making on composing CS1 assignments/exams. Collaboration is performed in three sequential steps: i) instructor provides as input an assignment/exam (we call it *master list*), ii) the AI method returns one or several recommended problem(s) for each problem of that master list to be used in the assignment/exam variation(s) iii) the instructor validates each recommendation and decides whether they can be used in the assignment/exam variation(s). The expected effort and topic needed to solve both (master list and its variation(s)) are expected to be equivalent.

Similar to the previous Chapter, for the sake of terminology, we call a Target Problem (TP) each problem from the master list. Given that, we have then an extension of our hypothesis presented in the previous Chapter, by now considering also the topic of problems. Our hypothesis H_1 is that if students have solved a TP and there are problems similar to TP in terms of *topic and effort required*, these problems can be used as potential Recommended Problems (RPs), to replace TP in new assignments or

	Topic	Description	Example of implementation	N
a0	print and input	reading a single variable and use of the print command	`print(20*35)`	19

s1	sequential structure	arithmetic operations and use of variables	`from math import *`
			`r = float(input())`
			`c = pi * r ^ 2`
			`print('%.2f', c)`
				157

s2	if-then-else	conditional structure	`from math import *`
			`r = float(input())`
			`c = pi * r ^ 2`
			`if r > 10:`
			`  print('big circle')`
			`else:`
			`  print('small circle')`
				136

s3	if-then-else (nested)	nested conditionals structure	`from math import *`
			`a = float(input())`
			`b = float(input())`
			`c = float(input())`
			`if (a > 0 and b > 0 and c > 0):`
			`  if ((a < b + c) and (b < a + c) and (c < a + b)):`
			`  s = (a + b + c) / 2.0`
			`  area = sqrt(s * (s - a) * (s - b) * (s - c))`
			`  print("Area:", area)`
			`  else:`
			`  print("Invalid area")`
			`else:`
			`  print("Invalid area")`
				161

s4	while-loop	repetition structure by condition - loops using the structure while	`num = int(input())`
			`while (num != -1):`
			`  if (num % 2 == 0):`
			`  print("even")`
			`  else:`
			`  print("odd")`
			`  num = int(input())`
				114

s5	for-loop	repetition structure by count - loops using the structure for	`import numpy as np`
			`n = int(input())`
			`print(np.ones(n, dtype=int))`
				117

s6	strings	operations on strings	`string = input()`
			`print(string.upper())`
			`print(string*500)`
				47

s6	vectors	operations on unidimensional vectors	`data = eval(str(input()))`
			`speed_limit = data[0]`
			`i = 1`
			`accumulator = 0`
			`minimum_limit = speed_limit + (speed_limit * 0.2)`
			`maximum_limit = speed_limit + (speed_limit * 0.5)`
			`for x in data[1:]:`
			`  if x > minimum_limit and x < maximum_limit:`
			`  print(i)`
			`  accumulator = accumulator + 1`
			`  i = i + 1`
			`print(accumulator)`
				160

s7	matrices	operations on bi-dimensional matrices	`from numpy import *`
			`m = eval(str(input()))`
			`n = eval(str(input()))`
			`matriz = np.zeros((m,n),dtype=int)`
			`print(matriz)`
				134

Table 12 – Description of Topics based on CS1 Syllabus on CodeBench.

exams (variation). As such, our recommendations are based on the following mapping $TP \rightarrow RP_1..RP_n$, where the effort is represented by a series of fine-grained features, based on the time students spent on the IDE to solve problems, software engineer code metrics extracted from the students' code submissions, and so forth, whereas the topics are based on CS1-syllabus presented in Table 12.

Our method then use the same behavioural features we presented in the previous Chapter, and the same process to reduce the dimensionality of our feature space by applying matrix factorisation. The difference is that we also added an NLP pipeline to detect the problems topics. Moreover, we explain how we adapted our mechanism of recommendation to assist CS1 instructors to create assignments/exams through a hybrid solution that combines our AI method with the CS1 instructor's knowledge. Following we present such adaptation.

6.5.1 Recommendation mechanism

After building our feature matrix based on the steps presented in the previous Chapter, we move towards accomplishing our goal to recommend problems to CS1 instructors, to support them in composing assignments/exams variations. Algorithm 1 shows our procedure to return possible variations of assignments/exams for the instructors, given a master list as input. Function *getNewVariation* receives as input a master list, represented by $L = \{q_1...q_m\}$, where *m* is the number of questions in L. Following we present a description of each variable:

- *K* is a global variable that defines the *i-th* new variation L' built upon L;
- q_i depicts the statement of a problem i;
- p_i is a vector that represents the effort required to solve a problem q_i, in which such effort is represented by the aggregation of features presented in Table 2;
- t is the topic related to the CS1-syllabus the NLP classifier detected to be associated with a problem q_i.

- k is a local variable used to find the *kth* nearest neighbour of a vector p_i. This variable is initialised with the global variable K, so that, a first variation assignment/exam will potentially use the first nearest neighbours of the questions present on the master list, whilst the second variation would use the second neighbours and so forth. An exception occurs when the current nearest neighbour is already used in the current L' or L. In which case, we use the next nearest neighbours to be inserted in L' (see line 8-10);

- p'_i is the *kth* nearest neighbour of p_i;

- q'_i is the statement of the questions associated with p'_i. Thus q'_i is the question recommended to replace q_i on the position i of the new variant list L';

In line 3 of the procedure, the variation L' is initialised empty. When iterating over the problems of the master list L, the procedure will automatically classify the topic t of each problem $q_i \in L$ (line 5). Then, in the subset of problems associated of topic t, the statement related to the *kth* nearest neighbour of p_i is assigned to be $q'_m \in L'$. To detect the topic of problems we use *getTopic*, which is a DL model, whist to find the nearest neighbour we use *findKthNearstNeighbour* function. Both auxiliary functions are employed to support our hypothesis H_1.

Algorithm 1 Creating new assignment/exam

```
1: global const K ← 1                                   ▷ K sets the ith L' created based on L.
2: procedure GETNEWVARIATION(L)
3:     L' ← {}
4:     for (q, p) ∈ L do
5:         t ← getTopic(q)
6:         k ← K                                        ▷ Kth nearest neighbour of p is first used as p'
7:         q' ← findKthNearstNeighbour(p, t, k)
8:         while q' ∈ L' OR q' ∈ L do
9:             k ← k + 1
10:            q' ← findKthNearstNeighbour(p, t, k)
11:        end while
12:        L' ← q' ∪ L'
13:    end for
14:    return L'                                        ▷ new assignment/exam L'
15: end procedure
```

6.5.1.1 Topic Detection

The function *getTopic* uses a predictive model constructed using the preprocessing steps provided in our previous work (PEREIRA *et al.*, 2021b). The preprocessing steps over the problems' descriptions are: i) remove all the html tags from the text; ii) translate our statements from Portuguese to English iii) stop-word removal; iv) *lemmatisation* v) converte each capital letter of the data to lowercase vi) tokenisation of the text vii) mask the numbers, viii) and replace line breaks with simple spaces.

Figure 36 illustrates the proposed evaluation methodology pipeline used in the experiments of our research for topic detection. We create here a unique, comprehensive pipeline, studying various combinations of the most popular and successful bleeding edge state-of-the-art techniques for natural language processing. For a deep explanation of each step of our pipeline and results we achieved, you can access a version of our paper (PEREIRA *et al.*, 2022)[3].

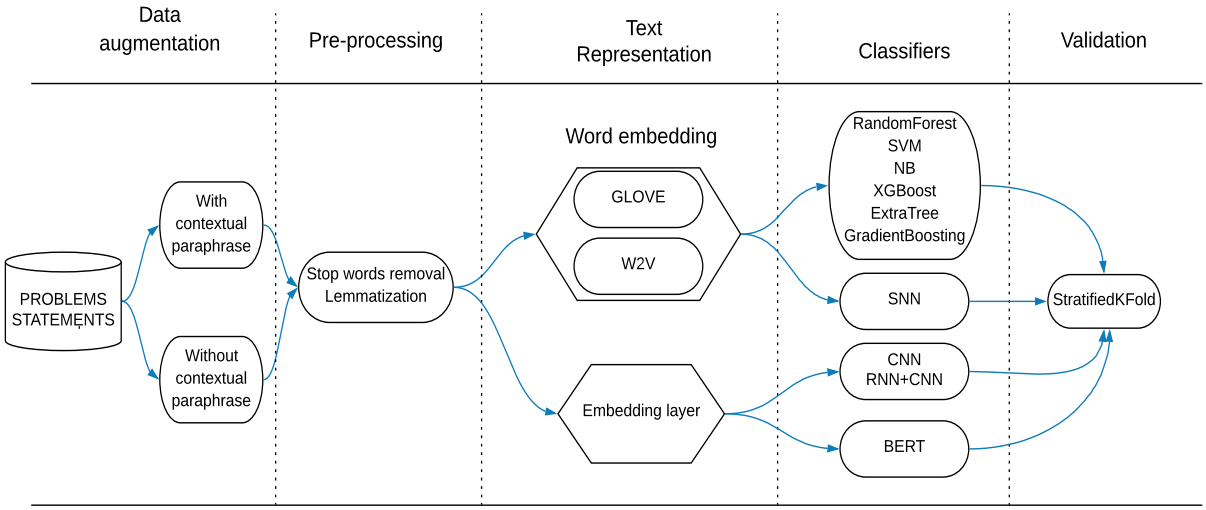

Figure 36 – Our front-heavy pipeline to automatically categorise programming problems based on their statements.

For classification, we achieved the best result using a Convolutional Neural Network (CNN). Thus, we employed such CNN architecture that employs the following sequence of layers: i) embedding layer using 100 dimensions, an input length of 300, and vocabulary equal to the number of tokens from our corpus plus 1. ii) 1D convolution layer that creates a convolution kernel, convolved with the previous layer over a single

[3] A version of this pre-print article is being evaluated by the reviewers of the IEEE Access journal

dimension to produce a tensor of outputs. iii) a convolutional pooling that downsamples the previous layer representation by taking the maximum value over the dimension. iv) a dense layer with 64 nodes using the activation function Relu; v) a dropout layer that randomly sets input unit to 0 using a rate of 10%; vi) a softmax layer to perform the multiclassification. For the optimisation, we used the cutting-edge Adam.

To calculate the performance confidence interval of our model, we run the 10-fold stratified[4] cross-validation 10 times with different seeds to shuffle the data before the split, varying the seed from 1 to 10. Thus, we obtained 100 (10 times 10) different results, i.e. one result for each test set. We did that due to statistical constraints. We implemented it using the *StratifiedKFold* from scikit-learn. As labels, we use the topics presented in Table 12. CodeBench problems are already annotated based on CS1-syllabus topics by UFAM instructors.

As a result, we found that our CNN model achieved an F1-score of 90%. We, thus, opted to use this CNN model as our classifier since, besides achieving a satisfactory result, the literature pointed out that this neural network can reach state-of-the-art results for multi-classification problems using text as source of the data (SONG *et al.*, 2019).

6.5.1.2 Finding the nearest neighbour

We use the nearest neighbour technique over the aggregation of features presented in Table 2. The features aggregations are allocated as dimensions of the vector *p* that represents the effort required to solve *q*. That is, given a pair *(q, p)*, the vector *p* has the aggregation of the features' values based on the learners who solved that question *q*. For instance, given a question q_a, the aggregation of attempts (a given feature presented in Table 2) is the average of attempts student made in trying to solve question q_a. Thus, one of the dimensions of the vector p_a will be the average $attempts_{q_a}$ for all students who submitted accepted solutions for the question q_a.

Using the aggregations represented by p, the similarity is the calculation of the

[4] Stratified in this context means that each *fold* was divided proportionally to the number of statements present in each class in the database (GÉRON, 2019)

correlations between these vectors (the angle between them, always ranging from 0 to 180). Furthermore, cosine arc is used to transform the result into a value between 0 and 180 degrees. In this sense, given two vectors p and p', the cosine between them is given by the product of $p \times p'$ divided by the product of l_2-$norm$ of p and p', given by $\sqrt{\sum_{k=1}^{n} {X_k}^2}$, where $X = \{p, p'\}$. This way, we guarantee that our results generated through the similarity of the cosine are always normalised, with values ranging from 0 to 1. The similarity formula is thus:

$$S_{pp'} = \frac{p \cdot p'}{\|p\| \, \|p'\|} = \frac{\sum_{i=1}^{n} p_i \times p'_i}{\sqrt{\sum_{i=1}^{n} p_i^2} \times \sqrt{\sum_{i=1}^{n} p_i'^2}} \tag{6.1}$$

6.6 Evaluation

In this study, two methods that recommend problems for OJ users are employed and compared, to support CS1 instructors in creating assignments/exams variations. The former is our Effort-topic based Recommender. The latter is (YERA; MARTÍNEZ, 2017)'s method, representing the state-of-the-art. To perform the recommendation, we employed the procedure illustrated in Algorithm 1. However, since (YERA; MARTÍNEZ, 2017) does not perform topic detection, we adapted lines 5 and 7 to make a recommendation based on (YERA; MARTÍNEZ, 2017)'s method.

Notice that our procedure illustrated in Algorithm 1 uses as input a master list L, to create variation L'. Moreover, L is associated with one topic related to the CS1-syllabus, as explained in Section 6.4. Thus, we opted to stratify the evaluation based on the CS1 topics. In order not to create an unnecessarily high number of groups and to reduce the number of questions asked to instructors, we unified similar topics, for the sake of simplicity of this evaluation. First, we merged the problems of "Simple Conditional Structures" and "Nested Conditional Structure" into a more general topic called "Conditional Structures'. Moreover, we also unified the topics "while loops" and "for loops" in a more general topic called "loops".

Notice that a master list L needs to be created in advance by instructors of previous classes, a group of experts, or the OJ maintainer. In our case, we opted to use

master lists created and used previously in CS1 classes at UFAM to simulate a more realistic educational scenario. We will use the following nomenclatures:

- Master list: an assignment/exam created by UFAM group of instructors;
- List A: variation of a master list created using our Effort-topic based Recommender;
- List B: variation of a master list created using (YERA; MARTÍNEZ, 2017)'s method (our baseline).

To assess the power of the methods to create different variations of the assignment/exam, each instructor evaluated a personalised List A and List B. To do this, for each topic, we used a different master list. We iterate with K (global variable in Algorithm 1) from 1 to 7 to create seven different variations of a master list on each topic. So that each instructor evaluates a different variation L', being created using our method (List A) and our baseline (List B).

In order not to overload the instructors by asking them to compare too many problems, we asked them to compare 3 pairs of questions for each list (A and B). To illustrate, if the master list has the problems $L_{master} = \{q_1, q_2, q_3\}$ then List A must have the problems $L_A = \{q'_1, q'_2, q'_3\}$, where each pair of questions (q_1, q'_1), (q_2, q'_2) and (q_3, q'_3) must require a similar effort and topic to be solved by the students. To carry out the evaluation of list variations with instructors, we compare pairs (q_1, q'_1) from each list (A and B), so if all pairs are equivalent, then the lists are equivalent.

Briefly, to carry out the assessment of lists A and B, 5 groups were created, stratified by topic, as shown in the Table 13. In each group we have 7 instructors who evaluate a List A and List B. Each instructor in each group assesses the pairs (q_i^{master}, q_i^A) and (q_i^{master}, q_i^B), where i varies from 1 to 3. In this way, the instructors assess the similarity of 6 pairs of questions.

Participants	Topics (after unification)
Group 01 - 7 instructors	Sequential Structure
Group 02 - 7 instructors	Conditional Structure (simple and nested)
Group 03 - 7 instructors	Repetition structure (while and for)
Group 04 - 7 instructors	Vectors and Strings
Group 05 - 7 instructors	Matrices

Table 13 – Unified topics for the experimental evaluation

6.6.1 Procedure

The procedure for comparing the methods was as follows: we present the i-th target problem, and the i-th recommended problem of a given list (A or B) for the participant to answer if there is a similarity between the target problem and the recommended problem, in relation to measures of effort and topic (presented in the following subsection).

The order of presentation of lists A and B was counterbalanced. As we applied the test 35 times (7x5), we started presenting problems from list A 18 times and with list B 17 times. Furthermore, the assessment is simply-blind, so the instructor did not know whether they would be evaluating problems from list A or list B.

6.6.2 Measures

To measure the equivalence between the pair of problems (target and recommended) we used the following measures:

- *Interchangeable*: whether the pair of problems is equivalent and, hence, whether the recommended problem could replace the target problem in a new assignment/exam.
- *Topic*: whether the topic of the question pairs is equivalent. We used the 5 topics unified and presented in Table 13.
- *Resolution time*: whether the expected time to resolve the question pairs is similar. To perform the similarity comparison, we set a 20 minute threshold. Thus, if the expected time difference for resolution was greater than this threshold, then

the instructor should indicate that the pair of questions does not have a similar resolution time. This time threshold value was chosen because the standard deviation of the times it takes students to solve problems is approximately 20-minutes when solving problems on CodeBench. Moreover, the instructors in the pilot (Section 6.6.3) agreed that this is a reasonable value to be used as threshold.

- *Coding effort*: whether the coding effort to resolve the question pairs is similar. We asked the instructors to understand as coding effort the number of lines of code, conditional structures and loops needed to solve the problem.

- *Hit Rate*: whether the expected hit rate for the question pairs is similar. To perform the similarity comparison, we set a threshold of 25% difference in hit rate. Thus, if the expected difference in the hit rate was greater or lower than this threshold, then the instructor should point out that the pair of questions does not have a similar hit rate. This time limit value was chosen because the standard deviation of student hit rates is approximately 25% when solving problems on CodeBench. Moreover, the instructors in the pilot (Section 6.6.3) agreed that this is a reasonable value to be used as threshold.

We utilised the *forced choice* technique to elaborate the alternatives of the questions since we are simulating the decision-making process of an instructor when checking whether a question recommended by a system will be used or not in a new assignment or exam. Table 14 shows the number of items from the questionnaire for each comparison (List A vs Master List and List B vs Master List), where topic i represents one of the topics presented in Table 13.

Topic i	Comparisons	
	Master List vs List A	Master List vs List B
Problem 1	5 items	5 items
Problem 2	5 items	5 items
Problem 3	5 items	5 items

Table 14 – List A: recommended-test list; List B = baseline-recommended list; Items = interchangeable, topic, resolution time, coding effort, hit rate.

6.6.3 Pilot

Before conducting the experiment, a pilot study was carried out with 5 CS1 instructors from different universities, in order to validate the questionnaire in each group. For that, we used the think-aloud technique to analyse the cognitive process of the instructors while they were solving the questionnaire. This was used to improve the instrument.

After this process, we refined the assumptions instructors should have known before answering the questionnaire. We asked the instructors to assume that there are 5 topics covered in the CS1 course, taught in this order (cumulative learning process): i) variables and conditional structure, ii) conditional structures (simple and nested), iii) repetition structures (while and for) iv) vectors and strings; v) matrices. In addition, we asked them to assume that a question from an assignment/exam belongs to a single topic. Thus, if a problem has elements from more than one topic (e.g., conditional structures and repetition structures), then they should consider only the most complex (repetition structures).

Moreover, we measured the time instructors take to read and answer the questions, so that, when inviting the participants we could provide an estimate of how long it would take to answer the questionnaire. We also asked whether the number of questions were appropriate and about the thresholds we determined to measure effort (Section 6.6.2).

6.6.4 Participants

We invited CS1 instructors from different universities to participate in our experiment through mailing lists and social networks. Next, we asked for their informed consent, as per (GELINAS *et al.*, 2017). A total of 35 CS1 instructors were allocated in one of the groups from Table 13 to answer the questionnaire, comparing the methods with the target list. All CS1 instructors had previous experience with OJs.

6.6.4.1 Data filtering

To monitor participants' fatigue, where they eventually answer without proper attention, and to avoid biasing our study with such data, control questions were added to our questionnaire. These tested if participants were reading the questions carefully. Errors in these control items would justify data exclusion. We also checked our data to exclude participants whose answers were all positive. This response pattern was interpreted as acquiescence, which means that the participant agreed with a question regardless of its content (DANNER *et al.*, 2015). None of our participants, were excluded because of the control questions, but one was excluded for acquiescence.

6.6.5 Instructors answer analysis

We used a binomial multi-level regression (HOX *et al.*, 2010) to compare our method to the one of (YERA; MARTÍNEZ, 2017). On one hand, the forced-choice questions led to categorical dependent variables (e.g., for *interchangeable*, the answer could be either yes or no). Accordingly, we coded all measures as binary (e.g., same topic? yes or no; same resolution time? yes or no) because our goal is to check whether there would be a difference between effort, hit rate and resolution time for a given pair of questions. Thus, for all variables, we assigned 1 if the pair of problems were equivalent and 0 otherwise. That led to the need for using a binomial regression. On the other hand, we used a repeated-measures design, to evaluate the pair of questions from each method. That is, each instructor evaluated 6 different pairs of questions, hence 3 pairs for each method. Because we had six data points for each participant, we had to adopt the multi-level approach, given that alternatives such as the McNemar test and standard regression are limited to 2x2 tables and independent data, respectively (GELMAN; HILL, 2006; LACHENBRUCH, 2014).

Overall, multi-level regression can be seen as a set of standard regressions that are hierarchically grouped (HOX *et al.*, 2010). In many cases, the grouping factor is the factor that makes the data dependent, such as the fact that each participant accounts for six rows in our dataset. To work in that way, multi-level regressions have two

kinds of coefficients: *fixed*, which represent overall properties of the data and do not vary across groups, and *random*, which represent grouping variations, by allowing each one to have a slope/intercept (MIRMAN, 2016). Because we have five dependent variables, we ran five binomial multi-level regressions in our analyses. Considering all dependent variables were coded as binary, higher regression outputs indicate the method's recommendation was more likely to be equivalent to the target problem. Then, to compare recommendation methods, we used the same approach for all regressions: *recommendation method* (ours or (YERA; MARTÍNEZ, 2017)'s) was the only predictor (independent variable) and *participant* was the grouping factor.

Hence, our analyses are based on fixed effects, because we want to test for overall differences between recommendation methods. Nevertheless, note that using the multi-level approach is imperative even though we disregard the random coefficients in interpreting our results. The reason is that fixed effects are estimated while accounting for them. Thus, we are able to test for overall variations, which we expect to be due to changing the recommendations methods, whilst considering within-person variations estimated by the random coefficients (HOX *et al.*, 2010). For these analyses, we adopted the standard 95% confidence level, due to its confirmatory nature. Note that we do not correct p-values, because we conducted a small number of previously planned comparisons (ARMSTRONG, 2014).

6.7 Results

For *interchangeable*, based on our results in Table 15, $\approx 87.6\%$ (92/(92+13)) of the questions recommended using our method are interchangeable, compared to the target problems, whilst $\approx 38\%$ in our baseline. The multi-level analysis confirms the methods' difference for *interchangeable*, showing that the probability of recommended problems being interchangeable is significantly different (p = 1.66e-10) when comparing the baseline to our proposed method (see Figure 37).

Secondly, we compared the methods in terms of *coding effort*. As can be computed based on the data on Table 15, $\approx 81,0\%$ of the problems recommended by our method

Interchangable		
Method	Baseline	Proposed
Equivalent	40	92
Non-Equivalent	65	13
Topic		
Method	Baseline	Proposed
Equivalent	50	97
Non-Equivalent	55	8
Resolution Time		
Method	Baseline	Proposed
Equivalent	53	86
Superior	26	11
Inferior	26	8
Coding Effort		
Method	Baseline	Proposed
Equivalent	34	85
Superior	31	12
Inferior	40	8
Hit Rate		
Method	Baseline	Proposed
Equivalent	47	83
Superior	28	8
Inferior	30	14

Table 15 – Comparison of number of problems evaluated as Equivalent (Eq) and Non-Equivalent (N-Eq) for each measure, where List A was created using our method, whereas List B used our baseline.

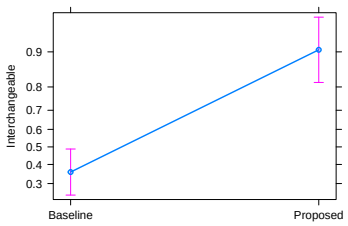

Figure 37 – Methods comparison. Measure: interchangeable.

were equivalent in terms of effort, whereas $\approx 32,4\%$ in our baseline. We can evaluate in Figure 38 that the probability of having problems with equivalent coding effort increases as the method changes from the baseline to the our proposed method. The difference is statistically significant (p = 2.28e-10).

The third measure is *resolution time*. Based on Table 15, we can determine that $\approx 82.4\%$ of the problems recommended by our method were evaluated as equivalent in terms of *resolution time* against $\approx 50.5\%$ of our baseline. As can be seen in Figure 39,

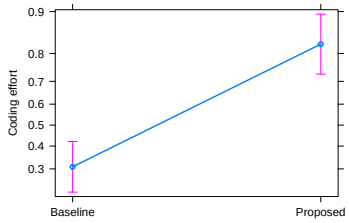

Figure 38 – Methods comparison. Measure: coding effort.

the probability of equivalence in resolution time increases using our method and that difference is statistically significant (p = 1.48e-06).

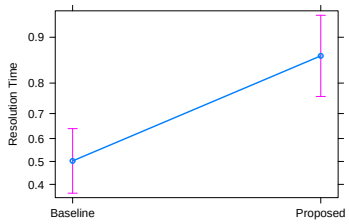

Figure 39 – Methods comparison. Measure: resolution time.

The fourth measure compared is the *hit rate*. Based on Table 15, we can compute that $\approx 79.1\%$ of the problems recommended by our method were assessed as equivalent in terms of *hit rate* against $\approx 44.7\%$ of our baseline. As can be seen in Figure 39, the probability of equivalence in hit rate grows using our method and that difference is statistically significant (p = 3.72e-07).

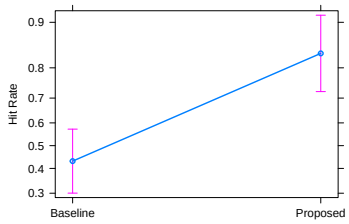

Figure 40 – Methods comparison. Measure: hit rate.

For our last measure, we compared the pair of problems of each method with regards to their *topic*. We can compute from Table 15 that $\approx 92\%$ of the problems recommended by our method were assessed as equivalent in terms of *topic*, whereas $\approx 48\%$ were assessed as equivalent in our baseline. As can be seen in Figure 39, the probability of equivalence in topic grows using our method and the difference is statistically significant (p = 1.97e-09).

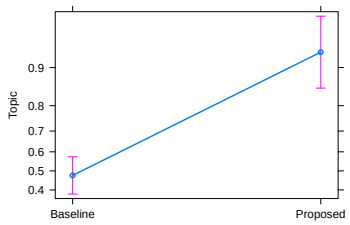

Figure 41 – Methods comparison. Measure: topic.

Thus, we have empirical evidence to support H_1. Indeed, the recommendations provided by our method have high chances of being utilised by instructors (see Figure 37). Moreover, we observed that the likelihood of topic and effort (measured in the experiments as *Resolution time*, *Coding Effort*, and *Hit Rate*) equivalence between the TP and RP provided on list A is high.

Moreover, based on our results, we can thus state that our method can be used to support instructors in selecting problems to compose assignments and exams using an enhanced version of our behavioural recommender, responding then our research question RQ6-1.

6.8 Discussion

6.8.1 General

In this study, we demonstrated how a hybrid human/AI OJ could perceive how students solve problems to make recommendations to instructors on how to compose one-fits-all or personalised (variations) assignments/exams.

Thinking about the organisation of the assignments/exams provided by our method (how problems are sorted), it is important to notice that a question q' is inserted in L' in the same interaction (of the for loop - line 4 of the Algorithm 1) when $q \in L$ is accessed. Notice that based on our results the pair of questions (q, q') is potentially interchangeable, from the same topic, and require a similar effort to be solved. In this way, the recommended list L' is potentially sorted in the same way that the instructor organised the list L. This is important if instructors have scaffolded the master list L by ordering the problems from the smallest to the greatest effort because L' would

be scaffolded as well. Additionally, based on how interchangeable are pair of questions provided by our method, we can state that it can be employed in CS1 classes to support instructors with a relatively low need of instructors endeavour to replace our recommendations.

In terms of validation, we demonstrate that we surpassed our cutting-edge baseline for all measures (*Interchangeable*, *Topic*, *Resolution time*, *Coding effort*, and *Hit rate*) tested. The first measure tested is potentially the most important one, since it simulates the decision-making process of the instructors about whether the pair of problems could be used in a new assessment. The other measures evaluate nuances of both methods, with regard to effort and the topic. About the later, although our baseline (and our related works) does not consider the topic when performing recommendations, we observed that their rules implicitly detected the topics with a precision of $\approx 48\%$. Nonetheless, such precision is far from ideal and, thus, we can state that our novel topic detector is crucial to perform the recommendations. It is worth noting that the detection of effort is not enough to perform precise recommendations in this task, as we have demonstrated in the previous Chapter.

Still, based on our empirical results, we can state that our extended fine-grained features represented better ($p < 0.05$) the effort required to solve problems better than our baseline, taking into consideration the effort being represented by the following measures: *Resolution time*, *Coding effort*, and *Hit rate*. This shows the importance of extracting features in such fine-grained level, differently from our related work, which provides features only based on students' attempts and correctness. Furthermore, we can claim that we propose a different way of performing recommendations, compared to related work, since we perform precise recommendations for instructors, instead of expert users.

Finally, our work has natural application for hybrid systems that combines human and AI. Next, we discuss possible ways of collaboration between CS1 instructors and our AI-method. Besides, we explore potential applications and implications of our method for CS1 classes supported by OJ, as a move that brings OJ from technology-enhanced learning towards smart learning environments that use fine-grained data for

interventions.

6.8.2 Human/AI collaboration

Searching problems in a huge volume of data - not organised by categories or topics - is typically a task that humans can not perform quickly and perhaps well. AI, on the other hand, has the potential to organise the collection of problems and to perform recommendations, as we have shown in this Chapter. In this sense, our AI method leverages the instructors' abilities to accomplish such task. However, our AI method lacks the intuition, flexibility, common sense, and other cognitive skills that only instructors have, due to everyday classroom experiences. In this sense, we propose to combine our AI method with the instructor knowledge, to perform recommendations. The basic idea of combining human and AI is that these heterogeneous intelligences can potentially create a social-technological ensemble that is able to overcome the limitations of both agents (DELLERMANN *et al.*, 2019; HOLSTEIN *et al.*, 2020; DELLERMANN *et al.*, 2021) to leverage perception, action and decision making of these agents.

Regarding perception, using our method, instructors end up evaluating work/exam arrangements that they might not have access to when manually looking for problems to compose the assessments. Additionally, by viewing the proposed assignments/exams, the instructor can gain insights to teach a specific concept needed to solve the AI's proposed assignments/exams questions.

Regarding actions, if the instructors notice high chances of plagiarism in a given class, it would be possible to create different variations assignments/exams in order to proactively avoid this practice. That is, the instructor can redefine the assessment strategy, considering their perception/intuition and alternatives supported by our AI method. Still, depending how large is the pool of questions available on the OJ, it would be possible to create a different assignment or exam for each student, which could foster personalisation an avoid plagiarism. Plagiarism would be discouraged because there's little chance that a student would find a colleague with the same problem that he/she has. Additionally, when the instructors identifies that the teaching conditions were

not effective (in the sense that they did not provide learning) or that the evaluative conditions failed (because they were not understood, had errors or were too difficult), the instructor cannot simply repeat what didn't work. Therefore, variety also matters for this reason, as an "ace up one's sleeve" for the instructor, after all, he/she can access new recommendations for problems based on our AI-method.

In decision making, we then have the most natural part performed from our method. That is, helping the instructors to compose lists and exams so that the instructors can assess and decide which questions will be most appropriate for students without the need for an exhaustive search for problems in an OJ. Our method allows the instructors to selectively overwrite questions suggested by the recommendation system to compose assignments/exams. In this way, the instructor acts as an agent that complements the suggestion revealed by the AI method. In addition, with the increase in perceptions and actions, the instructor has more power to select actions that can be strategic to leverage students' learning.

On the other hand, our method may also allow the instructor to teach AI. Our mapping process of a $TP \rightarrow RP$ has two components of errors: i) the topic associated with the RP may be misassigned due to a topic detector classification failure; ii) the effort required to solve both problems could not be equivalent. These two components of errors can be easily corrected by the instructor when assessing whether the mapping ($TP \rightarrow RP$) is valid. Thus, answering to literature calls for hybrid human/AI approaches (HOLSTEIN *et al.*, 2020; DELLERMANN *et al.*, 2019), our method, in addition to augmenting the instructor's ability to select problems, has also the potential to leverage the AI skills about how to recommend problems. In the last case (instructor teaches the AI), the instructor can relabel the topics of the problems when the topic detector perform misclassification.

Moreover, each time that the AI recommends a problem, the instructor may assign a penalty or a reward to the recommendation, so that the AI-method can use this information to perform better recommendations in the future. It is worth noting that based on our results, the chances of these errors occurring tend to be low. Thus, after the employment of our method, the tendency is that instructors would need to put

less endeavour in the validation of assignments/exams recommended by our method. With the relabelling of poorly classified topics, the tendency of topic missclassifications would be low since our topic detector is already 90% accurate, thus this 10% of missclassification would decrease after instructors relabelling.

Additionally, in the case of a topic missclassification, the nearest neighbour of the target problem is fetched in a different topic than the target problem. Thus, with the correction of the topic, the chances of the recommendation being accurate increase, since the nearest neighbour to the target problem is searched for in the correct subset of problems, that is, in the subset associated with the topic that is truly related to the target problem. To sum up, instructors and our AI method can support each other by influencing each other's decision making process. Furthermore, instructors would become more effective at their jobs with fewer operational duties, which usually competes with a closer, more attentive relationship between instructors and students.

6.8.3 Pedagogical Implications

First, our method can also be used to compose a pool of equivalent questions, which can make many pedagogical applications possible. Part of being a skilled programmer includes the ability of transferring understanding from solved problems to equivalent or analogous problems, that is, the ability to generalise a solution. So that students could solve pair of equivalent questions for self-testing purposes. Moreover, the testing effect theory has shown to be effective in many fields of education (ROWLAND, 2014). Such a pool of questions might facilitate the implementation of testing effect theory by providing pairs of equivalent questions in different moments of the course in order to leverage the long-term memory of students by retrieving the to-be-remembered knowledge required to solve the problems. As such, the pool of questions could be used to re-enforce the practice of a single concept/knowledge in a give topic.

Fragile learning (ROBINS, 2019; LEHTINEN *et al.*, 2021) might be exposed, in cases where learners are not able to solve equivalent questions, after a short or relatively long period, depending on the duration of the course. For instance, if the instructor

detects that some student had solved a problem, but cannot generalise her/his solutions in similar questions recommended by our method, then this is a sign of fragile learning. The instructor could apply some intervention to improve the student learning of the concept associated with the questions in evidence.

Apart of practice to improve students' learning, our method might also be employed in the computing education research. Applying pre- and post-tests is an important procedure to evaluate interventions in this field. However, creating equivalent tests is a very challenging task for the researcher. As our method can be used to recommend similar questions,it can create an inventory of equivalent questions to be used in pre- and post-tests. In other words, assume that an arbitrary exercise list L_1 is used as a pre-test in a given experiment. By using L_1 as an input to our Algorithm 1, we can then apply L'_1 as a post-test. The pool of equivalent questions could also be used for this purpose. This same rationale applies to researchers that conduct multiple-baseline experiments between subjects. They need several tests throughout the experiment to demonstrate that behavioural changes (learning) only occurs when a specific independent variable is manipulated, like a class to teach students Python functions that deal with problems involving strings.

6.9 Chapter Conclusion

In this Chapter, we proposed and validated that our method of intelligent recommendation, in collaboration with the instructor, can be useful in assisting CS1 instructors in the arduous task of selecting problems to compose one-fits-all and personalised assignments/exams. The experimental results obtained from a single-blinded controlled evaluation with CS1 instructors support our hypothesis that *if students have solved a TP and there are problems similar to the TP in terms of topic and effort required, these problems can be used as potential Recommended Problems (RPs), to replace the TP in new assignments or exams (variation)*. Chances of our method recommending viable questions for instructors are $\approx 88\%$ (*interchangeable*), which statistically surpassed our baseline ($p < 0.05$). Additionally, we show how fine-grained features represent effort combined with a

DL classifier that employs NLP techniques over problems' description can empower the system to perform highly accurate recommendations, in terms of four different measures (*Topic*, *Resolution Time*, *Coding Effort*, and *Hit Rate*).

We believe that such results are promising for advancing many technologies for data-driven interventions in in a hybrid human/AI OJ for CS1 classes. However, such belief should be tested and validated in the perspective of our target audience - the instructors. Indeed, more than the method presented in this Chapter, the usefulness of all the methods presented in this work should be systematically and consistently validated by CS1 instructors. Thus, in the next Chapter we move in this direction by proposing and validating concept-designs created derived from the methods presented in this work. Moreover, we also provide a human/AI architecture integrating those concept-designs doing then a final step towards our general goal, presented in the first Chapter.

7

VALIDATION OF THE HYBRID AI/HUMAN ONLINE JUDGE ARCHITECTURE

> Live as if you were to die tomorrow. Learn as if you were to live forever.
>
> - Mahatma Gandhi

7.1 Overview of the Chapter

In the previous Chapters we proposed and validated methods to extend the typical OJ architecture to support CS1. However, our methods have not been validated by our main target users, that is, CS1 instructors. Notice that, in our work, we have a premise that learning data must be translated to personalised instructional support for instructors, who in turn, can monitor and mediate the adaptation of the student learning. That is, instructors are in charge of which of our methods from our novel OJ architecture would be available for the students. Thus, in this Chapter we evaluate our proposal of a novel human/AI OJ architecture over the perspective of instructors. To do so, we use the results and potential application based on the methods we presented so far in format of

concept design to support CS1 instructors. Specifically, we focus on how those methods can be used in a hybrid architecture that combines human and AI intelligence. We then conducted a qualitative study, where we applied a user experience method (*Speed Dating*) to consistently validate our architecture's concept-design relevance with 22 CS1 instructors from different universities. Our results suggest that early prediction based on explainable AI (presented in Chapter 4), along with methods to support instructors to elaborate their assessments/exams (presented in Chapter 6) are the better evaluated by the instructors. Furthermore, instructors reported the relevance of using techniques to detect behaviours that can increase the students' chances of passing and clustering those behaviours to facilitate the intervention (presented in Chapter 2 and Chapter 4). Performance prediction based on black-box models were also well accepted, but with some concerns about the accuracy of the model and information provided about the predictive model. These findings can fill the gap between design and implementation of hybrid Human/AI OJ and shed light on monitoring and adaptation of OJ systems to support CS1 teaching.

What is already known about this topic:

- Typical OJ architecture to support CS1 needs to be extended (WASIK *et al.*, 2018).
- Application of methods that use data collected from OJ are not validated by real users (PEREIRA *et al.*, 2020a).
- Hybrid intelligence can achieve better results than humans and AI alone (AGRAWAL *et al.*, 2018; DELLERMANN *et al.*, 2019; HOLSTEIN *et al.*, 2020).

What this Chapter adds:

- A hybrid human/AI architecture that extends the Typical OJ architecture.
- Validation with CS1 instructors of concept-design based on AI methods that use data collected from OJ systems.

Implications for practice and/or policy:

- Demonstrating possibilities of AI augmenting the instructors' capabilities to improve students' learning, and vice-versa.

- Our findings can be used to fill a gap between the design and implementation of hybrid Human/AI OJ. Designers, developers, instructors, and policymakers within universities might use our novel architecture as a reference in such an aim.

7.2 Research Question Addressed in this Chapter

In this study, we collected fine-grained data from students while they were solving problems in an IDE built into an OJ. Such problems are arranged in assignments created by instructors and made available in an OJ that we use to support CS1 classes. More specifically, we collected logs, codes and interaction data on Codebench, which is the OJ we used as an instrument in this research.

This granular data was used to create a programming profile that represents students' programming behaviours. These programming profiles, in turn, are used to produce descriptive, predictive and prescriptive models, using various AI techniques: machine learning algorithms (deep and shallow), recommender systems, natural language processing and genetic algorithms. The programming profile unifies features presented in publications that we mapped in Pereira *et al.* (2020a) and joins them to new features formulated in this research.

Just to recap, the Figure 42 [1] presents the architecture proposed in this study. We employed descriptive models to gain insights from the data by visualizing and statistical analysis of correlations and relationships of programming profile attributes with student performance. We employ predictive models to estimate student performance at the end of the course, in order to provide early intervention, and intervention during the course (not necessarily early). We utilised prescriptive models to suggest to the instructor/student possibilities of effective coding behaviours, that is, that can increase the chances of the student being approved. Prescriptive models can also be used so that instructors have a better understanding of the possible factors that are driving students to succeed (pass). Finally, as another prescription step, based on an observation we had throughout the PhD, the programming profile can also be used to recommend

[1] We presented the same figure in Chapter 1.

programming problems to students and instructors.

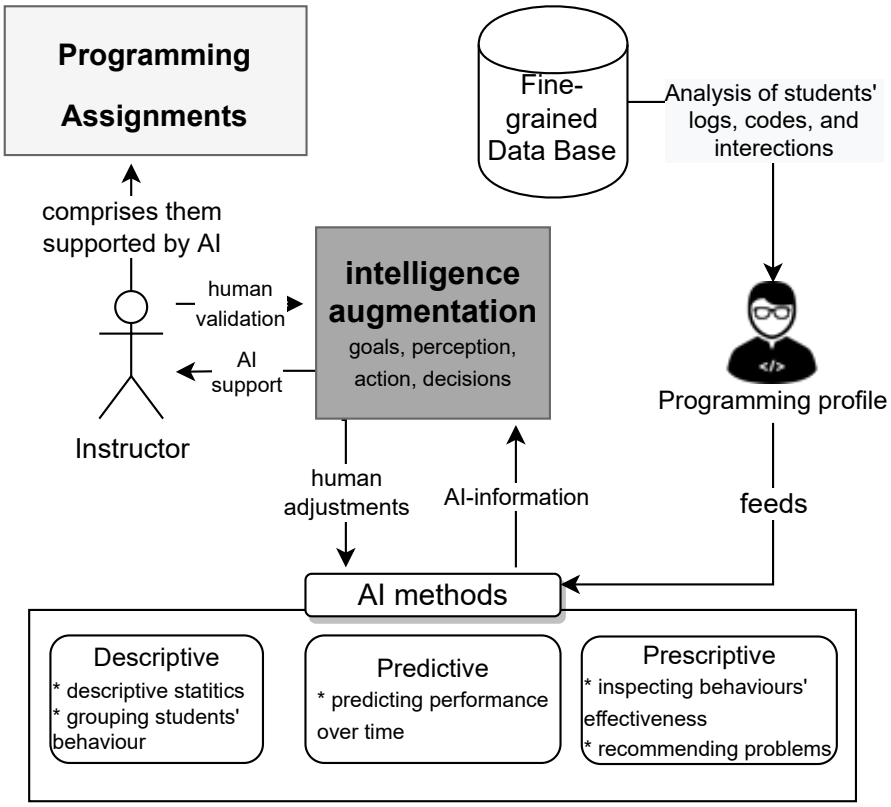

Figure 42 – Extension of a typical Human/AI OJ architecture to support CS1 teaching and learning.

When possible, we compared the results of our methods against methods presented in the literature through statistical comparisons of performance (e.g. comparison of f1-score of predictive models). In addition, the recommendation method was evaluated in a controlled study with students/instructors. Thus, all methods (descriptive, predictive and prescriptive) have been validated and must be adapted into a dashboard that can be incorporated into an online judge model that extends the typical architecture of these systems, that follows the arquitecture presented in Figure 42.

It is also noted that we model the interaction of AI with the instructor in a hybrid way, in which the instructors is helped by AI and vice versa. Therefore, this study makes a move towards hybrid online judge systems, in which AI can support the decision-making of instructors and in which instructors can also act by assisting AI (e.g., through the re-labeling of patterns discovered by the AI).

However, an essential step before the implementation of such dashboard is

the evaluation of these methods with real users. Note that the potential pedagogical implications of our methods have not been validated with users. Thus, in this Chapter we carry out a consistent and systematic evaluation of design concepts created from the methods proposed in this study. Thus, our research question is as follows:

RQ7-1) What is the relevance (quantitative and qualitative) of design concepts based on our methods, from the perspective of CS1 instructors?

To address the research question, in this Chapter we created 19 concept designs based on our method and evaluated them with 22 CS1 instructors. In addition, we point out directions on how these concepts can be used in a hybrid architecture that combines instructor intelligence with AI. The following is a brief explanation of hybrid intelligence.

7.2.1 Hybrid human/AI systems

Instead of combining homogeneous agents (humans, animals), hybrid intelligence combines the complementary strengths of heterogeneous intelligent agents (e.g., instructors and AI). Although AI is being adopted much less than ideal in educational practice (HOLSTEIN *et al.*, 2020), it could be applied to support instructors in structuring learners' data, recognising hidden patterns, making forecasts, and so forth. Notice that humans often act non-Bayesian making inconsistent decisions and violating probabilistic rules. In light of this, the analytical power of AI grounded by the statistical and probabilistic foundation can empower instructors in making more effective, consistent, and accurate decisions (AGRAWAL *et al.*, 2018; DELLERMANN *et al.*, 2019).

Moreover, humans' intuition can empower the choices from the AI predictions or prescriptions. Indeed, on the opposite path (humans helping AI), humans provide domain knowledge and instructions to teach machines in many tasks that they cannot do by themselves. A typical example is when humans provide labels for supervised machine learning approaches to train the models or to make sense of unsupervised approaches. Also, humans are notorious better to perform tasks that require intuition, flexibility, creativity, empathy, and common sense (DELLERMANN *et al.*, 2019).

As such, well-designed AIEd systems may allow both agents (AI and human) to augment the intelligence of each other (HOLSTEIN *et al.*, 2020; DE-ARTEAGA *et al.*, 2020). In other words, AIEd systems will potentially work more effectively by combining the complementary strengths of humans and vice-versa (MOLENAAR *et al.*, 2019; VANLEHN *et al.*, 2021); such a combination is called hybrid intelligence. Many authors claim that hybrid systems will be a dominant model in many fields (DELLERMANN *et al.*, 2019; HOLSTEIN *et al.*, 2019; MOLENAAR *et al.*, 2019; DE-ARTEAGA *et al.*, 2020). However, there is a lack of studies about proposing, designing and validating human/AI hybrid methods and systems. For instance, Dellermann *et al.* (2019) claim that more research is necessary to build domain-specific human/AI educational systems and to explore interface designs that allow users helpers to teach an AI system and vice-versa.

In light of this, Holstein *et al.* (2020) proposed a conceptual framework to map distinct ways of human/AI collaboration. The framework shows dimensions that capture crucial components of an AIEd system. These dimensions are based on frameworks available in the literature (NEWELL, 1994; RUMMEL, 2018; ALEVEN *et al.*, 2016; VANLEHN *et al.*, 2021), which were captured and adapted in a more general conceptual framework for human/AI hybrid systems. Due to the generality, we opted to use the dimensions presented in Holstein *et al.* (2020) to capture how humans and AI can augment each other's abilities in our hybrid human/AI OJ architecture.

7.2.1.1 Dimensions for hybrid Human-AI systems

Based on the conceptual framework proposed by Holstein *et al.* (2020), the first dimension is goal augmentation. One possible way of human/AI collaboration is positively affecting each other's instructional goals. However, the agents' goals might be conflicting. To illustrate, an AIEd system might guide students to complete the assignments at their own pace. However, as the instructor needs to finish the content of a given topic on schedule, sometimes the instructor might want to expedite the process to the detriment of the AIEd suggestions. In this case, the instructor's goals might harm

students' learning process. Thus, Holstein *et al.* (2020) explain that *future hybrid systems could help instructors reflect and refine their goals and methodology*. On the other hand, instructors could also support the AIEd systems to improve their goals as they hold essential knowledge that AIEd systems do not have access to.

The second dimension is perceptual augmentation. This is related to the systems' capability to infer the students' current knowledge from patterns of historical data or recent behaviour. In terms of perceptual augmentation, AIEd and humans can collaborate by leveraging their capabilities of *perceiving opportunities for actions that could improve the students learning process*. In other words, this dimension is related to how both agents can help each other perceive, sense, and notice the relevant information that can be used for timely and effective intervention in the educational environment.

The third dimension is action augmentation. Another way AIEd and humans can also collaborate is by leveraging each other abilities of *performing instructional actions and augmenting the number of activities available to each*. For instance, the AIEd system could augment the instructors' ability to adapt their methodology or interventions for a group of students.

The fourth (and last) dimension is decision augmentation. Human and AIEd systems may collaborate to help each on the decision-making process, maximising the chances of achieving effective decision to improve the learning process, that is, *helping them map perception to action*. Notice that the other three dimensions influence this one. To illustrate, perceptual augmentation might leverage decision-making by moving the instructors' attention to a learning phenomenon that requires their action or evaluation.

7.3 Method

7.3.1 Speed Dating Design Method

According to Davidoff *et al.* (2007), Speed Dating is a design method that can rapidly explore and compare design concepts without requiring any technology implementation. The Speed Dating method helps researchers and professionals to draw possible futures with target users based on their needs, reducing the risk of making products that they

will not adopt (ZIMMERMAN; FORLIZZI, 2017; DAVIDOFF *et al.*, 2007). This method consists of two main phases: user needs validation and user approval. In the validation phase, researchers/professionals present various predefined storyboards to the target users in order to synchronise the design opportunities researchers found with the needs users perceive (DAVIDOFF *et al.*, 2007; TRUONG *et al.*, 2006). These storyboards help researchers/professionals rapidly investigate many possible futures, prioritise user needs, and narrow the design space for potential applications (DAVIDOFF *et al.*, 2007; TRUONG *et al.*, 2006). In the user approval phase, participants must play a specific role that they play regularly (e.g., instructor) to evaluate critically the different design concepts demonstrated by the storyboards (DAVIDOFF *et al.*, 2007; TRUONG *et al.*, 2006).

The first proposal is a novel hybrid human/AI online judge architecture in this work. However, considering that it is a novel contribution, it is an open question of how future human/AI hybrid OJ systems that adopt our architecture will implement it from a design perspective. In the field of human-computer interaction, it is well-stated the relevance of participatory design, which is a process that includes the stakeholders in the stages of design (ROSENZWEIG, 2015), to recognise the rights of the stakeholders in having a voice in the technology design (BANNON; EHN, 2012). Therefore, this research must validate the architecture-based design concepts according to the target audience's needs (i.e., instructors). In this work, we adopt the "Speed Dating method" to validate the design concepts based on the proposed "human/AI hybrid Programming Online Judge architecture".

As previously stated, our human target audience is instructors, who will be responsible for making the final decision based on AI recommendations, according to our proposed architecture. Consequently, this research recruited 20 CS1 instructors from different universities with previous experience with OJ systems to participate in individual speed dating sessions. To recruit these instructors, we used convenience sampling by sending emails to CS1 instructors of different Brazilian universities. We conducted the sessions through video conference due to the Covid-19 pandemic, and the sessions lasted 45 to 75 minutes each. Two researchers were responsible to annotate

the instructors' responses. After the meetings, the annotators discuss and

All the concepts were presented in format of storyboards, as recommended by the literature. The storyboards can be found on this link[2]. The instructors' answers were of two types: qualitative and quantitative. For the qualitative questions, the instructors provided their perspective over a specific concept-design, explaining which aspects of the concept could be positive, negative, what could be improved upon, and potential limitations. The quantitative could be -1, 0 or 1, in which the first (-1) represent that the instructors evaluated the concepts as non-relevant. The second (0) represent that the instructor is in doubt about the relevance of the concept, whereas the last (1) means the instructor find the concept relevant. Moreover, during and at the end of the session we ask the instructors whether they would like to suggest a new concept-design. We did that because the Speed dating method allows new concepts being created by the instructors during the sessions.

7.3.2 Pilot and Concepts Definition

To define the concepts, we first explored the methods we proposed in the previous Chapters. We limited our scope for descriptive, predictive, and prescriptive analysis concepts since these analytical areas employ AI techniques to analyse data to improve the decision-making process. Overall, our methods can be used for performance prediction, group formation, analysis of effective and ineffective behaviours, and recommending problems in online judges to support instructors and students. We then held 3 discussion sessions among at least 6 researchers, including the authors of this work and external collaborators, to establish the initial set of concepts. At the end more then 30 concepts were elaborated based on our methods.

Thus, before carrying out the experiment we performed a pilot with 2 CS1 instructors with previous experience with OJs, which the main aim at reducing the number of concepts, by removing potential redundancies. During the 2 sessions, we reduced the number of concepts to 16 concepts. Moreover, the pilot were helpful to give

[2] sites.google.com/ic.ufal.br/speeddatingmethod

us insights on how to conduct the Speed Dating sessions, the duration of the sessions, and how to conduct the interviews.

7.3.3 Descriptions of the Validated Design Concepts

As stated previously, the speed dating is a dynamic method and the instructors might suggest ideas for the conception of new concepts during the evaluation process. In this sense, the instructor suggested 3 new concepts and, hence, in the end, we achieved a total of 19 design concepts. The concepts, their relation to the dimensions described in section 7.2.1.1, and their relation to our previous Chapters are described below:

- Concept 1 - *early performance prediction*: helping instructors to identify at the beginning of the course (e.g., first 2-4 weeks of course) the student's probabilities of passing based on their learning data. This concept is related to the second and third dimensions, and the findings from Chapter 3.

- Concept 2 - *report of performance prediction and abruptly changes over time*: helping instructors to visualise a periodically (e.g., biweekly) reporting of the probability of the student passing during the course. The idea is to create a graphical report for the instructor with notification of abrupt changes, such as a student who has 80% of a chance of passing in the fourth week and then in the sixth week has only 40%. This concept is related to the second and third dimensions, and the findings from Chapter 3.

- Concept 3 - *visualisation of at-risk and high-achievers students*: helping instructors to identify (even at the beginning of the course) at-risk and high-achievers students based on their probabilities of passing. In this concept, the instructor can use a threshold to filter the students with probability of passing below or above this value. This concept is related to the first, second, and third dimensions, and the findings from Chapter 3.

- Concept 4 - *inspecting individual effective and ineffective behaviours* - helping instructors to identify (even at the beginning of the course) individually which student

programming behaviour may increase (effective behaviour) or decrease (ineffective behaviour) their chances of passing. In this concept, the proposal uses XAI to inspect and explain why the model predicts that a student has high probability of passing or failing. Instructors can visualise it in precise but heavyweight plots such as a decision plot or forceplot. This concept is related to all dimensions, and the findings from Chapter 4.

- Concept 5 - *inspecting groups of effective and ineffective behaviours*: helping instructors to inspect and visualise a group of similar behaviours from different students that potentially lead to similar performance predictions. This concept is related to all dimensions, and the findings from Chapter 4

- Concept 6 - *inspecting effective and ineffective behaviours of the class*: the same instructor may teach CS1 for different classes. In this concept, the instructor can inspect effective and ineffective behaviours of a given *class* in a lightweight and straightforward plot (e.g., a bar plot). This concept is related to all dimensions, and the findings from Chapter 4.

- Concept 7 - *inspect effective and ineffective behaviours per student in a lightweight plot*: similar to concept 4, the instructor can inspect which are the effective and ineffective behaviours for a given *student*. However, in this concept, the idea is to provide this information in a lightweight and straightforward plot (e.g., a bar plot). Notice that such a plot is not as precise as a force or a decision plot. This concept is related to all dimensions, and the findings from Chapter 4

- Concept 8 - *comparing students behaviours against the low, border, and high reference values*: helping instructors to compare students' behaviours on the assignments against the low, medium, and high reference values. The reference values are obtained by clustering of students' programming behaviours and evaluation of the relationship of the clusters' centroids and students' outcomes. This concept is related to the first, second, and third dimensions, and the findings from Chapter 2.

- Concept 9 - *comparison of students behaviours with class average*: helping instructors to compare student's behaviours on the assignments against the average obtained

in her class. This concept is related to the first, second, and third dimensions, and the findings from Chapter 2.

- Concept 10 - *creating one-fits-all assignment lists*: helping instructors select problems to compose one single assignment list for all students from a given class. The instructors select a given topic of the CS1 syllabus. The system automatically creates one assignment based on the historical type of assignments that the same (or other) instructor used in previous classes, based on the fine-grained data of how students solve the problems[3]. After that, the instructor validated the assignment, replacing some questions. This concept is related to all dimensions, and the findings from Chapter 6.

- Concept 11 - *scaffolding assignment lists*: helping instructors to create more challenging assignments lists for a group of high achievers and slightly simpler lists for low achievers/average students. This concept is related to all dimensions, and the findings from Chapter3 and Chapter 6.

- Concept 12 - *creating variations of one-fits-all assignment*: helping instructors to create N variations/versions of a single one-fits-all assignment to avoid plagiarism practice. The difficulty of the variations should be equivalent to the original one-fits-all assignment. This concept is related to all dimensions, and the findings from Chapter 6.

- Concept 13 - *creating individual assignments for each student*: helping instructors to create a unique assignment for each student of a given class. The assignment would be a variation of one original one-fits-all assignment. The difficulty of the variations should be equivalent to the original one-fits-all assignment. This concept is related to all dimensions, and the findings from Chapter 6.

- Concept 14 - *creating one-fits-all exams*: helping instructors to select problems to compose one-fits-all exams[4]. This concept is related to all dimensions, and the

[3] This mechanism is assumed to be used in all other concepts employed to help instructors select the problems to compose the assignments and exams.

[4] We created specific concepts for exams and assignments because some instructors reported that they have different criteria to create them and, hence, their evaluation of the concepts might be different.

findings from Chapter 6.

- Concept 15 - *creating variations of one-fits-all exams*: similar to the mechanism of concept 12, the idea is to help instructors create different versions of the same exam to avoid plagiarism. This concept is related to all dimensions, and the findings from Chapter 6.

- Concept 16 - *creating individual exams for each student*: using the same mechanism of concept 13, here the idea is to help instructors to create a specific exam for each student to avoid plagiarism. This concept is related to all dimensions, and the findings from Chapter 6.

- Concept 17 - *personalised notification*: help instructors to send messages/notification to groups of students with similar performance estimates or similar behaviours to enable interventions. Also, the instructor can send these notifications to individual students based on their access to the previous concepts. This concept is related to the second, third, and fourth dimensions, and the findings from all previous Chapters.

- Concept 18 - *problem recommendation*: an instructor might use a recommender system for two purposes: i) when the instructors use the concept of automatic list/exam creation, they may want to replace some questions from the list/exam created by the AI method. For each question the instructors want to replace, they can ask the system to recommend substitute problems; ii) for students using the recommender in an unguided learning process. After the student finishes the assignment, the instructor could suggest that the learner uses the recommender to continue solving problems recommended by the system for the students in an unguided learning process. This concept is related to all dimensions, and the findings from Chapter5 and Chapter 6.

- Concept 19 - *help button for each concept describing its functionality*: helping instructors to understand the functionality of each concept. This concept is related to the second dimension. This concept is based on human experience interfaces, not necessarily related to our previous chapters.

7.4 Results and Discussion

7.4.1 Concepts Classification

We rated concepts as *high* ($\geq .7$), *moderate* ($> .4$ and $< .7$) and *poor* (≤ 4) based on the average rating given by instructors. The thresholds used for classification follow the same pattern as those recommended in (DAVIDOFF *et al.*, 2007; TENÓRIO *et al.*, 2021). Notice that a concept rate can range from -1 to 1 (DAVIDOFF *et al.*, 2007). Figure 43 presents the instructors' classification of the concepts. The rows in this figure indicate our concepts, whilst the columns depict the instructors who participated in the study (sorted in order of participation). Instructors came up with the last three concepts in the figure. We use blue cells to represent high classification and red cells to depict poor classification. Cases in which instructors doubt (neutral) about the concept use received 0, represented in green in this figure. The rightmost column shows the average of the ratings assigned by the instructors for the concepts.

Concepts	IP1	IP2	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15	I16	I17	I18	I19	I20	Average
Concept 1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.91
Concept 2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.00
Concept 3	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.91
Concept 4	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.95
Concept 5	1	1	1	1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.91
Concept 6	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0.86
Concept 7	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0.86
Concept 8	1	1	1	1	0	0	1	0	1	1	1	1	1	0	1	1	1	1	1	0	1	1	0.77
Concept 9	-1	1	1	1	0	0	0	0	1	1	-1	1	1	0	1	1	1	1	1	1	1	1	0.59
Concept 10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.00
Concept 11	1	1	1	1	1	1	1	1	-1	1	1	1	-1	1	1	1	1	1	1	-1	1	0	0.68
Concept 12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	1	0.91
Concept 13	0	1	1	1	-1	1	1	1	1	1	-1	1	1	-1	0	-1	1	-1	-1	1	-1	-1	0.18
Concept 14	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0.91
Concept 15	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	-1	1	1	1	0.86
Concept 16	0	1	1	1	1	1	0	1	1	1	1	1	1	1	0	-1	1	-1	-1	1	-1	-1	0.41
Concept 17			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0.95
Concept 18			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.00
Concept 19					1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0.94

Figure 43 – Quantitative evaluation of concepts performed by the CS1 instructors.

Given that, we can visually inspect Figure 43 and observe that 15 concepts were evaluated as high, the instructors evaluated two concepts as moderate, and the other two concepts as poor. Following, we present the qualitative evaluation of each concept from the instructors' perspective.

7.4.2 Concepts Sumarisation in Instructors' Perspectives

This section performs a qualitative analysis of how the instructors evaluated the concepts. To summarise the concepts from the instructors' perspective, we extract from the transcriptions of the interviews the exposed advantages, suggestions for improvements, and limitations for each concept. Confirmatory comments (e.g., the concept is relevant) encoded in the quantitative analysis were not considered for this section.

Concept 1 (*early performance prediction*) was highly rated (mean = 0.91). Since CS1 courses typically have a high failure rate, most of the instructors (N = 18) agreed that having a predictive model that estimates even in the first weeks of the course the chances of a student passing is important to enable early intervention. As stated by instructors I1 and I11, this is a fundamental concept for the instructors to help at-risk students complete the course, especially in classes with many students. Additionally, many instructors (I4, I8, I13, I20) stated that this concept would be important to combine the predictions with their daily experience in the classroom. I20 explained that instructors use this functionality (performance prediction) empirically based on classroom experiences, and with this concept, the instructor's predictive capability would be augmented. However, two instructors (I2 and I6) had doubts about using this concept because they did not trust such an early performance prediction. Whilst I6 was not confident about the prediction accuracy, I2 opined that as he does not have information on which data is used to make the prediction, he can't judge whether the concept is relevant. Moreover, I2 pointed out that to increase the trust in the predictive model, she/he would like to know what students' features the model would employ to perform the inference.

Concept 2 (*report of performance prediction and abruptly changes over time*) was highly rated (mean = 1.00). All instructors (N = 22) agreed that this concept is relevant. Notice that a student might begin the course well but, due to some reason, end up failing and vice-versa. So, we asked the instructors whether it is important to discover the inflection point, which is when this change occurred. Instructors I4 and I12 explained that they should receive a notification when this change happens. I14 claimed that this might reveal which subject those students are struggling with since the change would be associated with a period of time the instructors have taught some specific

subject. Furthermore, I5 and I17 claimed that this concept might also be important to check the effect of the intervention during the course or changes in their methodologies. These instructors (I5 and I17) explained that it would be helpful to add a label every time the instructor performed an intervention or change in their methodology to verify the potential effect of the intervention on the predicted performance of the students. Besides, instructor I8 opined that if many students are in an at-risk situation, thus the problems could be with the instructor methodology. Hence, this concept could be used as a self-evaluation tool for instructors. Finally, I6 explained that, differently from concept 1, in this concept, the performance of the predictive model tend to grow throughout the course since more data (from more weeks) will become available for the predictive model during the course, and it increases her/his confidence in using this concept.

Concept 3 (*visualisation of at-risk and high-achievers students*) was also highly rated (mean = 0.91). In this concept, an instructor could, for example, visualise potential at-risk or high achievers students by using a threshold. To illustrate, selecting all students that have less than 50% chances for passing in the course's fourth week (or in another moment). I2, I3, and I19 explain that this concept is crucial for the instructors to bring together students who are having difficulty and so that they can try to resolve their doubts. In addition, they explain that this allows the instructor to apply interventions to groups of at-risk learners to try to leave them in the same class level. I6 opines that this concept can be more helpful in remote or blended learning, in which the instructor may have little or no contact with the students. As a suggestion to improve this concept, I7 and I9 stated that it would be important to put a calendar with checkpoints that determine when they should visualise a report with the list of at-risk students. For example, this checkpoints could be in each assignment deadline or when the instructor finishes teaching a course topic. Moreover, I2 and I11 point out that identifying the students' difficulties would also be important. If the instructor knows in which part of the course the learner is struggling, then they could identify which topic was taught at that moment.

As a caveat, I16 states that it would be complex to assemble the at-risk students

without making them feel diminished for being in a low-performing group. In the same sense, I14 points out that he would use the concept as long as students did not have access to this information. Finally, I8 states that the concept is quite prominent for large classes, but it would not be as relevant for small classes.

Concept 4 (*inspecting individual effective and ineffective behaviours*) was highly rated (mean = 0.95). Almost all instructors (N = 21) agreed that this concept is relevant, although some have put some caveats to the use. I6, I13, I15 I19, and I20 stated that the combination of information revealed by this concept with the instructor's experience could be of paramount importance for the construction of effective interventions. For example, identifying which copy and paste and repeated error behaviours may affect the student's performance. It would allow the instructor to reflect on the lesson to re-plan the class to cover something about debugging or even reinforce the importance of writing the code from scratch when starting to the program. Still, on the example, I10 points out that the instructor could try to deepen the analysis to identify which parts of the code the student is copying and pasting so that he/she could identify whether the copy-paste behaviour is harmful or not. I18 summarises that this concept can help the instructor identify minimum behavioural changes needed to boost student performance. Besides, I9, 12, and I13 believe that this concept would help adopt predictive models as it would bring more confidence about how predictions are performed. On the other hand, I2 points out that it is tough to understand this concept. I3 also explains that it would use this concept if it is presented in a didactic way, with an intuitive interface or plot. I16 also explains that it is important to present only the most effective/ineffective behaviours to the instructor, not very extensive information (lightweight). In addition, I3 and I16 opine that it would be interesting to point out some recommendations for the instructor on what to do given the information presented in this concept. Finally, I2 and I13 explain that a potential limitation of this concept is that it will not consider external variables that the online judge cannot detect.

Concept 5 (*inspecting groups of effective and ineffective behaviours*) was highly rated (mean = 0.91). This concept is helpful as a complement to concept 4. To illustrate this concept, imagine that a student named Jack has 30% chance of passing based on data

collected until the fifth week of class. Suppose further that based on inspection of the predictive model, what caused this low percentage was that the student procrastinated too much on the first two assignment lists and made many consecutive identical errors, many of which were not resolved. With that, this concept serves to identify which are the *'Jacks'* in the class, that is, which students had behaviours similar to *'Jacks'* and which led to an approximately 30% chance of passing as well. With this, the instructor can carry out one intervention in the whole group of *'Jacks'*. I16 explains that the concept allows the instructor to tackle what can generate learning problems for students since the concept group's difficulties are related to assignments and topics that can be reinforced. I7 and I12 explain that this concept would be viable for them to identify opportunities to recommend pedagogical materials (e.g., videos, tutorials, extra exercises) to the groups. Instructors I5, I11, I14, I18, and I20 point out that the concept is crucial, especially for large classes. I14 and I18 highlight the importance of the concept for personalising pedagogical and intervention strategies that the instructor can provide to different student groups/profiles. Indeed, I4 explains that he already tries to do manually, intuitively, and empirically what the concept proposes and that, therefore, the concept would increase his ability to perform this task.

As a possibility for improvement, IP1 explains that it is interesting to add some labels in the groups to facilitate the search later from the teacher's point of view. For example, add labels for low-achievers who are procrastinators and who access the system with low frequency so that the instructor could easily find such students in a single search. I5 opines that the concept could also provide a behavioural filter so that the teacher could assess each group's behavior individually. In addition, IP2 states that it would be interesting to point out the percentage of the class that has a certain effective or ineffective behaviour. I6 explains that despite the importance of the concept, it must be presented to the teacher in a graphic or intuitive interface, otherwise it would not be useful. Furthermore, it would be great if the teacher had access to a set of strategies or intervention options given the group's effective and ineffective behaviours presented in this concept. Finally, I6 highlights that the concept would help in the interventions for the teacher. Still, for the student it would not be helpful to know

that their performance is classified in such a profile or group, especially if this would stigmatise them, conveying the idea that there is little chance of change.

Concept 6 (*inspect effective and ineffective behaviours of the class*) was also highly rated (mean = 0.86). Some teachers (IP1, I3, I11, and I12) claim that the visualisation of effective and ineffective behaviour by class is convenient for the instructor to have an overview of the class's difficulties and facilities and adapt the methodology based on this. Others (I4 and I19) explained that they already carry out this general analysis that the concept proposes manually and based on their perceptions. Therefore, for I4 and I19, the concept would help them to expand and even confront their perceptions. I6 and I15 opine that the concept would help teachers provide general technical and motivational guidelines for the class based on evidence-based data that teachers manually would not be able to obtain and synthesise if the concept is not available systematically.

As suggestions and a possibility for improvement, IP2 points out using macro graphs in this concept, where instructors could zoom in on the plot to visualize each student's behavior in the class. I14 warns that the concept is relevant if it is made available at appropriate times during the course, such as at the beginning of the course, before the deadline of an assignment, or when there is still time to intervene in the class. I16 suggests that message templates based on the results of this concept (and other concepts as well) should be available for the teachers so that instructors can customise these templates and send personalised messages to students from different classes without too much effort. I20 points out that the concept would be interesting to provide a temporal filter for classes to compare a given class with other classes in that same time window. It would help instructors to map good and bad methodological practices used in these classes from different periods. Furthermore, it would be useful for the teacher to test the power of generalising strategies with classes whose students may have other behaviours and belong to different generations.

On the other hand, I7 and I17 state that confidence in use may vary depending on the programming behaviours used in the concept. I18 explains that he doesn't know if he would use the concept because the students' behaviour can vary a lot in a given class. As such, this central measure or general behaviours of the class presented in this

concept may not be as accurate.

Concept 7 (*inspect effective and ineffective behaviours per student in a lightweight plot*) was highly rated (mean = 0.86). IP1 points out that the concept provides an intuitive visualisation through a bar plot ordered by the importance of the student's behaviour in his/her predicted performance. However, only this instructor (IP1) assessed that a lightweight view is more appropriate than a more detailed plot such as a decision plot or force plot. Four instructors (I1, IP2, I5, I14) prefer a more detailed view following concept 4, while 17 instructors (I2, I3, I4, I6, I7, I8, I9, I10, I11, I12, I13, I15, I16, I17, I18, I19, I20) prefer both visualisation to be available.

I12 explained that the concept is important to cross-reference the information with concept two quickly; that is, when a student inflection point is detected, it would be helpful to assess behavioural changes (using this concept) from before and at the time of change. Moreover, as of concept 4, the evaluations were generally positive. Even the suggestions for improvement and limitations pointed out were similar to concept 4.

Concept 8 (*comparing students behaviours against low, border and high reference values*) was highly rated (mean = 0.77). Some instructors (IP1, I5, I13, I15, I16, and I19) explained that having the option of viewing the student compared to low, medium, and high-performance references allow the teacher to carry out a systematic analysis of the students, what is important to enable the adaptation of classes and provide personalised support to the student. Furthermore, these instructors stated that the concept allows for interventions with greater chances of effectiveness. I5 points out that the idea of mapping student behaviours based on reference values would be complementary to the results of the previous concepts, enabling the instructor to cross this information with the detection of effective/ineffective behaviours and students' probabilities of passing. I19 points out that this information, along with some calibration in 5 years, can bring important values for a deeper analysis. For example, the instructor can conduct a longitudinal study of why students tend to decrease their performance throughout the course.

On the other hand, I3 and I14 point out that they would have doubts about using this concept since there is no recommendation associated with the information revealed

in the concept. I18 opines that it would take a long time to evaluate each student's graph and that the graph is complex as a complicating factor. In this sense, I17 points out that the amount of information (represented in the axes of the radar plot) presented in the graph is a potential reason why making the plot complex. I17 suggests presenting a few variables to make the plot simpler, more precise, and more intuitive. Thus, the ideal is possible to group the variables in different radar plots, for example, by presenting a radar plot with only code metrics variables (lloc, comments, etc), Another only with measures of effort (e.g., number of attempts with resilience - Chapter 2).

Lastly, IP1, I4, and I12 explain that this concept would be more critical for the student than instructors, as it would allow the students to reflect and perhaps self-regulate their learning. On the other hand, I6, I7, and I8 point out that the concept is relevant for the instructor, but they would not make it available to students as it could make them feel inferior or even overestimated, depending on how close they are to the references values. One alternative use of the concept is for the instructor to choose whether a student can visualize it or not. Furthermore, an analysis (using AI) of the student affective or emotional profile can help to choose which students should be exposed to this resource.

Concept 9 (*comparison of students behaviours with the class average*) achieved a moderate rate (mean = 0.59). This concept is a variation of the previous concept. In this case, the reference value is the class average. Instructors generally point out that both concepts bring relevant and complementary information. Only two instructors (IP1 and I5) opined that they preferred to use only concept 8. In this sense, I18 points out that the concept is helpful for the student to have a notion that specific problems take work to solve. For example, after the third failed attempt to solve a given question, the learner could visualise that the class's average attempts are six. Therefore it's expected that they made some mistake to solve that question. I12 opines that the concept would be relevant for students to visualise questions that have already been resolved by other students in the class and the effort required to solve that questions. It would attract former students to try to resolve the most popular questions. Still, I16 suggests a competition/comparison among the classes to engage more students. The instructor

stated that a plot could be shown to students comparing the classes. Furthermore, the teachers repeated the comments on the previous concept about concerns about making this concept available to students.

Concept 10 (*creating one-fits-all assignment lists*) was evaluated for all instructors as relevant (mean = 1.00), achieving a high rate. IP1, IP2, I2, I5, and I14 point out that the concept helps to optimise the instructor's time, which is relevant given that the teaching workload is generally high. I2 states that this concept is fundamental since problems are selected to compose assignments are recurrent throughout the terms. The CS1 instructor usually prepares many lists of exercises on different topics to reinforce and consolidate the students' learning. I18 explains that this concept is already used for some educational systems in majors but not for programming courses. Therefore, the concept would be a great addition to education systems to support CS1 classes. In addition, instructors I2, I13, I15, and I20 state that it is essential for the instructor to collaborate with AI in this concept so that they can validate the list before presenting it to the students.

As improvements, I1 explains that it would also be important to point out which exercises should have greater weight because they carry key subjects or require abstractions necessary for learning a given topic. I5 and I12 also state that it would be interesting to define the subjects even more fine-grained. Instructors may choose to have the assignment built on nested conditional structures, sequential structures, and other topics provided in the CS1 syllabus. Nonetheless, it would also be essential to have something more specific like basic arithmetic operations, increments, control variables, numeric sequences, etc. Finally, I11 clarifies that it is expected for the concept to avoid repetition of questions from lists from other semesters or other classes to avoid plagiarism, which is quite frequent in CS1.

Concept 11 (*scaffolding assignment lists*) achieved a moderate rate (mean = 0.68). I1, I2, I5, I9, I14, and I15 state that the concept is relevant to support and evaluate the students' learning process according to their differences in their learning pace. Specifically, I15 explains that the concept would help look at students according to their individualities. I6 explains that this concept would also help more advanced

students (fast-learners or who already have previous programming experience) get frustrated when they are either not challenged or need to solve fundamental problems. I14 states that the concept would be helpful even to find monitors to support future CS1 classes, based on the student's group and whether they could solve the most challenging questions. For I13, the concept would apply very well to students in Non-CS courses. According to the instructor, some students of Non-CS courses intend to learn just the basics of programming without having to face many challenges since their focus is the content directly related to their course. In these cases, according to the instructor, more straightforward assignments would be suitable to the intention of this student profile.

IP2 and I20 show that despite liking the idea, the concept can make it difficult for the instructor to identify how to employ the class level since assessments of different difficulty levels in the same class. That is, there would be no reference. Still, IP2 points out that she/he would use this concept to create different lists or challenges, whilst I20 explains that he/she would have doubts. I5 and I12 also reported that they doubt whether the concept would mask students' learning development who solve the most straightforward problems. Since they would not solve some problems (of a more challenging level), that may be key to learning some concepts taught in the course. Despite doubts about the implications of using the concept, both instructors (I5 and I12) stated that it is relevant and would use it in their classes.

I11 and I18 explain that they would not use the concept because it segregates the class in a potentially harmful way. For instance, I11 explains that the concept could bring psychological burdens to students allocated to the low-achievers group, making them feel less capable. I16 and I19 also point to this possibility of segregation. Still, these instructors explain that a solution would be to use this concept as a reinforcement assignment only for students struggling in the course. Instructor I16 also states that communicating this would also be compassionate for the student to see the concept as a possibility for improvement and not as a way to allocate it to a group inferior to the others.

Concept 12 (*creating variations of one-fits-all assignment*) was highly rated (mean = 0.91). IP1, IP2, I1, I4, I5, I7, I11, I15, and I20 state that the concept is important to

help reduce the chances of students' plagiarism, something recurrent in CS1. I5 and I17 explain that they would be even more inclined to use the concept in -learning or blended courses, where the practice of plagiarism can be even more frequent. I12, I17, and I18 point out that they already do this (generate multiple versions of an assignment) manually, so the concept would help a lot. I2 adds that she/he would also like accessing a way to evaluate the similarity rate of the codes or some metric that points to the possibility of plagiarism among the students' solutions.

I9 and I16 explain that they would use the concept as long as there were not so many assignments validations to be performed by them. Specifically, I16 points out that she/he wouldn't use the concept if she/he was pushed for time. However, according to I2, over time, the validation process tends to become faster if the instructor observes that the recommended assignments are accurate. On the other hand, I19 points out that she/he would not use the concept because it can make it more difficult for the instructor/monitor to address students' doubts. I19 explains that as there would be several different assignments, the instructor would need to have an in-depth knowledge of solutions to more questions, thus increasing planning time and teaching workload.

Concept 13 (*creating individual assignments for each student*) achieved a low rate (mean = 0.18). Despite the low rating, more than half of the instructors rated the concept as relevant (N = 11). I4 and I5 explain that the concept would probably be more helpful for small classes or for creating challenging assignments with few questions. IP2 points out that the concept is great. However, it must be challenging to implement as it would need a large number of questions registered in the online judge to create different assignments for each student.

Many instructors (I3, I9, I12, I13, I14, I16, I17, I19, and I20) explain that they would not use the concept because it would require too much time to validate the individual assignments. Specifically, I16 states that validation can be a big issue for large classes. I19 and I20 point out that, similar to the previous concept, it would be even more challenging to help students address code errors with different questions for each student.

Concept 14 (*creating one-fits-all exams*) was highly rated (mean = 0.91). IP1 ex-

plains that the validation should be accompanied by the exam used as a reference. In this way, the instructor could compare the similarities of the questions to judge equivalent questions between the reference exam and the recommended exam. I6 states that it would use this concept but would not use the automatic correction provided by the online judge. According to the instructor, a well-crafted assessment can directly benefit student learning, especially in exams. I17 and I19 report that they would perhaps use the concept as a starting point, modifying the necessary questions and adapting them.

I11 points out that the questions that had already been used in previous assignments should be labeled to facilitate the instructor's mapping of items that may be repeated in the assignments and exams. Two instructors (I5 and I17) were unsure about using the concept because they typically use exam questions covering key concepts in the assignment lists. In addition, the instructors (I5 and I17) explain that they customise the exams based on the observations they make in face-to-face classes. I18 adds that she/he often puts on the exams questions related to specific subjects that she/he has covered or given tips during face-to-face classes. These instructors (I5 and I17) believe that the concept might not meet these requirements.

Concept 15 (*creating variations of one-fits-all assignment*) was highly rated (mean = 0.86). The instructors were consistent with the evaluation of this concept and the previous one (concept 14). The main difference is that they argued that this concept can be helpful to avoid plagiarism but might increase the instructor's overload in validating the exams. Only instructor I17 marked the concept as -1, while she/he marked the previous concept as 0. The justification is that the exams must be identical for the instructor for all students to be fair. If it were to change something to minimise the plagiarism practice, it would only change the order of the questions in the exams. Furthermore, I18 explains that the system should choose the reference exam based on the similarity of the classes. For instance, if the class from the first term of 2016 is similar to the class from the second term of 2018, the system would automatically reference the questions used in the 2016 exam to recommend the questions for the 2018 exam.

Concept 16 (*creating individual exams for each student*) achieved a low rate (mean = 0.41). Although most instructors (IP2, I1, I2, I3, I4, I6, I7, I8, I9, I10, I11, I18) believe

that the concept can be beneficial in making plagiarism impractical, some (IP1, I3, I5, I12, I14, I16, I17, I20) explain that the use of the concept may be unfeasible as it requires a lot of work for validation, especially in classes with many students. Some instructors pointed out specific cases where the concept could be helpful. To illustrate, I3 and I12 state that they would not use this functionality to apply an exam to the entire class. However, I3 and I12 explain that they would use it to apply for a recovery exam or a final exam when the number of students is usually much smaller. I4 and I9 state that they would feel more confident using this concept in online exams as a way to mitigate plagiarism.

Concept 17 (*personalised notification*) was highly rated (mean = 0.95). I1, I3, and I4 explain that this concept is crucial to complement the previous ones. It allows the instructor to perform the intervention remotely for a group or even directly for a student, given the feedback or data provided in the other concepts. I1 points out that the concept is essential, especially if used from the beginning of the course, to enable the recovery of students at risk or to leverage the learning of high-performance students. I5 states that the system should allow data and graphics from previous concepts (e.g., Concept 1-9) to be coupled to the notification so that students can obtain more meaningful feedback. I20 explains that she/he already tried to do what this concept proposes manually in her/his classes and, hence, the functionality would be beneficial for her/him.

I7 and I9 state that it would be necessary for the concept to provide a message template with some standard feedback that could be reused so that the teacher could contextualise the template or use the standard feedback. I8 suggests that it is possible to change the list of students in a group that will receive a notification. For instance, when using the notification associated with concept 5, the instructor might want to remove a given student from the group that will be notified, either because she/he believed that the student has been misallocated to the group or because the student has already dropped out of the course and, therefore, it would no longer make sense to receive a notification. I13 and I17 suggest that messages for groups be stored in a forum so that students in the group can interact with each other and with the instructor asynchronously.

Finally, I6 states that she/he would use the concept only to deal with groups, as sending messages individually to students would be too much work. I17 explains that she/he would have doubts about the use because I17 believes that this communication should be dealt with in person.

Concept 18 (*problem recommendation*) was evaluated for all instructors as relevant (mean = 1.00), achieving a high rate. I3, I6, I11, I12, I19, and I20 state that it is important for the recommender to take action after the students finish the assignments. Students first meet the requirements and then explore their potential by solving additional recommended questions. I4 explains that he would use the recommender mainly to replace questions that she/he thought would not be valid in the recommended assignments/exams proposed in concepts 10-16. According to the instructor, this would facilitate the process as it would not be necessary to exhaustively search for substitute questions in the OJ's question database. I19 points out that it would be interesting to use the recommender during classes so that the instructor could select questions to solve in class with the students.

I2 points out that it would be relevant to have a way to monitor the effect of recommendations on student outcomes. In the same sense, I16 states that students could classify the recommendations. According to the instructor, it would be helpful for the instructor to assess the impact of the concept and continually improve the recommender system.

Concept 19 (*help button for each concept describing its functionality*) was highly rated (mean = 0.94). Only one instructor (I17) had doubts about using this concept. The others rated the relevance of this concept as positive. The teachers explained that a button is necessary to explain the functionalities of a system. I5 points out that an explanation (meta-data) for each programming behaviour used in the novel human/AI OJ architecture is also important.

However, I5 and I19 explain that it would be important to present this concept practically and intuitively to the teacher, for example, in a video tutorial or a storyboard format. I5 and I16 point out that they probably wouldn't use a text tutorial. I7 and I16 explain that it would also be relevant to have a "more details" option for some teachers

who want to understand some details of the concept, for example, in concept 1, to know the accuracy of the early prediction. I17 opines that she/he has doubts about using it due to lack of time. The instructor explained that if she/he needs to access the help button to understand the use of the tool, then she/he tend not to use it. Ideally, the tools should minimise the need to have another page explaining the tool itself, she/he said.

7.4.3 General Analysis

In the following, we present a list with concepts sorted by the average in descending order and grouped by our classification rate:

- Highest-rated concepts: Concept 2, Concept 10, Concept 18, Concept 4, Concept 17, Concept 19, Concept 1, Concept 3, Concept 5, Concept 12, Concept 14, Concept 6, Concept 7, Concept 15, Concept 8.
- Moderately-rated concepts: Concept 11, Concept 9
- Poorest-rated concepts: Concept 16, Concept 13.

We can observe that instructors tend to evaluate approaches to group interventions more positively than concepts that enable individualised interventions and require too much work for validation. For example, in concept 13 (lowest rate achieved), the instructor must validate a personalised assignment for each student. Notice that this concept would become unfeasible in a class with 100 students as it would dramatically increase the teaching workload.

On the other hand, the instructor better accepted the concepts that tend to group similar students in different aspects (based on behaviours, estimated performance, etc.). However, this does not mean the individual concepts are not relevant in our hybrid human/AI architecture, where the instructor is always the last node in the pipeline, responsible for making the decision. According to most instructors, the concepts that allow individual student analysis or intervention can be used for quick reference and in specific cases. For example, concepts based on XAI strategies (e.g., Concept 4) can increase teachers' confidence in predicting and driving human/AI collaboration. The

instructor can combine the information they have access to during lessons with the AI's decoded information.

Furthermore, as in CS1, instructors typically need to create many assignments and exams. The concepts that facilitate this process tend to be well accepted since they free up instructors' limited time and attention for other relevant tasks. Moreover, variations of assignments/exams tend to make the plagiarism practice more difficult, which explains why concepts 12 and 15 were also well evaluated. However, instructors poorly evaluated the cases of creating an assignment/exam for each student (Concept 13 and Concept 16). The first reason is that it would dramatically increase the instructor's work to validate an assignment/exam for each student. Besides that, some instructors explain that students would lose a reference for the assessment when using these concepts as there would be many variations. Still, some instructors point out that it would be challenging to clear the doubts of students with such a large number of different problems available in many different assignments/exams variations. Varied doubts would tend to arise, significantly increasing the number of exercises that the instructor should master.

Briefly, many of the concepts are good indicators for instructors providing "what if" questions that would aid in testing the different hypotheses of interventions and pedagogical strategies and monitor their effectiveness during the course using the concepts. They also help increase instructors' control of what happens during and after the interventions. Our findings can be used to fill a gap between the design and implementation of hybrid Human/AI OJ. Designers, developers, instructors, and policymakers within universities might use our novel architecture as a reference in such an aim.

Finally, our results provide support to address our research question (RQ7-1). Indeed, we show the relevance (quantitative and qualitative) of design concepts based on our methods, by validating them over the perspective of CS1 instructors with previous experience with OJ systems.

7.4.4 Hybrid Human/AI Collaboration

Our architecture can provide ground for superior achievement for both AI and instructors. A possible way is by aggregating the output of these agents to achieve an excellent outcome on the social and technical levels. In addition, human teaching machines and humans mediating the AI feedback is crucial for the AIEd adoption and safety, avoiding biases from the AI such as discrimination or stereotyping against students. Additionally, as instructors can learn students' preferences during the interaction, the AI can achieve a better level of personalisation using such a hybrid human/AI approach.

Mainly, the instructor will augment the AI perception by showing some unseen pattern or relabelling a student's description or prediction. For instance, the sequence of problems in an assignment that an AI system will recommend for the students might be selectively overridden or replaced by the instructors. Notice that when instructors correct the AI prescription, the decision (perception to action) is augmented. Indeed, with this new annotation from the instructor, the AIEd system can learn from that to prepare better assignments next time. Similarly, the instructor can improve their ability to create assignments by reflecting on how the AIEd system does that.

Additionally, it is important to consider what kind of change the support is expected to generate in learners in education. Using the hybrid human/AI architecture presented in this paper, the instructors might intervene proactively (see concepts 1-6) using the AI info with the goal of behavioural, cognitive, or even metacognitive change that might lead to better students outcomes. For example, suppose a student is making many consecutive syntax errors, which is related to low achieving behaviour (Concept 8). In that case, one of the possible actions from the system is to send a notification to the instructor or even ask the student whether the system should alert the instructor about that. Notice that without such AI support, the instructors' ability to adapt the methodology to improve the learning process of a group of students is limited to the instructors' repertoire and perception.

Moreover, it is essential to verify the moment that the system should perform a decision to perform some prescription. Regarding the granularity, the frequency of the prescription might be performed per task or step. When using the architecture,

both agents (instructors and AI) can be empowered to when/how to perform many tasks, such as presenting an example of code, providing feedback about the debugging process or the type of error the student got, or even asking the students to explain an example of code themselves (self-explanation).

Furthermore, the instructors might train the AIEd system about their exclusive expertise and methodological/pedagogical preferences, essential for scalability. The AIED system could then reach more students than a human being can.

Finally, relating to AI goal augmentation, our work makes possible improvements in achieving the intrinsic AI goals of maximising an objective function. For example, when an instructor relabels an instance misclassified, thus the AI algorithm has the opportunity to improve its rules to maximise its objective functions better.

7.5 Chapter Conclusions

Effectively improving the way instructors teach and, hence, the learning process in CS1 is a complex and dynamic problem since it is time-dependent and has no ground truth. Thus, it requires analytical skills that combine pattern recognition, consistency, probability theory, speed, and efficiency that the AI could provide through data analysis. On the other hand, this complex problem also requires intuition, flexibility, common sense, empathy, and creativity, in which humans are notoriously better than machines. Thus, combining this heterogeneous and complementary intelligence can achieve higher results than if they worked isolated.

As pointed out in our results, there are many possibilities of AI potentially augmenting the instructors' capabilities to improve students' learning. For instance, in concepts 1-3, the instructor's perception is augmented by the AI with information about the probability of students failing or passing. Still, concepts 4-7 point to explaining the predictive model's estimation. With this in hand, the set of potential actions the instructor can take is expanded due to the possible causes the explainable predictive model points out that jeopardise the student's learning and, hence, their performance. The instructor's decision-making is also empowered to carry out effective interventions.

As this explanation about a single student can be grouped by similarity (e.g., in Concept 5), the instructor's intervention can also be generalised for a group instead of to each individual. It not only delivers the adaptability needed but also provides workload reduction for instructors who can do one single action for the entire group of N students instead of N similar actions for N different students.

8

Conclusions

> To know what you know and what you do not know, that is true knowledge.
>
> - Confucius

In this study we propose and validate AI methods for descriptive, predictive, and prescriptive analysis to support cs1 teaching and learning. To feed these methods, we employed a programming profile from CS1 students, which is a set of features that represent the learners' effective and ineffective programming behaviours. We extracted these behaviours from fine-grained data collected from OJ systems. Thus, using this data, we model how students solve problems on a built-in IDE, embedded in the an OJ system called Codebench.

We also show in this work that our methods surpassed cutting-edge solutions proposed in the educational literature. Moreover, we demonstrated that concept designs derived from our methods can be used in a novel hybrid human/AI architecture that extends the typical OJ architecture. We validated those concept designs with CS1 instructors with previous experience with OJs. In other words, our qualitative analysis provides empirical evidence that our methods can be used as a human/AI hybrid complement for OJ. Such evidence is important to enlighten the field of CS1 teaching and learning, and to fill the gaps pointed by the literature to design and validate hybrid systems (CHEN *et al.*, 2018; DE-ARTEAGA *et al.*, 2020; HOLSTEIN *et al.*, 2020; TENÓRIO *et al.*, 2021).

Still, our findings are potentially useful to help instructors to improve their pedagogical and methodological practices, enabling early and long-term interventions that minimise the chances of at-risk students to end up failing, while enhancing the chances of students with a high probability of passing. Moreover, we showed that our prescriptive recommender can be useful to minimise the instructors' workload on selecting problems to compose assignments and exams. It is expected that such support for the instructors would be reflected in an learning enhancement. Thus, the students are benefiting as well. As such, we believe that we achieved our goal of designing and validating a hybrid human/AI OJ architecture to create mechanisms to support the decision-making of CS1 instructors and students.

Briefly, our hybrid human/AI OJ architecture provides developers and designers with design concepts to build OJ environments that can improve CS1 teaching. Moreover, instructors can benefit from our findings by observing the usefulness of the AI method applied to OJ systems in CS1. Finally, we provide discussions about the directions (based on the instructors' perspective) of novel hybrid approaches to support CS1 learning. Indeed, our findings can be used to reference designers, developers, instructors, and policymakers within universities.

8.1 Findings, Applications and Implications

Just to recap, following are the findings of this work:

- Creating a programming profile using features collected from a new Online Judge system, CodeBench, which allows fine-grained descriptive, predictive, and prescriptive analysis of student behaviour for CS1.
- Employing descriptive and predictive analytics to identify early effective behaviours for novice students and, for the first time in our knowledge, how these behaviours can be useful for ineffective students (prescription).
- A novel classification of students into effective, average and ineffective, based on their behaviour, which shows both semantic and significant statistical differences.

- A clear indication that student behaviour during programming influences learning outcomes for CS1.
- A proposal, design, and implementation of a large scale, longitudinal study of student behaviour in CS1.
- Cutting-edge classification performance for early performance prediction using a large scale, longitudinal data from introductory programming students.
- Going one step further than binary classification and constructing an interpretable stacking method that combines deep learning and easily explainable regularised linear regression model.
- A new non-linear interpretable predictive model as an important move towards Explainable, Transparent AI in Education, by demonstrating how to explain the predictive model's decision (individually and collectively), to better support students and instructors (and other stakeholders).
- Using our programming profile, we constructed and validated a novel behavioral recommender system based on students' expected effort to solve a given problem.
- Showing, through a double-blind controlled experiment, empirical evidence on how personalised recommendations based on effort influence achievement and affective states.
- A new, holistic methodology pipeline for the OJ contextual labelling problem, allowing to compare a variety of cutting edge shallow and deep learning models, to experiment with the most recent data augmentation techniques (with or without augmentation), NLP (based on BERT, Word2Vec, Glove), classifiers (based on BERT, Random Forest, SVM, XGBoost, GaussianNB, GradientBoosting, ExtraTree, Sequential ANN, CNN, RNN) and validation.
- Proposing and validating a novel human/AI collaboration method to select programming problems to compose assignments/exams in CS1 courses;

- Empirical evidence, validated by CS1 instructors, that our methods (represented as concept-desings) can be used together as a human/AI hybrid complement for OJ systems.

We also demonstrated that these findings have many pedagogical implications for the teaching and learning process in CS1. Some of these implications will be explained next.

First, it is worth remembering that CS1 classes usually have high heterogeneity among students. This was clear in this work, showing variations in the patterns of student behaviours, resulting in three different clusters (effective, average, and ineffective students). We further supported the findings via statistical differences in learning outcomes, programming behaviour and evaluative factors between these clusters. Importantly, our analyses showed which early programming behaviours potentially indicate effectiveness or ineffectiveness in learning.

Identified effective behaviours can be brought to the attention of instructors, effective, but also ineffective students – with care about non-disclosure of personal information. For example, late students can be warned when a certain number of students complete assignments or spend more time coding in the IDE than they do. Here, again, group membership can inform the feedback, and ineffective students comparing themselves with the best amongst their group, as opposed to the best in class, which may be tough for them to “beat". Additionally, the Online Judge can notify instructors about which students need extra help, in good time before any deadlines, allowing proactive instead of reactive pedagogical interventions. Finally, it is worth noting that, whilst correlation or even association rules *per se* do not imply causation, the two-pronged triangulation approach used is providing more evidence towards prediction power. Moreover, undesirable behaviours may need to be addressed in some cases, even if they may not directly cause ineffectiveness.

This result can support decision making of students as well as instructor intervention, such as designing specific guidance for a struggling group of students, proposing new and challenging exercises for effective students, and personalising exercises, according to different student needs. Furthermore, knowing which behaviour can

be effective might help students to improve their self-regulation and awareness of what kind of programming behaviour can be dangerous or beneficial to their performance.

Additionally, with early prediction, in a standard course, instructors could provide extra assignments for the high-achieving group and personalised support to those who are struggling. If the strategies of effective novices can be identified, it may be possible to promote effective strategies to all groups. Such early prediction allows personalised feedback, but this is not scalable to large classes without proper technological support. Such process of early intervention can be performed using dashboards, e-mails, etc. In other words, as *prevention is better than a cure*, likewise, it is better to prevent students from failure as soon as possible, instead of finding out students are struggling when their poor marks come in.

Most importantly, our high-performance predictive model is explainable, which can facilitate human/AI collaboration towards prescriptive analysis, where the instructors/monitors will have access to individual and collective analysis on which student behaviours should be encouraged and which ones should be inhibited. On the student side, such analysis can promote self-regulation and awareness of their strengths and their chances for improvement. To illustrate the usefulness of the approach from a student's point of view, they may trust more on a recommendation if they understand why they received it. From the instructors', understanding why students are failing or passing would allow them to apply effective efforts to tailor pedagogical material, instructions and interventions for future classes.

For a more generalist analysis for adaptation of instructional decisions, we presented the power of global explanation by the identification and analysis of typical prediction paths. Moreover, our focus not only on global behaviours but also on individual ones enabled by visualising and analysing feature effects at single-student granularity level can be used in an unprecedented variety of pedagogical applications.

Indeed, these early prediction empowered by its explanation might potentially allow an effective early intervention by the stakeholders. To illustrate, our interactive force plots of each student might be shown to the instructors at the end of the second week of course, who in turn might create some proactive way of minimising the chances

of at-risk students end up failing and creating more challenging tasks for the students with high probability of passing. In addition, for instructors and coordination, a visualisation dashboard including our force plots, decision plots (so forth) might contribute for a more formative assessment, in which not only the learner product is evaluated but also the process behind, that is, not only their codes are evaluated but also their learning paths and effort to produce their codes. Still, this dashboard combined with the interaction of instructors and students can increase the chances of the instructor reflecting and diagnosing potential causes of the students' lack of success.

More than that, we found that when provided with programming assignments recommended based on their behaviours, students showed higher achievements rates and lower failing and dropout rates in problem-solving. This finding suggests that our recommendation approach contributed to maintaining students trying to solve the assignments (less dropouts in problem-solving), preventing them to fail, and enhancing their achievements. Therefore, this finding implies the need to provide adequate recommendations for programming students to practice, instead of relying on their own non-guided choices when performing self-direct learning. Moreover, our behaviour recommender system can be used to facilitate the instructors' work when selecting problems to compose the assignments and exams. To illustrate, typically instructors need to create variations of programming assignments lists for different classes, in order to avoid plagiarism, for example. Using our method, consider each problem in a list of exercises already created by a instructor as a target problem. By generating N recommendations for each of these problems, we can compose N new lists of exercises that require effort and knowledge (same topic) similar to those required to solve the original. Thus, the instructor's workload to compose new programming assignment lists is significantly reduced.

Additionally, as OJs have large numbers of problems registered in their problem databases (WASIK *et al.*, 2018), in principle, there would be plenty of problems to select from, for both students as well as teachers, allowing for a mass personalisation - where one teacher could cater in parallel for the needs of many students. Nonetheless, as explained, the problems available on these systems often are collected or scraped from

various environments that do not provide labelling (ZHAO *et al.*, 2018), and thus it is laborious to find appropriate problems for CS1 students. Indeed, OJs might have problems that span over different topics. Understanding such categorisation allows learners to engage with specific algorithm techniques and could improve their skills. Furthermore, for curriculum design, it is essential to consider problem categories. In this sense, our NLP pipeline can be used for tasks of categorisation that might help CS1 students and instructors from CS and Non-CS courses. For CS learners and instructors, our model can be used to potentially address a limitation of our behavioural recommendation (as explained in Chapter 6).

Therefore, we, for the first time, to the best of our knowledge, are going in direction to fill all the research gaps of CS1 teaching and learning by conducting a longitudinal study, using fine-grained data from 2058 students. Moreover, we reached the prescription step, besides descriptive and predictive. We produced methods of individualized and group support for instructors and students by using a programming profile. Thus, we believe that this is an important move toward the construction of a human/AI hybrid OJ system, where learners would use AI-based recommendations, mediated by the instructors, to improve their learning through effective programming solving practice at the same time that reducing instructors' workload.

Finally, based on the validation of our design-concepts, we can observe that instructors tend to evaluate approaches to group interventions more positively than concepts that enable individualised interventions and require too much work for validation. However, this does not mean the individual concepts are not relevant in our hybrid human/AI architecture, where the instructor is always the last node in the pipeline, responsible for making the decision. According to most instructors, the concepts that allow individual student analysis or intervention can be used for quick reference and in specific cases. Indeed, the instructor can combine the information they have access to during lessons with the AI's decoded information.

Moreover, many of the concept-designs are good indicators for instructors providing "what if" questions that would aid in testing the different hypotheses of interventions and pedagogical strategies and monitor their effectiveness during the course

using the concepts. They also help increase instructors' control of what happens during and after the interventions.

Finally, the validation of our concept-designs can be used to fill a gap between the design and implementation of hybrid Human/AI OJ. Designers, developers, instructors, and policymakers within universities might use our novel architecture as a reference in such an aim.

8.2 Limitations

Among the limitations of this study is the behavioural data collected from a single institution, which may affect the generalisation of the results. However, considering data was collected from programming courses from several years and students from different majors, this limitation might be reduced.

It is indisputably important to provide human-friendly feedback to improve the students learning. Here we are going in this direction. However, our predictive methods are not 100% accurate and, hence, the method's feedback might not be precise in some cases. In this sense, we believe that a first attempt to employ our method should be done by combining humans and our AI, that is, our pipeline could be later validated with a human pipeline of experts, as we proposed in our human/AI architecture.

Moreover, in this work, we performed statistical inference and not causal inference. Our interpretable pipeline using SHAP values, however, offers clear insights to formulate causal hypotheses that could be assessed in future works. As a potential internal threat, we did not tackle plagiarism in-depth and some successful programming behaviours may have been misclassified. We used some features to try to encode the plagiarism such as *attempts*, *eventsActivity*, *copyPaste* and *ideUsage*, since learners who copy codes from others students usually do so from the first attempts (lower number of attempts) and actually spend little time programming (low use of IDE). To confirm that, we plan to carry out further studies.

In terms of recommending/prescription of problems, it is worth noting that the effort required to solve a given problem depends on the previous knowledge acquired

by the learner about the programming topic of that problem. The way in which the level of effort required to solve a problem in the recommender system based on behaviour was modelled in Chapter 5 does not take into account this prior knowledge that the student has about the domain. Thus, a potential limitation of our behavioural method (presented in Chapter 5) is recommending an easy problem from a difficult category for a student who wants an easy problem from an easy category. As a way to solve that problem, we take into consideration the categories of problems needed to be specified on each problems. Notice that a challenge of performing such analysis is that problems in general are not annotated with this information in OJs. Thus, we constructed a pipeline that automatically detect the problem category based on the problem statement. We then carried out experiments in Chapter 6 demonstrating the importance of the categorisation for the recommender. However, this task of categorisation could be further extended to a multi-label classification, where one problem could potentially be labelled with multiple labels/classes.

One limitation of our work is related to the number of CS1 instructors (N = 35) who acted as participants in our experiments. A larger sample is desired; however, finding and convincing such professionals to participate in experiments is notoriously challenging, since they already have a high work demand. Furthermore, to reduce the bias of dependence between individual observations, when a sample is taken from clusters from geographical areas, the participants of our experiment are from different universities, from different regions of Brazil. This leads to higher standard errors, which avoids spurious significant results. Future work could also consider replicating our study in other universities in different countries. Moreover, it is worth noting that we are interested in evaluating the equivalence of a pair of questions. In this sense, 210 pairs were evaluated, which is a reasonable number to draw statistical conclusions from (HOX *et al.*, 2010).

Another limitation is related to how we enhanced our behavioural recommender with the topic detector. Indeed, an open question in our study is how much our features contributed to measuring student effort, if we isolated the topic variable. That is, without topic detection, how useful our fine-grained features were. However, to address

this question, more items would be needed by instructors in our questionnaire, making it more challenging to find CS1 instructors available to participate in the experiment. That's why we, pragmatically, chose to compare how much our combined contributions (topic detection + effort based on fine-grained metrics) influence the quality of recommendations compared to our baseline.

In addition, we experimented and drew our conclusions considering data from the instructors' perspective in Chapter 6, which is essential and has not yet been done to the best of our knowledge. The other recommendation systems for OJs presented in the literature were tested with statistical metrics but were not evaluated by end-users. However, we assess with CS1 instructors in our method, but not with students. We envision performing such evaluation with students considering measures extracted from the IDE whilst students are solving problems recommended by our method, combined with the instructors, versus the instructors only. Another limitation is related to the number of variations tested in our experiments. Still in Chapter 6, we evaluated 7 variations of assignments for each topic. Thus, a number higher than 7 is out of the scope of this experiment.

Furthermore, in this study, we present and validate a method in which AI provides recommendations to the instructor, who in turn validates the recommendations and can improve them in a cyclical and mutual learning process between AI agents and humans. However, the human teaching process for AI has yet to be validated, which we will be exploring next.

Another limitation related to our validation with CS1 instructors in Chapter 7, we believe that the distrust of AI can be an obstacle to its adoption. However, XAI might make this process smoother since the instructor can understand why the model makes some decisions, giving more transparency to AI users. Moreover, to some extent, it's essential not to give overreliance on AI to keep control of which outcomes are effective and which ones need to be improved.

Another limitation might be the need for training the instructors to use a potential novel OJ that follows our architecture. Indeed, some of the concepts presented are not lightweight and require some background (e.g., in statistics) to be understood.

Thus, implementing these concepts should keep as a requirement to keep the software component easily understandable for the instructors. So that learning how to use it would not be one more work for the instructors.

Moreover, another threat to this work is related to the subjectivity of the storyboards we showed to the instructors in the validation step (Chapter 7). The instructors might interpret the storyboards in different ways. Nonetheless, during the speed dating section, we attempted to tackle this limitation by clarifying doubts and explaining to the participants the concepts.

8.3 Future Work

As future work, we envision to analyse how the data-driven approach used in this work can model students who begin the course with successful behaviours but end up with failure behaviours and grades. Similarly, we will analyse students who change their programming behaviour during the course and the impact of these changes on learning.

Moreover, we will investigate plagiarism behaviour in-depth and its influence in the model's decision. Moreover, we envision to analyse how the data-driven approach used in this work can model students who begin the course with successful behaviours, but end up with failure behaviours and grades.

Furthermore, a possible extension of the methods presented in Chapter 4 would be to not only create differential explainable models for different learners, but to also investigate whether different situations experienced by the same person have a different impact on the person's learning success, thereby applying a process *perspective on learning* (ZIMMERMAN, 2008). In order to do so, it would be necessary to collect time series data over a longer period, and to include information about the order of events into the prediction model.

Additionally, we are managing to adopt our clustering analysis of effective and ineffective behaviours, our interpretable machine learning model, and our recommender system to intervene in CS1 classes taken in UFAM. We will perform controlled experiments to analyse how those methods can influence in students' learning and

outcomes.

Besides that, using the method we presented in Chapter 6,we intend to create an inventory of equivalent questions, to be used by researchers who wish to employ pre- and post-tests in the computing education research field. Still, we envision to measure plagiarism practice with the use of this method.

Future work might also investigate ways to motivate instructors to teach the AIEd systems, such as gamification or monetary incentives. Moreover, it is important to see those concepts implemented in real scenarios and validated from the learners' perspective. In addition, we envision examining how each concept-design affects student learning individually, intending to prioritise how to assist instructors in employing these concepts.

Related to the dimension of action pointed out by (HOLSTEIN *et al.*, 2020), this study explored a few possibilities. Some instructors suggested that the system should provide a set of methodological/pedagogical strategies given the information provided in the concepts. According to the instructors, the design concepts presented do not address the dimension of action satisfactorily. A new study will target this aspect by providing options on how to intervene in future work.

Moreover, we designed our hybrid POJ architecture in a way that only works online. We envision studying ways of providing our methods in a novel architecture that allows offline description, prediction, and prescription - for fairness purposes. So that even places that do not have access to the internet (what is common in poor regions in Brazil), could access the benefits of our findings. This is important for the development of a political context aimed at distributional justice and equal opportunities (SMITH *et al.*, 2018). Still, the use of data-driven decision systems in domains apart from education has raised concerns about fairness, bias, and discrimination (KIZILCEC; LEE, 2020). Thus, it is crucial to conduct more studies about these topics in education.

Finally, we observed that our OJ hybrid architecture could be potentially adopted for a more generic AI in education hybrid architecture. We envision studying ways to perform such adaptation.

A

Appendix I

Following are the results (Figure 44) of our predictive model (Xgboost) for the remaining sessions (s1-s7). Notice that the model become more accurate throughout the course, as expected.

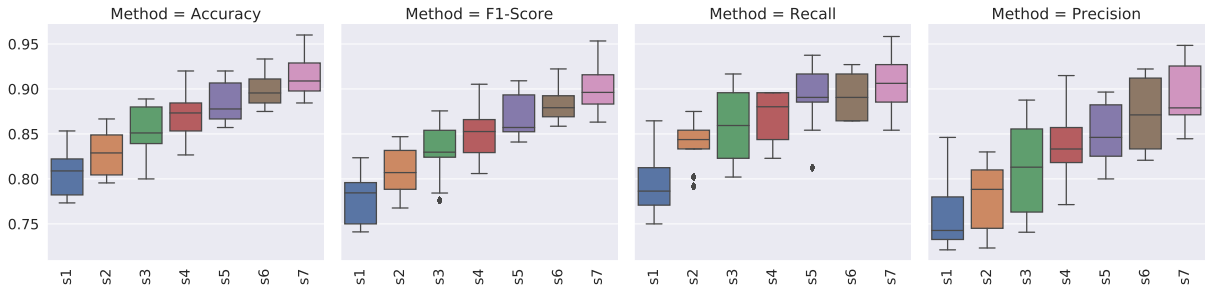

Figure 44 – Results of the XGboost model for all sessions.

B

Appendix II

Following are the results (Figure 44) of our descriptive method (clustering presented in Chapter 2) for the remaining sessions (s1-s7). Notice that the hidden patterns found become more explicit throughout the course.

It is worth noting that for the collected from the second session to the seventh session, the best number of clusters found was 2. Furthermore, the variables that did not have a moderate to high correlation with the evaluative variables started to have such correlation. In this way, the data of all the variables were kept in the radar plots presented in Figure 45.

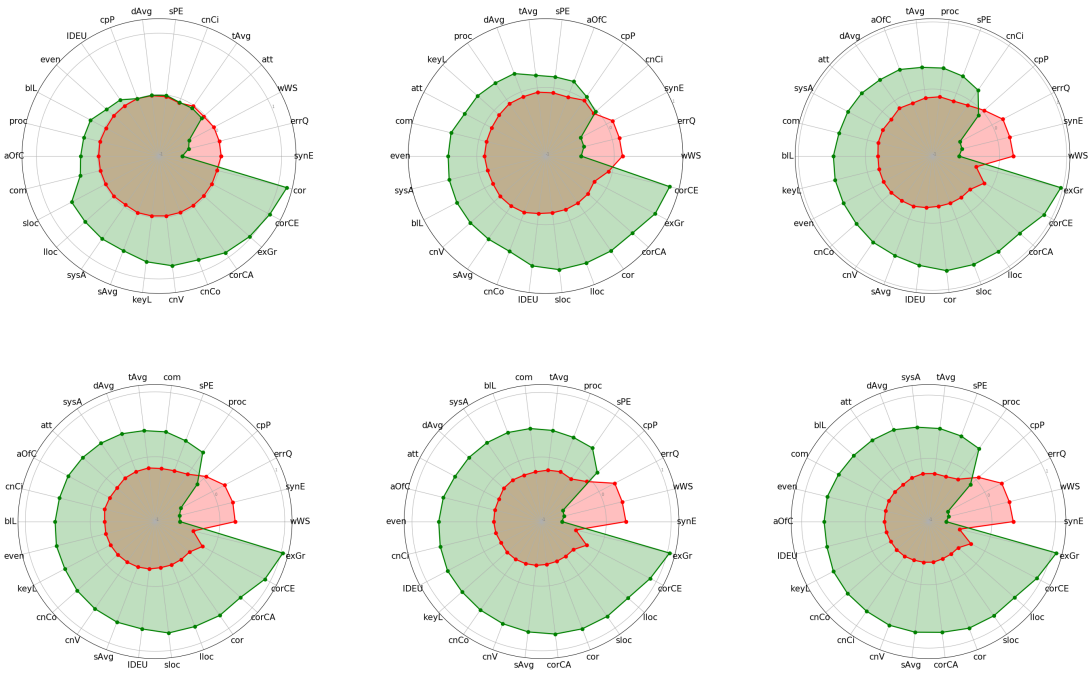

Figure 45 – Programming profile of students in each cluster for the remaining sessions (s2-s7). cpP stands for copyPaste, dAvg stands for deleteAvg, sPE stands for submissionsPerExercise, cnCI tands for countCicle, tAvg stands for testAvg, att stands for attempts, wWS stands for watWinScore, errQ stands for errorQuotient, synE stands for syntaxError, cor stands for correctness, corCE stands for eventActivity, exGr stands for examGrade, corCA stands for correctnessCodeAct, cnCo stands for countConditions, cnV stands for countVar, keyL stands for keystrokeLatency, sAvg stands for submissionAvg, sysA stands for systemAccess, lloc stands for logicalLinesOfCode, sloc stands for sourceLinesofCode, com stands for Comments, aOfC stands for amountOfChange, proc stands for procrastination, blL stands for blankLinesOfCode, even stands for numberOfEvents, IDEU stands for IDEUsage. The description of the variables is provided in Chapter 2 Table2, in Chapter 3 Section3.4.2, and in Chapter 5 Table 9.

C

APPENDIX III

C.0.1 Characterisation of Students' Programming Behaviours

For a general picture of the original values from the programming behaviours (that compose the programming profile) used as features by the ML models, we show distributions of these features in Figure 46, before any transformation performed. Overall, we can observe the lack of symmetry in most of the cases, by seeing that most features have high positive skewness ($skewness > 1.0$) (*procrastination*, *amountOfChange*, *comments*, *systemAccess*, *events*, *copyPaste*, *syntaxError*, *ideUsage*, *deleteAvg*, *errorQuotient*, *watWinScore*) with long tails ($kurtosis > 1.0$), which means that they tend to be concentrated in lower values of the distribution. Indeed, only the *eventActivity* has a high negative skewness ($skewness < -1.0$) , which means the values tend to be concentrated in the higher values. On the other hand, the other features (*attempts*, *lloc*, *firstExamGrade*, *correctness*, *correctnessCodeAct*, *keystrokeLatency*, *countVar*, *deleteAvg*, *finalGrade*) have low or moderate skewness. Moreover, we notice an overall high variation in the features, which indicates heterogeneity in the students' behaviours[1].

Following, we show the explanation and overall analysis results for each feature from our programming profile presented in Figure 46:

- *procrastination*: Here we are analysing the feature before the z-score transformation and multiplication by -1, thus, a higher value means lower procrastination (and vice-versa). In this feature, there is a high positive skewness (skewness > 1.0 and kurtosis > 1.0), indicating asymmetric distribution with a long tail. Indeed, some

[1] For more details about the distributions of the programming behaviours, see Appendix A

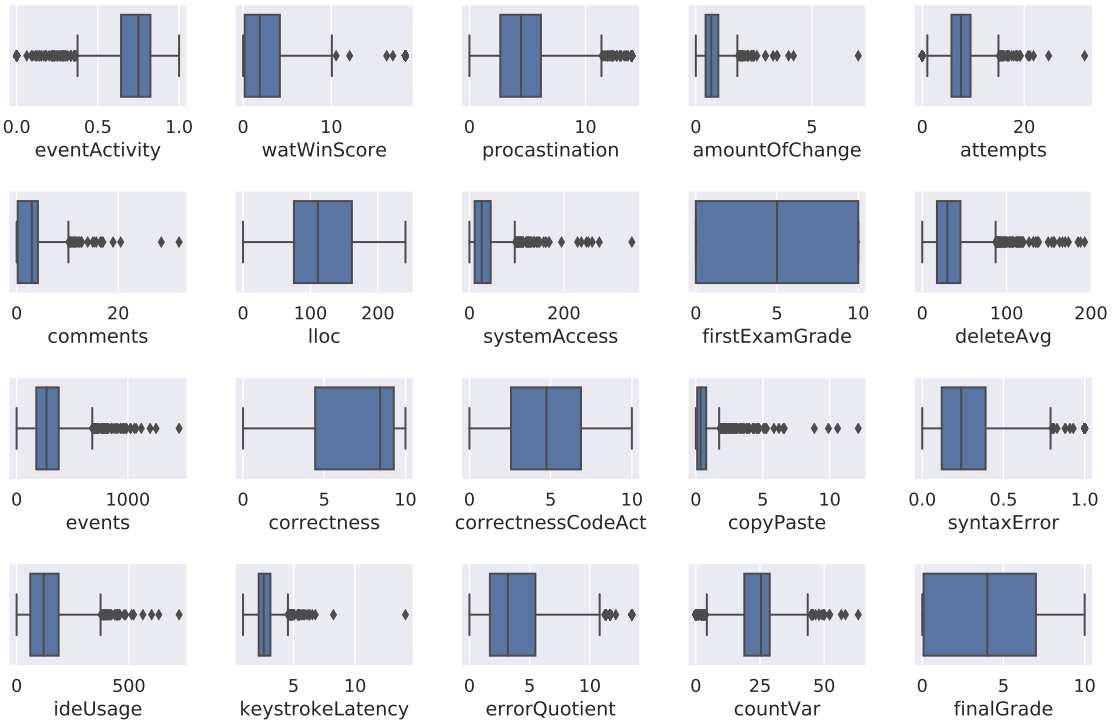

Figure 46 – Distributions of programming behaviours (features) and distribution of our dependent variable (final grade) before discretisation.

students solve the problems close to the deadline, however, most of the learners started to solve the problems around 5 days before the deadline (mean = 4.93, median = 4.45). Moreover, we can notice a high variation (std = 3.48, Coefficient of Variation (CV) = .71, and Inter Quartile Range (IQR) = 1.71) in this feature endorsing what we claimed about the heterogeneity of the students' behaviours.

- *amountOfChange*: High positive skewness and kurtosis (skewness > 1.0 and kurtosis > 1.0). This suggests that students tend to change their code slightly between submissions to the same problem (mean = .72, median = .67). This happens typically when students have not had their code accepted in the first submission. A high variation (std = .51, CV = .77, and IQR = 0.55) was observed.

- *eventActivity*: High negative skewness and positive kurtosis (skewness < -1.0 and kurtosis < -1.0). Most students (mean = .69, median = .75) solve the problems with few events (line of logs, see Figure 34). The variation (std = .23, CV = .31, and IQR = 0.18) is moderate to high.

- *attempts*: Symmetric distribution (skewness = 0.31), however with a high kurtosis

(kurtosis > 1.0), which can be explained by the presence of outliers: in this case, students who tried many times to solve a given problem. In average, students have attempts of 7.52 and a median of 7.62 per problem, with a moderate to high variation (std = 3.47, CV = .45, IQR = 3.74). The high average and variation in the first two weeks might be explained due to the students learning to manage the online judge system. To deal with the outliers, we applied a root square transformation, to make the distribution normal.

- *comments*: High positive skewness and kurtosis (skewness > 1.0 and kurtosis > 1.0), suggesting that, in general, the students do not document their code (mean = 2.86, median = 3.00), which is expected from novice programmers solving easy problems. Nonetheless, we observe a high variation (std = 2.86, CV = .99, IQR = 4.00) and the presence of outliers. As for attempts, here we also applied the root squared transformation.

- *lloc*: Symmetric distribution (skewness = -0.28, kurtosis = -0.82) with a moderate kurtosis, with a mean similar to the median, indicating a bell-shaped distribution. The average of total *lloc* (mean = 111.71, median = 110.10) is low, as the learners are submitting solutions for problems of arithmetic operations and sequential structures, which require just a few lines of code. Moderate to high variation (mean = 111.71, std = 58.98, CV = .52, IQR = 86.0) is observed.

- *systemAccess*: Most of the students have the number of access to the system in the first 2 weeks of the course concentrated in the lowest values, as the distribution is highly positively skewed (skewness > 1.0 and kurtosis > 1.0), with an average of 32.89 access and high variation (std = 30.98, CV = .94, IQR = 87.00).

- *firstExamGrade*: As the first exams had only 2 problems, students can achieve 0, 5 or 10, if they solve 0, 1 or 2 questions, respectively. That explains the multimodal nature of this distribution, with three potential values of 0, 5 and 10. A different grade is possible when students solve one of the problems partially, receiving a grade proportional to the number of test cases accepted. In average, students solve one problem from the first exam (mean = 5.01, median = 5.00). Notice that

the nature of this distribution explains the high variation (std = 4.64, CV = .92, IQR = 10.00).

- *events*: Most students have a lower value of events as the distribution is high positively skewed (skewness > 1.0 and kurtosis > 1.0), with an average of 290.63 events and high variation (std = 180.96, CV = .62, IQR = 201.86).

- *correctness*: Moderate negative skewed distribution (skewness = -0.94 and kurtosis = -0.39), which indicates that most students take the first assignment list seriously and solve the problems. In average, students solved approximately 69% of the problems (mean = 6.91, median = 8.44) in an assignment list comprising 10 or 12 problems. We also observe a moderate variation (std = 3.21, CV = .46, IQR = 4.84).

- *correctnessCodeAct*: Most of students have an average value (mean = 4.68, median = 4.75) of correctnessCodeAct, as the distribution is symmetrical (skewness = -0.14 and kurtosis = -0.95). However, a high variation was observed (std = 2.77, CV = 0.59, IQR = 4.31). Notice that the values of these distributions tend to be lower than for the correctness distribution, which means that, potentially, some students solved the problems just by copying and pasting, thus, generating only few events.

- *copyPaste*: Highly skewed with a long tail (skewness > 1.0 and kurtosis > 1.0), with an average value of .59 and high variation (std = 0.90, CV = 1.52, IQR = 0.67). As in attempts, here we also applied the root squared transformation due to the presence of outliers (values greater than 1, in this case). Notice that a value greater than 1 means that the learner has pasted more characters than typed (e.g., 50 characters pasted and 10 characters types would lead for a *copyPaste* = 5 (50/10)).

- *syntaxError*: Highly skewed, with a long tail (skewness > 1.0 and kurtosis > 1.0). In average, 29% of the attempts (mean = 0.29, median = 0.24) to solve problems in the first two weeks have this typical error. A high variation was observed (std = .25, CV = .84, IQR = .27).

- *ideUsage*: Highly skewed, with a long tail (skewness > 1.0 and kurtosis > 1.0). In average, students spend 133.93 minutes trying to solve problems in the embedded

IDE (mean = 133.93, median = 120.52). A high variation was observed (std = 98.08, CV = .73, IQR = 126.21).

- *keystrokeLatency*: Highly skewed, with a long tail (skewness > 1.0 and kurtosis > 1.0). The keystroke average latency of the learners is 2.59 (mean = 2.59, median = 2.61) and a moderate to high variance was observed (std = 1.04, CV = .73, IQR = .97). As in attempts, here we also applied the root squared transformation, due to the outliers.

- *errorQuotient*: Highly skewed distribution (skewness > 1.0), but with no long tail (kurtosis = .04). We found a low value of *errorQuotient* penalty in pair of errors between submission (mean = 4.19, median = 3.19). A high variation was observed (std = 3.54, CV = .84, IQR = 3.79).

- *watWinScore*: Highly skewed, with a long tail (skewness > 1.0 and kurtosis > 1.0). Students spent a few minutes (mean = 3.34, median = 1.90) between a pair of submissions with errors. A high variation (std = 4.35, CV = 1.30, IQR = 3.99) was observed, due to the presence of some outliers. As in attempts, here we also applied the root squared transformation.

- *countVar*: A moderate to high negative skewed distribution (skewness = -0.72), but with no long tail (kurtosis = .39), with an average of 22.38 (mean = 22.38, median = 3.19) variables in all the code instances submitted by learners. This relatively lower number of variables is due to the easy nature of the initial problem assignments. In addition, a moderate variation was observed (std = 10.56, CV = .47, IQR = 9.89).

- *deleteAvg*: Highly skewed, with a long tail (skewness > 1.0 and kurtosis > 1.0). In average, students make little use of delete (mean = 35.13, median = 29.72). A high variation was observed (std = 26.93, CV = .76, IQR = 27.88), due to the presence of some outliers. As in attempts, here we also applied the root squared transformation. Notice that learners who make more use of delete are potentially rewriting their code more frequently.

- *finalGrade*: Our target variable is a relatively symmetrical (skewness = .2) bimodal distribution (kurtosis > 1.0). Indeed, the left peak of the distribution concentrates

most of the failed students and the right peak, the ones that passed. Students achieved an average final grade of 3.93 (mean = 3.93, media = 4.00). A high variation was observed (std = 3.45, CV = .96, IQR = 6.74).

Bibliography

Abu Amra, I. A.; Maghari, A. Y. A. Students performance prediction using knn and naïve bayesian. In: *2017 8th International Conference on Information Technology (ICIT)*. [S.l.: s.n.], 2017. p. 909–913. 29, 75, 78

AGRAWAL, A.; GANS, J.; GOLDFARB, A. *Prediction machines: the simple economics of artificial intelligence*. [S.l.]: Harvard Business Press, 2018. 29, 31, 194, 197

AGRAWAL, R.; IMIELIńSKI, T.; SWAMI, A. Mining association rules between sets of items in large databases. In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 1993. (SIGMOD '93), p. 207–216. ISBN 0897915925. Disponível em: <https://doi.org/10.1145/170035.170072>. 63

AGUIAR, G. C. L.; PEREIRA, F. D. *Uma Abordagem Data-Driven Para Predição Precoce Da Evasão Em Turmas De Programação Que Utilizam Juízes Online*. Dissertação (Graduação em Ciência da Computação, Universidade Federal de Roraima, Boa Vista) — Universidade Federal de Roraima, Boa Vista, 2018. 75, 78

AHADI, A.; LISTER, R.; HAAPALA, H.; VIHAVAINEN, A. Exploring machine learning methods to automatically identify students in need of assistance. In: *Proceedings of the eleventh annual International Conference on International Computing Education Research*. [S.l.: s.n.], 2015. p. 121–130. 29, 81

AHADI, A.; LISTER, R.; VIHAVAINEN, A. On the number of attempts students made on some online programming exercises during semester and their subsequent performance on final exam questions. In: ACM. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. [S.l.], 2016. p. 218–223. 29, 51, 52, 61, 76, 78, 81, 143

AKATA, Z.; BALLIET, D.; RIJKE, M. D.; DIGNUM, F.; DIGNUM, V.; EIBEN, G.; FOKKENS, A.; GROSSI, D.; HINDRIKS, K.; HOOS, H. *et al.* A research agenda for hybrid intelligence: augmenting human intellect with collaborative, adaptive, responsible, and explainable artificial intelligence. *Computer*, IEEE Computer Society, v. 53, n. 08, p. 18–28, 2020. 22

ALAMRI, A.; ALSHEHRI, M.; CRISTEA, A.; PEREIRA, F. D.; OLIVEIRA, E.; SHI, L.; STEWART, C. Predicting moocs dropout using only two easily obtainable features from the first week's activities. In: SPRINGER. *International Conference on Intelligent Tutoring Systems*. [S.l.], 2019. p. 163–173. 75

ALBLUWI, I. Plagiarism in programming assessments: a systematic review. *ACM Transactions on Computing Education (TOCE)*, ACM New York, NY, USA, v. 20, n. 1, p. 1–28, 2019. 164

ALEVEN, V.; MCLAUGHLIN, E. A.; GLENN, R. A.; KOEDINGER, K. R. Instruction based on adaptive learning technologies. *Handbook of research on learning and instruction*, Routledge New York, NY, v. 2, p. 522–560, 2016. 31, 32, 198

ALJOHANI, T.; PEREIRA, F. D.; CRISTEA, A. I.; OLIVEIRA, E. Prediction of users' professional profile in moocs only by utilising learners' written texts. In: SPRINGER. *International Conference on Intelligent Tutoring Systems*. [S.l.], 2020. p. 163–173. 76

ALLEN, J. M.; VAHID, F. Concise graphical representations of student effort on weekly many small programs. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. [S.l.: s.n.], 2021. p. 349–354. 26

ALLEN, J. M.; VAHID, F.; DOWNEY, K.; EDGCOMB, A. D. Weekly programs in a cs1 class: Experiences with auto-graded many-small programs (msp). In: *2018 ASEE Annual Conference & Exposition*. [S.l.: s.n.], 2018. 26

ALTMANN, A.; TOLOŞI, L.; SANDER, O.; LENGAUER, T. Permutation importance: a corrected feature importance measure. *Bioinformatics*, Oxford University Press, v. 26, n. 10, p. 1340–1347, 2010. 109

ANDERSEN, P.-A.; KRÅKEVIK, C.; GOODWIN, M.; YAZIDI, A. Adaptive task assignment in online learning environments. In: ACM. *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*. [S.l.], 2016. p. 5. 20, 21, 27

ANSARI, M. H.; MORADI, M.; NIKRAH, O.; KAMBAKHSH, K. M. Coders: a hybrid recommender system for an e-learning system. In: IEEE. *2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS)*. [S.l.], 2016. p. 1–5. 142

ANTONENKO, P. D.; TOY, S.; NIEDERHAUSER, D. S. Using cluster analysis for data mining in educational technology research. *Educational Technology Research and Development*, Springer, v. 60, n. 3, p. 383–398, 2012. 62

ARAUJO, A.; FILHO, D. L. Z.; OLIVEIRA, E. H. T. de; CARVALHO, L. S. G. de; PEREIRA, F. D.; OLIVEIRA, D. B. F. de. Mapeamento e análise empírica de misconceptions comuns em avaliações de introdução à programação. In: SBC. *Anais do Simpósio Brasileiro de Educação em Computação*. [S.l.], 2021. p. 123–131. 20

ARMSTRONG, R. A. When to use the bonferroni correction. *Ophthalmic and Physiological Optics*, Wiley Online Library, v. 34, n. 5, p. 502–508, 2014. 183

ARTSTEIN, R.; POESIO, M. *Inter-coder agreement for computational linguistics*. [S.l.]: Educational and Psychological Measurement 20(1):37-46, 2008. 151

ATHAVALE, V.; NAIK, A.; VANJAPE, R.; SHRIVASTAVA, M. Predicting algorithm classes for programming word problems. In: *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*. [S.l.: s.n.], 2019. p. 84–93. 167

AUVINEN, T. Harmful study habits in online learning environments with automatic assessment. In: IEEE. *2015 International Conference on Learning and Teaching in Computing and Engineering*. [S.l.], 2015. p. 50–57. 52, 80, 81

BACH, S.; BINDER, A.; MONTAVON, G.; KLAUSCHEN, F.; MÜLLER, K.-R.; SAMEK, W. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, Public Library of Science, v. 10, n. 7, p. e0130140, 2015. 111

BAKER, R. S.; INVENTADO, P. S. Educational data mining and learning analytics. In: *Learning analytics*. [S.l.]: Springer, 2014. p. 61–75. 109

BANNON, L.; EHN, P. Design Matters in Participatory Design. In: SIMONSEN, J.; ROBERTSON, T. (Ed.). *Routledge International Handbook of Participatory Design*. Routledge, 2012. p. 37–36. Disponível em: <https://www.routledge.com/Routledge-International-Handbook-of-Participatory-Design/Simonsen-Robertson/p/book/9780415720212>. 200

BATISTA, G. E.; PRATI, R. C.; MONARD, M. C. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, ACM New York, NY, USA, v. 6, n. 1, p. 20–29, 2004. 84, 85

BELLHäUSER, H.; LöSCH, T.; WINTER, C.; SCHMITZ, B. Applying a web-based training to foster self-regulated learning — Effects of an intervention for large numbers of participants. *The Internet and Higher Education*, v. 31, p. 87–100, 2016. 132, 133

BERENDT, B.; LITTLEJOHN, A.; BLAKEMORE, M. Ai in education: learner choice and fundamental rights. *Learning, Media and Technology*, Taylor & Francis, v. 45, n. 3, p. 312–324, 2020. 73, 108, 109

BEZ, J. L.; TONIN, N. A.; RODEGHERI, P. R. Uri online judge academic: A tool for algorithms and programming classes. In: IEEE. *2014 9th International Conference on Computer Science & Education*. [S.l.], 2014. p. 149–152. 21, 26, 167

BIGGS, J. B.; COLLIS, K. F. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. [S.l.]: Academic Press, 2014. 102

BILEGJARGAL, D.; HSUEH, N.-L. Understanding students' acceptance of online judge system in programming courses: A structural equation modeling approach. *IEEE Access*, IEEE, v. 9, p. 152606–152615, 2021. 21, 26, 27

BLIKSTEIN, P. Using learning analytics to assess students behavior in open-ended programming tasks. *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, p. 110–116, 2011. ISSN 9781450310574. 102

BLIKSTEIN, P.; WORSLEY, M.; PIECH, C.; SAHAMI, M.; COOPER, S.; KOLLER, D. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, Taylor & Francis, v. 23, n. 4, p. 561–599, 2014. 29

BOCKMON, R.; COOPER, S.; GRATCH, J.; ZHANG, J.; DORODCHI, M. Can students' spatial skills predict their programming abilities? In: . New York, NY, USA: Association for Computing Machinery, 2020. (ITiCSE '20), p. 446–451. ISBN 9781450368742. Disponível em: <https://doi.org/10.1145/3341525.3387380>. 29, 63

BOULANGER, D.; KUMAR, V. Shaped automated essay scoring: Explaining writing features' contributions to english writing organization. In: SPRINGER. *International Conference on Intelligent Tutoring Systems*. [S.l.], 2020. p. 68–78. 111, 115

BRAZ, A. C. da R.; CARVALHO, L. S. G. de; OLIVEIRA, E. H. T. de; OLIVEIRA, D. B. F. de; PEREIRA, F. D.; BITTENCOURT, R. A.; SANTANA, B. L. Validação e análise de um inventário de conceitos sobre programação introdutória. In: SBC. *Anais Estendidos do I Simpósio Brasileiro de Educação em Computação*. [S.l.], 2021. p. 27–28. 20

BRAZ, A. C. R.; CARVALHO, L. S.; OLIVEIRA, E. H.; OLIVEIRA, D. B.; BITTENCOURT, R. A.; SANTANA, B. L.; PEREIRA, F. D. Tradução e validação de um inventário de conceitos sobre programação introdutória. In: SBC. *Anais do XXXII Simpósio Brasileiro de Informática na Educação*. [S.l.], 2021. p. 1253–1264. 20

BROWN, E.; CRISTEA, A. I.; STEWART, C.; BRAILSFORD, T. Patterns in authoring of adaptive educational hypermedia: A taxonomy of learning styles. *Journal of Educational Technology & Society*, JSTOR, v. 8, n. 3, p. 77–90, 2005. 132

CARTER, A.; HUNDHAUSEN, C.; OLIVARES, D. Leveraging the integrated development environment for learning analytics. In: *The Cambridge Handbook of Computing Education Research*. Cambridge: Cambridge University Press, 2019. cap. 23, p. 679–706. 21, 22, 27, 28, 29, 32, 51, 52, 57, 58, 73, 75, 76, 78, 101, 102, 109, 110, 131, 133, 143, 170

CARVALHO, L.; FERNANDES, D.; GADELHA, B. Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2016. v. 27, n. 1, p. 140. 20, 21, 26, 48, 102

CASTRO-WUNSCH, K.; AHADI, A.; PETERSEN, A. Evaluating neural networks as a method for identifying students in need of assistance. In: ACM. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. [S.l.], 2017. p. 111–116. 29, 51, 52, 58, 63, 76, 78, 80, 81, 93, 109, 143

CHAU, H.; BARRIA-PINEDA, J.; BRUSILOVSKY, P. Content wizard: concept-based recommender system for instructors of programming courses. In: ACM. *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*. [S.l.], 2017. p. 135–140. 139, 142, 143

CHEN, C.-M. Personalized e-learning system with self-regulated learning assisted mechanisms for promoting learning performance. *Expert Systems with Applications*, Elsevier, v. 36, n. 5, p. 8816–8829, 2009. 143

CHEN, N.-C.; SUH, J.; VERWEY, J.; RAMOS, G.; DRUCKER, S.; SIMARD, P. Anchorviz: Facilitating classifier error discovery through interactive semantic data exploration. In: *23rd International Conference on Intelligent User Interfaces*. [S.l.: s.n.], 2018. p. 269–280. 22, 225

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. [S.l.: s.n.], 2016. p. 785–794. 114

CHOLLET, F. *et al. Deep learning with Python*. [S.l.]: Manning New York, 2018. v. 361. 111

CHOY, M.; NAZIR, U.; POON, C. K.; YU, Y.-T. Experiences in using an automated system for improving students' learning of computer programming. In: SPRINGER. *International Conference on Web-Based Learning*. [S.l.], 2005. p. 267–272. 26

COHEN, J. A coefficient of agreement for nominal scales. In: . [S.l.]: Educational and Psychological Measurement 20(1):37-46, 1960. 151

COHEN, J. *Statistical power analysis for the behavioral sciences*. [S.l.]: Academic press, 2013. 12, 66, 92, 93

COSTA, E. B.; FONSECA, B.; SANTANA, M. A.; ARAÚJO, F. F. de; REGO, J. Evaluating the effectiveness of educational data mining techniques for early prediction of students' academic failure in introductory programming courses. *Computers in Human Behavior*, Elsevier, v. 73, p. 247–256, 2017. 29, 63, 75, 76, 78, 93

COSTA, T. L.; OLIVEIRA, E. H. T. de; PASSITO, A.; PINTO, M. A. de S.; CARVALHO, L. S. G. de; OLIVEIRA, D. B. F. de; PEREIRA, F. D. Material didático interativo para a disciplina de introdução à programação de computadores. In: SBC. *Anais Estendidos do I Simpósio Brasileiro de Educação em Computação*. [S.l.], 2021. p. 41–42. 20, 168

CRIŞAN, A. N. Case study on the importance of formative assessment in stimulating student motivation for learning and increasing the efficiency of the educational process. *Journal of Educational Sciences & Psychology*, v. 7, n. 1, 2017. 133

CSIKSZENTMIHALYI, M.; CSIKSZENTMIHALYI, I. S. *Optimal experience: Psychological studies of flow in consciousness*. [S.l.]: Cambridge university press, 1992. 139

CUI, C.; HU, M.; WEIR, J. D.; WU, T. A recommendation system for meta-modeling: A meta-learning based approach. *Expert Systems with Applications*, Elsevier, v. 46, p. 33–44, 2016. 139

DANNER, D.; AICHHOLZER, J.; RAMMSTEDT, B. Acquiescence in personality questionnaires: Relevance, domain specificity, and stability. *Journal of Research in Personality*, v. 57, p. 119–130, 2015. ISSN 0092-6566. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0092656615000495>. 182

DAVIDOFF, S.; LEE, M. K.; DEY, A. K.; ZIMMERMAN, J. Rapidly exploring application design through speed dating. In: SPRINGER. *International Conference on Ubiquitous Computing*. [S.l.], 2007. p. 429–446. 23, 35, 199, 200, 206

DE-ARTEAGA, M.; FOGLIATO, R.; CHOULDECHOVA, A. A case for humans-in-the-loop: Decisions in the presence of erroneous algorithmic scores. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. [S.l.: s.n.], 2020. p. 1–12. 22, 31, 198, 225

DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, IEEE, v. 6, n. 2, p. 182–197, 2002. 87

DECI, E. L.; RYAN, R. M. Intrinsic motivation. *The corsini encyclopedia of psychology*, Wiley Online Library, p. 1–2, 2010. 139, 141

DELLERMANN, D.; CALMA, A.; LIPUSCH, N.; WEBER, T.; WEIGEL, S.; EBEL, P. The future of human-ai collaboration: a taxonomy of design knowledge for hybrid intelligence systems. *arXiv preprint arXiv:2105.03354*, 2021. 188

DELLERMANN, D.; EBEL, P.; SÖLLNER, M.; LEIMEISTER, J. M. Hybrid intelligence. *Business & Information Systems Engineering*, Springer, v. 61, n. 5, p. 637–643, 2019. 22, 29, 31, 188, 189, 194, 197, 198

DICHEV, C.; DICHEVA, D.; ANGELOVA, G.; AGRE, G. From gamification to gameful design and gameful experience in learning. *Cybernetics and Information Technologies*, De Gruyter Open, v. 14, n. 4, p. 80–100, 2014. 141

D'MELLO, S.; CALVO, R. A. Beyond the basic emotions: what should affective computing compute? In: *CHI'13 extended abstracts on human factors in computing systems*. [S.l.: s.n.], 2013. p. 2287–2294. 148

D'MELLO, S. K.; LEHMAN, B.; PERSON, N. Monitoring affect states during effortful problem solving activities. *International Journal of Artificial Intelligence in Education*, IOS Press, v. 20, n. 4, p. 361–389, 2010. 141, 149

DORODCHI, M.; DEHBOZORGI, N.; FREVERT, T. K. "i wish i could rank my exam's challenge level!": An algorithm of bloom's taxonomy in teaching cs1. In: IEEE. *Frontiers in Education Conference (FIE'2017)*. [S.l.], 2017. p. 1–5. 165

DUCKWORTH, A. L.; EICHSTAEDT, J. C.; UNGAR, L. H. The mechanics of human achievement. *Social and Personality Psychology Compass*, Wiley Online Library, v. 9, n. 7, p. 359–369, 2015. 139, 140, 142, 146

DUTT, A.; AGHABOZRGI, S.; ISMAIL, M. A. B.; MAHROEIAN, H. Clustering algorithms applied in educational data mining. *International Journal of Information and Electronics Engineering*, IACSIT Press, v. 5, n. 2, p. 112, 2015. 62

DWAN, F.; OLIVEIRA, E.; FERNANDES, D. Predição de zona de aprendizagem de alunos de introdução à programação em ambientes de correção automática de código. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2017. v. 28, n. 1, p. 1507. 51, 75, 109, 142, 143, 145, 146, 170

EDWARDS, J.; LEINONEN, J.; HELLAS, A. A study of keystroke data in two contexts: Written language and programming language influence predictability of learning outcomes. In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. [S.l.: s.n.], 2020. p. 413–419. 29, 93, 116

EDWARDS, S.; LI, Z. Towards progress indicators for measuring student programming effort during solution development. In: *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. [S.l.: s.n.], 2016. p. 31–40. 140

EDWARDS, S. H.; SNYDER, J.; PÉREZ-QUIÑONES, M. A.; ALLEVATO, A.; KIM, D.; TRETOLA, B. Comparing effective and ineffective behaviors of student programmers. In: ACM. *Proceedings of the fifth international workshop on Computing education research workshop*. [S.l.], 2009. p. 3–14. 51, 52, 58, 76, 101

EKMAN, P. An argument for basic emotions. *Cognition & emotion*, Taylor & Francis, v. 6, n. 3-4, p. 169–200, 1992. 149

ELARDE, J. Toward improving introductory programming student course success rates: Experiences with a modified cohort model to student success sessions. *J. Comput. Sci. Coll.*, Consortium for Computing Sciences in Colleges, Evansville, IN, USA, v. 32, n. 2, p. 113–119, dez. 2016. ISSN 1937-4771. 76

ESTEY, A.; COADY, Y. Can interaction patterns with supplemental study tools predict outcomes in CS1? *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '16*, p. 236–241, 2016. 29, 52, 58, 63, 76, 78, 80, 81, 143

FANTOZZI, P.; LAURA, L. Collaborative recommendations in online judges using autoencoder neural networks. In: SPRINGER. *International Symposium on Distributed Computing and Artificial Intelligence*. [S.l.], 2020. p. 113–123. 138, 142, 143

FANTOZZI, P.; LAURA, L. Recommending tasks in online judges using autoencoder neural networks. *Olympiads in Informatics*, v. 14, p. 61–76, 2020. 168, 169, 170

FENTON, N.; BIEMAN, J. *Software metrics: a rigorous and practical approach*. [S.l.]: CRC press, 2014. 146

FEURER, M.; KLEIN, A.; EGGENSPERGER, K.; SPRINGENBERG, J.; BLUM, M.; HUTTER, F. Efficient and robust automated machine learning. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2015. p. 2962–2970. 86

FILHO, D. L. Z.; OLIVEIRA, E. H. T. de; CARVALHO, L. S. G. de; PESSOA, M.; PEREIRA, F. D.; OLIVEIRA, D. B. F. de. Uma análise orientada a dados para avaliar o impacto da gamificação de um juiz on-line no desempenho de estudantes. In: SBC. *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. [S.l.], 2020. p. 491–500. 29, 142

FONSECA, S.; OLIVEIRA, E.; PEREIRA, F.; FERNANDES, D.; CARVALHO, L. S. G. de. Adaptação de um método preditivo para inferir o desempenho de alunos de programação. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2019. v. 30, n. 1, p. 1651. 29, 52, 75, 76, 80

FONSECA, S. C.; PEREIRA, F. D.; OLIVEIRA, E. H.; OLIVEIRA, D. B.; CARVALHO, L. S.; CRISTEA, A. I. Automatic subject-based contextualisation of programming assignment lists. In: . [S.l.]: EDM, 2020. 167, 168

FORD, D.; PARNIN, C. Exploring causes of frustration for software developers. In: IEEE. *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*. [S.l.], 2015. p. 115–116. 139

FOWLER, M.; ZILLES, C. Superficial code-guise: Investigating the impact of surface feature changes on students' programming question scores. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. [S.l.: s.n.], 2021. p. 3–9. 27, 164

FWA, H. L. An architectural design and evaluation of an affective tutoring system for novice programmers. *International Journal of Educational Technology in Higher Education*, Springer, v. 15, n. 1, p. 38, 2018. 140

FWA, H. L. Predicting non-completion of programming exercises using action logs and keystrokes. In: IEEE. *2019 International Symposium on Educational Technology (ISET)*. [S.l.], 2019. p. 271–275. 109

GELINAS, L.; PIERCE, R.; WINKLER, S.; COHEN, I. G.; LYNCH, H. F.; BIERER, B. E. Using social media as a research recruitment tool: ethical issues and recommendations. *The American Journal of Bioethics*, Taylor & Francis, v. 17, n. 3, p. 3–14, 2017. 181

GELMAN, A.; HILL, J. *Data analysis using regression and multilevel/hierarchical models*. [S.l.]: Cambridge university press, 2006. 182

GÉRON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. [S.l.]: O'Reilly Media, 2019. 74, 76, 81, 84, 85, 87, 88, 89, 107, 111, 117, 118, 124, 167, 176

GERRITSEN, D.; ZIMMERMAN, J.; OGAN, A. Towards a framework for smart classrooms that teach instructors to teach. In: *International Conference of the Learning Sciences*. [S.l.: s.n.], 2018. v. 3. 30

GOLDSTEIN, E. B. *Cognitive psychology: Connecting mind, research and everyday experience*. [S.l.]: Nelson Education, 2014. 141

GUASTELLO, S. J.; KOOPMANS, M.; PINCUS, D. *Chaos and complexity in psychology: The theory of nonlinear dynamical systems*. [S.l.]: Cambridge University Press, 2008. 131

HADEN, P. Descriptive statistics. *The Cambridge Handbook of Computing Education Research, Sally A. Fincher and Anthony V. Robins (Eds.). Cambridge University Press, Cambridge*, p. 102–131, 2019. 144

HELLAS, A.; IHANTOLA, P.; PETERSEN, A.; AJANOVSKI, V. V.; GUTICA, M.; HYNNINEN, T.; KNUTAS, A.; LEINONEN, J.; MESSOM, C.; LIAO, S. N. Predicting academic performance: A systematic literature review. In: *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2018. (ITiCSE 2018 Companion), p. 175–199. ISBN 9781450362238. Disponível em: <https://doi.org/10.1145/3293881.3295783>. 73, 75, 77, 78, 82, 88

HERNÁNDEZ-BLANCO, A.; HERRERA-FLORES, B.; TOMÁS, D.; NAVARRO-COLORADO, B. A systematic review of deep learning approaches to educational data mining. *Complexity*, Hindawi, v. 2019, 2019. 76, 81, 87

HERODOTOU, C.; HLOSTA, M.; BOROOWA, A.; RIENTIES, B.; ZDRAHAL, Z.; MANGAFA, C. Empowering online teachers through predictive learning analytics. *British Journal of Educational Technology*, Wiley Online Library, v. 50, n. 6, p. 3064–3079, 2019. 51

HINKLE, D. E.; WIERSMA, W.; JURS, S. G. *Applied statistics for the behavioral sciences*. [S.l.]: Houghton Mifflin College Division, 2003. v. 663. 57

HINTON, G. E.; SRIVASTAVA, N.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 76, 88

HOLSTEIN, K.; ALEVEN, V.; RUMMEL, N. A conceptual framework for human–ai hybrid adaptivity in education. In: SPRINGER. *International Conference on Artificial Intelligence in Education*. [S.l.], 2020. p. 240–254. 22, 28, 29, 30, 31, 32, 34, 35, 170, 188, 189, 194, 197, 198, 199, 225, 236

HOLSTEIN, K.; MCLAREN, B. M.; ALEVEN, V. Designing for complementarity: Teacher and student needs for orchestration support in ai-enhanced classrooms. In: SPRINGER. *International Conference on Artificial Intelligence in Education*. [S.l.], 2019. p. 157–171. 27, 29, 31, 35, 198

HOSSEINI, R.; BRUSILOVSKY, P. A study of concept-based similarity approaches for recommending program examples. *New Review of Hypermedia and Multimedia*, Taylor & Francis, v. 23, n. 3, p. 161–188, 2017. 143

HOSSEINI, R.; VIHAVAINEN, A.; BRUSILOVSKY, P. Exploring problem solving paths in a java programming course. In: *Psychology of Programming Interest Group Conference, PPIG 2014*. [s.n.], 2014. p. 65 – 76. Disponível em: <http://d-scholarship.pitt.edu/21832/>. 51

HOX, J. J.; MOERBEEK, M.; SCHOOT, R. Van de. *Multilevel analysis: Techniques and applications*. [S.l.]: Routledge, 2010. 182, 183, 233

HUANG, H.; TORNERO-VELEZ, R.; BARZYK, T. M. Associations between socio-demographic characteristics and chemical concentrations contributing to cumulative exposures in the united states. *Journal of exposure science & environmental epidemiology*, Nature Publishing Group, v. 27, n. 6, p. 544–550, 2017. 63

HUTTER, F.; LÜCKE, J.; SCHMIDT-THIEME, L. Beyond manual tuning of hyperparameters. *KI-Künstliche Intelligenz*, Springer, v. 29, n. 4, p. 329–337, 2015. 86

IHANTOLA, P.; VIHAVAINEN, A.; AHADI, A.; BUTLER, M.; BöRSTLER, J.; EDWARDS, S. H.; ISOHANNI, E.; KORHONEN, A.; PETERSEN, A.; RIVERS, K.; RUBIO, M. ; SHEARD, J.; SKUPAS, B.; SPACCO, J.; SZABO, C.; TOLL, D. Educational data mining and learning analytics in programming: Literature review and case studies. *ACM. Proceedings of the 2015 ITiCSE on Working Group Reports*, p. 41–63, 2015. 20, 21, 22, 27, 28, 29, 32, 48, 68, 73, 75, 78, 82, 102, 103, 164, 168

IM, T.; KANG, M. Structural relationships of factors which impact on learner achievement in online learning environment. *International Review of Research in Open and Distributed Learning*, Athabasca University, v. 20, n. 1, 2019. 140

INTISAR, C. M.; WATANOBE, Y.; POUDEL, M.; BHALLA, S. Classification of programming problems based on topic modeling. In: *Proceedings of the 2019 7th International Conference on Information and Education Technology*. [S.l.: s.n.], 2019. p. 275–283. 163, 165, 166, 167

JADUD, M. C. Methods and tools for exploring novice compilation behaviour. *Proceedings of the second international workshop on Computing education research*, p. 73–84, 2006. 51, 52, 58, 76, 78, 79, 81, 93, 101, 143, 170

JÚNIOR, H. B. de F.; PEREIRA, F. D. Recomendação de problemas em juízes onlines utilizando técnicas de processamento de linguagem natural e análise dirigida aos dados. In: SBC. *Anais dos Workshops do IX Congresso Brasileiro de Informática na Educação*. [S.l.], 2020. p. 94–94. 102, 139

JÚNIOR, H. B. de F.; PEREIRA, F. D.; OLIVEIRA, E. H. T. de; OLIVEIRA, D. B. F. de; CARVALHO, L. S. G. de. Recomendação automática de problemas em juízes online usando processamento de linguagem natural e análise dirigida aos dados. In: SBC. *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. [S.l.], 2020. p. 1152–1161. 139

JÚNIOR, H. B. de F.; PEREIRA, F. D.; PEREIRA, A. L. da S.; SILVA, L. F. Reconhecimento de emoções em imagens utilizando técnicas de construção e otimização em métodos

ensembles baseados em árvores de decisão. *RCT-Revista de Ciência e Tecnologia*, v. 5, n. 8, 2019. 149

KARVELAS, I.; LI, A.; BECKER, B. A. The effects of compilation mechanisms and error message presentation on novice programmer behavior. In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2020. (SIGCSE '20), p. 759–765. ISBN 9781450367936. Disponível em: <https://doi.org/10.1145/3328778.3366882>. 27

KAZEROUNI, A. M.; EDWARDS, S. H.; HALL, T. S.; SHAFFER, C. A. Deveventtracker: Tracking development events to assess incremental development and procrastination. In: . New York, NY, USA: Association for Computing Machinery, 2017. (ITiCSE '17), p. 104–109. ISBN 9781450347044. Disponível em: <https://doi.org/10.1145/3059009.3059050>. 80

KEEFER, M. W.; WILSON, S. E.; DANKOWICZ, H.; LOUI, M. C. The importance of formative assessment in science and engineering ethics education: Some evidence and practical advice. *Science and Engineering Ethics*, Springer, v. 20, n. 1, p. 249–260, 2014. 133

KELLER, J. M. *Motivational design for learning and performance: The ARCS model approach*. [S.l.]: Springer Science & Business Media, 2009. 139, 142

KEUNING, H.; JEURING, J.; HEEREN, B. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)*, ACM New York, NY, USA, v. 19, n. 1, p. 1–43, 2018. 27

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 76, 88

KINNUNEN, P.; SIMON, B. Experiencing programming assignments in cs1: the emotional toll. In: *Proceedings of the Sixth international workshop on Computing education research*. [S.l.: s.n.], 2010. p. 77–86. 140, 141

KIZILCEC, R. F.; LEE, H. Algorithmic fairness in education. *arXiv preprint arXiv:2007.05443*, 2020. 236

KULIK, J. A.; FLETCHER, J. Effectiveness of intelligent tutoring systems: a meta-analytic review. *Review of Educational Research*, SAGE Publications Sage CA: Los Angeles, CA, v. 86, n. 1, p. 42–78, 2016. 27

KULKARNI, P. V.; RAI, S.; KALE, R. Recommender system in elearning: A survey. In: SPRINGER. *Proceeding of International Conference on Computational Science and Applications*. [S.l.], 2020. p. 119–126. 139

Kumar Veerasamy, A.; D'Souza, D.; Apiola, M. V.; Laakso, M. J.; Salakoski, T. Using early assessment performance as early warning signs to identify at-risk students in programming courses. In: *2020 IEEE Frontiers in Education Conference (FIE)*. [S.l.: s.n.], 2020. p. 1–9. 29

KURNIA, A.; LIM, A.; CHEANG, B. Online judge. *Computers Education*, v. 36, n. 4, p. 299–315, 2001. ISSN 0360-1315. 21, 26, 68

LACHENBRUCH, P. A. Mcnemar test. *Wiley StatsRef: Statistics Reference Online*, Wiley Online Library, 2014. 182

LEE, D. M. C.; RODRIGO, M. M. T.; BAKER, R. S. d; SUGAY, J. O.; CORONEL, A. Exploring the relationship between novice programmer confusion and achievement. In: SPRINGER. *International conference on affective computing and intelligent interaction*. [S.l.], 2011. p. 175–184. 139, 142

LEEUWEN, A. van; BOS, N.; RAVENSWAAIJ, H. van; OOSTENRIJK, J. van. The role of temporal patterns in students' behavior for predicting course performance: A comparison of two blended learning courses. *British Journal of Educational Technology*, Wiley Online Library, v. 50, n. 2, p. 921–933, 2019. 51, 143, 170

LEHTINEN, T.; LUKKARINEN, A.; HAARANEN, L. Students struggle to explain their own program code. *arXiv preprint arXiv:2104.06710*, 2021. 164, 166, 190

LEINONEN, J.; LONGI, K.; KLAMI, A.; VIHAVAINEN, A. Automatic inference of programming performance and experience from typing patterns. In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. New York, NY, USA: Association for Computing Machinery, 2016. (SIGCSE '16), p. 132–137. ISBN 9781450336857. 29, 52, 58, 76, 78, 79, 80, 81, 93, 101, 143, 145, 146

LIMA, M.; CARVALHO, L. S. G. de; OLIVEIRA, E. H. T. de; OLIVEIRA, D. B. F.; PEREIRA, F. D. Classificação de dificuldade de questões de programação com base em métricas de código. In: SBC. *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. [S.l.], 2020. p. 1323–1332. 21, 78

LIMA, M. A.; CARVALHO, L. S.; OLIVEIRA, E. H. de; OLIVEIRA, D. B. de; PEREIRA, F. D. Uso de atributos de código para classificar a dificuldade de questões de programação em juízes online. *Revista Brasileira de Informática na Educação*, v. 29, p. 1137–1157, 2021. 27, 142

LIMA, M. A. P. de L.; CARVALHO, L. S. G. de; OLIVEIRA, E. H. T. de; OLIVEIRA, D. B. F. de; PEREIRA, F. D. Uso de atributos de código para classificação da facilidade de questões de codificação. In: SBC. *Anais do Simpósio Brasileiro de Educação em Computação*. [S.l.], 2021. p. 113–122. 20

LIPOVETSKY, S.; CONKLIN, M. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, Wiley Online Library, v. 17, n. 4, p. 319–330, 2001. 111

LIU, C.; GONG, S.; LOY, C. C.; LIN, X. Person re-identification: What features are important? In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2012. p. 391–401. 109

LOPES, N.; FIGUEIREDO, J.; GARCÍA-PEÑALVO, F. Predicting student failure in an introductory programming course with multiple back-propagation. ACM, 2019. 143

LUNDBERG, S. M.; ERION, G.; CHEN, H.; DEGRAVE, A.; PRUTKIN, J. M.; NAIR, B.; KATZ, R.; HIMMELFARB, J.; BANSAL, N.; LEE, S.-I. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, v. 2, n. 1, p. 56–67, 2020. 107, 110, 113, 114, 115

LUNDBERG, S. M.; LEE, S.-I. A unified approach to interpreting model predictions. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2017. p. 4765–4774. 30, 110, 111, 112, 113, 115, 120

LUNDBERG, S. M.; NAIR, B.; VAVILALA, M. S.; HORIBE, M.; EISSES, M. J.; ADAMS, T.; LISTON, D. E.; LOW, D. K.-W.; NEWMAN, S.-F.; KIM, J. *et al.* Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature biomedical engineering*, Nature Publishing Group, v. 2, n. 10, p. 749–760, 2018. 122

LUXTON-REILLY, A.; ALBLUWI, I.; BECKER, B. A.; GIANNAKOS, M.; KUMAR, A. N.; OTT, L.; PATERSON, J.; SCOTT, M. J.; SHEARD, J.; SZABO, C. Introductory programming: a systematic literature review. In: *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. [S.l.: s.n.], 2018. p. 55–106. 20, 21, 27, 30, 48, 75, 78, 102, 139, 141, 168, 170

MACQUEEN, J. *et al.* Some methods for classification and analysis of multivariate observations. In: OAKLAND, CA, USA. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. [S.l.], 1967. v. 1, n. 14, p. 281–297. 62

MAHAJAN, G.; SAINI, B. Educational data mining: A state-of-the-art survey on tools and techniques used in edm. *International Journal of Computer Applications & Information Technology*, International Journal of Computer Applications & Information Technology, v. 12, n. 1, p. 310–316, 2020. 87

MANGIAFICO, S. S. Summary and analysis of extension program evaluation in r. *Rutgers Cooperative Extension: New Brunswick, NJ, USA*, v. 125, p. 16–22, 2016. 63

MANOUSELIS, N.; DRACHSLER, H.; VUORIKARI, R.; HUMMEL, H.; KOPER, R. Recommender systems in technology enhanced learning. In: *Recommender systems handbook*. [S.l.]: Springer, 2011. p. 387–415. 139

MANSOURY, M.; BURKE, R.; MOBASHER, B. Flatter is better: Percentile transformations for recommender systems. *arXiv preprint arXiv:1907.07766*, 2019. 146

MARGULIEUX, L. E.; MORRISON, B. B.; DECKER, A. Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education*, Springer, v. 7, p. 1–16, 2020. 143, 170

MARWAN, S.; LYTLE, N.; WILLIAMS, J. J.; PRICE, T. The impact of adding textual explanations to next-step hints in a novice programming environment. In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. [S.l.: s.n.], 2019. p. 520–526. 27

MARWAN, S.; WILLIAMS, J. J.; PRICE, T. An evaluation of the impact of automated programming hints on performance and learning. In: *Proceedings of the 2019 ACM Conference on International Computing Education Research*. [S.l.: s.n.], 2019. p. 61–70. 27

MELO, R.; PESSOA, M.; HAYDAR, G.; OLIVEIRA, D. B. F. de; OLIVEIRA, E. H. T. de; CARVALHO, L. S. G. de; CONTE, T.; PEREIRA, F. D. Um estudo sobre a relação entre os elementos de jogos e os tipos de usuários de sistemas gamificados. In: SBC. *Anais Estendidos do I Simpósio Brasileiro de Educação em Computação*. [S.l.], 2021. p. 35–36. 143

MENDONÇA, E.; CARVALHO, L. S.; SANTOS, A. L. M. dos; OLIVEIRA, E. H.; OLIVEIRA, D. B.; PEREIRA, F. D. Vivência acadêmica e desempenho acadêmico de ingressantes em cursos de computação. In: SBC. *Anais do XXIX Workshop sobre Educação em Computação*. [S.l.], 2021. p. 458–467. 20

MENÉNDEZ, I. Y. C.; NAPA, M. A. C.; MOREIRA, M. L. M.; ZAMBRANO, G. G. V. The importance of formative assessment in the learning teaching process. *International Journal of Social Sciences and Humanities*, v. 3, n. 2, p. 238–249, 2019. 133

MIIKKULAINEN, R.; LIANG, J.; MEYERSON, E.; RAWAL, A.; FINK, D.; FRANCON, O.; RAJU, B.; SHAHRZAD, H.; NAVRUZYAN, A.; DUFFY, N. *et al.* Evolving deep neural networks. In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. [S.l.]: Elsevier, 2019. p. 293–312. 81

MIRMAN, D. *Growth curve analysis and visualization using R*. [S.l.]: CRC press, 2016. 183

MOLENAAR, I.; HORVERS, A.; BAKER, R. S. Towards hybrid human-system regulation: Understanding children'srl support needs in blended classrooms. In: *Proceedings of the 9th International Conference on Learning Analytics & Knowledge*. [S.l.: s.n.], 2019. p. 471–480. 30, 31, 198

MOLNAR, C. *Interpretable Machine Learning*. [S.l.]: Lulu. com, 2020. 75, 111, 112

MONTGOMERY, D. C.; PECK, E. A.; VINING, G. G. *Introduction to linear regression analysis*. [S.l.]: John Wiley & Sons, 2012. v. 821. 89, 95

MÜLLER, A.; BELLHÄUSER, H.; KONERT, J.; RÖPKE, R. Effects of group formation on student satisfaction and performance: A field experiment. *Small Group Research*, SAGE Publications Sage CA: Los Angeles, CA, p. 1046496420988592, 2021. 134

MUNSON, J. P.; ZITOVSKY, J. P. Models for early identification of struggling novice programmers. In: ACM. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. [S.l.], 2018. p. 699–704. 52, 63, 76

MURDOCH, W. J.; SINGH, C.; KUMBIER, K.; ABBASI-ASL, R.; YU, B. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 116, n. 44, p. 22071–22080, 2019. 111

NEWELL, A. *Unified theories of cognition*. [S.l.]: Harvard University Press, 1994. 31, 32, 198

NEWMAN, M. E. Power laws, pareto distributions and zipf's law. *Contemporary physics*, Taylor & Francis, v. 46, n. 5, p. 323–351, 2005. 99

NGAI, G.; LAU, W. W.; CHAN, S. C.; LEONG, H.-v. On the implementation of self-assessment in an introductory programming course. *ACM SIGCSE Bulletin*, ACM New York, NY, USA, v. 41, n. 4, p. 85–89, 2010. 139

OLIVEIRA, D. B. de; FILHO, R. M. L.; OLIVEIRA, E. H.; CARVALHO, L. S.; PEREIRA, F. D.; COLONNA, J. G.; MENEZES, A. Um método de detecção de plágio para sistemas juiz on-line baseado no comportamento dos alunos. In: SBC. *Anais do XXXII Simpósio Brasileiro de Informática na Educação*. [S.l.], 2021. p. 836–848. 51

OLIVEIRA, J. C.; CARVALHO, L. S. G. de; OLIVEIRA, E. H. T. de; OLIVEIRA, D. B. F. de; PEREIRA, F. D. Análise de correlação entre habilidade de resolução de problemas e desempenho em disciplinas de programação. In: SBC. *Anais Estendidos do I Simpósio Brasileiro de Educação em Computação*. [S.l.], 2021. p. 43–44. 27

OLIVEIRA, J. de; SALEM, F.; OLIVEIRA, E. H. T. de; OLIVEIRA, D. B. F.; CARVALHO, L. S. G. de; PEREIRA, F. D. Os estudantes leem as mensagens de feedback estendido exibidas em juízes online? In: SBC. *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. [S.l.], 2020. p. 1723–1732. 20, 27, 52, 142

OLSON, R. S.; BARTLEY, N.; URBANOWICZ, R. J.; MOORE, J. H. Evaluation of a tree-based pipeline optimization tool for automating data science. In: ACM. *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. [S.l.], 2016. p. 485–492. 74, 86, 87

OTERO, J.; JUNCO, L.; SUAREZ, R.; PALACIOS, A.; COUSO, I.; SANCHEZ, L. Finding informative code metrics under uncertainty for predicting the pass rate of online courses. *Information Sciences*, Elsevier, v. 373, p. 42–56, 2016. 51, 52, 58, 76, 81, 143, 145, 146

PEDREGOSA, F.; GAëL, V.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, JMLR. org, v. 12, p. 2825–2830, 2011. 84, 85

PEREIRA, F.; OLIVEIRA, E.; FERNANDES, D.; JUNIOR, H.; CARVALHO, L. S. G. de. Otimização e automação da predição precoce do desempenho de alunos que utilizam juízes online: uma abordagem com algoritmo genético. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2019. v. 30, n. 1, p. 1451. 78, 114, 116, 142

PEREIRA, F. D.; FONSECA, S. C.; OLIVEIRA, E. H.; OLIVEIRA, D. B.; CRISTEA, A. I.; CARVALHO, L. S. Deep learning for early performance prediction of introductory programming students: a comparative and explanatory study. *Brazilian journal of computers in education.*, RBIE Editors, v. 28, p. 723–749, 2020. 116, 138

PEREIRA, F. D.; FONSECA, S. C.; OLIVEIRA, E. H.; CRISTEA, A. I.; BELLHÄUSER, H.; RODRIGUES, L.; OLIVEIRA, D. B.; ISOTANI, S.; CARVALHO, L. S. Explaining individual and collective programming students' behavior by interpreting a black-box predictive model. *IEEE Access*, IEEE, v. 9, p. 117097–117119, 2021. 29, 30, 73, 138, 170

PEREIRA, F. D.; FONSECA, S. C.; WIKTOR, S.; OLIVEIRA, D.; CRISTEA, A.; BENEDICT, A.; FALLAHIAN, M.; DORODCHI, M.; CARVALHO, L. S.; OLIVEIRA, E. Towards supporting cs1 instructors and learners with fine-grained topic detection in online judges. TechRxiv, 2022. 46, 175

PEREIRA, F. D.; JUNIOR, H. B.; RODRIGUEZ, L.; TODA, A.; OLIVEIRA, E. H.; CRISTEA, A. I.; OLIVEIRA, D. B.; CARVALHO, L. S.; FONSECA, S. C.; ALAMRI, A. *et al.* A recommender system based on effort: Towards minimising negative affects and maximising achievement in cs1 learning. In: SPRINGER. *International Conference on Intelligent Tutoring Systems*. [S.l.], 2021. p. 466–480. 21, 27, 138, 163, 170

PEREIRA, F. D.; JÚNIOR, H. B. de F.; OLIVEIRA, E. H. T. de; CARVALHO, L. S. G. de; OLIVEIRA, D. B. F. de; BENEDICT, A.; DORODCHI, M.; CRISTEA, A. I. Automatic creating variation of cs1 assignments and exams. In: SBC. *Anais Estendidos do I Simpósio Brasileiro de Educação em Computação*. [S.l.], 2021. p. 21–22. 21

PEREIRA, F. D.; OLIVEIRA, E.; CRISTEA, A.; FERNANDES, D.; SILVA, L.; AGUIAR, G.; ALAMRI, A.; ALSHEHRI, M. Early dropout prediction for programming courses supported by online judges. In: SPRINGER. *International Conference on Artificial Intelligence in Education*. [S.l.], 2019. p. 67–72. 75, 142

PEREIRA, F. D.; OLIVEIRA, E. H.; FERNANDES, D.; CRISTEA, A. Early performance prediction for cs1 course students using a combination of machine learning and an evolutionary algorithm. In: IEEE. *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*. [S.l.], 2019. v. 2161, p. 183–184. 29, 75, 78, 142

PEREIRA, F. D.; OLIVEIRA, E. H.; OLIVEIRA, D.; CRISTEA, A. I.; CARVALHO, L.; FONSECA, S.; TODA, A.; ISOTANI, S. Using learning analytics in the amazonas: understanding students' behaviour in introductory programming. *British journal of educational technology.*, John Wiley, 2020. 20, 21, 27, 33, 75, 110, 130, 138, 140, 170

PEREIRA, F. D.; OLIVEIRA, E. H. T.; OLIVEIRA, D. F. B. *Uso de um método preditivo para inferir a zona de aprendizagem de alunos de programação em um ambiente de correção automática de código*. Dissertação (Mestrado em Informática) — Universidade Federal do Amazonas, Manaus, 2018. 78

PEREIRA, F. D.; OLIVEIRA, E. H. T. de; OLIVEIRA, D. B. F. de; CARVALHO, L. S. G. de; CRISTEA, A. I. Interpretable ai to understand early effective and ineffective programming behaviours from cs1 learners. In: SBC. *Anais Estendidos do I Simpósio Brasileiro de Educação em Computação*. [S.l.], 2021. p. 16–17. 109

PEREIRA, F. D.; PIRES, F.; FONSECA, S. C.; OLIVEIRA, E. H. T.; CARVALHO, L. S. G.; OLIVEIRA, D. B. F.; CRISTEA, A. I. Towards a human-ai hybrid system for categorising programming problems. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2021. (SIGCSE '21), p. 94–100. 165, 166, 167, 168, 175

PEREIRA, F. D.; SOUZA, L. M. de; OLIVEIRA, E. H. T. de; OLIVEIRA, D. B. F. de; CARVALHO, L. S. G. de. Predição de desempenho em ambientes computacionais para turmas de programação: um mapeamento sistemático da literatura. In: SBC. *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. [S.l.], 2020. p. 1673–1682. 21, 22, 26, 27, 28, 29, 30, 32, 33, 48, 57, 63, 68, 73, 75, 76, 77, 78, 107, 131, 142, 164, 168, 194, 195

PEREIRA, F. D.; TODA, A.; OLIVEIRA, E. H.; CRISTEA, A. I.; ISOTANI, S.; LARANJEIRA, D.; ALMEIDA, A.; MENDONÇA, J. Can we use gamification to predict students' performance? a case study supported by an online judge. In: SPRINGER. *International Conference on Intelligent Tutoring Systems*. [S.l.], 2020. p. 259–269. 76, 143

PESSOA, M.; MELO, R.; HAYDAR, G.; OLIVEIRA, D. B. de; CARVALHO, L. S.; OLIVEIRA, E. H. de; CONTE, T.; PEREIRA, F. D.; RODRIGUES, L.; ISOTANI, S. Uma análise dos tipos de jogadores em uma plataforma de gamificação incorporada a um sistema juiz on-line. In: SBC. *Anais do XXXII Simpósio Brasileiro de Informática na Educação*. [S.l.], 2021. p. 474–486. 21, 143

PRICE, T. W.; DONG, Y.; BARNES, T. Generating data-driven hints for open-ended programming. *International Educational Data Mining Society*, ERIC, 2016. 27

PRICE, T. W.; MARWAN, S.; WINTERS, M.; WILLIAMS, J. J. An evaluation of data-driven programming hints in a classroom setting. In: SPRINGER. *International Conference on Artificial Intelligence in Education*. [S.l.], 2020. p. 246–251. 27, 32

PROAG, V. Assessing and measuring resilience. *Procedia Economics and Finance*, Elsevier, v. 18, p. 222–229, 2014. 140

QIU, C.; SCHMITT, M.; MOU, L.; GHAMISI, P.; ZHU, X. X. Feature importance analysis for local climate zone classification using a residual convolutional neural network with multi-source datasets. *Remote Sensing*, Multidisciplinary Digital Publishing Institute, v. 10, n. 10, p. 1572, 2018. 109

QUILLE, K.; BERGIN, S. 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*. [S.l.: s.n.], 2018. p. 15–20. 75, 82, 109

QUILLE, K.; BERGIN, S. Cs1: how will they do? how can we help? a decade of research and practice. *Computer Science Education*, Taylor & Francis, v. 29, n. 2-3, p. 254–282, 2019. 28, 29, 51, 73, 77, 88, 93, 109, 110, 131, 143, 170

RAI, A. Explainable ai: From black box to glass box. *Journal of the Academy of Marketing Science*, Springer, v. 48, n. 1, p. 137–141, 2020. 111

RANGEL, J. G. C.; KING, M.; MULDNER, K. An incremental mindset intervention increases effort during programming activities but not performance. *ACM Transactions on Computing Education (TOCE)*, ACM New York, NY, USA, v. 20, n. 2, p. 1–18, 2020. 140

RIBEIRO, M. T.; SINGH, S.; GUESTRIN, C. Why should i trust you? explaining the predictions of any classifier. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. [S.l.: s.n.], 2016. p. 1135–1144. 30, 111

RICCI, F.; ROKACH, L.; SHAPIRA, B. Introduction to recommender systems handbook. In: *Recommender systems handbook*. [S.l.]: Springer, 2011. p. 1–35. 139

RIVERA, A. C.; TAPIA-LEON, M.; LUJAN-MORA, S. Recommendation systems in education: A systematic mapping study. In: SPRINGER. *International Conference on Information Technology & Systems*. [S.l.], 2018. p. 937–947. 139

RIVERS, K.; KOEDINGER, K. R. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education*, Springer, v. 27, n. 1, p. 37–64, 2017. 27, 51

ROBINS, A. V. Novice programmers and introductory programming. In: *The Cambridge Handbook of Computing Education Research*. Cambridge: Cambridge University Press, 2019. cap. 12, p. 327–376. 20, 21, 29, 33, 48, 50, 73, 75, 76, 77, 81, 82, 88, 102, 103, 108, 109, 110, 131, 164, 166, 190

RODRIGO, M. M. T.; BAKER, R. S. Coarse-grained detection of student frustration in an introductory programming course. In: *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*. New York, NY, USA: Association for Computing Machinery, 2009. (ICER '09), p. 75–80. ISBN 9781605586151. Disponível em: <https://doi.org/10.1145/1584322.1584332>. 139, 140, 141

RODRÍGUEZ-TRIANA, M. J.; PRIETO, L. P.; MARTÍNEZ-MONÉS, A.; ASENSIO-PÉREZ, J. I.; DIMITRIADIS, Y. The teacher in the loop: Customizing multimodal learning analytics for blended learning. In: *Proceedings of the 8th international conference on learning analytics and knowledge*. [S.l.: s.n.], 2018. p. 417–426. 22

ROEDIGER-III, H. L.; KARPICKE, J. D. Test-enhanced learning: Taking memory tests improves long-term retention. *Psychological science*, SAGE Publications Sage CA: Los Angeles, CA, v. 17, n. 3, p. 249–255, 2006. 141

ROMERO, C.; VENTURA, S. Guest editorial: special issue on early prediction and supporting of learning performance. *IEEE Transactions on Learning Technologies*, IEEE, v. 12, n. 2, p. 145–147, 2019. 63, 75, 78, 107, 109

ROSENZWEIG, E. Chapter 3 - UX Thinking. In: ROSENZWEIG, E. (Ed.). *Successful User Experience: Strategies and Roadmaps*. Boston: Morgan Kaufmann, 2015. p. 41–67. ISBN 9780128009857. Disponível em: <https://www.sciencedirect.com/science/article/pii/B978012800985700003X>. 200

ROUSSEEUW, P. J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, Elsevier, v. 20, p. 53–65, 1987. 62

ROWLAND, C. A. The effect of testing versus restudy on retention: a meta-analytic review of the testing effect. *Psychological Bulletin*, American Psychological Association, v. 140, n. 6, p. 1432, 2014. 123, 133, 141, 157, 190

RUMMEL, N. One framework to rule them all? carrying forward the conversation started by wise and schwarz. *International Journal of Computer-Supported Collaborative Learning*, Springer, v. 13, n. 1, p. 123–129, 2018. 31, 32, 198

SAITO, T.; WATANOBE, Y. Learning path recommendation system for programming education based on neural networks. *International Journal of Distance Education Technologies (IJDET)*, IGI Global, v. 18, n. 1, p. 36–64, 2020. 138, 139, 142, 143, 144, 168, 169, 170

SANTOS, I. L. dos; OLIVEIRA, D. B. F.; CARVALHO, L. S. G. de; PEREIRA, F. D.; OLIVEIRA, E. H. T. de. Tempos de transição em estados de corretude e erro como indicadores de desempenho em juízes online. In: SBC. *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. [S.l.], 2020. p. 1283–1292. 52

SATOPAA, V.; ALBRECHT, J.; IRWIN, D.; RAGHAVAN, B. Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In: IEEE. *2011 31st international conference on distributed computing systems workshops*. [S.l.], 2011. p. 166–171. 124

SETTLE, A.; LALOR, J.; STEINBACH, T. Reconsidering the impact of cs1 on novice attitudes. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2015. (SIGCSE '15), p. 229–234. ISBN 9781450329668. Disponível em: <https://doi.org/10.1145/2676723.2677235>. 139

SHAPLEY, L. S. A value for n-person games. In: ____. *Contributions to the Theory of Games (AM-28), Volume II*. [S.l.: s.n.], 1953. p. 307–317. 112

SHI, L.; CRISTEA, A. I. In-depth exploration of engagement patterns in moocs. In: SPRINGER. *International conference on web information systems engineering*. [S.l.], 2018. p. 395–409. 62

SHI, L.; CRISTEA, A. I.; TODA, A. M.; OLIVEIRA, W. Revealing the hidden patterns: A comparative study on profiling subpopulations of mooc students. *arXiv preprint arXiv:2008.05850*, 2020. 62

SHRIKUMAR, A.; GREENSIDE, P.; KUNDAJE, A. Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*, 2017. 111

SMITH, L. M.; TODD, L.; LAING, K. Students' views on fairness in education: the importance of relational justice and stakes fairness. *Research Papers in Education*, Taylor & Francis, v. 33, n. 3, p. 336–353, 2018. 236

SONG, P.; GENG, C.; LI, Z. Research on text classification based on convolutional neural network. In: IEEE. *2019 International Conference on Computer Network, Electronic and Automation (ICCNEA)*. [S.l.], 2019. p. 229–232. 176

SOUZA, D. M.; FELIZARDO, K. R.; BARBOSA, E. F. A systematic literature review of assessment tools for programming assignments. In: IEEE. *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*. [S.l.], 2016. p. 147–156. 20, 21, 26, 27

SPACCO, J.; FOSSATI, D.; STAMPER, J.; RIVERS, K. Towards improving programming habits to create better computer science course outcomes. In: ACM. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. [S.l.], 2013. p. 243–248. 52

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014. 76, 88

STASH, N. V.; CRISTEA, A. I.; BRA, P. M. D. Authoring of learning styles in adaptive hypermedia: problems and solutions. In: *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. [S.l.: s.n.], 2004. p. 114–123. 132

STEEL, P. The nature of procrastination: A meta-analytic and theoretical review of quintessential self-regulatory failure. *Psychological Bulletin*, v. 133, n. 1, p. 65–94, 2007. 130, 131

TENÓRIO, K.; DERMEVAL, D.; MONTEIRO, M.; PEIXOTO, A.; SILVA, A. P. d. Exploring design concepts to enable teachers to monitor and adapt gamification in adaptive learning systems: A qualitative research approach. *International Journal of Artificial Intelligence in Education*, Springer, p. 1–25, 2021. 21, 22, 27, 29, 30, 73, 206, 225

TODA, A.; PEREIRA, F. D.; KLOCK, A. C. T.; RODRIGUES, L.; PALOMINO, P.; OLIVEIRA, W.; OLIVEIRA, E. H. T.; GASPARINI, I.; CRISTEA, A. I.; ISOTANI, S. For whom should we gamify? insights on the users intentions and context towards gamification in education. In: SBC. *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. [S.l.], 2020. p. 471–480. 141

TOLEDO, R. Y.; MOTA, Y. C. An e-learning collaborative filtering approach to suggest problems to solve in programming online judges. *International Journal of Distance Education Technologies (IJDET)*, IGI Global, v. 12, n. 2, p. 51–65, 2014. 142, 143, 144

TOLEDO, R. Y.; MOTA, Y. C.; MARTÍNEZ, L. A recommender system for programming online judges using fuzzy information modeling. In: MULTIDISCIPLINARY DIGITAL PUBLISHING INSTITUTE. *Informatics*. [S.l.], 2018. v. 5, n. 2, p. 17. 138, 139, 143, 144

TOMASEVIC, N.; GVOZDENOVIC, N.; VRANES, S. An overview and comparison of supervised data mining techniques for student exam performance prediction. *Computers & Education*, Elsevier, v. 143, p. 103676, 2020. 93

TOMEK, I. Two modifications of cnn. IEEE Transactions on Systems Man and Communications SMC-6, p. 769–772, 1976. 84

TRUONG, K. N.; HAYES, G. R.; ABOWD, G. D. Storyboarding: an empirical determination of best practices and effective guidelines. In: *Proceedings of DIS*. [S.l.]: Association for Computing Machinery (ACM), 2006. p. 12. 200

ULLAH, Z.; LAJIS, A.; JAMJOOM, M.; ALTALHI, A.; AL-GHAMDI, A.; SALEEM, F. The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. *Computer Applications in Engineering Education*, Wiley Online Library, v. 26, n. 6, p. 2328–2341, 2018. 20, 55

UMAPATHY, K.; RITZHAUPT, A. D. A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education (TOCE)*, ACM New York, NY, USA, v. 17, n. 4, p. 1–13, 2017. 140

VANLEHN, K. Regulative loops, step loops and task loops. *International Journal of Artificial Intelligence in Education*, Springer, v. 26, n. 1, p. 107–112, 2016. 30

VANLEHN, K.; BURKHARDT, H.; CHEEMA, S.; KANG, S.; PEAD, D.; SCHOENFELD, A.; WETZEL, J. Can an orchestration system increase collaborative, productive struggle in teaching-by-eliciting classrooms? *Interactive Learning Environments*, Taylor & Francis, v. 29, n. 6, p. 987–1005, 2021. 31, 32, 198

VICIAN, C.; CHARLESWORTH, D. D.; CHARLESWORTH, P. Students' perspectives of the influence of web-enhanced coursework on incidences of cheating. *Journal of Chemical Education*, ACS Publications, v. 83, n. 9, p. 1368, 2006. 164

VIHAVAINEN, A. Predicting students' performance in an introductory programming course using data from students' own programming process. *Proceedings - 2013 IEEE 13th International Conference on Advanced Learning Technologies, ICALT 2013*, p. 498–499, 2013. ISSN 9780769550091. 52

VIHAVAINEN, A.; LUUKKAINEN, M.; IHANTOLA, P. Analysis of source code snapshot granularity levels. In: ACM. *Proceedings of the 15th Annual Conference on Information technology education*. [S.l.], 2014. p. 21–26. 28, 29

WASIK, S.; ANTCZAK, M.; BADURA, J.; LASKOWSKI, A.; STERNAL, T. A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)*, ACM, v. 51, n. 1, p. 3, 2018. 21, 22, 26, 27, 32, 68, 77, 78, 102, 138, 139, 143, 153, 164, 165, 166, 167, 168, 194, 230

WATSON, C.; LI, F. W. B.; GODWIN, J. L. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. *2013 IEEE 13th International Conference on Advanced Learning Technologies*, p. 319–323, 2013. 29, 52, 58, 76, 78, 79, 81, 93, 101, 143, 170

WOLPERT, D. H. Stacked generalization. *Neural networks*, Elsevier, v. 5, n. 2, p. 241–259, 1992. 88

WOLPERT, D. H. The supervised learning no-free-lunch theorems. In: *Soft computing and industry*. [S.l.]: Springer, 2002. p. 25–42. 114

WOOLF, B. P. *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. [S.l.]: Morgan Kaufmann, 2010. 21

YADIN, A. Using unique assignments for reducing the bimodal grade distribution. *ACM Inroads*, ACM New York, NY, USA, v. 4, n. 1, p. 38–42, 2013. 76

YERA, R.; MARTÍNEZ, L. A recommendation approach for programming online judges supported by data preprocessing techniques. *Applied Intelligence*, Springer, v. 47, n. 2, p. 277–290, 2017. 60, 102, 138, 142, 143, 144, 165, 166, 169, 170, 177, 178, 182, 183

YU, R.; CAI, Z.; DU, X.; HE, M.; WANG, Z.; YANG, B.; CHANG, P. The research of the recommendation algorithm in online learning. *International Journal of Multimedia and Ubiquitous Engineering*, v. 10, n. 4, p. 71–80, 2015. 138, 139

ZHAO, W. X.; ZHANG, W.; HE, Y.; XIE, X.; WEN, J.-R. Automatically learning topics and difficulty levels of problems in online judge systems. *ACM Transactions on Information Systems (TOIS)*, ACM, v. 36, n. 3, p. 27, 2018. 135, 138, 139, 150, 163, 164, 165, 166, 167, 168, 231

ZHOU, W.; PAN, Y.; ZHOU, Y.; SUN, G. The framework of a new online judge system for programming education. In: ACM. *Proceedings of ACM Turing Celebration Conference-China*. [S.l.], 2018. p. 9–14. 26, 27, 29

ZIMMERMAN, B. J. Investigating self-regulation and motivation: Historical background, methodological developments, and future prospects. *American educational research journal*, SAGE Publications Sage CA: Thousand Oaks, CA, v. 45, n. 1, p. 166–183, 2008. 235

ZIMMERMAN, J.; FORLIZZI, J. Speed Dating: Providing a Menu of Possible Futures. *She Ji*, Tongji University Press, v. 3, n. 1, p. 30–50, 3 2017. ISSN 24058718. 200

ZUTTY, J.; LONG, D.; ADAMS, H.; BENNETT, G.; BAXTER, C. Multiple objective vector-based genetic programming using human-derived primitives. In: ACM. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. [S.l.], 2015. p. 1127–1134. 86