UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

USO DE UM MÉTODO PREDITIVO PARA INFERIR A ZONA DE APRENDIZAGEM DE ALUNOS DE PROGRAMAÇÃO EM UM AMBIENTE DE CORREÇÃO AUTOMÁTICA DE CÓDIGO

FILIPE DWAN PEREIRA

Março de 2018

Manaus - AM

FILIPE DWAN PEREIRA

USO DE UM MÉTODO PREDITIVO PARA INFERIR A ZONA DE APRENDIZAGEM DE ALUNOS DE PROGRAMAÇÃO EM UM AMBIENTE DE CORREÇÃO AUTOMÁTICA DE CÓDIGO

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas (PPGI/IComp, UFAM) como parte dos requisitos necessários à obtenção do título de Mestre em Informática

Orientadora: Elaine Harada Teixeira de Oliveira, D.Sc.

Co-orientador: David Braga Fernandes de Oliveira, D.Sc.

Março de 2018

Manaus - AM

Ficha Catalográfica

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

Pereira, Filipe Dwan
P436u Uso de um método preditivo para inferir a zona de aprendizagem de alunos de programação em um ambiente de correção automática de código / Filipe Dwan Pereira. 2018
II f.: 31 cm.

Orientadora: Elaine Harada Texeira de Oliveira
Coorientadora: David Braga Fernandes de Oliveira
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. alunos de programação. 2. aprendizagem de máquina. 3. learning analytics. 4. métricas de software data-driven. 5. juízes online. I. Oliveira, Elaine Harada Texeira de II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Uso de um método preditivo para inferir a zona de aprendizagem de alunos de programação em um ambiente de correção automática de código"

FILIPE DWAN PEREIRA

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Profa. Elaine Harada Teixeira de Oliveira - PRESIDENTE

Prof. Marco Antonio Pinheiro de Cristo - MEMBRO INTERNO

Prof. Seiji Isotani - MEMBRO EXTERNO

Manaus, 29 de Março de 2018

USO DE UM MÉTODO PREDITIVO PARA INFERIR A ZONA DE APRENDIZAGEM DE ALUNOS DE PROGRAMAÇÃO EM UM AMBIENTE DE CORREÇÃO AUTOMÁTICA DE CÓDIGO

FILIPE DWAN PEREIRA

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DO INSTITUTO DE COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DO AMAZONAS COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM INFORMÁTICA

.

Aprovado por:

Profa. Elaine Harada Teixeira de Oliveira, D.Sc.
Instituto de Computação (IComp)
Universidade Federal do Amazonas (UFAM)

Prof. David Braga Fernandes de Oliveira, D.Sc.
Instituto de Computação (IComp)
Universidade Federal do Amazonas (UFAM)

Prof. Marco Antônio Pinheiro de Cristo, D.Sc.
Instituto de Computação (IComp)
Universidade Federal do Amazonas (UFAM)

Prof. Seiji Isotani, D.Sc.
Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP/São Carlos)

MARÇO DE 2018

MANAUS – AM

Que darei eu ao Senhor, por todos os benefícios que me tem feito? Tomarei o cálice da salvação, e invocarei o nome do Senhor.
Salmos 116:12,13.

Agradecimentos

Agradeço primeiramente a Deus, pois tudo que eu tenho e sou provém Dele, o criador e detentor de todo o saber. As Tuas infinitas maravilhas resplandecem em todos os lugares. Deus meu, Tu és bom para comigo. Quando olho para trás, ao longo desses dois anos de mestrado, eu vejo a tua glória e misericórdia na minha vida. Sem Ti, meu Deus, certamente eu não teria conseguido. Tua grandeza, soberania e bondade não cessam.

Depois do meu Deus, agradeço minha maior incentivadora, minha mãe. Desde cedo me mostrou que o caminho para vencer era seguir a Deus e estudar. Mostrou-me que não existe atalhos para lugares que valem a pena chegar e que nós precisamos subir as escadas, um passo de cada vez. Agradeço minha esposa, que levantava de madrugada para cuidar da nossa filhinha e me dava condições para estudar dia após dia. Você faz parte deste processo, meu amor, minha incentivadora, amo-te. Agradeço minha filhinha, Isabela, que nasceu durante esse processo e deixou os meus dias muito mais felizes. Agradeço a vó Rosa, pois o que ela fez por mim na minha infância tem impacto no que eu sou hoje.

Formalmente, agradeço a minha orientadora, professora Elaine, que sempre se mostrou solícita a me ajudar, com conselhos cirúrgicos, conduziu-me nesta jornada. Bem como, agradeço ao meu co-orientador, professor David, que com o seu saber e mansidão me deu orientações valiosas. Agradeço a Deus por ter sido orientado por vocês.

Agradeço a todos os professores da UFAM que me ajudaram a construir conhecimento, a entender mais sobre pesquisa (ainda tenho muito mesmo a aprender). Agradeço aos meus amigos por entenderem que este é só um processo e que depois eu voltarei a ter vida social (depois do doutorado). Não posso deixar de lembrar dos colegas do mestrado e doutorado fora de sede. Carlos Bruno, Dion, Gustavo, Jonathas, Manoel, Ornélio (está em ordem alfabética), eu aprendi muito com vocês.

Finalmente, agradeço a UFRR, especialmente a todos os professores do DCC pelas dicas e pelo apoio.

Resumo do Dissertação apresentada ao PPGI/IComp/UFAM como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática.

USO DE UM MÉTODO PREDITIVO PARA INFERIR A ZONA DE APRENDIZAGEM DE ALUNOS DE PROGRAMAÇÃO EM UM AMBIENTE DE CORREÇÃO AUTOMÁTICA DE CÓDIGO

FILIPE DWAN PEREIRA

Março/2018

Orientadora: Profa. Elaine Harada Teixeira de Oliveira, D.Sc.

Co-orientador: Prof. David Braga Fernandes de Oliveira, D.Sc.

Em média, um terço dos alunos no mundo reprova em disciplinas de introdução à programação de computadores (IPC). Assim, muitos estudos vêm sendo conduzidos a fim de inferir o desempenho de estudantes de turmas de IPC. Inicialmente, pesquisadores investigavam a relação das notas dos alunos com fatores estáticos como: notas no ensino médio, gênero, idade e outros. Entretanto, o comportamento dos estudantes é dinâmico e, dessa forma, abordagens orientadas aos dados vêm ganhando atenção, uma vez que muitas universidades utilizam ambientes web para turmas de programação como juízes *online*. Com efeito, muitos pesquisadores vêm extraindo e tratando os dados dos estudantes a partir desses ambientes e usando-os como atributos de algoritmos de aprendizagem de máquina para a construção de modelos preditivos. No entanto, a comunidade científica sugere que tais estudos sejam reproduzidos a fim de investigar se eles são generalizáveis a outras bases de dados educacionais. Neste sentido, neste trabalho apresentou-se um método que emprega um conjunto de atributos correlacionados com as notas dos estudantes, sendo alguns baseados em trabalhos relacionados e outros propostos nesta pesquisa, a fim de realizar a predição do desempenho dos alunos nas avaliações intermediárias e nas médias finais. Tal método foi aplicado a uma base de dados com 486 alunos de IPC. O conjunto de atributos chamado de perfil de programação foi empregado em algoritmos de aprendizagem de máquina e otimizado utilizando duas abordagens: a) ajuste de hiperparâmetros com *random search* e b) construção do

pipeline de aprendizagem de máquina utilizando algoritmos evolutivos. Como resultado, atingiu-se 74,44% de acurácia na tarefa de identificar se os alunos iriam ser reprovados ou aprovados usando os dados das duas semanas de aula em uma base de dados balanceada. Esse resultado foi estatisticamente superior ao *baseline*. Destaca-se ainda que a partir da oitava semana de aula, o método atingiu acurácias entre 85% e 90,62%.

Palavras-chave: inferência de zona de aprendizagem, alunos de programação, aprendizagem de máquina, learning analytics, educational data mining, métricas de software data-driven, juízes online.

Abstract of Qualifying Exam presented to PPGI/IComp/UFAM as a partial fulfillment of the requirements for the degree of Master in Informatics.

USE OF A PREDICTIVE METHOD TO INFER THE LEARNING ZONE OF PROGRAMMING STUDENTS IN AN ONLINE JUDGE

FILIPE DWAN PEREIRA

March/2018

Advisor: Prof. Elaine Harada Teixeira de Oliveira, D.Sc.

Co-advisor: Prof. David Braga Fernandes de Oliveira, D.Sc.

CS1 (first year programming) classes are known to have a high dropout and non-pass rate. Thus, there have been many studies attempting to predict and alleviate CS1 student performance. Knowing about student performance in advance can be useful for many reasons. For example, teachers can apply specific actions to help learners who are struggling, as well as provide more challenging activities to high-achievers. Initial studies used static factors, such as: high school grades, age, gender. However, student behavior is dynamic and, as such, a data-driven approach has been gaining more attention, since many universities are using web-based environments to support CS1 classes. Thereby, many researchers have started extracting student behavior by cleaning data collected from these environments and using them as features in machine learning (ML) models. Recently, the research community has proposed many predictive methods available, even though many of these studies would need to be replicated, to check if they are context-sensitive. Thus, we have collected a set of successful features correlated with the student grade used in related studies, compiling the best ML attributes, as well as adding new features, and applying them on a database representing 486 CS1 students. The set of features was used in ML pipelines which were optimized with two approaches: hyperparameter-tuning with random search and genetic programming. As a result, we achieved an accuracy of

74.44%, using data from the first two weeks to predict student final grade, which outperforms a state-of-the-art research applied to the same dataset. It is also worth noting that from the eighth week of class, the method achieved accuracy between 85% and 90.62%.

Keywords: CS1, programming students, machine learning, learning analytics, genetic programming, data-driven software metrics.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AB AdaBoost

ACAC Ambiente de Correção Automática de Código

ACM Association for Computing Machinery

AD Árvore de Decisão

AG Algoritmo Genético

AM Aprendizagem de Máquina

AutoML Automated Machine Learning

EDM Educational Data Mining

EQ Error Quocient

ERT Extremely Randomized Trees

FNN Feedforward Neural Network

FPR False Negative Rate

GP Genetic Programming

GTB Gradient Tree Boosting

IDE Integrated Development Environment

IPC Introdução a Programação de Computadores

KNN K Nearest Neighbor

LA Learning Analytics

LAPES Laboratório de Engenharia de Software

MD Mineração de Dados

MMQ Método dos Mínimos Quadrados

MSE Mean Squared Error

MSL Mapeamento Sistemático da Literatura

RED Repeated Error Density

RF Random Forest

RFE Recursive Feature Elimination

RMSE Root Mean Squared Error

RNP Redes Neurais Profundas

ROC Receiver Operating Characteristics

RSL Revisão Sistemática da Literatura

SLOC Source Lines Of Code

SVM Support Vector Machine

SVR Support Vector Regression

TNR True Negative Rate

TPOT Tree based Pipeline Optimization Tool

TPR True positive Rate

WEKA Waikato Environment for Knowledge Analysis

ZA Zona de Aprendizagem

ZD Zona de Dificuldade

ZE Zona de Expertise

Capítulo 1

Introdução

Turmas de introdução à programação de computadores (IPC) costumam ser numerosas (ANDERSEN et al., 2016), o que dificulta o acompanhamento dos alunos e impossibilita uma abordagem individualizada. Além disso, a literatura considera as disciplinas introdutórias de programação de alta complexidade para alunos iniciantes, pois demandam uma série de habilidades cognitivas que, segundo Pea e Kurland (1984), são: a) compreender o problema; b) planejar ou projetar uma solução; c) escrever código do plano de solução; d) depurar o programa. Outros autores como Lahtinen, Ala-Mutka e Järvinen (2005), Gomes e Mendes (2015) ponderam que a aprendizagem de programação requer competências prévias como: pensamento crítico, generalização, abstração, transferência, resolução de problemas, entre outros.

Com efeito, existe um índice significativo de evasão e desistência em turmas de IPC (LAHTINEN; ALA-MUTKA; JÄRVINEN, 2005; TAN; TING; LING, 2009; BLIKSTEIN, 2011; IHANTOLA et al., 2015; ANDERSEN et al., 2016). Um grupo de pesquisadores conduziu um estudo que apontou que, em média, um terço dos estudantes de IPC do mundo não consegue obter o mínimo suficiente para a aprovação (WATSON; LI, 2014).

Em função disso, muitas pesquisas foram conduzidas para instrumentalizar o professor e dar suporte à sua tomada de decisão. Para exemplificar, alguns pesquisadores propuseram o uso de ambientes de correção automática de código (também conhecidos como juízes *online*), que além de disponibilizar as atividades criadas pelos instrutores, podem também possuir um ambiente de desenvolvimento integrado, onde o aluno é capaz de desenvolver e submeter as soluções dos problemas e receber um *feedback* imediato, isto é, se o código desenvolvido como solução para um dado exercício está certo ou errado.

Essas ferramentas abriram novas oportunidades de pesquisa, posto que é possível embutir nelas componentes de software capazes de monitorar e registrar todas as ações realizadas pelo aluno durante suas tentativas de solucionar os exercícios de programação propostos, tais como: teclas pressionadas, movimentos do mouse, erros e acertos, eventos de navegação no ambiente, número de tentativas e outros, sendo todos esses eventos datados com um *timestamp*[1] (TOLL, 2016). Esses dados vêm sendo utilizados principalmente nas áreas de *learning analytics* (LA) e *educational data mining* (EDM) para serem empregados na predição do desempenho dos discentes, usando técnicas de aprendizagem de máquina, análise estatística e mineração de dados, de modo que não se analise tão somente o produto enviado pelo aluno (nesse caso o código-fonte), mas todo o processo por trás dele. Segundo Blikstein et al. (2014), Ihantola et al. (2015), Becker (2016) essa abordagem *data-driven* permite uma avaliação formativa e contínua.

Nesse caminho, o presente trabalho propôs e validou um método para estimar: *a)* se alunos de IPC terão um rendimento baixo ou alto nas avaliações intermediárias; *b)* se estudantes de IPC serão aprovados ou reprovados no final da disciplina; e *c)* qual será a média final de alunos de IPC em uma escala contínua. Neste texto, adotou-se o termo Zona de Aprendizagem (ZA) para representar o rendimento dos estudantes nos itens *a*, *b* e *c*. Destaca-se que o rendimento, mencionado anteriormente, é calculado com base nas notas das avaliações intermediárias (item *a*) ou nas médias finais (item *b* e *c*). A ZA foi dividade em Zona de Expertise (ZE) e Zona de Dificuldade (ZD). O aluno é alocado na ZE quando tem rendimento maior ou igual a 5, caso contrário na ZD.

O método foi avaliado em uma base de dados extraída de 9 turmas de IPC da Universidade Federal do Amazonas (UFAM) com 486 alunos. A UFAM desenvolveu e vem usando o CodeBench[2] (que é um ACAC) em turmas de IPC. Esse sistema vem sendo utilizado pelos docentes para disponibilizar exercícios de programação para seus alunos, que por sua vez podem codificar e submeter as soluções de tais exercícios usando uma *Integrated Development Environment* (IDE) que fica embutida no próprio CodeBench. O sistema corrige automaticamente o código submetido pelo estudante, gerando um *feedback* para o aluno em tempo real, isto é, se a resposta está certa ou errada (CARVALHO; OLIVEIRA; GADELHA, 2016).

[1] Timestamp é uma estampa de tempo ou uma marca temporal que identifica quando certo evento aconteceu.

[2] http://codebench.icomp.ufam.edu.br

Para realizar a predição da ZA dos alunos, foram realizados 6 experimentos. Nos três primeiros experimentos, a zona de aprendizagem foi representada pelas notas das avaliações intermediárias e nos três últimos, pela média final dos alunos. Para viabilizar essa tarefa, foi identificado na literatura um conjunto de atributos correlacionados com as notas dos alunos, e neste trabalho são propostas novas evidências úteis para o problema apresentado. Chamamos esse conjunto de atributos de perfil de programação do aluno. De posse desse conjunto, foram utilizadas técnicas de aprendizagem de máquina na composição de modelos preditivos[3]. Utilizou-se duas abordagens para a construção dos modelos, sendo na primeira (A1) um processo manual de transformação, construção e seleção dos atributos do perfil de programação para um posterior uso de algoritmo de AM e otimização de hiperparâmetros. A segunda abordagem (A2) realiza a construção e otimização do *pipeline* de AM automaticamente utilizando um algoritmo genético. Ao final, os resultados das duas abordagens foram comparados e, em geral, os resultados da primeira foram superiores. Entretanto, a segunda abordagem se mostrou promissora após alguns ajustes manuais realizados nos *pipelines* exportados automaticamente. Em geral, os algoritmos baseados em árvores de decisão obtiveram os melhores resultados em ambas as abordagens.

Frisa-se a importância da abordagem automatizada A2, pois automatizar o processo de criação dos modelos preditivos é algo importante neste contexto, visto que acredita-se que ao longo dos semestres os padrões para realizar a predição poderão mudar. Assim, a abordagem A2 se mostra como uma alternativa viável para substituir um humano na produção de modelos preditivos que acompanhem essas mudanças.

Como resultados em uma base de dados balanceada, atingiu-se uma acurácia de 74,44% utilizando os dados apenas das duas primeiras semanas de aula na tarefa de classificar se o estudante iria passar ou reprovar na disciplina. Tal resultado foi estatisticamente superior (p-*value*$<0{,}05$) a um método proeminente proposto por Estey e Coady (2016), o qual foi aplicado na base de dados utilizada neste estudo. Destaca-se ainda que a partir da oitava semana, atingiu-se acurácias entre 85% e 90,62% na mesma tarefa.

De forma resumida, as principais contribuições deste estudo são:

[3]Neste estudo, entende-se como modelo preditivo o algoritmo de aprendizagem de máquina após o treinamento. Esse modelo é empregado para realizar inferências de variáveis dependentes utilizando observações multidimensionais. Entende-se como método preditivo o processo (as etapas) para a construção de tais modelos preditivos. Observe que uma primeira visão geral do método será apresentada na seção 1.5 desta capítulo e uma explicação mais detalhada será dada no capítulo 4.

- Propor e validar um conjunto de atributos baseados em estudos proeminentes. Demonstrou-se que esse conjunto chamado de perfil de programação pode ser usado para produzir modelos preditivos capazes de inferir a desempenho de alunos de IPC ainda no começo da disciplina.

- Demonstrar que uma abordagem automatizada de produção de modelos preditivos com o emprego de algoritmos genéticos pode ser utilizada na tarefa de predição da zona de aprendizagem dos alunos.

1.1 Justificativa

As principais dificuldades de alunos de turmas iniciais de programação estão relacionadas ao raciocínio algorítmico, o que vai muito além do paradigma de programação ou ainda da especificidade de uma linguagem. Além disso, diversos modelos e métodos têm sido propostos para minimizar esse fato recorrente em cursos de computação, tais como: aprendizagem baseada em problemas, novas estratégias motivacionais, abordagens ativas, gamificação, aprendizagem cooperativa e colaborativa (GOMES; MENDES, 2015).

Acredita-se que é possível auxiliar os alunos de turmas iniciais de programação, mas para isso é necessário criar condições de aprendizagem específicas para o discente, utilizando técnicas e métodos que complementam o modelo tradicional que vem se mostrando não tão eficaz ao longo do tempo (HELMINEN et al., 2012).

Métodos para identificação automática de estudantes de turmas de programação que precisam de suporte vêm sendo pesquisado por décadas. Inicialmente, os trabalhos que tinham por intento prever o desempenho do aluno utilizavam fatores estáticos, como o histórico no ensino médio e em disciplinas da própria graduação, ou resultados de vários questionários aplicados durante a disciplina, ou ainda, de variáveis demográficas, como a renda familiar, idade, gênero e etc. (AHADI et al., 2015). Recentemente, o registro constante do progresso do estudante e do seu comportamento ao resolver os exercício de programação, extraído de ambientes computacionais voltados para turmas de programação, vem ganhando muito atenção. Assim, muitas pesquisas foram conduzidas usando as interações dos discentes com esses sistemas para melhorar o seu processo de ensino e aprendizagem (OTERO et al., 2014b; CARTER; HUNDHAUSEN; ADESOPE, 2015; AHADI et al., 2015; AHADI; VIHAVAINEN; LISTER, 2016).

Saber antecipadamente o desempenho do aluno pode ser bem útil por várias razões. Os professores poderiam direcionar orientação específica aos discentes com um valor estimado de desempenho baixo e prover atividades mais desafiadoras aos alunos com valor estimado alto. Além disso, esses estudantes em risco poderiam ser monitorados para evitar que eles acabassem reprovando na disciplina (AHADI et al., 2015).

Pesquisas recentes, em geral, utilizam abordagens *data-driven* (dirigida aos dados), isto é, usam dados coletados automaticamente do processo de programação dos alunos para fazer a predição do desempenho, sem nenhuma informação adicional relacionada ao histórico, *background*, renda e outras (IHANTOLA et al., 2015). Blikstein (2011) explica que ao utilizar uma abordagem *data-driven*, os pesquisadores podem fazer estimativas empregando técnicas de aprendizagem de máquina e mineração de dados e avaliar o processo de desenvolvimento do aluno e não apenas o produto, isto é, o código fonte gerado, como é normalmente realizado no sistema de educação superior tradicional.

Nesse sentido, o presente estudo propôs e validou uma estrutura que pode ser adaptada a um juiz online, na qual o professor conseguirá identificar antecipadamente: a) se o aluno tem mais chance de reprovar ou de ser aprovado b) se aluno irá bem ou mal nas avaliações intermediárias c) qual será a média final dos alunos. Quando o professor tem a informação automatizada se o estudante está em uma zona de dificuldade ou expertise, esse poderá, entre outras coisas: a) monitorar os alunos da zona de dificuldade b) ter um *feedback* mais acurado do processo de ensino/aprendizagem da turma; c) a partir do ponto b), o docente poderá tomar decisões para estimular o aprendizado do aluno com dificuldade, seja com atividades específicas para um grupo de alunos com o nível de dificuldade similar ou ainda com sugestões de conteúdo, revisões, etc.; d) criar provas mais condizentes com a realidade de sua turma. Percebe-se que todas essas extensões aumentariam as possibilidades de aplicação do sistema CodeBench como um todo e poderiam ser generalizadas para outros juízes *online*. Tais funcionalidades são possíveis, entre outras coisas, através da identificação do estado de aprendizagem do aluno, isto é, se ele está em uma zona de dificuldade ou expertise.

1.2 Objetivo Geral

Propor e validar um método preditivo para inferir a zona de aprendizagem dos alunos de turmas iniciais de programação em um ambiente de correção automática de código.

1.3 Objetivos Específicos

a. Propor e avaliar um perfil de programação, baseado em atributos encontrados na literatura e outros propostos neste trabalho, que pode ser usado para inferir o sucesso ou a falha de estudantes de IPC;

b. Identificar no perfil de programação dos alunos a relevância dos atributos no decorrer da disciplina;

c. Construir e avaliar modelos preditivos capazes de realizar a inferência da zona de aprendizagem dos alunos ainda nas duas semanas de aula.

d. Investigar se os modelos preditivos vão ficando mais precisos no decorrer da disciplina;

e. Investigar se o uso de um algoritmo genético multi-objetivo é eficaz na construção automática de modelos preditivos para realizar a predição da zona de aprendizagem de alunos de IPC;

f. Interpretar os comportamentos *data-driven* de alunos de IPC que podem levar ao fracasso ou ao sucesso do aluno na disciplina.

1.4 Questões de Pesquisa

Neste trabalho propôs-se e validou-se um modelo preditivo capaz de inferir a zona de aprendizagem de estudantes utilizando evidências de quando os alunos estavam resolvendo exercícios de programação em uma IDE integrada a um sistema de correção automática de código. Especificamente, este trabalho está focada em responder as questões abaixo:

- **QP1** - Até que ponto o estudo de Estey e Coady (2016) pode ser reproduzido na base de dados apresentada neste trabalho? Além disso, o método preditivo deste estudo

apresenta avanços em relação à predição da zona de aprendizagem se comparado ao método de Estey e Coady (2016)?

- **QP2** - Utilizando o perfil de programação, é possível inferir a zona de aprendizagem de alunos de IPC com apenas os dados das duas primeiras semanas de aula? Além disso, os modelos preditivos aumentam a precisão no decorrer do curso ou ficam estagnados?

- **QP3** - O uso de um método que emprega algoritmos genéticos para a construção de modelos preditivos automaticamente proposto por Olson et al. (2016) é viável para a predição da zona de aprendizagem? Se sim, essa abordagem supera a construção manual de *pipelines* para a predição da zona de aprendizagem?

- **QP4** - Dentre os atributos do perfil de programação utilizados para realizar as predições, desde os usados na literatura até os apresentados aqui que são derivados deles, quais são os mais relevantes em relação à acurácia do modelo preditivo?

- **QP5** - A diferença entre o grupo de alunos que passaram na disciplina de IPC e os que reprovaram pode ser visualizada através do perfil de programação?

1.5 Metodologia

A Figura 1.1 ilustra o processo de construção dos modelos preditivos nos experimentos conduzidos. Primeiramente, foi realizado o pré-processamento dos dados, onde códigos submetidos e os *logs* dos alunos no ACAC foram analisados a fim de gerar os valores numéricos que compõem o perfil de programação. Depois disso, foi realizado um ciclo entre seleção de atributos, escolha do algoritmo de aprendizagem de máquina e ajuste de hiperparâmetros até se obter um modelo que maximizasse a acurácia na predição da zona de aprendizagem.

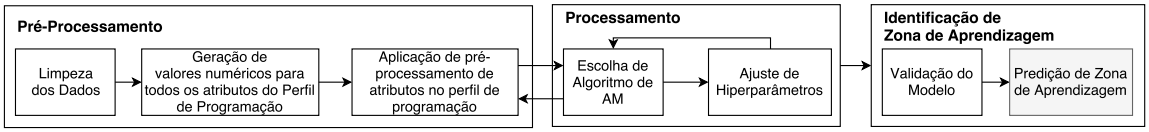


Figura 1.1: Fluxograma de aplicação do método proposto.

Utilizou-se algoritmos de AM para classificação e regressão. Os algoritmos de classificação utilizados para a construção dos modelos preditivos foram *Random Forest* (RF),

Support Vector Machine (SVM), *K-Nearest Neighbor* (KNN), *Árvore de Decisão* (AD), *Extremely Randomized Trees* (ERT), *Gradient Tree Boosting* (GTB) e *AdaBoost* (AB). Para a predição da média final dos alunos em uma escala contínua, foram utilizados os seguintes algoritmos de regressão: Redes Neurais Profundas (RNP), *Support Vector Regression* (SVR), ERT, RF e GTB.

O processo para construção dos modelos preditivos apresentado na Figura 1.1 foi realizado utilizando duas abordagens:

1. Manual: Foram testadas técnicas de transformação, construção e seleção de atributos, depois testados os algoritmos de AM supracitados e finalmente empregou-se um método para otimização dos modelos preditivos através de ajustes de hiperparâmetros;

2. Automatizado: todo o processo de construção dos modelos preditivos foi realizado automaticamente com o uso de algoritmos genéticos (OLSON et al., 2016).

Destaca-se que em alguns casos o modelos preditivos exportados na segunda abordagem automatizada eram ajustados manualmente a fim de maximizar a acurácia dos modelos.

Por fim, a seguir temos as etapas realizadas neste estudo para atingir o objetivo estabelecido:

1. Seleção de atributos extraídos de sistemas *online* para turmas iniciais de programação, com no base no estado da arte, que se correlacionem com a nota do aluno;

2. Coleta e tratamento da base de dados, a fim de gerar atributos que possam ser usados em algoritmos de AM;

3. Definição de novos atributos que possam se relacionar com o desempenho de alunos de turmas de IPC;

4. Construção de um perfil de programação baseado em atributos dos itens a) e c);

5. Identificação de atributos relevantes no perfil de programação para construção de modelos preditivos;

6. Construção e avaliação de modelos preditivos capazes de realizar a inferência da zona de aprendizagem dos alunos ainda no início da disciplina;

7. Verificação se a precisão dos modelos preditivos vai aumentando no decorrer da disciplina;

8. Análise estatística para comparação dos resultados da construção de modelos preditivos com e sem o emprego de algoritmos evolutivos para encontrar um *pipeline* de aprendizagem de máquina otimizado utilizando o perfil de programação;

9. Comparação do modelo proposto neste estudo com o modelo proposto por Estey e Coady (2016);

10. Análise do uso de redes neurais profundas na realização da predição das médias finais dos alunos com uma estratégia para aumentar os dados de treinamento.

1.6 Contexto Educacional

Para a validação do método de predição proposto, foram coletados dados de 9 turmas de IPC da Universidade Federal do Amazonas, durante o período de 05/06/2016 a 13/09/2016. No total, 486 alunos resolveram 7 listas usando a linguagem de programação Python e tinham permissão de fazer um número ilimitado de tentativas de submissão, desde que atendesse ao prazo máximo estipulado para a resolução da lista de exercícios.

As listas de exercícios sempre precediam uma avaliação, ambas realizadas no Codebench. Cada lista tinha em média 10 questões e as provas tinham 2 questões práticas. Assim, cada lista juntamente com a prova sucessora formava uma sessão para fins desta pesquisa. No total, foram realizadas 7 sessões. Cada sessão durava em média 2 semanas.

As médias finais dos alunos foram calculadas com base em 7 provas parciais, 7 listas de exercícios, 7 exercícios no formato de *quizz* e em uma prova final. As notas das provas tinham um peso alto na média final do aluno, no entanto as notas das listas de exercícios não tinham um peso substancial nessa média. Desta forma, a importância das listas de exercícios estava mais relacionada com a oportunidade do aluno praticar o conteúdo da disciplina e preparar-se para as provas parciais e final.

1.7 Organização do documento

Com relação à organização, este documento encontra-se dividido em 04 (quatro) capítulos.

No Capítulo 2 é apresentada a fundamentação teórica necessária para o entendimento do presente estudo, onde é explanado o contexto do uso de tecnologias em turmas iniciais de programação e como a interação dos alunos com essas ferramentas podem ser úteis para modelar o perfil de programação do aluno e inferir automaticamente a zona de aprendizagem, utilizando técnicas de aprendizagem de máquina.

No Capítulo 3 são apresentados os trabalhos que possuem objetivos e métodos de pesquisa semelhantes ao deste e que foram destacados como o corrente estado da arte. Fez-se ainda uma descrição das publicações, categorizando-as pelo tipo de coleta de dados realizado, isto é, qual a granularidade dos eventos de interação dos alunos com o sistema computacional utilizado no experimento e como foi realizada a montagem do modelo preditivo utilizado. Posteriormente, apresenta-se uma taxonomia para os tipos de eventos utilizados em modelos preditivos e, finalmente, faz-se uma comparação sintetizada dessas pesquisas.

No Capítulo 4 é explanada a arquitetura de trabalho, na qual é apresentada a forma como os alunos interagem com um juiz *online* e como esses dados são persistidos em uma base de dados. Depois disso, é apresentado o método preditivo proposto para inferir a zona de aprendizagem dos alunos baseado em um perfil de programação.

No Capítulo 5 é apresentado o planejamento e projeto dos 6 experimentos conduzidos para atingir os objetivo deste estudo. Neste capítulo, explica-se como foi realizado o pré-processamento do perfil de programação e como foram construídos e avaliados os modelos preditivos.

No Capítulo 6, apresentou-se e discutiu-se sobre os resultados obtidos nos 6 experimentos conduzidos. Também foram respondidas as questões de pesquisa estabelecidas.

No Capítulo 7 são explanadas as considerações finais, as limitações do método e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Entender o motivo pelo qual estudantes de turmas iniciais de programação têm alto índice de retenção é uma questão fundamental para a área de pesquisa educacional de ciência da computação (CSEd) (IHANTOLA et al., 2015). Faz-se necessário a condução de mais estudos sobre a retenção dessas turmas de programação, uma vez que tal problema vem sendo herdado por gerações (LISTER, 2010).

Com efeito, muitos trabalhos vem pesquisando meios para mitigar esse problema, principalmente com o uso da metodologia de aprendizagem híbrida em turmas de programação, que usa sistemas *online* aliados às aulas presenciais (CARVALHO; OLIVEIRA; GADELHA, 2016; IHANTOLA et al., 2015; SINGH; SRIKANT; AGGARWAL, 2016; ESTEY; COADY, 2016; HEINONEN et al., 2014). Neste cenário, surgiu a possibilidade de coletar os dados de interação do estudante com o sistema computacional e usá-los para fazer uma avaliação formativa do discente, não analisando apenas o código enviado, mas o processo por trás dele (BLIKSTEIN et al., 2014). Destaca-se que quando se coleta esses dados, então é possível entender melhor o comportamento de aprendizagem do aluno e criar um perfil de programação desse, que pode ser usado para fazer predições do desempenho do acadêmico.

A busca por inferir o desempenho de alunos de turmas iniciais de programação vem ganhando força nas áreas de *educational data mining* e *learning analytics*, entretanto ainda de forma incipiente (IHANTOLA et al., 2015). Uma abordagem apontada como potencial para tais estimativas é o uso de técnicas *data-driven*, isto é, orientadas aos dados coletados no sistema de programação utilizado pelo discente. Desta forma, é possível identificar alunos em risco, com maus hábitos de estudo e com dificuldade e facilidade na

aprendizagem. Assim, neste capítulo será explicada toda a teoria envolvida e necessária para a compreensão da presente pesquisa.

Mais especificamente, neste capítulo será explorado brevemente como a metodologia de aprendizagem híbrida atua em turmas iniciais de programação, como os dados do processo de desenvolvimento dos alunos são coletados, quais atributos de AM podem ser utilizados para inferir a zona de aprendizagem de alunos de turmas de IPC, baseados nas evidências extraídas dos dados coletadas. Finalmente, serão apresentadas as técnicas de aprendizagem de máquina utilizadas neste estudo para a construção dos modelos preditivos.

2.1 Metodologia Híbrida de Ensino-aprendizagem em Turmas de Programação

A metodologia híbrida de ensino-aprendizagem (*blended learning*) tem como base o uso de ambientes de ensino *online* juntamente com o modelo tradicional de ensino presencial. Desta forma, é possível manter os lados positivos do modelo *face-to-face* (síncrono) e aprimorá-lo com o uso de ferramentas online (assíncrona). Esse paradigma vem sendo usado principalmente em instituições de ensino superior (GARRISON; KANUKA, 2004; CARVALHO; OLIVEIRA; GADELHA, 2016).

Com efeito, muitas dessas instituições pelo mundo vêm adotando essa metodologia em turmas introdutórias de programação, a fim de enfrentar o problema do alto índice de retenção (BOYLE et al., 2003) .Para ilustrar, a metodologia de aprendizagem híbrida proporciona a disponibilização das aulas ministradas, listas de exercícios, fóruns de discussões e outros, para o acesso do estudante com menos limitação temporal e espacial.

Existem sistemas que facilitam ainda mais o processo de ensino e aprendizagem dos alunos e dos professores de turmas de programação: os juízes *online*. Tais sistemas são fundamentados na metodologia híbrida de ensino-aprendizagem (CARVALHO; OLIVEIRA; GADELHA, 2016). Na próxima seção, será explanado um pouco mais sobre os juízes *online*.

2.1.1 Juízes Online

É imprescindível que alunos de turmas de programação resolvam bastantes exercícios práticos, no entanto é impraticável para o professor corrigir acuradamente e em tempo hábil as tarefas de várias turmas (CHEANG et al., 2003; CARVALHO; OLIVEIRA; GADELHA, 2016). Para ilustrar a dificuldade do docente na correção, um código fonte pode passar em alguns casos de teste, mas falhar em outros. Note ainda que um programa pode não estar inteiramente correto, mas deve ser corrigido em uma escala relativa, baseado em muitos fatores. Finalmente, a avaliação humana leva consigo subjetividade o que pode enviesar o parecer do professor por favoritismos e outros motivos (CHEANG et al., 2003). Em função disso, a automatização do processo de correção de código foi estudada por muitos anos (OTERO et al., 2016).

Segundo Otero et al. (2016) o *WebToTeach* foi o primeiro sistema a corrigir códigos automaticamente no contexto de programação. Além disso, uma publicação notável foi a de Cheang et al. (2003), que propôs a comparação da saída do código do estudante com a saída de um programa correto, isto é, usando casos de teste, que pode ser dado conforme definições I, II e III, adaptadas de (CHEANG et al., 2003).

(I) Definição (Caso de Teste): C = (E,S) é uma tupla, onde E é o conjunto de entradas e_i e S é o conjunto de saídas s_i, i = 1..n.

(II) Definição (Programa): Dada uma saída s_i e uma entrada e_i existe uma família de funções $F = f : E \rightarrow S | e_i \rightarrow s_i$, que representa o conjunto de programas que satisfazem as condições de corretude de uma determinada questão q.

(III) Definição (Análise Lógica do Código): Dada uma função f' fornecida pelo aluno, essa irá satisfazer as condições de corretude da questão q, se $f' \in F$.

Assim, os juízes *online* ou sistemas de correção automática de código disponibilizam exercícios para os alunos, os quais submetem o código fonte para o veredicto do sistema (FRANCISCO; JÚNIOR; AMBRÓSIO, 2016). Esse veredicto é feito usando vários casos de teste de entrada e saída cadastrados no sistema, conforme proposto por Cheang et al. (2003). Além disso, o verericto pode ser dado com o uso de um programa referência conhecido como oráculo, o qual é uma solução correta para um dado exercício proposto no juiz online (CHAVES et al., 2013). Destaca-se que esse mecanismo alavanca substancialmente o processo de ensino e aprendizagem de programação, uma vez que é atendido o requisito de realização de vários exercícios práticos necessários para a construção do

saber do aluno e, além do mais, não sobrecarrega o professor com a correção.

Em geral, os juízes *online* fornecem um feedback imediato para os alunos, normalmente binário, isto é, se a resposta está certa ou errada. Por um lado, isso é considerado muito positivo, pois possibilita uma avaliação contínua que é extremamente importante para manter o aluno motivado (CARVALHO; OLIVEIRA; GADELHA, 2016). Mas por outro, muitos estudos afirmam que os veredictos desses sistemas precisam ser aprimorados, isto é, o feedback precisa ser mais adequado para as individualidades dos estudantes e dos erros cometidos (IHANTOLA et al., 2015). Pensando nisso, muitos autores vêm conduzindo pesquisas nessa questão, como Piech et al. (2015), que gera dicas automaticamente baseado na maneira que o aluno resolve os problemas e Souza, Felizardo e Barbosa (2016), que realizou uma RSL sobre os pontos fortes e fracos dessas ferramentas.

De uma forma geral, as arquiteturas lógicas dos ambientes *online* utilizados em turmas de programação possuem, além do juiz *online*, mais recursos conforme Figura 2.1. Assim sendo, estudantes podem, em geral, desenvolver, depurar e testar o seu código em um ambiente de desenvolvimento integrado ao juiz *online* e todo o seu processo de desenvolvimento pode ser registrado em diferentes níveis de granularidade, desde o mais detalhado como o registro de cada tecla digitada, ao menos, que é gravado apenas as submissões dos alunos (IHANTOLA et al., 2015).

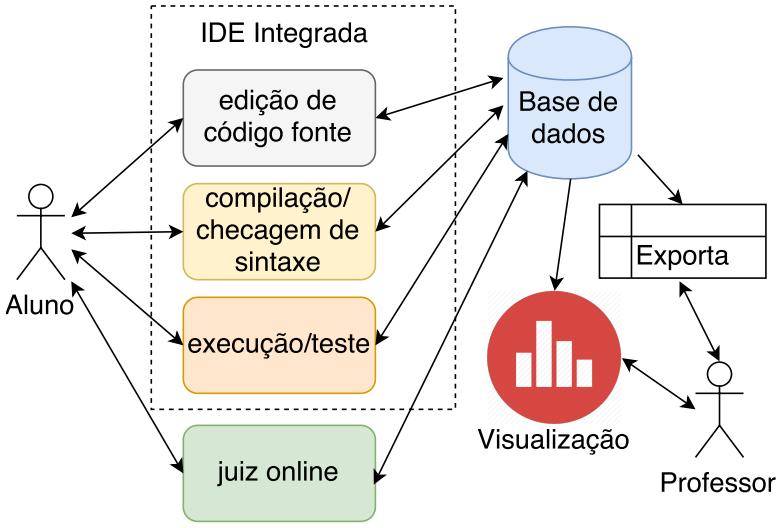


Figura 2.1: Arquitetura lógica comum dos ambientes online voltado para turmas de programação. Adaptado de (IHANTOLA et al., 2015).

Assim, o intuito principal deste trabalho é analisar esses *logs* gerados durante as tentativas dos alunos de resolver as questões de programação propostas pelo professor em uma IDE integrada ao juiz online, a fim de investigar a relação desses dados com o desempe-

nho deles nas avaliações e na média final. Mais especificamente, o objetivo é usar esses dados, depois de tratados, como atributos de algoritmos de aprendizagem de máquina, a fim de inferir a zona de aprendizagem desses alunos com antecedência. Dessa forma, nas próximas seções serão explicadas as técnicas de aprendizagem de máquina, mineração de dados e análise estatística utilizadas no presente estudo.

2.2 Aprendizagem de Máquina e Mineração de Dados

Métodos de aprendizagem de máquina (AM) e mineração de dados (MD) podem ser empregados para fazer predições, estimativas, análises de atributos relevantes, etc. Assim, destaca-se que AM, em uma definição mais clássica, é um campo de estudo que dá aos computadores a habilidade de aprender, sem ser explicitamente programado, isto é, os programadores criam algoritmos para ensinar à máquina a aprender, usando fundamentos matemáticos (SAMUEL, 1959).

Mitchell et al. (1997) explica que um programa de computador aprende através de uma experiência E, com respeito a algumas classes de tarefas T e uma medida de desempenho P, se o desempenho do programa na tarefa T, conforme medido em P, pode ser melhorado com a experiência E. Para ilustrar, uma das tarefas que será realizada nesta pesquisa é estimar a nota do aluno baseada na interação (nas evidências) dele com um juiz *online*, assim temos:

E = a experiências de observar as interações de vários alunos com um juiz online e qual a média final deles;

T = estimar a média do aluno baseado na sua interação com o juiz *online*;

P = cálculo da média aritmética da soma das diferenças ao quadrado entre as notas estimadas e as notas que o aluno tirou, conhecido como Método dos Mínimos Quadrados (MMQ).

Além disso, qualquer problema de AM pode ser classificado em aprendizagem supervisionada e não supervisionada. Em aprendizagem supervisionada é dada uma base de dados e nela, as respostas já são conhecidas. Neste trabalho foi empregado aprendizagem de máquina supervisionada, o qual pode ser categorizada em problemas de classificação e regressão. Em classificação, as saídas (variáveis dependentes, classes ou *target*) são discretas, enquanto que em regressão os resultados são contínuos, isto é, dada uma entrada

existe uma função de saída que gera valores contínuos (MITCHELL et al., 1997). Para exemplificar, ao tentarmos estimar as médias finais de alunos estamos lidando com uma regressão, uma vez que a nota é um número real e, portanto, contínuo. Por outro lado, pode-se facilmente transformar esse problema em classificação, discretizando os valores das notas dos alunos em duas categorias: os estudantes que tiraram abaixo de 5 estão no grupo de risco e os que tiraram notas maiores ou iguais a 5 na categoria fora de risco.

Mais formalmente, segundo Russell e Norvig (2005) a tarefa de aprendizagem supervisionada é realizada utilizando uma base de treino com N pares de amostras de entrada e saída

$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$

onde y_j é a **classe** ou variável dependente ao qual tentamos estimar com uso de uma função $h(x_j)$ chamada de **hipótese**, que é uma aproximação de uma função desconhecida $f(x_j)$ que gera y_j. Assim, a tarefa de aprendizagem é na verdade uma busca no espaço de hipóteses por aquela que terá um bom desempenho em uma base de teste (base esta diferente da utilizada para treino). Para ilustrar, na classificação o intuito é que no treino seja criada uma hipótese que gere uma superfície de decisão na qual as classes possam ser segregadas corretamente no teste.

Para um maior contextualização desta seção com o objeto de estudo deste trabalho, observe que à medida que as evidências das interações dos alunos com um juiz *online* estão acessíveis, serão utilizados algoritmos de aprendizagem de máquina supervisionada e técnicas de MD para que os dados sejam tratados e, assim, sejam selecionadas quais evidências são mais relevantes para fazer a predição da zona de aprendizagem do aluno.

Perceba que existe um *pipeline* para a construção e otimização do modelos preditores, isto é, uma série de etapas que são o pré-processamento de atributos, a escolha do algoritmos de AM e um ajuste dos hiperparâmetros do algoritmo. Assim, nas próximas seções será explicada uma breve intuição dos algoritmos que foram empregados nos experimentos realizados neste estudo para a construção e otimização desses *pipelines* de AM.

2.2.1 Pré-processamento de Atributos

Em aprendizagem de máquina uma das tarefas essenciais é escolher e combinar características (evidências ou atributos) que descrevem um conceito, a fim de fazer predições.

Assim, uma das etapas iniciais neste processo normalmente é realizar a transformação dos atributos. Para ilustrar, se um atributo estiver a uma ordem de magnitude muito maior que os outros, ele pode dominar a função objetivo do estimador e, dessa forma, torná-lo incapaz de aprender com os outros atributos (PEDREGOSA et al., 2011). Além disso, a transformação dos dados é um requerimento comum para muitos algoritmos de aprendizagem de máquina como as redes neurais, onde é necessário realizar um normalização antes de treinar o modelo. Neste trabalho, os dados foram escalados, atribuindo zero para o valor médio de cada atributo e os demais valores receberam um valor referente à quantidade de desvios padrão que eles estavam dessa média. Esse valor é também conhecido como *z-score* na estatística (CROCKER; ALGINA, 1986). O *z-score* é calculado conforme segue:

$$z\text{-}score = \frac{\overline{x} - x_i}{\sigma(x)},$$

onde x_i representa cada valor do atributo, $\overline{x}$ representa a média do atributo e o $\sigma(X)$ o desvio padrão do atributo.

Além dessa etapa inicial, um processo que foi realizado em alguns experimentos foi a construção de novos atributos a partir dos atributos existentes. Finalmente, a seleção de atributos relevantes e eliminação dos irrelevantes é um dos problemas mais centrais em aprendizagem de máquina, uma vez que os modelos preditivos devem usar essas evidências para treinar e serem testados até chegarem a um nível desejado de acurácia (BLUM; LANGLEY, 1997). Abaixo serão explicados brevemente os algoritmos de seleção de atributos utilizados nesta pesquisa, que são os algoritmos *SelectionKBest*, *Recursive Feature Elimination* (RFE) e o *BestFirst*.

- *SelectionKBest* seleciona os k atributos mais relevantes e remove o restante. Para tanto, o algoritmo usa uma função objetivo que gera uma pontuação para cada atributo (PEDREGOSA et al., 2011). Para ilustrar, nesta pesquisa foi utilizada a função objetivo *fclassif* para os problemas de classificação. Nessa função, é analisada a variância (ANOVA) dos atributos em relação à classe, utilizando *F-tests* (LOMAX; HAHS-VAUGHN, 2013).

- *Recursive Feature Elimination* (RFE) é um método de seleção de atributos chamado de *wrapper*, porque ele identifica a relevância dos atributos baseado no processo de aprendizagem de um estimador, em outras palavras, o método de aprendizagem de

máquina é empacotado no procedimento de seleção. Mais especificamente, o RFE elimina recursivamente atributos menos promissores e constrói modelos preditivos com os atributos que restaram. Assim, o método utilizada a acurácia do modelo para identificar quais são os atributos mais relevantes da base (PEDREGOSA et al., 2011).

- *BestFirst* - este método de seleção de atributos também é classificado como *wrapper*. Ele constrói subconjuntos a partir do conjunto de atributos e os avalia usando uma função objetivo. Observe como não é possível avaliar todos os subconjuntos possíveis, já que o número de subconjuntos cresce exponencialmente. Assim, o método usa uma estratégia gulosa que emprega as estratégias *forward selection* e *backward elimination*, onde no primeiro caso, adiciona-se um atributo ao subconjunto que está sendo avaliado e no segundo, remove-se um atributo. Perceba que no *forward selection* a ideia é começar com o subconjunto vazio, enquanto que no *backward elimination*, inicia-se com todos os atributos (XU; YAN; CHANG, 1988). O *BestFirst* mantém uma lista de todos os subconjuntos avaliados, ordenada pelo desempenho observado pelo algoritmo avaliador. Assim, o algoritmo consegue revisitar um subconjunto avaliado anteriormente e, consequentemente, se for dado tempo suficiente para o BestFirst visitar todo o espaço de busca, então ele encontrará o ótimo global, do contrário ele encontrará o ótimo local. No presente estudo, o avaliador de subconjuntos aplicado em conjunto com o *BestFirst* foi o *CfsSubsetEval*, o qual objetiva identificar os subconjuntos que são mais correlacionados com a classe e menos correlacionados entre si (WITTEN et al., 2016).

2.2.2 Algoritmos de Classificação e Regressão Utilizados

Nesta subseção, serão explicados os métodos de classificação e regressão utilizados nos experimentos que foram conduzidos. Tais métodos não serão expostos em detalhes, entretanto será explicitada uma intuição para o entendimento por trás de cada algoritmo. Haverá uma ênfase maior para o algoritmo árvore de decisão, uma vez que foram os métodos *ensembles* baseados em árvores de decisão que obtiveram os melhores resultados nos experimentos conduzidos.

2.2.2.1 Árvore de Decisão

Árvore de decisão é um algoritmo determinístico utilizado para classificação ou regressão. Nele, toma-se como entrada um vetor de atributos e retorna-se um valor de saída único (QUINLAN, 1986). A árvore é construída recursivamente através de uma sequência de divisões, onde cada nodo representa uma questão e as folhas representam a decisão, isto é, as estimativas.

Para construir uma árvore de decisão consistente, o primeiro passo é identificar qual é o atributo mais promissor no que tange a predição, isto é, qual atributo possui maior relação com a classe, a fim de defini-lo como nodo raiz da árvore. Uma forma de realizar essa tarefa é calculando a entropia de cada atributo, isto é, mensurar a impureza ou incerteza de cada atributo. Assim, pode-se verificar a diferença entre a entropia antes e depois da divisão em uma subárvore, a fim de calcular o ganho de informação de cada atributo em uma subamostra da base de treino (QUINLAN, 1986). A entropia é um valor entre [0-1] e quanto mais próximo de 1, maior a impureza (SHANNON, 2001). Ela é calculada conforme segue:

$$E(A) = -\sum_{k=1}^{n} P(v_k)\log_2 P(v_k),$$

sendo n o número de valores únicos para um dado Atributo A e $p(v_k)$ a probabilidade de um dos possíveis eventos do atributos A ocorrer.

Assim, o algoritmo calcula a entropia da classe e a entropia da classe dado cada atributo $A_1, A_2, ..., A_n$. Para ilustrar, em um caso de classificação binária, um atributo A_i com j valores distintos, estratifica o base de treino B em subconjuntos $B_1, ..., B_j$, onde cada subconjunto B_k possui p_k exemplos positivos e n_k exemplos negativos (RUSSELL; NORVIG, 2005). Dessa forma, para calcular a entropia da classe C, dado A_i, temos:

$$E(C,A) = -\sum_{i}(P(a_i)\sum_{j}P(c_j/a_i)log_2(P(c_j/a_i))),$$

onde $P(a_i)$ é a probabilidade a priori para todos os valores únicos de A e $P(c_j/a_i)$ a probabilidade a posteriori (probabilidade da ocorrência do evento c_j dado que ocorreu a_i).

O cálculo do ganho de informação (SHANNON, 2001) de cada atributo é dado pela diferença entre a entropia da classe, $E(C)$, pela entropia da classe com a divisão no atributo, $E(C,A)$ (entropia da classe dado o atributo A):

$$GanhoInfo(A_i) = E(C) - E(C,A_i).$$

Perceba que a ideia principal é fazer uma divisão mais pura possível, a fim de ter um ganho de informação máximo. Para ilustrar, um caso ideal é que na ocorrência de um evento v no atributo A_1 só ocorra um mesmo evento na classe C. Nesse caso, haverá uma divisão pura, isto é, a entropia do atributo é zero, e o ganho de informação é total. Assim, quanto mais o valor da entropia da classe, após a divisão no atributo, se aproxima de zero, maior é o ganho de informação. Isso é verdade porque como a entropia é uma medida de incerteza ou impureza, se diminuí-la, então tendemos a criar uma divisão mais pura na árvore.

Na Figura 2.2 podemos ver o pseudocódigo do algoritmo Árvore de decisão, que, como vimos, verifica qual o atributo mais promissor utilizando uma função objetivo (nesse caso explicamos a entropia, mas podem ser usadas outras como o coeficiente de gini (DORFMAN, 1979)). Após isso, esse atributo é utilizado como nodo raiz da árvore e, assim, uma parte dos dados mutuamente exclusiva vai para cada um dos filhos desse nodo. Esse processo é realizado recursivamente para cada um dos nodos filhos até se atingir uma divisão pura ou algum critério de parada ser atingido (RUSSELL; NORVIG, 2005). Existem vários critérios que podem causar a parada da divisão como, por exemplo, uma subamostra só pode ser dividida se tiver mais que 10 instâncias a fim de evitar *over-fitting*. Em geral, a tarefa de ajustar esses critérios é uma parte importante em AM, pois esses ajustes de hiperparâmetros podem otimizar a precisão do modelo preditor, conforme será explicado na seção 2.2.3.

```
função APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO(exemplos, atributos, exemplos-pais)
 retorna uma árvore de decisão
   se exemplos é vazio então retornar VALOR-DA-MAIORIA (exemplos_pais)
   senão se todos os exemplos têm a mesma classificação então retornar a classificação
   senão se atributos é vazio então retornar VALOR-DA-MAIORIA(exemplos)
   senão
      A ← argmax a ∈ atributes IMPORTÂNCIA (a, exemplos)
      árvore ← uma nova árvore de decisão com teste de raiz A
      para cada valor v_k de A faça
         exs ← {e : e ∈ exemplos e e.A = v_k}
         subárvore ← APRENDIZAGEM-EM-ÁRVORE-DE-DECISÃO (exs, atributos — A,
exemplos)
         adicionar uma ramificação à árvore com rótulo (A = v_k) e subárvore subárvore
      retornar árvore
```

Figura 2.2: Pseudocódigo do algoritmo de árvore de decisão. A função IMPORTÂNCIA pode ser calculada baseada na entropia, conforme descrito na seção 2.2.2.1. Já a função VALOR-MAIORIA seleciona a valor mais comum na classe e no caso de empate ela escolhe de modo aleatório (RUSSELL; NORVIG, 2005).

2.2.2.2 Random Forest

Floresta Aleatória ou *Random Forest* (RF) é um método *ensemble*, isto é, um conjunto de estimadores. Os *ensembles* partem do princípio de que a opinião de um grupo é mais forte que a de um indivíduo, especialmente quando os membros do grupo tem seus próprios vieses (DAUMÉ, 2012). Essa abordagem usa um conjunto de preditores ”simples”, a fim de gerar uma estimativa ”robusta” baseada na composições das estimativas simples. Para ilustrar, Breiman (1996) propôs um método chamado *Bagging* que é um *ensemble* baseado em votação. Assim, dada um conjunto de M estimadores $h_1...h_m$ e uma tarefa de classificação para saber se uma dada instância pertence a um grupo A ou B e que temos um exemplo X para teste, então usa-se os estimadores $h_1(X)...$, $h_m(X)$ e verifica-se a quantidade de As e Bs produzidas por tais preditores. Caso tenham sido estimados mais As, então o *ensemble* estimará A e do contrário, B.

Quando usado para classificação, o algoritmo *Random Forest* (BREIMAN, 2001) emprega o mesmo mecanismo de votação do algoritmo *Bagging*, destacando-se que todos os estimadores ”simples” do *Random Forest* são árvores de decisões, por isso o nome floresta. Esses estimadores simples, no entanto, são algoritmos determinísticos, o que significa que se esse algoritmo for treinado várias vezes com a mesma base de treino, então ele produzirá a mesma árvore de decisão. Perceba que os *ensembles* precisam de variação dentro dos seus estimadores constituintes, a fim de que todos os preditores não comentam o mesmo erro. Nesse sentido, uma forma de minimizar esse problema é reamostrar a base de dados utilizando técnicas provenientes da estatística.

Uma estratégia amplamente adotada é o *bootstrap resampling* (EFRON; TIBSHIRANI, 1994), que a partir de um conjunto de treino N que segue uma distribuição D, seleciona aleatoriamente instâncias da base original, com ou sem reposição, a fim de gerar vários subconjuntos. Essa abordagem permite gerar subconjuntos de treino $S_1...S_n$ que seguem a mesma distribuição D que o conjunto de treino N, já que as instâncias de $S_1...S_n$ pertencem ao conjunto N (BREIMAN, 1996). Destarte, tais subamostras podem ser usadas como base de treino para a construção das árvores da floresta.

Breiman (2001) explica que para aumentar ainda mais a variação entre as árvores da floresta, o autor propôs que dado M atributos de uma base de dados N, uma constante m menor que M seria especificada, de tal modo que em cada nodo, m atributos seriam escolhidos aleatoriamente a partir de M em cada chamada recursiva para a ramificação da

árvore. Assim, o melhor atributo em m é empregado para a divisão do nodo. A Figura 2.3 apresenta o arcabouço do funcionamento do algoritmo *Random Forest* na tarefa de classificação. Perceba que para realizar a regressão com o *Random Forest* uma estratégia é calcular a média dos valores estimados pelas árvores de regressão.

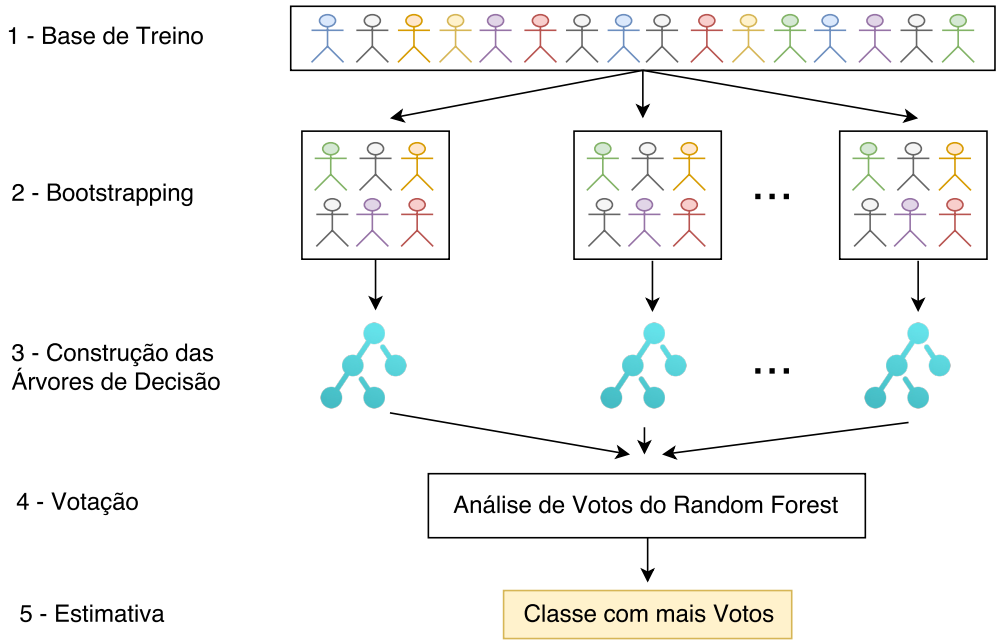


Figura 2.3: Arcabouço do funcionamento do algoritmo *Random Forest* na tarefa de classificação.

2.2.2.3 Extremely Randomized Trees

Assim como o *Random Forest*, o algoritmo *Extremely Randomized Trees* (ERT) é um método *ensemble* que utiliza como classificadores bases árvores de decisão. A principal diferença entre os dois é que o ERT deixa ainda mais aleatória a escolha dos atributos e o local de divisão dos nodos das árvores constituintes. Nos casos extremos, o ERT constrói suas árvores completamente aleatórias, isto é, com estruturas independentes (GEURTS; ERNST; WEHENKEL, 2006).

Resumidamente, as principais diferenças entre o ERT e o RM, são:

- o ERT usa a base de treino inteira para a construção de todas as árvores de decisão constituintes, diferentemente do RF que usa subamostras com *bootstrapping* na base de treino para a construção das árvores (GEURTS; ERNST; WEHENKEL, 2006).

- as ramificações são realizadas por k tentativas de divisão completamente aleatórias, sendo cada uma determinada por uma seleção aleatória de um dos atributos da base (GEURTS; ERNST; WEHENKEL, 2006).

Observe que no ERT, as árvores são mais diversificadas, o que aumenta a variança da estimativa das classes bases. Dessa forma, os erros cometidos pelas árvores tendem a ser menos correlacionados que na RF. Em alguns testes realizados por Geurts, Ernst e Wehenkel (2006) com *benchmarking* públicos, demonstrou-se que a ERT compete em pé de igualdade com o RF.

2.2.2.4 AdaBoosting

Boosting (FREUND, 1995) é um método *ensemble* que emprega classificadores bases que se complementam iterativamente. Em outras palavras, o boosting é utilizado para melhorar o desempenho de um algoritmo de aprendizagem de máquina simples (FREUND; SCHAPIRE et al., 1996). Assim como o *bagging* (BREIMAN, 1996), ele também usa votação para classificação e média para regressão e combina modelos do mesmo tipo, ou seja, é possível criar um conjunto de árvore de decisão ou um conjunto de SVM, mas não um conjunto híbrido. A principal diferença é que nos *ensembles* que usam o método *boosting*, os novos modelos construídos são influenciados pelo desempenho dos modelos construídos anteriormente, enquanto que no *bagging* os modelos são construídos de forma independente (WITTEN et al., 2016). Essa estratégia do *boosting* é implementada através do uso de pesos para as instâncias e para os votos dos modelos.

Existem muitas implementações do método *boosting*. Uma delas é o AdaBoost.M1 (FREUND; SCHAPIRE et al., 1996), que na primeira iteração inicializa o peso de todos as instâncias igualmente, chama o algoritmo base para realizar a estimativa, verifica o valor estimado e então recalcula os pesos. Os pesos das instâncias corretamente classificadas são diminuídos, enquanto que os pesos das incorretamente classificadas são aumentados. Note que depois da primeira iteração, existem um conjunto de amostras "fácil" de classificar (com pesos baixos) e outro conjunto "difícil" (com pesos altos). Assim, nas próximas iterações o AdaBoost.M1 se concentra em classificar corretamente as amostras com maiores pesos, isto é, as que foram incorretamente classificadas (WITTEN et al., 2016).

Observe que a cada iteração um novo modelo é criado. Ao final, são criados k mode-

los, onde os pesos dos modelos são atribuídos com base na precisão atingida no decorrer das iterações. Assim, o AdaBoost.M1 realizada a predição fazendo uma votação, na qual alguns modelos possuem maior poder de voto (RUSSELL; NORVIG, 2005).

2.2.2.5 Gradient Tree Boosting

Gradient Tree Boosting (GTB) é um método de *ensemble* baseado em árvores de regressão (FRIEDMAN, 2001). Ele utiliza a técnica *boosting* (FREUND, 1995), ao invés de *bagging* (BREIMAN, 1996). Em outras palavras, no GTB as árvores constituintes não possuem um mesmo peso na votação, isto é, a intenção principal do algoritmo é encontrar a combinação ótima de árvores com pesos. Este algoritmo pode ser utilizado para regressão e classificação.

O GTB cria uma árvore de regressão simples e depois usa uma variação do gradiente descendente para ir otimizando as árvores nas iterações com o uso de uma função de custo. Formalmente a decisão do GTB é dada pela soma dos estimativas das árvores (FREUND, 1995):

$$GTB_{m-1}(x) = a_1(x) + a_2(x) + ... + a_{m-1}(x),$$

assim a próxima árvore tenta reduzir o erro entre o valor previsto e o valor real através da estimativa do residual. Para ilustrar, idealmente o GTB estimaria o valor exato:

$$GTB_{m-1} + a_m(x) = y.$$

O que pode ser reescrito da seguinte forma:

$$a_m(x) = y - GTB_{m-1},$$

logo o objetivo é estimar de forma ótima o residual, a fim de encontrar o erro mínimo. Assim, no próximo modelo teremos um erro menor, caso a operação de otimização tenha sido bem sucedida:

$$GTB_m = GTB_{m-1} + a_m(x).$$

Perceba que isso corrobora com o princípio do método *boosting*, pois o GTB_m consegue melhorar o desempenho do seu antecessor GTB_{m-1} (HASTIE; FRIEDMAN; TIBSHIRANI, 2001).

Freund (1995) ainda explica que este método também pode ser aplicado para proble-

mas de classificação binária através do uso da verossimilhança binomial negativa, conforme seque:

$log(1+e^{-2y\bar{y}})$,

onde y é o valor real e $\bar{y}$ é o valor estimado.

2.2.2.6 Support Vector Machine

O *Support Vector Machine* (SVM) é um algoritmo que cria superfícies de decisão para segregar instâncias de classes diferentes. Em outras palavras, neste método é criado um hiperplano ótimo que maximize a margem (CORTES; VAPNIK, 1995), que é a distância entre os *support vectors* de classes distintas. Para criar uma superfície de decisão em problemas que não podem ser separados linearmente, o SVM utiliza funções *kernel* que podem realizar transformações lineares no vetor de atributos a fim de criar dimensões maiores, nas quais as classes podem ser separadas por um hiperplano $\vec{w}$ (VAPNIK, 2013).

Para ilustrar o funcionamento do algoritmo em um caso em que os dados podem ser separados linearmente, observe a Figura 2.4. A linha contínua representa a superfície de decisão (o hiperplano que nesse caso é uma reta) que foi criada com o SVM com o *Kernel* linear utilizando a biblioteca *scikit-learn* (PEDREGOSA et al., 2011). A margem é a distância entre as linhas pontilhadas que separam as classes azuis das classes marrons. Os pontos que interceptam as linhas pontilhadas que estão nas coordenadas [(7.27 -4.84), (5.95 -6.82), (7.89 -7.41)] são chamados de *support vectors* e são eles que inspiraram o nome do algoritmo.

Por outro lado, existem casos que não podem ser separados linearmente, como pode ser visto na Figura 2.5 (a), onde também foi aplicado o SVM com *Kernel* linear de forma ineficaz. Nesses casos, uma alternativa encontrada por Cortes e Vapnik (1995) foi o truque do *Kernel* que realiza uma transformação linear nos dados, isto é, os eleva para dimensões maiores que podem ser separadas por um hiperplano, como pode ser visto na Figura 2.5 (b), onde os dados podem separados por um plano, por exemplo, em $z = 0.6$. No que nesse caso, a transformação linear aplicada foi $T : \mathbb{R}^2 \to \mathbb{R}^3$, onde $T([x_1, x_2]) = [x_1, x_2, x_1^2 + x_2^2]$.

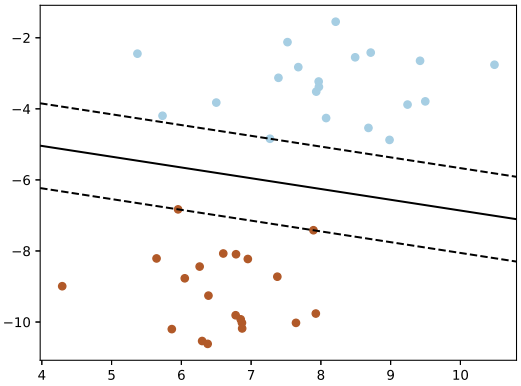
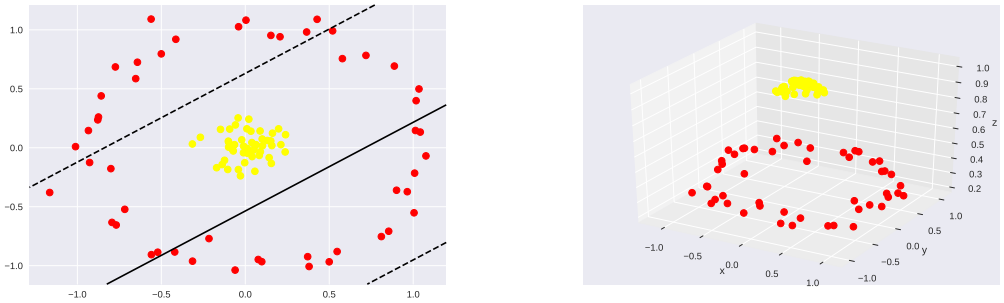


Figura 2.4: Exemplo de classes separadas linearmente pelo SVM. A distância entre os linhas pontilhadas representa a margem e a linha contínua é a superfície de decisão representada pelo hiperplano, que nesse caso é uma reta. (PEDREGOSA et al., 2011)



(a) Dados não-linearmente separáveis.

(b) Após o truque do kernel.

Figura 2.5: Exemplo de Truque do Kernel do SVM.

2.2.2.7 k-Nearest Neighbors

O *k-Nearest Neighbors* (KNN) é um algoritmo não paramétrico, que realiza a classificação de uma amostra não rotulada com base nas amostras rotuladas mais semelhantes a ela (COVER; HART, 1967). A semelhança é calculada no espaço euclidiano, isto é, os dados de treino são colocados no $\mathbb{R}^n$ e à medida que faz-se uma consulta para verificar qual a classe de x, então coloca-se o x nesse espaço e verifica quais são os k vizinhos mais próximos de x, utilizando por exemplo a distância euclidiana, conforme segue:

$$L^p(x,y) = \sqrt{\sum_{i}^{n}(x_i - y_i)^2}.$$

Assim, a classe de x será definida pela o voto dos k vizinhos mais próximos. Para ilustrar, em uma classificação binária cujas classes são A e B, caso o valor de k seja 5 e

dos 5 vizinhos 4 pertencem a classe B, então x será classificado como B.

Note que para fazer regressão, pode-se tirar a média dos valores estimados pelos vizinhos, ou ainda fazer uma regressão linear sobre os vizinhos. Observe ainda que valores muito baixos de k podem levar ao *underfitting* e valores muito altos podem acarretar em *overfitting*, assim o melhor valor para a constante k varia de acordo com a base (RUSSELL; NORVIG, 2005).

2.2.2.8 Redes Neurais Profundas

Uma rede neural padrão consiste em vários nodos chamados de neurônios, os quais se conectam cada um produzindo uma sequência de ativações de valores reais (SCHMIDHUBER, 2015). Uma rede neural multicamada possui uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída, conforme ilustrado na Figura 2.6. Note que cada aresta leva consigo um peso e os nodos possuem um viés.

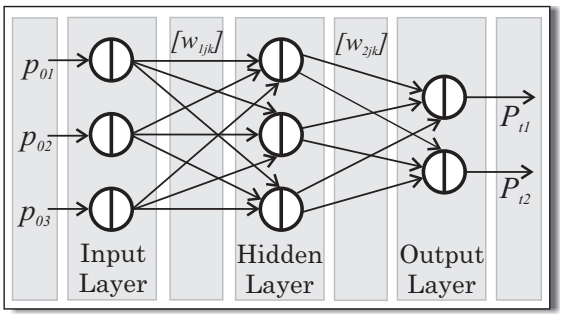


Figura 2.6: Exemplo de uma rede neural com uma camada de entrada, uma camada oculta e uma camada de saída. Cada aresta leva consigo um peso e os nodos possuem um viés.

Os neurônios da camada de entrada são ativados através de sensores que percebem o ambiente, enquanto que os outros neurônios são ativados através de conexões ponderadas de neurônios previamente ativos (GOODFELLOW; BENGIO; COURVILLE, 2016). Assim, o processo de aprendizagem desses algoritmos é realizado com ajustes dos pesos dos nodos, para que a rede neural execute um comportamento desejado. Entretanto, dependendo do problema e de como os neurônios estão conectados, esse comportamento desejado pode ser caro computacionalmente. Resumidamente, Redes Neurais Profundas (RNP) tratam de como atribuir esses pesos de forma precisa podendo utilizar muitas camadas ocultas (SCHMIDHUBER, 2015).

A arquitetura de RNP utilizada neste trabalho foi uma *Feedforward Neural Network* (FNN) com múltiplas camadas ocultas entre as camadas de entrada e saída. Segundo

Bengio et al. (2009) FNN são tipicamente redes neurais *feed forward*, isto é, os dados fluem da camada de entrada para as camada de saída sem retornar, em outras palavras, grafos acíclicos. Tais redes podem modelar relacionamentos não lineares complexos e podem gerar modelos de composição onde o objeto é expresso como uma composição em camadas primitivas. As camadas extras permitem a composição dos atributos das camadas mais baixas, potencialmente modelando dados complexos com um número limitado de nodos (BENGIO et al., 2009).

As RNPs foram usadas apenas na tarefa de regressão no presente trabalho. A ideia era investigar se ao aumentar o número de dados, o método RNP superaria os métodos de aprendizagem de máquina tradicional supracitados.

2.2.3 Ajuste de Hiperparâmetros

Algoritmos de AM possuem vários hiperparâmetros que podem ser ajustados a fim de que o modelo preditivo seja melhor encaixado aos dados. Por exemplo, uma árvore de decisão pode verificar a importância dos atributos usando a entropia ou o *gini-index* (DORFMAN, 1979). Além disso, pode ser preestabelecido, por exemplo, que a árvore deve ter no mínimo 15 amostras da base de treino nas folhas, a fim de que o modelo tenha maior poder de generalização.

Entretanto, essa tarefa de otimização é exploratória e o número de possíveis ajustes de hiperparâmetros cresce exponencialmente. Por ilustrar, imagine o algoritmo *Random Forest* com as seguintes possibilidades de construção de modelos preditivos:

- numero de árvores de decisões: [100, 200, 300, 400, 500];
- máximo de atributos por árvores: [5, 10, 15, 20];
- profundidade máxima das árvores: [3, 4, 5, 6];
- mínimo de amostras para realizar uma nova ramificação na árvore: [10, 20, 30, 40, 50];
- critério de importância de atributos: [entropia, gini].

Observe que se todos os hiperparâmetros supracitados forem testados, seriam construídas 800 florestas aleatórias, sendo que cada floresta possui no mínimo 100 e no máximo 500 árvores de decisão. Além disso, existem vários métodos de pré-processamento

de atributos que podem ser aplicados aos dados antes mesmo da construção do modelo preditivo. Para exemplificar, imagine que antes da construção das florestas aleatórias, fosse realizado um dos seguintes tipos de transformação dos dados: normalização com *MinMax* (PEDREGOSA et al., 2011), normalização com *z-score*, sem normalização. E mais, se forem testados três métodos de seleção de atributos cada um com 3 variações, seria multiplicadas 27 possibilidades as 800 existentes, ou seja, seriam construídas 21600 florestas aleatórias. Vale ressaltar que um método de validação amplamente utilizado é a validação cruzada (será explicada na seção 2.2.4) e nela são realizadas *k* testes em cada modelo, isto é, no total seriam realizadas k vezes 21600 validações.

Existe um método chamado busca em grade, *GridSearch* em inglês, que testa todas essas possibilidades e retorna o melhor *pipeline* de AM. Por outro lado, existem pesquisas que demonstram que a busca aleatória (BERGSTRA; BENGIO, 2012), Random Search em inglês, de um número limitado de possíveis *pipelines* de AM é mais eficiente e pode ter propriedades favoráveis quando comparado com a busca em grade. No algoritmo *Random Search,* determina-se um número *k* de iterações para a construção dos *pipelines*, como resultado o *Random Search* retorna o melhor *pipeline* encontrado.

Observe que quando o espaço de busca é pequeno como, por exemplo, encontrar o valor mais promissor de *k* para o algoritmo KNN, então o GridSearch é uma escolha viável. Dessa forma, neste estudo foram usados os dois métodos de otimização de hiperparâmetro, sendo a busca em grade em casos em que havia poucas possibilidades de otimização para explorar e a busca aleatória, caso contrário.

2.2.4 Métricas de Avaliação

Para verificar a precisão de algoritmos de classificação pode-se utilizar a acurácia, que calcula a razão entre o número de itens corretamente estimados pelo total de itens. Além disso, pode-se utilizar a validação cruzada (k-fold) para estimar a acurácia final. Este método divide aleatoriamente a amostra em k subconjuntos mutuamente exclusivos de tamanhos aproximadamente iguais. O classificador é treinado e testado k vezes, assim o modelo é validado em um conjunto de dados diferente de sua construção (MITCHELL et al., 1997).

Além da acurácia, existem outras métricas como o *recall* e *precision*, que são mais adequadas para bases desbalanceadas. Tais métricas são baseadas nas quantidades de ver-

dadeiros positivos (VP), verdadeiros negativos (VN), falsos positivos (FP) e os falsos negativos (FN), e são calculadas conforme segue: *recall* positivo: VP/(VP+FN); *recall* negativo: VN/(VN+FP); *precision* positivo: VP/(VP+FP) e *precision* negativo: VN/(VN+FN) (PEDREGOSA et al., 2011). O *recall* positivo também é conhecido como *True positive Rate* (TPR), enquanto que o *recall* negativo é analogamente conhecido como *True Negative Rate* (TNR).

Outra métrica que é constantemente utilizada na área de AM e MD é o gráfico *Receiver Operating characteristics* (ROC), ou curva ROC em português, uma vez que ela é útil para analisar problemas de AM onde as classes estão desbalanceadas (distribuição caudal de classes) e diferentes valores de cálculo de taxas de erro, isto é, *False Positivo Rate* (FPR). Mesmo em tarefas de classificação, alguns modelos podem produzir uma saída contínua (por exemplo, uma estimativa da probabilidade de uma instância ser associada a uma classe). Assim, diferentes limiares (*thresholds*) probabilísticos podem ser aplicados para realizar a estimativa. Para ilustrar, imagine um problema de classificação binária, em que dada uma instância x_i, o classificador estimou que a probabilidade da instância ser positiva é de 60%. Se o limiar (*threshold*) for de 50% (mais comum), então o classificador irá rotular a instância como positiva. Entretanto se o limiar for 70%, então a instância será rotulada como negativa (FAWCETT, 2006).

As curvas ROC são gráficos bidimensionais, onde no eixo y são plotados os valores do TPR, enquanto que no eixo x são plotados os valores de FPR (FP/(FP+VN)) para diferentes limiares (Figura 2.7). A coordenada (0,0) representa que não houve estimativas positivas, enquanto que (1,1) representa uma emissão incondicional de classificações positivas. O ponto (0,1) representa uma classificação perfeita. Observe que a curva ROC ideal é aquela que conglomera pontos no noroeste do gráfico (TPR alto ou FPR baixo, ou ambos). Observe ainda que se a área sob a curva estiver muito próxima do eixo X, significa que o classificador está sendo ”conservador”, isto é, ele estima positivo apenas quando tem evidências fortes, o que acarreta em uma baixo FPR e uma baixo TPR. Por outro lado se a curva tende ao nordeste do gráfico, significa que o classificador é mais ”liberal”, isto é, ele estima positivo mesmo com evidências fracas, o que o leva a um alto TPR, mas a um alto FPR também. A linha pontilhada na diagonal representa a estimativa aleatória (x=y) (FAWCETT, 2006).

Observe que a curva ROC apresentada na Figura 2.7 é uma extensão da versão tra-

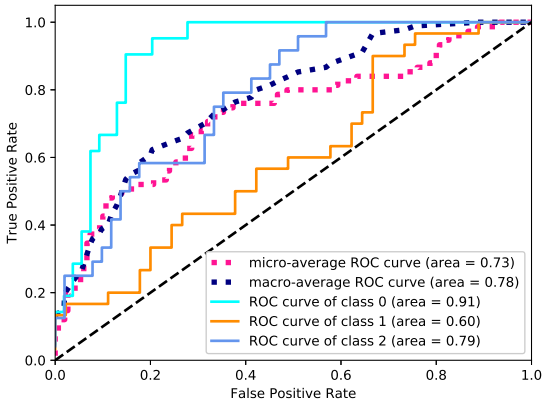


Figura 2.7: Versão estendida do gráfico de curvas ROC utilizando o algoritmo SVM para treino e teste na base de dados iris.

dicional, na qual é apresentada a curva de ROC de cada classe, além da *micro* e *macro average ROC curve*. Essas duas últimas são calculadas usando a média dos valores do TPR e FPR.

No caso da regressão pode-se utilizar o MMQ, que calcula a média aritmética da soma das diferenças ao quadrado entre o valor estimado e o valor real da nota do aluno, conforme Equação 2.1, onde n é a quantidade de instâncias da base, h(x) é o valor estimado e y o valor real (OTERO et al., 2014b).

$$MMQ = \frac{1}{2n}\sum_{i=1}^{n}(h(x_i - y_i)^2) \tag{2.1}$$

O valor é dividido por 2, pois normalmente é utilizado um gradiente descendente para encontrar o valor mínimo desta função de custo, que faz uso do cálculo da derivada no ponto para encontrar a inclinação da reta tangente até achar o ponto mínimo local em caso de uma região côncava e o ponto mínimo global em caso de uma região convexa.

Como no MMQ as erros (diferença entre o valor estimado e o valor real) são elevados ao quadrado e somados, então uma forma de suavização é usar a raiz quadrada desse valor, isto é o *Root Mean Square Error* (RMSE).

2.3 Algoritmos Genéticos

Algoritmos genéticos (AG) são uma família de modelos computacionais inspirados pela teoria da evolução. Tais algoritmos codificam potenciais soluções para problemas es-

pecíficos utilizando um estrutura de dados similar a de um cromossomo. Tais soluções codificadas são submetidas a um processo de recombinações utilizando operadores que preservam as informações críticas (WHITLEY, 1994).

Inicialmente, os algoritmos genéticos criam uma população de indivíduos (também conhecidos como soluções candidatas ou fenótipos) tipicamente de forma aleatória, onde cada indivíduo possui um conjunto de propriedades (também conhecidas como genótipos) que podem ser evoluídas. O processo de evolução acontece de forma iterativa, onde a população de cada iteração é chamada de geração. Cada indivíduo da população em cada geração é avaliado por uma função objetivo chamada *fitness*. Assim, os indivíduos com *fitness* mais altos possuem maiores probabilidades de serem selecionados para reprodução (MITCHELL, 1998).

A reprodução acontece utilizando operadores de recombinação. Para ilustrar, existe um operador de cruzamento (ou *crossover*) que combina os genótipos entre os indivíduos selecionados para reprodução, o que faz com que os indivíduos da próxima geração herdem os genes dos indivíduos eleitos da geração anterior. Após o cruzamento, adiciona-se uma mutação nas soluções cruzadas, a fim de que haja mais variação entre as soluções nas próximas gerações. Do contrário, as soluções candidatas poderiam ficar rapidamente prezas em um mínimo local do espaço de busca. Note que o AG pode ser visto como uma busca guiada em um espaço de soluções (WHITLEY, 1994).

Por fim, tipicamente um AG pode ser sumarizado em 4 etapas (SHIFFMAN, 2012), conforme segue:

- Etapa 1: **Inicialização** - criar uma população com *N* indivíduos, cada um com um genótipo aleatório.
- *Repita*:

  Etapa 2: **Seleção** - avaliar o *fitness* de cada indivíduo da população.

  Etapa 3: **Reprodução**

  Repita N vezes:

  Eleger *K* indivíduos usando uma função de probabilidade baseada nos valores de *fitness* obtidos na etapa anterior. Os indivíduos eleitos serão os pais da uma nova geração de indivíduos.

Cruzamento - criar novos indivíduos (filhos) através da combinação do genótipo dos K indivíduos (pais) selecionados na etapa anterior.

Mutação - adicionar uma variação no genótipo dos filhos.

Adicionar os novos filhos na nova população

Etapa 4: Substituir a população anterior pela nova população e voltar a etapa 2.

2.3.1 Construção Automática de Pipelines de Aprendizagem de Máquina Utilizando Algoritmos Genéticos

Existe um campo de pesquisa que estuda a automação do processo de aprendizagem de máquina. Esse campo é conhecido como *Automated Machine Learning* (AutoML). Inicialmente, os pesquisadores investigaram técnicas para otimização de subconjuntos do *pipeline* de AM, como automatização do ajuste de hiperparâmetros ou seleção de atributos ou ainda da escolha do algoritmo de aprendizagem de máquina (HUTTER; LÜCKE; SCHMIDT-THIEME, 2015).

Com a evolução da inteligência artificial, métodos sofisticados de *AutoML* vêm sendo propostos. Para ilustrar, Feurer et al. (2015) apresentaram o *auto-sklearn* que é uma ferramenta que emprega otimização *bayesiana* para a produção inteira de *pipelines* de AM, isto é, o pré-processamento dos atributos, seleção do algoritmo de AM e ajuste de hiperparâmetros. Entretanto, o *auto-sklearn* não é capaz de produzir um número grande de *pipelines*, uma vez que ele explora um número fixo de etapas dos *pipelines* que incluem apenas um algoritmo de pré-processamento de dados, uma algoritmo de pré-processamento de atributos e um algoritmo de AM por *pipeline*.

Por outro lado, Zutty et al. (2015) demonstrou que a otimização com programação genética, *genetic programming* (GP) em inglês, pode superar os seres humanos na busca de um *pipeline* de AM que melhor se encaixa em uma tarefa de aprendizagem supervisionada. Nesse caminho, Olson et al. (2016) proporam um método de construção e otimização de árvores de *pipelines* de AM com emprego de algoritmos genéticos. O método criado por Olson et al. (2016) é chamado de *Tree-based Pipeline Optimization Tool* (TPOT) e a ideia principal é criar inicialmente uma população inteira de *pipelines* de AM aleatórios e evoluí-los com mutações e cruzamentos de geração em geração. Para montar os *pipelines*, Olson et al. (2016) empregaram vários algoritmos de seleção, construção e

transformação de atributos e algoritmos de AM juntamente com ajuste de hiperparâmetros. Olson et al. (2016) explicam que a árvore é criada utilizando operadores de *pipeline*, onde cada operador é responsável por adicionar, remover, transformar atributos ou dados, a fim de que um operador final realizasse o processo de classificação ou regressão.

Para ilustrar, a Figura 2.8 apresenta um exemplo do processo de construção de um *pipeline* em árvore, onde cada círculo representa um operador. Note que são criadas cópias da base de treinamento para que os operadores possam combinar os atributos e dados modificados. Posteriormente, pode haver um processo de seleção de atributos para que enfim seja realizada a construção do modelo preditivo. Observe ainda que o *pipeline* mostrado na Figura 2.8 é um indivíduo da população que será evoluído ao longo das gerações. Note que para realizar a evolução emprega-se uma seleção eletiva avaliando os indivíduos da população com uma função *fitness* que mensura o desempenho do *pipeline* na base de treino. A função *fitness* nesse contexto pode ser a acurácia ou uma outra função, conforme explicada na seção 2.2.4, que pode ser passada como parâmetro para o algoritmo para avaliar o desempenho do *pipeline* no treino.

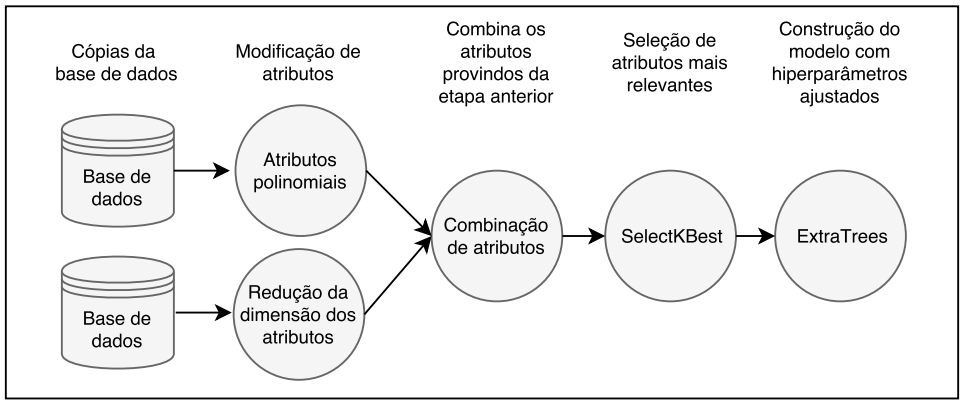


Figura 2.8: Operadores para construção dos *pipelines* em árvore utilizando o método TPOT. Adaptado de Olson et al. (2016).

O TPOT emprega o algoritmo NSGA-II (DEB et al., 2002), que é uma algoritmo genético multiobjetivo. As funções objetivos utilizadas no NSGA-II foram de maximizar a acurácia dos modelos e minimizar a complexidade dos *pipelines*, onde a complexidade é medida pelo número de operadores do *pipeline*.

2.4 Coleta de Dados em Ambientes Online para Turmas de Programação

Conforme exposto nas seção 2.1, os sistemas *online* em turmas de programação abriram uma oportunidade de capturar os dados do processo de codificação do aluno, o que aliado a técnicas de AM e MD, pode ser bastante útil para melhorar o processo de ensino e aprendizagem de programação e, consequentemente, diminuir os índices de retenção. Nesta seção, serão abordados os tópicos relacionados a ambientes de turmas de programação que seguem, em geral, a arquitetura lógica apresentada na Figura 2.1. Desta forma, serão elucidados os níveis de granularidade da coleta desses dados e, por fim, serão mostrados atributos empregados para realizar a predição do desempenho do aluno encontradas nas publicações analisadas em um Mapeamento Sistemático da Literatura (MSL)[1] conduzido pelos autores.

Primeiramente, a coleta de dados em ambientes *online* para turmas de programação pode ocorrer em variados níveis de granularidade, isto é, os dados podem ser coletados em tamanhos e frequências distintos. Quanto mais detalhado o nível, mais granular, como é o caso de sistemas que coletam cada tecla digitada pelo aluno, representando cada passo de alteração no código, estando no nível *keystroke*. Por outro lado, outros ambientes registram as mensagens de compilação e execução geradas e, além do mais, outras gravam, de forma menos granular, as submissões dos estudantes (IHANTOLA et al., 2015).

Alguns pesquisadores vêm usando esses dados geralmente para fazer predição do desempenho do aluno e, a partir disso, identificar estudantes em risco (ESTEY; COADY, 2016; HUNG et al., 2016), com hábitos de estudo ruins (AUVINEN, 2015), modelar o comportamento do aluno para ações reflexivas (VIHAVAINEN, 2013; GUERRA-HOLLSTEIN; BRUSILOVSKY, 2016), atribuição de atividade baseada em habilidade (ANDERSEN et al., 2016), detecção automática de alunos com experiência em programação anterior ao curso de computação (LEINONEN et al., 2016) e etc. Existe ainda pesquisas como as de Otero et al. (2014a), Otero et al. (2016) que fizeram um *ranking* das métricas de código com maior poder de predição, utilizando algoritmos de lógica difusa[2]

[1] http://goo.gl/4RD1hD

[2] Na lógica difusa as variáveis não assumem os valores verdadeira ou falso, mas um valor entre 0 e 1 que representa a probabilidade de uma sentença ser falsa ou verdadeira (NOVÁK; PERFILIEVA; MOCKOR, 2012).

e técnicas de AM e MD.

Na próxima seção, essas características utilizadas para predição de nota serão categorizadas pelo nível de granularidade da coleta dos dados dos alunos. Destaca-se que, quando essas características são baseadas no código dos estudantes, são também conhecidas como métricas de código ou métricas de software e, como elas são orientadas aos dados do processo de desenvolvimento do estudantes, também são conhecidas como métricas de código *data-driven* (VIHAVAINEN; LUUKKAINEN; IHANTOLA, 2014; IHANTOLA et al., 2015).

2.4.1 Eventos de Submissão

Os juízes *online* registram automaticamente as soluções submetidas pelos alunos, o que permite uma análise delas. Além disso, esses sistemas de correção automática de código, normalmente, salvam também meta-dados com um *time-stamp*, identificador do estudante e o *feedback* associado ao código enviado. Um problema de analisar somente nesse nível, está na não visualização dos estados de evolução do aluno e as ações entre os eventos de submissão (IHANTOLA et al., 2015).

Entretanto, muitos trabalhos utilizam essa abordagem com relativo sucesso, como Otero et al. (2016), que calcula a quantidade de linhas sem e com comentário do código, número de palavras-chave, número de blocos e outros para prever a média da turma. Ainda nesse caminho, Bishop et al. (2015) disponibilizou publicamente uma base de dados para pesquisa com submissões de 258 usuários, que resolveram problemas em C# e Java em uma ambiente chamado *The Code Hunt*.

Além disso, alguns trabalhos como o de Ahadi, Vihavainen e Lister (2016), analisaram a correlação do número de tentativas de submissão e a quantidade de atividades corretas e erradas com as notas intermediárias dos alunos. Já Auvinen (2015) pesquisou sobre a inferência de hábitos ruins de estudos, usando além da quantidade de tentativas, a média de tempo entre as tentativas e a média de tempo que o aluno levava para resolver os exercícios.

2.4.2 Eventos Snapshots

Alguns sistemas educacionais para turmas iniciais de programação possuem componentes de *software* para capturar *snapshots* do progresso do estudante. Eles podem ser registra-

dos de duas formas: a) em intervalos definidos, no qual o sistema irá salvar automaticamente o estado do código ou b) quando o estudante realiza uma ação como salvar, executar, compilar e testar o código (VIHAVAINEN; LUUKKAINEN; IHANTOLA, 2014).

Para ilustrar, no trabalho de Allevato e Edwards (2010) foi descoberto que quando os estudantes editam seus programas, frequentemente deletando grandes blocos (por exemplo métodos inteiros) e começam a fazer tudo desde o começo, então tais alunos estão com dificuldade. Além disso, existem pesquisas que estudam especificamente os erros dos alunos, usando para tanto as compilações (serão tratados na próxima subseção, mas são considerados eventos de *snapshots*) ou execução do programa para inferir a nota ou identificar alunos em risco de reprovação ou com hábitos ruins de estudo (AUVINEN, 2015; PETERSON; SPACCO; VIHAVAINEN, 2015; CARTER; HUNDHAUSEN; ADESOPE, 2015; JADUD, 2006b).

2.4.3 Eventos de Compilação

Os eventos de compilação são um subconjunto dos eventos *snapshots* supracitados e muitos autores investigaram esse tipo de característica em detalhe. Uma publicação relevante neste contexto é a de Jadud (2006b), que instrumentalizou o popular *BlueJ* para registrar eventos de compilação. Essa abordagem traz como benefício o fato de ser mais granular que a análise da submissão do código enviado, uma vez que erros podem ser coletados antes que os alunos corrijam e enviem o programa para o veredicto de um juiz *online* (TOLL, 2016). Usando esses dados, Jadud apresentou uma lista de erros de compilação comuns e descobriu que estudantes tendem a cometer os mesmos erros repetidas vezes. Assim, o autor propôs uma métrica chamada *Error Quotient* (EQ), que pode ser usada para prever as notas intermediárias dos estudantes ou detectar estudantes em risco de reprovação em turmas que usam a linguagem de programação Java.

Muitos trabalhos fizeram adaptações do EQ, como Becker (2016) que calculava a frequência de erros repetidos, Watson e Li (2014) fizeram um algoritmo que atribua penalidades para erros cometidos e Carter, Hundhausen e Adesope (2015) que usou o EQ, o *Watwin Score* e informações de mensagens de execução do código. Entretanto esses trabalhos ainda não apresentaram desempenhos satisfatórios em contextos diferentes dos usado por Jadud, isto é, usando linguagens de programação diferente de Java.

2.4.4 Eventos Keystroke

Esse é o nível mais alto de granularidade de coleta de dados, uma vez que são registradas as teclas digitadas pelos alunos quando esses estão resolvendo os exercícios. O sistema precisa de um *keylogger* para ter essa funcionalidade. Existem mais publicações que tratam do nível de submissão e um número reduzido que trata de eventos *keystroke*. Entretanto, a quantidade de pesquisas com esse tipo de evento vem crescendo (IHANTOLA et al., 2015).

Essas evidências representam um retrato mais fiel do processo de aprendizagem do aluno, posto que registra cada mudança no código fonte dele. Além disso, é possível observar os alunos que não terminam suas atividades e nem chegam a submetê-las. Finalmente, quando a análise é realizada usando evidências de *snapshots* e submissões, é assumido que o estudante pegou os caminhos mais diretos até a solução final, ocultando detalhes de seu esforço para chegar até o envio de uma solução bem sucedida (VIHAVAINEN; LUUKKAINEN; IHANTOLA, 2014).

Neste sentido, destaca-se alguns trabalhos como o de Leinonen et al. (2016), que usou a latência das teclas digitadas e a quantidade de dígrafos digitados para prever a nota do aluno e identificar alunos com experiência prévia em programação. Ainda usando eventos deste nível, a pesquisa de Vihavainen, Helminen e Ihantola (2014) analisou como os estudantes enfrentam suas primeiras linhas de código em uma IDE.

2.4.5 Eventos de Interação

Alguns sistemas registram outros eventos como cliques, tempo gasto usando os recursos, eventos do *debugger*, foco da aplicação, atividade de console e outros (VIHAVAINEN; LUUKKAINEN; IHANTOLA, 2014). Essas informações podem ser usadas para verificar quando o estudante não está mais usando o sistema, qual o recurso mais usado, em quais momentos e quantas vezes o aluno executou ou compilou o código, ou ainda, quanto tempo por dia o estudante gasta usando o sistema.

No capítulo de trabalhos relacionados será apresentada uma taxonomia de eventos registrados em ambientes para programação utilizados em modelos preditivos, representada em um diagrama de *Venn*. Acredita-se que, por fins estéticos e de entendimento, sem perda de generalidade, importa que o diagrama possua três macro conjuntos inter-

relacionados, no máximo. Em função disso, eventos de interação foram categorizados como eventos de compilação, quando relacionados à quantidade de vezes que o aluno compilou o código e como eventos de submissão, quando relacionados à quantidade de vezes que o aluno tentou responder atividades ou ao tempo que ele levou para solucionar o problema.

2.4.6 Ética e Privacidade dos Dados

Nesse contexto, a privacidade está relacionada à habilidade de identificar estudantes a partir do seus dados do processo de programação. É comum esconder os dados pessoais em uma base de dados compartilhada, como nome, email, matrícula e etc. No entanto, existe a possibilidade da identificação do estudante ocorrer em diferentes níveis através de informações que podem passar despercebidos dentro da base de dados, como idade, sexo, organização, endereço IP usado e outros, principalmente, quando a base de dados é pequena. Para ilustrar, alguns dados podem ser identificados em comentários, ou através de análise manual pelos nomes das variáveis e métodos (estilo de programação) que o estudante costuma utilizar (IHANTOLA et al., 2015). Observe que até o tempo entre as teclas pressionadas pode ser usada para distinguir estudantes, como aponta o estudo de Leinonen et al. (2016). Todos esses cuidados devem ser tomadas quando os dados são preparados para serem compartilhados publicamente.

Destaca-se que os alunos devem estar cientes que os dados estão sendo coletados, que serão usados para experimentos e que poderão ser compartilhados com outros pesquisadores. A participação do aluno deve ser facultativa (VIHAVAINEN; LUUKKAINEN; IHANTOLA, 2014). Ihantola et al. (2015) explica que para a coleta de dados em universidades, sem o interesse de disponibilização pública, não é necessário seguir um protocolo institucional, apenas a anuência dos alunos.

Por outro lado, as questões éticas estão relacionadas não apenas à coleta dos dados e transparências disso com os alunos, mas também ao uso, frisando-se que a divulgação do histórico de aprendizagem de um aluno pode influenciar na carreira e na vida social dele (IHANTOLA et al., 2015). As práticas científicas devem ser controlas pelos comitês de ética das universidades onde as pesquisas estão sendo conduzidas.

2.5 Considerações Finais do Capítulo

Os benefícios das métricas *data-driven* são significativos, uma vez que podem ser usados para analisar como os alunos resolvem os problemas ou como eles lidam com os erros, ou ainda, como estão gerenciando o tempo e prazos das atividades. Tudo isso está diretamente ligado ao comportamento de programação do aluno (VIHAVAINEN; LUUKKAINEN; IHANTOLA, 2014). Além disso, como os dados são referentes ao processo de aprendizagem do discente e não apenas ao produto final gerado e submetido (código com a solução), pode-se, assim, avaliar o processo e não apenas o produto, como em juízes *online* que julgam o programa enviado, mas não percebe os caminhos traçados até chegar a esse fim.

Destaca-se ainda que um passo que as áreas de LA e EDM vêm tentando dar, a fim de melhorar o processo de ensino e aprendizagem de programação está em inferir o desempenho dos alunos, baseado no comportamento de programação deles. A partir desse valor estimado, pode-se identificar alunos em risco, com dificuldade e facilidade, com maus hábitos de estudo e muito mais.

Nesse sentido, foram coletadas as interações dos alunos com o sistema CodeBench, isto é, as evidências *keystroke*, *snapshot*, submissão e de interação, que representam de forma mais detalhada o processo de aprendizagem do aluno. As evidências mais relevantes foram usadas para compor um modelo que irá realizar a predição de alunos de turmas de IPC. Para tanto, foram utilizados os métodos de AM e MD apresentados neste capítulo.

Capítulo 3

Trabalhos Relacionados

Uma vez que o tema da pesquisa foi delimitado, o objetivo definido e as questões de pesquisa estabelecidas, conduziu-se um Mapeamento Sistemático da Literatura (MSL)[1] com o intuito de identificar o estado da arte relacionado ao uso de técnicas de aprendizagem de máquina e mineração de dados para predição de desempenho ou inferência de dificuldade e expertise de alunos de turmas iniciais de programação na graduação. Nesse sentido, serão apresentados os resultados desta pesquisa e os principais trabalhos encontrados nela, que estão fortemente relacionados a este.

Os trabalhos a seguir utilizam dados coletados em ambientes de turmas de programação, a fim de melhorar o processo de ensino e aprendizagem nesse contexto. Eles usam eventos *keystroke*, *snapshots*, compilação, submissão e de interação aliado a técnicas de AM, MD ou análise estatística para fazer predição do desempenho do aluno.

3.1 Métodos Preditivos com Uso de Eventos Keystroke

Eventos *keystroke* representam de forma mais detalhada o comportamento de programação do aluno, uma vez que é registrado o processo de construção da solução do início ao fim, inclusive de alunos que não conseguiram submeter corretamente e pararam no meio do caminho. Nesse sentido, Leinonen et al. (2016) teve por objetivo detectar alunos que já possuem experiência prévia em programação em turmas introdutórias de programação e verificar qual a relação da *keystroke latency*[2] e da quantidade de dígrafos específicos

[1] http://goo.gl/4RD1hD

[2] Latência do uso das teclas/digitação, isto é, a diferença de tempo entre teclas sucessivas. Mais especificamente pares de caracteres digitados chamados de dígrafos, neste contexto.

digitados com a nota final do aluno.

Como resultado, o estudo mostrou que 5% da variância da nota de exames de cursos de programação podem ser explicadas pela *keystroke latency*. Em um experimento, o autor usou como classificador o *Random Forest* e obteve acurácia de 71,77% para prever a nota dos alunos com dados da primeira à sétima semana de curso, usando evidências *keystroke*. Em outro experimento, no qual a ideia era apenas categorizar os alunos em duas classes, acima da média e abaixo da média da turma, mais de 19% da variância foi explicado pelo modelo e o autor obteve 74,56% de acurácia usando redes Bayesianas, usando as mesmas evidências. Entretanto, foi observado que existe uma baixa correlação da *keystroke latency* com a nota nas primeiras semanas. Por fim, o pesquisa explica que ainda está em um modo embrionário no que tange a distinguir alunos com e sem experiência prévia em programação.

No trabalho de Ahadi et al. (2015), objetivou-se a detecção de alunos com baixo e alto desempenho, usando um conjunto híbrido de evidências, baseada nos dados do processo de desenvolvimento do aluno de programação com granularidade *keystroke* e informações coletados utilizando questionário. Os autores fizeram uma análise do número de *logs* que os alunos geravam para resolver as questões de programação e a curretude das submissões. Assim para cada questão, foi criado um atributo referente ao número de *logs* e outro para a corretude. Tais atributos foram empregados em algoritmos de AM. Além disso, os autores usaram atributos como o gênero do aluno, idade e a médias das notas em turmas anteriores.

Fazendo uma reprodução do trabalho de Ahadi et al. (2015), Castro-Wunsch, Ahadi e Petersen (2017) avaliaram o uso de Redes Neurais e algoritmos de tradicionais de aprendizagem de máquina para identificar estudantes com risco de reprovação. Castro-Wunsch, Ahadi e Petersen (2017) constatou que o número de *logs* gerados pelos alunos e corretude dos exercícios submetidos possui um relacionamento com o desempenho dos alunos.

Munson e Zitovsky (2018) empregaram modelos de regressão para realizar a predição da média final de 86 estudantes de turmas de IPC. Os autores analisaram os *logs* e eventos de compilação dos alunos enquanto os estudantes resolviam questões de programação em Java em uma IDE chamada *CodeWork*, desenvolvida pelos próprios autores do artigo. Naquele trabalho, todas as atividades foram corrigidas manualmente pelos instrutores. Os atributos utilizados para a construção dos modelos preditivos eram baseados no número e

tamanho das sessões[3] de programação, tamanho dos códigos fontes submetidos, número de erros de compilação, horário do dia que os alunos estudavam, local onde os alunos resolviam os exercícios (em casa ou no laboratório); tempo de uso da IDE, notas das atividades e quantidade de vezes que os estudantes copiam e colavam. Como resultado, os modelos preditivos conseguiram explicar aproximadamente 86% da variação da nota dos alunos.

3.2 Métodos Preditivos com Uso de Eventos Snapshots

Os eventos de *snapshots* podem ser registrados de duas formas: a) em intervalos definidos, onde o sistema registra automaticamente o estado do código ou b) quando o estudante realiza uma ação como salvar, executar, compilar e testar o código (VIHAVAINEN; LUUKKAINEN; IHANTOLA, 2014).

Em relação às mensagens de compilação, um dos pioneiros foi Jadud (2006a), o qual detectou um ciclo de edição, compilação e execução de códigos em alunos que estavam aprendendo a linguagem Java, conforme Figura 3.1. Baseado nisso, o autor propôs um algoritmo para quantificar os erros de compilação usando a métrica Error Quotient (EQ), no qual uma pontuação é calculada quando os alunos cometem erros, conforme fluxograma apresentado na Figura 3.2.

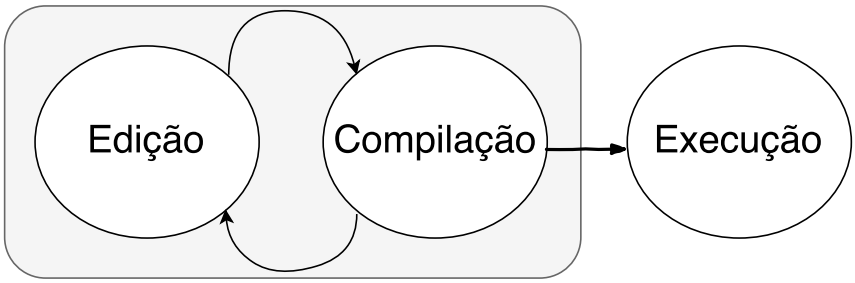


Figura 3.1: Ciclo de edição-compilação-execução de alunos de turmas introdutórias. Adaptado de Jadud (2006a).

O Algoritmo de EQ funciona em quatro etapas, a saber (JADUD, 2006a):

1 **Coleta** - cria pares consecutivos de compilações das sessões de atividades dos alunos.

2 **Cálculo** - calcula a nota de acordo com a Figura 3.2.

[3]No contexto do trabalho de Munson e Zitovsky (2018), uma sessão era o o intervalo de tempo em que o aluno estava programando na IDE de forma efetiva. Em outras palavras, se o aluno passasse 10 minutos sem gerar nenhum *log* então aquela sessão era encerrada.

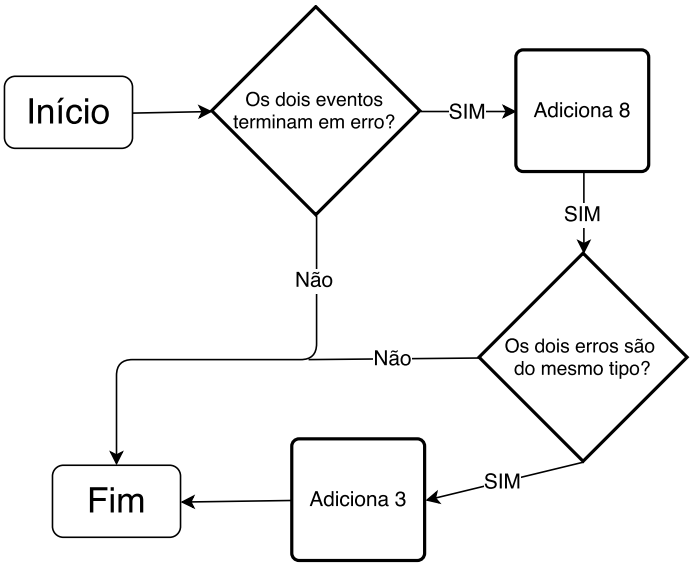


Figura 3.2: Fluxograma do algoritmo de EQ. Valores são normalizados e uma média é calculado de acordo com o algoritmo EQ (BECKER, 2016)

3 **Normalização** - divide a nota atribuída por 11, que é o valor máximo possível para cada valor.

4 **Média** - soma as notas atribuídas e divide por n, que é o número de pares.

Ainda nesse caminho, um trabalho notável foi o de Watson e Li (2014), que teve por intento inferir a nota de estudantes de turmas iniciais de programação, usando uma derivação da métrica EQ proposta por Jadud. Para tanto, foi criado um algoritmo chamado de *Watwin Algorithm* que usa uma abordagem de pontuação, onde um aluno é relativamente penalizado com base na quantidade de tempo que ele leva para resolver tipos específicos de erro, em comparação com os tempos de resolução de seus pares de compilação. Essa pontuação é usada para fazer a predição do desempenho do estudante.

Por outro lado, Carter, Hundhausen e Adesope (2015) relatam que trabalhos de Jadud e Whatson obtiveram resultados modestos quando usados para a predição da nota de alunos de programação, levando em consideração apenas as tentativas de compilação, usando exclusivamente métricas como Error Quocient e Watwin Score. Assim, Carter faz uma expansão desses dois, tendo por objetivo analisar o processo de programação dos alunos de forma mais holística, checando dinamicamente a correção sintática e semântica dos alunos de turmas iniciais de programação, a fim de aprimorar os resultados obtidos por trabalhos que usaram as métricas supracitadas. Para tanto é utilizado um modelo normalizado do estado do aluno de programação que caracteriza as atividades realizadas pelos

estudantes (NPSM).

O autor Becker (2016) apresentou uma métrica chamada RED para quantificar a frequência de erros de compilação repetidos, baseado no quociente de erros (EQ) proposto por Jabud. O autor demonstrou que a métrica RED é menos sensível ao contexto, útil para códigos fontes curtos e tem uma maior capacidade de predição de desempenho se comparado ao EQ. Ser menos sensível ao contexto é importante, pois isso dá mais poder de generalização.

Destaca-se que a linguagem de programação e o juiz *online* utilizados têm impacto na eficácia das características apresentadas nesta seção (eventos de *snapshots*), uma vez que erros de compilação em Java e C são mais específicos do que, por exemplo, em python e os juízes *online* possuem seus nuances. Frisa-se ainda que os experimentos supracitados foram realizados em contextos semelhantes. Assim, Peterson, Spacco e Vihavainen (2015) tiveram por objetivo explorar as principais características que diferenciam os contextos e avaliar o desempenho da métrica EQ em cenários distintos, com 4 bases de dados distintas, sendo com 2 turmas que usavam Python, uma com C e outra Java. Foi constatado que a métrica EQ não é efetiva para todas as linguagens de programação experimentadas. O autor explica que, apesar de todo o esforço, na linguagem C o modelo preditivo não teve um desempenho significativo. No contexto do uso de Java o autor conseguiu resultados similares ao de estudos anteriores, como os de Jadud. Já com o uso de Python na base de dados do CloudCoder, um cenário similar ao apresentado por Jadud, o autor obteve resultados mais significativos, entretanto na segunda base que usava Python obteve um resultado com baixa significância estatística.

Finalmente, Estey e Coady (2016) apresenta o BitFit [4], uma ferramenta desenvolvida para que alunos de turmas iniciais de programação possam fazer atividades semanais e para a coleta dos dados do processo de desenvolvimento deles. A ferramenta, além de ser um juiz *online*, disponibiliza uma série progressiva de dicas para cada atividade. Destarte, o estudo tem por intento encontrar padrões sutis de aprendizagem, a fim de criar um modelo preditivo para identificar estudantes que possivelmente não irão passar na disciplina. O modelo preditivo apresentado obteve uma acurácia de 81% já no final do curso na tarefa de classificação binária para inferir se os alunos iriam ser aprovados ou reprovados. Além disso, neste estudo os autores revelam que os atributos propostos

[4]https://github.com/ModSquad-AVA/BitFit

relacionados ao número de compilações, consumo de dicas, desempenho nas listas de exercícios e número de tentativas de submissão podem ser usados para a construção de modelos preditivos precoces, isto é, capazes de inferir o desempenho dos alunos ainda nas primeiras semanas de aula. O estudo revelou ainda que os estudantes em risco não só tiveram menos tentativas de submissões e frequência de compilação, mas também tinham padrões diferentes em relação ao consumo repetido das dicas geradas pelo BitFit para cada questão.

3.3 Métodos Preditivos com Uso de Eventos de Submissões

O trabalho de Otero et al. (2016) teve por objetivo prever o índice de aprovação de turmas iniciais de programação em um curso de ciência da computação, usando eventos de submissão. Utilizou-se um algoritmo de seleção de extensão de lógica difusa e, assim, estabeleceu-se um ranque das 11 métricas mais significativas para compor modelos preditivos usando *ensembles* (conjunto com vários algoritmos de predição cooperativos). Os métodos mais promissores elencados pelo artigo, no geral, foram usando SVM, Random Forest e seleção de características com lógica difusa e *Fisher Selection*.

O índice de aprovação foi dado pela razão entre o número de alunos que ultrapassaram um limiar e o total de alunos. O artigo apresenta uma abordagem indireta, onde é necessário primeiramente prever qual a nota de cada aluno, para então, prever a média da turma. E uma abordagem direta, onde são usadas apenas uma série de características dentre as 11 selecionadas (não foram as mesmas do modelo indireto) para inferir o índice de aprovação.

Ahadi, Vihavainen e Lister (2016) exploraram o relacionamento entre o desempenho nos exercícios realizados pelos estudantes e a sua subsequente nota em uma avaliação. O estudo faz uma comparação, calculando o coeficiente de correlação de *phi* entre o fato do aluno ter passado ou não com o fato dele ter feito uma dada atividade ou não. Outrossim, estimou-se a correlação da nota com a quantidade de vezes que o aluno produziu ou não uma solução correta para cada exercício. Os autores relataram que os resultados do método que utiliza o número de tentativas correlacionam-se melhor com o desempenho no exame. Frisou-se que o método proposto pode ser usado como um *benchmark* para

métodos mais sofisticados de previsão de desempenho de estudantes.

No trabalho de Auvinen (2015) foi investigado se os alunos apresentam hábitos de estudo indesejáveis em termos de prática de gerenciamento de tempo e comportamento de tentativa e erro. Foram analisados dados de dois cursos de informática. Eram utilizados dois tipos de exercícios: a) o aluno envia o código fonte para um juiz *online* b) exercícios de simulação de algoritmos, nos quais o aluno deve manipular estruturas de dados usando uma interface gráfica a fim de demonstrar como o algoritmo funciona.

Os resultados desse artigo estão coerentes com as descobertas anteriores de Spacco et al. (2013), Vihavainen (2013) de que começar a estudar perto do prazo final das atividades está relacionado a um baixo desempenho. Através das observações presentes no experimento, sugere-se que a relação é causal. Além disso, observou-se em alguns alunos, sinais de resolução de problemas por tentativa e erro e isso também está correlacionado com um desempenho inferior nos exercícios e no exame.

3.4 Taxonomia de Evidências

A taxonomia KE-SN-SU, apresentada na Figura 3.3, foi alicerçada nos achados de Vihavainen, Luukkainen e Ihantola (2014), Ihantola et al. (2015). Segundo Fuller et al. (2007), taxonomias podem ser utilizadas em pesquisas educacionais para classificar itens, isto é, elas são um sistema de classificação que podem ser organizadas em vários formatos, como hierárquico (em formato de árvore), um diagrama de Venn, etc. Assim sendo, com base no tipo de evidência coletada das interações dos alunos com os sistemas de programação, construiu-se e usou-se a taxonomia KE-SN-SU para a categorização dos tipos de eventos que compõe os modelos preditos, que foram aqui apresentados como o corrente estado arte para a inferência de desempenho e zona de aprendizagem dos alunos. Assim, as siglas são referentes aos eventos mencionados na seção 2.4, onde os significados são:

- KE: *keystroke*.
- SN: *snapshot*.
- SU: submissão.
- CO: compilação, que é um subconjunto de eventos SN.

- EX: execução, que é um subconjunto de eventos SN. Um exemplo, seria a mensagem de erro de execução gerado por um código em Java.
- TE: teste, que é um subconjunto de eventos SN. Um exemplo, seria a quantidade de testes que um aluno faz em um determinado sistema de correção automática de código.
- KE-SU: *keystroke* e submissão.
- KE-SN: *keystroke* e *snapshot*.
- SN-SU: *snapshot* e submissão.
- KE-SN-SU: *keystroke*, *snapshot* e submissão.

Essas categorias serão usadas na tabela de comparação dos trabalhos relacionados apresentados neste capítulo.

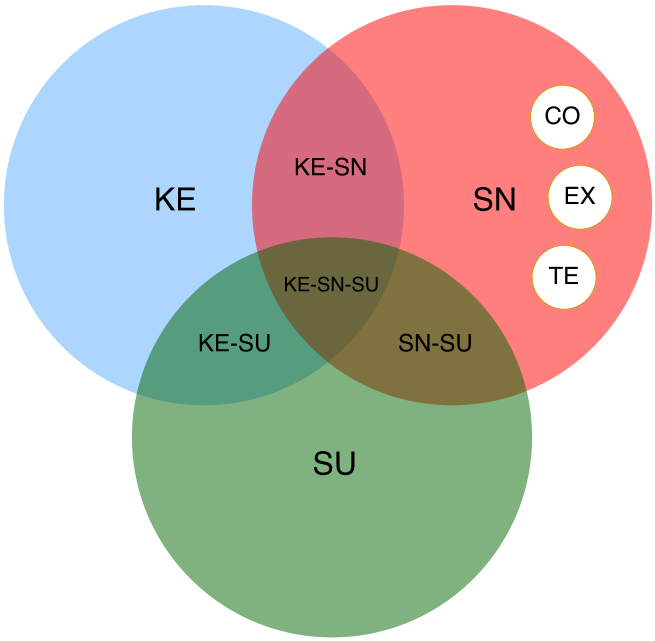


Figura 3.3: Taxonomia KE-SN-SU para categorização de eventos do processo de aprendizagem de alunos de turmas de programação. Adaptado de Vihavainen, Luukkainen e Ihantola (2014), Ihantola et al. (2015).

3.5 Taxonomia de R.A.P

Um grupo de 16 pesquisadores das área de LA e EDM, encabeçado por Ihantola et al. (2015), conduziram uma Revisão Sistemática da Literatura (RSL) sobre coleta de dados

em ambientes de programação e analisaram, de forma manual e criteriosa artigos do período de 2005-2015. Observou-se que, em geral, existe uma tendência crescente na quantidade de publicações que estudam o processo de solução de questões de programação em ambientes de desenvolvimento. Entretanto, várias publicações foram dirigidas com o uso de análises *post-hoc*, tal qual conexões entre as notas do curso e variáveis extraídas do contexto. Tais estudos são valiosos para a área, no entanto é uma nítida amostra da imaturidade deste campo.

Muitos artigos são publicados com estratégias diferentes, o que é bastante válido. No entanto, Ihantola et al. (2015) enfatizam a necessidade de reprodução dos estudos existentes, com o intento de construir um retrato coerente dos fatores que contribuem para a evolução desse campo. Assim, Ihantola et al. (2015) pesquisaram estratégias de reprodução reanálise e replicação de pesquisas, a fim de que as artigos existentes dessa área fossem estendidos, reanalisados, replicados e reproduzidos.

Gomes e Mendes (2015) explica como experimentos podem ser verificados. Ele identificou três grupos de métodos:

- reanálise - mesma base de dados e possivelmente o mesmo método para ratificar os resultados encontrados;
- replicação - mesmo método em um contexto diferente. Por exemplo, mudando a base de dados ou os artefatos;
- reprodução - mesma hipótese. As hipóteses são testadas para verificar se elas são independentes dos métodos e contexto nos quais o experimento foi previamente testado.

Assim, baseado nos métodos de Gomes e Mendes (2015), Ihantola et al. (2015) criaram critérios para reprodução e fizeram uma taxonomia chamada *The R.A.P taxonomy*. Cada letra representa, respectivamente: pesquisador, análise de dados e produção. O critério pesquisador verifica se o mesmo grupo de pesquisadores está envolvido na reprodução do estudo ou na *baseline* do estudo. O critério de análise de dados é para a checagem se o mesmo método foi utilizado, enquanto que o critério produção faz a checagem se novos dados, informações ou resultados foram produzidos. A Figura 3.4 mostra as possibilidades de reprodução em um diagrama de *Venn*.

Assim sendo, cada critério e combinação desses pode ser entendido conforme segue:

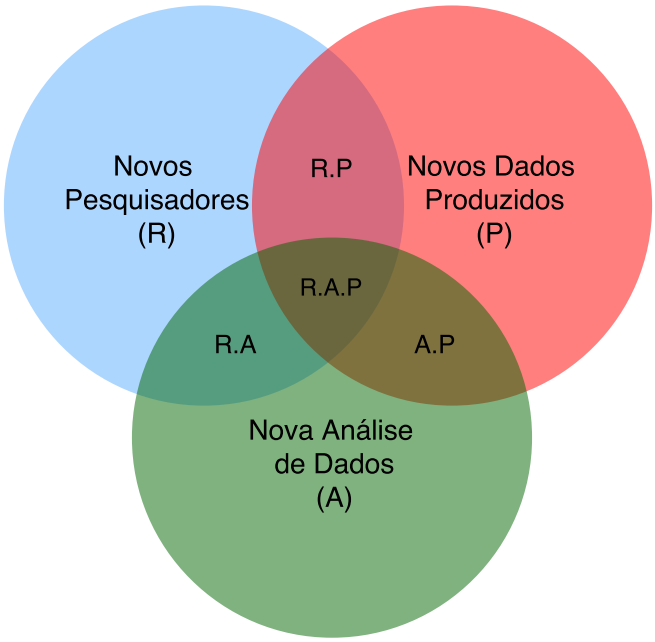


Figura 3.4: Taxonomia R.A.P para reanálise, replicação e reprodução de estudos. Adaptado de (IHANTOLA et al., 2015).

- R = **reanálise** - diferentes pesquisadores que usam o mesmo método e hipóteses do artigo base, com a mesma base de dados;
- A = **análise estendida** - o pesquisador faz uma extensão no artigo base, olhando para a base de dados previamente analisada, mas usando métodos diferentes;
- P = **repetição** - o pesquisador repete a mesma análise, mas com uma nova base de dados;
- R.A = **verificação** - a mesma base de dados é analisada de novo por pesquisadores diferentes, usando métodos diferentes, a fim de verificar os resultados;
- R.P = **estudo de replicação** - um pesquisador diferente segue o mesmo método de análise de uma *baseline*, mas usando uma nova base de dados;
- A.P = **triangulação** - o pesquisador está coletando uma nova base de dados para ser analisado com um novo método;
- R.A.P = **reprodução** - um pesquisador diferente está analisando sua própria base de dados e seguindo um novo método de análise construído no estudo, a fim de testar a hipótese de um artigo base.

Baseado nas recomendações de Ihantola et al. (2015) e na taxonomia R.A.P, realizou-se uma R.P parcial dos atributos utilizados nos estudos de Ahadi et al. (2015), Ahadi, Vihavainen e Lister (2016), Estey e Coady (2016), Jadud (2006a), Otero et al. (2016), Castro-Wunsch, Ahadi e Petersen (2017), Auvinen (2015), Leinonen et al. (2016), Munson e Zitovsky (2018). Não houve uma replicação completa de todos os atributos utilizados nestes estudos, visto que o contexto educacional utilizado neste trabalho é diferente dos apresentados pelos pesquisadores supracitados. Ou seja, utilizou-se linguagens de programação diferentes, com ambientes *online* para turmas de programação diferentes, materiais e metodologias didáticas diferentes, duração de disciplina diferentes e etc.

Assim, alguns atributos não eram viáveis de replicar, uma vez que acredita-se que eles não teriam resultados promissores no presente cenário educacional. Para ilustrar, Otero et al. (2016) sugeriram que fossem utilizadas algumas métricas de código relacionadas ao número de funções dos códigos dos alunos. Entretanto, observando a base de dados utilizada, percebeu-se que, como os alunos são iniciantes, poucos criam funções e quando usam, fazem isso quase no final do curso, quando já não vale mais tanto a pena inferir o desempenho dele. Além disso, alguns atributos eram impraticáveis de replicar, como a frequência de erros de compilação proposta por (ESTEY; COADY, 2016). Note que os alunos deste cenário utilizaram *Python*, dessa forma ao invés de calcular a frequência de erros de compilação, calculou-se a frequência de erros sintáticos.

Finalmente, a intenção deste estudo era, além de gerar modelos preditivos capazes de inferir acuradamente a zona de aprendizagem dos alunos, analisar atributos propostos em outras pesquisas e propor novos atributos que possam, juntos, ser generalizáveis em outros contextos educacionais, conforme sugerido no estudo de Ihantola et al. (2015). Dessa forma, um dos critérios utilizados para usar um atributo no perfil de programação oriundo de outra pesquisa, era se o atributo tinha boas chances de ser válido em contextos educacionais variados. Mais explicações sobre os atributos utilizados neste estudo serão expostas no próximo capítulo.

3.6 Síntese dos Trabalhos Relacionados

Com o intento de analisar as publicações apresentadas neste capítulo, foi criada a Tabela 3.1 que mostra os pontos em comum e diferenças em relação aos principais trabalhos

analisados. Abaixo segue uma descrição de cada item da tabela:

- Granularidade na coleta de dados: qual foi o nível de granularidade dos eventos coletados no ambiente de programação;
- Taxonomia KE-SN-SU: qual foi o tipo de evento usado para compor os atributos do modelo preditivo na publicação, conforme Figura 3.3;
- Coleta de dados da IDE: se houve coleta de dados a partir de uma IDE integrada ao juiz *online* ou a partir de um plugin instalado no ambiente de desenvolvimento no computador pessoal do aluno;
- Dados históricos ou demográficos: se foi utilizada elicitação de dados junto ao aluno para a coleta de dados do perfil escolar anterior ao ingresso universitário ou dados demográficos;
- Predição de nota: se a publicação usou as evidências para prever de alguma forma a nota intermediária ou final do aluno;
- Turma: período em que os alunos de programação estavam;
- Linguagem(ns) de programação: quais linguagens de programação foram utilizadas pelos alunos durante os experimentos.

Observa-se na Tabela 3.1 que a maioria dos estudos não foram conduzidos utilizando o nível de granularidade *keystrole* exposto por (IHANTOLA et al., 2015) para a construção dos modelos preditivos. Como este é o nível mais granular, acredita-se que ele é mais promissor, visto que cada ação dentro da ambiente de programação do aluno é persistida. Observe que através deste nível é possível reconstruir a submissão, selecionar os eventos de compilação uma vez que são registrados os *logs* dos alunos enquanto os estudantes estão resolvendo os problemas de programação.

Nota-se ainda na Tabela 3.1 que em muitos estudos os alunos usavam a linguagem de programação Java. Autores conduzidos pelos achados de Jadud (2006a) analisavam as mensagens de compilação do Java gerada pelos compiladores dos alunos para realizar a predição do desempenho do aluno de IPC. Entretanto, conforme explicado na subseção 3.2, as variações da métrica de código EQ se mostraram dependentes do contexto educacional.

Tabela 3.1: Tabela comparativa dos trabalhos relacionados.

Autor	Granularidade Coleta de Dados	Tipo de Evento M.P.	Coleta de dados da IDE	Dados históricos ou demográficos	Predição da nota	Número de Estudantes	Turma	Linguagem(s) de programação
Otero2016	SU	Submissão	Não	Não	Sim	73	1º semestre	Python
Estey2016	SN-SU	Compilação, Submissão e Dicas	Sim	Não	Sim	652	1º, 2º e 3º semestre	Java
Jadud2006	SN	Compilação	Sim	Não	Sim	234	1º semestre	Java
Watson2013	SN	Compilação	Sim	Não	Sim	45	1º semestre	Java
Becker2016	SN	Compilação	N/I	Não	Não	100	1º semestre	Java
Carter2015	SN	Compilação, Execução, Debug e Teste	Sim	Não	Sim	95	2º ano	C++
Ahadi2015	SN-SU	Keystroke, Compilação, Submissão, Demográficas	Sim	Sim	Sim	86	1º semestre	Java
Ahadi2016	SU	Submissão	Não	Não	Sim	70	1º semestre	Java
Petersen2015	SN	Compilação, Execução, Debug e Teste	Sim	Não	Sim	557	1º semestre	C, Python e Java
Auvinen2015	SU	Submissão	Não	Não	Sim	3299	1º semestre	Python
Leinonen2016	KE	Keystroke	Sim	Não	Sim	223	1º semestre	Java
Munson2018	KE-SN-SU	Keystroke, Compilação, Submissão, Teste	Sim	Não	Sim	86	1º semestre	Java
Proposta do Trabalho	KE-SU	Keystroke, Submissão	Sim	Não	Sim	429	1º semestre	Python

Alguns estudos foram conduzidos com uma amostra relativamente pequena, o que pode ameaçar a validade do método preditivo para outras bases de dados. Ahadi et al. (2015) utilizou dados históricos ou demográficos como atributo dos algoritmo de AM a fim de alavancar a acurácia dos modelos preditivos. Ahadi et al. (2015) alcançou aproximadamente 90% de acurácia na tarefa de identificar antecipadamente se um aluno iria passar ou reprovar. Entretanto, ao interpretar a relação das variáveis demográficas ou históricas com as notas dos alunos e identificar as causas, pouco se pode fazer para evitar que o aluno acabe reprovando ou evadindo. Por isso, neste trabalho optou-se por usar apenas variáveis *data-driven*.

Muitos estudos como os de Becker (2016), Ahadi et al. (2015), Ahadi, Vihavainen e Lister (2016), Auvinen (2015), Leinonen et al. (2016), Estey e Coady (2016) realizaram a predição do desempenho dos alunos de forma binária, isto é, se o aluno iria passar ou reprovar. O presente estudo realizou essa tarefa também, mas além disso, estimou a média

final do aluno em uma escala contínua, visto que dessa forma podem ser empregadas medidas de precaução para os alunos que fiquem na margem da nota mínima exigida para ser aprovado.

Finalmente, a maior contribuição deste estudo está em construir um perfil de programação, que é um conjunto de atributos, onde boa parte desses atributos foram baseados nas métricas de código presentes nos trabalhos expostos neste capítulo. Um dos critérios utilizados para usar um atributo no perfil de programação proveniente de outra pesquisa, era se o atributo tinha boas chances de ser válido em contextos educacionais variados. Dessa forma, acredita-se que o perfil de programação terá mais chances de atender a demanda de ser generalizável, isto é, de ter poder preditivo em bases de dados oriundas de cenários educacionais diferentes do apresentado no presente trabalho.

3.7 Considerações Finais do Capítulo

Foi apresentado o estado da arte relacionado à predição de desempenho e inferência de zona de aprendizagem de alunos de turmas iniciais de programação. Foram elucidados quais tipos de evidência *data-driven* são usadas para compor modelos preditivos e como elas são utilizadas. O que ficou evidente na condução e análise das publicações investigadas, é que ainda existem muitas questões de pesquisa abertas e, consequentemente, muitas oportunidades.

De forma geral, o número de publicações que investigam e modelam o comportamento de alunos de programação de turmas introdutórias tem aumentado significativamente e a ideia, geralmente, é diminuir o índice de evasão dessas turmas, através do uso de ferramentas e técnicas que auxiliem o professor e o estudante.

Atualmente, o grande objetivo dos pesquisadores desta área é implementar um modelo preditivo genérico o suficiente para ser incorporado a qualquer ambiente educacional voltado para turmas iniciais de programação, usando o comportamento do aluno no processo de desenvolvimento de suas soluções, sem o uso de dados históricos escolares ou demográficos. Destaca-se que este anseio é um tanto quanto ambicioso e para alcançá-lo, faz-se necessário, por exemplo, a reprodução das pesquisas existentes em instituições diferentes e com linguagens de programações diferentes, isto é, em um contexto educacional diferente. Além disso, faz-se necessário uma composição das atributos e métricas de código

aqui apresentadas para compor modelos preditivos mais precisos e mais generalizáveis.

Em virtude disso, neste estudo empregou-se uma série de atributos em algoritmos de aprendizagem de máquina, a fim de inferir a zona de aprendizagem dos alunos. Os atributos utilizados foram derivados de métricas de código apresentadas neste capítulo (AHADI et al., 2015; AHADI; VIHAVAINEN; LISTER, 2016; ESTEY; COADY, 2016; JADUD, 2006a; OTERO et al., 2016; CASTRO-WUNSCH; AHADI; PETERSEN, 2017; AUVINEN, 2015; LEINONEN et al., 2016; MUNSON; ZITOVSKY, 2018), juntamente com outras propostas pelos autores deste trabalho. A esse conjunto de atributos deu-se o nome de perfil de programação.

Destaca-se que um dos critério utilizados para usar um atributo no perfil de programação era se a variável poderia ser empregado em um contexto educacional diferente do apresentado neste estudo. Pois dessa forma, as chances do método preditivo nesta pesquisa ser generalizável em outros contextos educacionais são mais altas.

Finalmente, importa dizer que foi realizada a reprodução R.A.P de um trabalho que apresenta um cenário similar ao das turmas de introdução à programação da UFAM, que é o de Estey e Coady (2016). Os resultados foram comparados com o método apresentado no próximo capítulo.

Capítulo 4

Método Proposto

O método proposto nesta pesquisa utiliza evidências extraídas de um juiz *online* para inferir a nota discretizada das avaliações intermediárias e a média final discretizada e contínua dos alunos de turmas de IPC. Para tanto, em um primeiro momento, os estudantes foram classificados em uma zona de aprendizagem que foi dividida em zona de dificuldade (ZD) e zona de expertise (ZE). Os estudantes eram classificados na ZD caso obtivessem uma nota inferior a 5. Do contrário, os alunos eram classificados na ZE. Em um segundo momento, as média finais dos alunos foram estimadas, usando algoritmos de regressão. Nesse caso, a zona de aprendizagem apresentava além da segregação em zonas (ZD ou ZE), um valor (*z-score*) que representa o quão bem ou o quão mal o aluno estava dentro de uma das zonas que ele estivesse inserido.

Para atingir esse objetivo foram identificados na literatura atributos que se relacionam com a nota do aluno, conforme explicado no capítulo 3. Tais características, orientadas aos dados da interação do aluno com um juiz online, foram filtradas através de um pré-processamento de atributos, a fim de encontrar um conjunto mínimo de atributos que satisfaçam o dilema de viés-variância para a predição das notas. Os atributos mais relevantes formaram uma coleção de atributos chamada de perfil de programação. Esse perfil foi utilizado para compor modelos preditivos, baseados em algoritmos de aprendizagem de máquina supervisionada presentes na literatura.

Para resumir, neste capítulo será explanado o método utilizado para a predição da zona de aprendizagem, a arquitetura proposta, o cenário educacional e o perfil de programação.

4.1 Contexto Educacional e Dados

Nesta seção serão apresentados o instrumento e os dados utilizados para a validação do método preditivo proposto neste estudo. Mais especificamente, será apresentado o Codebench, que foi o ambiente de correção automática utilizado nas turmas de IPC estudadas nesta pesquisa. Além disso, será apresentado como foram coletados os dados e a metodologia de ensino adotada nas turmas de IPC.

4.1.1 CodeBench

O CodeBench é um sistema de correção automática de código-fonte (juiz *online*) desenvolvido pelo Instituto de Computação (IComp) da UFAM, com o intuito de dar suporte aos professores e alunos nas disciplinas de programação. Nele, o professor pode cadastrar atividades com enunciado e casos de teste. Desta forma, esse juiz *online* corrige o código submetido pelo aluno, em duas etapas, a saber (CARVALHO; OLIVEIRA; GADELHA, 2016):

1 Análise sintática do código, em que é checado se o programa possui erros sintáticos, o que é realizado baseado na gramática da linguagem de programação adotada. Se erros desse tipo forem encontrados, o aluno é notificado, com o erro e a linha da ocorrência desse. Caso contrário, é realizada a análise lógica do código.

2 Análise lógica do código, em que é verificado a corretude do código-fonte submetido pelo aluno, o que é feito através do uso dos casos de teste fornecidos pelo próprio professor no momento em que ele cadastrou a questão no sistema.

Uma questão q_i, do CodeBench, fornecida pelo professor, possui vários casos de teste que podem ser representados por um conjunto S de pares ordenados (e_j, s_j) de valores de entrada e saída, respectivamente. Desta forma, as várias entradas e_j são usadas durante a execução do código fonte do aluno e a saída do programa dele é comparada com a saída esperada s_j. Um exemplo de uma questão formulada pelo professor teria o enunciado: *Faça um programa que verifica se um determinado número inteiro positivo é primo. Em caso positivo imprima "sim", do contrário imprima "nao".* O professor poderia fornecer junto com esse enunciado, três casos de teste, conforme Tabela 4.1.

Tabela 4.1: Exemplo de Casos de Teste fornecidos pelo professor à medida que ele cria uma questão no CodeBench

Primeiro Caso de Teste		**Segundo Caso de Teste**		**Terceiro Caso de Teste**	
Entrada	21	Entrada	18	Entrada	7
Saída Esperada	nao	Saída Esperada	nao	Saída Esperada	sim

É importante frisar que o sistema mostra um exemplo de uma entrada e saída correspondente, para o melhor entendimento do aluno. Entretanto, os casos de teste da questão ficam ocultos para os discentes. Se o programa do aluno funcionar corretamente, isto é, apresentar as saídas esperadas para todas as entradas do conjunto de teste, então o sistema informa que a submissão está correta. Caso contrário, existem duas possibilidades de *feedback*: a) "sucesso parcial", no qual é informado o percentual de casos de teste que o aluno obteve êxito; b) falha, neste caso, o aluno falhou em todos os casos de teste. Segundo Carvalho, Oliveira e Gadelha (2016), essa abordagem emula o desenvolvimento de código na vida real, uma vez que os programadores devem identificar erros de seus códigos e solucioná-los.

O CodeBench possui uma IDE integrada. Dessa forma, os alunos desenvolvem, submetem e recebem o feedback em um mesmo sítio. Além disso, cada tecla digitada do aluno, evento de clique, de foco e etc. são registrados e esses dados podem ser usados em trabalhos de pesquisa envolvendo as áreas de *learning analytics* e *educational data mining*, tal como o problema de inferir a zona de aprendizagem dos alunos de uma dada disciplina.

A Figura 4.1 mostra a resolução de uma questão na IDE[1] embutida do CodeBench. Existe um console na parte inferior da imagem que propicia ao aluno o teste (execução) do código antes de submetê-lo para a avaliação do CodeBench. Note que é quando o aluno vai escrevendo e testando o código-fonte nessa IDE que são coletados os *logs* deste aluno.

Para ilustrar a coleta de *log* no CodeBench, a Figura 4.2 apresenta as treze primeiras linhas de log referente ao processo de codificação de uma aluno ao tentar resolver a primeira questão de programação apresentada na disciplina de IPC. Nesse problema, o aluno tinha que imprimir na tela a mensagem *Universidade Federal do Amazonas*. A

[1]Observe que a palavra IDE será usada como sinônimo de editor de código no presente estudo.

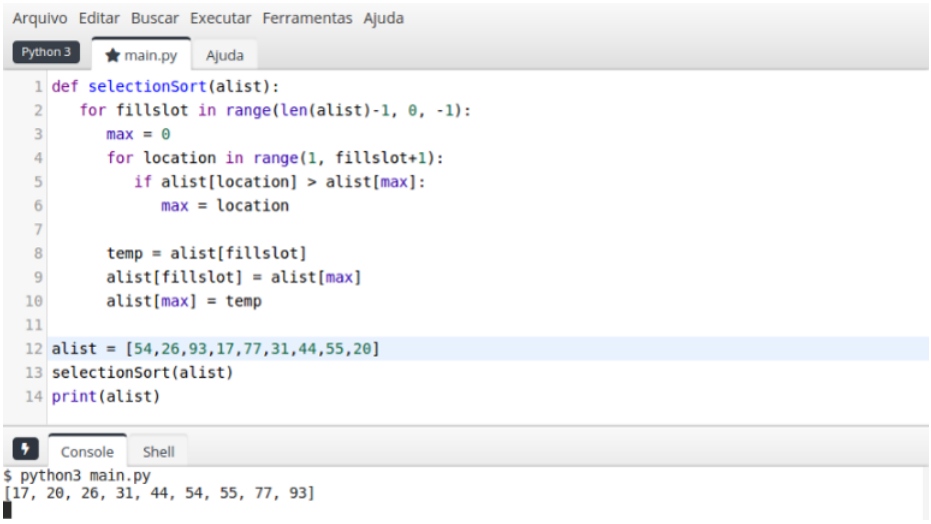


Figura 4.1: Exemplo de uso da IDE embutida no CodeBench.

primeira linha da Figura 4.2 mostra que às 14:41h do dia 20/05/2016 o aluno acessou a IDE. A Linha 2 demonstra que o aluno colou (*origin:"paste"*) da área de transferência o texto *print("Universidade Federal do Amazonas").* Depois disso, na linha 4, o aluno pressionou a tecla de controle *home*. Nas linha 5, 8 e 13 ele realizou algumas edições de caracteres. Percebe-se ainda que o aluno pressionou a tecla *enter* duas vezes e pressionou a seta direcional esquerda do teclado também duas vezes.

```
1   20/5/2016@14:41:23:450:focus
2   20/5/2016@14:41:25:680:change:{"from":{"line":0,"ch":0},"to":{"line":0,"ch":0},"text":["print(
    \"Universidade Federal do Amazonas\")"],"removed":[""],"origin":"paste"}
3   20/5/2016@14:41:28:923:keyHandled:"Home"
4   20/5/2016@14:41:32:12:change:{"from":{"line":0,"ch":0},"to":{"line":0,"ch":0},"text":["#"],"re
    moved":[""],"origin":"+input"}
5   20/5/2016@14:41:32:732:change:{"from":{"line":0,"ch":1},"to":{"line":0,"ch":1},"text":["",""],"
    removed":[""],"origin":"+input"}
6   20/5/2016@14:41:32:732:keyHandled:"Enter"
7   20/5/2016@14:41:32:734:viewportChange:0
8   20/5/2016@14:41:32:922:change:{"from":{"line":1,"ch":0},"to":{"line":1,"ch":0},"text":["",""],"
    removed":[""],"origin":"+input"}
9   20/5/2016@14:41:32:922:keyHandled:"Enter"
10  20/5/2016@14:41:32:924:viewportChange:0
11  20/5/2016@14:41:33:334:keyHandled:"Left"
12  20/5/2016@14:41:33:515:keyHandled:"Left"
13  20/5/2016@14:41:34:224:change:{"from":{"line":0,"ch":1},"to":{"line":0,"ch":1},"text":["
    "],"removed":[""],"origin":"+input"}
```

Figura 4.2: Exemplo de log coletado enquanto o aluno resolve o exercícios no editor de código do CodeBench.

Na Figura 4.3 temos outro exemplo mais simples de *log* coletado, onde nota-se que, analisando as linhas 1 até 7, o aluno escreveu o comando *print()* no editor de código do CodeBench. Observe o nível de granularidade da coleta de dados realizada neste estudo. É com base nesses dados que foram construídos os modelos preditivos para a inferência da zona de aprendizagem dos alunos.

```
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@8:34:44:494:change:{"from":{"line":0,"ch":5},"to":{"line":0,"ch":5},"text
   ":["()"],"removed":[""],"origin":"+input"}
```

Figura 4.3: Exemplo de log do CodeBench de quando o aluno está escrevendo o comando *print()*.

4.1.2 Coleta de Dados e Metodologia de Ensino

Foram coletadas evidências *keystroke*, *snapshots* e submissão de alunos no ambiente CodeBench. Os estudantes resolviam uma lista de exercícios que precedia uma avaliação, ambas realizadas neste sistema. Cada lista tinha em média 10 questões. Assim, cada lista juntamente com a prova sucessora formava uma **sessão** para fins desta pesquisa. No total foram realizadas 7 sessões (S1, S2, ..., S7).

Os dados foram coletados em 9 turmas de IPC da UFAM, durante o período de 05/06/2016 a 13/09/2016. No total, 486 alunos resolveram as listas usando a linguagem de programação *Python* e tinham permissão de fazer um número ilimitado de tentativas de submissão, desde que atendesse ao prazo máximo estipulado para a resolução de cada lista de exercícios. Os tópicos estudados em cada sessão seguem abaixo:

1. Variáveis e estrutura de programação sequencial;
2. Estrutura condicional simples e composta;
3. Estrutura condicional encadeada;
4. Estrutura de repetição por condição;
5. Vetores e Strings;
6. Estrutura de repetição por contagem;
7. Matrizes.

O desempenho dos alunos foi calculado com base em 7 avaliações intermediárias (A1, A2,..., A7), 7 listas de exercícios de codificação, 7 exercícios no formato de *quizz* e em

uma prova final. A média parcial foi calculada conforme Equação 4.1.

$$MP = \frac{(A1+A2)*1+(A3+A4+A5)*2+(A6+A7)*3+ML*2}{16} \tag{4.1}$$

Sendo ML a média aritmética da nota dos laboratórios (codificação e *quizzes*) e MP a média parcial. A média final (MF) foi calculada conforme Equação 4.2.

$$MF = \frac{2*MP+PF}{3} \tag{4.2}$$

Desta forma, as notas das listas de exercícios não tinham um peso muito grande na média final, entretanto é fundamental para o aluno realizá-las para ter um bom desempenho nas avaliações parciais e na prova final.

Do conjunto de evidências extraídas[2], foram selecionadas as características mais relevantes para serem as variáveis independentes do modelo preditivo, utilizando o método que será apresentado neste capítulo. Além disso, as notas de cada avaliação, realizada após as listas de exercícios, e a média final foram escolhidas como variáveis dependentes[3].

4.2 Arquitetura de Trabalho

Esta pesquisa tem por intento alavancar o processo de ensino e aprendizagem em turmas iniciais de programação que usam metodologias de aprendizagem híbridas, através da instrumentalização do professor com uma ferramenta que infere automaticamente a zona de aprendizagem de cada aluno, durante seu progresso nas listas de exercícios.

Deste modo, na Figura 4.4 é apresentado a arquitetura da proposta. Em suma, os alunos resolvem questões criadas pelos professores, em uma IDE integrada a um ACAC que possibilita a edição do código, execução, teste, compilação (para linguagens como C, Java) e correção sintática. À medida que os alunos estão codificando, os dados de interação deles com o sistema são persistidos em uma base de dados como, por exemplo,

[2]Uma descrição de todos os eventos podem ser encontradas em: http://codemirror.net/doc/manual.html#events

[3]Variáveis independentes (neste contexto) são as características ou atributos baseados em evidências coletadas do CodeBench, que compõem um modelo que realiza a predição da nota do aluno. Esta é considerada uma variável dependente ou a classe, pois depende das variáveis independentes. Um exemplo de variável independente é o número de tentativas de submissão para uma dada questão. Este valor poderia ser usado para verificar se possui relação com a nota do aluno, caso existe, diz-se que a nota é uma variável dependente e a quantidade de tentativas uma variável independente.

quais caracteres foram digitados, quais foram apagados, quais foram copiados de outras aplicações, quantas vezes o usuário saiu do editor de código fonte e quantas vezes voltou, quantas tentativas de submissão e etc. Cada evento possui *timestamp* vinculado.

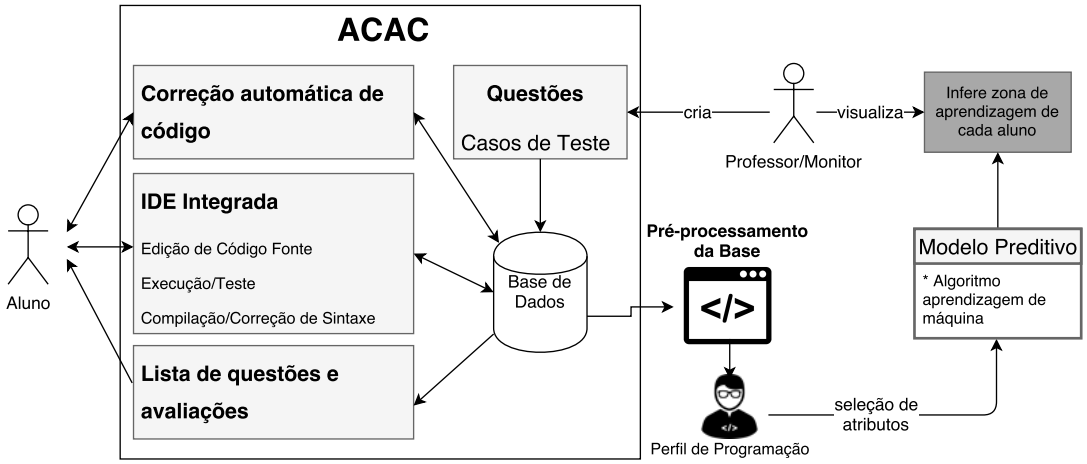


Figura 4.4: Arquitetura da Proposta (PEREIRA; OLIVEIRA; OLIVEIRA, 2017).

Esse conjunto de evidências composto por códigos submetidos pelos alunos e seus respectivos *logs* são pré-processados, usando técnicas de análise estática de código e verificação de frequência de eventos nos *logs*. Os dados pré-processados e preparados são usados pelo método proposto neste trabalho para modelar cada estudante como um conjunto de valores numéricos que representam várias facetas do aluno. A esse conjunto foi dado o nome de perfil de programação do estudante e nele algoritmos de seleção de atributos são aplicados para identificar as características mais relevantes para o problema de predição da zona de aprendizagem. As características selecionadas são utilizadas como atributos de um processo de aprendizagem de máquina para construir modelos preditivos.

4.2.1 Zona de Aprendizagem do Aluno

O presente trabalho propôs e validou um método para estimar: *a)* se alunos de IPC terão um rendimento baixo ou alto nas avaliações intermediárias; *b)* se estudantes de IPC serão aprovados ou reprovados no final da disciplina; e *c)* qual será a média final de alunos de IPC em uma escala contínua. Neste texto, adotou-se o termo Zona de Aprendizagem (ZA) para representar o rendimento dos estudantes nos itens *a*, *b* e *c*. Destaca-se que o rendimento, mencionado anteriormente, é calculado com base nas notas das avaliações intermediárias (item *a*) ou nas médias finais (item *b* e *c*). A ZA foi dividade em Zona de Expertise (ZE) e Zona de Dificuldade (ZD). O aluno é alocado na ZE quando tem

rendimento maior ou igual a 5, caso contrário na ZD. O processo de predição é realizado usando algoritmos de classificação e regressão:

- Na classificação, as notas dos alunos são primeiramente discretizadas usando a média 5 como limiar. Para ser mais específico, ao aluno que obtivesse uma nota menor que 5 é atribuído o valor 0 (ZD) na base de dados, do contrário, é atribuído 1 (ZE). Dessa forma, o professor poderia visualizar se o aluno iria passar ou não em uma das provas intermediárias e na média final.

- Já na regressão, a saída é contínua e, assim, as notas dos alunos são inferidas de fato. Depois que a regressão é realizada, as notas são normalizadas utilizando o *z-score* dentro de cada zona, isto é, a quantos desvios padrão o aluno está da média daquela zona. Isso é importante pois gera um indicativo para o professor que poderá visualizar, além da nota estimada, o quão bem ou quão mal o aluno está dentro da zona de aprendizagem, conforme Figura 4.5.

Observa-se que a regressão é empregada apenas na tarefa de predição da média final dos alunos. O motivo é que as notas das avaliações intermediárias dos alunos tiveram pouca variação, isto é, mais de 90% dos alunos tiravam 0, 5 ou 10 (distribuição multimodal), uma vez que as provas possuíam 2 questões valendo 5 pontos cada. As poucas exceções em que os estudantes tiravam notas diferentes desses três valores, foram quando eles erravam um ou dois dos casos de teste presentes em uma das questões da prova. Nesse caso, eles ficavam com apenas um percentual dos pontos daquela questão.

4.2.2 Perfil de Programação do Aluno

Para os fins da presente pesquisa, cada aluno é representado por um perfil de programação, que foi compilado baseado numa série de características destiladas das publicações elencadas no capítulo de trabalhos relacionados. Desta forma, um perfil é contextualizado com o comportamento de programação do aluno extraído do processo de desenvolvimento das soluções, quando esse está resolvendo as questões que precedem a prova.

O perfil é formado pelas seguintes métricas de código que compõem uma matriz de características, onde os alunos são as linhas e as métricas são as colunas:

- **attempts** (M1): número de tentativas de submissão, independente se o código está certo ou errado (AHADI; VIHAVAINEN; LISTER, 2016);

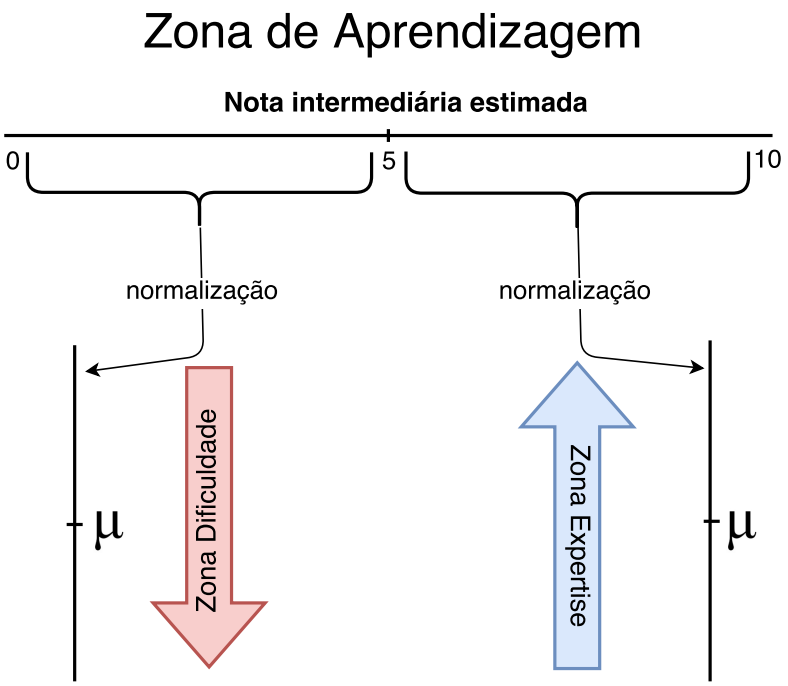


Figura 4.5: Zona de aprendizagem do aluno discriminada em Dificuldade X Expertise.

- **comments** (M2): média de número de linhas com comentários nos códigos submetidos (OTERO et al., 2016);
- **blank_line** (M3): média de número de linhas em branco nos códigos (OTERO et al., 2016);
- **lloc** (M4): média de número de linhas lógicas dos códigos submetidos, sem contabilizar importação de bibliotecas, comentários e linhas em branco (OTERO et al., 2016);
- **loc** (M5): média de número de linhas dos códigos submetidos (OTERO et al., 2016);
- **single_comments** (M6): média de comentários simples. No caso do Python, comentários que usam o caractere # (OTERO et al., 2016);
- **system_access** (M7): número de acessos (logins) que cada aluno faz dentro do prazo de entrega de uma dada lista;
- **exam_grade_codebench** (M8): nota na avaliação realizada em cada sessão.
- **difficult** (M9): foi requisitado do aluno que reportasse o grau de dificuldade de cada questão das listas de exercícios. Nessa ocasião, o aluno poderia escolher para cada

questão um dentre os seguintes graus de dificuldade: 0 - Questão fácil, 1 - Questão de dificuldade mediana, e 2 - Questão difícil. Esta evidência mostra a média dos graus de dificuldade escolhidos por um dado aluno para as questões de uma lista;

- **delete_average** (M10): média de caracteres apagados através do uso das teclas *delete* e *backspace* em cada lista de exercícios;

- **average_log_rows** (M11): média do número de linhas do log gerado por cada questão de uma dada lista de exercícios (CASTRO-WUNSCH; AHADI; PETERSEN, 2017; AHADI et al., 2015)[4];

- **submission_per_exercice** (M12): média de submissões por exercícios, independente se o código estava certo ou errado.

- **sucess_average** (M13): razão entre o número de questões corretamente avaliadas pelo número de tentativas de submissão (AUVINEN, 2015; ESTEY; COADY, 2016);

- **test_average** (M14): número total de testes feitos durante as tentativas de solucionar todas as questões de uma lista dividido pelo número total de questões da lista (ESTEY; COADY, 2016);

- **exercices_list_grade** (M15): nota do aluno em cada lista de exercícios (AHADI; VIHAVAINEN; LISTER, 2016; CASTRO-WUNSCH; AHADI; PETERSEN, 2017; ESTEY; COADY, 2016);

- **exercices_list_grade_check_plagiarism** (M16): nota do aluno em cada lista de exercícios, considerando apenas questões resolvidas que geraram no mínimo 50 linhas no log;

- **copy_past_proportion** (M17): proporção entre número de caracteres colados (com CTRL+V) e número caracteres digitados (MUNSON; ZITOVSKY, 2018);

- **sintaxe_error** (M18): proporção entre o número total de submissões e o número total de submissões com erros de sintaxe (JADUD, 2006b; ESTEY; COADY, 2016; MUNSON; ZITOVSKY, 2018);

[4]Os autores Castro-Wunsch, Ahadi e Petersen (2017), Ahadi et al. (2015) chamaram esse atributo de *number_of_steps*. Entretanto, diferente deles, não foi utilizada um atributo quantidade de linhas de log para cada questão do curso e sim uma média do número de logs para cada sessão.

- **IDE_usage** (M19): tempo total (em minutos) que o aluno passa dentro do IDE tentando solucionar as questões de uma dada lista de exercícios, ou seja, digitando algo (AUVINEN, 2015; MUNSON; ZITOVSKY, 2018);

- **keystroke_latency** (M20): velocidade de digitação de um dado aluno durante suas tentativas de solucionar as questões de uma dada lista de exercícios (LEINONEN et al., 2016);

Destaca-se que métrica de código é uma terminologia utilizada na área de engenharia de *software* para calcular o esforço e o custo de desenvolvimento de um sistema (OTERO et al., 2016). No contexto da presente pesquisa, as métricas de código mensuram as habilidades do aluno para resolver questões de um juiz *online* e são candidatas a serem utilizadas como atributos de modelos preditivos, conforme realizado por Becker (2016), Leinonen et al. (2016), Otero et al. (2014b), Guerra-hollstein e Brusilovsky (2016), Ahadi et al. (2015), Castro-Wunsch, Ahadi e Petersen (2017). Alguns dos atributos do perfil de programação foram propostos neste trabalho e outros foram encontrados na literatura e elucidados no capítulo 3.

Como Ihantola et al. (2015) apontam a necessidade de métodos preditivos que não sejam sensíveis ao contexto educacional, importa dizer que um dos critérios utilizados para adicionar um atributo ao perfil de programação era se a característica poderia ser implementada em diferentes contextos educacionais, isto é, em turmas de IPC que usaram outra linguagem de programação, outra metodologia de ensino, outro ambiente *online* e etc.

Finalmente, destaca-se que a métrica de código *exam_grade_codebench* (M8) é usada como classe quando são realizadas as predições das notas das avaliações intermediárias aplicadas em cada sessão. Por outro lado, quando é realizada a estimativa da média final, ela é usada apenas como um dos outros atributos.

4.3 Construindo os Pipelines de AM

As etapas realizadas para a aplicação do método preditivo proposto neste estudo podem ser vistas na Figura 4.6. Primeiramente, é realizada a preparação dos dados, onde códigos submetidos e os *logs* dos alunos no ACAC são analisados a fim de gerar os valores

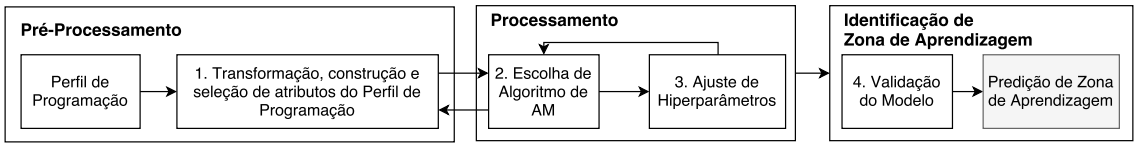


Figura 4.6: Etapas para construção de *pipelines* de AM para a predição da zona de aprendizagem. Adaptado de Pereira, Oliveira e Oliveira (2017).

numéricos que compõem o perfil de programação. Depois disso, é realizado um ciclo entre pré-processamento de atributos, escolha do algoritmo de aprendizagem de máquina e ajuste de hiperparâmetros até se obter um modelo que maximizasse a acurácia na predição da zona de aprendizagem (PEREIRA; OLIVEIRA; OLIVEIRA, 2017). Para a construção dos modelos preditivos são utilizadas duas abordagens: ajuste de hiperparâmetros com *RandomSearch* ou *GridSearch* (A1) e construção e otimização utilizando um algoritmo genético.

Na abordagem A1, os passos 1, 2, 3 e 4 da Figura 4.6 são ajustados de forma exploratória. Para exemplificar, são testados métodos de transformação (descritos na seção 4.3.1.1), construção (descritos na seção 4.3.1.2) e seleção de atributos (descritos na seção 4.3.1.3) e algoritmos de aprendizagem de máquina (descritos na seção 4.3.2), a fim de obter um modelo que se encaixasse aos dados. Após a obtenção do modelo preditivo (etapas 1 e 2 da Figura 4.6), aplica-se uma otimização de hiperparâmetros, a fim de encontrar ajustes promissores nos algoritmos de AM (etapa 3 da Figura 4.6) e, consequentemente, otimizar a acurácia dos modelos preditivos construídos. Finalmente, compara-se os resultados de todos os estimadores e seleciona-se aqueles que apresentaram maior precisão em uma base isolada para teste, sendo esse um processo cíclico.

Vale a pena mencionar que é utilizada a busca aleatória na otimização de hiperparâmetros porque Bergstra e Bengio (2012) explicam que o *RandomSearch* é mais eficiente e pode ter propriedades favoráveis quando comparado com a busca em grade, também conhecida como *GridSearch*. Bergstra e Bengio (2012) também mostram que o *GridSearch* pode ser proibitivamente caro computacionalmente quando o cientista de dados está lidando com vários algoritmos de aprendizagem, hiperparâmetros e conjuntos de dados.

Por outro lado, em relação à abordagem A2, Zutty et al. (2015) demonstrou que a otimização com programação genética pode superar os seres humanos na busca de um *pipeline* de aprendizado de máquinas que melhor se encaixa em uma tarefa de aprendizagem supervisionada. Assim, em A2, usa-se uma abordagem de AM automatizada, como

sugerido por um estudo recente conduzido por Olson et al. (2016) que recomendam o uso da ferramenta de otimização de *pipeline* baseada em árvore (TPOT), que é uma ferramenta de *AutoML*.

Essa biblioteca automatiza e otimiza o projeto de *pipeline* usando algoritmos genéticos em implementações existentes no scikit-learn (PEDREGOSA et al., 2011). O TPOT realiza pré-processamento automático, construção de atributos, seleção de atributos, seleção de modelo e ajuste de hiperparâmetro. Assim, a ideia é encontrar, em um grande espaço de busca, um bom *pipeline* que melhor se adeque aos dados, usando uma pesquisa guiada baseada em uma versão de um algoritmo genético multi-objetivo (DEB et al., 2002) encapsulado pela biblioteca TPOT[5]. Observe que em A2, todo os passos 1, 2 e 3 da Figura 4.6 são automatizados.

A fim de que haja uma análise de qual foi a abordagem superior na aplicação do método proposto, foram definidas uma hipótese nula H_{0A} e uma hipótese alternativa H_{1A}, conforme segue.

H_{0A}: Não há diferença estatística entre as duas abordagens em relação à acurácia dos *pipelines* de AM produzidos.

H_{1A}: Há diferença estatística entre as duas abordagens em relação à acurácia dos *pipelines* de AM produzidos.

4.3.1 Pré-processamento dos Atributos

Em aprendizagem de máquina, uma das tarefas centrais é selecionar e combinar atributos, para que os modelos preditivos obtenham a precisão desejada. Como as instâncias de TPOT (OLSON et al., 2016) realizam o pré-processamento de atributos automaticamente, essa etapa foi realizada manualmente somente em A1 e, assim, as explicações dadas nas seções 4.3.1.1, 4.3.1.2 e 4.3.1.3 se aplicam apenas à abordagem A1.

4.3.1.1 Transformação de Atributos

Para encontrar os subconjuntos de atributos mais relevantes no perfil de programação para cada sessão, os atributos são primeiramente normalizados, já que esse é normalmente um requerimento comum para muitos algoritmos de aprendizagem de máquina. Nesse sentido, é realizada uma normalização, atribuindo zero para o valor médio dos atributos

[5]http://epistasislab.github.io/tpot/

e os demais valores recebem um valor referente a quantidade de desvios padrão que eles estão dessa média (*z-score*). Por exemplo, se um valor estiver a 2.5 desvios padrões a mais que a média então ele receberá 2.5.

4.3.1.2 Construção de Atributos

Após a etapa de transformação dos atributos, realiza-se um processo de construção de atributos, isto é, a criação de novos atributos a partir dos atributos pertencentes ao perfil de programação. Um atributo que é incorporado ao perfil de programação é o *code_ratio* (OTERO et al., 2016), que nada mais é do que uma divisão entre o tamanho médio do código do aluno (M5) pelo tempo médio que ele passava programando na IDE do *Codebench* (M19). Além dele, usa-se os atributos *engajamento* e *coeficiente de variação*, sendo ambos derivados do atributo **attempts** (M1). No engajamento é verificado se, de uma sessão para outra sessão, existe um crescimento no número de tentativas de submissão. Em caso positivo, o atributo recebia 0, do contrário, 1. Partindo do princípio que a complexidade no processo de aprendizagem de programação é algo incremental, então acredita-se que esse atributo poderia representar superficialmente o engajamento do aluno ao tentar muitas vezes resolver as questões. Já o coeficiente de variação nada mais é do que a divisão entre o desvio padrão do **attempts** em uma sessão pelo a média aritmética do **attempts**.

4.3.1.3 Seleção de Atributos

A seleção de atributos é realizada com um dos seguintes algoritmos: a) *SelectKBest* com a função *f-classif* que calcula o ANOVA *F-value* das amostras; b) *Recursive Feature Elimination* (RFE) e c) *BestFirst* que constrói subconjuntos a partir do conjunto de atributos e os avalia usando o algoritmo *CfsSubsetEval*.

4.3.2 Escolha do Algoritmo de AM

Os algoritmos empregados para classificação são o *Random Forest* (RF), *Support Vector Machine* (SVM), *K-Nearest Neighbor* (KNN), *Árvore de Decisão* (AD), *Extremely Randomized Trees* (ERT), *Gradient Tree Boosting* (GTB) e *AdaBoosting* (AB). Quando a predição da média final dos alunos foi estimada em uma escala contínua, são utilizados os seguintes algoritmos de regressão: Rede Neural Profunda (RNP), *Support Vector*

Regression (SVR), ERT, RF e GTB.

4.4 Baseline

Estey e Coady (2016) apresentam uma ferramenta chamada BitFit que explora a interseção entre a identificação precoce de estudantes com risco de reprovação e a análise *data-driven* de comportamentos de programação de alunos de IPC no BitFit. O BitFit é um ambiente de correção automática de código, semelhante ao CodeBench, que coleta os dados dos alunos enquanto eles resolvem problemas de programação. A maior diferença entre os dois é que no BitFit os alunos podem consumir uma série de dicas cada vez mais próximas da resposta esperada. A última dica é a resposta final, isto é, o código-fonte que resolve a questão.

Nesse contexto, Estey e Coady (2016) revelam que atributos *data-driven* relacionados ao número de compilações, consumo de dicas, desempenho nas listas de exercícios e número de tentativas de submissão podem ser usados para a construção de modelos preditivos precoces, isto é, capazes de inferir o desempenho dos alunos ainda nas primeiras semanas de aula.

Os atributos utilizados por Estey e Coady (2016) foram empregados na base de dados apresentada neste estudo e incorporados ao perfil de programação (atributos M1, M13, M14, M15 e M18). Assim, os atributos de Estey e Coady (2016) são uma versão reduzida do perfil de programação, isto é, com menos atributos.

4.5 Comparação com Baseline

A comparação entre os atributos do perfil de programação e os atributos propostos por Estey e Coady (2016) é realizada para analisar se uma versão reduzida do perfil de programação pode ser utilizada para realizar a predição da zona de aprendizagem dos alunos no contexto educacional apresentado nesta pesquisa.

Para verificar se houve diferença estatística entre os atributos propostos por Estey e Coady (2016) e o perfil de programação proposto neste estudo, foram definidas uma hipótese nula H_{0B} e uma hipótese alternativa H_{1B}, conforme segue.

H_{0B}: Não há diferença estatística entre os atributos propostos por Estey e Coady

(2016) e os atributos do perfil de programação em relação à acurácia dos *pipelines* de AM empregando tais atributos.

H_{1B}: Há diferença estatística entre os atributos propostos por Estey e Coady (2016) e os atributos do perfil de programação em relação à acurácia dos *pipelines* de AM empregando tais atributos.

4.6 Testes Estatísticos

Para identificar qual foi a abordagem (A1 ou A2) superior nos experimentos, foram definidas uma hipótese nula H_{0A} e uma hipótese alternativa H_{1A}. Por outro lado, para verificar se houve diferença estatística entre os atributos propostos por Estey e Coady (2016) e o perfil de programação proposto neste estudo, foi definida outra hipótese nula H_{0B} e hipótese alternativa H_{1B}.

No primeiro caso, a hipótese nula é confirmada se os *pipelines* de AM produzidos em ambas as abordagens (H_{0A}) ou métodos (H_{0B}) obtiverem precisões similares. Do contrário, a hipótese nula é refutada. Em caso de refutação, temos que uma abordagem (H_{0B}) ou um método (H_{1B}) foi superior a outra naquele experimento. Analogamente, esse processo é realizado para as hipóteses H_{0B} e H_{1B}

O teste estatístico foi realizado com base na média e desvio padrão de 10 testes executados para calcular a acurácia dos modelos. Em todos os experimentos utilizou-se um nível de confiança de 95% e aplicou-se o teste estatístico *t-test* assumindo que as distribuições eram normais.

4.7 Considerações Finais do Capítulo

No presente capítulo, apresentou-se o método proposto da pesquisa para atingir o objetivo definido e o cenário que o método preditivo foi validado. Com base no estado da arte, elucidou-se quais evidências extraídas de juízes *online* possivelmente se relacionam com a nota do aluno e podem ser usadas como atributos de algoritmos de AM. Tais atributos formam um conjunto de valores numéricos chamado de perfil de programação dos estudantes. Além disso, evidenciou-se como os alunos são segregados na zona de aprendizagem definida neste estudo.

No próximo capítulo será explicado como o perfil de programação foi empregado para a construção dos modelos preditivos que realizaram a inferência da zona de aprendizagem dos alunos. Os modelos que apresentaram melhores resultados foram comparados com um método concebido em um cenário similar a este, proposto por Estey e Coady (2016).

Capítulo 5

Resultados Experimentais

No presente capítulo será apresentado o planejamento e a execução de 6 experimentos que convergem para o objetivo deste estudo que é realizar a predição da zona de aprendizagem dos alunos de turmas de IPC. Frisa-se que uma das metas deste estudo é realizar essa estimativa ainda nas primeiras semanas de aula, a fim de que o professor possa tomar alguma medida de precaução para evitar a reprovação de alunos em risco. Além disso, é desejado investigar até que ponto do curso os modelos preditivos vão ficando mais precisos e qual a influência dos atributos do perfil de programação na precisão dos modelos preditivos.

A seção 5.1 detalhará a metologia empregada nos experimentos, enquanto da seção 5.2 a 5.7 serão descritos os resultados de cada experimento conduzido neste estudo. A discussão e análise mais aprofundada dos resultados experimentais será realizada na seção que responde às questões de pesquisa, seção 5.8, visto que todos os experimentos foram conduzidos a fim de responder tais questões.

5.1 Metodologia de Experimentação

Nos experimentos em que houve a predição das notas discretizadas das avaliações intermediárias, todos os alunos que faltaram essa prova foram removidos. Quando usou-se a média final como classe, os alunos que trancaram a disciplina foram removidos da base. Além disso, alunos que não geravam nenhum *log* em uma sessão foram removidos daquela sessão. Em função disso, não foram utilizados todos os dados dos 486 alunos em todos os experimentos, mas apenas os dados dos alunos ativos, isto é, dos alunos que ao

menos tentavam resolver algum exercício na IDE do Codebench.

Foram conduzidos 6 experimentos neste estudo, chamados de E1, E2, E3, E4, E5 e E6. Nos Experimentos E1, E2 e E3 a nota discretizada da avaliação intermediária foi utilizada como classe para a predição da zona de aprendizagem. Já nos experimentos E4, E5 e E6 a média final foi utilizada como classe.

Nos experimentos onde foram empregados algoritmos de classificação, isto é nos experimentos E1 ao E5, as notas dos alunos (nas avaliações intermediárias e nas médias finais) foram discretizadas utilizando 5 como limiar. Assim, os alunos que alcançassem notas maiores ou iguais a 5 foram alocados na zona de expertise, do contrário, na zona de dificuldade. Apenas no E6 foi realizada a regressão, isto é, as médias finais dos alunos foram estimadas em uma escala contínua. A justificativa para a não estimativa em escala contínua das notas das avaliações intermediárias se deu em função da baixa variação dessas notas.

Cada experimento foi conduzido usando duas abordagens, A1 e A2, onde a primeira usou técnicas de AM tradicional e a segunda empregou técnicas de AM e otimização com um método que emprega algoritmos genéticos proposto por Olson et al. (2016). Os experimentos que usaram A2 foram executados com um tamanho da população de 2500 e 300 minutos como tempo limite de pesquisa para o melhor *pipeline*. O número de gerações foi limitado pelo tempo e a taxa de mutação foi definida para 0,9 e a taxa de cruzamento para 0,1 (conforme recomendado pela documentação do TPOT). Observa-se que, nesse contexto, um indivíduo da população é representado por um *pipeline* de AM.

Ressalta-se que para responder às questões de pesquisa, fazia-se necessário conduzir experimentos utilizando apenas os dados das sessões iniciais, a fim de analisar o poder preditivo do perfil de programação com dados ainda do início do curso. Além disso, era necessário conduzir experimentos onde os algoritmos de aprendizagem de máquina fossem treinados com os dados de cada sessão do curso, com o intuito de investigar até que ponto do curso os modelos preditivos vão ficando mais precisos na predição da zona de aprendizagem dos alunos e, bem como, quais são os atributos mais relevantes do perfil de programação ao longo do curso. Com efeito, nos experimentos E1, E2 e E3 foram realizados testes com os dados das 4 primeiras sessões. No E4, apenas os dados da primeira sessão, isto é, das duas primeiras semanas de aula foram usados para fazer a predição da média final discretizada. No E5 e E6 foram realizados testes com os dados de cada sessão,

isto é, com as 7 sessões.

Resumidamente, a principal diferença entre os experimentos está em:

1. como os dados foram divididos para treinamento e para teste, o que será explicado com mais detalhes na seção 5.1.1;

2. qual atributo foi utilizado como classe (avaliações intermediárias ou média final).

5.1.1 Avaliando os Pipelines de AM

Inicialmente, a base de dados estava desbalanceada, já que havia mais estudantes que passaram do que reprovaram. Dessa forma, houve um processo de subamostragem, onde removeu-se aleatoriamente instâncias da classe majoritária até que a base de dados estivesse equilibrada. Assim, todos os experimentos, com exceção de um teste no E1 e do experimento E6, foram conduzidos em uma base de dados balanceada, a fim de diminuir o viés dos classificadores. O E6 foi conduzido com a base desbalanceada porque foram empregados algoritmos de regressão. Por outro lado, no teste realizado no E1, cujo preditor desempenhava uma tarefa de classificação, percebeu-se que mesmo atribuindo ao classificador um peso maior à classe minoritária, os modelos preditivos ainda tendiam a estimar com mais frequência na classe majoritária.

Os modelos preditivos que realizaram classificação foram avaliados utilizando as seguintes métricas estatísticas: *True Positive Rate* (TPR), *True Negative Rate* (TNR), *Precision* e acurácia (Acc.). Já os modelos de regressão foram avaliados usando o *Root Mean Square Error* (RMSE).

5.1.2 Descrição Geral dos Experimentos

Na descrição de cada experimento, há uma tabela (Tabela 5.1, Tabela 5.4, Tabela 5.7, Tabela 5.10, Tabela 5.12 e Tabela 5.1) que apresenta uma visão geral e resumida (um guia rápido) de como foram conduzidos os 6 experimentos em relação aos objetivos, como foram empregados os atributos do perfil de programação, ao balanceamento da base de dados, ao pré-processamento dos atributos do perfil de programação, aos ajustes de hiperparâmetros, à otimização de *pipelines* AM com algoritmos genéticos, à avaliação dos modelos preditivos e a divisão da base em relação ao treino e teste. Tais tabelas estão alocadas nas seções que descrevem cada experimento. Nas próximas seções serão

detalhados como foram planejados e executados os experimentos deste estudo em termos de construção dos *pipelines* de AM e de como o método proposto foi comparado com o *baseline*.

5.2 Experimento 1

No E1, os modelos preditivos foram validados aplicando o método de validação cruzada com 10 partições (*folds*). Os atributos das sessões anteriores foram transformados em novos atributos das sessões seguintes. Por exemplo, quando o experimento foi conduzido na S2, a nota da lista na S1 foi usada como atributo *nota_lista_s1* e a nota da lista da própria S2 foi usada como *nota_lista_s2*. Isso aumentou substancialmente o número de colunas das sessões diferentes da primeira, mas diminuiu o número de instâncias no decorrer das sessões. Para ilustrar, um aluno podia ter estado ativo (resolvendo exercícios e fazendo as avaliações intermediárias) nas sessões S1 e S2, mas ter faltado a prova da S3. Nesse caso, ele era removido do experimento na S3.

Além disso, neste experimento foi realizado um teste com a base desbalanceada e outro teste que a base balanceada. A Figura 5.1 apresenta a quantidade de alunos com notas abaixo de 5 (ZD) e a quantidade de estudantes com notas maiores ou igual a 5 (ZE) em cada uma das 4 primeiras sessões deste experimento. Destaca-se que esse foi o único experimento no qual foi calculado a métrica *precision*, uma vez que apenas nele foram realizados testes com a base desbalanceada. A Tabela 5.1 apresenta uma visão geral do experimento E1.

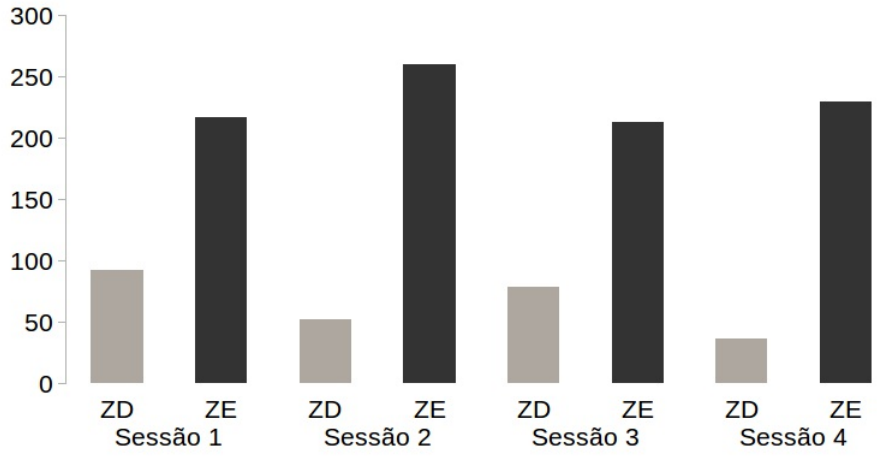


Figura 5.1: Quantidade de alunos na ZE e ZD em cada sessão do E1.

Tabela 5.1: Visão geral de como foi conduzido o experimento E1.

Experimento 1:	
Objetivo:	predição do valor discretizado das avaliações intermediárias das sessões S1 até S4
Perfil de Programação:	atributos das sessões anteriores eram transformados em novos atributos das sessões seguintes
Balanceamento da Base:	Testes com a base desbalanceada e balanceada
Pré-processamento de Atributos	
Transformação de Atributos:	não houve
Construção de Atributos:	Acrescentou-se o atributo code_ratio (M5 dividido por M19)
Seleção de Atributos:	*BestFirst* para busca e *CfsSubsetEval* para avaliação dos subconjuntos
Escolha Algoritmo de AM	
Classificação:	RF, SVM, KNN, DT, AB
Ajuste de Hiperparâmetros (A1):	*GridSearch*
Otimização com AG (A2):	não houve
Avaliação dos Modelos Preditivos:	Acc., TNR, TPR, Precision
Divisão Treino e Teste:	validação cruzada (10 folds)

5.2.1 Resultados do Experimento 1

A Tabela 5.2 mostra a acurácia (Acc.), *True Negative Rate* (TNR), *True Positive Rate* (TPR) e o *precision*[1] dos modelos preditivos construídos para predição da zona de aprendizagem dos alunos em cada sessão. As maiores acurácias dos modelos preditivos foram nas sessões 2 e 4, que são as mais desbalanceadas dessa base. Destaca-se que os modelos estão conseguindo identificar os alunos da ZE de forma contundente, já que o TPR e o

[1]Positivos são os alunos da ZE e negativos os alunos da ZD

precision positivo também são valores expressivos (PEREIRA; OLIVEIRA; OLIVEIRA, 2017).

O baixo número de alunos na ZD justifica o TNR menor do que o TPR nas sessões, pois os algoritmos tinham poucos exemplos nessa zona para aprender os seus padrões, assim a tendência era estimar na classe majoritária. Entretanto, frisa-se que o *precision* negativo é relevante [65.2% - 72.7%] (com exceção da sessão 4), o que mostra que ao identificar um aluno na ZD as chances eram altas do modelo preditivo estar certo (PEREIRA; OLIVEIRA; OLIVEIRA, 2017).

Tabela 5.2: Apresentação da acurácia, TPR, *precision* dos positivos, TNR, *precision* dos negativos dos algoritmos de aprendizagem de máquina com a base desbalanceada. Os algoritmos utilizados foram *Support Vector Machine* (SVM), Random Forest (RF), Árvore de Decisão (AD) e AdaBoost (AB) com um AD como estimador base. Melhores resultados em negrito.

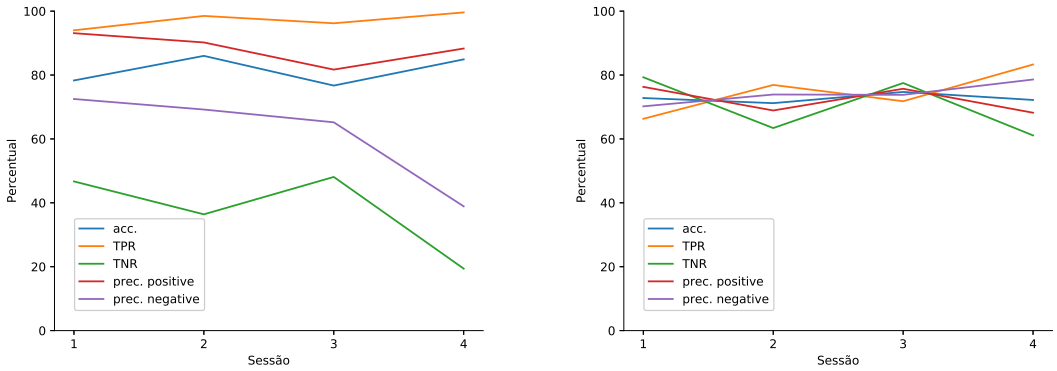
	Sessão 1				Sessão 2				Sessão 3				Sessão 4			
	SVM	RF	AD	AB	SVM	RF	AD	AB	SVM	RF	AD	AB	SVM	RF	AD	AB
Acc.(%)	71,5	74,1	**78,3**	73,1	84,9	**86,0**	84,0	84,3	75,3	**76,7**	72,2	75,6	83.8	**84,9**	78,6	86,5
TPR(%)	**94,0**	90,3	85,8	84,3	**98,5**	96,2	95,8	96,2	**96,2**	92,5	85,0	85,9	95,7	95,2	89,6	**99,6**
Prec. Pos.(%)	73,1	76,9	**93,1**	78,9	85,6	**90,2**	85,6	86,5	76,2	78,8	78,7	**81,7**	87,0	**88,3**	86,2	86,7
TNR(%)	18,5	35,9	43,5	**46,7**	17,3	**36,4**	25,0	25,0	19,0	32,9	38,0	**48,1**	8,3	**19,4**	8,3	3
Prec. Neg.(%)	56,7	61,1	**72,7**	55,8	**69,2**	61,6	54,2	56,5	**65,2**	61,9	48,4	55,9	23,1	**38,9**	11,1	50

Por outro lado, foi realizado outro teste com a base de dados balanceada. A Tabela 5.3 mostra a avaliação dos modelos preditivos com maior acurácia nesse teste. Os algoritmos com maior acurácia para cada sessão foram o SVM, RF, RF e KNN, respectivamente, para cada sessão. Vale ressaltar que, em linhas gerais, o RF foi o algoritmo mais consistente nas sessões estudadas neste experimento. Percebe-se que os valores da acurácia diminuíram em relação ao teste com a base desbalanceada, enquanto que os valores do TNR e *precision* negativo subiram. De fato as métricas de avaliação dos modelos preditivos se equilibraram no teste com a base balanceada, como pode-se ver na Figura 5.2, que mostra o desempenho nos dois testes. A explicação para esse fenômeno se dá pelo fato da ausência do viés dos algoritmos em escolherem a classe majoritária. Além disso, a acurácia pode ter diminuído pelo fato de ter menos instâncias e, consequentemente, menos exemplos para treinar (PEREIRA; OLIVEIRA; OLIVEIRA, 2017). Os resultados deste experimento foram publicados no Simpósio Brasileiro de Informática na Educação (SBIE).

Após a realização deste experimento e inúmeras tentativas de treinamento com a base desbalanceada, percebeu-se que a opção mais viável era trabalhar apenas com a base

Tabela 5.3: Avaliação dos modelos preditivos mais precisos com a base de dados balanceada. Os melhores resultados foram obtidos com SVM, RF, RF e KNN, respectivamente, em cada sessão.

Sessão	Acc.	TPR	Prec. Pos.	TNR	Prec. Neg.
S1	72,8%	66,3%	76,3%	79,3%	70,2%
S2	71,2%	76,9%	68,9%	63,4%	73,9%
S3	74,7%	71,8%	75,7%	77,5%	73,8%
S4	72,2%	83,3%	68,2%	61,1%	78,6%



(a) Resultados base desbalanceada.

(b) Resultados base balanceada.

Figura 5.2: Comparação dos melhores resultados obtidos no E1 com a base balanceada e desbalanceada.

balanceada, visto que o modelo preditivo fica menos suscetível a optar arbitrariamente pela classe majoritária com o intuito de maximizar a acurácia, conforme apresentado na Figura 5.2.

5.3 Experimento 2

No E2, a base estava balanceada e foram usados os dados de uma sessão inteira para treinamento e da sessão subsequente para teste. Para ilustrar, foram usados os dados de S1 para estimar o **exam_codebench** em S2. Assim, neste experimento havia mais instâncias para o algoritmo de AM treinar do que no E1. A Figura 5.3 mostra a quantidade de alunos em cada sessão, após o balanceamento do base. Note que essa quantidade de instâncias apresentada na Figura 5.3 vale para o E2 e E3.

A Tabela 5.4 apresenta uma visão geral do experimento E2.

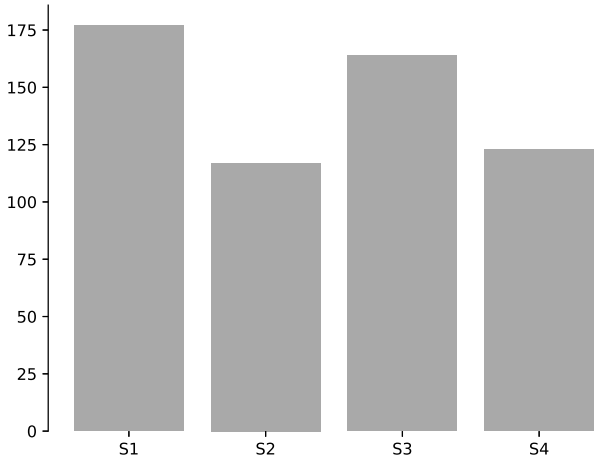


Figura 5.3: Quantidade de alunos em cada sessão depois do balanceamento da base no E2 e E3, sendo metade dos alunos na ZD e a outra metade na ZE.

Tabela 5.4: Visão geral de como foi conduzido o experimento E2.

Experimento 2:	
Objetivo:	predição do valor discretizado das avaliações intermediárias das sessões S1 até S4
Perfil de Programação:	não houve alterações
Balanceamento da Base:	base balanceada
Pré-processamento de Atributos	
Transformação de Atributos:	não houve
Construção de Atributos:	code_ratio, engajamento e coeficiente de variação (sendo os dois últimos baseados no atributo M1)
Seleção de Atributos:	testes com RFE, *SelectkBest*
Escolha Algoritmo de AM	
Classificação:	RF, SVM, KNN, ERT, GTB
Ajuste de Hiperparâmetros (A1):	*RandomSearch*
Otimização com AG (A2):	sim

Avaliação dos Modelos Preditivos:	Acc., TNR, TPR
Divisão Treino e Teste:	dados de uma sessão inteira para treinamento e da sessão subsequente para teste

5.3.1 Resultados do Experimento 2

A Tabela 5.5 mostra a avaliação dos modelos que melhores se encaixaram aos dados, usando as abordagens A1 e A2. Em geral, os métodos de *ensembles* baseados em árvores de decisão obtiveram as maiores acurácias em ambas abordagens.

Tabela 5.5: Resultados do E2, usando as duas abordagens. Melhores resultados estão em negrito.

	A1			A2		
	Acc.	**TPR**	**TNR**	**Acc.**	**TPR**	**TNR**
S2	71,62%	83,33%	60,53%	**75,68%**	**82,05%**	**68,57%**
S3	72,15%	68,35%	**75,95%**	**74,49%**	**81,63%**	67,35%
S4	72,50%	65,00%	**80,00%**	**72,73%**	**88,89%**	61,54%

Especificamente usando A1, os melhores resultados foram obtidos com RF em S2, ERT em S3 e KNN em S4. Por outro lado, usando A2, os melhores resultados foram obtidos com GTB em S2 e ERT em S3 e S4. Em todas as sessões, as maiores acurácias e TPR foram obtidas usando A2. Apenas o TNR em S3 e S4 usando A1 foram superiores ao A2.

Analisando as hipóteses H_{0A} e H_{1A}, podemos verificar na Tabela 5.6 que a acurácia obtida com o emprego de A2 foi estatisticamente superior a A1 em S2 e S3, o que nos leva a refutar a hipótese nula neste experimento apenas para essas duas sessões. Esse resultado é expressivo, pois mostra que a hipótese de Olson et al. (2016) (abordagem A2) de que é possível construir pipelines de AM automaticamente utilizando um algoritmo genético se mostrou promissora neste experimento e chegou a ser estatisticamente superior aos modelos construídos manualmente (abordagem A1) em duas sessões. Entretanto, tal superioridade só foi alcançada depois de alguns ajustes manuais realizados nos *pipelines* exportados na A2.

Finalmente, se compararmos os melhores resultados em E2 com os de E1 (teste com

Tabela 5.6: Teste estatístico no E2 para verificar se houve diferença estatística entre as duas abordagens.

	S2	S3	S4
p-value	0,0022	0,0428	0,83

a base balanceada), observa-se que os dois experimentos obtiveram resultados similares em todas as sessões. Entretanto, no E2 o professor recebe a informação da zona de aprendizagem do aluno com uma antecedência maior se comparado ao E1, conforme explicado na seção 5.1. Com efeito, a estratégia de manipulação do perfil de programação adotada no E1 de usar os atributos das sessões anteriores como novos atributos nas sessões posteriores não foi mais adotada visto que não seria possível treinar o modelo com os dados de uma sessão e testar em outra sessão, já que em AM o número de atributos na base de treino tem que ser o mesmo na base de teste. Além disso, a estratégia utilizada no E1 se mostrou ineficaz, pois além de diminuir o número de instâncias nas sessões mais próximas do fim do curso, os atributos das sessões anteriores acabam ficando redundantes com as das sessões correntes.

5.4 Experimento 3

No E3, a base estava balanceada (Figura 5.3). Foram usados apenas os dados da S1 inteira para treinamento e a predição foi realizada nas sessões S3 e S4. A objetivo principal era verificar se apenas os dados das duas primeiras semanas de aula (S1) são suficientes para realizar a predição do valor discretizado das avaliações intermediárias nas sessões seguintes, ou seja, investigar a possibilidade de construir preditores precoces capazes de prever a zona de aprendizagem dos alunos ainda no início do curso. A Tabela 5.7 apresenta uma visão geral do experimento E3.

Tabela 5.7: Visão geral de como foi conduzido o experimento E3.

Experimento 3:	
Objetivo:	predição do valor discretizado das avaliações intermediárias das sessões S3 e S4 usando apenas os dados do S1 para treinamento

Perfil de Programação:	não houve alterações
Balanceamento da Base:	base balanceada
Pré-processamento de Atributos	
Transformação de Atributos:	Normalização
Construção de Atributos:	code_ratio, engajamento e coeficiente de variação
Seleção de Atributos:	testes com RFE, *SelectkBest*
Escolha Algoritmo de AM	
Classificação:	RF, SVM, KNN, ERT, GTB
Ajuste de Hiperparâmetros (A1):	*RandomSearch*
Otimização com AG (A2):	sim
Avaliação dos Modelos Preditivos:	Acc., TNR, TPR
Divisão Treino e Teste:	treinamento apenas com os dados da S1 e teste em S3 e S4

5.4.1 Resultados do Experimento 3

A Tabela 5.8 apresenta os resultados do E3. Note que não existe os resultados da S2, visto que eles são os mesmos apresentados no E2 (Tabela 5.5), uma vez que o treino ocorre no S1 e o teste no S2. Na S3 e S4, quase todos os resultados que empregaram a abordagem A1 foram superiores, com exceção do TPR na S3.

Os algoritmos que atingiram os melhores resultados em A1 foram RF e KNN nas sessões S3 e S4, respectivamente. Enquanto que em A2, os melhores resultados foram obtidos com GTB e ERT nas sessões S3 e S4, respectivamente.

Analisando as hipóteses H_{0A} e H_{1A}, podemos verificar na Tabela 5.9 que a acurácia obtida com o emprego de A1 foi estatisticamente superior a A2 em todos os testes, o que nos leva a refutar a hipótese nula neste experimento.

Note ainda que neste experimento foram obtidos resultados estatisticamente superiores aos do E1 (*p-value*<0,05), o que nos dá mais evidências que o uso da estratégia

Tabela 5.8: Resultados do E3, usando A1 e A2. Melhores resultados estão em negrito. Note que os resultados obtidos na S2 são os mesmos apresentados na Tabela 5.8, uma vez que foram usados os dados da S1 para treinamento e S2 para teste.

	A1			A2		
	Acc.	**TPR**	**TNR**	**Acc.**	**TPR**	**TNR**
S3	**78,21%**	68,75%	**84,78%**	75,00%	**73,13%**	77,05%
S4	**78,15%**	**88,24%**	**66,67%**	71,86%	82,35%	60,00%

Tabela 5.9: Teste estatístico no E3 para verificar se houve diferença estatística entre as duas abordagens.

	S3	S4
p-value	0,0278	0,0002

estabelecida no E1 foi ineficaz em relação a maximização da acurácia dos modelos preditivos. Ademais, inesperadamente, os resultados deste experimento superaram os valores obtidos no E2. Uma explicação para esse fenômeno pode ser dada baseada no comportamento dos alunos de IPC representado pelo perfil de programação, pois nas sessões S1, S3 e S4 esse comportamento foi similar, em contraste ao S2, onde houve uma diferença menor entre os alunos da ZD e ZE em relação aos atributos do perfil de programação, isto é, em S1, S3 e S4 os alunos de cada zona de aprendizagem (ZD e ZE) mantiveram um padrão similar de interação com o CodeBench, enquanto que na S2 esse padrão foi menos acentuado. Esse padrão será detalhado na Seção 5.8.

5.5 Experimento 4

No E4, foram utilizados apenas os dados do S1 para treino e teste. Entretanto, diferentemente do E3, a zona de aprendizagem foi representada pela média final discretizada. Assim, o objetivo neste caso foi de prever a média final dos alunos usando apenas os dados das duas primeiras semanas de aula. Para tanto, foram usadas as abordagens A1 e A2. A quantidade de instâncias utilizadas neste experimento pode ser vista na barra S1 da Figura 5.4. A Tabela 5.10 apresenta uma visão geral do experimento E4.

Tabela 5.10: Visão geral de como foi conduzido o experimento E4.

Experimento 4:

Objetivo:	predição do valor discretizado da média final dos alunos usando apenas a S1
Perfil de Programação:	não houve alterações
Balanceamento da Base:	base balanceada
Pré-processamento de Atributos	
Transformação de Atributos:	Normalização
Construção de Atributos:	code_ratio, engajamento e coeficiente de variação
Seleção de Atributos:	testes com RFE, *SelectkBest*
Escolha Algoritmo de AM	
Classificação:	RF, SVM, KNN, ERT, GTB
Ajuste de Hiperparâmetros (A1):	*RandomSearch*
Otimização com AG (A2):	sim
Avaliação dos Modelos Preditivos:	Acc., TNR, TPR
Divisão Treino e Teste:	65% da S1 para treinamento e 35% para teste

5.5.1 Resultados do Experimento 4

A Tabela 5.11 mostra os resultados obtidos. Com exceção do TPR, a A1 foi superior a A2. Analisando as hipóteses H_{0A} e H_{1A}, a acurácia obtida com o emprego de A1 foi estatisticamente superior a A2 (*p-value* = 0.0005), o que nos leva a refutar a hipótese nula neste experimento. Os algoritmos que atingiram os melhores resultados em A1 e A2 foram ERT e GTB, respectivamente.

Tabela 5.11: Resultados do E4 em cada sessão usando as abordagens A1 e A2. Melhores resultados estão em negrito.

	A1			A2		
	Acc.	**TPR**	**TNR**	**Acc.**	**TPR**	**TNR**
S1	**74,44%**	71,88%	**76,81%**	69,93%	**73,44%**	66,67%

5.6 Experimento 5

O E5 é uma extensão do E4. Foram realizados testes para predição da média final discretizada com os dados da S2 até a S7. Cada sessão foi estratificada em 65% para treino e 35% para teste. A intenção deste experimento é verificar se a precisão dos modelos preditivos é incrementada no decorrer das sessões. Para tanto, foram empregadas as abordagens A1 e A2 na construções dos *pipelines* de AM. Assim como no E4, o treino e teste foi realizado na mesma sessão. A quantidade de instâncias em cada sessão deste experimento pode ser vista na Figura 5.4. Note que o número de alunos vai diminuindo no decorrer das sessões porque houve subamostragem para o balanceamento da base.

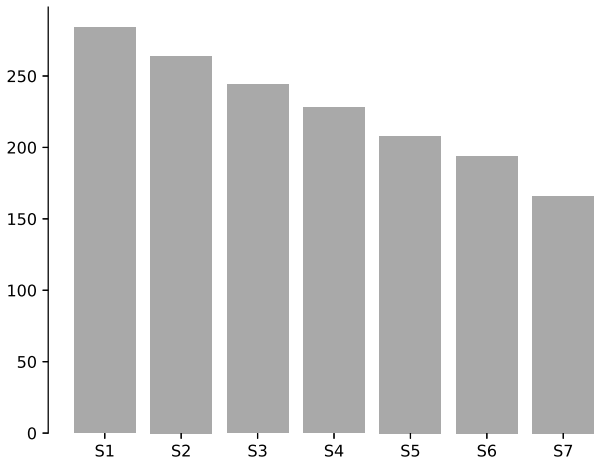


Figura 5.4: Quantidade de alunos em cada sessão depois do balanceamento da base no E4 e E5, sendo metade dos alunos na ZD e a outra metade na ZE.

A Tabela 5.12 apresenta uma visão geral do experimento E5.

Tabela 5.12: Visão geral de como foi conduzido o experimento E5.

Experimento 5:	
Objetivo:	predição do valor discretizado da média final dos alunos em todas as sessões
Perfil de Programação:	não houve alterações
Balanceamento da Base:	base balanceada

Pré-processamento de Atributos	
Transformação de Atributos:	Normalização
Construção de Atributos:	code_ratio, engajamento e coeficiente de variação
Seleção de Atributos:	testes com RFE, *SelectkBest*
Escolha Algoritmo de AM	
Classificação:	RF, SVM, KNN, ERT, GTB
Ajuste de Hiperparâmetros (A1):	*RandomSearch*
Otimização com AG (A2):	sim
Avaliação dos Modelos Preditivos:	Acc., TNR, TPR
Divisão Treino e Teste:	65% para treinamento e 35% para os estimadores construídos com dados de uma sessão.

5.6.1 Resultados do Experimento 5

A Tabela 5.13 mostra os resultados obtidos nas sessões investigadas neste experimento. A A1 foi superior a A2 em termos de acurácia e TPR em todas as sessões. Quanto ao TNR, a abordagem A2 levou vantagem nas sessões S2, S4 e S6. Note que a partir da S3 a acurácia obtida fica entre 81,30% e 90,62%, o que mostra que os modelos preditivos alcançaram resultados expressivos. Em outras palavras, a partir da sexta semana de curso o modelo já consegue identificar com uma precisão considerável se o aluno vai passar ou não na disciplina. E mais, a partir da sessão S5 o modelo atinge valores próximos de 90%, o que mostra que o modelo vai realmente ficando mais preciso no decorrer das sessões.

Os algoritmos que obtiveram melhores resultados nas abordagens A1 e A2 podem ser vistos na Tabela 5.14. Em geral, os *ensembles* baseados em árvores de decisão novamente se destacaram. Vale a pena notar que o resultado obtido não depende somente do algoritmo de AM, mas sim de todo o *pipeline* construído.

Analisando as hipóteses H_{0A} e H_{1A}, observa-se na Tabela 5.15 que a acurácia obtida com o emprego de A1 foi estatisticamente superior a A2 nas sessões S3, S4 e S6 (*p-value*<0,05), o que nos leva a refutar a hipótese nula para essas sessões.

Tabela 5.13: Resultados do E5 em cada sessão usando as abordagens A1 e A2. Melhores resultados estão em negrito.

	A1			A2		
	Acc.	**TPR**	**TNR**	**Acc.**	**TPR**	**TNR**
S2	**74,59%**	**76,67%**	72,58%	73,86%	73,33%	**74,42%**
S3	**81,30%**	**83,64%**	**78,85%**	75,30%	75,61%	75,00%
S4	**85,00%**	**93,62%**	77,36%	77,63%	72,50%	**83,33%**
S5	**89,02%**	**90,00%**	**88,10%**	85,51%	85,71%	85,29%
S6	**88,57%**	**94,12%**	83,33%	84,66%	83,87%	**85,29%**
S7	**90,62%**	**85,29%**	**96,67%**	89,09%	85,19%	92,86%

Tabela 5.14: Algoritmos de AM com melhores resultados no E5.

	S2	S3	S4	S5	S6	S7
A1	RF	ERT	ERT	ERT	GTB	RF
A2	RF	ERT	ERT	ERT	RF	ERT

Tabela 5.15: Teste estatístico no E3 para verificar se houve diferença estatística entre as duas abordagens.

	S2	S3	S4	S5	S6	S7
p-value	0,5051	<0,0001	<0,0001	0,4026	0,0093	0,224

5.7 Experimento 6

No E6 foram utilizados os dados de cada sessão para treino e teste. A quantidade de instâncias utilizadas neste experimento pode ser vista na Figura 5.5. O número de alunos vai diminuindo no decorrer das sessões de forma menos acentuada do que na Figura 5.4. A razão disso, é que não houve subamostragem, assim, o motivo da redução de instâncias é somente que à medida que o curso vai passando mais alunos vão evadindo, não gerando nenhum *log* para a construção do seu respectivo perfil de programação.

A principal diferença deste experimento para o E5 é que a zona de aprendizagem foi representada pela média final dos alunos não discretizada. Além disso, a predição da zona de aprendizagem foi realizada utilizando duas estratégias: a) - treino e teste com os dados de mesma sessão; b) - dados das sessões anteriores a sessão usada para teste eram concatenadas e usados para treino. Observe ainda que neste experimento aplicou-se um modelo de Rede Neural Profunda (RNP), a fim de verificar se, principalmente na estratégia b), as

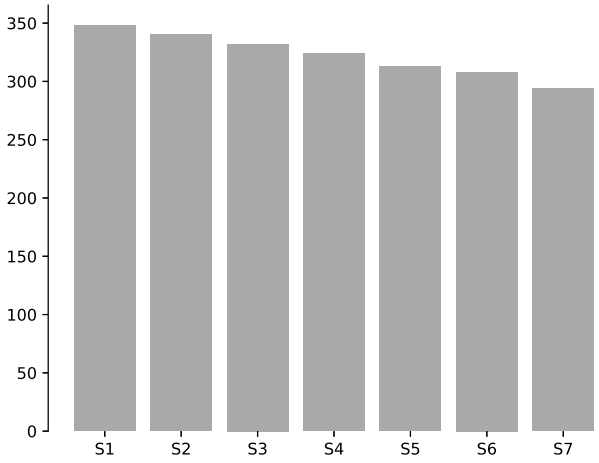


Figura 5.5: Quantidade de alunos em cada sessão no experimento E6.

RNP poderiam superar as duas abordagens. Observa-se que a estratégia b) foi utilizada no experimento E6, a fim de investigar se aumentando os dados de treinamento o uso de redes neurais profundas seria viável em termos de precisão. A Tabela 5.16 apresenta uma visão geral do experimento E6.

Tabela 5.16: Visão geral de como foi conduzido o experimento E6.

Experimento 6:	
Objetivo:	predição da média final dos alunos em todas as sessões
Perfil de Programação:	não houve alterações
Balanceamento da Base:	Não houve balanceamento
Pré-processamento de Atributos	
Transformação de Atributos:	Normalização
Construção de Atributos:	code_ratio, engajamento e coeficiente de variação
Seleção de Atributos:	não houve
Escolha Algoritmo de AM	
Regressão:	RNP, RF, ERT, GTB e SVR

Ajuste de Hiperparâmetros (A1):	não houve
Otimização com AG (A2):	sim
Avaliação dos Modelos Preditivos:	RMSE
Divisão Treino e Teste:	65% para treinamento e 35% para teste. Foram utilizadas duas estratégias de divisão. Na primeira, os estimadores foram construídos com dados de uma sessão. Na segunda, os modelos foram treinados com os dados de todas as sessões anteriores ao teste

5.7.1 Resultados do Experimento 6

Os melhores resultados atingidos com A1, A2 e com RNP, empregando a estratégia de treinamento *a)* podem ser vistos na Tabela 5.17, enquanto que os melhores resultados utilizando a estratégia b) são apresentados na Tabela 5.18.

Tabela 5.17: RMSE do teste no E6 em cada sessão utilizando a estratégia a). Utilizou-se as abordagens A1, A2 e um algoritmo de regressão com Redes Neurais Profundas (RNP). Melhores resultados estão em negrito.

	A1	**A2**	**RNP**
S1	**2,5009**	2,5763	3,3327
S2	**2,3051**	2,4192	2,4390
S3	**2,1383**	2,1408	2,5577
S4	**1,8424**	2,2230	2,2795
S5	**1,6051**	2,0728	1,8312
S6	**1,7543**	1,9019	2,1431
S7	1,4615	**1,4013**	1,6726

Utilizando a estratégia a), a abordagem A1 foi superior em todos as sessões com exceção da S7 na qual a A2 foi superior. Utilizando a estratégia b) a abordagem A1, bem como, foi preponderante. Entretanto nas sessões a partir da S5, onde havia mais instâncias, a RNP obteve resultados similares aos do A1, superando essa abordagem na sessão S5. Note que nas sessões finais existiam mais exemplos para treinar, visto que os dados

Tabela 5.18: RMSE no E6 empregando a estratégia de treino b). Utilizou-se as abordagens A1, A2 e RNP. Melhores resultados estão em negrito.

	A1	**A2**	**RNP**
S2	**2,3832**	2,4524	2,8370
S3	**2,1433**	2,2213	2,2069
S4	**2,0191**	2,0377	2,1062
S5	1,9281	2,0573	**1,9078**
S6	**1,7364**	1,8917	1,7590
S7	**1,5239**	1,6571	1,5713

das sessões anteriores ao teste eram acumulados. Isso justifica o fato do crescimento de performance da RNP, pois redes profundas necessitam de muitos dados para que não haja *overfitting*.

A Tabela 5.19 mostra os algoritmos que obtiveram os melhores resultados nas duas abordagens e em cada uma das estratégias de treino. Em geral, percebe-se que os algoritmos baseados em árvores de decisão novamente foram mais frequentes.

Tabela 5.19: Algoritmos de AM com melhores resultados no E6.

	S1	S2	S3	S4	S5	S6	S7
A1 - Estratégia a	ERT	SVR	ERT	RF	ERT	ERT	ERT
A2 - Estratégia a	ERT	GTB	ERT	GB	GB	ERT	ERT
A1 - Estratégia b		SVR	ERT	ERT	ERT	RF	RF
A2 - Estratégia b		GTB	ERT	ERT	GTB	RF	GTB

Os resultados deste experimento são promissores, pois no E6 o professor tem a média final do aluno com uma margem de erro cada vez menor no decorrer das sessões. Note que como a métrica RMSE aplica a média dos erros das predições e que não houve remoção de *outliers* no treinamento, em alguns momentos o algoritmo fazia estimativas distantes do valor real o que ocasionou um aumento significativo no valor obtido no RMSE, já que médias aritméticas são afetadas significativamente por *outliers*. Entretanto, em geral as notas estimadas tinham uma margem de erro baixa, isto é, as médias finais estimadas eram muito próximas as médias finais reais. Para demonstrar esse fato é possível verificar os erros dos modelos preditivos com melhores resultados em cada sessão na Figura 5.6 e Figura 5.7. Tais gráficos apresentam as diferenças entre os valores estimados e o valores reais das médias finais dos alunos nos testes realizados com as estratégia a e estratégia b,

respectivamente. Note que se um ponto interceptar a linha horizontal em zero, significa que não houve erro. Perceba ainda que a Figura 5.7 possui uma densidade maior de pontos porque na estratégia b usou-se os dados de uma sessão inteira para teste.

Ressalta-se ainda que realizar a predição da média final do aluno em uma escala contínua habilita os professores a tomarem medidas de precaução para estudantes que obtiveram resultados às margens da ZD e ZE, isto é, que tiveram valores estimados próximos da média 5. É possível também identificar o quão mal ou o quão bem um estudantes estão dentro de cada zona de aprendizagem. Observe que os estudantes que ficam na média podem ser encorajados a fim de atingir todo seu potencial. Os estudantes acima da média podem ser ainda mais estimulados com questões e conteúdos ainda mais desafiadores. Enquanto que medidas de precaução podem ser tomadas para que os estudantes com notas abaixo da média não acabem reprovando.

Analisando as hipóteses H_{0A} e H_{1A}, observa-se na Tabela 5.20 que a acurácia obtida com o emprego de A1 com a estratégia a) foi estatisticamente superior a A2 nas sessões S1, S2, S4, S5 e S6 (*p-value*<0,05) e que a A2 foi estatisticamente superior em S7, o que nos leva a refutar a hipótese nula para essas sessões. Já na estratégia b), a abordagem A1 foi estatisticamente superior nas sessões S2, S3, S6 e S7.

Tabela 5.20: Teste estatístico no E6 para verificar se houve diferença estatística entre as abordagens A1 e A2.

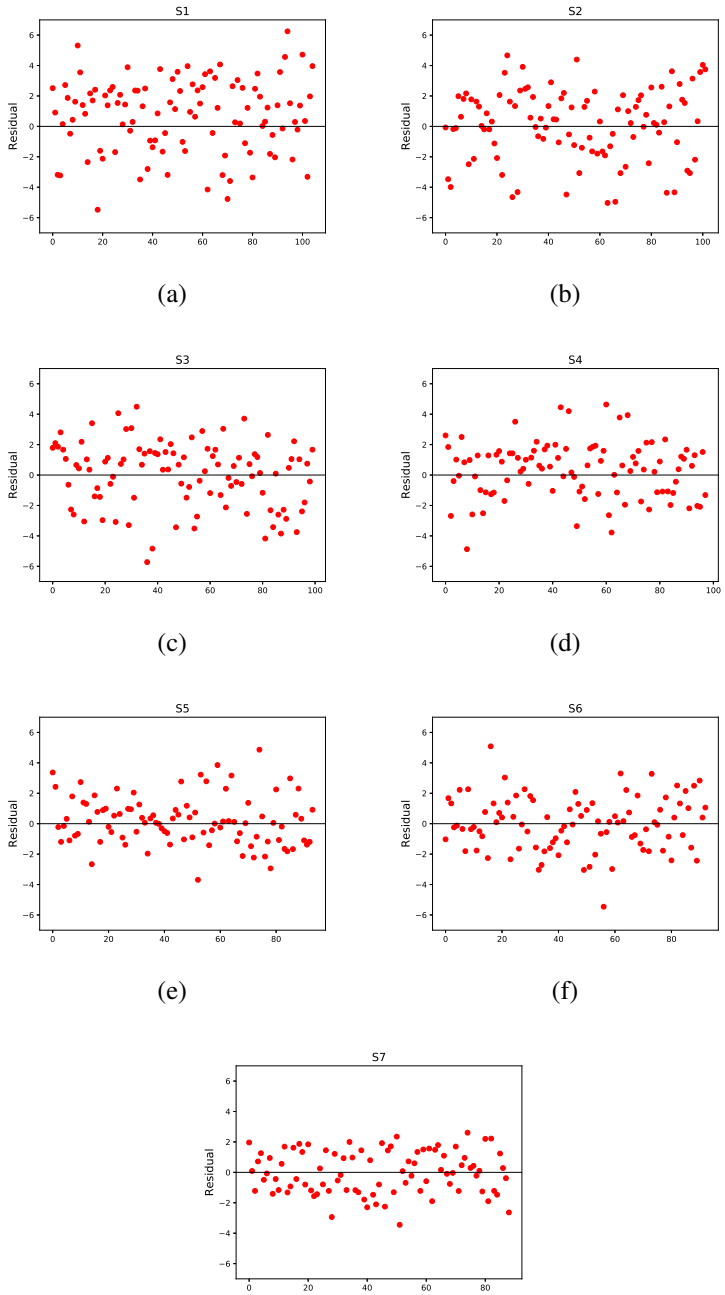
	S1	S2	S3	S4	S5	S6	S7
***p-value* Estratégia a**	0,0031	0,003	0,9957	<0,0001	<0,0001	0,0016	0,0151
***p-value* Estratégia b**		<0,0001	0,0005	0,3265	<0,0001	<0,0001	<0,0001

5.8 Discussão e Respostas às Questões de Pesquisa

Nesta seção os resultados dos 6 experimentos serão discutidos e as questões de pesquisa estabelecidas neste estudo serão respondidas.

Primeiramente, como a ideia do método é inferir a zona de aprendizagem e informar ao instrutor o quanto antes, a seguir tem-se uma breve descrição de quando o professor teria acesso a essa informação em cada experimento:

- No E1, o professor tem a informação da zona de aprendizagem do aluno há alguns dias da avaliação intermediária, uma vez que o modelo preditivo foi treinado e



(a) (b) (c) (d) (e) (f) (g)

Figura 5.6: Resultados Experimento 6 - Análise de Erros da Regressão (Estratégia b).

testado com os dados de uma mesma sessão e os dados das sessões foram coletados há alguns dias das avaliações intermediárias.

- No E2, o professor teria essa informação com cerca 2 ou 3 semanas de antecedência, visto que o modelo preditivo treinou com os dados de uma sessão e foi testado com os dados de uma sessão subsequente.

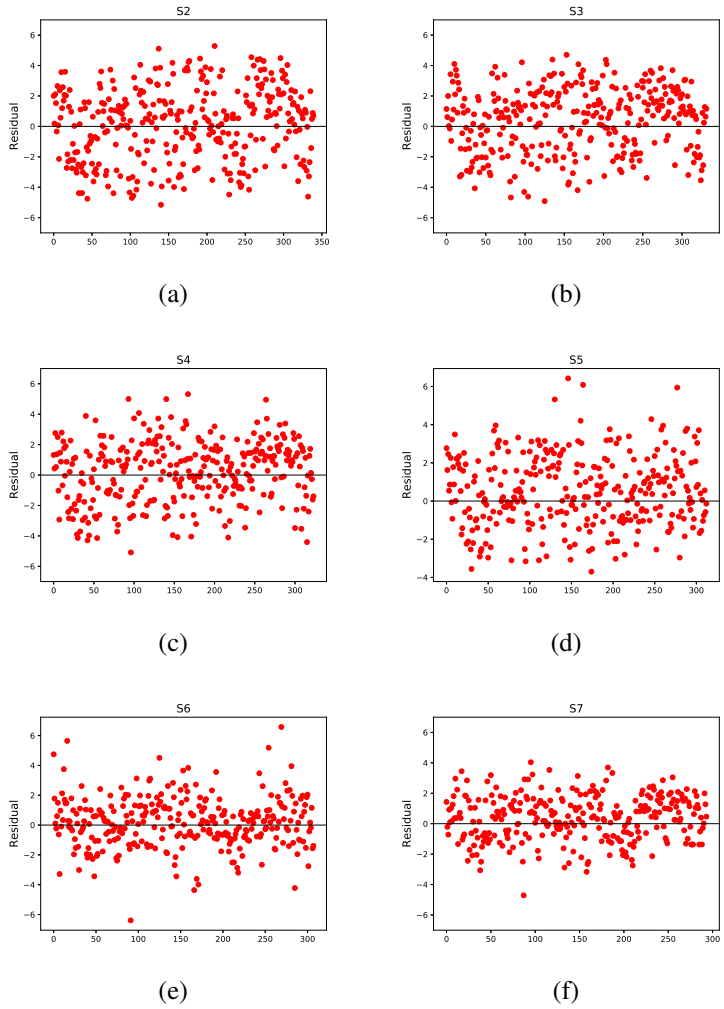


Figura 5.7: Resultados Experimento 6 - Análise de Erros da Regressão (Estratégia b).

- No E3 e E4, o professor saberia a zona de aprendizagem dos alunos já nas duas primeiras semanas de aula, já que os modelos preditivos foram treinados apenas com os dados da S1. A diferença entre o E3 e o E4 é que no primeiro o professor saberia a performance dos alunos nas avaliações intermediárias, já no segundo, o docente teria a informação da desempenho do aluno na média final.

- No E5 e E6 o professor saberia o desempenho do aluno na média final, entretanto ele teria uma informação mais acurada no decorrer do curso, uma vez que foram construídos diversos modelos preditivos treinados com os dados de cada uma das sessões. Para exemplificar, com os dados da S2 o professor teria uma estimativa da zona de aprendizagem dos alunos, já na S3 ele teria uma nova estimativa, já que o comportamento do aluno pode mudar, ou seja, a cada sessão é realizada uma nova predição da zona de aprendizagem do aluno. A diferença entre os dois experimentos

é que no E5 o docente teria a informação discretizada da média final, isto é, ele saberia se o aluno iria passar ou não, enquanto que no E6 ele teria de fato a média final do aluno estimada em uma escala contínua.

A seguir os resultados dos 6 experimentos serão discutidos mais profundamente à medida que as questões de pesquisa são respondidas.

5.8.1 Respondendo à QP1

QP1 *- Até que ponto o estudo de Estey e Coady (2016) pode ser reproduzido na base de dados apresentada neste trabalho? Além disso, o método preditivo deste estudo apresenta avanços em relação à predição da zona de aprendizagem se comparado ao método de Estey e Coady (2016)?*

O método de Estey e Coady (2016) pode ser reproduzido até certo ponto na base de dados apresentada neste estudo. Primeiramente, neste estudo os alunos usaram a linguagem de programação *Python*, enquanto que no contexto apresentado por Estey e Coady (2016) os estudantes usaram *Java*. A fim de remanejar as métricas de código baseadas em eventos de compilação, foi empregada a métrica de código M19 (syntax_error), que é uma adaptação. Além disso, o CodeBench não possui um esquema de geração de dicas como o do BitFit (ESTEY; COADY, 2016), o que impossibilitou neste trabalho o uso de qualquer métrica de código relacionada ao consumo de dicas dos alunos enquanto eles estavam resolvendo os exercícios.

Apesar dessas diferenças, no mais, o CodeBench e o BitFit são similares e todas as outras métricas de código utilizadas por Estey e Coady (2016) foram propriamente implementadas e empregadas no perfil de programação apresentado neste estudo. Assim, os atributos de Estey e Coady (2016) eram um subconjunto do perfil de programação. Especificamente os atributos implementados para a reprodução do método de Estey e Coady (2016) nos experimentos E2, E3 e E4 foram o M1, M13, M14, M15 e M18. Aplicou-se as abordagens A1 e A2 e coletou-se os melhores resultados. Note que a comparação foi realizada apenas nesses experimentos porque no trabalho de Estey e Coady (2016) os autores objetivavam identificar alunos com risco de reprovação ainda no início do curso.

Por um lado, quando foi realizada a predição da zona de aprendizagem na reprodução do E2 e E3 apenas com os atributos de Estey e Coady (2016), os resultados foram

próximos aos produzidos com o emprego do perfil de programação. Com o perfil de programação obteve-se acurácias entre 71,62% a 78,21%, enquanto que com os atributos de Estey e Coady (2016) obteve-se acurácia entre 69,29% a 75,00%. Nota-se que apesar dos atributos de Estey e Coady (2016) serem apenas um subconjunto do perfil de programação, os resultados obtidos mostram o poder preditivo dessas evidências, uma vez que o perfil de programação possui 20 atributos, enquanto eram somente 5 os atributos de Estey e Coady (2016).

Por outro lado, os atributos de Estey e Coady (2016) atingiram 69.55% de acurácia na tarefa de realizar a predição da média final usando apenas os dados das duas primeiras semanas, enquanto o perfil de programação atingiu 74,44%. Os resultados dos dois métodos podem ser visualizados na Tabela 5.21.

Tabela 5.21: Comparação dos melhores resultados nos experimentos E2, E3 e E4 com os do *baseline*. Em ambos foram utilizadas as abordagens A1 e A2. Note que **PP** foi utilizado com uma abreviação para perfil de programação. Melhores resultados estão em negrito.

	E2			E3		E4
	S2	**S3**	**S4**	**S5**	**S6**	**S7**
Acc. Estey	69,29%	69,62%	71,18%	72,15%	75,00%	69,55%
Acc. PP	**75,68%**	**74,49%**	**72,73%**	**78,21%**	**78,15%**	**74,44%**

Analisando as hipóteses H_{0B} e H_{1B}, observa-se na Tabela 5.22 que a acurácia obtida com o emprego dos atributos do perfil de programação foi estatisticamente superior ao método de Estey e Coady (2016) nos experimentos E3 e E4 (*p-value*<0,05). No E2, o perfil de programação foi superior nas sessões S2 e S3. Assim, o perfil de programação só não foi estatisticamente superior na sessão S4 do experimento E2.

Tabela 5.22: Teste estatístico da comparação dos melhores resultados nos experimentos E2, E3 e E4 com os do *baseline*.

	E2			E3		E4
	S2	**S3**	**S4**	**S5**	**S6**	**S7**
p-value	<0,0001	0,0003	0,1659	<0,0001	0,0089	0,0002

5.8.2 Respondendo à QP2

QP2 *- Utilizando o perfil de programação, é possível inferir a zona de aprendizagem de alunos de IPC com apenas os dados das duas primeiras semanas de aula? Além disso, os modelos preditivos aumentam a precisão no decorrer do curso ou ficam estagnados?*

Como pode ser visto na Tabela 5.5 (primeira linha) e Tabela 5.8, os modelos preditivos que treinaram apenas com os dados da S1 (duas primeiras semanas) atingiram acurácias entre 71,62% e 78,21% na tarefa de inferir a zona de aprendizagem nas sessões S2, S3 e S4 dos experimentos E2 e E3 com a base de dados balanceada. Note que no E2 e E3 a zona de aprendizagem foi representada pelo nota discretizada das avaliações intermediárias, enquanto que no E4 a zona de aprendizagem foi representada pela média final discretizada. No E4, o modelo preditivo atingiu 74.44% bem como usando para treinamento apenas os dados da S1.

A Figura 5.8 apresenta as curvas ROC do classificador construído no E4. A *micro-average ROC curve* foi de 0,81 e a *macro-average ROC curve* foi de 0,83, o que mostra que o classificador segregou bem os estudantes nas classes dos alunos que passaram e dos que reprovaram, mesmo quando o *threshold* foi diferente do valor central. Além disso, pode-se observar que as linhas contínuas azul e preta estão praticamente sobrepostas o que ratifica o fato de que o estimador classificou bem os alunos na ZD ou ZE.

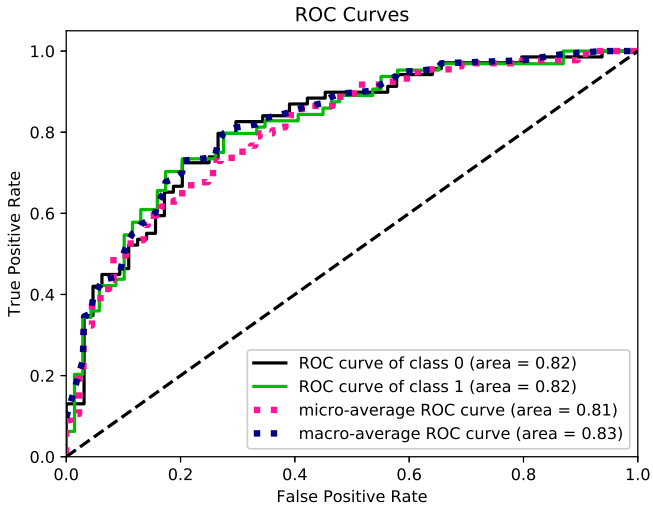


Figura 5.8: ROC curve da validação do modelo preditivo construído com a abordagem A1 no E4.

Em virtude disso, evidencia-se que os atributos apresentados no perfil de programa-

ção possuem um relacionamento com a performance dos estudantes, tanto nas avaliações intermediárias quanto na média final. Dessa forma, demonstra-se que o perfil pode ser empregado para a construção de modelos preditivos capazes de inferir a zona de aprendizagem do aluno ainda nas duas semanas de aula com uma precisão significativa. Destaca-se que outros estudos já sugerem que o desempenho dos alunos em exercícios no início do curso tem um relacionamento com a média final dos estudantes de IPC (ESTEY; COADY, 2016; AHADI et al., 2015).

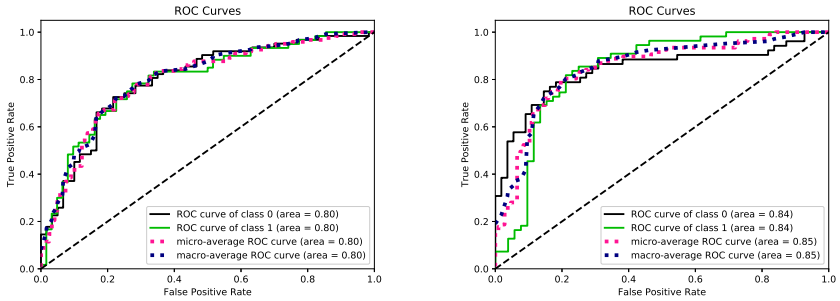
Vale a pena mencionar que se o professor tem ainda nas duas primeiras semanas de aula a informação se o estudante vai passar ou reprovar ou se ele irá mal ou bem nas próximas avaliações, algumas precauções podem ser tomadas. Para ilustrar, o professor ou um sistema de recomendações pode fazer provas mais adequadas à realidade da turma ou adaptar listas de problemas para grupos de estudantes com o perfil de programação semelhante.

Finalmente, ressalta-se que no decorrer do curso a precisão do modelo vai aumentando até chegar em um efeito platô que acontece a partir da sessão S6, como pode ser visto na Figura 5.9 que apresenta as curvas ROC dos modelos apresentado no experimento E5.

5.8.3 Respondendo à QP3

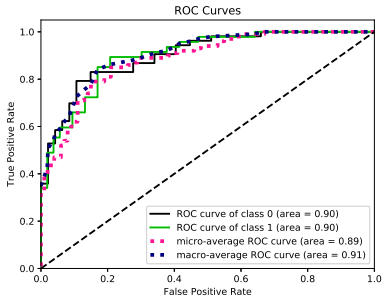
QP3 *- O uso de uma método que emprega algoritmos genéticos para a construção de modelos preditivos automaticamente proposto por Olson et al. (2016) é viável para a predição da zona de aprendizagem? Se sim, essa abordagem supera a construção manual de pipelines para a predição da zona de aprendizagem?*

A partir do Experimento E2 os *pipelines* de AM foram construídos usando duas abordagens: A1 e A2. Em A1, foi preciso encontrar algoritmos promissores de pré-processamento de atributos, escolher o algoritmo de aprendizagem e encontrar bons hiperparâmetros usando RandomSearch ou GridSearch, para otimizar o modelo preditivo. Esse foi um processo cíclico e exaustivo. Por outro lado, em A2, a busca guiada automatizada para encontrar *pipelines* de AM realizada com o emprego de algoritmos evolutivos proposta por (OLSON et al., 2016) requer um tempo considerável (300 minutos em média) para exportar o *pipeline*. Apesar da praticidade da abordagem A2, alguns modelos produzidos atingiram resultados ruins, em função de *overfitting* no treinamento. Entretanto, após algumas adaptações nos *pipelines* exportados, os resultados obtidos com A2

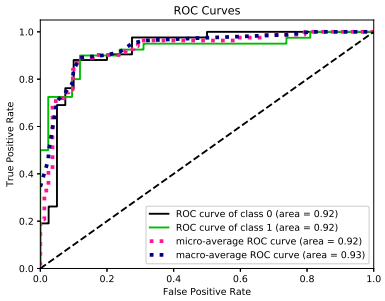


(a) S2

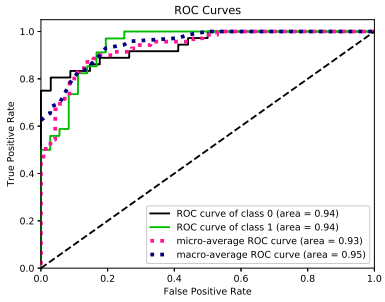
(b) S3



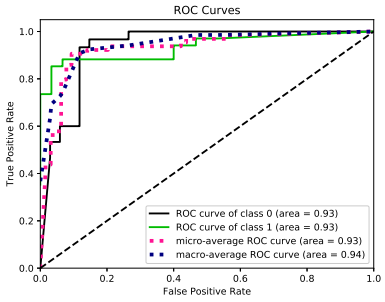
(c) S4



(d) S5



(e) S6



(f) S7

Figura 5.9: Curvas ROC do desempenho dos modelos preditivos do experimento E5.

foram, em geral, similares aos de A1 (Tabela 5.5, Tabela 5.8, Tabela 5.13, Tabela 5.17, Tabela 5.18). Dessa forma, conclui-se que a abordagem de (OLSON et al., 2016) é promissora para o objeto deste estudo, entretanto essa abordagem não é completamente automatizada, uma vez que precisou de adaptações baseada nos nossos *insights* sobre a base de dados para atingir acurácias mais significantes.

5.8.4 Respondendo à QP4

QP4 *- Dentre os atributos do perfil de programação utilizados para realizar as predições, desde os usados na literatura até os apresentados aqui que são derivados deles, quais*

são os mais relevantes em relação à acurácia do modelo preditivo?

Para responder esta questão, primeiramente a Tabela 5.23 mostra a Correlação de Pearson entre os atributos do perfil de programação e a média final ao longo das sessões. Observe que em linhas gerais a correlação vai aumentando ao longo das sessões. A correlação é mensurada entre cada variável de forma independente, utilizando os dados de cada sessão. Perceba que grande parte dos algoritmos de aprendizagem de máquina criam superfícies de decisão para realizar a classificação. Assim, à medida que um amostra passa por uma superfície de decisão, os dados são divididos em subamostras. Com efeito, haverá novas correlações entre as variáveis preditoras e a classe, isto é, a relevância dos atributos pode alterar nessas subamostras. Em outras palavras, alguns atributos que apresentaram valores com baixa correlação com os dados da sessão inteira, podem apresentar alta correlação em uma subconjunto desses dados. Portanto, a correlação de Pearson é apresentada com o intuito de mostrar o relevância individual dos atributos do perfil de programação.

Tabela 5.23: Correlação de Pearson entre os atributos do perfil de programação e a média final dos alunos em cada sessão.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**
attempts (M1)	0,1833	0,09	0,2585	0,3423	0,3209	0,2971	0,4046
comments (M2)	0,173	0,0991	0,0462	0,0565	-0,0308	0,09	0,1864
blank_line (M3)	0,1823	0,0672	0,1942	0,2071	0,2469	0,2221	0,3421
lloc (M4)	0,2387	0,0854	0,2267	0,3168	0,3254	0,3017	0,3901
loc (M5)	0,2472	0,0962	0,2251	0,3062	0,3139	0,3035	0,3878
single_comments (M6)	0,1706	0,0992	0,0682	0,0624	0,0256	0,0815	0,1783
system_access (M7)	0,2597	0,1819	0,2942	0,3054	0,2967	0,3017	0,4164
exam_grade_codebench (M8)	0,3304	0,4041	0,5189	0,5384	0,6385	0,634	0,6408
difficult (M9)	-0,2605	-0,2294	-0,3056	-0,2427	-0,3369	-0,3522	-0,2961
delete_average (M10)	0,1614	0,0595	0,2598	0,2218	0,2666	0,2491	0,3236
average_log_rows (M11)	0,2373	0,0705	0,2417	0,0759	0,326	0,2803	0,353
submission_per_exercice (M12)	0,15	0,1425	0,1567	0,1571	0,2048	0,081	0,1758
sucess_average (M13)	0,1555	0,0971	0,1011	0,1557	0,2642	0,2468	0,2534
test_average (M14)	0,2104	0,0017	0,1849	0,1707	0,1514	0,2335	0,3146
list_grade (M15)	0,358	0,3279	0,4452	0,4072	0,4813	0,4144	0,4866
list_grade_check_plagiarism (M16)	0,3686	0,31	0,4375	0,3714	0,3944	0,2636	0,4064
copy_past_proportion (M17)	-0,0097	0,0401	-0,0663	-0,0535	-0,029	0,0461	0,0473
sintaxe_error (M18)	-0,3108	-0,2317	-0,3624	-0,3731	-0,4446	-0,4116	-0,4151
IDE_usage (M19)	0,2189	-0,0537	0,3004	0,2215	0,3299	0,2359	0,385
keystroke_latency (M20)	0,1652	0,0918	0,2313	0,1628	0,2038	0,1359	0,3311

Por outro lado, para analisar a relevância dos atributos na construção dos modelos preditivos, foram aplicados métodos de seleção de atributos embutidos (LAL et al., 2006; GUYON; ELISSEEFF, 2003) durante a execução dos estimadores a fim de analisar e plotar os atributos mais relevantes em cada sessão, no que tange a predição da média final dos alunos. Como a relevância dos atributos do perfil de programação entre as sessões varia,

então não foi possível extrair um subconjunto homogêneo dos atributos mais importantes no curso inteiro, isto é, um subconjunto mínimo de atributos do perfil de programação que possa ser usado em todas as sessões. Dessa forma, optou-se por mostrar na Figura 5.10 apenas o top 5 dos atributos mais relevantes de cada sessão. A importância dos atributos foi mensurada pelo critério empregado no classificador para esse fim. Como os modelos com resultados mais expressivos foram baseados em árvores de decisão, a entropia foi o critério utilizado para mensurar a importância de cada atributo.

O M8 foi o atributo mais relevante em todas as sessões. Os atributos M1, M15 e M16 também aparecem diversas vezes entre os atributos mais significativos nas sessões. Esse fato corresponde às pesquisas de Ahadi, Vihavainen e Lister (2016), Castro-Wunsch, Ahadi e Petersen (2017) que apontam a relevância dos atributos M1 e M15 em seus estudos na construção de modelos preditivos precoce. Os atributos M8 e M15 mensuram o desempenho dos alunos ao longo do curso de forma tradicional, visto que elas são as notas da avaliações intermediárias e notas das listas, respectivamente. Por outro lado, as métricas M1 e M16 mensuram o desempenho de forma implícita, já que M1 calcula o número de submissões e M16 é a nota da lista baseada nas questões resolvidas e no número de *logs* gerados pelos alunos ao resolverem os problemas de programação.

Existiram ocorrências dos atributos M9 e M18, os quais mensuram as dificuldades reportadas pelos estudantes e erros de sintaxe aos resolver os problemas, respectivamente. Note que tais métricas de códigos podem ser úteis para identificar alunos com dificuldade.

Os atributos M4 e M5 mensuram a quantidade de linhas das submissões dos estudantes. Esses valores são muitas vezes utilizados na engenharia de software para quantificar o esforço no desenvolvimento de softwares. Neste contexto, essas métricas juntamente com a quantidade de *logs* gerados podem representar o esforço empregado para solucionar os exercícios. Por fim, o atributo M7 também apareceu com frequência entre os mais relevantes. Esse atributo calcula o número de vezes que os alunos acessaram o CodeBench. Em outras palavras, ele pode representar superficialmente o comprometimento e assiduidade do aluno no ACAC.

Destaca-se ainda que foram realizados testes durante os experimentos E4 e E5 removendo o atributo M8 (nota das avaliações intermediárias) na predição das médias finais dos alunos. Os resultados comprovaram que o perfil de programação não é dependente das notas das avaliações intermediárias dos estudantes, uma vez que os resultados se mantive-

ram expressivos ao longo das sessões. Especificamente, houve uma redução média entre 3% e 9% da acurácia dos modelos após a remoção do atributo M8.

Para resumir, acredita-se que os atributos do perfil de programação podem ser generalizados em outras bases de dados educacionais de turmas de IPC, uma vez que, em geral, eles não dependem de nuances de mensagens do compilador, especificidades da linguagem de programação ou do ACAC[2]. Além disso, esses atributos podem não ter uma relevância alta isoladamente (Tabela 5.23), entretanto, juntos demonstrou-se no cenário apresentado que eles possuem um poder preditivo significante.

5.8.5 Respondendo à QP5

QP5 *- A diferença entre o grupo de alunos que passaram na disciplina de IPC e os que reprovaram pode ser visualizada através do perfil de programação?*

A Figura 5.11 apresentam os gráficos de radar do valor médio normalizado dos alunos que ficaram na ZD (fail) e ZE (pass) ao longo das sessões de cada um dos atributos do perfil de programação. O valor zero nos eixos dos gráficos representa a média amostral, assim, se o grupo ultrapassar ou estiver abaixo de zero, significa que ele está acima ou abaixo da média, respectivamente. Os valores variam entre -1 e 1 que mostram a quanto desvios padrão os grupos estão da média.

Observa-se no gráfico que os únicos atributos do grupo ZD com valores superiores em todas as sessões são o M9 e M18, os quais são respectivamente a dificuldade reportada e a frequência de erros de sintaxe. Além disso, no atributo M17 os dois grupos apresentaram o mesmo padrão. Esse atributo é referente à frequência de "copy"e "paste"que os alunos fazem quando estão resolvendo os exercícios. Note que copiar e colar não é necessariamente um indício de que o aluno está cometendo plágio, uma vez que o estudante pode estar reaproveitando parte de um outro código.

Por outro lado, os alunos que passaram tentam mais vezes resolver os problemas, tem códigos maiores, comentam mais os códigos, acessam mais o CodeBench, conseguem atingir notas mais altas nas listas de exercícios e nas avaliações, passam mais tempo programando, pressionam mais vezes as teclas de remoção de caracteres e digitam mais rápido.

Finalmente, com exceção da S2, observa-se um desenho cada vez mais acentuada de

[2]O ACAC precisa ter um componente de software que realizada o coleta de dados.

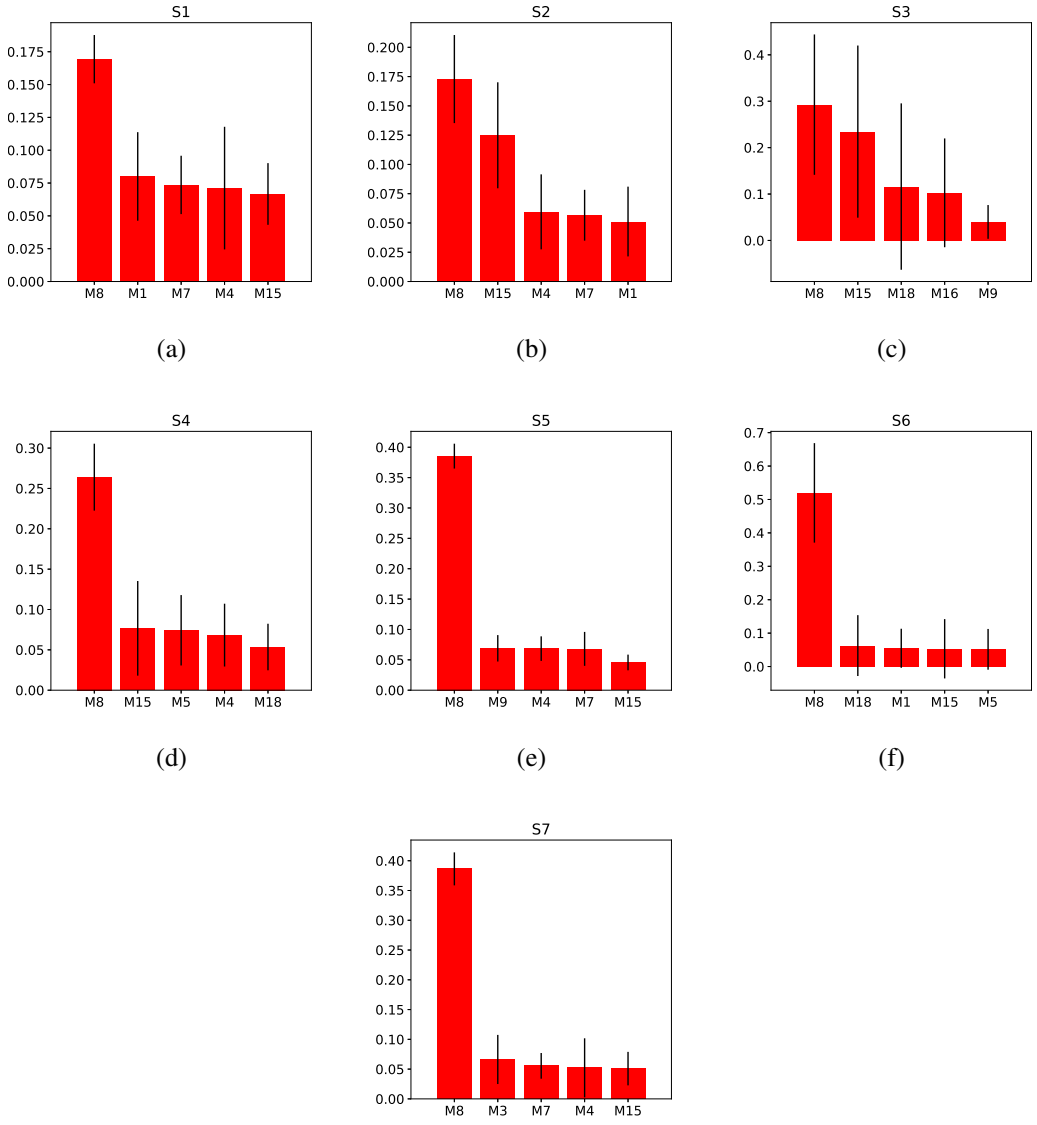


Figura 5.10: Os 5 atributos mais relevantes do perfil de programação na construção dos modelos preditivos de cada sessão.

um padrão em cada grupo ao longo das sessões, onde os alunos da ZE vão cada vez mais obtendo valores mais baixos que os da ZD nos atributos não relacionados a erros. Assim, conclui-se que existe uma diferença entre o perfil de programação dos alunos de ZD e os alunos da ZE e essa diferença no comportamento de programação dos estudantes pode ser representada visualmente, conforme apresentado na Figura 5.11.

5.9 Considerações Finais do Capítulo

Neste capítulo apresentou-se os resultados de 6 experimentos conduzidos com o intuito de inferir a zona de aprendizagem dos alunos. Para a construção dos modelos preditivos usou-se duas abordagens, sendo uma manual e a outra automatizada com uso de algoritmos genéticos.

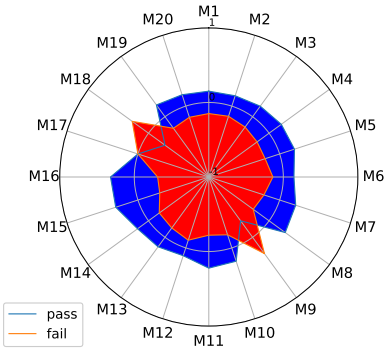
Apesar dos resultados do A2 terem sido inferiores em grande parte dos experimentos, essa abordagem ainda se mostra uma alternativa interessante para colocar o presente método em produção, visto que ele não requer um cientista de dados para criação e ajuste dos *pipelines* de predição nos próximos semestres.

No experimento E6, testou-se um método de redes neurais profundas acumulando os dados das sessões a fim de aumentar a quantidade de dados para a realização do treinamento. Observou-se que à medida que aumentava a quantidade de dados para treinamento a precisão da rede neural profunda também aumentava. Entretanto, os resultados da abordagem A1 foram superiores aos da rede neural em quase todas as sessões.

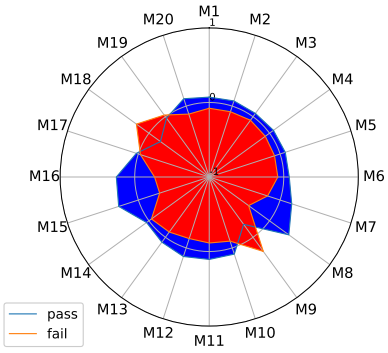
Ao longo dos experimentos os algoritmos baseados em árvores de decisão se mostraram mais promissores, mesmo na regressão.

Observa-se ainda que foram respondidas as questões de pesquisa estabelecidas no presente trabalho. Demonstrou-se que o perfil de programação atingiu resultados estatisticamente superiores aos do *baseline* na base de dados estudada. Finalmente, apresentou-se os atributos mais relevantes para o processo de predição em cada uma das sessões e como os grupos de alunos que reprovaram e os que passaram apresentam visualmente padrões diferentes de interação com o CodeBench.

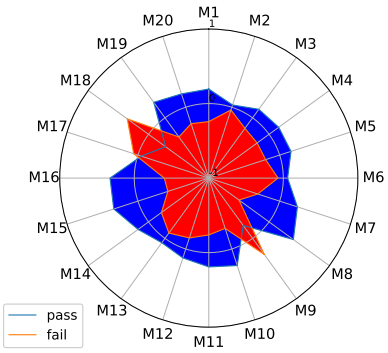
No próximo capítulo serão explanadas as considerações finais, as limitações do método e os trabalhos futuros.



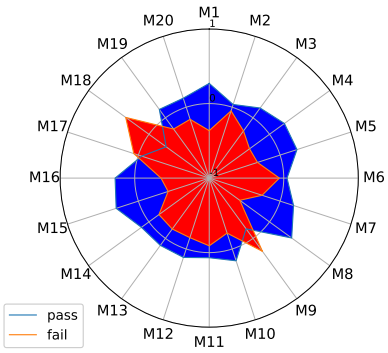
(a) RadarPlot S1.



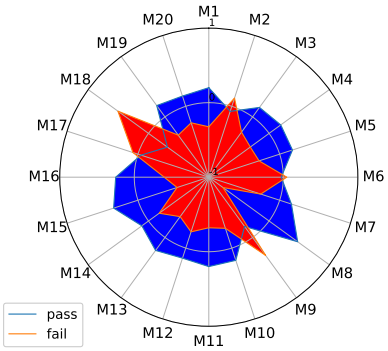
(b) RadarPlot S2.



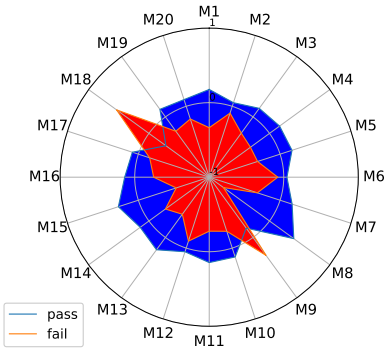
(c) RadarPlot S3.



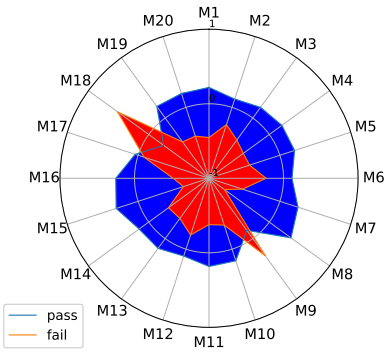
(d) RadarPlot S4.



(e) RadarPlot S5.



(f) RadarPlot S6.



(g) RadarPlot S7.

Figura 5.11: *RadarPlots* do perfil de programação das sessões.

Capítulo 6

Considerações Finais

Neste trabalho propôs-se e validou-se um método para realizar a predição da zona de aprendizagem de alunos de turmas de IPC que fazem uso de juízes online. Para tanto, realizou-se: a) a coleta dos *logs* dos alunos enquanto eles resolviam exercícios propostos em um juiz online; b) a construção de um perfil de programação de cada aluno, baseado em métricas de código inspiradas em estudos recentes e outras propostas pelos autores; c) a predição das notas das avaliações intermediárias e da média final de cada estudante; d) a otimização da acurácia do método preditivo através da aplicação de uma abordagem com métodos tradicionais de aprendizagem de máquina e outra com emprego de algoritmos genéticos.

O principal achado dessa pesquisa está no conjunto de evidências baseadas nos dados da interação de alunos de IPC com ACAC, o qual foi rotulado como perfil de programação do estudante. Esse artefato é uma compilação de atributos propostos pelos autores e outros derivados de pesquisas que realizaram a predição do desempenho de alunos de programação, entretanto com estratégias diferentes.

Demonstrou-se em seis experimentos que os atributos do perfil de programação podem ser empregados em algoritmos de AM para a tarefa de inferir a zona de aprendizagem dos alunos, seja na estimativa do desempenho dos alunos nas avaliações intermediárias ou na inferência da desempenho na média final. Mais especificamente, o modelo preditivo produzido atingiu uma precisão de 74,44% ainda nas duas primeiras semanas de aula. Além disso, o modelo fica cada vez mais preciso ao longo do curso, chegando a atingir uma acurácia próxima entre 85% e 90,62% em uma base de dados balanceada a partir da oitava semana de curso. Os métodos *ensembles* baseados em árvores de decisão (*Ran-*

domForest e *ExtraTrees*) foram, em geral, mais promissores ao longo dos experimentos.

Comparou-se os resultados com o emprego do perfil com um método proposto por Estey e Coady (2016). Os resultados com os atributos do perfil de programação foram estatisticamente superiores (*p-value*<00,5) aos resultados com uso dos atributos de Estey e Coady (2016) na predição da zona de aprendizagem usando para treinamento apenas os dados das duas semanas de aula.

No mais, existem benefícios significativos com relação ao uso dos atributos presentes no perfil de programação, já que além do seu poder preditivo, com algumas adaptações, isto é, ao invés de calcular a média geral dos atributos nas sessões, pode-se calcular esse valor para cada questão das listas de exercícios. Assim, acredita-se que é possível analisar de forma mais holística como os estudantes resolvem os problemas, como eles lidam com os erros e gerenciam os prazos das atividades e muito mais. Vale frisar que esses dados dos estudantes são referentes ao processo de aprendizagem do aluno e não somente ao produto gerado (códigos submetidos pelos estudantes), o que mostra que o perfil de programação proporciona uma avaliação mais formativa do discente, levando em consideração os caminhos, desafios e dificuldades enfrentadas durante a construção das soluções para os exercícios.

Acredita-se que o perfil de programação não é sensível às nuances da linguagem de programação utilizada no contexto educacional, o que os torna mais propícios à generalização. No estudo de Ihantola et al. (2015) é destacada a necessidade de tais métricas de código com poder de generalização, isto é, que podem obter resultados significativos em outros cenários educacionais.

Frisa-se ainda que identificar a zona de aprendizagem do aluno antecipadamente é algo relevante por diversas razões, dentre as quais ressalta-se: i) os professores poderiam direcionar orientação específica aos discentes com alta probabilidade de desempenho baixo; ii) os professores poderiam prover atividades mais desafiadoras aos alunos com valor estimado alto e iii) estudantes em risco poderiam ser monitorados, a fim de evitar que eles acabassem reprovando ou evadindo da disciplina.

Finalmente, demonstrou-se que a abordagem A2, que emprega um algoritmo genético para a produção automática de *pipelines* de AM, pode ser utilizada na tarefa em foco neste estudo, bastando apenas aplicar um esforço final no ajuste dos *pipelines* exportados. Essa abordagem atingiu em alguns momentos resultados similares e até superiores aos

da abordagem A1, onde o pesquisador cria os *pipelines* de AM manualmente. Automatizar o processo de criação dos modelos preditivos é algo importante neste contexto, pois acredita-se que ao longo dos semestres os padrões para realizar a predição irão mudar, assim, a abordagem A2 se mostra como uma alternativa viável para substituir um humano na produção de modelos preditivos que acompanhem essas mudanças.

6.1 Contribuições

Os resultados do experimento E1 foram publicados no XXVIII Simpósio Brasileiro de Informática na Educação (SBIE) em outubro de 2017 (PEREIRA; OLIVEIRA; OLIVEIRA, 2017). O Mapeamento Sistemático da Literatura (MSL) sobre predição de desempenho em ambientes *online* para turmas de programação foi submetido à Revista Brasileira de Informática (RBIE) em outubro de 2017, entretanto o resultado ainda não foi divulgado. Os resultados dos experimentos E2, E3 e E4 foram submetidos para a conferência *Educational Data Mining* em março de 2018, no entanto a divulgação do resultado será em abril de 2018.

6.2 Limitações

As maiores limitações do presente trabalho está relacionada à base de dados utilizada. Primeiramente, em termos de validação externa, foram utilizados os dados de 486 alunos do semestre 2016.1 e esse amostra pode não representar a população geral. Assim, é importante que o presente método seja replicado com outras bases de dados educacionais de outras instituições de ensino, a fim de validar a robustez do método. Observa-se ainda que em alguns experimentos a base de dados estava bastante desbalanceada, uma vez que a maioria dos alunos tiraram notas maiores ou igual a 5 nas avaliações intermediárias e na média final. Além disso, muitos estudantes trancaram ou evadiram a disciplina o que diminui ainda mais a amostra. Nessas condições, é mais complexo avaliar o poder de generalização do método.

Existem também ameaças internas à validade do método, como a possibilidade de estudantes trabalharem em grupo, o que acarretaria em uma mudança de comportamento individual na interação com o CodeBench. Além disso, no início do curso o aluno pode

ir explorando os recursos do CodeBench para conhecê-los, gerando comportamentos fora de uma padrão que deve se estabelecer depois desse processo de ambientação. Note que os alunos que já tiverem experiência em programação ou que reprovaram na disciplina poderão transpor essa etapa de ambientação com mais facilidade.

Um outro aspecto interno que pode afetar o desempenho dos modelos preditivos é relacionado ao plágio de soluções dos exercícios e até nas provas. Usou-se o atributo M11 e M16 para tentar contornar esse problema, sendo ambos relacionados ao número de *logs* que os alunos geram quando estão resolvendo os exercício na IDE embutida ao CodeBench. No entanto, estudos mais holísticos podem ser empregados para investigação de casos de plágio utilizando abordagens *data-driven*, o que pode ser levado em consideração em trabalhos futuros.

Finalmente, o método foi treinado e testado com dados de um semestre. Um outro semestre poderá ter uma didática distinta, com uso de materiais diferentes. Assim, por mais que o perfil de programação tenha sido desenhado para ser generalizável, existe a possibilidade de haver a necessidade de refazer o processo de construção de *pipeline* de AM para esses semestre. Isto é, realizar novamente o pré-processamento dos atributos, a escolha do algoritmo de AM e ajuste de hiperparâmetros. Entretanto, caso haja essa necessidade, a abordagem A2 descrita neste estudo aparece como um estratégia promissora, pois não é tão dependente de um cientista de dados para a realização deste processo.

6.3 Trabalhos Futuros

Como trabalhos futuros, tem-se como intuito adaptar os modelos preditivos produzidos utilizando mais dados, a fim de ter uma maior compreensão do comportamento de programação dos estudantes. Com mais dados, o uso de redes neurais profundas pode ser uma boa opção para maximizar a acurácia dos preditores. Além disso, deseja-se investigar o uso de séries temporais para a representação do progresso dos estudantes com o uso dos atributos do perfil de programação.

Pretende-se ainda dar dicas baseadas no comportamento *data-driven* do aluno representado pelo perfil de programação. Para exemplificar, neste estudo demonstrou-se que existe uma correlação positiva entre o número de submissões e as notas finais dos alunos. Dessa forma, o juiz *online* poderia dar dicas automáticas para o aluno, explicando que

normalmente algumas questões demandam muitas tentativas até que se consiga uma submissão que passe em todos os casos de teste. Se o aluno estiver cometendo muitos erros de sintaxe em uma mesma questão, o professor ou um monitor poderia ser notificado ou ainda o juiz *online* poderia sugerir um conteúdo, parte de um livro ou uma apresentação que ajudasse na resolução daquela questão. Perceba ainda que se o estudante passa pouco tempo programando (pouco tempo de uso de IDE), ele pode ser advertido a fim de que ele fique mais engajado na disciplina. É desejado ainda criar um sistema de recomendações de questões para juízes *online*, utilizando como base os atributos do perfil de programação.

Em relação a importância dos atributos, temos a intenção de agrupar as variáveis do perfil de programação de acordo com a competência de cada variável. Em outras palavras, atributos vinculados às notas dos alunos como notas das listas de exercícios, notas das avaliações intermediárias ficariam no mesmo grupo. Atributos relacionados ao esforço do aluno como quantidade de linhas de código, tempo de uso de IDE, número de submissões e outros ficaria em outro grupo. Depois do agrupamento, pretende-se paulatinamente remover grupo por grupo, a fim de analisar os modelos preditivos após as remoções. Dessa forma, seria possível identificar em quais momentos do curso quais grupos são mais relevantes.

Finalmente, pretende-se replicar esse estudo em outras instituições de ensino, com um formato de avaliação diferente. Além disso, é importante aplicar o método apresentado e verificar o resultado da intervenção do professor ou tutor à medida que ele sabe antecipadamente a zona de aprendizagem do aluno.

Referências

AHADI, A.; LISTER, R.; HAAPALA, H.; VIHAVAINEN, A. Exploring machine learning methods to automatically identify students in need of assistance. *International Computing Education Research - ICER'15*, p. 121–130, 2015. ISSN 9781450336307. Citado 9 vezes nas páginas 4, 5, 42, 51, 53, 55, 65, 66 e 98.

AHADI, A.; VIHAVAINEN, A.; LISTER, R. On the number of attempts students made on some online programming exercises during semester and their subsequent performance on final exam questions. *ACM Conference on Innovation and Technology in Computer Science Education*, p. 218–223, 2016. ISSN 9781450342315. Citado 9 vezes nas páginas 4, 36, 46, 51, 53, 55, 63, 65 e 101.

ALLEVATO, A.; EDWARDS, S. H. Discovering patterns in student activity on programming assignments. In: *2010 ASEE Southeastern Section Annual Conference and Meeting*. [S.l.: s.n.], 2010. Citado na página 37.

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. Citado 2 vezes nas páginas 1 e 35.

AUVINEN, T. Harmful study habits in online learning environments with automatic assessment. *2015 International Conference on Learning and Teaching in Computing and Engineering*, p. 50–57, 2015. ISSN 978-1-4799-9967-5. Citado 9 vezes nas páginas 35, 36, 37, 47, 51, 53, 55, 65 e 66.

BECKER, B. A. A new metric to quantify repeated compiler errors for novice programmers. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, p. 296–301, 2016. ISSN 9781450342315. Citado 6 vezes nas páginas 2, 37, 44, 45, 53 e 66.

BENGIO, Y. et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, Now Publishers, Inc., v. 2, n. 1, p. 1–127, 2009. Citado na página 28.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, v. 13, n. Feb, p. 281–305, 2012. Citado 2 vezes nas páginas 29 e 67.

BISHOP, J.; HORSPOOL, R. N.; XIE, T.; TILLMANN, N.; HALLEUX, J. de. Code hunt: Experience with coding contests at scale. In: IEEE PRESS. *Proceedings of the 37th International Conference on Software Engineering-Volume 2*. [S.l.], 2015. p. 398–407. Citado na página 36.

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*, n. October, p. 110–116, 2011. ISSN 9781450310574. Disponível em: <http://dl.acm.org/citation.cfm?id=2090116.2090132>. Citado 2 vezes nas páginas 1 e 5.

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*, v. 23, n. 4, p. 561–599, 2014. ISSN 1050-8406. Disponível em: <http://www.tandfonline.com/doi/abs/10.1080/10508406.2014.954750>. Citado 2 vezes nas páginas 2 e 11.

BLUM, A. L.; LANGLEY, P. Selection of relevant features and examples in machine learning. *Artificial intelligence*, Elsevier, v. 97, n. 1, p. 245–271, 1997. Citado na página 17.

BOYLE, T.; BRADLEY, C.; CHALK, P.; JONES, R.; PICKARD, P. Using blended learning to improve student success rates in learning to program. *Journal of Educational Media*, v. 28, n. 2-3, p. 165–178, 2003. ISSN 1358-1651. Citado na página 12.

BREIMAN, L. Bagging predictors. *Machine learning*, Springer, v. 24, n. 2, p. 123–140, 1996. Citado 3 vezes nas páginas 21, 23 e 24.

BREIMAN, L. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001. Citado na página 21.

CARTER, A. S.; HUNDHAUSEN, C. D.; ADESOPE, O. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. *International Computing Education Research - ICER'15*, p. 141–149, 2015. ISSN 9781450336307. Citado 3 vezes nas páginas 4, 37 e 44.

CARVALHO, L.; OLIVEIRA, 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. Citado 7 vezes nas páginas 2, 11, 12, 13, 14, 57 e 58.

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. Citado 6 vezes nas páginas 42, 51, 55, 65, 66 e 101.

CHAVES, J. O. M.; CASTRO, A. F.; LIMA, R. W.; LIMA, M. V. A.; FERREIRA, K. H. Integrando moodle e juízes online no apoio a atividades de programação. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2013. v. 24, n. 1, p. 244. Citado na página 13.

CHEANG, B.; KURNIA, A.; LIM, A.; OON, W. C. On automated grading of programming assignments in an academic institution. *Computers and Education*, v. 41, n. 2, p. 121–131, 2003. ISSN 0360-1315. Citado na página 13.

CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995. Citado na página 25.

COVER, T.; HART, P. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IEEE, v. 13, n. 1, p. 21–27, 1967. Citado na página 26.

CROCKER, L.; ALGINA, J. *Introduction to classical and modern test theory.* [S.l.]: ERIC, 1986. Citado na página 17.

DAUMÉ, H. A course in machine learning. *chapter*, v. 11, p. 149, 2012. Citado na página 21.

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. Citado 2 vezes nas páginas 34 e 68.

DORFMAN, R. A formula for the gini coefficient. *The Review of Economics and Statistics*, JSTOR, p. 146–149, 1979. Citado 2 vezes nas páginas 20 e 28.

EFRON, B.; TIBSHIRANI, R. J. *An introduction to the bootstrap*. [S.l.]: CRC press, 1994. Citado na página 21.

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. ISSN 9781450342315. Disponível em: <http://dl.acm.org/citation.cfm?doid=2899415.2899428>. Citado 18 vezes nas páginas 3, 6, 7, 9, 11, 35, 45, 51, 53, 55, 65, 70, 71, 72, 95, 96, 98 e 107.

FAWCETT, T. An introduction to roc analysis. *Pattern recognition letters*, Elsevier, v. 27, n. 8, p. 861–874, 2006. Citado na página 30.

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. Citado na página 33.

FRANCISCO, R.; JÚNIOR, C. P.; AMBRÓSIO, A. P. Juiz online no ensino de programação introdutória-uma revisao sistemática da literatura. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2016. v. 27, n. 1, p. 11. Citado na página 13.

FREUND, Y. Boosting a weak learning algorithm by majority. *Information and computation*, Elsevier, v. 121, n. 2, p. 256–285, 1995. Citado 2 vezes nas páginas 23 e 24.

FREUND, Y.; SCHAPIRE, R. E. et al. Experiments with a new boosting algorithm. In: BARI, ITALY. *Icml*. [S.l.], 1996. v. 96, p. 148–156. Citado na página 23.

FRIEDMAN, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, JSTOR, p. 1189–1232, 2001. Citado na página 24.

FULLER, U.; JOHNSON, C. G.; AHONIEMI, T.; CUKIERMAN, D.; HERNÁN-LOSADA, I.; JACKOVA, J.; LAHTINEN, E.; LEWIS, T. L.; THOMPSON, D. M.; RIEDESEL, C. et al. Developing a computer science-specific learning taxonomy. In: ACM. *SIGCSE Bulletin*. [S.l.], 2007. v. 39, n. 4, p. 152–170. Citado na página 47.

GARRISON, D. R.; KANUKA, H. Blended learning: Uncovering its transformative potential in higher education. *Internet and Higher Education*, v. 7, n. 2, p. 95–105, 2004. ISSN 1096-7516. Citado na página 12.

GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees. *Machine learning*, Springer, v. 63, n. 1, p. 3–42, 2006. Citado 2 vezes nas páginas 22 e 23.

GOMES, A. J.; MENDES, A. J. À procura de um contexto para apoiar a aprendizagem inicial de programação. *Educação, Formação & Tecnologias-ISSN 1646-933X*, v. 8, n. 1, p. 13–27, 2015. Citado 3 vezes nas páginas 1, 4 e 49.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado na página 27.

GUERRA-HOLLSTEIN, J. D.; BRUSILOVSKY, P. Modeling skill combination patterns for deeper knowledge tracing patterns in programming expertise. 2016. Citado 2 vezes nas páginas 35 e 66.

GUYON, I.; ELISSEEFF, A. An introduction to variable and feature selection. *Journal of machine learning research*, v. 3, n. Mar, p. 1157–1182, 2003. Citado na página 100.

HASTIE, T.; FRIEDMAN, J.; TIBSHIRANI, R. Boosting and additive trees. In: *The Elements of Statistical Learning*. [S.l.]: Springer, 2001. p. 299–345. Citado na página 24.

HEINONEN, K.; HIRVIKOSKI, K.; LUUKKAINEN, M.; VIHAVAINEN, A. Using codebrowser to seek differences between novice programmers. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*, p. 229–234, 2014. ISSN 9781450326056. Disponível em: <http://dl.acm.org/citation.cfm?doid=2538862.2538981>. Citado na página 11.

HELMINEN, J.; IHANTOLA, P.; KARAVIRTA, V.; MALMI, L. How do students solve parsons programming problems?: an analysis of interaction traces. In: ACM. *Proceedings of the ninth annual International Conference on International Computing Education Research*. [S.l.], 2012. p. 119–126. Citado na página 4.

HUNG, J.-L.; WANG, M.; WANG, S.; ABDELRASOUL, M.; LI, y.; HE, W. Identifying at-risk students for early interventions? a time-series clustering approach. *IEEE Transactions on Emerging Topics in Computing*, v. 6750, n. c, p. 1–1, 2016. Disponível em: <http://ieeexplore.ieee.org/document/7339455/>. Citado na página 35.

HUTTER, F.; LÜCKE, J.; SCHMIDT-THIEME, L. Beyond manual tuning of hyperparameters. *KI-Künstliche Intelligenz*, Springer, v. 29, n. 4, p. 329–337, 2015. Citado na página 33.

IHANTOLA, P.; VIHAVAINEN, A.; AHADI, A.; BUTLER, M.; BÖRSTLER, J.; EDWARDS, S. H.; ISOHANNI, E.; KORHONEN, A.; PETERSEN, A.; RIVERS, K. et al. Educational data mining and learning analytics in programming: Literature review and case studies. In: ACM. *Proceedings of the 2015 ITiCSE on Working Group Reports*. [S.l.], 2015. p. 41–63. Citado 17 vezes nas páginas 1, 2, 5, 11, 14, 35, 36, 38, 39, 47, 48, 49, 50, 51, 52, 66 e 107.

JADUD, M. C. *An exploration of novice compilation behaviour in BlueJ*. [S.l.]: Thesis submitted to the University of Kent at Canterbury in the subject of Computer Science, 2006. Citado 4 vezes nas páginas 43, 51, 52 e 55.

JADUD, M. C. Methods and tools for exploring novice compilation behaviour. In: ACM. *Proceedings of the second international workshop on Computing education research*. [S.l.], 2006. p. 73–84. Citado 2 vezes nas páginas 37 e 65.

LAHTINEN, E.; ALA-MUTKA, K.; JÄRVINEN, H.-M. A study of the difficulties of novice programmers. In: ACM. *ACM SIGCSE Bulletin*. [S.l.], 2005. v. 37, n. 3, p. 14–18. Citado na página 1.

LAL, T.; CHAPELLE, O.; WESTON, J.; ELISSEEFF, A. Embedded methods. *Feature extraction*, Springer, p. 137–165, 2006. Citado na página 100.

LEINONEN, J.; LONGI, K.; KLAMI, A.; VIHAVAINEN, A. Automatic inference of programming performance and experience from typing patterns. p. 132–137, 2016. Citado 8 vezes nas páginas 35, 38, 39, 41, 51, 53, 55 e 66.

LISTER, R. Cs education research the naughties in csed research: a retrospective. *ACM Inroads*, ACM, v. 1, n. 1, p. 22–24, 2010. Citado na página 11.

LOMAX, R. G.; HAHS-VAUGHN, D. L. *Statistical concepts: A second course*. [S.l.]: Routledge, 2013. Citado na página 17.

MITCHELL, M. *An introduction to genetic algorithms*. [S.l.]: MIT press, 1998. Citado na página 32.

MITCHELL, T. M. et al. *Machine learning. WCB*. [S.l.]: McGraw-Hill Boston, MA:, 1997. Citado 3 vezes nas páginas 15, 16 e 29.

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. Citado 6 vezes nas páginas 42, 43, 51, 55, 65 e 66.

NOVÁK, V.; PERFILIEVA, I.; MOCKOR, J. *Mathematical principles of fuzzy logic*. [S.l.]: Springer Science & Business Media, 2012. v. 517. Citado na página 35.

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. Citado 9 vezes nas páginas 7, 8, 33, 34, 68, 74, 81, 98 e 99.

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. Citado 9 vezes nas páginas 13, 35, 36, 46, 51, 55, 64, 66 e 69.

OTERO, J.; PALACIOS, A.; SUáREZ, R.; JUNCO, L.; COUSO, I.; SáNCHEZ, L. A procedure for extending input selection algorithms to low quality data in modelling problems with application to the automatic grading of uploaded assignments. *Scientific World Journal*, v. 2014, 2014. Citado na página 35.

OTERO, J.; SUÁREZ, M. D. R.; PALACIOS, A.; COUSO, I.; SÁNCHEZ, L. Selecting the most informative inputs in modelling problems with vague data applied to the search of informative code metrics for continuous assessment in computer science online courses. In: SPRINGER. *International Conference on Rough Sets and Current Trends in Computing*. [S.l.], 2014. p. 299–308. Citado 3 vezes nas páginas 4, 31 e 66.

PEA, R. D.; KURLAND, D. M. On the cognitive effects of learning computer programming. *New ideas in psychology*, Elsevier, v. 2, n. 2, p. 137–168, 1984. Citado na página 1.

PEDREGOSA, F.; VAROQUAUX, G.; 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. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Citado 7 vezes nas páginas 17, 18, 25, 26, 29, 30 e 68.

PEREIRA, F.; OLIVEIRA, E.; OLIVEIRA, 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. Citado 4 vezes nas páginas 62, 67, 78 e 108.

PETERSON, A.; SPACCO, J.; VIHAVAINEN, A. An exploration of error quotient in multiple contexts. *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, p. 77–86, 2015. ISSN 9781450340205. Disponível em: <http://dl.acm.org/citation.cfm?id=2828966>. Citado 2 vezes nas páginas 37 e 45.

PIECH, C.; SAHAMI, M.; HUANG, J.; GUIBAS, L. Autonomously generating hints by inferring problem solving policies. *Proceedings of the Second (2015) ACM Conference on Learning @ Scale - L@S '15*, p. 195–204, 2015. ISSN 9781450334112. Disponível em: <http://dl.acm.org/citation.cfm?id=2724660.2724668>. Citado na página 14.

QUINLAN, J. R. Induction of decision trees. *Machine learning*, Springer, v. 1, n. 1, p. 81–106, 1986. Citado na página 19.

RUSSELL, S.; NORVIG, P. Ai a modern approach. *Learning*, v. 2, n. 3, p. 4, 2005. Citado 5 vezes nas páginas 16, 19, 20, 24 e 27.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, IBM, v. 3, n. 3, p. 210–229, 1959. Citado na página 15.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural networks*, Elsevier, v. 61, p. 85–117, 2015. Citado na página 27.

SHANNON, C. E. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, ACM, v. 5, n. 1, p. 3–55, 2001. Citado na página 19.

SHIFFMAN, D. *The Nature of Code: Simulating Natural Systems with Processing*. [S.l.]: Daniel Shiffman, 2012. Citado na página 32.

SINGH, G.; SRIKANT, S.; AGGARWAL, V. Question independent grading using machine learning: The case of computer program grading. In: . [S.l.]: Association for Computing Machinery, 2016. p. 263–272. ISSN 9781450342322. Citado na página 11.

SOUZA, D. M.; FELIZARDO, K. R.; BARBOSA, E. F. A systematic literature review of assessment tools for programming assignments. *IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, p. 147–156, 2016. ISSN 978-1-5090-0765-3. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7474479>. Citado na página 14.

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. Citado na página 47.

TAN, P.-H.; TING, C.-Y.; LING, S.-W. Learning difficulties in programming courses: undergraduates' perspective and perception. In: IEEE. *Computer Technology and Development, 2009. ICCTD'09. International Conference on*. [S.l.], 2009. v. 1, p. 42–46. Citado na página 1.

TOLL, D. Measuring programming assignment effort. Faculty of Technology, Linnaeus University, 2016. Citado 2 vezes nas páginas 2 e 37.

VAPNIK, V. *The nature of statistical learning theory*. [S.l.]: Springer science & business media, 2013. Citado na página 25.

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. Citado 2 vezes nas páginas 35 e 47.

VIHAVAINEN, A.; HELMINEN, J.; IHANTOLA, P. How novices tackle their first lines of code in an ide: Analysis of programming session traces. *Koli Calling '14*, p. 109–116, 2014. ISSN 9781450330657. Citado na página 38.

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. Citado 8 vezes nas páginas 36, 37, 38, 39, 40, 43, 47 e 48.

WATSON, C.; LI, F. Failure rates in introductory programming revisited. *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, p. 39–44, 2014. ISSN 9781450328333. Citado 3 vezes nas páginas 1, 37 e 44.

WHITLEY, D. A genetic algorithm tutorial. *Statistics and computing*, Springer, v. 4, n. 2, p. 65–85, 1994. Citado na página 32.

WITTEN, I. H.; FRANK, E.; HALL, M. A.; PAL, C. J. *Data Mining: Practical machine learning tools and techniques*. [S.l.]: Morgan Kaufmann, 2016. Citado 2 vezes nas páginas 18 e 23.

XU, L.; YAN, P.; CHANG, T. Best first strategy for feature selection. In: IEEE. *Pattern Recognition, 1988., 9th International Conference on*. [S.l.], 1988. p. 706–708. Citado na página 18.

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. Citado 2 vezes nas páginas 33 e 67.