Otimização e automação da predição precoce do desempenho de alunos que utilizam juízes online: uma abordagem com algoritmo genético

Filipe Dwan Pereira[1], Elaine H. T. Oliveira[2], David F. Oliveira[2], Leandro S. G. Carvalho[2], Hermino Barbosa[1]

[1]Departamento de Ciência da Computação - Universidade Federal de Roraima (UFRR)

[2]Instituto de Computação - Universidade Federal do Amazonas (UFAM)

{filipe.dwan,hermino}@ufrr.br,{elaine,david,leandro}@icomp.ufam.edu.br

Abstract. *In this work, we present an approach to predict student performance in the very first two weeks from CS1 classes, which use programming online judges. We performed the prediction with a binary classification, i.e., we estimated whether the student succeeded or failed. To do so, we employed a method using an evolutionary algorithm to build and optimize automatically the machine learning pipeline. We trained the predictive model with data from 9 different courses run during 6 terms (2016-2018). As a result, we achieved an AUC of 0.87 on the validation set.*

Resumo. *Neste trabalho é apresentada uma abordagem para a predição ainda nas duas primeiras semanas de aula do desempenho de alunos de turmas de Introdução à Programação de Computadores que utilizam sistemas de correção automática de código. O desempenho do aluno foi inferido através de uma classificação binária, isto é, foi estimado se o aluno iria ser aprovado ou reprovado na disciplina. Para tanto, foi utilizado um método que emprega uma algoritmo genético para a construção automática de um pipeline de aprendizagem de máquina. O modelo foi treinado com dados de alunos de 9 cursos diferentes ao longo de 6 semestres distintos (2016-2018). Como resultado, obteve-se uma área sob a curva ROC de 0,87 na base de validação.*

1. Introdução

Apesar da importância de disciplinas de Introdução à Programação de Computadores (IPC) para cursos de Ciências Exatas e Engenharias, o alto índice de reprovações e desistências em IPC é um problema grave e recorrente [Galvão et al. 2016, Pereira et al. 2019]. Inúmeros trabalhos da literatura procuraram elencar as razões desses altos índices de reprovação para minimizar esse problema [Pereira et al. 2018].

Para auxiliar os alunos de IPC em seu processo de aprendizagem, alguns trabalhos propuseram o uso de ferramentas pedagógicas capazes estimular e simplificar as atividades de programação propostas durante a disciplina [Souza et al. 2016]. Dentro desse conjunto de ferramentas estão os Ambientes de Correção Automática de Código (ACAC) ou Juízes Online (JO). Em geral, os Juízes Online dispõem de um conjunto de exercícios de programação e um *Integrated Development Environment* (IDE), que permitem o aluno desenvolver, compilar, executar e submeter a resolução de cada exercício. Após cada submissão, o aluno é imediatamente informado se seu código está correto ou não [Francisco et al. 2016, Pereira et al. 2018].

DOI: 10.5753/cbie.sbie.2019.1451

No presente trabalho, foram utilizados componentes de softwares embutidos na IDE do juiz online CodeBench[1], implementado por um dos autores, para monitorar e registrar as ações dos alunos durantes suas tentativas de resolver os exercícios de programação propostos pelo professor. Os dados coletados são *logs* e possuem informações tais como as tentativas de compilação, teclas pressionadas, saídas (*outputs*) do programa que está sendo desenvolvido, entre outros. Nesse contexto, [Dwan et al. 2017] propuseram um método capaz de usar um conjunto específico de atributos dos *logs* para prever o desempenho dos alunos nas avaliações que acontecem durante as disciplinas. O objetivo dos autores foi permitir que o professor identifique alunos com alta probabilidade de reprovação para assim auxiliá-los em suas dificuldades, minimizando a taxa de reprovação normalmente alta de IPC.

Entretanto, a tarefa de construção e otimização do modelo preditivo adotada por [Dwan et al. 2017] requer um grande esforço manual, exploratório e exaustivo, pois demanda uma série de etapas não triviais: 1) seleção, transformação e construção de atributos; 2) seleção do algoritmo de AM; 3) ajuste de parâmetros do algoritmo de AM; e por fim 4) a validação do modelo preditivo. Como o perfil dos alunos varia bastante de um local para outro, seria necessário refazer esse *pipeline* (sequência de etapas) em cada nova instituição de ensino em que o método de predição fosse aplicado. Desta forma, um mecanismo de automatização desse *pipeline* é desejável, para que a construção e otimização do modelo preditivo seja feita de forma direta e sem necessidade de um cientista de dados experiente.

No trabalho de [Olson et al. 2016], foi apresentado um método de otimização de *pipelines* baseado em árvore (TPOT) que automatiza as etapas 1, 2 e 3 citadas no parágrafo anterior. Esse método é capaz de automatizar e otimizar o projeto de *pipeline* usando algoritmos genéticos em implementações existentes no *scikit-learn*. Os algoritmos genéticos são técnicas capazes de encontrar soluções para problemas específicos utilizando uma estrutura de dados semelhantes a de um cromossomo. As soluções codificadas são submetidas a um processo de recombinação utilizando operadores que preservam as informações críticas [Whitley 1994].

Resumidamente, neste trabalho pretende-se adaptar o método de [Olson et al. 2016] para construção de modelos preditivos a partir dos atributos apresentados por [Dwan et al. 2017] mais alguns atributos propostos no presente trabalho, com o objetivo de identificar os alunos com risco de reprovação ainda nas duas primeiras semanas de aula da disciplina de IPC. Além disso, este trabalho também tem por objetivo construir um modelo de predição genérico e adaptável aos perfis de alunos de IPC. Como resultado, o modelo preditivo construído obteve um área sob a curva ROC de 0,87 na tarefa de inferir se o aluno iria ser aprovado ou reprovado em IPC.

2. Trabalhos Relacionados

No estudo de [Munson and Zitovsky 2018] foi empregado modelos de regressão para prever a média final de 86 estudantes de turmas de IPC. Os autores analisaram os logs e eventos de compilação dos alunos enquanto resolviam questões de programação em Java em uma IDE chamada CodeWork. Os atributos utilizados para a construção dos modelos preditivos eram baseados no número e tamanho das sessões de programação, tamanho dos códigos fontes submetidos, e no número de erros de compilação.

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

[Jadud 2006] propôs um método de predição de notas baseado em um quociente de erros (EQ) que detecta um ciclo de edição, compilação e execução de códigos em alunos que estavam aprendendo a linguagem Java. [Watson et al. 2013] estenderam o EQ com um algoritmo que calcula uma métrica denominada *Watwin Score* (WS), que classifica os pares de execução/compilação de códigos submetidos pelos alunos considerando adicionalmente o tempo de resolução do problema.

Finalmente, [Dwan et al. 2017] propuseram um método para inferir se os alunos irão obter uma nota maior ou igual a 5 nas avaliações que ocorrem durante uma disciplina de IPC. Para isso, foi construído um perfil de programação para cada aluno a partir dos *logs* gerados pelos estudantes durante as atividades de programação em um OJ. Entretanto, o método proposto nesse trabalho gera as previsões de notas com pouca antecedência, uma vez que o modelo preditivo é treinado e testado com *logs* que devem ser coletados até uma data próxima da avaliação. Esse fato dificulta uma atuação proativa por parte do professor, já que ele tem pouco tempo hábil para tentar reverter as possíveis reprovações nas avaliações.

Pensando nisso, neste trabalho pretende-se identificar alunos com grande possibilidade de reprovação já nas duas primeiras semanas de aula, possibilitando assim que os professores tomem atitudes proativas para evitar que tais alunos acabem realmente reprovando. Além disso, a principal contribuição deste estudo é adaptar o método de [Olson et al. 2016] para a construção automática dos modelos preditivos utilizando os atributos de aprendizagem de máquina sugeridos por [Dwan et al. 2017] mais novos atributos propostos neste estudo com base nos trabalhos de [Jadud 2006], [Watson et al. 2013] e [Munson and Zitovsky 2018].

3. Construção automática de pipelines de aprendizagem de máquina com algoritmos genéticos

[Zutty et al. 2015] demonstraram que a otimização com algoritmos genéticos pode superar os seres humanos na busca de um *pipeline* de aprendizagem de máquina[2] que melhor se encaixa em uma tarefa de aprendizagem supervisionada. Nesse sentido, [Olson et al. 2016] propuseram um método para construção automática de pipelines chamado de *Tree-based Pipeline Optimization Tool* (TPOT), no qual a intenção é criar, inicialmente, uma população inteira de *pipelines* aleatórios e evoluí-los com operadores de mutações e cruzamentos ao longo de gerações. Para construir os *pipelines*, o TPOT emprega vários algoritmos de seleção, construção e transformação de atributos e algoritmos de aprendizagem de máquina 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.

Resumidamente, a ideia principal do TPOT é encontrar, em um grande espaço de busca, um *pipeline* de aprendizagem de máquina que melhor se adapte aos dados, utilizando uma busca guiada baseada em uma versão de um algoritmo genético multiobjetivo NSGA-II [Deb et al. 2002]. Para construção do *pipeline*, o TPOT usa implementações da biblioteca de AM *scikit-learn*.

[2]O pipeline é um conjunto de etapas necessárias para rodar um experimento de aprendizagem de máquina sobre um conjunto de dados. Essas etapas normalmente são: engenharia de atributos, escolha do algoritmo de aprendizagem de máquina, otimização e validação do modelo preditivo.

4. Contexto Educacional

Nesta pesquisa foram utilizados registros de *logs* coletados nas duas primeiras semanas de aula de 9 turmas de IPC da Universidade Federal do Amazonas. Essas turmas foram ofertadas nos dois semestres de 2016, 2017 e 2018 para diferentes cursos de graduação, e os módulos estudados nas duas primeiras semanas de aula foram (i) variáveis e (ii) estrutura de programação sequencial. Dentro desses módulos, os alunos tiveram acesso a uma lista de exercícios e foram submetidos a uma avaliação no final da segunda semana.

Os estudantes resolviam problemas de programação da lista e da avaliação diretamente na IDE embutida no CodeBench. Eles desenvolviam, submetiam e recebiam o feedback se a questão estava certa ou errada em um mesmo sítio. À medida que os estudantes resolviam os problemas, um componente de software escrito em Javascript coletava as ações desempenhadas na IDE. Assim, eram registrados as teclas pressionadas pelo usuário em cada instante, eventos de clique, submissões de código e etc.

Para ilustrar a coleta de *logs* no juiz online, a Figura 1 mostra que às 08:34h o aluno clicou na IDE (evento *focus* na linha 1 da Figura 1). As linhas 2 à 7 mostram o aluno escrevendo o comando *print()* no editor de códigos (IDE) do juiz online. 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 precoce do desempenho 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 1. Exemplos de *logs* do juiz online CodeBench de quando o aluno está escrevendo o comando *print()*.

Destaca-se que esse tipo de coleta de dados permite o armazenamento do processo de construção da solução do aluno. Assim, os modelos preditivos realizam a inferência de desempenho com base em atributos extraídos desse processo de codificação, que contém todos os passos realizados pelo aluno para a construção de seu código.

4.1. Construção dos Pipelines de Aprendizagem de Máquina

As etapas realizadas para a construção dos *pipelines* de aprendizagem de máquina podem ser vistas na Figura 2. Primeiramente, é realizado um ciclo entre pré-processamento de atributos, escolha do algoritmo de aprendizagem de máquina e ajuste de parâmetros até se obter um modelo preditivo capaz de maximizar a acurácia na indução precoce do desempenho do aluno.

Um dos maiores diferenciais deste artigo para o trabalho de [Dwan et al. 2017] é que esse último realizou as etapas 1, 2 e 3 da Figura 2 de forma manual, enquanto que no corrente trabalho essas etapas são realizadas de forma automática, utilizando o método

proposto por [Olson et al. 2016]. Observe que, ao invés de um especialista humano, será o algoritmo genético encapsulado no TPOT o responsável por encontrar um algoritmo de seleção, transformação e construção de atributos, escolher o algoritmo de aprendizagem de máquina e realizar os ajustes dos parâmetros desse algoritmo. Como o perfil dos alunos varia bastante de um local para outro, seria necessário refazer o *pipeline* em cada nova universidade em que método de predição fosse aplicado. Com a produção automática dos *pipelines*, as universidades podem utilizar o modelo preditivo sem a necessidade de recorrer a um especialista da área de aprendizagem de máquina. Assim sendo, a construção automatizada (via TPOT) de classificadores é uma alternativa facilitadora desse processo.

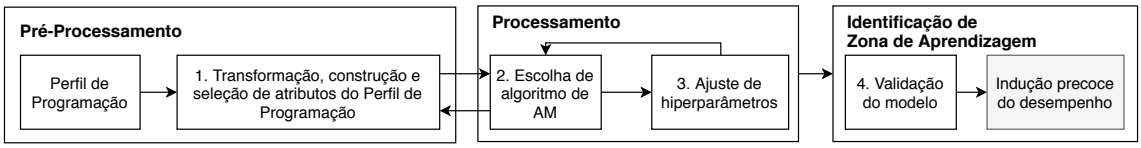

Figura 2. Etapas para construção de pipelines de AM para a predição da zona de aprendizagem. Adaptado de [Dwan et al. 2017].

Ainda, neste trabalho foram extraídos 11 dos 22 atributos do perfil de programação proposto por [Dwan et al. 2017] e acrescentados 3 novos atributos com base nos trabalhos relacionados, totalizando 14. Para extrair os 11 atributos dos 22 proposto por [Dwan et al. 2017], foi analisada a correlação (Pearson e Spearman) de todos os pares de atributos possível e em casos de alta correlação ($> 0,9$ ou $< -0,9$) um dos dois foi removido. Para ilustrar, os atributos *num_comentarios* (que quantifica o número de comentários de linhas únicas nos códigos dos alunos) e *num_multi* (que quantifica o número de comentários em linhas múltiplas) são altamente correlacionados e, por isso, apenas o primeiro foi mantido. Além disso, analisou-se a importância dos atributos no processo de aprendizagem em uma Floresta Aleatória (*Random Forest*). Atributos que tinham uma contribuição tendendo a zero foram excluídos.

Abaixo são listados os 14 atributos, onde aqueles que foram propostos neste estudo estarão em negrito: *num_tentativas* (M1): número de tentativas de submissão do código; *sloc* (M2): número de linhas; *tempo_uso_ide* (M3): tempo (minutos) codificando as soluções; *num_acessos* (M4): número de acessos (logins) do aluno no juiz online; *nota_avaliacao* (M5): nota na primeira avaliação; *nota_lista_esforço* (M6): nota do aluno na primeira lista, considerando apenas questões resolvidas que geraram no mínimo 50 linhas no log; *num_comentarios* (M7): número de linhas com comentários nos códigos; *num_linhas_logs* (M8): número de linhas do log gerado durante a codificação; *nota_lista* (M9): nota do aluno na primeira lista de exercícios; *velocidade_digitacao* (M10): velocidade de digitação do aluno; *media_deletes* (M11): média de caracteres apagados através do uso das teclas delete e *backspace*; **EQ** (M12): quociente de erro, calculado com base em erros de sintaxe repetidos [Jadud 2006]; **WS** (M13): *Watwin Score* (WS) é o tempo (normalizado) que o aluno leva para resolver um erro [Watson et al. 2013]; **cont_var** (M14): quantidade de variáveis utilizadas no código.

Finalmente, neste estudo, ao invés de usar os dados de apenas um semestre (2016.1), foram usados dados de 6 semestres com o total de 2058 estudantes, a fim de mostrar o poder de generalização do método. Os dados dos alunos de diversos cursos e semestres diferentes foram colocados juntos, com a finalidade de compor um método que não fosse útil para apenas um único semestre.

No total, a base de dados tinha 939 estudantes reprovados e 707 aprovados. Para avaliar o modelo preditivo, foi primeiramente separado 30% dos dados para validação e no restante foi aplicado validação cruzada com 10 *folds* para treino/teste. Essa separação foi realizada de forma estratificada, a fim de que a base de validação mantivesse a mesma proporção de aprovados e reprovados que a base de treino/teste. Para mensurar a precisão do modelo na validação, utilizou-se as curvas ROC e curvas *precision/recall*, a fim de que fossem analisados os erros de ambas as classes (aprovados ou reprovados). Por fim, a função a ser otimizada pelo TPOT com os dados de treino/teste foi a *f1-score*, uma vez que essa métrica de avaliação é indicada para treinamento em bases de dados desbalanceadas já que ela é uma média harmônica do *precision* e *recall*.

5. Resultados e Discussões

Durante o experimento para a construção dos modelos preditivos, foi configurada uma instância do TPOT com uma população inicial de 150 indivíduos (possíveis *pipelines*). O número de gerações foi limitado pelo tempo de experimento, e foi estabelecido um tempo limite de busca de 30 minutos. A taxa de mutação foi definida para 90% e a taxa de cruzamento para 10%. Esses parâmetros foram atribuídos com base nas próprias recomendações de [Olson et al. 2016]. Observe que uma taxa de mutação alta como essa é utilizada para maximizar a exploração do espaço de busca. O algoritmo convergiu após 8 gerações.

Como resultado, o TPOT exportou o modelo com o algoritmo Floresta Aleatória regularizada, que sempre levava em consideração 30% dos atributos para realizar as ramificações das árvores de decisão constituintes e que só efetuava divisões (ramificação) se houvesse no mínimo 8 amostras no nodo. A floresta possuía 100 árvore de decisão, sendo que a decisão da floresta era obtida através da votação das árvores constituintes (método *hard-voting*). Ainda, o modelo exportado empregava reamostragem com reposição (na estatística, *bootstrapping*) para a formação das subamostras utilizadas para o treinamento das árvores de decisão constituintes.

Para avaliar o modelo preditivo na base de validação, utilizou-se curvas precision/recall (Figura 3) e curvas ROC (Figura 4). Note que a primeira métrica plota o *recall* e o *precision* para diferentes limiares[3], enquanto que a segunda plota a taxa de falsos positivos e a taxa de verdadeiros positivos também para diferentes limiares, onde os positivos são representados por alunos aprovados e negativos por alunos reprovados. As curvas ROC tiveram uma área de 0,87 sob a curva (para ambas as classes) e a curva de precision/recall obteve 0,8 para a classe 1 (aprovados) e 0,91 para a classe 0 (reprovados), o que mostra que o classificador segregou bem os estudantes nas classes dos alunos que passaram e dos que reprovaram, mesmo quando o *limiar* foi diferente do valor central, isto é, 0,5. Isso pode ser visto analisando as linhas contínuas (verde e preta) da Figura 3 e Figura 4 que estão próximas, por vezes sobrepostas, o que ratifica a afirmação de que o estimador classificou bem os alunos das diferentes classes.

[3]Em uma tarefa de classificação binária, o algoritmo Floresta Aleatória utiliza uma probabilidade (média das probabilidades das árvores de decisão constituintes) para classificar uma amostra como positiva ou negativa. O limiar padrão é 0,5 (50%). Para ilustrar, se a probabilidade de um aluno ser aprovado é de 60% e o limiar for igual a 50%, então o aluno será classificado como aprovado. Note que ao aumentar o limiar aumenta-se o nível de confiança para classificar uma amostra como positiva. Por exemplo, o mesmo aluno seria classificado como reprovado se o limiar fosse igual a 70%.

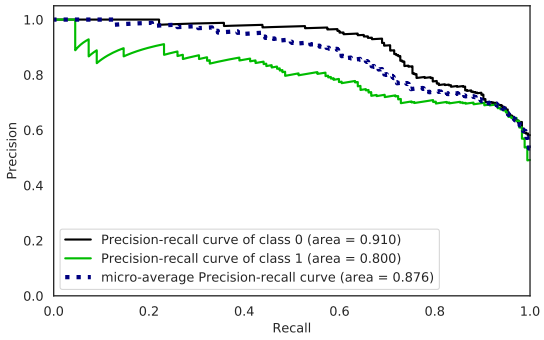

Figura 3. Precision e Recall Curve na base de validação.

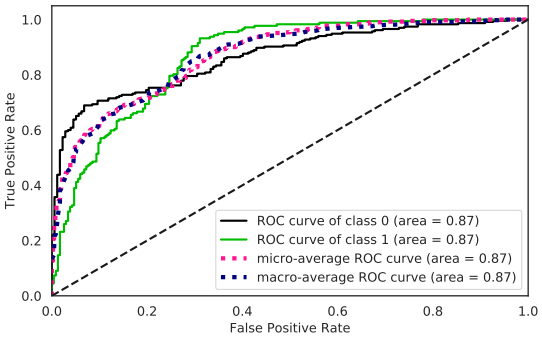

Figura 4. ROC Curve na base de validação.

Além disso, a Figura 6 apresenta a curva de aprendizagem do modelo preditivo em função da quantidade de instâncias de treinamento, utilizando validação cruzada com 10 partições. Observa-se que a partir do treino com 1500 instâncias, o modelo preditivo atinge uma acurácia entre 71% a 82% nas partições utilizadas para teste. O desempenho na base de validação já mostra que o classificador generalizou precisamente. Outra evidência disso é que as linhas contínuas no treino e teste da Figura 6 estão relativamente próximas.

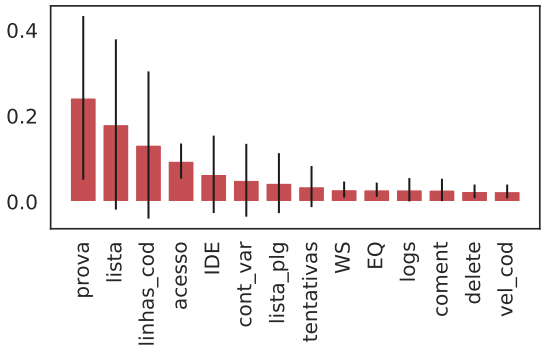

Figura 5. Importância dos atributos para a predição do desempenho.

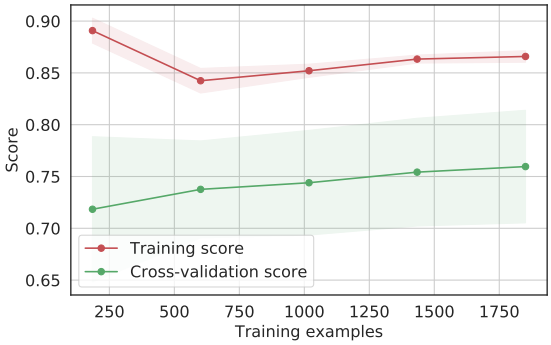

Figura 6. Curva de aprendizagem do modelo preditivo na base de treinamento.

Além de predizer o desempenho do aluno ainda no começo do curso com uma precisão desejável, principalmente no contexto educacional, é importante entender quais fatores podem ter influenciado o aluno a ser bem ou mal sucedido na disciplina de IPC. Desta forma, neste estudo foi realizada uma análise dos atributos mais relevantes para o processo de aprendizagem da Floresta Aleatória encontrada pelo TPOT. Para calcular a importância dos atributos, analisou-se o quanto de impureza[4] um nodo que usa um dado atributo reduz, levando em consideração também a quantidade de instância de treino que alcançam aquele nodo. Como o modelo preditivo utilizado é uma floresta com 100 árvores de decisão, o cálculo de contribuição dos atributos é feito com base na média das importância dos atributos nas árvores de decisão constituintes. Essa relevância pode ser

[4]A impureza foi mensurada utilizando o coeficiente Gini.

vista na Figura 5 que mostra a importância média (com intervalo de confiança) de cada atributo para cada árvore constituinte da Floresta Aleatória. Observa-se uma alta variação das importâncias dos atributos nas árvores constituintes da Floresta, o que é justificado pelo fato de apenas 30% dos atributos estarem disponíveis para a criação de cada divisão nos nodos das 100 árvore da Floresta Aleatória.

Apesar dessa variação, pode ser visto na Figura 5 que os dois atributos mais relevantes são a nota do prova realizada na segunda semana de aula e nota da lista de exercícios entregue na segunda semana, com uma contribuição média de 24% e 18%, respectivamente, o que era algo esperado. Com isso, observa-se que é necessário um cuidado especial com alunos que vão mal na primeira prova e na primeira lista de exercício. Percebe-se ainda uma contribuição alta para moderada ($[4\% - 14\%]$) dos atributos *linha_cod*, *acesso*, *lista_plg*, *cont_var*, *IDE*, *tentativas* e uma contribuição mais baixa ($[2\% - 3\%]$) dos atributos *coment*, *logs*, *EQ*, *WS*, *delete*, e *vel_cod*.

Com base nas importâncias dos atributos, pode-se analisar que os fatores iniciais que foram preponderantes para um bom desempenho do aluno foram a assiduidade no acesso ao ACAC, ter um bom desempenho nas listas de exercícios e na primeira prova, passar bastante tempo resolvendo os problemas (*IDE*) e ser resiliente, isto é, tentar muitas vezes resolver os exercícios (*tentativas*, *IDE*, *linhas_cod*, *cont_var*), mesmo quando errar. Destaca-se que este trabalho não está afirmando que esses fatores causam um bom desempenho por parte do aluno, mas que foi observada essa correlação no cenário educacional apresentado.

Por fim, para endossar a afirmação supracitada, a Tabela 1 mostra as médias e desvios padrão (dv) dos atributos em relação aos dados das duas primeiras semanas de aula agregadas pela situação do aluno, isto é, se ele foi aprovado ou reprovado. Foi aplicado o teste estatístico *MannWhitney* para verificar se existe diferença entre as distribuições dos alunos que foram aprovados e dos que foram reprovados. Em todos os casos as distribuições dos aprovados são estatisticamente significante ($p - value < 0,01$) maiores que as dos reprovados.

Tabela 1. Estatísticas descritivas dos dados das duas primeiras semanas de aula agrupadas pela situação (aprovado ou reprovado). Nota: dv significa desvio padrão.

	tentativas		**coments**		**linhas_cod**		**acesso**		**prova**		**delete**		**log**	
Situação	média	dv	média	dv	média	dv	média	dv	média	dv	média	dv	média	dv
Reprovado	50,53	52,78	19,63	23,71	79,36	59,08	26,13	31,27	2,57	4,05	29,45	28,3	238,21	191,81
Aprovado	83,05	62,93	36,19	30,28	143,35	42,34	42,76	28,71	7,6	3,77	37,01	24,6	319,73	165,89
	lista		**lista_plg**		**IDE**		**vel_cod**		**EQ**		**WS**		**cont_var**	
Situação	média	dv	média	dv	média	dv	média	dv	média	dv	média	dv	média	dv
Reprovado	3,65	2,68	2,41	2,09	94,86	94,25	2,4	1,3	10,98	18,64	7,53	20,14	1,81	1,3
Aprovado	6,49	1,3	4,48	1,67	164,85	89,13	2,75	0,76	11,42	15,97	9,27	20,85	2,67	0,58

6. Considerações Finais

Neste artigo foi apresentado um método de predição de desempenho de alunos de turmas de IPC que fazem uso de Juízes Online, onde buscou-se identificar nas duas primeiras semanas de aula os alunos que seriam e os que não seriam aprovados na disciplina. Para tanto, utilizou-se o conjunto de atributos de aprendizagem de máquina proposto por [Dwan et al. 2017] e um método que emprega algoritmos genéticos para a automatização da construção e otimização de pipelines de aprendizagem de máquina proposto por [Olson et al. 2016].

Para realizar a predição de desempenho, foram utilizados logs de baixa granularidade gerados pelos alunos durante a construção de seus códigos, e não somente aos códigos-fonte submetidos para avaliação do juiz online. Assim, uma avaliação do aluno baseada nesses dados é mais formativa que uma avaliação baseada apenas no corretude do código-fonte submetido, uma vez que leva em consideração os desafios e dificuldades enfrentadas pelos alunos durante a construção das soluções.

Frisa-se ainda que, conforme reportado por [Alamri et al. 2019], identificar o desempenho do aluno de forma antecipada (ainda nas duas primeiras semanas de aula) é algo relevante por diversas razões, dentre as quais ressalta-se: (i) a partir dessas informações, os professores podem direcionar orientações específicas aos discentes com dificuldades; (ii) os estudantes com dificuldades podem ser monitorados, a fim de evitar a reprovação ou a evasão do curso; (iii) os professores podem prover atividades mais desafiadoras aos alunos com alto desempenho.

Ainda, demonstrou-se que os algoritmos genéticos podem ser empregados para a produção automática de *pipelines* de AM para a predição precoce do desempenho de alunos de turmas de IPC. Acredita-se que automatizar o processo de criação dos modelos preditivos é algo importante neste contexto, pois ao longo dos semestres os padrões e atributos de aprendizagem de máquina para realizar a predição do desempenho dos alunos poderão mudar. Dessa forma, um método que constrói automaticamente o pipeline de aprendizagem de máquina se mostra como uma alternativa viável para substituir um cientista de dados na produção desses modelos preditivos que acompanhem essas mudanças. Observe que essa automação através do algoritmo genético apoia um sistema adaptativo para a predição de desempenho de alunos de IPC, isto é, que possa ser facilmente retreinado à medida que surjam novos dados de usuários.

Finalmente, as maiores limitações deste trabalho estão relacionada à base de dados utilizada. Apesar de haver uma quantidade expressiva de alunos, os dados foram coletados em uma única instituição, o que pode influenciar no poder de generalização do método. No entanto, a diversidade dos dados pode ser justificada em função deles terem sidos coletados a partir de turmas de 9 cursos diferentes. Como parte de trabalhos futuros, pretende-se utilizar dados de diferentes instituições de ensino e realizar intervenções com recomendações automáticas (dicas, conteúdo, problemas) baseadas nos comportamentos de programação mensurados pelos atributos compilados neste estudo, a fim de otimizar o processo de aprendizagem e alavancar a qualidade da educação em disciplinas de IPC apoiadas por juízes online.

Referências

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

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197.

Dwan, F., Oliveira, E., and Fernandes, D. (2017). 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)*, volume 28, page 1507.

Francisco, R., Júnior, C. P., and Ambrósio, A. P. (2016). 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)*, volume 27, page 11.

Galvão, L., Fernandes, D., and Gadelha, B. (2016). 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)*, volume 27, page 140.

Jadud, M. C. (2006). Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research*, pages 73–84. ACM.

Munson, J. P. and Zitovsky, J. P. (2018). Models for early identification of struggling novice programmers. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 699–704, New York, NY, USA. ACM.

Olson, R. S., Bartley, N., Urbanowicz, R. J., and Moore, J. H. (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 485–492. ACM.

Pereira, F. D., Oliveira, E., Cristea, A., Fernandes, D., Silva, L., Aguiar, G., Alamri, A., and Alshehri, M. (2019). Early dropout prediction for programming courses supported by online judges. In *International Conference on Artificial Intelligence in Education*, pages 67–72. Springer.

Pereira, F. D., Oliveira, E., and Oliveira, D. (2018). Uso de um método preditivo para inferir a zona de aprendizagem de alunos de programação em um ambiente de correção automática de código. *Dissertação (Mestrado em Ciência da Computação) – Instituto de Computação, Universidade Federal do Amazonas. Manaus*.

Souza, D. M., Felizardo, K. R., and Barbosa, E. F. (2016). A systematic literature review of assessment tools for programming assignments. In *Software Engineering Education and Training (CSEET), 2016 IEEE 29th International Conference on*, pages 147–156. IEEE.

Watson, C., Li, F. W. B., and Godwin, J. L. (2013). Predicting performance in an introductory programming course by logging and analyzing student programming behavior. *Proceedings - 2013 IEEE 13th International Conference on Advanced Learning Technologies, ICALT 2013*, pages 319–323.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85.

Zutty, J., Long, D., Adams, H., Bennett, G., and Baxter, C. (2015). Multiple objective vector-based genetic programming using human-derived primitives. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1127–1134. ACM.