# Predição de Zona de Aprendizagem de Alunos de Introdução à Programação em Ambientes de Correção Automática de Código

**Filipe Dwan Pereira** [1], **Elaine Harada Teixeira de Oliveira** [2], **David Fernandes de Oliveira** [2]

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

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

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

***Abstract.*** *The present work proposes and validates a method to infer the learning zone of CS1 students in an online judge. To do so, the student's grade will be predicted, using a programming profile based on the data left by them as they solve exercises in that system. Students who scored lower than 5 were classified in a difficulty zone, otherwise in an expertise zone. Machine learning algorithms were used to make the prediction. The proposed predictive model obtained an accuracy of 78.3 % at the first two weeks of class, which overcomes research results that were conducted in similar scenarios.*

***Resumo.*** *Este trabalho tem por objetivo propor e validar um método para inferir a zona de aprendizagem de alunos de turmas de Introdução à Programação (IPC) em ambientes de correção automática de código (ACAC). Para tanto, foi construído um perfil de programação baseado nos dados deixados pelos estudantes à medida que eles resolvem exercícios nesses sistemas. Os alunos que tiraram notas inferiores a 5 foram classificados em uma zona de dificuldade, do contrário em uma zona de expertise. Utilizou-se algoritmos de aprendizagem de máquina para fazer a predição. O modelo preditivo construído obteve 78,3% de acurácia já nas duas primeiras semanas de aula, o que ultrapassa os resultados de pesquisas que foram conduzidas em cenários semelhantes.*

## 1. Introdução

Estudos recentes sugerem que, em média, um terço dos alunos no mundo reprova em disciplinas de Introdução à Programação de Computadores (IPC) [Watson and Li 2014]. Com efeito, muitas instituições vêm utilizando a metodologia de aprendizagem híbrida para alavancar o processo de ensino e aprendizagem. Nessa metodologia, faz-se uso de sistemas *online* aliados às aulas presenciais [Galvão et al. 2016]. Para ilustrar, muitas instituições de ensino superior vêm utilizando ambientes de correção automática de código (ACAC), também conhecidos como juízes *online*. Muitos desses ambientes possuem Editores de Código Fonte (ECF) integrados, o que possibilita ao estudante resolver os exercícios propostos pelos professores e receber suas respectivas correções no próprio ACAC.

Essas ferramentas abriram novas oportunidades de pesquisa, posto que através delas é possível registrar todas as ações realizadas pelos alunos durante suas tentativas de solucionar os exercícios de programação propostos. Dessa forma, os tipos de análises possibilitadas por tais dados permitem que o docente não avalie tão somente o produto

DOI: 10.5753/cbie.sbie.2017.1507 

enviado pelo aluno, mas todo o processo por trás dele, ou seja, permite uma avaliação formativa [Blikstein et al. 2014].

Nesse caminho, o presente trabalho propõe o uso de técnicas de aprendizagem de máquina sobre tais dados para identificar a zona de aprendizagem dos alunos de turmas IPC, antes das avaliações programadas para essas turmas. Para tanto, é proposto e avaliado um método de predição das notas dos alunos, baseado nos registros das atividades de programação deixados por eles à medida que vão resolvendo exercícios propostos em ACAC. Se a nota prevista para um dado aluno for maior ou igual que 5, o aluno é classificado como pertencente a uma Zona de Expertise; por outro lado, se a nota prevista for menor que 5, o aluno é classificado como pertencente a uma Zona de Dificuldade.

Os modelos usados para predição de notas são construídos a partir de dados gerados há alguns dias ou semanas da avaliação cuja nota deseja-se prever. Para viabilizar essa classificação, 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 adotadas técnicas de seleção de atributos para identificar as evidências mais importantes para o processo de predição de notas. Os atributos mais relevantes foram utilizados na composição de modelos preditivos.

O modelo preditivo proposto neste trabalho atingiu 78,3% de acurácia já nas duas primeiras semanas de aula, em um experimento com a base de dados desbalanceada. Em outro experimento com a base balanceada, a precisão foi de 72,8%. Saber antecipadamente o desempenho dos alunos pode ser útil para os professores, que podem direcionar um tipo de orientação específica para os discentes com alta probabilidade de desempenho baixo, além de prover atividades mais desafiadoras aos alunos classificados na Zona de Expertise.

Para apresentar o método proposto, o presente artigo foi dividido em 6 seções. A seção 2 apresenta trabalhos relacionados que também realizaram predição de desempenho usando dados coletados a partir de turmas de IPC. A seção 3 apresenta o método proposto. A seção 4 descreve a metodologia utilizada na condução dos experimentos. A seção 5 mostra os resultados e a seção 6 apresenta as considerações finais e trabalhos futuros.

## 2. Trabalhos Relacionados

Segundo [Peterson et al. 2015], métricas de *software*[1] baseadas no progresso do estudante, código ou comportamento são consideradas um caminho para revolucionar a educação. Mesmo uma heurística simples como a recorrência de um mesmo erro subsequente de compilação tem o potencial para identificar estudantes com dificuldade.

Nesse sentido, [Jadud 2006] detectou um ciclo de edição, compilação e execução de códigos dos alunos em turmas de IPC que usavam a linguagem Java. Baseado nisso, o autor propôs um algoritmo para quantificar os erros de compilação usando a métrica *Error Quotient* (EQ). Essa métrica é baseada no número de submissões subsequentes que retornam o mesmo erro de compilação, e é capaz de indicar a dificuldade ou a destreza do aluno em corrigir os eventuais erros encontrados. O autor verificou que o EQ possui uma correlação com a nota do aluno. Nesse mesmo caminho, [Watson et al. 2013] criou

[1]Métricas de *software* é um termo utilizado na engenharia de *software* para calcular o esforço e o custo de desenvolvimento de um sistema e, no contexto da presente pesquisa, elas irão ajudar a mensurar o esforço do aluno para resolver questões em um ACAC.

uma derivação da métrica EQ, levando em consideração o tempo que o aluno levava para consertar o erro. O algoritmo proposto por Watson, chamado *Watwin Score*, obteve uma acurácia média de 68,8% ao longo do curso e atingiu 75,6% no final do curso, enquanto no trabalho de Jadud os resultados foram de 55,8% de acurácia média e de 60,0% ao final do curso, ambos para predição de desempenho [Watson et al. 2013].

O trabalho de [Estey and Coady 2016] apresentou o BitFit[2], uma ferramenta desenvolvida para que alunos de turmas IPC possam fazer atividades semanais. A ferramenta, além de ser um ACAC, disponibiliza uma série progressiva de dicas para cada atividade. O estudo teve por intento encontrar padrões sutis de aprendizagem, utilizando dados como frequência de consumo de dicas, submissões, compilações, etc. Com esses valores, foi construído um modelo preditivo que fazia uma classificação binária para identificar se o estudante iria passar ou não na disciplina. O modelo preditivo apresentado obteve uma acurácia média de 81% ao final do curso, entretanto com uma taxa de identificação de 30% dos alunos que reprovaram nas duas primeiras semanas.

[Ahadi et al. 2016] exploraram o relacionamento entre o desempenho nos exercícios realizados pelos estudantes e a sua subsequente nota em uma avaliação. O estudo relata que o número de tentativas, independente se o aluno conseguiu responder a atividade corretamente, possui correlação com a nota do aluno na prova. 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. O autor relata que começar a estudar perto do prazo final das atividades está relacionado a um baixo desempenho. 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.

O estudo de [Leinonen et al. 2016] analisou a correlação da *keystroke latency*[3] e da quantidade de dígrafos específicos digitados com a situação do aluno, isto é, se ele vai reprovar ou não. Como resultado, os autores obtiveram 65.8% de acurácia nas duas primeiras semanas e 71,8% na sétima semana. Os autores explicam que existe uma baixa correlação da *keystroke latency* com a nota nas primeiras semanas.

Finalmente, no trabalho de [Otero et al. 2016] foi apresentada uma série de métricas de software baseadas na análise estática dos códigos submetidos por discentes de turmas de IPC em ACAC, como o número médio de linhas de código, comentários, caracteres, blocos e etc. Tais métricas apresentaram correlação com a nota do estudantes e podem assim ser usadas como atributos de algoritmos de aprendizagem de máquina.

## 3. Método de Predição de Zona de Aprendizagem

O método proposto nesta pesquisa usa evidências extraídas das tentativas de resolução de questões de listas de exercícios realizadas pelos alunos em um ACAC. Uma lista é composta por exercícios de programação para turmas de IPC e sempre precede uma prova. Usando o conjunto de evidências obtido durante as tentativas de resolução de uma dada lista, o objetivo deste trabalho é desenvolver um modelo preditivo capaz de prever as zonas de aprendizagem dos alunos durante a prova subsequente. Neste trabalho, as zonas de

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

[3]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.

aprendizagem são classificadas em: a) Zona de Dificuldade (ZD) para os alunos que irão tirar notas menores que 5 na próxima prova e b) Zona de Expertise (ZE) para os alunos que irão tirar notas maiores ou iguais a 5 na próxima prova.

Tais evidências são derivadas de registros em diferentes níveis de granularidade, isto é, eles podem ser coletados em frequências e tamanhos 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. Por outro lado, alguns ambientes registram as mensagens de compilação e execução geradas e outras gravam, de forma menos granular, as submissões dos estudantes [Ihantola et al. 2015].

Para ilustrar, a parte a) da Figura 1 apresenta um modelo típico de ACAC, no qual os alunos resolvem questões criadas pelos professores, em um ECF integrado ao ACAC que possibilita a edição do código, execução, teste, compilação 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, quais caracteres foram digitados, quais foram apagados, quais foram colados da área de transferência, quantas vezes o usuário saiu do ECF e quantas vezes voltou, quantas tentativas de submissão juntamente com um *timestamp* de cada evento.

Como mostrado na parte b) da Figura 1, essas e outras informações são usadas pelo método proposto neste trabalho para modelar cada estudante como um conjunto de valores numéricos que representam várias facetas do estilo de programação 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 capazes de classificar os alunos nas zonas ZD e ZE.

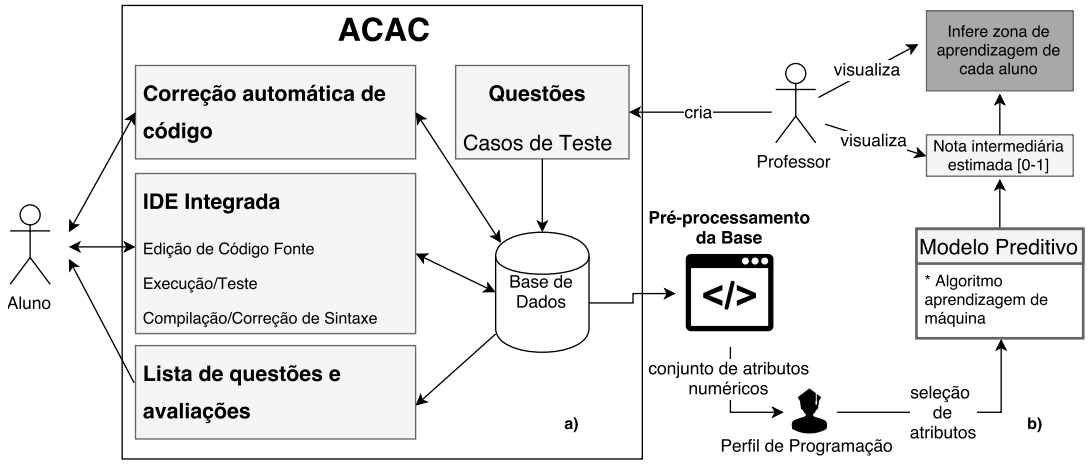


**Figura 1. Arquitetura do Método de Predição de Zona de Aprendizagem. Parte a) adaptada de [Ihantola et al. 2015]**

Frisa-se a complexidade desse problema, uma vez que a ideia deste trabalho é inferir um acontecimento futuro (zona de aprendizagem na prova) usando evidências do passado (comportamento do aluno durante a resolução das listas). Para ilustrar essa complexidade, um aluno pode estudar bastante nos últimos dias que antecedem a prova e então ter um nível de dificuldade diferente daquele que tinha durante a resolução da lista de exercícios.

### 3.1. Perfil de Programação

Para compor o perfil de programação dos alunos, foram utilizadas evidências baseadas nos códigos fontes submetidos pelos alunos e nos logs deixados por eles à medida que resolviam os exercícios de cada lista. Os logs são registros datados que especificam a ação realizada pelo estudante no ECF integrado ao ACAC utilizado no experimento. Os registros podem ser uma ação de inserção ou remoção de um caractere, um redimensionamento do ECF, uma ação de colar um texto que vem da área de transferência, um clique do botão direito do mouse, a perda de foco da caixa de edição, etc.

Antes de submeter um código fonte para a avaliação do juiz *online*, o aluno pode testá-lo com uma entrada própria, de forma a verificar se o código apresenta algum erro sintático ou se a saída de seu programa condiz com a saída esperada para a entrada informada. Dito isso, os seguintes atributos foram selecionados para compor os perfis de programação dos alunos:

- **num_comentarios** (M1): média de número de linhas com comentários nos códigos submetidos. *Strings* em mais de uma linha não são contadas como comentários e sim como num_multi (M3) [Otero et al. 2016];
- **num_linhas_branco** (M2): média de número de linhas em branco nos códigos [Otero et al. 2016];
- **num_multi** (M3): média de número de *strings* que ficam em mais de uma linha nos códigos submetidos;
- **loc** (M4): média de número de linhas dos códigos submetidos [Otero et al. 2016];
- **lloc** (M5): média de número de linhas lógicas dos códigos, sem contabilizar importação de bibliotecas, comentários e linhas em branco [Otero et al. 2016];
- **tempo_uso_ecf** (M6): tempo total (em minutos) que o aluno passa dentro do ECF tentando solucionar as questões de uma dada lista de exercícios, ou seja, digitando algo [Auvinen 2015];
- **code_ratio** (M7): a razão entre loc e o tempo_uso_ecf [Otero et al. 2016];
- **num_linhas_logs** (M8): média do número de linhas do log gerado por cada questão de uma dada lista de exercícios [Leinonen et al. 2016];
- **nota_lista** (M9): nota do aluno em cada lista de exercícios [Ahadi et al. 2016];
- **nota_lista_esforço** (M10): nota do aluno em cada lista de exercícios, considerando apenas questões resolvidas que geraram no mínimo 50 linhas no log;
- **num_acessos** (M11): número de acessos (logins) que cada aluno faz dentro do prazo de entrega de uma dada lista;
- **num_tentativas** (M12): número de tentativas de submissão, independente se o código está certo ou errado [Ahadi et al. 2016];
- **nota_avaliacao_ant** (M13): nota na avaliação anterior (quando não é a primeira avaliação);
- **coeficiente_variacao** (M14): razão entre o desvio padrão do número de tentativas de submissão em cada lista de exercícios pela média de tentativas de submissão;
- **cresc_engajamento** (M15): o valor desta evidência é 0 quando o número de tentativas entre duas listas de exercícios subsequentes diminui, e 1 caso contrário;
- **media_testes** (M16): número total de testes feitos durante as tentativas de solucionar todas as questões de uma lista dividido pelo número total de questões da lista;
- **prop_colados_digitados** (M17): proporção entre número de caracteres colados (com CTRL+V) e número caracteres digitados;

- **velocidade_digitacao** (M18): velocidade de digitação de um dado aluno durante suas tentativas de solucionar as questões de uma dada lista de exercícios [Leinonen et al. 2016];
- **media_deletes** (M19): média de caracteres apagados através do uso das teclas *delete* e *backspace* em cada lista de exercícios;
- **media_sucesso** (M20): razão entre o número de questões corretamente avaliadas pelo número de tentativas de submissão [Auvinen 2015];
- **erro_sintaxe** (M21): proporção entre o número total de submissões e o número total de submissões com erros de sintaxe [Jadud 2006];
- **dificuldade_reportada** (M22): 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.

## 4. Cenário Experimental

Nesta seção será descrito o contexto educacional e os dados nos quais o método proposto neste trabalho foi aplicado. Além disso, será apresentado o processo de construção dos modelos preditivos que utilizaram o perfil de programação, as ferramentas e implementações usadas e uma descrição do processo de separação das bases de dados para treinamento e teste dos estimadores.

### 4.1. Contexto Educacional e Dados

Para a validação do método de predição proposto, foram coletados dados de 9 turmas de IPC da Universidade Federal do Amazonas (UFAM), 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 em um ACAC chamado Codebench[4] desenvolvido pela UFAM, o qual segue a mesma configuração mostrada na Figura 1 a). 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 notas 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.

### 4.2. Construção do Modelo Preditivo

A Figura 2 ilustra o processo de construção do modelo preditivo proposto neste trabalho. Primeiramente, foi realizado o pré-processamento dos dados, onde códigos submetidos e

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

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.

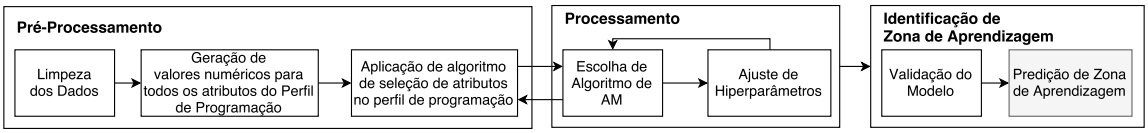


**Figura 2. Fluxograma de aplicação do método proposto.**

Para encontrar os subconjuntos de atributos mais relevantes no processo de seleção de atributos foi utilizado o algoritmo de busca guloso *BestFirst*, enquanto que a avaliação desses subconjuntos foi realizada pelo algoritmo *CfsSubsetEval*, ambos na ferramenta Weka [Hall et al. 2009]. Além disso, foram utilizados módulos da biblioteca *scikit-learn* para a implementação dos modelos preditivos, em função da facilidade para manipular os hiperparâmetros dos estimadores. Especificamente, foi utilizada a técnica *GridSearchCV*, que é uma busca exaustiva com todas as possibilidades sobre valores específicos de um modelo preditivo [Pedregosa et al. 2011], onde o retorno são os melhores hiperparâmetros encontrados para o estimador.

Os algoritmos de classificação utilizados para a construção dos modelos preditivos foram o *Support Vector Machine* (SVM), *Random Forest* (RF), *AdaBoosting* (AB), Árvore de Decisão (AD) e *K-Nearest Neighbours* (KNN).

Os modelos preditivos foram validados aplicando o método de validação cruzada com 10 partições (*folds*). Esse método divide a base em $k$ partições, usando $k$-1 para treino e 1 para teste. Após isso, calcula-se a acurácia na partição de teste. Esse processo é repetido $k$ vezes, até que todas as partições tenham sido usadas como teste. Finalmente, computa-se a média das acurácias obtidas nos testes [Pedregosa et al. 2011].

## 5. Experimentos e Resultados

Foram realizados dois experimentos, sendo o primeiro com a base de dados desbalanceada e o segundo com a base balanceada, através de uma técnica de subamostragem. Nos dois experimentos foram removidos os alunos que faltaram à avaliação parcial naquela sessão, para não gerar resultados tendenciosos.

A Tabela 1 mostra quais evidências do perfil de programação foram selecionadas pelos algoritmos de seleção de atributos em cada sessão dos dois experimentos.

**Tabela 1. Atributos selecionados em cada sessão dos experimentos 1 e 2.**

| | Sessão 1 | Sessão 2 | Sessão 3 | Sessão 4 |
|---|---|---|---|---|
| **Experimento 1** | M8, M9, M11, M16, M19 | M5, M9, M10, M11, M13, M14, M16 | M6, M8, M9, M10, M11, M12, M13, M14 | M8, M16, M17, M19, M20 |
| **Experimento 2** | M2, M9, M11, M16, M19, M21, M22 | M4, M5, M6, M7, M9, M10, M11, M12, M13, M14, M16, M21, M22 | M1, M2, M5, M6, M7, M8, M10, M11, M12, M13, M15, M17, M19, M20, M21 | M1, M10, M12, M15, M19 |

Além da acurácia, os resultados obtidos neste trabalho também são avaliados através das métricas *recall* e *precision*, que são mais adequadas para bases desbalanceadas. Tais métricas são baseadas nas quantidades de verdadeiros 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].

Note que, no contexto deste trabalho, a métrica *recall* mostra o quão acurado o modelo está em identificar os alunos da ZE (*recall* positivo) ou da ZD (*recall* negativo), enquanto que a métrica *precision* apresenta a taxa de acerto quando o modelo afirma que o aluno está na ZE (*precision* positivo) ou na ZD (*precision* negativo).

### 5.1. Experimento 1 - Base de Dados Desbalanceada

O primeiro experimento foi conduzido com a base desbalanceada. A Figura 3 apresenta a quantidade de alunos na ZD e ZE em cada uma das 4 sessões descritas na subseção 4.1.

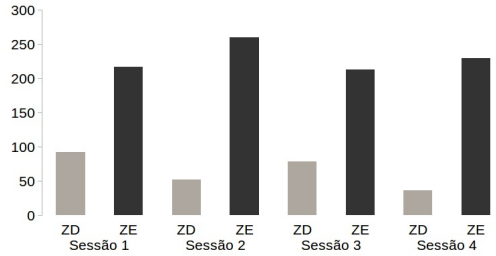


**Figura 3. Quantidade de alunos na ZE e ZD em cada sessão.**

A Tabela 2 mostra a acurácia, *precision* e *recall*, tanto dos positivos quanto dos negativos. As maiores acurácias dos modelos preditivos foram nas sessões 2 e 4, que são as mais desbalanceadas de nossa base. Destaca-se que os modelos estão conseguindo identificar os alunos da ZE de forma contundente, já que o *recall* e o *precision* positivo também são valores significativos.

Por outro lado, o baixo número de alunos na ZD justifica o *recall* negativo menor do que o *recall* positivo 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.

**Tabela 2. Apresentação da acurácia, *recall* dos positivos, *precision* dos positivos, *recall* dos negativos, *precision* dos negativos dos algoritmos de aprendizagem de máquina com melhores resultados (em negrito). Base de dados Desbalanceada.**

| | 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 |
| Rec. Pos.(%) | **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 |
| Rec. Neg.(%) | 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 |

### 5.2. Experimento 2 - Base de Dados Balanceada

No experimento 2, todas as sessões tinham um número igual de alunos na ZE e ZD. Assim, as sessões possuíam 184, 104, 158, e 72 instâncias, respectivamente, sendo metade para cada classe.

A Tabela 3 mostra a avaliação dos modelos preditivos com maior acurácia. Os algoritmos com maior precisão para cada sessão foram o SVM, RF, RF e KNN, respectivamente. Percebe-se que os valores da acurácia diminuíram, enquanto que os valores do *recall* e *precision* negativo subiram, se compararmos com o experimento 1. 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.

**Tabela 3. Avaliação dos Modelos Preditivos mais Precisos Com a Base de Dados Balanceada.**

| Sessão | Acc. | *Recall* Pos. | Prec. Pos. | *Recall* Neg. | Prec. Neg. |
|---|---|---|---|---|---|
| 1 | 72.8% | 66.3% | 76.3% | 79.3% | 70.2% |
| 2 | 71.2% | 76.9% | 68.9% | 63.4% | 73.9% |
| 3 | 74.7% | 71.8% | 75.7% | 77.5% | 73.8% |
| 4 | 72.2% | 83.3% | 68.2% | 61.1% | 78.6% |

## 6. Condiderações Finais e Trabalhos Futuros

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. Os itens desse conjunto foram usados como atributos de algoritmos de aprendizagem de máquina para a construção de modelos preditivos. Tais modelos obtiveram resultados expressivos.

Para ilustrar, o método proposto alcançou nas duas primeiras semanas acurácia de 78.3% no experimento 1 e 72.8% no experimento 2, enquanto que [Leinonen et al. 2016] obtiveram sua maior acurácia na semana 8, sendo 71,8%. Já [Estey and Coady 2016] atingiu 81% de taxa média de reconhecimento de alunos na ZD ao longo do experimento, entretanto os autores obtiveram 30% nas duas primeiras semanas de aula, enquanto o método proposto atingiu 79,3% de *recall* negativo da sessão 1 do experimento 2. Vale mencionar que os métodos propostos por [Leinonen et al. 2016, Estey and Coady 2016] foram aplicados em bases de dados diferentes desta pesquisa. Entretanto, ambos foram realizados em cenários educacionais semelhantes ao deste estudo.

Além disso, acredita-se que os atributos do perfil de programação não são sensíveis 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.

Como trabalho futuro, pretende-se replicar esse estudo em uma outra instituição de ensino, com um formato de avaliação diferente. Além disso, realizar um estudo longitudinal, aplicando o método apresentado e verificando o resultado da intervenção do instrutor à medida que ele sabe antecipadamente a zona de aprendizagem do aluno.

## Referências

Ahadi, A., Vihavainen, A., and Lister, R. (2016). 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*, pages 218–223.

Auvinen, T. (2015). Harmful study habits in online learning environments with automatic assessment. In *Learning and Teaching in Computing and Engineering (LaTiCE), 2015 International Conference on*, pages 50–57. IEEE.

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., and Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 23(4):561–599.

Estey, A. and Coady, Y. (2016). 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*, pages 236–241.

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.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S. H., Isohanni, E., Korhonen, A., Petersen, A., Rivers, K., et al. (2015). Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*, pages 41–63. ACM.

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.

Leinonen, J., Longi, K., Klami, A., and Vihavainen, A. (2016). Automatic inference of programming performance and experience from typing patterns.

Otero, J., Junco, L., Suarez, R., Palacios, A., Couso, I., and Sanchez, L. (2016). Finding informative code metrics under uncertainty for predicting the pass rate of online courses. 373:42–56.

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., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Peterson, A., Spacco, J., and Vihavainen, A. (2015). An exploration of error quotient in multiple contexts. *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, pages 77–86.

Watson, C. and Li, F. (2014). Failure rates in introductory programming revisited. *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 39–44.

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