Proposta para Avaliação de Códigos Fonte com TF-IDF

Ricardo Lemos de Souza[1], **Fabiana Zaffalon Ferreira**[2], **Silvia da Costa Botelho**[3]

[1]PPG Modelagem Computacional, Universidade Federal do Rio Grande - FURG, Rio Grande, RS, Brasil

[2]PPG Educação em Ciências, Universidade Federal do Rio Grande - FURG, Rio Grande, RS, Brasil

[3]Centro de Ciências Computacionais, Universidade Federal do Rio Grande - FURG, Rio Grande, RS, Brasil

rcrdsou@hotmail.com,{fabinhazaffalon,silviacbbotelho}@gmail.com

Abstract. *The growing demand for professionals capable of creating and maintaining software also leads to an increase in programming teaching courses. This work proposes a model for evaluating source codes produced in programming courses. Uses TF-IDF to identify and estimate computational thinking skills in source codes. The results show to be a promising assessment approach for comparing the skills present in different submissions.*

Resumo. *A crescente demanda de profissionais capazes de elaborar e manter softwares acarreta também no aumento de cursos voltados ao ensino de programação. O presente trabalho propõe um modelo de avaliação de códigos fonte produzidos em cursos de programação. O modelo é baseado em TF-IDF para identificar e estimar habilidades do pensamento computacional em códigos fonte. Os resultados demostram ser uma abordagem de avaliação promissora na comparação de habilidades em diferentes fontes.*

1. Introdução

A presença da computação cresce nos mais diferentes aspectos da contemporaneidade. Seja em computadores, telefones inteligentes, veículos ou eletrodomésticos capazes de conectar-se à internet, é cada vez maior o número de objetos que possuem algum tipo de software embarcado [SU et al. 2016]. A maior presença da computação em todos os setores acarreta também em um aumento na demanda de profissionais capazes de desenvolver e manter softwares.

Cursos introdutórios de programação possuem reconhecidamente um alto número de taxas de reprovação e evasão [RAIGOZA 2017]. O aprendizado de programação de computadores engloba o desenvolvimento de um conjunto de habilidades como lógica, matemática, assim como outras não diretamente relacionados com as ciências da computação [ROBINS 2010]. Os novatos em programação deparam-se não apenas com o desafio de aprender uma linguagem de programação, mas também em como aplicar seus conhecimentos prévios em uma solução algorítmica. A dificuldade de relacionar diferentes habilidades para esse fim é apresentada como uma das principais justificativas para os níveis de reprovação e evasão [CABO 2019].

DOI: 10.5753/cbie.sbie.2020.112

Em cursos presenciais, editores de código fonte são amplamente utilizados nas disciplinas de programação e, com o aumento do número de estudantes, o número de códigos resultado das atividades realizadas pelos alunos pode gerar uma demanda de trabalho elevada para os educadores. Em consequência disso, é comum a busca por um modelo de avaliação automática de códigos fonte [ALA-MUTKA 2005].

Ainda, para atender à crescente demanda por programadores, observa-se a propagação de novos ambientes de ensino de programação, como cursos em plataformas *online*, as quais constantemente procuram desenvolver metodologias para proporcionar um melhor ensino bem como metodologias mais precisas de avaliação [SOUZA et al. 2016]. Cursos e plataformas Web oferecem a oportunidade de avaliar automaticamente os códigos fonte gerados pelos seus estudantes, e muitas plataformas desenvolveram seus próprios sistemas de avaliação [SOUZA et al. 2016].

Métodos de avaliação automática são comuns em questões objetivas, entretanto o universo de respostas possíveis em um código de programação assemelha-se à questões dissertativas, ou seja, mesmo que dois códigos fonte sejam diferentes eles podem chegar ao mesmo resultado. Diversos trabalhos na literatura abordam a avaliação automática de códigos [ALA-MUTKA 2005], sendo que as abordagens podem ser classificadas em dinâmica e estática [ULLAH et al. 2018b].

O presente trabalho tem por objetivo propor uma metodologia das avaliações de habilidades de estudantes associadas à escrita de um código em uma linguagem formal de programação. O modelo proposto foi aplicado em um estudo de caso associado ao desenvolvimento de códigos na linguagem "C". Destaca-se como contribuição deste artigo um modelo para identificação e mapeamento de habilidades associadas ao Pensamento Computacional presentes em códigos fonte, bem como sua implementação em ferramenta computacional capaz de ser usada em larga escala para avaliação de códigos em cursos de computação. Para a extração de informações sobre as habilidades de estudantes de programação a partir de códigos fonte, escolheu-se aplicar uma abordagem de recuperação de informações embasada em *Term Frequency - Inverse Document Frequency*. Como referência de habilidades, utilizou-se conceitos relacionados ao pensamento computacional [BARR and STEPHENSON 2011]. Nas seções a seguir, apresenta-se os trabalhos relacionados, a metodologia proposta e resultados obtidos a partir da aplicação da metodologia em um conjunto de dados.

2. Trabalhos Relacionados

A programação de computadores é um dos fundamentos centrais de cursos associados à ciência da computação [ALA-MUTKA 2005], e no mundo contemporâneo, também se faz presente no currículo de outras áreas. Com a crescente demanda de programadores, e consequentemente crescente oferta de cursos, diferentes metodologias de avaliação das produções de alunos de programação tem sido desenvolvidas. A seguir destacam-se alguns trabalhos que abordam essas metodologias, em especial as metodologias de avaliação automática de códigos fonte.

Além de listar os 17 principais sistemas que utilizam avaliação automática de algoritmos, o trabalho de ULLAH et al.(2018b) faz uma análise sobre a principal contribuição das literaturas relacionadas aos métodos de avaliação automática (AA) de códigos fonte. A literatura contemporânea classifica os métodos de AA em: (i) dinâmicos, aqueles que

contemplam a execução do código fonte; (ii) estáticos, aqueles embasados na análise de métricas de software, sem a necessidade da execução do código; e (iii) híbridos, uma combinação de estático e dinâmico.

A avaliação estática de códigos fonte requer que ele seja representado de uma forma quantificável, na qual um ferramental estatístico possa ser aplicado. O trabalho de AZCONA et al. (2019) demonstra esses processos de transformação de dados, para posterior uso em um modelo de classificação de perfis de estudantes.

Com o propósito de verificar a existência de plágio entre dois códigos fonte escritos em linguagens diferentes, KARNALIM (2020) utilizou-se de uma técnica inspirada em TF-IDF (*term frequency inverse document frequency*). Analisa-se a importância dos tokens não somente no contexto da resposta individual, como também no conjunto, utilizando-se do TF-IDF para esse balanceamento.

Além da acurácia e eficácia de um algoritmo, outros aspectos presentes em um código fonte são objetos de estudos educacionais. Em um trabalho que apresenta uma pesquisa bem ampla sobre as metodologias de avaliação automática e semiautomática de códigos fonte, ALA-MUTKA (2005) aborda o tema de um ponto de vista educacional. Faz considerações quanto as diferentes finalidades de uma avaliação automática, as dificuldades encontradas por educadores no uso dessas ferramentas, e os pontos fortes e fracos de suas aplicações.

O pensamento computacional (PC) trata do conjunto de habilidades relacionadas a uma abordagem de resolução de problemas análoga ao modo como um computador opera, i.e., "O processo de pensamento envolvido na formulação de um problema e na expressão de sua solução de forma que um computador – humano ou máquina – possa efetivamente realizar"[WING 2006] .

Ainda segundo WING (2006), o processo de construção do pensamento computacional baseia-se em três estágios: formulação de um problema (abstração), expressão de uma solução (automação) e a execução da solução e análise (análise). A abstração é a chave do pensamento computacional, e o desenvolvimento desse pensamento se dá não apenas na elaboração de soluções, mas também na elaboração de problemas.

BARR e STEPHENSON (2011) realizam um estudo de como inserir as competências do pensamento computacional no currículo do K-12. Nesse processo, com a ajuda de especialistas, relaciona as competências do pensamento computacional com diversas áreas do currículo, entre elas as ciências da computação, e nesse contexto com estruturas presentes em um código fonte. Nesse trabalho, os autores apresentam as capacidades ou atividades relacionadas de cada área relacionadas com 9 conceitos elaborados a partir do pensamento computacional de WING (2006). Para a ciência da computação, as capacidades ou atividades elencadas envolvem estruturas ou elementos que podem ser observados em um código fonte como uso de comandos de repetição, condicionais e definição de funções.

3. Metodologia

Métodos de extração de informações têm sido utilizados com sucesso na avaliação de códigos fonte. Nesse sentido, o uso de técnicas de processamento de linguagem natural demostram-se apropriados, e serão utilizados na metodologia proposta, como foram

nos trabalhos de AZCONA et al. (2019), ULLAH et al. (2018a) e GANGULY et al. (2018), para a transformação da linguagem na qual os códigos foram submetidos para uma representação numérica, mais adequada à aplicação de ferramental estatístico.

3.1. Competências e Suas Representações no Código Fonte

BARR e STEPHENSON (2011) identificaram e relacionaram nove conceitos do pensamento computacional de WING (2006) com exemplos de atividades segundo cinco áreas do conhecimento do K-12 (currículo da educação básica em alguns países de língua inglesa), das quais a ciência da computação faz parte. Essa relação foi escolhida como referência para um conjunto de habilidades a ser tratada neste artigo por possuir elementos facilmente identificáveis em um código fonte, relacionados às competências do pensamento computacional.

A posterior transformação desses elementos em uma representação numérica requer que eles sejam definidos de forma específica em relação à uma linguagem formal de programação, de forma a serem estabelecidos como parâmetros de identificação. Escolheu-se a linguagem "C" para referência do presente trabalho e, respeitando-se os conceitos iniciais do pensamento computacional conforme o trabalho escolhido como referência [BARR and STEPHENSON 2011], e expandindo-se os exemplos àqueles das outras áreas quando necessário ao entendimento do contexto, a seguir apresenta-se o conceito original do trabalho utilizado como referência, bem como as adaptações que julgou-se necessárias para o propósito deste trabalho:

1. Coleção de dados - originalmente como "encontrar uma fonte de dados para um problema", em um contexto mais amplo entende-se que essa habilidade corresponde à estruturas de entrada de dados em um código fonte, dessa forma os parâmetros definidos são os comandos de leitura de dispositivos de entrada, bancos de dados e arquivos.
2. Análise de dados - originalmente como "escrever um programa para solucionar cálculos estatísticos básicos", observando-se o contexto, entende-se como a solução de cálculos básicos, que podem ser representados pela ocorrência de operadores aritméticos.
3. Representação de dados - originalmente "usar estruturas de dados", com exemplos diretos no trabalho original, entende-se como estruturas utilizadas em programação para definir e armazenar dados como identificadores, *arrays*, ponteiros ou *structs*.
4. Decomposição de um problema - originalmente "definir objetos, métodos e funções", também com exemplos diretos no trabalho original, entende-se como atividades que visam decompor o problema geral em problemas menores, como funções e procedimentos.
5. Abstração - originalmente "... encapsular um conjunto de comandos que se repetem com um mesmo fim, usar condicionais, laços, recursividade, etc.", com exemplos diretos, essa habilidade contempla a ocorrência de operadores lógicos utilizados em condições, comandos de repetição e chamadas recursivas.
6. Algoritmos e procedimentos - originalmente "estudar algoritmos clássicos, implementar um algoritmo para uma área específica", não será utilizado no presente trabalho.

7. Paralelização - originalmente voltado ao processamento paralelo, não será utilizado no presente trabalho.
8. Simulação - originalmente "animação de algoritmo, busca de parâmetros", ampliando-se o contexto do conceito apresentado, entende-se que é uma habilidade que pode ser ligada à depuração e saída de dados, observando-se a ocorrência de comentários e comandos de saída.
9. Automação - originalmente sem exemplos, não será utilizado no presente trabalho.

Dessa forma serão estabelecidos parâmetros que identifiquem os tokens específicos para cada uma das habilidades conforme a Tabela 1.

Habilidade/Conceito	Tokens
Coleção de dados	Scanf, *DB calls*, *File Calls*
Análise de dados	Operadores aritméticos
Representação de dados	*Identifiers* , *Arrays*, *Pointers*, *Structs*
Decomposição do problema	Definição de funções, Chamadas de bibliotecas, Definição de Macro
Abstração	Chamadas de função, Operadores lógicos, Laços de repetição, Chamadas recursivas, Seleções
Simulação	Comentários, printf, plots

Tabela 1. Habilidades e tokens extraídos

3.2. Matriz de Representação

Objetiva-se com esse estudo a análise de conjuntos de dados, dessa forma cada código fonte processado individualmente compõe um conjunto maior. Para o presente trabalho determinou-se como o conjunto de soluções de um mesmo problema. Para cada problema, propõe-se a construção de três matrizes: uma matriz documento por token, uma matriz documento por frequência, e uma matriz balanceada.

A matriz documento por token é formada pelo conjunto de soluções onde cada linha corresponde à uma submissão, e cada coluna corresponde aos tokens extraídos em sua forma sequencial. Essa matriz corresponde aos dados brutos, processadas de forma a transformar o código original em uma representação numérica.

A matriz documento por frequência é formada pelo conjunto de soluções onde cada linha corresponde a uma submissão, e cada coluna corresponde à frequência dos tokens relacionados a cada uma das habilidades, conforme a Tabela 1. Para a construção da frequência das habilidades, conta-se o número de ocorrências para cada token e posteriormente soma-se aqueles relacionados com a mesma habilidade, ou seja, a frequência de uma habilidade é a soma de todas as ocorrências dos tokens relacionados a ela.

A matriz de frequência é então balanceada aplicando-se o método de TF-IDF (abreviação dos inglês *term frequency-inverse document frequency*), conforme as Equações 1, 2, 3, onde f representa a frequência, t termo ou token, d documento ou submissão e N o número total de submissões. A informação balanceada é então armazenada

em outra matriz, uma matriz balanceada documento por TF-IDF.

$$tf_{t,d} = \frac{f_{t,d}}{max\left(f_{t,d} : t \in d\right)} \tag{1}$$

$$idf_{t,D} = log\frac{N}{\{d \in D : t \in d\}} \tag{2}$$

$$tfidf_{t,d,D} = tf_{t,d} \times idf_{t,D} \tag{3}$$

Observou-se em experimentos preliminares que todas as habilidades estavam presentes em todos os códigos fonte válidos analisados, ainda que em diferentes níveis. Aplicando-se os agrupamentos propostos de tokens por habilidades já esperava-se que em nenhuma habilidades a frequência seria igual a zero. No caso de uma habilidade estar presente em todos os códigos fonte, como é esperado mesmo que em diferentes proporções, observa-se que a Equação 2 obteria um resultado zero, inviabilizando o método. Com o intuito de ajustar o uso do IDF considerou-se apenas a frequência de documentos nos quais a contagem de tokens é maior que a soma da média $m_{t,d}$ com o seu desvio padrão $std_{t,D}$, dessa forma a equação 2 é balanceada como Equação 4.

$$idf_{t,D} = log\frac{N}{\{d \in D : t > (m_{t,D} + std_{t,D}) \in d\}} \tag{4}$$

O uso da TF-IDF no presente trabalho objetiva emergir a importância de uma habilidade para a construção de uma solução a partir de um código fonte. Observa-se também a importância média de uma habilidade para o conjunto de soluções. Contudo, entende-se que a TF-IDF não se aplica a comparação da qualidade das habilidades presentes em diferentes soluções. A seguir busca-se atender esse propósito.

3.3. Habilidade Proporcional

O número de tokens presentes em um código fonte já foi utilizado em outros trabalhos [MOREIRA and FAVERO 2009][CAPITÁN and VOGEL-HEUSER 2017] para determinar a qualidade de um software. O número de elementos presentes em um software, entre outras características, pode ser utilizado como uma indicação de eficiência, de forma que se dois softwares executam a mesma tarefa, aquele com a menor complexidade é provavelmente o mais eficiente [CAPITÁN and VOGEL-HEUSER 2017].

Da mesma forma, ao considerarmos uma habilidade utilizada para elaborar uma solução, a soma direta de um elemento pode não representar uma proficiência elevada, isto é, ter uma ocorrência alta de elementos em um código fonte pode inferir uma baixa habilidade relacionada. Dada a característica do presente trabalho, no qual a informação é extraída a partir da frequência de tokens e utilizada para estimar habilidades, dado um determinado conjunto de soluções de um mesmo problema, assumiu-se que ou uma habilidade tem uma proporção inversa θI à menor frequência de tokens observada, indicada pela Equação 5, ou uma proporção direta θD à maior frequência de tokens observada, indicado pela Equação 6. Determinou-se a proporcionalidade da seguinte forma: coleção de dados, análise de dados, representação de dados, e abstração são inversamente proporcionais; decomposição do problema e simulação são consideradas diretamente proporcionais.

$$\theta I_{t,d} = \frac{\min f_{t,d}}{f_t} \tag{5}$$

$$\theta D_{t,d} = \frac{f_t}{\max f_{t,d}} \tag{6}$$

O uso da proporção θ no presente trabalho objetiva determinar a qualidade de uma solução escrita em um código fonte em relação às demais presentes no conjunto de dados. A qualidade determinada para cada habilidade representa uma comparação entre os códigos presentes no conjunto de dados, de forma que a qualidade das soluções presentes nesse conjunto é o que determina o máximo e mínimo calculado. Se considerarmos a soma dos θ pode-se determinar que uma solução é melhor no universo do conjunto de dados que ela está inserida, no entanto não se pode determinar que ela é a melhor solução possível para o problema.

3.4. Habilidade *Theta*

A TF-IDF objetiva emergir a importância de uma habilidade em uma submissão, o valor θ por sua vez visa estabelecer uma comparação de qualidade de uma solução em relação às outras respostas referentes a um mesmo problema. Considera-se que o produto da Proporção θ e a TF-IDF, representado por $H\theta$ na Equação 7, representa tanto a importância de uma habilidade para uma solução, como também permite a comparação da qualidade das habilidades em diferentes soluções, denominou-se esse produto como Habilidade θ.

$$H\theta_{t,d} = tfidf_{t,d,D} \times \theta_{t,d} \tag{7}$$

Considera-se como $\theta_{t,d}$ na Equação 7 ou o resultado da Equação 5 ou o resultado da Equação 6. A seguir apresenta-se um experimento no qual aplica-se a metodologia proposta em um conjunto de dados.

4. Experimento e Resultados Obtidos

Para o desenvolvimento do presente experimento escolheu-se utilizar uma base de dados composta por códigos fonte submetidos em uma competição realizada em uma plataforma de *Online Judge*. Escolheu-se a linguagem "GNU C", e dessa forma todos os códigos fonte que não correspondiam a essa linguagem foram excluídos. Ainda, apenas os problemas que totalizavam pelo menos 300 submissões foram selecionados. Dessa forma, foram processados 3144 códigos fonte distribuídos em seis problemas. Elaborou-se um parser com as habilidades citadas nas seções anteriores como parâmetro de procura. A construção desse parser contemplou um analisador léxico utilizando-se do pacote FLEX do sistema LINUX, e um analisador semântico utilizando-se o pacote BISON do sistema LINUX. Ambos os analisadores foram compilados em conjunto para processar o fluxo de entrada simultaneamente.

Os dados iniciais foram processados conforme metodologia descrita nas seções anteriores. Para o processamento inicial no seguinte experimento, algumas funções padrão do software MATLAB R2016a foram utilizadas como std(), sum(), mean(),etc, enquanto que outras necessárias à realização do experimento foram elaboradas pelo autor.

A primeira observação considerada é que a TF-IDF demostrou ser um normalizador eficiente para comparar a importância das habilidades em diferentes problemas.

Outra observação é que, até o presente estágio de desenvolvimento, TF-IDF não foi capaz de emergir habilidades discriminatórias que se qualifiquem como necessárias para a solução de um problema, visto que a média das habilidades consideradas corretas em uma resposta são consideradas similares à média das consideradas incorretas. A Figura 1 ilustra as frequências médias das habilidades dos problemas selecionados, bem como a TF-IDF média para das habilidades deles.

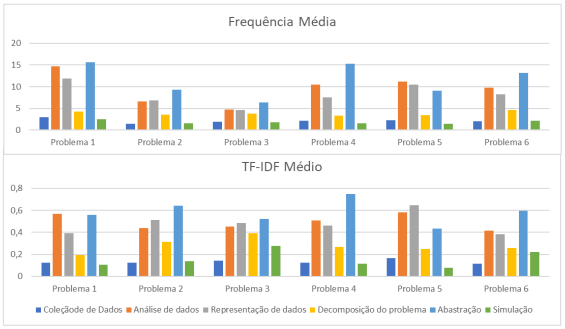

Figura 1. TF-IDF e frequência média por habilidade

No passo seguinte, para determinar o quão diferente são as habilidades utilizadas para resolver um mesmo problema, apenas as submissões com resultado correto foram observadas, dessa forma é possível afirmar que os códigos fonte comparados satisfazem com sucesso a solução do mesmo problema. Escolheu-se o problema 1 por conter a maior frequência média de tokens, e aplicou-se a metodologia para calcular $H\theta$. A Figura 2 demostra a comparação das diferentes habilidades utilizadas em seis submissões, todas referentes ao problema 1.

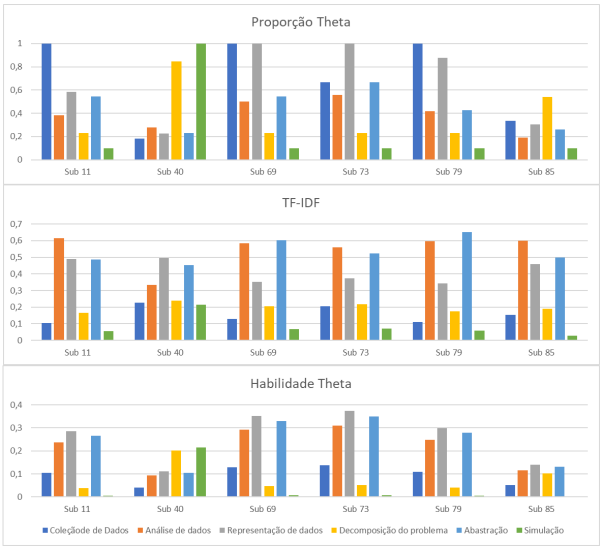

Figura 2. Comparação das habilidades em submissões

Observa-se através da Figura 2 que as submissões possuem diferenças tanto na importância das habilidades para a solução, dada pela TF-IDF, quanto na qualidade das habilidades utilizadas, dada pela proporção θ. Considerando-se a soma apenas das proporções

θ, determinou-se a solução 69 (soma θ = 3,31) como sendo a mais eficiente, e a 85 (soma θ = 1,67) como sendo a menos eficiente. Considerando-se apenas a TF-IDF não observa-se uma diferença considerável entre as habilidades utilizadas pela Sub 69 e Sub 85, contudo ao utilizarmos a Habilidade θ oberva-se uma diferença significativa entre as habilidades das duas submissões.

5. Considerações finais

A avaliação de códigos fonte em programação de computadores tem sido amplamente discutida em diversos aspectos e aplicações, entre eles o propósito educacional. No presente trabalho apresentou-se uma proposta de metodologia de avaliação de códigos fonte que utiliza TF-IDF para a identificação de habilidades do pensamento computacional em códigos fonte, na busca de elementos que auxiliem no ensino e aprendizado de programação de computadores.

A diversidade de soluções possíveis trouxe a dificuldade de determinar a significância de uma avaliação média das habilidades presentes nas soluções, buscou-se tratar essa dificuldade adaptando-se a metodologia clássica de TF-IDF para contabilizar apenas as frequências de tokens significativamente acima da média. A princípio essa abordagem demonstrou-se eficiente, contudo, a validação é executada de forma predominantemente manual e a amostra verificada com esse objetivo ainda é pequena em relação à população.

Quanto à avaliação utilizando TF-IDF individualmente para as submissões, demonstrou-se eficiente ao determinar a importância de cada habilidade para solução. No entanto, não determina um nível de habilidade utilizado na construção da resposta. Em busca de estabelecer esses níveis de habilidade desejados, considerou-se a proporcionalidade que chamamos de θ, a qual estabelece um comparativo de uma solução em relação às demais em um conjunto de respostas. Sendo um comparativo das soluções presentes, não garante que a melhor solução possa ser considerada de elevada habilidade de forma abrangente, isto é, não determina que a melhor solução encontrada é a melhor solução possível, mas sim a de melhor desempenho entre as observadas. Ao utilizarmos a Habilidade $H\theta$ obteve-se uma representação mais próxima do que julgamos serem os níveis de habilidade utilizados na elaboração das diferentes soluções.

Espera-se que as informações extraídas a partir dessa metodologia possam ser utilizadas para fins educacionais, de modo a contribuir para o desenvolvimento de metodologias educacionais voltadas ao acompanhamento do desenvolvimento das habilidades e perfil de estudantes em programação de computadores. Como trabalhos futuros almeja-se a expansão da base de dados utilizada, bem como a definição de métricas para uma avaliação mais acurada do modelo.

Agradecimentos

Universidade Federal do Rio Grande e Instituto de Educação, Ciência e Tecnologia Sul-rio-grandense. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

ALA-MUTKA, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer science education*, 15(2):83–102.

AZCONA, D., Arora, P., Hsiao, I.-H., and Smeaton, A. (2019). user2code2vec: Embeddings for profiling students based on distributional representations of source code. In *Proceedings of the 9th International Conference on Learning Analytics & Knowledge*, pages 86–95.

BARR, V. and STEPHENSON, C. (2011). Bringing computational thinking to k-12: what is involved and what is the role of the computer science education community? *Acm Inroads*, 2(1):48–54.

CABO, C. (2019). Student progress in learning computer programming: Insights from association analysis. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–8. IEEE.

CAPITÁN, L. and VOGEL-HEUSER, B. (2017). Metrics for software quality in automated production systems as an indicator for technical debt. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 709–716. IEEE.

GANGULY, D., Jones, G. J., Ramírez-De-La-Cruz, A., Ramírez-De-La-Rosa, G., and Villatoro-Tello, E. (2018). Retrieving and classifying instances of source code plagiarism. *Information Retrieval Journal*, 21(1):1–23.

KARNALIM, O. (2020). Tf-idf inspired detection for cross-language source code plagiarism and collusion. *Computer Science*, 21(1).

MOREIRA, M. P. and FAVERO, E. L. (2009). Um ambiente para ensino de programação com feedback automático de exercícios. *Workshop sobre Educação em Computação (WEI 2009)*.

RAIGOZA, J. (2017). A study of students' progress through introductory computer science programming courses. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–7. IEEE.

ROBINS, A. (2010). Learning edge momentum: A new account of outcomes in cs1. *Computer Science Education*, 20(1):37–71.

SOUZA, D., FELIZARDO, k., and BARBOSA, E. (2016). A systematic literature review of assessment tools for programming assingnments. In *2016 IEEE 29th international Conference on Software Engeneering Education and Training (CSEET)*, pages 147–156. IEEE.

SU, X., Qiu, J., Wang, T., and Zhao, L. (2016). Optimization and improvements of a moodle-based online learning system for c programming. In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–8. IEEE.

ULLAH, F., Wang, J., Farhan, M., Jabbar, S., Wu, Z., and Khalid, S. (2018a). Plagiarism detection in students' programming assignments based on semantics: multimedia e-learning based smart assessment methodology. *Multimedia Tools and Applications*, pages 1–18.

ULLAH, Z., Lajis, A., Jamjoom, M., Altalhi, A., Al-Ghamdi, A., and Saleem, F. (2018b). The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. *Computer Applications in Engineering Education*, 26(6):2328–2341.

WING, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35.