Abordagem Não-Supervisionada para Inferência do Tópico de um Exercício de Programação a partir do Código Solução

Jonas M. Moreira[1], **Carlos Eduardo Paulino Silva**[1,2], **André G. Santos**[1], **Lucas N. Ferreira**[1], **Julio C. S. Reis**[1]

[1]Departamento de Informática, Universidade Federal de Viçosa (UFV) – Brasil

[2]Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais (IFMG) – Brasil

{jonas.moreira,carlos.e.silva,andresantos,lucas.n.ferreira,jreis}@ufv.br

Abstract. *In the current educational context, where most educators use extracurricular problem sets to strengthen learning, the need to create materials that cater to the comprehension level of all students is evident. This challenge extends to programming education, where initial learning is not straightforward and the dropout and failure rates are high. Thus, this work proposes an investigation into the potential use of machine learning techniques, specifically unsupervised approaches, in conjunction with natural language processing (NLP) techniques to group different programming exercises. Specifically, the exercises are grouped into clusters based on specific topics, allowing for easier identification of exercises that meet the needs of students. Our results demonstrate the potential of this approach to make the creation of programming exercise clusters faster and more effective.*

Resumo. *No contexto educacional atual, onde a maioria dos educadores utiliza listas de exercícios extraclasses para fortalecer o aprendizado, a necessidade de criar materiais que atendam ao nível de compreensão de todos os alunos é evidente. Esse desafio se estende ao ensino de programação, onde o aprendizado inicial não é simples e a taxa de reprovação e evasão são altas. Assim, este trabalho propõe uma investigação do potencial uso de técnicas de aprendizado de máquina, especificamente abordagens não supervisionadas, em conjunto com técnicas de processamento de linguagem natural (PLN) para agrupar diferentes exercícios de programação. Particularmente, as questões são agrupadas em (clusters) com base em tópicos específicos, permitindo encontrar, com maior facilidade, exercícios que atendam às necessidades dos alunos. Os resultados demonstram o potencial desta abordagem para tornar a criação de listas de exercícios de programação mais rápida e eficaz.*

1. Introdução

No cenário educacional atual, é comum que a maioria dos professores adote uma metodologia pedagógica que inclui o uso de listas de exercícios extraclasse para complementar e/ou reforçar o conteúdo ensinado em sala de aula. Nesse sentido, a construção desse material deve ser personalizada de modo a contemplar o nível de compreensão de todos os alunos [Giraffa et al. 2015].

Todavia, criar um conteúdo que não seja muito avançado nem muito básico, pode ser uma tarefa desafiadora e árdua para um professor. Como resultado, os alunos podem

se sentirem desmotivados em relação à disciplina e, por consequência, ao curso como um todo, o que está intimamente relacionado ao aumento das taxas de evasão escolar observado nos últimos anos [Rodrigues 2002, Bosse and Gerosa 2015].

No ensino de computação, essa realidade é semelhante. Com a crescente automação em várias áreas, a demanda por profissionais com habilidades de programação está em alta, levando muitos cursos universitários a incluírem disciplinas introdutórias de programação em suas grades. No entanto, como essas disciplinas, em muitos casos, são o primeiro contato de muitos alunos com a temática, e considerando que o aprendizado da lógica de programação envolve conceitos abstratos e o domínio de uma linguagem específica, é comum uma alta taxa de reprovação e evasão [Ferreira et al. 2010].

Em resposta a esse fenômeno de desistência estudantil, surge a necessidade de desenvolver ferramentas para dar suporte aos professores na identificação das necessidades individuais de cada estudante visando solucionar possíveis problemas no atraso da compreensão do conteúdo. A existência dessas ferramentas pode se tornar um fator determinante na reversão do atual cenário de evasão, uma vez que auxiliaria os alunos a avançar no entendimento do conteúdo de forma personalizada e progressiva [Silva et al. 2023].

Considerando este contexto, esse estudo visa explorar abordagens de aprendizado de máquina, especificamente não-supervisionadas, em conjunto com técnicas de processamento de linguagem natural (PLN), para agrupar questões de programação em subgrupos (*clusters*), permitindo que um professor de computação consiga reunir questões com base em suas semelhanças (*i.e.*, tópicos e/ou assuntos em comum). Essa abordagem é motivada pela necessidade que os docentes possuem de lidar com uma grande quantidade de exercícios de programação disponíveis publicamente, muitos dos quais não são rotulados nem agrupados por tópicos ou temas, mas apenas pela ordem em que foram incluídos em uma base de dados específica. No entanto, os códigos de soluções associados a esses exercícios podem revelar similaridades significativas. Portanto, essa abordagem oferece uma maneira mais eficiente de selecionar e usar os exercícios como ferramentas de ensino, permitindo que os professores analisem apenas os exercícios semelhantes.

A validação dos resultados por meio da avaliação de um especialista, sugere que que a inferência automática dos tópicos para o agrupamento de exercícios de programação com base no código de solução é viável e efetiva, para auxliar professores na criação de listas de exercícios de programação. Isso implica em um avanço no desenvolvimento de ferramentas para personalizar o ensino e facilitar o papel do professor. Ademais, essa abordagem não apenas simplifica o processo de organização e seleção de exercícios, mas também colabora para a elaboração de estratégias de ensino mais eficientes.

O restante do trabalho está organizado como segue. Na próxima seção, elencamos trabalhos relacionados. Em seguida, na Seção 3, é apresentada a metodologia proposta para este estudo, incluindo uma descrição detalhada do conjunto de dados utilizado e uma explicação sobre as estratégias adotadas para representação do código. Os resultados experimentais são discutidos na Seção 4 e, por fim, a Seção 5 conclui o trabalho, e apresenta direções para pesquisas futuras neste contexto.

2. Trabalhos Relacionados

A extração de atributos de um texto qualquer (*e.g.*, código solução) comumente envolve a geração de *tokens* por meio da análise léxica realizada com o uso de uma linguagem de

programação [de Lima et al. 2021]. Esse processo possibilita a identificação de palavras-chave (*keywords*) essenciais para a categorização do assunto abordado por um exercício, com potencial para revelar os elementos necessários para sua resolução. Embora essa extração também possa ser feita manualmente, isso é custoso e muitas vezes culmina em uma rotulação inadequada dos problemas [Chau et al. 2017].

A aplicação de uma abordagem não-supervisionada neste contexto pode ser bastante interessante, uma vez que possibilita a descoberta de padrões que não são conhecidos previamente, sem a necessidade de rótulos para cada exemplo de treinamento [Ghahramani 2003]. Dentro desse cenário, a pesquisa conduzida por [Hosseini and Brusilovsky 2017] propôs a utilização de técnicas oriundas do processamento de linguagem natural (PLN) em códigos solução para recomendar partes úteis desses códigos aos alunos durante o desenvolvimento de um determinado exercício.

O trabalho apresentado por [de Freitas Júnior et al. 2020] também analisou similaridade entre exercícios com uso de técnicas de PLN, mas pré-processando os enunciados dos problemas. No entanto, os resultados revelam que a análise isolada dos enunciados pode levar a uma compreensão equivocada da semelhança entre os exercícios e não necessariamente refletiria em um agrupamento que facilite a compreensão do professor sobre a proximidade semântica entre os códigos.

Nesse sentido, em [dos Santos et al. 2019], os autores apresentam uma abordagem na qual métricas de inteligibilidade[1] são utilizadas para agrupar textos por meio do processamento do conteúdo textual, a fim de classificar a dificuldade das questões introdutórias de programação. Contudo, é crucial observar que essa abordagem depende da capacidade das métricas de inteligibilidade selecionadas representarem de forma precisa os indicadores de dificuldade escolhidos.

Já o estudo proposto por [Laranjeira et al. 2020] utiliza uma estratégia de filtragem colaborativa para classificar os exercícios de acordo com seus níveis de dificuldade e uma fase subsequente envolve a previsão da dificuldade de exercícios ainda não resolvidos pelo aluno. É importante destacar que este estudo reconhece algumas limitações do método, como a possível diminuição da qualidade das classificações caso haja uma baixa quantidade de exercícios submetidos pelo estudante.

No entanto, apesar dos avanços significativos nos estudos relacionados à análise de exercícios de programação e classificação de exercícios, ainda há lacunas a serem exploradas. Por exemplo, em [Azcona et al. 2019], é necessária uma grande quantidade de códigos para uma identificação adequada de similaridade entre exercícios de programação. Em [Kanade et al. 2020] existe uma dependência de modelos pré-treinados e pré-ajustes para obtenção de resultados mais condizentes. Já em [de Lima et al. 2021] os classificadores desenvolvidos segmentam as questões em módulos, considerando apenas um único tópico de programação, porém existem questões que envolvem mais de um conceito. Nesse sentido, o presente trabalho busca encontrar maneiras de mitigar esses desafios enfrentados por outros pesquisadores, por meio da investigação do potencial de técnicas de aprendizado de máquina não-supervisionado para inferência do tópico de um exercício de programação a partir do código solução.

[1]Capacidade de uma informação, mensagem, discurso, texto ou outro tipo de comunicação ser percebida e compreendida de forma eficaz [Scarton and Aluísio 2010].

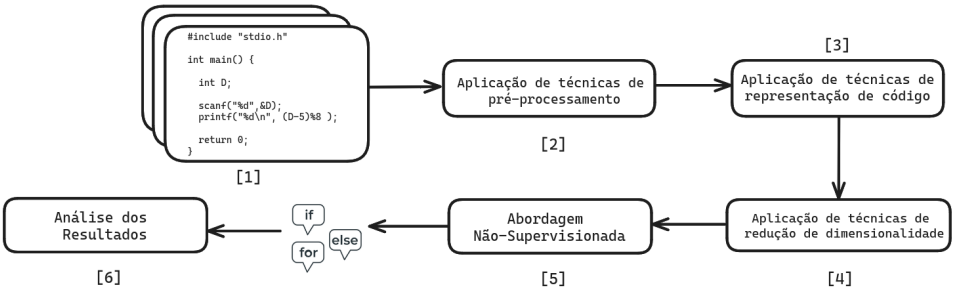


Figura 1. Visão geral da metodologia proposta.

3. Metodologia

Nesta seção, apresentamos a metodologia adotada para realização deste estudo, que tem como objetivo a proposição de uma abordagem não-supervisionada para agrupamento de exercícios de programação semelhantes a partir de tópicos de conhecimento necessários para resolução de um exercício, como estratégia para suporte a professores durante o processo de seleção de questões que irão compor uma determinada atividade. A Figura 1 apresenta uma visão geral da proposta. Inicialmente, são aplicadas técnicas de pré-processamento dos dados. Em seguida, diferentes abordagens para representação do código e técnicas de redução de dimensionalidade são exploradas. Posteriormente, os tópicos são agrupados (*i.e.,* clusterizados) por uma abordagem não-supervisionada, a fim de proporcionar uma organização eficiente e compreensível dos exercícios. A seguir, apresentamos um detalhamento de cada uma das etapas.

3.1. Base de Dados

A primeira etapa [1] consiste na obtenção de um conjunto de códigos que são fornecidos como entrada para análise/processamento posterior. Para isso, foi utilizada uma base de dados extraída das edições anteriores da Olimpíada Brasileira de Informática (OBI)[2]. A OBI é uma competição que visa despertar nos alunos o interesse por uma ciência importante na formação básica hoje em dia (no caso, Ciência da Computação), através de uma atividade que envolve desafio e competição.

Coletamos todos os exercícios disponibilizados das edições anteriores da OBI, entre os anos de 2000 e 2022, que possuíam soluções desenvolvidas com o uso das linguagens de programação C e C++. Neste contexto, é importante destacar que essas soluções representam soluções corretas, feitas pelos juízes da OBI. No total, foram obtidos 578 códigos solução (códigos-fonte), implementados por diferentes juízes. Os exercícios presentes ao longo das edições dessa olimpíada contemplam conteúdos que englobam os seguintes tópicos[3]: (1) “Fundamentos de Matemática”; (2) “Fundamentos de Computação”, e; (3) “Algoritmos e Estruturas de Dados”.

Em função da amplitude com a qual esses exercícios abordam conteúdos de diferentes naturezas, é comum que professores busquem essas questões para apresentarem aos alunos como exercícios de fixação ou aprendizagem de programação [Garcia et al. 2008]. No entanto, o grande desafio é que os exercícios não estão agrupados por tópicos e/ou nível de dificuldade, mas unicamente pelo critério temporal (*i.e.,* ano em que foram aplicados para determinada modalidade e fase da competição).

[2] https://olimpiada.ic.unicamp.br/

[3] https://olimpiada.ic.unicamp.br/prepare/ementas/ementa_prog

3.2. Técnicas de Pré-Processamento

Depois de coletados, os dados (*i.e.,* código contendo a solução de determinado exercício) são submetidos a uma etapa de pré-processamento para remoção de informações irrelevantes para o contexto do problema. Nesta etapa [2] (Figura 1), as estratégias executadas incluem: *(i)* a conversão de todo o texto (do código) para letras minúsculas e *(ii)* a remoção de valores numéricos (*e.g.,* em $PI = 3.14$, o valor 3.14 é removido), textos dentro de *strings* (*e.g.,* "Digite um valor para N:") e comentários presentes nos códigos (*e.g.,* "/* Esta função recebe dois argumentos, a e b e retorna a soma */"). Quanto a *(i)*, esta seleção foi feita considerando que a diferenciação entre maiúsculas e minúsculas não seria representativa dentro do contexto explorado neste estudo. Além disso, em relação a *(ii)* a presença de números, *strings* e comentários, poderia prejudicar as posteriores análises, dado que poderiam inserir palavras não representativas para a categorização dos tópicos.

3.3. Estratégias para Representação do Código

Na próxima etapa [3] (Figura 1), exploramos diferentes estratégias para representar o código de forma eficiente, uma vez que uma abordagem adequada é crucial para a classificação subsequente dos tópicos. Neste estudo, foram investigadas as seguintes estratégias: *(i)* BoW; *(ii) TF-IDF*; e por fim, *(iii)* BERT. Essas abordagens oferecem diferentes perspectivas na representação dos códigos solução (códigos-fonte), permitindo a extração de características relevantes que serão fundamentais para a identificação e agrupamento dos tópicos abordados nos exercícios. A seguir, é apresentada uma descrição de cada uma das técnicas exploradas.

***Bag of Words* (BoW).** A primeira estratégia investigada consiste no processo de construção de um vocabulário a partir de todas as palavras únicas encontradas nos documentos de um *corpus*. Cada documento é então representado como um vetor, onde cada posição corresponde a uma palavra do vocabulário e o valor em cada posição indica a frequência ou a presença dessa palavra no documento. O BoW é amplamente utilizado em tarefas de PLN devido à sua simplicidade e eficácia [Zhang et al. 2010].

TF-IDF. Esta é a sigla para *Term Frequency — Inverse Data Frequency*, e consiste em uma abordagem que permite a determinação do peso para cada termo (ou palavra) em cada documento, levando em consideração tanto a frequência do termo no documento quanto a sua frequência inversa em todo o *corpus*. O TF-IDF é especialmente útil para destacar termos que são frequentes em um documento específico, mas raros no *corpus* como um todo [Trstenjak et al. 2014].

BERT. Por fim, foi avaliada a utilização do BERT (*Bidirectional Encoder Representations from Transformers*), um modelo de linguagem baseado em redes neurais que revolucionou o campo de PLN. O BERT foi projetado para capturar o contexto de palavras em uma frase de maneira bidirecional, incorporando informações de palavras anteriores e posteriores na sequência. Isso o torna extremamente eficaz em uma variedade de tarefas, incluindo a compreensão de texto e a geração de resumos [Devlin et al. 2018].

Para o problema explorado neste estudo, percebe-se que o agrupamento de códigos solução (códigos-fonte) permite a utilização de tais técnicas, uma vez que a solução de um exercício consiste na compreensão e expressão de conceitos específicos de linguagens de programação, assim como um texto descrito em uma linguagem natural. Dessa forma, ao buscar a adoção das técnicas citadas anteriormente, espera-se que seja possível

ocorrer automaticamente a identificação de padrões nos códigos solução (códigos-fonte), realizando agrupamentos significativos relacionados a tópicos semelhantes entre si.

3.4. Técnicas de Redução de Dimensionalidade

A partir da representação definida, comumente esparsa, foi utilizado um algoritmo para diminuir a dimensionalidade do resultado obtido. Nesta etapa [4] (Figura 1) aplicamos o t-SNE, ou *t-distributed Stochastic Neighbor Embedding*. Em resumo, esta abordagem busca representar dados de alta dimensão em um espaço de menor dimensão (geralmente 2D ou 3D) de forma que a estrutura de similaridade entre os dados, e no contexto deste trabalho códigos contendo soluções de exercícios de programação, seja preservada tanto quanto possível [Van der Maaten and Hinton 2008].

3.5. Abordagem Não-Supervisionada

Depois, na etapa [5] (Figura 1) selecionamos um classificador não-supervisionado para agrupamento dos códigos contendo a solução dos exercícios. Especificamente, selecionamos o *K-means* [Arthur and Vassilvitskii 2007], uma abordagem amplamente explorada e consolidada na literatura em diversos contextos, como por exemplo o reconhecimento de padrões [Jain 2010]. Uma característica fundamental deste algoritmo é agrupar pontos de dados em k grupos distintos, onde cada grupo é representado por um centroide, de forma que a soma das distâncias quadráticas entre os pontos de dados e o centroide de seus grupos (ou *clusters*) correspondentes seja minimizada [Likas et al. 2003].

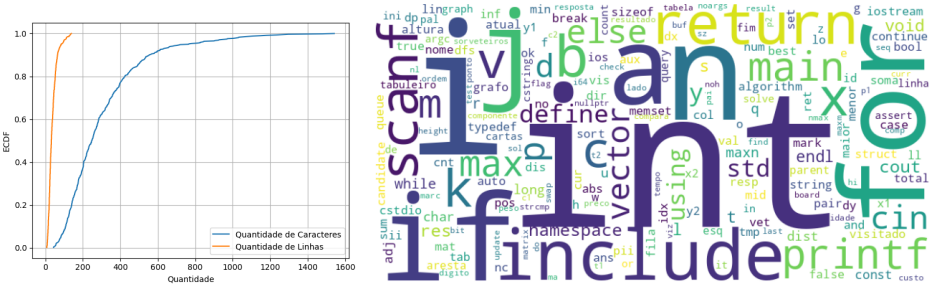
3.6. Análise dos Resultados

Por fim, a partir do agrupamento dos dados realizado anteriormente, a última etapa [6] da metodologia consiste na análise e validação dos resultados (*i.e.*, *clusters*) por um especialista em programação. O objetivo dessa etapa foi avaliar, qualitativamente, o desempenho da abordagem na tarefa de agrupar exercícios de programação que envolvem tópicos semelhantes. Para isso, foram fornecidos ao especialista o centroide de cada agrupamento (*cluster*), que consistem em exemplos representativos dos mesmos, *i.e.*, os pontos médios de cada grupo, e um conjunto de questões de cada *cluster* que possuem a maior distância em relação ao centroide. Essas questões, por estarem mais distantes dos centroides, são consideradas como representativas da borda dos agrupamentos (*cluster*). É interessante analisar questões de borda porque provavelmente elas possuirão características semelhantes a outras de seu próprio *cluster* e também de outro próximo, bem como características um pouco diferentes de seu centroide. Em suma, o especialista foi responsável por analisar essas questões e realizar um casamento entre os agrupamentos feitos pelo modelo e os tópicos reais dos exercícios[4].

4. Resultados

Nesta seção, apresentamos os principais resultados obtidos neste estudo, incluindo uma caracterização dos dados explorados (Seção 4.1), cujo objetivo é uma melhor compreensão do cenário em questão, e análises de desempenho da abordagem não-supervisionada proposta para inferência de tópicos de exercícios de programação (Seção 4.2).

[4]Projeto submetido e aprovado pelo Comitê de Ética - CAAE 75327023.0.0000.5153.



(a) (b)

Figura 2. (a) Função de Distribuição Acumulada (ECDF) da quantidade de caracteres e linhas de código da base de dados; (b) Nuvem de termos mais frequentes nos códigos solução (códigos-fonte).

4.1. Caracterização dos Dados

A Figura 2(a) apresenta uma função de distribuição acumulada (ECDF) da quantidade de caracteres e linhas de código nos dados explorados. De forma geral, percebemos que $\approx$ 90% dos códigos solução da base de dados explorada possuem até 600 caracteres. Por outro lado, todos eles estão limitados a cerca de 170 linhas de código em sua composição.

Com o objetivo de entender características relacionadas aos termos presentes nos códigos solução analisados, conduzimos uma análise de frequência dos mesmos. Uma vez que as soluções exploradas são implementadas em C e C++, exclusivamente, percebe-se, por meio da nuvem de palavras apresentada na Figura 2(b), um resultado consistente com o esperado. Particularmente, termos como `int`, `i`, `j`, `if`, `include`, `return` e `for` são bastante presentes nos códigos analisados.

Depois, com o intuito de compreender ainda mais os códigos solução em termos dos tópicos que eles abarcam, utilizamos o *Latent Dirichlet Allocation* (LDA), um modelo probabilístico generativo para inferir automaticamente os tópicos em uma coleção de documentos (*i.e.*, códigos, no contexto deste estudo). Em outras palavras, a ideia básica é que os códigos contendo a solução são representados como misturas aleatórias de tópicos latentes, onde cada tópico é caracterizado por uma distribuição de palavras [Blei et al. 2003]. Seu principal objetivo é descobrir automaticamente quais são esses tópicos e como eles estão distribuídos nos documentos. Uma vantagem de utilizar o LDA é que ele é um dos métodos mais estabelecidos e amplamente aplicados na análise de tópicos, pois fornece uma interpretação intuitiva dos temas presentes nos documentos, tornando mais fácil a compreensão da coleção de texto. Embora ele possua limitações, como o fato de o número de tópicos ser um número inteiro fixo e poder exigir um pré-processamento significativo dos dados, o método se mostra como um ponto de partida interessante para análises.

Executamos a versão do LDA disponível na biblioteca *gensim*[5] [Řehřek and Sojka 2010], uma biblioteca Python para modelagem de tópicos. O melhor número de tópicos k foi definido a partir da métrica de coerência do tópico [Newman et al. 2010], que captura se tópicos diferentes, na verdade, têm poucos termos em comum. Especificamente, executamos o algoritmo LDA variando o número de tópicos k de 4 a 12 e escolhemos o modelo LDA que produziu a pontuação mais alta de

[5] `https://pypi.org/project/gensim/`

Tabela 1. Tópicos inferidos pelo algoritmo LDA.

Tópicos	Top-10 Termos Representativos	# Exercícios Associados
1	flag, lin, col, result, nec, felix, dif, not, max_num, distancia	485
2	mid, candidate, auto, aux, last, hi, int32_t, pw, falta, lado	491
3	arc_sum, raiz_cubica, dist, nivel, dx, static, direcao, arc_dist, tot, i64	489
4	soma, string, cnt, cur, sum, val, maxn, pb, push_back, menor	501
5	int, if, for, return, else, namespace, using, cin, cout, long	578

coerência do tópico, que foi para $k = 5$. Esses tópicos são apresentados na Tabela 1, que apresenta os 10 termos mais representativos (de acordo com LDA) para cada tópico, bem como o número de questões associadas a cada um deles. Neste contexto, é importante ressaltar que existem questões associadas a mais de um tópico identificado pelo LDA.

Por meio da Tabela 1, é possível observar que os 4 (quatro) primeiros tópicos possuem principalmente nomes de declaração de variáveis, enquanto o quinto tópico agrupa um conjunto de palavras comuns em praticamente todos os programas escritos em C/C++. Nesse sentido, foi realizada uma análise auxiliar para identificar a interseção de questões entre os tópicos. Em função disso, constatou-se que o tópico 5 engloba todas as questões da base de dados, enquanto os tópicos de 1 a 4 apresentam variações nos conjuntos de palavras representativas. Nesse sentido, o tópico 2 possui 22 questões que não estão no tópico 1, o tópico 3 possui 20 questões não presentes no tópico 1, o tópico 3 possui 14 questões não presentes no tópico 2, o tópico 4 possui 32 questões não presentes no tópico 1, o tópico 4 possui 29 questões não presentes no tópico 2, e o tópico 4 possui 30 questões não presentes no tópico 3. Quanto ao tópico 5, foram identificados 93 questões exclusivas que não estão presentes no tópico 1, 87 questões exclusivas que não estão presentes no tópico 2, 89 questões exclusivas que não estão presentes no tópico 3, e 77 questões exclusivas que não estão presentes no tópico 4. Essa análise detalhada revela padrões distintos entre os tópicos e corrobora para a premissa de que é possível realizar uma *clusterização* das questões por meio de técnicas de PLN, uma vez que o LDA permitiu uma visualização inicial sobre a distribuição e a inter-relação das questões em cada um dos tópicos.

4.2. Abordagem Não-Supervisionada

Para entendermos como e se os códigos solução do conjunto de dados explorados cobrem diferentes tópicos associados as questões, os agrupamos, a partir das representações vetoriais descritas na Seção 3.3. Para agrupar esses códigos-fontes, usamos o K-Means, conforme detalhado na Seção 3.5., que consiste em um algoritmo baseado em distâncias euclidianas [Arthur and Vassilvitskii 2007]. Neste trabalho, usamos $K = 8$, que foi o valor ideal encontrado a partir de uma análise do *Silhouette Score* [Rousseeuw 1987], que captura, em média, quão agrupados todos os membros em diferentes grupos (*i.e.*, *clusters*) são, e seleciona o valor de K, para o qual a Pontuação da Silhueta é o mais alto. Por questões de brevidade, e considerando que o processo de aplicação de técnicas das diferentes estratégias para representação de códigos-fonte (Seção 3.3) apresentaram resultados similares, exibimos na Figura 3 apenas os obtidos utilizando o BERT, que se destacou em relação aos outros métodos.

Podemos observar que a distribuição das questões nos *clusters* apresentou uma distinção considerável entre os diversos tópicos abordados nas questões de programação (Figura 3). A capacidade do BERT em compreender o contexto das palavras em uma frase, de maneira bidirecional, resultou em uma representação dos dados bastante pre-

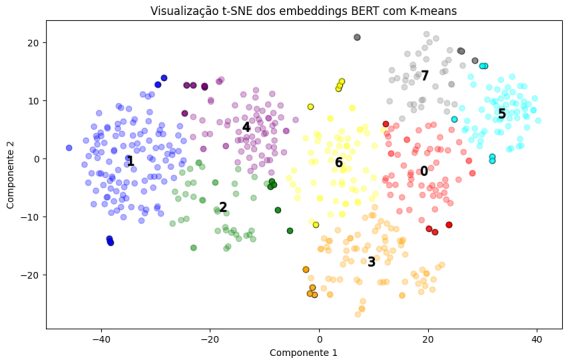


Figura 3. Representação vetorial dos códigos-fonte utilizando BERT e t-sne como técnica de redução de dimensionalidade dos dados.

cisa e informativa. No entanto, ressalta-se que é necessária uma validação manual dos resultados obtidos por um especialista, a fim de comprovar seu desempenho satisfatório.

Análise dos Resultados/Validação Manual. Para avaliar os resultados da abordagem não-supervisionada proposta, foram extraídos dados representativos (*i.e.*, centroides) de cada agrupamento identificado. Além disso, também selecionamos as 6 questões que apresentaram as maiores distâncias (diferenças) em relação ao centroide de cada *cluster*[6]. No contexto deste trabalho estas questões são referenciadas como "questões de borda". O objetivo é que o especialista comparasse manualmente as questões para validar se de fato os agrupamentos realizados estavam condizentes.

Na Figura 3, essas questões são destacadas em cores mais intensas e contorno em negrito. É importante destacar que a distância entre essas questões e os centroides é calculada como a diferença normalizada entre o *embedding* capturado pelo t-SNE e o centroide correspondente. Esse método ajuda a identificar questões que se destacam significativamente dentro do *cluster*, sugerindo que possam estar próximas a outros *clusters*.

O especialista recebeu inicialmente apenas os centroides, desconhecendo a quais *clusters* estavam associados. Dessa forma, ele identificou manualmente quais tópicos estavam relacionados a cada uma das questões. O resultado da avaliação é apresentado na Tabela 2. Como conclusão, o especialista analisou que de fato as questões de borda possuíam características muito semelhantes em relação ao centroide apresentado, de modo que, para quase todas as questões, os temas abordados no centroide se repetiam para as bordas. Além disso, notou que havia uma diferenciação considerável entre a estrutura e assunto de um *cluster* para outro.

5. Conclusão e Trabalhos Futuros

Neste estudo foi apresentada uma investigação acerca do potencial de abordagens não-supervisionadas para inferência dos tópicos abordados em um exercício de programação a partir do código solução. Em suma, os resultados são promissores, o que reforça o potencial dessas estratégias como mecanismo de suporte ao professor. Por meio da aplicação de técnicas como BoW, TF-IDF e BERT, foi possível identificar padrões significativos e criar representações eficazes dos tópicos abordados nos exercícios.

[6]A definição deste número está associada à disponibilidade do especialista para avaliação das questões. No entanto, futuramente, pretendemos explorar outros cenários e/ou conjuntos baseados em diferentes critérios.

Tabela 2. Avaliação de centroides e bordas conduzida pelo especialista.

Número	Centroide	Bordas
0	Repetições simples; Vetores avançados e leitura/saída de dados	Vetores básicos e avançados; Funções básicas e avançadas; Recursividade; Condicionais simples e compostas; Operadores lógicos e relacionais; Matemática básica; Leitura/saída de dados; Matrizes; e repetições simples e aninhada
1	Leitura/saída de dados; Condicionais aninhados; Operadores relacionais e operações básicas de matemática	Leitura/saída de dados; Vetores; Matrizes; Repetições simples e aninhadas; Condicionais simples e compostas; Funções e matemática básica
2	Estruturas de dados; Vetores; Ponteiros; Alocação de memória; Operações avançadas com strings; Funções; Repetição e condicionais	Funções; Matemática básica; Estruturas de dados; Vetores; Matrizes; Recursividade; Condicionais e manipulação de strings
3	Funções avançadas; Estruturas de dados heterogêneas; Repetições e condicionais básicas	Funções; Matemática básica; Estruturas de dados heterogêneas; Vetores; Condicionais; Operadores lógicos e relacionais; Recursividade e matrizes
4	Leitura/saída de dados; Vetores; Operadores lógicos e relacionais; Condicionais simples e compostas e matemática básica	Repetições; Operações com strings; Funções avançadas e condicionais aninhadas
5	Leitura/saída de dados; Manipulação de vetores; Operações relacionais; Condicionais simples e matemática básica	Manipulação de strings; Funções avançadas; Operadores lógicos; Repetições aninhadas e estruturas de dados heterogêneas
6	Vetores; Funções; Operações simples de repetição e condicionais; Operações avançados de matemática e recursividade	Matrizes; Recursividade; Operações com strings e estruturas de dados heterogêneas
7	Leitura/saída de dados e matemática básica	Operações com strings; Repetições simples; Condicionais aninhadas e compostas; Funções avançadas e operadores relacionais

A distribuição equitativa e a clara distinção entre os grupos observados, especialmente com o uso do modelo BERT, sugerem que esta abordagem não-supervisionada pode ser capaz de capturar com precisão a essência dos exercícios de programação e fornecer informações valiosas para professores. É importante ressaltar que essa abordagem é especialmente interessante, pois, ao analisar as questões de forma geral, pode ser trabalhoso agrupá-las devido à similaridade de conhecimentos abordados, mas esse trabalho é facilitado caso elas estejam agrupadas de acordo com os tópicos que contemplam.

As descobertas deste estudo incentivam a continuidade e o aprimoramento da aplicação de técnicas não-supervisionadas na análise e compreensão de exercícios relacionados à programação. Nesse sentido, torna-se possível uma tentativa de avaliar também a dificuldade de tais exercícios, uma vez que o agrupamento feito pelo código não reflete necessariamente a dificuldade do exercício. Há exercícios que demandam um raciocínio complexo e um tempo considerável para se chegar a uma ideia cuja implementação em código é particularmente simples. Portanto, tendo grupos de exercícios bem definidos, é possível criar um classificador que pontue os tópicos presentes na solução mediante uma métrica de dificuldade. Além disso, identificar a dificuldade dos exercícios pode contribuir significativamente para o desenvolvimento de mecanismos mais eficientes para que o professor consiga selecionar exercícios de acordo com sua necessidade. Uma vez que, atualmente, não há uma descrição para o conteúdo dos grupos (*clusters*) e eles e precisam ser avaliados por um especialista. Com isso, os professores poderiam ter uma ferramenta eficaz para desenvolver um ensino personalizado, possivelmente reduzindo as taxas de reprovação e evasão.

Agradecimentos. CAPES, FAPEMIG, UFV, Sicoob-UFVCredi e IFMG.

Referências

Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proc. of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1027–1035.

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 International Learning Analytics Knowledge Conference (LAK)*. ACM.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Bosse, Y. and Gerosa, M. A. (2015). Reprovações e trancamentos nas disciplinas de introdução à programação da universidade de são paulo: um estudo preliminar. In *Anais do Workshop sobre Educaçao em Computaçao (WEI)*, pages 426–435.

Chau, H., Barria-Pineda, J., and Brusilovsky, P. (2017). Content wizard: concept-based recommender system for instructors of programming courses. In *Adjunct Publication of the Conference on User Modeling, Adaptation and Personalization*.

de Freitas Júnior, H. B., Pereira, F. D., de Oliveira, E. H. T., de Oliveira, D. B. F., and de Carvalho, L. S. G. (2020). Recomendação automática de problemas em juízes online usando processamento de linguagem natural e análise dirigida aos dados. In *Anais do Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 1152–1161.

de Lima, M. A. P., de Carvalho, L. S. G., de Oliveira, E. H. T., de Oliveira, D. B. F., and Pereira, F. D. (2021). Uso de atributos de código para classificação da facilidade de questões de codificação. In *Anais do Simpósio Brasileiro de Educação em Computação (SBIE)*, pages 113–122.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

dos Santos, P., de Carvalho, L. S. G., Oliveira, E., and Fernandes, D. (2019). Classificação de dificuldade de questões de programação com base na inteligibilidade do enunciado. In *Anais do Simpósio Brasileiro de Educação em Computação (SBIE)*, pages 1323–1332.

Ferreira, C., Gonzaga, F., and Santos, S. (2010). Um estudo sobre a aprendizagem de lógica de programaçao por demonstraçao. In *Anais do Workshop sobre Educação em Computação (WEI)*.

Garcia, R. E., Correia, R. C. M., and Shimabukuro, M. H. (2008). Ensino de lógica de programação e estruturas de dados para alunos do ensino médio. In *Workshop sobre o Ensino de Computação (WEI)*, pages 246–249.

Ghahramani, Z. (2003). Unsupervised learning. In *Summer school on machine learning*, pages 72–112. Springer.

Giraffa, L., Muller, L., and Moraes, M. C. (2015). Ensinado programação apoiada por um ambiente virtual e exercícios associados a cotidiano dos alunos: compartilhando

alternativas e lições aprendidas. In *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*.

Hosseini, R. and Brusilovsky, P. (2017). A study of concept-based similarity approaches for recommending program examples. *New Review of Hypermedia and Multimedia*, 23(3):161–188.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666.

Kanade, A., Maniatis, P., Balakrishnan, G., and Shi, K. (2020). Learning and evaluating contextual embedding of source code. In *International Conference on Machine Learning*, pages 5110–5121.

Laranjeira, D. R. et al. (2020). Recomendação de exercícios para alunos de programação em um ambiente de correção automática de códigos. *Dissertação. Universidade Federal do Amazonas. Disponível em:* `https://tede.ufam.edu.br/handle/tede/7775`.

Likas, A., Vlassis, N., and Verbeek, J. J. (2003). The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461.

Newman, D., Lau, J. H., Grieser, K., and Baldwin, T. (2010). Automatic evaluation of topic coherence. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (ACL)*, pages 100–108.

Řehřek, R. and Sojka, P. (2010). Software framework for topic modelling with large corpora.

Rodrigues, M. (2002). Como ensinar programação. *Informática–Boletim Informativo Ano I*, (01).

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

Scarton, C. E. and Aluísio, S. M. (2010). Análise da inteligibilidade de textos via ferramentas de processamento de língua natural: adaptando as métricas do coh-metrix para o português. *Linguamática*, 2(1):45–61.

Silva, C. E. P., Solano, J. L. S., dos Santos, A. G., and Reis, J. C. S. (2023). Previsão de reprovações em disciplinas introdutórias de programação: Um estudo em um ambiente de correção automática de códigos. In *Anais do Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 1524–1535.

Trstenjak, B., Mikac, S., and Donko, D. (2014). Knn with tf-idf based framework for text categorization. *Procedia Engineering*, 69:1356–1364.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Zhang, Y., Jin, R., and Zhou, Z.-H. (2010). Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1:43–52.