UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

MetaPath2Vec++: avaliação experimental de hiperparâmetros na geração de embeddings em grafos

Aureo Aparecido Beligolli Saldanha

Trabalho de Conclusão de Curso
MBA em Inteligência Artificial e Big Data

ICMC USP SÃO CARLOS

UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

MetaPath2Vec++

Aureo Aparecido Beligolli Saldanha

USP - São Carlos
2023

Aureo Aparecido Beligolli Saldanha

MetaPath2Vec++: avaliação experimental de hiperparâmetros na geração de embeddings em grafos

Trabalho de conclusão de curso apresentado ao Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, como parte dos requisitos para obtenção do título de Especialista em Inteligência Artificial e Big Data.

Área de concentração: Inteligência Artificial

Orientador: Prof. Dr. Ricardo M. Marcacini

USP - São Carlos

2023

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

B431m

Beligolli Saldanha, Aureo Aparecido
MetaPath2Vec++: avaliação experimental de hiperparâmetros na geração de embeddings em grafos / Aureo Aparecido Beligolli Saldanha; orientador Ricardo Marcondes Marcacini . -- São Carlos, 2023.
83 p.

Trabalho de conclusão de curso (MBA em Inteligência Artificial e Big Data) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2023.

1. Rede Heterogêneas. 2. Embeddings. 3. Grafos. 4. Aprendizado de Máquina. 5. metapath2vec++. I. , Ricardo Marcondes Marcacini, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

DEDICATÓRIA

Dedico este trabalho aos meus pais, os quais sempre me suportaram nos estudos. Não obstante, a minha filha, irmão, Deus e a NOEM Medical.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Ricardo Marcondes Marcacini, elemento chave que expandiu meu horizonte de conhecimento e oportunidades, apresentando o tema deste trabalho.

A Profa. Dra. Solange Oliveira Rezende, pelo apoio que transcendeu sua orientação à área acadêmica.

RESUMO

SALDANHA, A. A. B. **MetaPath2Vec++**: avaliação experimental de hiperparâmetros na geração de *embeddings* em grafos. 2023. 83 p. Trabalho de conclusão de curso (MBA em Inteligência Artificial e Big Data) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2023.

Este trabalho aborda o estudo das redes heterogêneas e a importância da geração de *embeddings* para a representação eficaz de grafos. Utilizando o modelo MetaPath2Vec++, o estudo explora o impacto de dois parâmetros: o comprimento da caminhada e o tamanho do contexto. O foco recai sobre a ausência de investigações empíricas detalhadas desses parâmetros no modelo MetaPath2Vec++. O comprimento da caminhada, que dita a extensão de passeios aleatórios nos grafos, e o tamanho do contexto, que define o número de nós vizinhos considerados durante o treinamento, são analisados quanto à sua influência na acurácia dos testes do modelo. A pesquisa emprega seis *datasets* heterogêneos da biblioteca PyTorch-Geometric, variando o comprimento da caminhada de 3 a 32 e o tamanho do contexto de 3 a 33, para determinar as configurações mais eficazes. A implementação do modelo MetaPath2Vec++ foi acompanhada de análises gráfica e estatísticas, percorrendo a construção de gráficos de contorno para visualização de tendências, a aplicação de regressão linear e polinomial para quantificar a relação entre os parâmetros do modelo e a acurácia dos testes, além da utilização da ANOVA para validar a significância estatística dessas relações. As análises revelam variações expressivas na correlação entre os parâmetros e a acurácia em diferentes conjuntos de dados, evidenciando que certas combinações de comprimento da caminhada e tamanho do contexto otimizam a acurácia do teste, mas com variações na localização e amplitude dessas zonas ótimas entre os *datasets*. Os resultados sublinham a importância de ajustes específicos do modelo MetaPath2Vec++ para cada conjunto de dados a fim de alcançar resultados mais robustos. Destaca-se a necessidade de aprimoramentos nas técnicas de análise para uma compreensão mais abrangente e precisa da acurácia de teste em em função de sua peculiaridade. A adoção de métodos estatísticos mais elaborados é recomendável igualmente como incrementar o número de épocas e ampliar a faixa de variação dos hiperparâmetros. Reiterando, é recomendável uma abordagem customizada para cada *dataset*, considerando suas características únicas, incluindo a exploração de diferentes configurações de meta-caminhos.

Palavras-chave: redes heterogêneas; *embeddings*; grafos; aprendizado de máquina; metapath2vec++.

ABSTRACT

SALDANHA, A. A. B. **MetaPath2Vec++**: experimental evaluation of hyperparameters in graph embedding generation. 2023. 83 p. Trabalho de conclusão de curso (MBA em Inteligência Artificial e Big Data) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2023.

This study explores heterogeneous networks and the importance of creating embeddings for effective graph representation. It uses the MetaPath2Vec++ model to examine the effects of two parameters: walk length and context size. The research addresses the gap in detailed empirical studies of these parameters in this model. Walk length, which determines the scope of random walks in graphs, and context size, which sets the number of neighboring nodes included during training, are examined for their impact on the model's test accuracy. The study employs six datasets from the PyTorch-Geometric library, adjusting the walk length from 3 to 32 and the context size from 3 to 33, to identify their behavior in test accuracy. The MetaPath2Vec++ model's implementation involved both graphical and statistical analyses. This included creating contour graphs to visualize trends, applying linear and polynomial regression to measure the relationship between model parameters and test accuracy, and using ANOVA to confirm the statistical significance of these relationships. The findings show notable differences in how parameters and accuracy correlate across various datasets. They indicate that specific combinations of walk length and context size can optimize test accuracy, though the optimal zones' locations and extents vary among datasets. The results highlight the need for tailored adjustments to the MetaPath2Vec++ model for each dataset to achieve stronger results. The study also points out the necessity for enhanced analysis techniques for a deeper and more precise understanding of test accuracy and its unique aspects. It suggests adopting more advanced statistical methods, increasing the number of epochs, and broadening the range of hyperparameter variations. The study concludes by recommending a dataset-specific approach, taking into account their distinct features, including the exploration of different meta-path configurations.

Keywords: heterogeneous networks; embeddings; graphs; machine learning; metapath2vec++.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Redes heterogêneas são estruturas que incorporam uma variedade de tipos de nós e arestas, representando dados complexos em múltiplas áreas, incluindo biologia, redes sociais, economia e engenharias. A utilização de *embeddings* para representar grafos nessas redes consiste em um método eficaz para condensar complexas informações estruturais e semânticas em formatos vetoriais de dimensões reduzidas, o que facilita o processamento e a análise desses dados por algoritmos de aprendizado de máquina [1].

Várias técnicas são aplicadas na geração de *embeddings* de grafos, visando capturar estas relações em dados estruturados. Modelos notáveis incluem o MetaPath2Vec e o MetaPath2Vec++. O primeiro utiliza meta-caminhos para orientar passeios aleatórios para mapear as relações semânticas em redes heterogêneas. O MetaPath2Vec++, em sua expansão, incorpora as características estruturais e semânticas dos grafos de maneira mais integrada.

Embora a literatura sobre o MetaPath2Vec++ reconheça a importância de diversos parâmetros no aprendizado de *embeddings*, persiste uma lacuna em estudos empíricos que os explorem em profundidade. Este trabalho acadêmico se concentra especificamente no comprimento da caminhada e no tamanho do contexto, dois parâmetros imprescindíveis que exercem influência significativa na qualidade dos *embeddings* aprendidos. O comprimento da caminhada, que determina quantos passos serão dados em cada passeio aleatório pelo grafo, é essencial na captura da informação estrutural. Um comprimento maior permite explorar relações mais distantes entre os nós, potencialmente incorporando uma contextualização mais abrangente, contudo, faz-se necessário cautela, pois pode introduzir ruído nas representações aprendidas. Por outro lado, o tamanho do contexto refere-se ao número de nós adjacentes considerados em torno de um nó central durante o treinamento. Um contexto maior pode enriquecer a representação de um nó ao considerar uma gama mais ampla de suas conexões, mas também pode diluir o foco nas relações mais imediatas e relevantes.

Este trabalho dedica-se a um estudo experimental para avaliar como o comprimento da caminhada e o tamanho do contexto influenciam a acurácia de teste no modelo MetaPath2Vec++. Esta pesquisa visa contribuir para a literatura existente, fornecendo uma análise sobre o impacto de destes parâmetros na qualidade dos *embeddings* gerados pelo MetaPath2Vec++ e, consequentemente, na performance do modelo em contextos de aprendizado de máquina.

Neste estudo, foram utilizados seis *datasets* heterogêneos da biblioteca PyTorch-Geometric para analisar a influência de ambos hiperparâmetros do MetaPath2Vec++. Experimentos foram conduzidos, variando o comprimento da caminhada entre 3 a 32 e o tamanho do contexto de 3 a 33, com a finalidade de identificar configurações ótimas que aprimorem a performance do modelo em diversos contextos, proporcionando um entendimento sistemático sobre o impacto desses parâmetros nas *embeddings* em redes heterogêneas.

A monografia é estruturada para abordar os principais aspectos do estudo sobre o MetaPath2Vec++. No Capítulo 2, a fundamentação teórica é apresentada a estrutura do MetaPath2Vec, com ênfase nas diferenças entre os *embeddings* em redes homogêneas e heterogêneas, convergindo para sua expansão, o modelo MetaPath2Vec++. O Capítulo 3, dedicado à metodologia e desenvolvimento, explica a implementação do modelo MetaPath2Vec++, abordando as condições de contorno e premissas, além dos *datasets* e configurações utilizadas. Os resultados e a discussão são explorados no Capítulo 4, contemplando análises de gráficos de contorno, correlação da acurácia de teste, regressão linear múltipla, regressão polinomial e ANOVA. Conclui-se, com o Capítulo 5 a monografia, resumindo as descobertas e propondo caminhos para futuros trabalhos na área de aprendizado de máquina aplicado a grafos.

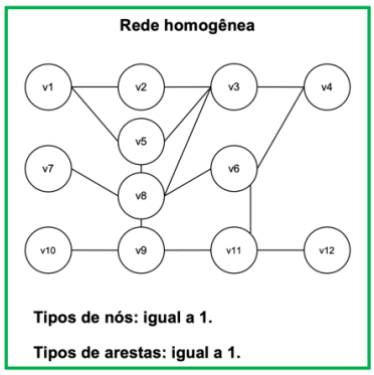
2. FUNDAMENTAÇÃO TEÓRICA

Esta seção explora a teoria subjacente ao framework MetaPath2Vec++, percorrendo os conceitos básicos de redes homogêneas e a técnica utilizada para formação de *embeddings*. Conseguinte, introduz-se as redes heterogêneas, sua distinção da anterior, sucedido da problemática envolvida em adaptar as técnicas de redes homogêneas para heterogêneas. Por fim, apresenta-se o desenvolvimento matemático básico do modelo MetaPath2Vec e sua expansão, MetaPath2Vec++.

2.1 Rede homogênea e embedding

A Figura 1 ilustra uma rede homogênea, também conhecida como *network*, caracterizada por ser um conjunto de entidades interligadas, denominadas nós, e pelas relações entre estas, conhecidas como arestas. Neste contexto, tanto os nós quanto as arestas simbolizam um único tipo de entidade ou relação. Os nós, elementos centrais de qualquer rede, podem exemplificar uma variedade de entidades, como indivíduos em uma rede social, neurônios em uma rede neural, ou computadores em uma rede de informática. No entanto, em uma rede homogênea, esses nós são uniformes, compartilhando o mesmo tipo ou possuindo atributos similares. As arestas, por sua vez, representam as conexões entre os nós, indicando algum tipo de relação ou interação entre eles. Neste tipo de rede, as arestas simbolizam um único tipo de relação. A homogeneidade, neste contexto, alude à uniformidade e consistência tanto dos elementos quanto das relações presentes na rede, sendo uma premissa que os nós e as arestas mantenham uniformidade em suas características e funções [1].

Figura 1 – Exemplo de rede homogênea.



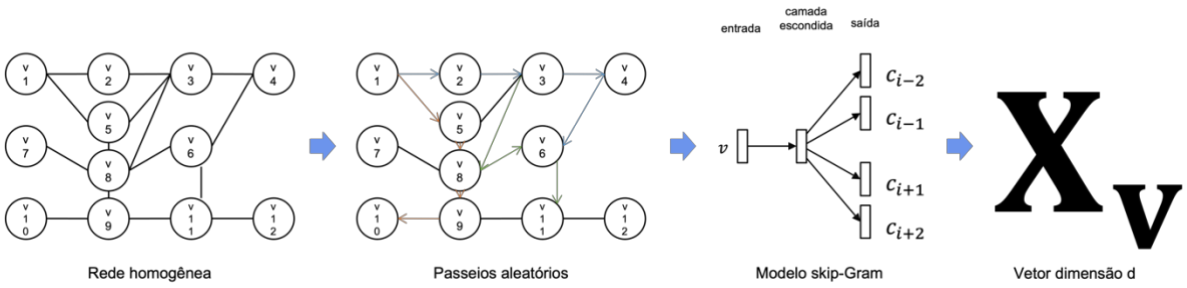
Fonte: autoria própria.

O trabalho de Mikolov et al., o qual propôs o Word2Vec **Error! Reference source not found.**[5], modelo aplicado ao aprendizado distribuído de palavras em um corpus textual, inspirou o desenvolvimento de modelos como DeepWalk **Error! Reference source not found.** e Node2Vec **Error! Reference source not found.**. Estes algoritmos utilizaram passeios a leatórios para gerar sequências de nós, semelhantes às sequências de palavras em textos, e adotam o modelo *skip-gram* para aprender a representação de um nó. Essa abordagem propiciou a previsão do seu contexto estrutural, ou seja, sua vizinhança, em uma rede homogênea.

Formaliza-se uma rede homogênea $G = (V, E)$, onde V são os nós e E, as arestas. É obtido como saída, uma matriz $\mathbf{X} \in \mathbb{R}^{|V| \times d}$ onde $d \ll |V|$, sendo $\mathbf{X_v}$ um vetor de dimensão d para cada nó v [1].

Em linhas gerais, a geração de *embeddings*, esquematizado de maneira simplificada na **Error! Reference source not found.**, inicia-se pelo passeio aleatório na rede. Em seguida, por meio do modelo *skip-gram*, dado, um nó contido em V, é gerado um vetor $\mathbf{X_v}$ de dimensionalidade d. Este processo é repetido até todos os nós presentes na rede possuírem sua própria *embedding*.

Figura 2 – Etapas de formação de *embeddings* em redes homogêneas.



Fonte: autoria própria.

Vale ressaltar a distinção entre o modelo DeepWalk **Error! Reference source not found.** e Node2Vec **Error! Reference source not found.** na elaboração de *embeddings* para redes, cada qual com suas características intrínsecas. O DeepWalk adota uma estratégia de caminhadas aleatórias uniformes para a formação de sequências nodais. Iniciando de um ponto aleatório, este método procede pela seleção aleatória do nó subsequente, baseando-se na proximidade imediata. Durante o processo de aprendizagem, o DeepWalk incorpora o modelo Word2Vec **Error! Reference source not found.**[5], especificamente a arquitetura *skip-gram*. Esta abordagem conceptualiza cada nó como uma entidade análoga a uma palavra, e a sequência completa de nós como uma sentença. O objetivo é aprimorar a representação dos nós de forma

que aqueles frequentemente co-localizados nas caminhadas aleatórias apresentem *embeddings* semelhantes. O DeepWalk é primordialmente orientado para a análise macroestrutural da rede, apresentando limitações quanto à adaptabilidade para a exploração de aspectos locais.

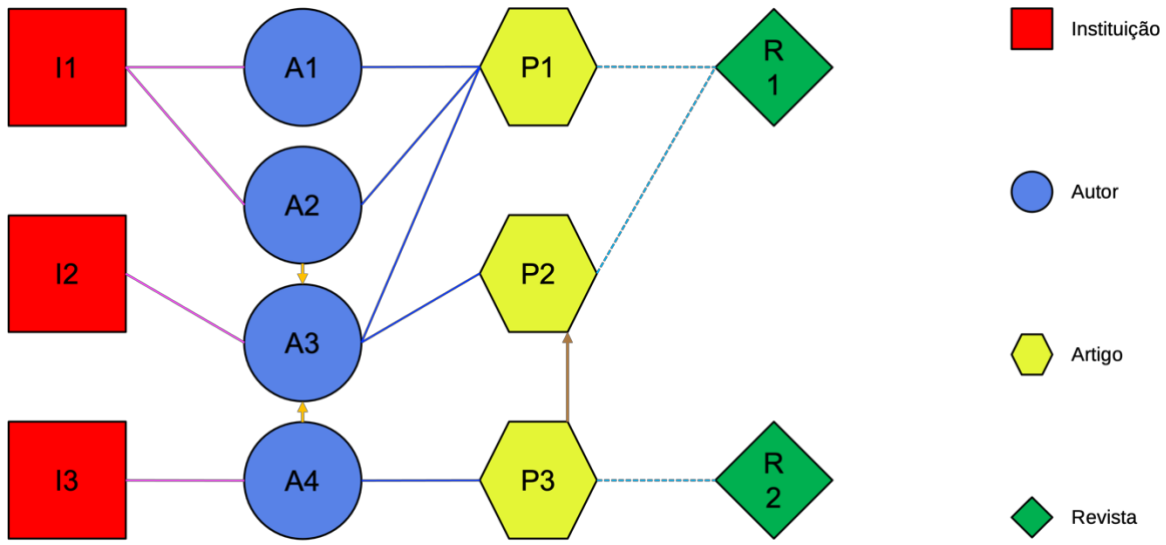
Em contrapartida, o Node2Vec, mantendo a base das caminhadas aleatórias, introduz um mecanismo diferenciado, através de caminhadas aleatórias enviesadas, reguladas por dois parâmetros: retorno (p) e in-out (q). Esses parâmetros são cruciais para balancear a exploração entre a vizinhança imediata nodal e regiões mais distantes da rede. O Node2Vec também se vale do modelo *skip-gram* para a geração de *embeddings*, diferenciando-se, contudo, na maneira como as sequências de entrada são influenciadas pelos parâmetros de enviesamento. Esta técnica se destaca pela sua versatilidade na exploração da rede, possibilitando um ajuste mais preciso entre a análise local, assemelhada à busca em largura, e a investigação de elementos mais afastados, comparável à busca em profundidade.

2.2 Metapath2vec

Em redes heterogêneas, encontram-se diferentes tipos de nós e arestas, cada qual representando distintas entidades e relações. Caracteriza-se pela diversidade, contrastando com as redes homogêneas onde a uniformidade prevalece. Nestas redes, a complexidade estrutural é notável, exigindo abordagens analíticas multifacetadas para compreender as variadas interações e influências entre os diversos tipos de nós. Comum em diversos campos, como biologia, sistemas de recomendação e análise de redes sociais, apresentam desafios únicos devido à sua complexidade. A análise e modelagem dessas redes requerem métodos capazes de lidar com a diversidade e a riqueza de informações, tornando-as fundamentais para sua compreensão.

A ilustração da Figura 3 exemplifica tal complexidade. Neste exemplo, há 12 nós, dos quais, subdividem em 4 tipos: instituição, autor, artigo e revista. Não obstante, existem 14 arestas representando relações distintas, como o segmento cor rosa, que correlaciona quais autores estão associados a cada instituição.

Figura 3 – Representação de uma rede heterogênea na área acadêmica.

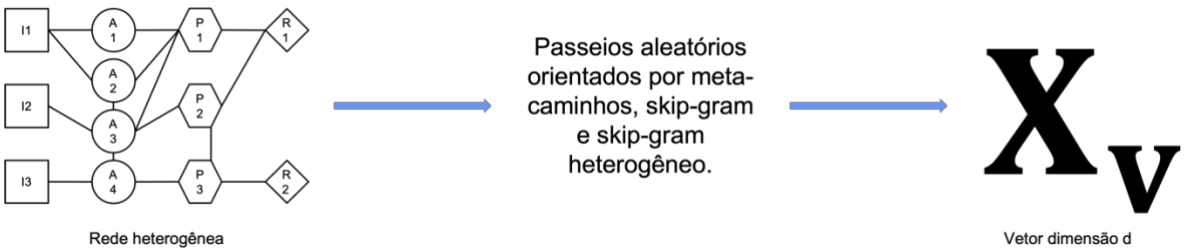


Fonte: autoria própria.

Diante das características apresentadas, define-se uma rede heterogênea como um grafo $G = (V, E, T)$ no qual cada nó v e cada aresta e estão associados às suas funções de mapeamento $\phi(v): V \rightarrow T_V$ e $\varphi(e): E \rightarrow T_E$. T_V e T_E denotam os conjuntos de tipos de objetos e relações, atendendo ao critério, $|T_V| + |T_E| > 2$, [2][3][4].

Dada a definição, emergiu o questionamento de como adaptar o modelo de *embedding* para redes heterogêneas, Figura 4. A solução proposta por Dong et al. [1], reside em implementar o *skip-gram* com passeios aleatórios com a condição de contorno de atender a meta-caminhos orientados, modelo este, denominado MetaPath2Vec.

Figura 4 – Problemática *embeddings* em redes heterogêneas.



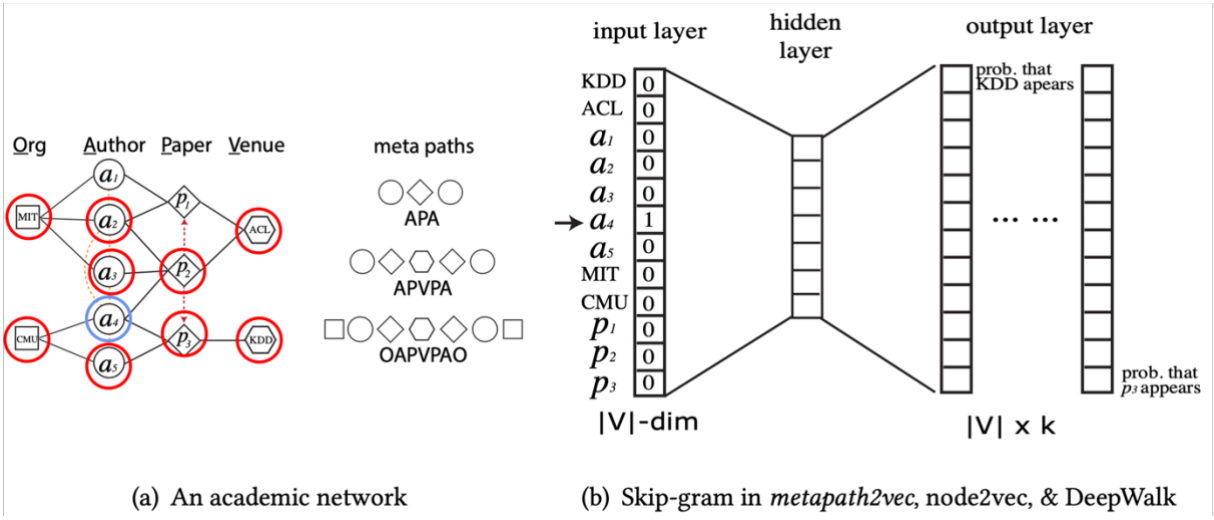
Fonte: autoria própria.

O modelo matemático começa com a formalização do método skip-gram a redes heterogêneas, como mostrado na Expressão 1. Dada a condição de contorno $|T_V| > 1$, o objetivo é maximizar a probabilidade do contexto $N_t(v), t \in T_V$ dado um nó v [1]. $N_t(v)$ é uma função que infere a vizinhança do nó v para seu respectivo tipo t, sendo $p(c_t \mid v; \theta)$ a função

softmax definida como $\frac{e^{X_{c_t} \cdot X_v}}{\sum_{u \in V} e^{X_u \cdot X_v}}$, onde X_v é o vetor *embedding* de cada nó da matriz **X** [5][6][7][8].

$$\arg\max_{\theta} \sum_{v \in V} \sum_{t \in T_V} \sum_{c_t \in N_t(v)} \log p(c_t \mid v; \theta) \quad (1)$$

Figura 5 – Rede heterogênea, meta-caminhos e skip-gram.



Fonte: imagem adaptada do artigo: DONG, Y.; CHAWLA, N. V.; SWAMI, A. Metapath2vec: Scalable representation learning for heterogeneous networks. In: Proceedings Of The 23rd Acm Sigkdd International Conference On Knowledge Discovery And Data Mining. Proceedings [...]. Halifax, NS, CA. ACM, p. 135–144, 2017.

A expressão é ilustrada na Figura 5(a), com os nós exemplificados circunscritos. Dado a vizinhança do nó *autors* (a_4), estruturalmente, este está próximo dos nós autors (a_2, a_3 e a_5), *venues* (ACL e KDD), *organizations* (CMU e MIT) assim como dos *papers* (p_2 e p_3) [1].

A amostragem negativa, desenvolvida por Mikolov et al. [5], cujo objetivo é a otimização de processamento, foi implementada por Dong et al. [1] adaptando-a de sua aplicação nativa, corpus textuais, para uma rede heterogênea. Ou seja, a metodologia utilizada para uma frase está para uma rede, assim como uma palavra está para um nó. Portanto, dada uma amostragem negativa M, a Expressão 1 é atualizada conforme a Expressão 2, cujo fator sigma é $\sigma(x) = \frac{1}{1+e^{-x}}$ e $P(u)$ é a distribuição pré-definida onde um nó negativo u^m é escolhido para a amostragem negativa. Em contrapartida, o modelo não diferencia o tipo de nó para a amostragem negativa, tratando-os de forma homogênea [1]. Ou seja, amostragem negativa é baseada em uma distribuição que é independente do tipo de nó. As amostras são selecionadas aleatoriamente, sem considerar a estrutura ou o tipo dos nós na rede. Isso pode induzir a

amostras negativas que não são tão informativas, uma vez que a probabilidade de seleção de cada nó é a mesma, independentemente da sua relevância ou frequência na rede.

$$\log \sigma\left(X_{c_t} \cdot X_v\right) + \sum_{m=1}^{M} \mathbb{E}_{u^m \sim P(u)}[\log\ \sigma(-X_{u^m} \cdot X_v)] \tag{2}$$

Em consonância, Sun et al. [9] demonstrou que passeios aleatórios tendem a ser enviesados dado que a probabilidade é normalizada sobre os vizinhos do nó atual ignorando seus respectivos tipos, implicando que cada nó vizinho tem a mesma chance de ser escolhido. Em função disso, definiu-se os passeios aleatórios orientados por meta-caminhos, formalizado por meio do esquema $\mathcal{P}$: $V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \cdots V_t \xrightarrow{R_t} V_{t+1} \cdots \xrightarrow{R_{l-1}} V_l$, em que $R = R_1 \circ R_2 \circ \cdots \circ R_{l-1}$ são as relações compostas entre os tipos de nós V_1 a V_l [2].

A Equação 3 define como os meta-caminhos orientam os passeios aleatórios no grafo $G = (V, E, T)$ com o intuito de discriminar o tipo de nó e aresta, em sintonia com o esquema definido [1].

O termo $p\left(v^{i+1} \mid v_t^i, \mathcal{P}\right)$ retorna a probabilidade de transição para o próximo nó v^{i+1}, dado o tipo do nó atual v_t^i e o esquema $\mathcal{P}$. Esta formalização, abrange 3 cenários, cujo primeiro retorna o valor 0 caso a aresta não exista, ou seja, $\left(v^{i+1}, v_t^i\right) \notin E$. O segundo caso, o qual também retorna um valor igual a 0, atende a condição da existência da aresta $\left(v^{i+1}, v_t^i\right) \in E$, todavia, o tipo de nó v^{i+1} verificado pela função mapeamento difere do critério do esquema, $\phi\left(v^{i+1}\right) \neq t + 1$. Por último, o qual ambas as condições são satisfeitas, a probabilidade de transição é calculada avaliando quantitativamente e qualitativamente (tipo de nó) todos os nós vizinhos v^{i+1} de v_t^i, reiterando, atendendo à premissa imposta pelo esquema. Isto é, 1 divido pela cardinalidade da vizinhança, representado pela expressão $\frac{1}{|N_{t+1}(v_t^i)|}$.

$$p\left(v^{i+1} \mid v_t^i, \mathcal{P}\right) = \begin{cases} \dfrac{1}{\left|N_{t+1}\left(v_t^i\right)\right|} & \left(v^{i+1}, v_t^i\right) \in E, \phi\left(v^{i+1}\right) = t + 1 \\ 0 & \left(v^{i+1}, v_t^i\right) \in E, \phi\left(v^{i+1}\right) \neq t + 1 \\ 0 & \left(v^{i+1}, v_t^i\right) \notin E \end{cases} \tag{3}$$

Por último, há uma boa prática em utilizar meta-caminhos de forma simétrica, proporcionando recursividade para os passeios aleatórios. Isto é atendido quando a igualdade $t = l$ é verdade no esquema $\mathcal{P}$, obtendo-se a identidade $p(v^{i+1} \mid v_t^i) = p(v^{i+1} \mid v_1^i)$ [2][9][10].

2.3 Metapath2vec++

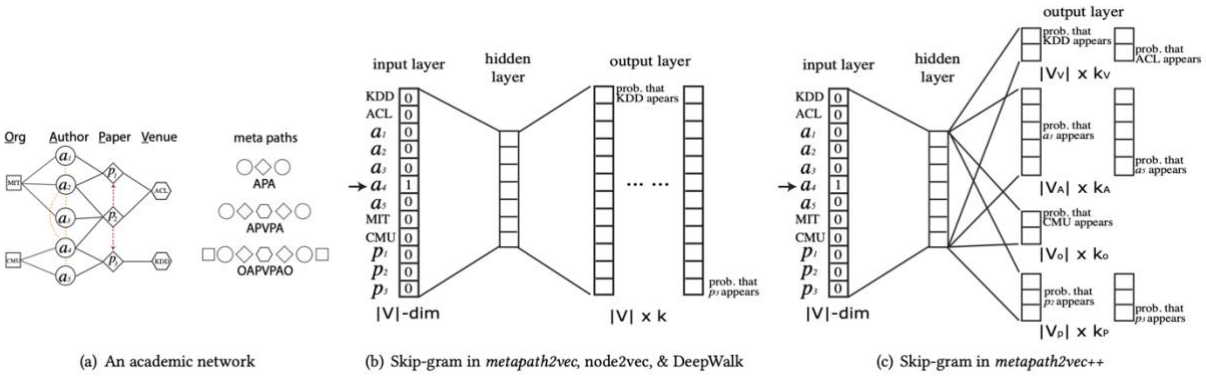
Embora o MetaPath2Vec, discrimine o contexto dos nós vizinhos baseado em seu tipo, por meio da função vizinhança, esta informação não é propagada para a função *softmax*. Portanto, ao inferir um específico tipo de contexto c_t na função vizinhança dado um nó v, o modelo não diferencia o tipo de nó para a amostra negativa [1].

Para contornar esta indistinção do tipo de nó, Dong et al. [1] propuseram a normalização da função *softmax* em função do tipo de nó do contexto c_t, Equação 4 (destaque em amarelo). Desta maneira, V_t é o conjunto de nós de tipo t na rede, especificando, um conjunto de distribuições multinomiais para cada tipo de vizinhança na camada de saída do modelo *skip-gram*.

$$p(c_t \mid v; \theta) = \frac{e^{X_{c_t} \cdot X_v}}{\sum_{u_t \in V_t} e^{X_{u_t} \cdot X_v}} \tag{4}$$

A Figura 6(b,c) apresenta a diferença da dimensão da distribuição multinomial na camada de saída do *skip-gram* entre os modelos MetaPath2Vec e MetaPath2Vec++. Ao passo que no primeiro, esta é igual ao número de nós da rede, no segundo modelo, sua dimensão na camada de saída é determinada pelo t números de nós dado t tipos de nós [1]. Dong et al. [1] exemplificaram esta dessemelhança, utilizando o nó a_4 na camada de entrada do *skip-gram* do modelo MetaPath2Vec++. Na camada de saída, evidenciado na Figura 6(c), há 4 conjuntos de distribuição multinomial, cada um correspondendo a um tipo de nó: *venues* V , *authors* A, *organizations* O , e *papers* P [1].

Figura 6 – Diferença entre skip-gram do modelo metapath2vec e metapath2vec++



Fonte: DONG, Y.; CHAWLA, N. V.; SWAMI, A. Metapath2vec: Scalable representation learning for heterogeneous networks. In: Proceedings Of The 23rd Acm Sigkdd International Conference On Knowledge Discovery And Data Mining. Proceedings [...]. Halifax, NS, CA. ACM, p. 135–144, 2017.

Fundamentado pelo modelo PTE de Tang et al. [11], a distribuição de amostragem também é definida considerando o tipo de nó do vizinho c_t o qual é o alvo a ser predito [1]. A Equação 5 apresenta a função objetivo.

$$O(\mathrm{X}) = \log\, \sigma\left(X_{c_t} \cdot X_v\right) + \sum_{m=1}^{M} \mathbb{E}_{u_t^m \sim P_t(u_t)}\left[\log\, \sigma\left(-X_{u_t^m} \cdot X_v\right)\right] \tag{5}$$

O primeiro termo, $\log\, \sigma\left(X_{c_t} \cdot X_v\right)$, é o logaritmo da probabilidade de c_t e v serem vizinhos no espaço de características. Um valor alto indica que c_t e v estão próximos no espaço de características, o que é desejável para representações que capturam a estrutura de vizinhança. A expressão é desenvolvida, calculando o produto escalar entre os dois vetores, $X_{c_t} \cdot X_v$. Em seguida, a função sigmoide $\sigma(x) = \frac{1}{1+e^{-x}}$ é aplicada ao produto escalar. Esta função mapeia o valor real obtido pelo produto escalar para um intervalo entre 0 e 1, interpretado como probabilidade. Finalmente, o logaritmo natural é aplicado ao resultado. A razão para utilizá-lo é transformar a multiplicação de probabilidades em uma soma de log-probabilidades, o que é matematicamente mais estável e conveniente para otimização, especialmente quando o produto de muitas probabilidades pode levar a um *underflow* numérico.

Já o segundo, $\sum_{m=1}^{M} \mathbb{E}_{u_t^m \sim P_t(u_t)}\left[\log\, \sigma\left(-X_{u_t^m} \cdot X_v\right)\right]$ é a soma dos logaritmos das probabilidades negativas, ou seja, a probabilidade de que um nó amostrado aleatoriamente u_t não seja um vizinho de v. Este termo contribui em segregar os vetores de representação de não-vizinhos no espaço de características. A expressão $\mathbb{E}_{u_t^m \sim P_t(u_t)}\left[\log\, \sigma\left(-X_{u_t^m} \cdot X_v\right)\right]$ é o valor esperado sobre a distribuição de amostragem $P_t(u_t)$, pelo cálculo da média ponderada de

$\log \sigma\left(-X_{u_t^m} \cdot X_v\right)$, onde os pesos são dados pela probabilidade de cada amostra negativa. É crucial mencionar o sinal negativo antes do produto escalar, pois este inverte a similaridade, de forma que valores altos do produto escalar resultem em probabilidades baixas após a aplicação da sigmoide. Por último, realiza-se o somatório sobre as M amostras negativas das probabilidades das sigmóides invertidas, o que de fato, representa a integração da contribuição das múltiplas amostras negativas para a função de perda, promovendo a diferenciação entre o nó alvo v e nós que não são seus vizinhos.

Em resumo, a função objetivo combina o log-probabilidade de uma observação positiva (vizinhos reais no grafo) com a expectativa do log-probabilidade de observações negativas (nós que não são vizinhos). O propósito é maximizar a função de perda, o que equivale a encontrar representações de nós que mantenham a proximidade entre vizinhos reais e distanciamento entre nós não relacionados. A maximização desta função de perda tende a resultar em representações vetoriais que expresse o mais fielmente possível a estrutura da rede heterogênea original. Adicionalmente, esta abordagem contribui para a eficácia do modelo, pois permite um ajuste dos parâmetros para um subconjunto representativo de não-contextos, em vez de todos os possíveis não-contextos.

Os gradientes da função objetivo são apresentados nas Equações 6 e 7, onde $\mathbb{I}_{c_t}$ é uma função indicadora que determina se u_t^m está no contexto de vizinhança do nó c_t, onde $m = 0$ implica em $u_t^0 = c_t$ [1].

$$\frac{\partial O(\mathrm{X})}{\partial X_{u_t^m}} = \left(\sigma\left(X_{u_t^m} \cdot X_v - \mathbb{I}_{c_t}[u_t^m]\right)\right) X_v \tag{6}$$

$$\frac{\partial O(\mathrm{X})}{\partial X_v} = \sum_{m=0}^{M} \left(\sigma\left(X_{u_t^m} \cdot X_v - \mathbb{I}_{c_t}[u_t^m]\right)\right) X_{u_t^m} \tag{7}$$

O primeiro gradiente é derivado em relação ao vetor de características do nó amostrado, seja ele uma amostra positiva ou negativa. Este gradiente é obtido avaliando a transformação logística da similaridade entre o vetor de características do nó amostrado e o vetor do nó de contexto, ajustada pela presença ou ausência do nó amostrado na vizinhança do nó de contexto, conforme indicado pela função indicadora. O resultado desta avaliação é então multiplicado pelo vetor de características do nó de contexto, fornecendo a direção e a magnitude para ajustar o vetor de características do nó amostrado, a fim de alinhar mais precisamente as representações de nós que são vizinhos.

O segundo gradiente, por sua vez, é calculado em relação ao vetor de características do nó de contexto. Neste caso, o procedimento envolve a acumulação dos gradientes derivados para cada uma das amostras, tanto positivas quanto negativas, consideradas na função objetivo. Similarmente ao primeiro gradiente, a transformação logística é aplicada à similaridade entre cada vetor de características amostrado e o vetor de características do nó de contexto. Entretanto, esta medida é ponderada para todas as amostras, e o acúmulo destas contribuições ponderadas resultam no gradiente para o vetor de características do nó de contexto. O ajuste subsequente deste vetor busca promover uma representação mais fidedigna da estrutura da vizinhança na rede.

Adicionalmente, estes gradientes são fundamentais no processo de otimização do modelo, sendo utilizados na atualização iterativa dos vetores de características através do algoritmo de descida do gradiente estocástico (SGD), visando a minimização da função de perda e, consequentemente, a melhoria das representações de nós na estrutura da rede heterogênea.

3 METODOLOGIA E DESENVOLVIMENTO

A Tabela 1 exibe os seis *datasets* heterogêneos selecionados da biblioteca PyTorch-Geometric. A escolha desses *datasets* foi pautada na máxima similaridade com a estrutura do conjunto de dados AMiner, que também integra a seleção. Tal critério de seleção fundamenta-se pela utilização do AMiner no artigo que constitui a base teórica deste estudo. Adicionalmente, este tópico do trabalho apresentará as características de cada conjunto de dados, incluindo seus parâmetros e, quando aplicável, as modificações implementadas. Por seguinte, discorre-se a respeito do algoritmo desenvolvido em *Python*, acompanhado de suas condições de contorno específicas.

Tabela 1 – *Datasets* com seus respectivos links

Dataset	**Link**
AMiner	https://pytorch-geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.AMiner.html#torch_geometric.datasets.AMiner
DBLP	https://pytorch-geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.DBLP.html#torch_geometric.datasets.DBLP
HGBD_ACM	https://pytorch-geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.HGBDataset.html#torch_geometric.datasets.HGBDataset
IMDB	https://pytorch-geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.IMDB.html#torch_geometric.datasets.IMDB
MovieLens1M	https://pytorch-geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.MovieLens1M.html#torch_geometric.datasets.MovieLens1M
OGB_MAG*	https://pytorch-geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.OGB_MAG.html#torch_geometric.datasets.OGB_MAG

Fonte: autoria própria.

3.1 Principais parâmetros do modelo metapath2vec++

Na implementação do modelo MetaPath2Vec++, a integração com o ambiente do Google Colab e a montagem do Google Drive foram as etapas iniciais. A utilização desta infraestrutura proporciona acesso a recursos computacionais avançados e onerosos, como GPUs, cruciais em operações intensivas de aprendizado de máquina.

Os tópicos abaixo apresentam de forma sucinta os principais parâmetros do modelo MetaPath2Vec++ e suas respectivas relevâncias. Esses parâmetros são fundamentais para a correta funcionalidade e eficácia, influenciando diretamente a qualidade e a precisão dos *embeddings* gerados. A compreensão detalhada de cada um desses parâmetros é essencial para otimizar o desempenho do modelo e garantir resultados robustos em aplicações práticas.

i. Dimensões dos *embeddings* (embedding_dim): este parâmetro define o tamanho dos vetores de *embedding* para cada nó na rede. Um *embedding* maior pode capturar mais informações, mas também requer mais memória e pode levar a um treinamento mais lento.
ii. Metapath (metapath): o metapath é uma sequência de tipos de nós e arestas que define o caminho a ser seguido nas caminhadas aleatórias. Ele é crucial para determinar como diferentes tipos de nós e relações são explorados pelo modelo. Cada metapath é uma lista de tuplas, onde cada tupla representa um tipo de relação na rede (por exemplo, (tipo_nó_origem, tipo_relacionamento, tipo_nó_destino). O conjunto destes mencionado no capítulo anterior é denominado esquema.
iii. Comprimento da caminhada (walk_length): este parâmetro define quantos passos uma caminhada aleatória irá percorrer. Caminhadas mais longas podem capturar informações de contexto mais amplas, mas também podem introduzir ruído e aumentar o tempo de computação.
iv. Tamanho do contexto (context_size): refere-se ao número de nós vizinhos considerados para cada nó durante o treinamento. Um contexto maior pode fornecer mais informações, mas também pode diluir as características específicas de cada nó.
v. Caminhadas por nó (walks_per_node): este parâmetro define quantas caminhadas aleatórias serão iniciadas para cada nó. Um número maior de caminhadas por nó aumenta a quantidade de dados de treinamento, o que pode melhorar a qualidade do *embedding*, mas também aumenta o tempo de treinamento.
vi. Número de amostras negativas (num_negative_samples): no treinamento com amostragem negativa, este parâmetro define quantas amostras negativas são usadas para cada amostra positiva. A amostragem negativa, como dito anteriormente, ajuda o modelo a diferenciar entre conexões reais e aleatórias.
vii. Esparsidade (sparse): Este parâmetro booleano determina se os gradientes em relação à matriz de pesos serão esparsos. A esparsidade pode ser benéfica em termos de eficiência de memória, especialmente para grandes conjuntos de dados.
viii. Dicionário de índices de arestas (edge_index_dict): embora não seja um parâmetro ajustável como os outros, é importante mencionar que o modelo requer um dicionário mapeando cada tipo de aresta para seus respectivos índices de arestas no grafo.
ix. Dicionário de número de nós (num_nodes_dict): similarmente, este dicionário, que mapeia cada tipo de nó para seu número total, é essencial para a configuração do modelo, embora não seja um parâmetro ajustável pelo usuário.

3.2 Condições de contorno e premissas

Com o intuito de maximizar a representatividade dos resultados tanto quanto sua reprodutibilidade, a Figura 7 apresenta o ambiente operacional e as configurações de hardware do *Colab Pro+* o qual foi utilizado por todos os *datasets*.

Figura 7 – Configuração ambiente operacional

```
PRETTY_NAME="Ubuntu 22.04.3 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.3 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
Tue Nov 21 12:35:20 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   47C    P8    11W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Fonte: autoria própria.

Paralelo ao raciocínio anterior, a Figura 8 apresenta as versões do PyTorch-Geometric utilizadas.

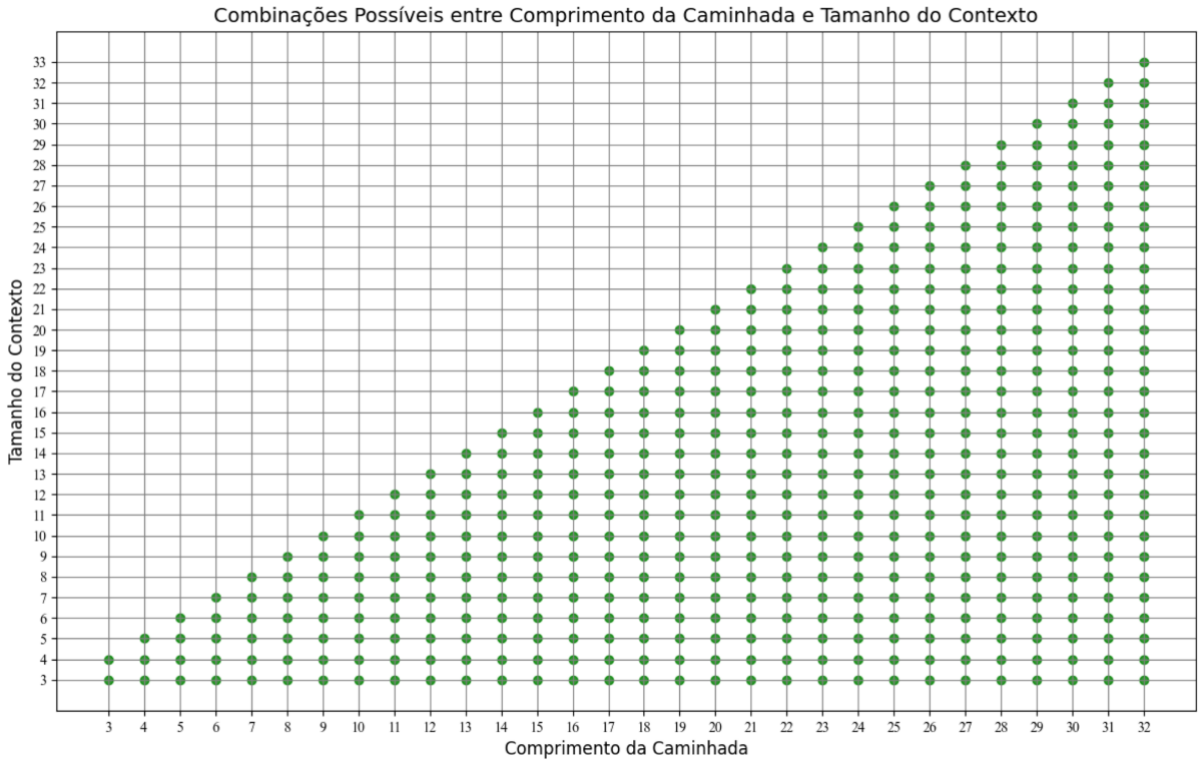
Figura 8 – Bibliotecas e versões do PyTorch-Geometric

```
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
!pip install -q torch-scatter torch_sparse -f https://data.pyg.org/whl/torch-2.1.0+cu118.html
```

Fonte: autoria própria.

Em virtude de limitações computacionais, todos os *datasets* executaram uma única época para cada combinação entre comprimento da caminhada e tamanho do contexto, variando respectivamente entre [3;32] e [3;33]. Totalizou-se 495 combinações, respeitando ao critério de continuação, as quais estão representadas graficamente pelos pontos verdes na Figura 9.

Figura 9 – Combinações entre tamanho do contexto e comprimento da caminhada



Fonte: autoria própria.

3.3 Datasets e configurações

A Tabela 2 apresenta os *datasets* utilizados com suas principais características.

Tabela 2 – Características dos *datasets* utilizados

Dataset	**Tipos de Nós**	**Nome (Quantidade de Nós)**	**Detalhes das Arestas**
AMiner	3	Author (1693531), Venue (3883), Paper (3194405)	(paper, written_by, author): edge_index=[2, 9323605], (author, writes, paper): edge_index=[2, 9323605], (paper, published_in, venue): edge_index=[2, 3194405], (venue, publishes, paper): edge_index=[2, 3194405]
DBLP	4	Author (4057), Paper (14328), Term (7723), Conference (20)	(author, to, paper): edge_index=[2, 19645], (paper, to, author): edge_index=[2, 19645], (paper, to, term): edge_index=[2, 85810], (paper, to, conference): edge_index=[2, 14328], (term, to, paper): edge_index=[2, 85810], (conference, to, paper): edge_index=[2, 14328]
HGBD_ACM	4	Paper (3025), Author (5959), Subject (56), Term (1902)	(paper, cite, paper): edge_index=[2, 5343], (paper, ref, paper): edge_index=[2, 5343], (paper, to, author): edge_index=[2, 9949], (author, to, paper): edge_index=[2, 9949], (paper, to, subject): edge_index=[2, 3025], (subject, to, paper): edge_index=[2, 3025], (paper, to, term): edge_index=[2, 255619], (term, to, paper): edge_index=[2, 255619]
IMDB	3	Movie (4278), Director (2081), Actor (5257)	(movie, to, director): edge_index=[2, 4278], (movie, to, actor): edge_index=[2, 12828], (director, to, movie): edge_index=[2, 4278], (actor, to, movie): edge_index=[2, 12828]

MovieLens1M	2	Movie (3883), User (6040)	(user, rates, movie): edge_index=[2, 1000209], (movie, rated_by, user): edge_index=[2, 1000209]
OGB_MAG*	4	Paper (36355), Author (13966), Institution (242), Field of Study (1374)	(author, affiliated_with, institution): edge_index=[2, 3072], (author, writes, paper): edge_index=[2, 25033], (paper, cites, paper): edge_index=[2, 19535], (paper, has_topic, field_of_study): edge_index=[2, 35748], (field_of_study, contains, paper): edge_index=[2, 35748]

Fonte: autoria própria.

O asterisco presente no *dataset* "OGB_MAG" refere-se ao fato que este foi modificado, não representando, portanto, sua configuração original. O motivo decorreu do seu tamanho, incompatível com o limite de tempo de execução do ambiente operacional (24 horas) como também, a necessidade de ter adicionado uma aresta para que fosse possível definir um esquema cíclico.

A Figura 10 ilustra as características originais do "OGB_MAG", um *dataset* integrante do *Open Graph Benchmark: datasets for Machine Learning on Graphs* e parte do Microsoft *Academic Graph* (MAG). Este *dataset* se destaca pela magnitude de seu tamanho, abrangendo uma variedade de entidades como artigos acadêmicos, autores, instituições educacionais e campos de estudo, formando uma complexa rede de inter-relações. Cada artigo no *dataset* é enriquecido com um vetor de características de 128 dimensões, gerado pelo modelo Word2Vec, enquanto outros tipos de nós não possuem características adicionais. A principal tarefa analítica proposta pelo “OGB_MAG” é a predição do local de publicação dos artigos, abrangendo 349 locais únicos.

Para carregar o *dataset* “OGB_MAG”, são utilizados parâmetros obrigatórios e opcionais, dos quais *root, transform* e *pre_transform* são comuns aos seis conjuntos selecionados neste trabalho. O parâmetro *root* especifica o diretório base para o armazenamento dos dados, já os parâmetros *transform* e *pre_transform* permitem a aplicação de transformações no objeto *HeteroData*. Enquanto *transform* é aplicado antes de cada acesso ao *dataset*, o *pre_transform*, por sua vez, é executado uma única vez antes de salvar os dados localmente ou em nuvem. Esses parâmetros conferem, inclusive ao "OGB_MAG", uma flexibilidade notável para o *benchmark* de modelos de aprendizado de máquina, propiciando uma extensa gama referente à exploração das dinâmicas e interconexões.

Figura 10 – Características originais do *dataset* OGB_MAG

```
HeteroData(
  paper={
    x=[736389, 128],
    year=[736389],
    y=[736389],
    train_mask=[736389],
    val_mask=[736389],
    test_mask=[736389],
  },
  author={ num_nodes=1134649 },
  institution={ num_nodes=8740 },
  field_of_study={ num_nodes=59965 },
  (author, affiliated_with, institution)={ edge_index=[2, 1043998] },
  (author, writes, paper)={ edge_index=[2, 7145660] },
  (paper, cites, paper)={ edge_index=[2, 5416271] },
  (paper, has_topic, field_of_study)={ edge_index=[2, 7505078] }
)
```

Fonte: autoria própria.

Implementou-se, através do *pre_transform*, uma função que adicionou a aresta ('field_of_study', 'contains', 'paper'), invertendo-a de sua original, ('paper', 'has_topic', 'field_of_study'). Posteriormente, procedeu-se reduzindo o *dataset* para 6% de seu tamanho nativo, mediante o desenvolvimento de uma classe que assegurou a reprodutibilidade e a eliminação de nós isolados, mitigando eventuais inconsistências. Estas transformações, em virtude da simplicidade em unificá-las por meio de uma terceira função, evitou a necessidade de implementar classes específicas do PyTorch-Geometric para que o parâmetro *pre_transform* aceitasse mais de uma função, Figura 11.

Figura 11 – Script para modificação do *dataset* OGB_MAG

```
# Inverter a aresta [('paper', 'has_topic', 'field_of_study')].edge_index
from torch_sparse import transpose
def add_inverse_edges(data):  # Define a função add_inverse_edges que
recebe um parâmetro data

    # Adiciona arestas inversas para a relação ('field_of_study',
'contains', 'paper')
    data[('field_of_study', 'contains', 'paper')].edge_index = transpose(
        data[('paper', 'has_topic', 'field_of_study')].edge_index,  #
Arestas originais
        None,  # Não é especificado um tensor para armazenar os valores das
arestas transpostas
        m=data['paper'].num_nodes,  # Número de nós do tipo 'paper'
```

```
        n=data['field_of_study'].num_nodes)[0]  # Número de nós do tipo
'field_of_study'

    return data  # Retorna o objeto data modificado

# Reduzir Dataset para 6% do tamanho original
from torch_geometric.data import HeteroData
from torch_geometric.transforms import BaseTransform
import random
class ReduceDatasetSize(BaseTransform):
    def __call__(self, data):
        # Iniciar o cronômetro para calcular o tempo de execução
        start_time = time.time()

        # Definir a semente para garantir a reprodutibilidade
        seed = 1
        torch.manual_seed(seed)
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)  # se estiver usando multi-GPU.
        np.random.seed(seed)  # se estiver usando numpy
        random.seed(seed)  # se estiver usando o módulo random

        # Garantir o comportamento determinístico no CUDA
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False

        # Passo 1: Selecionar 6% de nós de cada tipo
        print("Passo 1: Selecionando 6% dos nós de cada tipo...")
        subset_dict = {}
        for node_type in data.node_types:
            total_nodes = data[node_type].num_nodes
            retain_count = int(total_nodes * 0.06)
            subset = torch.randperm(total_nodes)[:retain_count]
            subset_dict[node_type] = subset
        print("Passo 1 concluído - {:.2f}s".format(time.time() -
start_time))

        # Configurar a semente novamente
        torch.manual_seed(seed)

        # Passo 2: Criar o subgrafo a partir dos subconjuntos de nós
selecionados
        print("Passo 2: Criando o subgrafo...")
        subset_data = data.subgraph(subset_dict)
        print("Passo 2 concluído - {:.2f}s".format(time.time() -
start_time))

        # Passo 3: Remover nós isolados
        print("Passo 3: Removendo nós isolados...")
```

```
        removed_isolated_nodes = True
        while removed_isolated_nodes:
            removed_isolated_nodes = False
            degrees = {}
            for edge_type in subset_data.edge_types:
                src, _, dst = edge_type
                edge_index = subset_data[edge_type].edge_index
                for s, d in zip(edge_index[0], edge_index[1]):
                    degrees[src, s.item()] = degrees.get((src, s.item()),
0) + 1
                    degrees[dst, d.item()] = degrees.get((dst, d.item()),
0) + 1

            isolated_nodes = {node_type: [] for node_type in
subset_data.node_types}
            for node_type in subset_data.node_types:
                for i in range(subset_data[node_type].num_nodes):
                    if degrees.get((node_type, i), 0) == 0:
                        isolated_nodes[node_type].append(i)
                        removed_isolated_nodes = True

            for node_type, isolated in isolated_nodes.items():
                if isolated:
                    valid_indices = [i for i in
range(subset_data[node_type].num_nodes) if i not in isolated]
                    valid_original_indices =
subset_dict[node_type][valid_indices]
                    subset_dict[node_type] = valid_original_indices
                    subset_data = data.subgraph(subset_dict)
        print("Passo 3 concluído - {:.2f}s".format(time.time() -
start_time))

        print("Processo concluído - {:.2f}s".format(time.time() -
start_time))

        return subset_data

# Unir ReduceDatasetSize e add_inverse_edges
def pre_transform(data):
    heterodata_reduce = ReduceDatasetSize()
    data = heterodata_reduce(data)
    data = add_inverse_edges(data)

    return data

# Carregar dataset aplicando o parâmetro pre_transform
path = osp.join('OGB_MAG_folder', 'OGB_MAG')
dataset = OGB_MAG(path, pre_transform=pre_transform)
data = dataset[0]
```

```
print(data)
# Definir o dispositivo de processamento (GPU ou CPU).
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

Fonte: autoria própria.

No desenvolvimento do código para a implementação do modelo MetaPath2Vec++, a Tabela 3 detalha os meta-caminhos empregados em cada conjunto de dados. Paralelamente, a Tabela 4 delineia os principais parâmetros de configuração do modelo, especificando suas respectivas variáveis entre parênteses. Notadamente, o conjunto de dados "OBG_MAG" apresenta uma peculiaridade em comparação aos demais no que tange ao número máximo de iterações necessárias para a convergência, parâmetro *max_iter*. Essa particularidade exigiu uma alteração na configuração, mesmo reduzindo-o 6% do tamanho original. Tal medida foi adotada em resposta a repetidos avisos de *"ConvergenceWarning: lbfgs failed to converge (status=1)"*, que persistiram mesmo após sucessivos incrementos no número máximo de iterações, até que finalmente, estabilizou-se com uma configuração de 600. Ou seja, tal aviso implica que o número de iterações realizadas pelo algoritmo otimizador, neste caso o LBFGS (regressão logística), foi insuficiente para atingir a convergência dentro do limite inicialmente estipulado.

O incremento do parâmetro para 4 vezes em relação aos demais pode ser atribuído à sofisticação intrínseca deste, que potencialmente apresenta uma estrutura mais desafiadora do que os outros. A complexidade do espaço de soluções, juntamente com a superfície de erro caracterizada por múltiplos mínimos locais, demandou um número maior de iterações para uma execução estável e eficaz com o intuito de encontrar um mínimo, teoricamente global, satisfatório. Esse ajuste foi uma medida técnica necessária para superar as dificuldades impostas, visando uma convergência aceitável.

Tabela 3 – *Datasets* com seus respectivos meta-caminhos

Dataset	**Meta-caminho**
AMiner	('author', 'writes', 'paper'), ('paper', 'published_in', 'venue'), ('venue', 'publishes', 'paper'), ('paper', 'written_by', 'author')
DBLP	('author', 'to', 'paper'), ('paper', 'to', 'conference'), ('conference', 'to', 'paper'), ('paper', 'to', 'author')
HGBD_ACM	('paper', 'to', 'author'), ('author', 'to', 'paper'), ('paper', 'to', 'subject'), ('subject', 'to', 'paper')
IMDB	('movie', 'to', 'director'), ('director', 'to', 'movie'), ('movie', 'to', 'actor'), ('actor', 'to', 'movie')
MovieLens1M	('user', 'rates', 'movie'), ('movie', 'rated_by', 'user')
OBG_MAG*	('paper', 'cites', 'paper'), ('paper', 'has_topic', 'field_of_study'), ('field_of_study', 'contains', 'paper')

Fonte: autoria própria.

Tabela 4 – *Datasets* com principais parâmetros do modelo MetaPath2Vec++

Parâmetros	**AMiner**	**DBLP**	**HGBD_ACM**	**IMDB**	**MovieLens1M**	**OBG_MAG***
Dimensão do e*mbedding* (embedding_dim)	128	128	128	128	128	128
Caminhadas por nó (walks_per_node)	3	3	3	3	3	3
Amostragem negativa (num_negative_samples)	1	1	1	1	1	1
Gradiente de peso esparso (sparse)	True	True	True	True	True	True
Taxa de aprendizado SparseAdam (lr)	0.01	0.01	0.01	0.01	0.01	0.01
Máximo de iterações (max_iter)	150	150	150	150	150	600

Fonte: autoria própria.

4 RESULTADOS E DISCUSSÃO

Os resultados serão analisados percorrendo sequencialmente uma análise qualitativa por meio de gráficos de contorno, sucedido de correlação da acurácia de teste, regressão linear múltipla, regressão polinomial e ANOVA.

4.1 Gráficos de contorno

Os gráficos da Figura 12 apresentam aspectos distintos na interação entre os parâmetros comprimento da caminhada e tamanho do contexto com a acurácia de teste. No caso do conjunto AMiner, é irrefutável que um aumento no comprimento da caminhada aliado a tamanho de contexto menores conduziram a uma acurácia maior, sugerindo que de caminhos mais extensos são eficazes na captura da estrutura e semântica dos dados. Contudo, essa melhoria apresenta um ponto de saturação, após o qual, incrementos adicionais no comprimento da caminhada não proporcionaram ganhos significativos na acurácia.

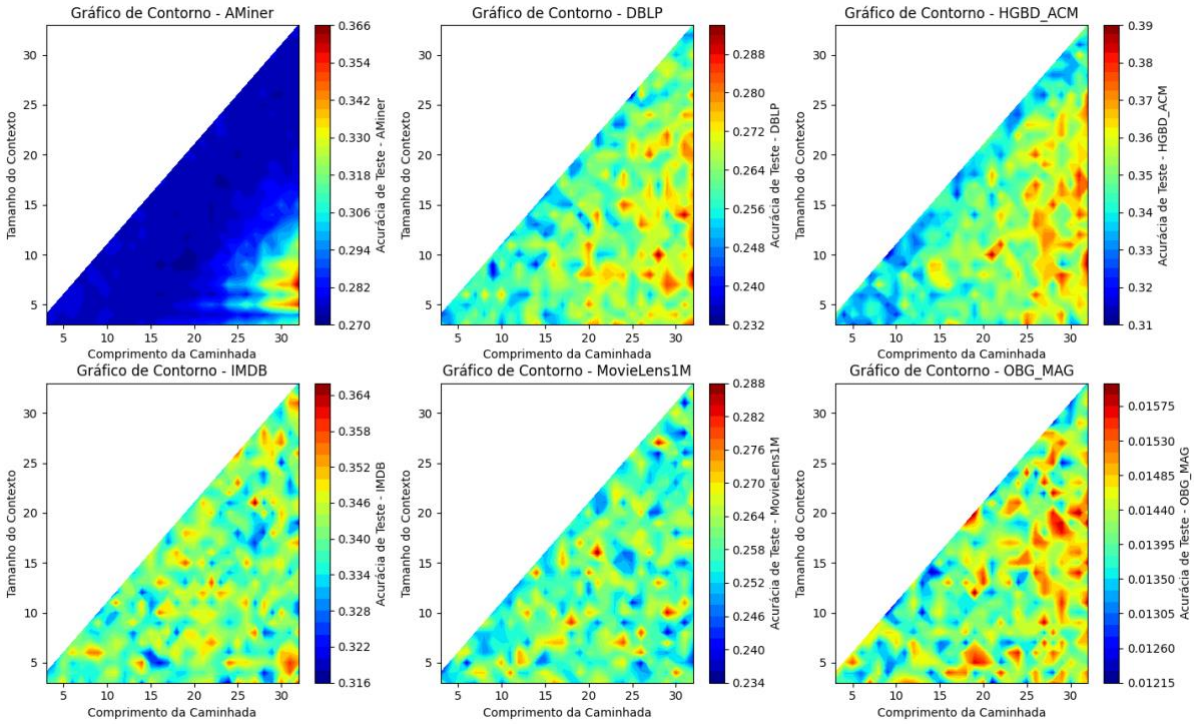
Em relação ao conjunto DBLP, observa-se um padrão similar ao AMiner em relação ao comprimento da caminhada, porém menos pronunciado, tamanho de contexto maiores e com uma zona de alta acurácia menos evidente. Isso indica que, embora o aumento do comprimento da caminhada possa ser vantajoso, este conjunto de dados pode ser menos influenciado por este parâmetro, ou talvez possua pontos ótimos esparsos. Esse comportamento pode representar uma estrutura de dados diferenciada ou a necessidade de ajustes de outros parâmetros do modelo.

Nos conjuntos HGBD_ACM e IMDB, os gráficos mostram uma distribuição mais dispersa, sugerindo uma interação mais complexa e menos previsível entre os parâmetros testados e a acurácia. A variação na acurácia é mais gradual, indicando que esses conjuntos podem ter estruturas que reagem de forma mais sutil às modificações nos parâmetros do modelo.

Para o conjunto MovieLens1M, a distribuição de alta acurácia é mais abrangente, sugerindo uma menor sensibilidade do modelo às variações nos parâmetros, eventualmente devido às características específicas dos dados ou a robustez do modelo referente a configuração de outros parâmetros. A zona de maior acurácia não se localiza nos extremos, mas em valores medianos de ambos os parâmetros, sugerindo um equilíbrio entre o comprimento da caminhada e o tamanho do contexto.

Por fim, o conjunto OBG_MAG exibe uma faixa de acurácia mais restrita e uma alta acurácia concentrada em áreas limitadas, enfatizando a necessidade de uma calibração precisa dos parâmetros para este *dataset*. De maneira geral, em todos os conjuntos analisados, percebe-se que determinadas combinações de comprimento da caminhada e tamanho do contexto otimizam a acurácia de teste. Contudo, a localização e a amplitude dessas regiões ótimas variam consideravelmente entre os conjuntos, sublinhando a importância de um ajuste específico do modelo MetaPath2Vec++ para cada conjunto de dados, visando um desempenho ótimo.

Figura 12 – Gráficos de contornos dos *datasets*



Fonte: autoria própria.

4.2 Correlação da acurácia de teste

Esta análise, sintetizada pela Figura 13, demonstra variações notáveis na correlação destes parâmetros com a acurácia nos respectivos conjuntos de dados.

Especificamente no AMiner, identifica-se uma correlação positiva de magnitude moderada (0.398) entre o comprimento da caminhada e a acurácia, sugerindo que na caminhadas maiores favorecem a precisão do teste. Em contrapartida, o tamanho do contexto

exibe uma correlação negativa também moderada (-0.314), o que implica que incrementos neste parâmetro podem não ser vantajosos para a acurácia neste *dataset*.

No DBLP, a correlação do comprimento da caminhada com a acurácia é ligeiramente superior (0.405) em comparação ao AMiner, enquanto o tamanho do contexto apresenta uma correlação insignificante (-0.023), indicando um impacto mínimo na acurácia.

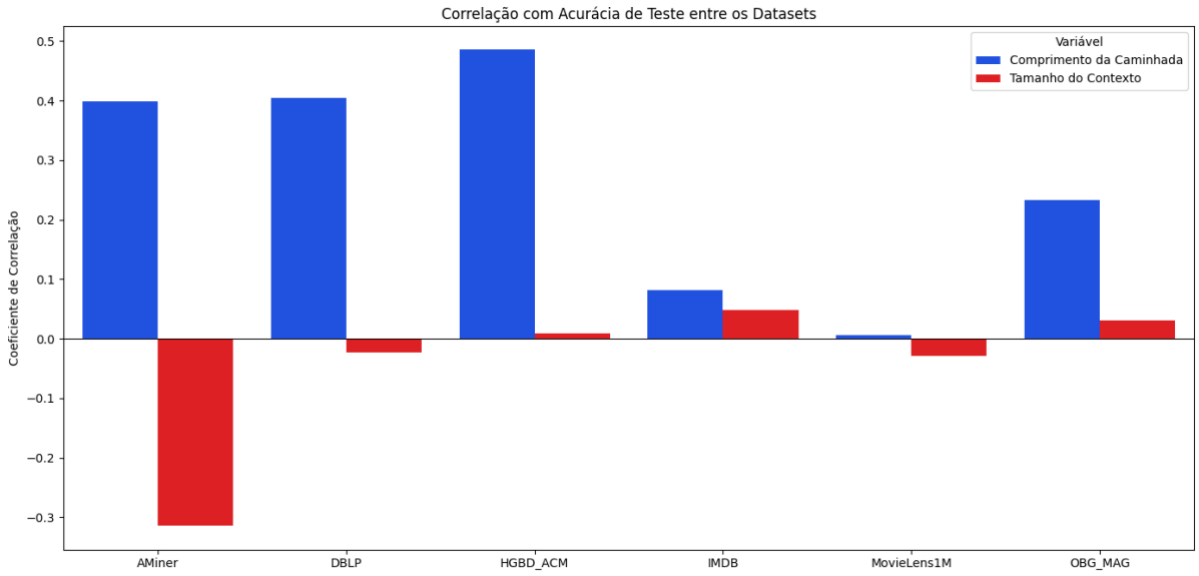
Para o HGBD_ACM, a correlação entre comprimento da caminhada e acurácia atinge o pico entre todos os conjuntos (0.486), revelando uma forte associação positiva. Por outro lado, o tamanho do contexto mostra uma correlação praticamente nula (0.009) com a acurácia.

No IMDB, ambas as variáveis apresentam correlações baixas, sendo 0.081 para o comprimento da caminhada e 0.048 para o tamanho do contexto, indicando uma influência marginal destes parâmetros.

No caso do MovieLens1M, as correlações de ambos os parâmetros com a acurácia são mínimas, registrando 0.006 para o comprimento da caminhada e -0.029 para o tamanho do contexto, sugerindo uma influência ínfima destes fatores na acurácia de teste.

Finalmente, no OBG_MAG, observa-se uma correlação moderada de 0.232 para o comprimento da caminhada e uma correlação baixa de 0.031 para o tamanho do contexto, sugerindo um efeito moderado do primeiro parâmetro na acurácia de teste.

Figura 13 – Correlação da acurácia de teste



Fonte: autoria própria.

4.3 REGRESSÃO LINEAR MÚLTIPLA

A Tabela 5 revela nuances importantes na relação entre o comprimento da caminhada, o tamanho do contexto e a acurácia de teste.

No AMiner, o modelo explica 50.7% da variação na acurácia de teste (R-quadrado ajustado de 50.5%), com um coeficiente positivo significativo para o comprimento da caminhada (0.0014, $p < 0.0001$) e um coeficiente negativo significativo para o tamanho do contexto (-0.0013, $p < 0.0001$). Isso indica que, neste *dataset*, um aumento no comprimento da caminhada melhora a acurácia, enquanto um aumento no tamanho do contexto a reduz.

Para o DBLP, o modelo tem um R-quadrado ajustado de 22.8%, com coeficientes menores para ambos os parâmetros (comprimento da caminhada: 0.0007, $p < 0.0001$; tamanho do contexto: -0.0004, $p < 0.0001$), sugerindo uma influência mais moderada destes parâmetros na acurácia de teste.

No HGBD_ACM, o R-quadrado ajustado é de 30.5%, com coeficientes positivos para ambos os parâmetros (comprimento da caminhada: 0.0011, $p < 0.0001$; tamanho do contexto: -0.0005, $p < 0.0001$), indicando uma relação positiva, porém mais fraca, entre os parâmetros e a acurácia de teste.

O IMDB apresenta um R-quadrado ajustado muito baixo (0.26%), com coeficientes próximos de zero para ambos os parâmetros (comprimento da caminhada: 0.00008, $p = 0.144$; tamanho do contexto: 0.00001, $p = 0.843$), sugerindo que outros fatores, não incluídos no modelo, podem ser mais relevantes para a acurácia de teste neste *dataset*.

Para o MovieLens1M, o R-quadrado ajustado é negativo (-0.27%), com coeficientes insignificantes para ambos os parâmetros (comprimento da caminhada: 0.00003, $p = 0.598$; tamanho do contexto: -0.00005, $p = 0.412$), indicando que os parâmetros estudados não têm impacto significativo na acurácia de teste, ou seja, não há representatividade.

Finalmente, no OBG_MAG, o R-quadrado ajustado é de 5.97%, com coeficientes pequenos, mas estatisticamente significativos (comprimento da caminhada: 0.00003, $p < 0.0001$; tamanho do contexto: -0.00001, $p = 0.026$), sugerindo uma influência limitada.

Esses resultados reforçam a ideia de que a relação entre comprimento da caminhada, tamanho do contexto e acurácia de teste varia consideravelmente entre os diferentes *datasets*, com alguns evidenciando uma relação mais forte em detrimento de outros sendo influenciados por outros fatores não capturados pelos parâmetros analisados.

Tabela 5 – Principais métricas regressão linear múltipla

Métricas	AMiner	DBLP	HGBD_ACM	IMDB	MovieLens1M	OBG_MAG
Intercepto	2.669787e-01	2.522088e-01	3.312536e-01	3.382243e-01	2.582478e-01	1.368510e-02
Coef - Comprimento da Caminhada	1.383671e-03	7.228868e-04	1.138729e-03	8.248376e-05	3.065144e-05	2.989404e-05
Coef - Tamanho do Contexto	-1.271495e-03	-3.878618e-04	-5.475873e-04	1.112430e-05	-4.738333e-05	-1.160727e-05
p-valor - Intercepto	0	0	0	0	0	0
p-valor - Comprimento da Caminhada	1.860972e-66	7.838493e-30	3.434575e-41	1.435010e-01	5.975943e-01	1.739172e-08
p-valor - Tamanho do Contexto	4.037284e-59	1.470996e-10	3.174525e-12	8.425713e-01	4.119900e-01	2.568278e-02
R-quadrado	5.070533e-01	2.308113e-01	3.076642e-01	6.640663e-03	1.407175e-03	6.347377e-02
R-quadrado ajustado	5.050495e-01	2.276845e-01	3.048498e-01	2.602617e-03	-2.652146e-03	5.966675e-02

Fonte: Autoria própria.

4.4 Regressão polinomial

Este modelo emprega, Tabela 6, um polinômio de segundo grau para capturar as interações e os efeitos não lineares entre as variáveis. Inclui tanto os termos lineares quanto os quadráticos das variáveis independentes, além de suas interações cruzadas, proporcionando assim uma compreensão mais detalhada da dinâmica subjacente. As métricas chaves extraídas do modelo incluem os coeficientes para cada termo polinomial, seus respectivos p-valores, além do R-quadrado e R-quadrado ajustado, com o objetivo de esclarecer se há uma melhor representatividade deste aos dados.

A análise dos resultados da regressão polinomial revelou percepções distintas em comparação com a análise de regressão linear múltipla.

No modelo polinomial aplicado ao conjunto de dados AMiner, observa-se um R-quadrado ajustado de 74.1%, evidenciando uma forte capacidade de explicar a variação na acurácia de teste. A relação entre as variáveis independentes e a acurácia de teste é complexa, como indicado pelos coeficientes significativos para os termos polinomiais ($p < 0.0001$), sugerindo que a relação não é meramente linear.

Em relação ao DBLP, o modelo polinomial mostra um R-quadrado ajustado de 24.8%, que é significativamente mais alto do que o obtido na regressão linear múltipla. Isso indica que as relações não-lineares entre as variáveis são mais acentuadas neste conjunto de dados. Contudo, a falta de significância estatística de alguns coeficientes ($p > 0.05$) pode sugerir que nem todas as interações polinomiais têm a mesma relevância.

Para o conjunto de dados HGBD_ACM, o modelo polinomial registrou um R-quadrado ajustado de 39.0%, representando uma melhoria em relação a análise anterior. A maioria dos

coeficientes apresenta significância ($p < 0.05$), indicando a relevância das relações polinomiais na explicação da variação na acurácia de teste.

O conjunto de dados IMDB, por outro lado, apresenta um R-quadrado ajustado praticamente nulo 0.1%, sugerindo que o modelo polinomial não é eficaz para explicar a variação na acurácia de teste. Esta conclusão é reforçada pela falta de significância estatística dos coeficientes ($p > 0.05$).

No caso do MovieLens1M, o R-quadrado ajustado é de 1.5%, ligeiramente melhor do que o resultado da regressão linear múltipla, mas ainda considerado baixo. A significância estatística de alguns coeficientes ($p < 0.05$) indica que determinadas relações polinomiais podem ser relevantes.

Por fim, o modelo polinomial aplicado ao conjunto de dados OBG_MAG mostra um R-quadrado ajustado de 8.0%, revelando uma capacidade limitada de explicar a variação na acurácia de teste. A relevância de algumas interações polinomiais é indicada pela significância de certos coeficientes ($p < 0.05$).

Considerações sobre multicolinearidade e outros problemas numéricos: a nota sobre a alta condição numérica (5.47e+03) em todos os modelos sugere a presença de multicolinearidade forte ou outros problemas numéricos. Isso pode afetar a confiabilidade dos coeficientes estimados e suas interpretações.

Comparando com a análise de regressão linear múltipla, a regressão polinomial revela uma complexidade adicional nas relações entre as variáveis, especialmente para os conjuntos de dados AMiner, DBLP e HGBD_ACM. No entanto, a presença de multicolinearidade e a baixa capacidade explicativa em alguns conjuntos de dados (como IMDB e MovieLens1M) sugerem cautela na interpretação dos resultados. Essas análises indicam que, enquanto as relações lineares fornecem uma visão inicial, as interações mais complexas capturadas pela regressão polinomial podem oferecer percepções mais profundas, embora com limitações devido a problemas numéricos potenciais.

Tabela 6 – Principais métricas modelo polinomial

Métrica	AMiner	DBLP	HGBD_ACM	IMDB	MovieLens1M	OBG_MAG
p-valor - Termo 0	0	2.179620e-319	0	0	7.905050e-323	1.366402e-227
p-valor - Termo 1	6.751116e-22	2.081081e-01	7.766950e-03	4.970410e-01	2.964831e-03	7.113023e-01
p-valor - Termo 2	5.167230e-17	2.526130e-01	7.618117e-01	6.760646e-01	7.366947e-02	9.004774e-01
p-valor - Termo 3	8.461556e-64	9.823748e-01	1.417499e-01	9.256730e-01	1.041478e-03	6.246845e-01
p-valor - Termo 4	1.721147e-56	9.450477e-03	6.448676e-07	2.292417e-01	1.793669e-02	2.950784e-02

p-valor - Termo 5	1.366485e-21	1.449831e-04	2.892897e-16	2.051762e-01	1.377278e-01	6.202740e-04
R-quadrado	7.440429e-01	2.553128e-01	3.963474e-01	1.123960e-02	2.500664e-02	8.967599e-02
R-quadrado ajustado	7.414257e-01	2.476984e-01	3.901751e-01	1.129579e-03	1.503738e-02	8.036797e-02
Condição numérica	5.47e+03	5.47e+03	5.47e+03	5.47e+03	5.47e+03	5.47e+03

Fonte: autoria própria.

4.5 ANOVA

A Tabela 7 e Figura 14 apresentam as relações entre as variáveis independentes comprimento da caminhada e tamanho do contexto e a acurácia de teste em cada conjunto de dados. Esta técnica, com seus p-valores significativos, é fundamental para validar as relações identificadas nas outras técnicas aplicadas, oferecendo uma compreensão mais ampla das dinâmicas entre as variáveis estudadas.

Para o conjunto de dados AMiner, observa-se uma tendência clara de que tanto o comprimento da caminhada quanto o tamanho do contexto têm um impacto significativo na acurácia de teste, evidenciado pela dispersão dos pontos e uma distribuição clara e distinta entre as variáveis independentes e a acurácia de teste. Os p-valores extremamente baixos ($p < 0.0001$ para comprimento da caminhada e tamanho do contexto) indicam uma forte relação estatística entre as variáveis.

No caso de DBLP, a variação na acurácia de teste em relação às variáveis independentes é mais moderada. A distribuição é menos pronunciada em comparação com AMiner, com uma superposição mais evidente, sugerindo uma relação existente, porém não pronunciada. Os p-valores ($p < 0.0001$ para comprimento da caminhada e $p < 0.05$ para tamanho do contexto) ainda indicam significância estatística, embora inferior em relação ao AMiner.

Similarmente, HGBD_ACM mostra uma relação positiva entre as variáveis independentes e a acurácia de teste, com p-valores indicando significância estatística ($p < 0.0001$ para comprimento da caminhada e $p < 0.05$ para tamanho do contexto).

Em contraste, IMDB apresenta uma superposição considerável e uma distribuição menos clara, indicando uma relação muito fraca ou inexistente entre as variáveis independentes e a acurácia de teste. Os p-valores mais altos ($p > 0.05$ para ambas as variáveis independentes) reforçam a falta de significância estatística nesta relação.

Para MovieLens1M, os pontos estão distribuídos de maneira que sugere uma relação insignificante entre as variáveis independentes e a acurácia de teste, com p-valores também reforçando uma ausência de representatividade ($p > 0.05$ para ambas as variáveis).

Finalmente, OBG_MAG apresenta pontuais variações na acurácia de teste em relação parâmetros independentes, mas com p-valores indicando uma relação estatisticamente significativa ($p < 0.05$ para comprimento da caminhada e para tamanho do contexto), embora menos pronunciada do que em conjuntos de dados como AMiner e HGBD_ACM.

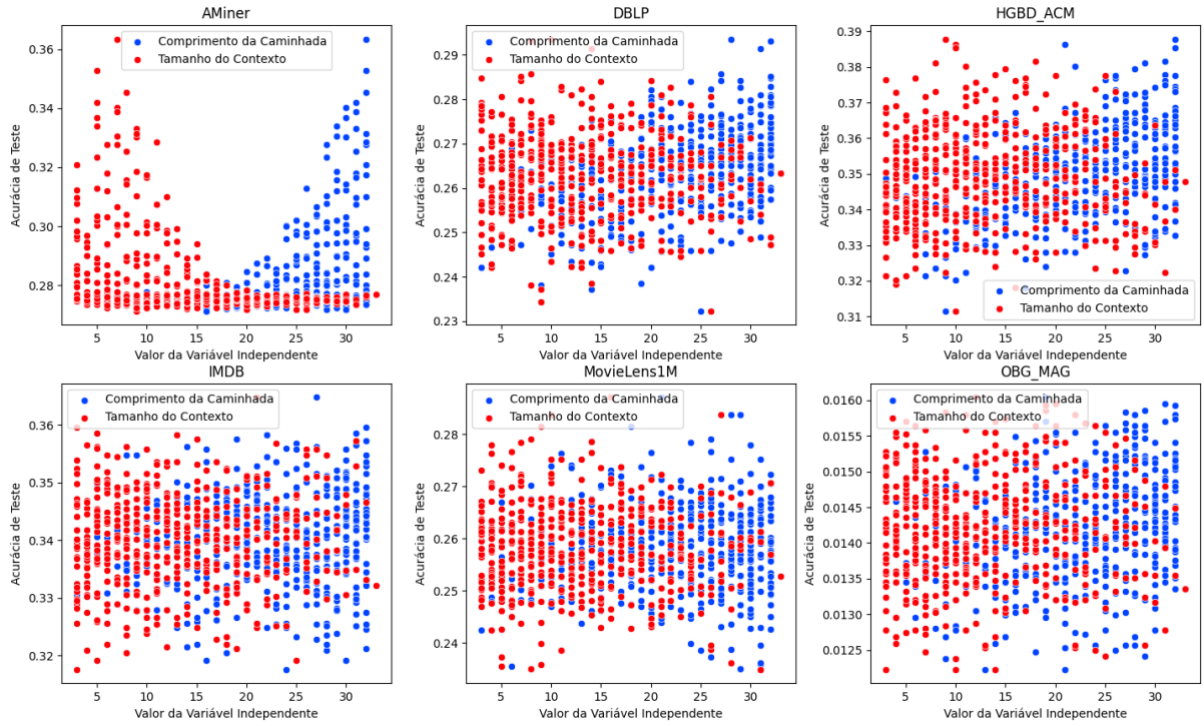
A geometria dos gráficos, com a superposição e a distribuição das variáveis independentes em relação à acurácia de teste, fornece um panorama das relações identificadas pelas técnicas estatísticas. A superposição indica áreas onde as variáveis independentes têm efeitos semelhantes na variável dependente, enquanto uma clara separação sugere uma influência mais distinta. Essas observações visuais, juntamente com os p-valores da ANOVA, complementam as análises estatísticas, confirmando a significância estatística das influências observadas.

Tabela 7 – Métricas ANOVA

Dataset	**Métrica**	**Soma dos Quadrados**	**df**	**Estatística F**	**Valor-p da Estatística F**
AMiner	Comprimento da caminhada	3.873835e-02	1	4.074665e+02	1.860972e-66
AMiner	Tamanho do contexto	3.307770e-02	1	3.479253e+02	4.037284e-59
AMiner	Residual	4.677505e-02	492	-	-
DBLP	Comprimento da caminhada	1.057344e-02	1	1.472847e+02	7.838493e-30
DBLP	Tamanho do contexto	3.077936e-03	1	4.287469e+01	1.470996e-10
DBLP	Residual	3.532024e-02	492	-	-
HGBD_ACM	Comprimento da caminhada	2.623713e-02	1	2.185835e+02	3.434575e-41
HGBD_ACM	Tamanho do contexto	6.134968e-03	1	5.111089e+01	3.174525e-12
HGBD_ACM	Residual	5.905599e-02	492	-	-
IMDB	Comprimento da caminhada	1.376615e-04	1	2.146855e+00	1.435010e-01
IMDB	Tamanho do contexto	2.531927e-06	1	3.948584e-02	8.425713e-01
IMDB	Residual	3.154822e-02	492	-	-
MovieLens1M	Comprimento da caminhada	1.900979e-05	1	2.790019e-01	5.975943e-01
MovieLens1M	Tamanho do contexto	4.593637e-05	1	6.741966e-01	4.119900e-01
MovieLens1M	Residual	3.352241e-02	492	-	-
OBG_MAG	Comprimento da caminhada	1.808193e-05	1	3.284811e+01	1.739172e-08
OBG_MAG	Tamanho do contexto	2.756549e-06	1	5.007619e+00	2.568278e-02
OBG_MAG	Residual	2.708317e-04	492	-	-

Fonte: Autoria própria.

Figura 14 – Distribuição ANOVA – Acurácia de teste x variável independente



Fonte: autoria própria.

5. CONCLUSÃO E TRABALHOS FUTUROS

O estudo realizado sobre as variáveis comprimento da caminhada e tamanho do contexto no contexto do modelo MetaPath2Vec++ revelaram um panorama complexo e randômico em relação à acurácia de teste nos diferentes *datasets* analisados. Em conjuntos de dados como AMiner e DBLP, foi identificada uma correlação forte e estatisticamente significativa, sugerindo um alinhamento efetivo entre as características do modelo e as peculiaridades destes *datasets*. Todavia, em *datasets* como MovieLens1M e IMDB, as relações identificadas foram mais tênues sugerindo que as variáveis em questão podem não ser satisfatoriamente explicadas pelas análises estatísticas implementadas neste trabalho.

Este cenário aponta para a importância do contexto específico de cada conjunto de dados na determinação da eficácia do modelo MetaPath2Vec++. As técnicas utilizadas, como regressão linear múltipla, regressão polinomial e a ANOVA, embora tenham proporcionado respostas relevantes, demonstraram não capturar integralmente a complexidade e a singularidade inerentes a cada *dataset*. Isso sugere a necessidade de uma abordagem mais refinada e adaptativa, que pondere as características únicas de cada conjunto de dados com o intuito de apresentar métricas plenamente representativas.

Para futuras investigações, é imperativo adotar técnicas estatísticas mais avançadas complementado pela exploração de uma extensão mais ampla de parâmetros, além de métodos de modelagem mais sofisticados. Estas abordagens podem ser mais eficazes na captura de relações não-lineares e interações complexas entre variáveis, oferecendo uma compreensão mais profunda e matizada das dinâmicas em questão.

É também crucial ampliar a variação das faixas das variáveis independentes e aumentar o número de épocas de treinamento para cada *dataset*. Esta abordagem permitirá que o modelo MetaPath2Vec++ atinja uma convergência assintótica, proporcionando uma compreensão mais acurada e abrangente do impacto dessas variáveis na acurácia de teste. Ou seja, experimentações com amplitudes maiores do comprimento de caminhada e tamanho de contexto podem revelar padrões e percepções adicionais, especialmente em relação a interação destas variáveis com as características específicas de cada *dataset*.

Finalmente, a singularidade de cada conjunto de dados deve ser ponderada. Cada *dataset* possui características únicas que podem influenciar significativamente a relação entre as variáveis estudadas e a eficácia do modelo. Portanto, uma abordagem personalizada e

adaptativa é recomendada, incluindo a exploração de diferentes configurações de meta-caminhos, especialmente nos cenários menos representativos.

Em resumo, o estudo atual, embora tenha fornecido resultados relativamente expressivos, destaca-se a necessidade de um aprofundamento e refinamento nas análises. Adotando abordagens analíticas mais sofisticadas e métodos de avaliação de desempenho avançados, como validação cruzada para avaliar robustez do modelo e algoritmos como Gradient Boosting para aprimorar e validar as previsões dos resultados. Futuras pesquisas podem alcançar uma compreensão mais robusta e detalhada das relações entre as variáveis em diferentes contextos de dados. A implementação de otimização de hiperparâmetros, como busca em grade ou otimização bayesiana, e a realização de análises de sensibilidade e exploração de outras variáveis, incluindo a experimentação com modelos ensemble, podem contribuir significativamente para a compreensão da influência destas variáveis independentes na acurácia de teste de forma generalizada.

REFERÊNCIAS

[1] DONG, Y.; CHAWLA, N. V.; SWAMI, A. Metapath2vec: Scalable representation learning for heterogeneous networks. In: Proceedings Of The 23rd Acm Sigkdd International Conference On Knowledge Discovery And Data Mining. **Proceedings [...]**. Halifax, NS, CA. ACM, p. 135–144, 2017.

[2] SUN, Y.; HAN, J. **Mining heterogeneous information networks: Principles and methodologies**. Morgan & Claypool Publishers. São Rafael, CA, USA, 2012.

[3] SUN, Y.; NORICK, B.; HAN, J.; YAN, X.; YU, P. S.; YU, X. Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. In: Proceedings Of The 18th Acm Sigkdd International Conference On Knowledge Discovery And Data Mining. **Proceedings [...]**. Beijing, China: ACM, p. 1348–1356, 2012.

[4] NEVILLE, J.; JENSEN, D. Leveraging relational autocorrelation with latent group models. In: Proceedings Of The 4th International Workshop On Multi-Relational Mining - Mrdm '05. **Proceedings [...]**. Chicago, IL, USA: ACM, p. 49–55, 2005.

[5] MIKOLOV, T.; SUTSKEVER, I.; CHEN, K.; CORRADO, G.; DEAN, J. **Distributed representations of words and phrases and their compositionality**. In: NIPS '13. p. 3111-3119, 2013.

[6] BENGIO, Y.; COURVILLE, A.; VINCENT, P. **Representation learning: A review and new perspectives**. IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 35, n. 8, p. 1798-1828, 2012.

[7] GOLDBERG, Y.; LEVY, O. **Word2Vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method**. [s. l.], 2014. Disponível em: <http://arxiv.org/abs/1402.3722>.

[8] RONG, X. **word2vec Parameter Learning Explained**. [s. l.], , 2014. Disponível em: <http://arxiv.org/abs/1411.2738>.

[9] SUN, Y.; HAN, J.; YAN, X.; YU, P. S.; WU, T. PathSim: Meta path-based top-K similarity search in heterogeneous information networks. In: Proceedings of the VLDB Endowment Volume 4 Issue 11. Proceedings [...]. Seatle, USA: VLDB, p. 992–1003, 2011.

[10] SUN, Y.; YU, Y.; HAN, J. Ranking-based clustering of heterogeneous information networks with star network schema. In: Proceedings Of The 15th Acm Sigkdd International Conference On Knowledge Discovery And Data Mining. **Proceedings [...]**. Paris, France. ACM, p. 797-805, 2009.

[11] TANG, J.; QU, M.; MEI, Q. PTE: Predictive text embedding through large-scale heterogeneous text networks. In: Proceedings Of The 21th Acm Sigkdd International

Conference On Knowledge Discovery And Data Mining. **Proceedings [...]**. Sydney, NSW, Australia: ACM, p. 1165–1174, 2015.

APÊNDICE A – Código completo *dataset* AMiner

```
# Instalar a biblioteca pytorch_geometric diretamente do repositório do
GitHub.
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
# Instalar as dependências torch-scatter e torch_sparse da biblioteca
pytorch_geometric.
!pip install -q torch-scatter torch_sparse -f
https://data.pyg.org/whl/torch-2.1.0+cu118.html
# Importar módulos necessários para o funcionamento do script.
import os
import os.path as osp
import numpy as np
import torch
import pandas as pd
from torch_geometric.datasets import AMiner
from torch_geometric.nn import MetaPath2Vec
import time
# Importar módulos do Google Colab para upload de arquivos e montagem
do Google Drive.
from google.colab import files
from google.colab import drive
# Montar o Google Drive para acesso aos arquivos e armazenamento.
drive.mount('/content/drive/')
# Link Dataset: https://pytorch-
geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.AM
iner.html#torch_geometric.datasets.AMiner
path = osp.join('AMiner_folder', 'AMiner')
dataset = AMiner(path)
data = dataset[0]
print(data)
# Definir o dispositivo de processamento (GPU ou CPU).
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
# Definir a metapath para análise no grafo.
metapath = [
    ('author', 'writes', 'paper'),
    ('paper', 'published_in', 'venue'),
    ('venue', 'publishes', 'paper'),
    ('paper', 'written_by', 'author'),
]
# Obter os valores únicos para a classe 'author' e imprimir seu
comprimento.
k = data['author'].y.cpu().numpy()
unique_values = len(np.unique(k, axis=0))
print(unique_values)
# Imprimir o índice da classe 'author'.
```

```
print(data.y_index_dict['author'])
# Imprimir o dicionário de índices das arestas do grafo.
print(data.edge_index_dict)
# Imprimir o índice 'y_index' para a classe 'author'.
print(data['author'].y_index)
# Imprimir o dicionário 'y_dict' para a classe 'author'.
print(data.y_dict['author'])
# Imprimir o tipo de dado do número de nós para a classe 'author'.
print(type(data.num_nodes_dict['author']))
# Imprimir o tipo do dicionário 'y_dict' e o valor para a classe
'author'.
print(type(data.y_dict))
print(data.y_dict["author"])
# Imprimir o tipo do dicionário 'y_index_dict' e o valor para a classe
'author'.
print(type(data.y_index_dict))
print(data.y_index_dict["author"])
# Criar um DataFrame vazio com colunas pré-definidas para armazenar
resultados.
results_df = pd.DataFrame(columns=['Walk_Length_P', 'Context_Size_P',
'Train_Loss', 'Train_Acc', 'Test_Acc','Time_Epoch'])
# Iniciar a medição do tempo total do processo.
tempo_total_inicial = time.time()

# Iterar sobre os valores de walk_length e context_size.
for walk_length in range(3, 33):
    for context_size in range(3, 34):
        # Verificar a condição de continuação.
        if not (walk_length + 1 >= context_size):
            continue

        # Iniciar a medição do tempo de uma época.
        tempo_inicio = time.time()

        # Inicializar o modelo MetaPath2Vec com os parâmetros
definidos.
        model = MetaPath2Vec(data.edge_index_dict,
                             embedding_dim=128,
                             metapath=metapath,
                             walk_length=walk_length,
                             context_size=context_size,
                             walks_per_node=3,
                             num_negative_samples=1,
                             sparse=True
                            ).to(device)

        # Criar um carregador de dados para o modelo.
        loader = model.loader(batch_size=128, shuffle=True,
num_workers=3)
```

```
        # Inicializar o otimizador SparseAdam para o modelo.
        optimizer = torch.optim.SparseAdam(list(model.parameters()),
lr=0.01)

        # Definir a função de treino do modelo.
        def train(epoch, log_steps=500, eval_steps=1000):
            model.train()
            total_loss = 0
            for i, (pos_rw, neg_rw) in enumerate(loader):
                optimizer.zero_grad()
                loss = model.loss(pos_rw.to(device), neg_rw.to(device))
                loss.backward()
                optimizer.step()
                total_loss += loss.item()

                # Imprimir logs a cada número definido de passos.
                if (i + 1) % log_steps == 0:
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Loss: {total_loss / log_steps:.4f}'))
                    total_loss = 0

                # Avaliar e imprimir o desempenho a cada número
definido de passos.
                if (i + 1) % eval_steps == 0:
                    acc = test()
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Acc: {acc:.4f}'))
                    print("Tempo_parcial", f"{time.time() -
tempo_inicio:.4f}")

            return total_loss / len(loader)

        # Definir a função de teste do modelo.
        @torch.no_grad()
        def test(train_ratio=0.1):
            model.eval()
            z = model('author',
batch=data.y_index_dict['author'].to(device))
            y = data.y_dict['author']
            perm = torch.randperm(z.size(0))
            train_perm = perm[:int(z.size(0) * train_ratio)]
            test_perm = perm[int(z.size(0) * train_ratio):]

            return model.test(z[train_perm], y[train_perm],
z[test_perm], y[test_perm], max_iter=150)
```

```
        # Treinar e testar o modelo para uma época.
        train_loss = train(epoch=1)
        train_acc = test(train_ratio=0.9)
        test_acc = test(train_ratio=0.1)

        # Calcular e armazenar o tempo de duração de uma época.
        tempo_epoca = f"{time.time() - tempo_inicio:.6f}"

        # Armazenar os resultados no DataFrame.
        context_size_p = context_size
        walk_length_p = walk_length
        new_row = pd.DataFrame({'Walk_Length_P': [walk_length_p],
'Context_Size_P': [context_size_p],'Train_Loss': [train_loss],
'Train_Acc': [train_acc], 'Test_Acc': [test_acc], 'Time_Epoch':
[tempo_epoca]})
        results_df = pd.concat([results_df, new_row],
ignore_index=True)

        # Imprimir o DataFrame atualizado.
        print(results_df)

        # Salvar o DataFrame em um arquivo Excel.
        results_df.to_excel("AMiner - MetaPath2Vec.xlsx")
        results_df.to_excel("/content/drive/MyDrive/Colab
Notebooks/FINAIS/data_AMiner/AMiner - MetaPath2Vec.xlsx")

# Calcular e imprimir o tempo total de execução do processo.
tempo_total_final = time.time() - tempo_total_inicial
print("TEMPO TOTAL:", format(tempo_total_final, '.6f'), "SEGUNDOS")
```

APÊNDICE B – Código completo *dataset* DBLP

```
# Instalar a biblioteca pytorch_geometric diretamente do repositório do
GitHub.
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
# Instalar as dependências torch-scatter e torch_sparse da biblioteca
pytorch_geometric.
!pip install -q torch-scatter torch_sparse -f
https://data.pyg.org/whl/torch-2.1.0+cu118.html
# Importar módulos necessários para o funcionamento do script.
import os
import os.path as osp
import numpy as np
import torch
import pandas as pd
from torch_geometric.datasets import DBLP
from torch_geometric.nn import MetaPath2Vec
import time
# Importar módulos do Google Colab para upload de arquivos e montagem
do Google Drive.
from google.colab import files
from google.colab import drive
# Montar o Google Drive para acesso aos arquivos e armazenamento.
drive.mount('/content/drive/')
# Link Dataset: https://pytorch-
geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.DB
LP.html#torch_geometric.datasets.DBLP
path = osp.join('DBLP_folder', 'DBLP')
dataset = DBLP(path)
data = dataset[0]
print(data)
# Definir o dispositivo de processamento (GPU ou CPU).
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
# Definir a metapath para análise no grafo.
metapath = [
    ('author', 'to', 'paper'),
    ('paper', 'to', 'conference'),
    ('conference', 'to', 'paper'),
    ('paper', 'to', 'author')
]
# Obter a representação bag-of-words dos autores com base nas palavras-
chave dos seus artigos.
k = data['author'].x.cpu().numpy()
# Calcular e imprimir o número de valores únicos nessa representação.
unique_values = len(np.unique(k, axis=0))
```

```
print(unique_values)
# Obter as áreas de pesquisa dos autores (database, data mining,
artificial intelligence, information retrieval).
k = data['author'].y.cpu().numpy()
# Calcular e imprimir o número de áreas de pesquisa únicas.
unique_values = len(np.unique(k, axis=0))
print(unique_values)
# Imprimir a forma do vetor 'y' para a classe 'author' e o vetor em si.
print(data['author'].y.shape)
print(data['author'].y)
# Imprimir a forma do vetor 'x' para a classe 'author' e o vetor em si.
print(data['author'].x.shape)
print(data['author'].x)

# Imprimir o dicionário de índices das arestas do grafo.
print(data.edge_index_dict)
# Criar um DataFrame vazio com colunas pré-definidas para armazenar
resultados.
results_df = pd.DataFrame(columns=['Walk_Length_P', 'Context_Size_P',
'Train_Loss', 'Train_Acc', 'Test_Acc','Time_Epoch'])
# Iniciar a medição do tempo total do processo.
tempo_total_inicial = time.time()

# Iterar sobre os valores de walk_length e context_size.
for walk_length in range(3, 33):
    for context_size in range(3, 34):
        # Verificar a condição de continuação.
        if not (walk_length + 1 >= context_size):
            continue

        # Iniciar a medição do tempo de uma época.
        tempo_inicio = time.time()

        # Inicializar o modelo MetaPath2Vec com os parâmetros
definidos.
        model = MetaPath2Vec(data.edge_index_dict,
                             embedding_dim=128,
                             metapath=metapath,
                             walk_length=walk_length,
                             context_size=context_size,
                             walks_per_node=3,
                             num_negative_samples=1,
                             sparse=True
                            ).to(device)

        # Criar um carregador de dados para o modelo.
        loader = model.loader(batch_size=128, shuffle=True,
num_workers=3)
```

```
        # Inicializar o otimizador SparseAdam para o modelo.
        optimizer = torch.optim.SparseAdam(list(model.parameters()),
lr=0.01)

        # Definir a função de treino do modelo.
        def train(epoch, log_steps=500, eval_steps=1000):
            model.train()
            total_loss = 0
            for i, (pos_rw, neg_rw) in enumerate(loader):
                optimizer.zero_grad()
                loss = model.loss(pos_rw.to(device), neg_rw.to(device))
                loss.backward()
                optimizer.step()
                total_loss += loss.item()

                # Imprimir logs a cada número definido de passos.
                if (i + 1) % log_steps == 0:
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Loss: {total_loss / log_steps:.4f}'))
                    total_loss = 0

                # Avaliar e imprimir o desempenho a cada número
definido de passos.
                if (i + 1) % eval_steps == 0:
                    acc = test()
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Acc: {acc:.4f}'))
                    print("Tempo_parcial", f"{time.time() -
tempo_inicio:.4f}")

            return total_loss / len(loader)

        # Definir a função de teste do modelo com processamento de nós
individuais.
        @torch.no_grad()
        def test(train_ratio=0.1):
            model.eval()
            zs = []
            # Obter a representação vetorial para cada nó do tipo
'author'.
            for idx, _ in enumerate(data['author'].x):  # idx é o
índice do nó
                z = model('author',
batch=torch.tensor([idx]).to(device))
                zs.append(z)
            z = torch.cat(zs)
            y = data['author'].y
```

```
            perm = torch.randperm(z.size(0))
            train_perm = perm[:int(z.size(0) * train_ratio)]
            test_perm = perm[int(z.size(0) * train_ratio):]

            return model.test(z[train_perm], y[train_perm],
z[test_perm], y[test_perm], max_iter=150)

        # Treinar e testar o modelo para uma época.
        train_loss = train(epoch=1)
        train_acc = test(train_ratio=0.9)
        test_acc = test(train_ratio=0.1)

        # Calcular e armazenar o tempo de duração de uma época.
        tempo_epoca = f"{time.time() - tempo_inicio:.6f}"

        # Armazenar os resultados no DataFrame.
        context_size_p = context_size
        walk_length_p = walk_length
        new_row = pd.DataFrame({'Walk_Length_P': [walk_length_p],
'Context_Size_P': [context_size_p],'Train_Loss': [train_loss],
'Train_Acc': [train_acc], 'Test_Acc': [test_acc], 'Time_Epoch':
[tempo_epoca]})
        results_df = pd.concat([results_df, new_row],
ignore_index=True)

        # Imprimir o DataFrame atualizado.
        print(results_df)

        # Salvar o DataFrame em um arquivo Excel.
        results_df.to_excel("DBLP-MetaPath2Vec.xlsx")
        results_df.to_excel("/content/drive/MyDrive/Colab
Notebooks/FINAIS/data_DBLP/DBLP-MetaPath2Vec.xlsx")

# Calcular e imprimir o tempo total de execução do processo.
tempo_total_final = time.time() - tempo_total_inicial
print("TEMPO TOTAL:", format(tempo_total_final, '.6f'), "SEGUNDOS")
```

APÊNDICE C – Código completo *dataset* HGBD_ACM

```
# Instalar a biblioteca pytorch_geometric diretamente do repositório do
GitHub.
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
# Instalar as dependências torch-scatter e torch_sparse da biblioteca
pytorch_geometric.
!pip install -q torch-scatter torch_sparse -f
https://data.pyg.org/whl/torch-2.1.0+cu118.html
# Importar módulos necessários para o funcionamento do script.
import os
import os.path as osp
import numpy as np
import torch
import pandas as pd
from torch_geometric.datasets import HGBDataset
from torch_geometric.nn import MetaPath2Vec
import time
# Importar módulos do Google Colab para upload de arquivos e montagem
do Google Drive.
from google.colab import files
from google.colab import drive
# Montar o Google Drive para acesso aos arquivos e armazenamento.
drive.mount('/content/drive/')
# Link Dataset: https://pytorch-
geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.HG
BDataset.html#torch_geometric.datasets.HGBDataset
path = osp.join('HGBDataset_ACM_folder', 'ACM')
dataset = HGBDataset(path, "ACM")
data = dataset[0]
print(data)
# Definir o dispositivo de processamento (GPU ou CPU).
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
# Definir a metapath para a análise no grafo.
metapath = [
    ('paper', 'to', 'author'),
    ('author', 'to', 'paper'),
    ('paper', 'to', 'subject'),
    ('subject', 'to', 'paper')
]
# Imprimir os tipos de nós presentes no dataset.
print(data.node_types)
# Imprimir a representação dos nós do tipo 'paper'.
print(data['paper'].x)
# Imprimir os rótulos (labels) dos nós do tipo 'paper'.
```

```
print(data['paper'].y)
# Obter os rótulos dos nós do tipo 'paper' e calcular o número de
valores únicos.
k = data['paper'].y.cpu().numpy()
unique_values = len(np.unique(k, axis=0))
print(unique_values)
# Imprimir o dicionário de índices das arestas do grafo.
print(data.edge_index_dict)
# Criar um DataFrame vazio com colunas pré-definidas para armazenar
resultados.
results_df = pd.DataFrame(columns=['Walk_Length_P', 'Context_Size_P',
'Train_Loss', 'Train_Acc', 'Test_Acc','Time_Epoch'])
# Iniciar a medição do tempo total do processo.
tempo_total_inicial = time.time()

# Iterar sobre os valores de walk_length e context_size.
for walk_length in range(3, 33):
    for context_size in range(3, 34):
        # Verificar a condição de continuação.
        if not (walk_length + 1 >= context_size):
            continue

        # Iniciar a medição do tempo de uma época.
        tempo_inicio = time.time()

        # Inicializar o modelo MetaPath2Vec com os parâmetros
definidos.
        model = MetaPath2Vec(data.edge_index_dict,
                             embedding_dim=128,
                             metapath=metapath,
                             walk_length=walk_length,
                             context_size=context_size,
                             walks_per_node=3,
                             num_negative_samples=1,
                             sparse=True
                            ).to(device)

        # Criar um carregador de dados para o modelo.
        loader = model.loader(batch_size=128, shuffle=True,
num_workers=3)

        # Inicializar o otimizador SparseAdam para o modelo.
        optimizer = torch.optim.SparseAdam(list(model.parameters()),
lr=0.01)

        # Definir a função de treino do modelo.
        def train(epoch, log_steps=500, eval_steps=1000):
            model.train()
            total_loss = 0
```

```
            for i, (pos_rw, neg_rw) in enumerate(loader):
                optimizer.zero_grad()
                loss = model.loss(pos_rw.to(device), neg_rw.to(device))
                loss.backward()
                optimizer.step()
                total_loss += loss.item()

                # Imprimir logs a cada número definido de passos.
                if (i + 1) % log_steps == 0:
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Loss: {total_loss / log_steps:.4f}'))
                    total_loss = 0

                # Avaliar e imprimir o desempenho a cada número
definido de passos.
                if (i + 1) % eval_steps == 0:
                    acc = test()
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Acc: {acc:.4f}'))
                    print("Tempo_parcial", f"{time.time() -
tempo_inicio:.4f}")

            return total_loss / len(loader)

        # Definir a função de teste do modelo com processamento de nós
individuais.
        @torch.no_grad()
        def test(train_ratio=0.1):
            model.eval()
            zs = []
            # Obter a representação vetorial para cada nó do tipo
'paper'.
            for idx, _ in enumerate(data['paper'].x):  # idx é o índice
do nó
                z = model('paper',
batch=torch.tensor([idx]).to(device))
                zs.append(z)
            z = torch.cat(zs)
            y = data['paper'].y
            perm = torch.randperm(z.size(0))
            train_perm = perm[:int(z.size(0) * train_ratio)]
            test_perm = perm[int(z.size(0) * train_ratio):]

            return model.test(z[train_perm], y[train_perm],
z[test_perm], y[test_perm], max_iter=150)

        # Treinar e testar o modelo para uma época.
```

```
        train_loss = train(epoch=1)
        train_acc = test(train_ratio=0.9)
        test_acc = test(train_ratio=0.1)

        # Calcular e armazenar o tempo de duração de uma época.
        tempo_epoca = f"{time.time() - tempo_inicio:.6f}"

        # Armazenar os resultados no DataFrame.
        context_size_p = context_size
        walk_length_p = walk_length
        new_row = pd.DataFrame({'Walk_Length_P': [walk_length_p],
'Context_Size_P': [context_size_p],'Train_Loss': [train_loss],
'Train_Acc': [train_acc], 'Test_Acc': [test_acc], 'Time_Epoch':
[tempo_epoca]})
        results_df = pd.concat([results_df, new_row],
ignore_index=True)

        # Imprimir o DataFrame atualizado.
        print(results_df)

        # Salvar o DataFrame em um arquivo Excel.
        results_df.to_excel("HGBD-ACM-MetaPath2Vec.xlsx")
        results_df.to_excel("/content/drive/MyDrive/Colab
Notebooks/FINAIS/data_HGBDataset/HGBD-ACM-MetaPath2Vec.xlsx")

# Calcular e imprimir o tempo total de execução do processo.
tempo_total_final = time.time() - tempo_total_inicial
print("TEMPO TOTAL:", format(tempo_total_final, '.6f'), "SEGUNDOS")
```

APÊNDICE D – Código completo *dataset* IMDB

```
# Instalar a biblioteca pytorch_geometric diretamente do repositório do
GitHub.
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
# Instalar as dependências torch-scatter e torch_sparse da biblioteca
pytorch_geometric.
!pip install -q torch-scatter torch_sparse -f
https://data.pyg.org/whl/torch-2.1.0+cu118.html
# Importar módulos necessários para o funcionamento do script.
import os
import os.path as osp
import numpy as np
import torch
import pandas as pd
from torch_geometric.datasets import IMDB
from torch_geometric.nn import MetaPath2Vec
import time
# Importar módulos do Google Colab para upload de arquivos e montagem
do Google Drive.
from google.colab import files
from google.colab import drive
# Montar o Google Drive para acesso aos arquivos e armazenamento.
drive.mount('/content/drive/')
# Link Dataset: https://pytorch-
geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.IM
DB.html#torch_geometric.datasets.IMDB
path = osp.join('IMDB_folder', 'IMDB')
dataset = IMDB(path)
data = dataset[0]
print(data)
# Definir o dispositivo de processamento (GPU ou CPU).
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
# Definir a metapath para a análise no grafo.
metapath = [
    ('movie', 'to', 'director'),
    ('director', 'to', 'movie'),
    ('movie', 'to', 'actor'),
    ('actor', 'to', 'movie')
]
# Obter os rótulos dos nós do tipo 'movie' e calcular o número de
valores únicos.
k = data['movie'].y.cpu().numpy()
unique_values = len(np.unique(k, axis=0))
print(unique_values)
```

```
# Obter a representação dos nós do tipo 'movie' e calcular o número de
valores únicos.
k = data['movie'].x.cpu().numpy()
unique_values = len(np.unique(k, axis=0))
print(unique_values)
# Imprimir a representação dos nós do tipo 'movie'.
print(data['movie'].x)
# Imprimir os rótulos (labels) dos nós do tipo 'movie'.
print(data['movie'].y)
# Imprimir o dicionário de índices das arestas do grafo.
print(data.edge_index_dict)
# Criar um DataFrame vazio com colunas pré-definidas para armazenar
resultados.
results_df = pd.DataFrame(columns=['Walk_Length_P', 'Context_Size_P',
'Train_Loss', 'Train_Acc', 'Test_Acc','Time_Epoch'])
# Iniciar a medição do tempo total do processo.
tempo_total_inicial = time.time()

# Iterar sobre os valores de walk_length e context_size.
for walk_length in range(3, 33):
    for context_size in range(3, 34):
        # Verificar a condição de continuação.
        if not (walk_length + 1 >= context_size):
            continue

        # Iniciar a medição do tempo de uma época.
        tempo_inicio = time.time()

        # Inicializar o modelo MetaPath2Vec com os parâmetros
definidos.
        model = MetaPath2Vec(data.edge_index_dict,
                             embedding_dim=128,
                             metapath=metapath,
                             walk_length=walk_length,
                             context_size=context_size,
                             walks_per_node=3,
                             num_negative_samples=1,
                             sparse=True
                            ).to(device)

        # Criar um carregador de dados para o modelo.
        loader = model.loader(batch_size=128, shuffle=True,
num_workers=3)

        # Inicializar o otimizador SparseAdam para o modelo.
        optimizer = torch.optim.SparseAdam(list(model.parameters()),
lr=0.01)

        # Definir a função de treino do modelo.
```

```
        def train(epoch, log_steps=500, eval_steps=1000):
            model.train()
            total_loss = 0
            for i, (pos_rw, neg_rw) in enumerate(loader):
                optimizer.zero_grad()
                loss = model.loss(pos_rw.to(device), neg_rw.to(device))
                loss.backward()
                optimizer.step()
                total_loss += loss.item()

                # Imprimir logs a cada número definido de passos.
                if (i + 1) % log_steps == 0:
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Loss: {total_loss / log_steps:.4f}'))
                    total_loss = 0

                # Avaliar e imprimir o desempenho a cada número
definido de passos.
                if (i + 1) % eval_steps == 0:
                    acc = test()
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Acc: {acc:.4f}'))
                    print("Tempo_parcial", f"{time.time() -
tempo_inicio:.4f}")

            return total_loss / len(loader)

        # Definir a função de teste do modelo com processamento de nós
individuais.
        @torch.no_grad()
        def test(train_ratio=0.1):
            model.eval()
            zs = []
            # Obter a representação vetorial para cada nó do tipo
'movie'.
            for idx, _ in enumerate(data['movie'].x):  # idx é o índice
do nó
                z = model('movie',
batch=torch.tensor([idx]).to(device))
                zs.append(z)
            z = torch.cat(zs)
            y = data['movie'].y
            perm = torch.randperm(z.size(0))
            train_perm = perm[:int(z.size(0) * train_ratio)]
            test_perm = perm[int(z.size(0) * train_ratio):]
```

```
            return model.test(z[train_perm], y[train_perm],
z[test_perm], y[test_perm], max_iter=150)

        # Treinar e testar o modelo para uma época.
        train_loss = train(epoch=1)
        train_acc = test(train_ratio=0.9)
        test_acc = test(train_ratio=0.1)

        # Calcular e armazenar o tempo de duração de uma época.
        tempo_epoca = f"{time.time() - tempo_inicio:.6f}"

        # Armazenar os resultados no DataFrame.
        context_size_p = context_size
        walk_length_p = walk_length
        new_row = pd.DataFrame({'Walk_Length_P': [walk_length_p],
'Context_Size_P': [context_size_p],'Train_Loss': [train_loss],
'Train_Acc': [train_acc], 'Test_Acc': [test_acc], 'Time_Epoch':
[tempo_epoca]})
        results_df = pd.concat([results_df, new_row],
ignore_index=True)

        # Imprimir o DataFrame atualizado.
        print(results_df)

        # Salvar o DataFrame em um arquivo Excel.
        results_df.to_excel("IMDB-MetaPath2Vec.xlsx")
        results_df.to_excel("/content/drive/MyDrive/Colab
Notebooks/FINAIS/data_IMDB/IMDB-MetaPath2Vec.xlsx")

# Calcular e imprimir o tempo total de execução do processo.
tempo_total_final = time.time() - tempo_total_inicial
print("TEMPO TOTAL:", format(tempo_total_final, '.6f'), "SEGUNDOS")
```

APÊNDICE E – Código completo *dataset* MovieLens1M

```
# Instalar a biblioteca pytorch_geometric diretamente do repositório do
GitHub.
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
# Instalar as dependências torch-scatter e torch_sparse da biblioteca
pytorch_geometric.
!pip install -q torch-scatter torch_sparse -f
https://data.pyg.org/whl/torch-2.1.0+cu118.html
# Importar módulos necessários para o funcionamento do script.
import os
import os.path as osp
import numpy as np
import torch
import pandas as pd
from torch_geometric.datasets import MovieLens1M
from torch_geometric.nn import MetaPath2Vec
import time

# Importar módulos do Google Colab para upload de arquivos e montagem
do Google Drive.
from google.colab import files
from google.colab import drive
# Montar o Google Drive para acesso aos arquivos e armazenamento.
drive.mount('/content/drive/')
# Link Dataset: https://pytorch-
geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.Mo
vieLens1M.html#torch_geometric.datasets.MovieLens1M
path = osp.join('MovieLens1M_folder', 'MovieLens1M')
dataset = MovieLens1M(path)
data = dataset[0]
print(data)
# Definir o dispositivo de processamento (GPU ou CPU).
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
# Definir a metapath para a análise no grafo.
metapath = [
            ('user', 'rates', 'movie'),
            ('movie', 'rated_by', 'user')
]
# Imprimir as avaliações (ratings) na relação 'user'-'rates'-'movie'.
print(data['user', 'rates', 'movie'].rating)
# Obter as avaliações (ratings) e calcular o número de valores únicos.
k = data['user', 'rates', 'movie'].rating.cpu().numpy()
unique_values = len(np.unique(k, axis=0))
print(unique_values)
```

```
# Obter a representação dos nós do tipo 'movie' e calcular o número de
valores únicos.
k = data['movie'].x.cpu().numpy()
unique_values = len(np.unique(k, axis=0))
print(unique_values)
# Imprimir a representação dos nós do tipo 'movie'.
print(data['movie'].x)
# Imprimir o índice das arestas na relação 'movie'-'rated_by'-'user'.
print(data['movie', 'rated_by', 'user'].edge_index)
# Imprimir o dicionário de índices das arestas do grafo.
print(data.edge_index_dict)
# Criar um DataFrame vazio com colunas pré-definidas para armazenar
resultados.
results_df = pd.DataFrame(columns=['Walk_Length_P', 'Context_Size_P',
'Train_Loss', 'Train_Acc', 'Test_Acc','Time_Epoch'])
# Iniciar a medição do tempo total do processo.
tempo_total_inicial = time.time()

# Iterar sobre os valores de walk_length e context_size.
for walk_length in range(3, 33):
    for context_size in range(3, 34):
        # Verificar a condição de continuação.
        if not (walk_length + 1 >= context_size):
            continue

        # Iniciar a medição do tempo de uma época.
        tempo_inicio = time.time()

        # Inicializar o modelo MetaPath2Vec com os parâmetros
definidos.
        model = MetaPath2Vec(data.edge_index_dict,
                             embedding_dim=128,
                             metapath=metapath,
                             walk_length=walk_length,
                             context_size=context_size,
                             walks_per_node=3,
                             num_negative_samples=1,
                             sparse=True
                            ).to(device)

        # Criar um carregador de dados para o modelo.
        loader = model.loader(batch_size=128, shuffle=True,
num_workers=3)

        # Inicializar o otimizador SparseAdam para o modelo.
        optimizer = torch.optim.SparseAdam(list(model.parameters()),
lr=0.01)

        # Definir a função de treino do modelo.
```

```
        def train(epoch, log_steps=500, eval_steps=1000):
            model.train()
            total_loss = 0
            for i, (pos_rw, neg_rw) in enumerate(loader):
                optimizer.zero_grad()
                loss = model.loss(pos_rw.to(device), neg_rw.to(device))
                loss.backward()
                optimizer.step()
                total_loss += loss.item()

                # Imprimir logs a cada número definido de passos.
                if (i + 1) % log_steps == 0:
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Loss: {total_loss / log_steps:.4f}'))
                    total_loss = 0

                # Avaliar e imprimir o desempenho a cada número
definido de passos.
                if (i + 1) % eval_steps == 0:
                    acc = test()
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Acc: {acc:.4f}'))
                    print("Tempo_parcial", f"{time.time() -
tempo_inicio:.4f}")

            return total_loss / len(loader)

        # Definir a função de teste do modelo com processamento de nós
individuais.
        @torch.no_grad()
        def test(train_ratio=0.1):
            model.eval()
            zs = []
            # Obter a representação vetorial para cada nó do tipo
'user'.
            for idx, _ in enumerate(data['user'].x):  # idx é o índice
do nó
                z = model('user', batch=torch.tensor([idx]).to(device))
                zs.append(z)
            z = torch.cat(zs)
            y = data['user', 'rates', 'movie'].rating
            perm = torch.randperm(z.size(0))
            train_perm = perm[:int(z.size(0) * train_ratio)]
            test_perm = perm[int(z.size(0) * train_ratio):]

            return model.test(z[train_perm], y[train_perm],
z[test_perm], y[test_perm], max_iter=150)
```

```
        # Treinar e testar o modelo para uma época.
        train_loss = train(epoch=1)
        train_acc = test(train_ratio=0.9)
        test_acc = test(train_ratio=0.1)

        # Calcular e armazenar o tempo de duração de uma época.
        tempo_epoca = f"{time.time() - tempo_inicio:.6f}"

        # Armazenar os resultados no DataFrame.
        context_size_p = context_size
        walk_length_p = walk_length
        new_row = pd.DataFrame({'Walk_Length_P': [walk_length_p],
'Context_Size_P': [context_size_p],'Train_Loss': [train_loss],
'Train_Acc': [train_acc], 'Test_Acc': [test_acc], 'Time_Epoch':
[tempo_epoca]})
        results_df = pd.concat([results_df, new_row],
ignore_index=True)

        # Imprimir o DataFrame atualizado.
        print(results_df)

        # Salvar o DataFrame em um arquivo Excel.
        results_df.to_excel("MovieLens1M-MetaPath2Vec.xlsx")
        results_df.to_excel("/content/drive/MyDrive/Colab
Notebooks/FINAIS/data_MovieLens1M/MovieLens1M-MetaPath2Vec.xlsx")

# Calcular e imprimir o tempo total de execução do processo.
tempo_total_final = time.time() - tempo_total_inicial
print("TEMPO TOTAL:", format(tempo_total_final, '.6f'), "SEGUNDOS")
```

APÊNDICE F – Código completo *dataset* OGB_MAG

```
# Instalar a biblioteca pytorch_geometric diretamente do repositório do
GitHub.
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
# Instalar as dependências torch-scatter e torch_sparse da biblioteca
pytorch_geometric.
!pip install -q torch-scatter torch_sparse -f
https://data.pyg.org/whl/torch-2.1.0+cu118.html
# Importar módulos necessários para o funcionamento do script.
import os
import os.path as osp
import numpy as np
import torch
import pandas as pd
from torch_geometric.datasets import OGB_MAG
from torch_geometric.nn import MetaPath2Vec
import time
# Importar módulos do Google Colab para upload de arquivos e montagem
do Google Drive.
from google.colab import files
from google.colab import drive
# Montar o Google Drive para acesso aos arquivos e armazenamento.
drive.mount('/content/drive/')
# Inverter a aresta [('paper', 'has_topic',
'field_of_study')].edge_index
from torch_sparse import transpose
def add_inverse_edges(data):  # Define a função add_inverse_edges que
recebe um parâmetro data

    # Adiciona arestas inversas para a relação ('field_of_study',
'contains', 'paper')
    data[('field_of_study', 'contains', 'paper')].edge_index =
transpose(
        data[('paper', 'has_topic', 'field_of_study')].edge_index,  #
Arestas originais
        None,  # Não é especificado um tensor para armazenar os valores
das arestas transpostas
        m=data['paper'].num_nodes,  # Número de nós do tipo 'paper'
        n=data['field_of_study'].num_nodes)[0]  # Número de nós do tipo
'field_of_study'

    return data  # Retorna o objeto data modificado
# Reduzir Dataset para 6% do tamanho original
from torch_geometric.data import HeteroData
from torch_geometric.transforms import BaseTransform
```

```
import random
class ReduceDatasetSize(BaseTransform):
    def __call__(self, data):
        # Iniciar o cronômetro para calcular o tempo de execução
        start_time = time.time()

        # Definir a semente para garantir a reprodutibilidade
        seed = 1
        torch.manual_seed(seed)
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)  # se estiver usando multi-
GPU.
        np.random.seed(seed)  # se estiver usando numpy
        random.seed(seed)  # se estiver usando o módulo random

        # Garantir o comportamento determinístico no CUDA
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False

        # Passo 1: Selecionar 6% de nós de cada tipo
        print("Passo 1: Selecionando 6% dos nós de cada tipo...")
        subset_dict = {}
        for node_type in data.node_types:
            total_nodes = data[node_type].num_nodes
            retain_count = int(total_nodes * 0.06)
            subset = torch.randperm(total_nodes)[:retain_count]
            subset_dict[node_type] = subset
        print("Passo 1 concluído - {:.2f}s".format(time.time() -
start_time))

        # Configurar a semente novamente
        torch.manual_seed(seed)

        # Passo 2: Criar o subgrafo a partir dos subconjuntos de nós
selecionados
        print("Passo 2: Criando o subgrafo...")
        subset_data = data.subgraph(subset_dict)
        print("Passo 2 concluído - {:.2f}s".format(time.time() -
start_time))

        # Passo 3: Remover nós isolados
        print("Passo 3: Removendo nós isolados...")
        removed_isolated_nodes = True
        while removed_isolated_nodes:
            removed_isolated_nodes = False
            degrees = {}
            for edge_type in subset_data.edge_types:
                src, _, dst = edge_type
                edge_index = subset_data[edge_type].edge_index
```

```
                for s, d in zip(edge_index[0], edge_index[1]):
                    degrees[src, s.item()] = degrees.get((src,
s.item()), 0) + 1
                    degrees[dst, d.item()] = degrees.get((dst,
d.item()), 0) + 1

            isolated_nodes = {node_type: [] for node_type in
subset_data.node_types}
            for node_type in subset_data.node_types:
                for i in range(subset_data[node_type].num_nodes):
                    if degrees.get((node_type, i), 0) == 0:
                        isolated_nodes[node_type].append(i)
                        removed_isolated_nodes = True

            for node_type, isolated in isolated_nodes.items():
                if isolated:
                    valid_indices = [i for i in
range(subset_data[node_type].num_nodes) if i not in isolated]
                    valid_original_indices =
subset_dict[node_type][valid_indices]
                    subset_dict[node_type] = valid_original_indices
                    subset_data = data.subgraph(subset_dict)
        print("Passo 3 concluído - {:.2f}s".format(time.time() -
start_time))

        print("Processo concluído - {:.2f}s".format(time.time() -
start_time))

        return subset_data
# Unir ReduceDatasetSize e add_inverse_edges
def pre_transform(data):
    heterodata_reduce = ReduceDatasetSize()
    data = heterodata_reduce(data)
    data = add_inverse_edges(data)

    return data
#LINK DATASET: https://pytorch-
geometric.readthedocs.io/en/2.4.0/generated/torch_geometric.datasets.OG
B_MAG.html#torch_geometric.datasets.OGB_MAG
path = osp.join('OGB_MAG_folder', 'OGB_MAG')
dataset = OGB_MAG(path, pre_transform=pre_transform)
data = dataset[0]
print(data)
# Definir o dispositivo de processamento (GPU ou CPU).
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
# Definir a metapath para a análise no grafo.
metapath = [
    ('paper', 'cites', 'paper'),
```

```
    ('paper', 'has_topic', 'field_of_study'),
    ('field_of_study', 'contains', 'paper')
]
# Obter a representação dos nós do tipo 'paper' e calcular o número de
valores únicos.
k = data['paper'].x.cpu().numpy()
unique_values = len(np.unique(k, axis=0))
print(unique_values)
# Obter os rótulos dos nós do tipo 'paper' e calcular o número de
valores únicos.
k = data['paper'].y.cpu().numpy()
unique_values = len(np.unique(k, axis=0))
print(unique_values)
# Imprimir o tipo e a representação dos nós do tipo 'paper'.
print(type(data['paper'].x))
print(data['paper'].x)
# Imprimir o tipo e os rótulos dos nós do tipo 'paper'.
print(type(data['paper'].y))
print(data['paper'].y)
# Definir a função para verificar a existência de nós isolados no
dataset.
def has_isolated_nodes(dataset):
    for node_type in dataset.node_types:
        if not hasattr(dataset[node_type], 'num_nodes'):
            continue  # Pular se num_nodes não está definido para o
tipo de nó

        num_nodes = dataset[node_type].num_nodes
        edge_indices = [dataset[edge_type].edge_index for edge_type in
dataset.edge_types if edge_type[0] == node_type or edge_type[2] ==
node_type]

        connected_nodes = set()
        for edge_index in edge_indices:
            if edge_index.size(1) > 0:
                src, dst = edge_index[0], edge_index[1]
                connected_nodes.update(src.tolist())
                connected_nodes.update(dst.tolist())

        isolated_nodes = set(range(num_nodes)) - connected_nodes
        if isolated_nodes:
            print(f"Nós isolados em {node_type}: {isolated_nodes}")
# Executar a função para verificar nós isolados.
has_isolated_nodes(data)
# Imprimir o dicionário de índices das arestas do grafo.
print(data.edge_index_dict)
# Criar um DataFrame vazio com colunas pré-definidas para armazenar
resultados.
```

```
results_df = pd.DataFrame(columns=['Walk_Length_P', 'Context_Size_P',
'Train_Loss', 'Train_Acc', 'Test_Acc','Time_Epoch'])
# Iniciar a medição do tempo total do processo.
tempo_total_inicial = time.time()

# Iterar sobre os valores de walk_length e context_size.
for walk_length in range(3, 33):
    for context_size in range(3, 34):
        # Verificar a condição de continuação.
        if not (walk_length + 1 >= context_size):
            continue

        # Iniciar a medição do tempo de uma época.
        tempo_inicio = time.time()

        # Inicializar o modelo MetaPath2Vec com os parâmetros
definidos.
        model = MetaPath2Vec(data.edge_index_dict,
                             embedding_dim=128,
                             metapath=metapath,
                             walk_length=walk_length,
                             context_size=context_size,
                             walks_per_node=3,
                             num_negative_samples=1,
                             sparse=True
                            ).to(device)

        # Criar um carregador de dados para o modelo.
        loader = model.loader(batch_size=128, shuffle=True,
num_workers=3)

        # Inicializar o otimizador SparseAdam para o modelo.
        optimizer = torch.optim.SparseAdam(list(model.parameters()),
lr=0.01)

        # Definir a função de treino do modelo.
        def train(epoch, log_steps=500, eval_steps=1000):
            model.train()
            total_loss = 0
            for i, (pos_rw, neg_rw) in enumerate(loader):
                optimizer.zero_grad()
                loss = model.loss(pos_rw.to(device), neg_rw.to(device))
                loss.backward()
                optimizer.step()
                total_loss += loss.item()

                # Imprimir logs a cada número definido de passos.
                if (i + 1) % log_steps == 0:
```

```
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Loss: {total_loss / log_steps:.4f}'))
                    total_loss = 0

                # Avaliar e imprimir o desempenho a cada número
definido de passos.
                if (i + 1) % eval_steps == 0:
                    acc = test()
                    print((f'Epoch: {epoch}, Step: {i +
1:05d}/{len(loader)}, '
                           f'Acc: {acc:.4f}'))
                    print("Tempo_parcial", f"{time.time() -
tempo_inicio:.4f}")

            return total_loss / len(loader)

        # Definir a função de teste do modelo com processamento de nós
individuais e ajuste de max_iter.
        @torch.no_grad()
        def test(train_ratio=0.1):
            model.eval()
            zs = []
            # Obter a representação vetorial para cada nó do tipo
'paper'.
            for idx, _ in enumerate(data['paper'].x):  # idx é o índice
do nó
                z = model('paper',
batch=torch.tensor([idx]).to(device))
                zs.append(z)
            z = torch.cat(zs)
            y = data['paper'].y
            perm = torch.randperm(z.size(0))
            train_perm = perm[:int(z.size(0) * train_ratio)]
            test_perm = perm[int(z.size(0) * train_ratio):]

            return model.test(z[train_perm], y[train_perm],
z[test_perm], y[test_perm], max_iter=600)
            # Ajuste de max_iter para 600 para evitar o erro de
convergência.
            #ConvergenceWarning: lbfgs failed to converge (status=1):
            #STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

        # Treinar e testar o modelo para uma época.
        train_loss = train(epoch=1)
        train_acc = test(train_ratio=0.9)
        test_acc = test(train_ratio=0.1)

        # Calcular e armazenar o tempo de duração de uma época.
```

```
        tempo_epoca = f"{time.time() - tempo_inicio:.6f}"

        # Armazenar os resultados no DataFrame.
        context_size_p = context_size
        walk_length_p = walk_length
        new_row = pd.DataFrame({'Walk_Length_P': [walk_length_p],
'Context_Size_P': [context_size_p],'Train_Loss': [train_loss],
'Train_Acc': [train_acc], 'Test_Acc': [test_acc], 'Time_Epoch':
[tempo_epoca]})
        results_df = pd.concat([results_df, new_row],
ignore_index=True)

        # Imprimir o DataFrame atualizado.
        print(results_df)

        # Salvar o DataFrame em um arquivo Excel.
        results_df.to_excel("OGB-MAG-MetaPath2Vec.xlsx")
        results_df.to_excel("/content/drive/MyDrive/Colab
Notebooks/FINAIS/data_OBG_MAG/OGB-MAG-MetaPath2Vec.xlsx")

# Calcular e imprimir o tempo total de execução do processo.
tempo_total_final = time.time() - tempo_total_inicial
print("TEMPO TOTAL:", format(tempo_total_final, '.6f'), "SEGUNDOS")
```