UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

JOÃO PAULO DE SOUZA TAVARES

LINK PREDICTION APLICADO EM GRAFOS DE CITAÇÕES DE ARTIGOS USANDO O NODE2VEC

PATO BRANCO

2022

JOÃO PAULO DE SOUZA TAVARES

LINK PREDICTION APLICADO EM GRAFOS DE CITAÇÕES DE ARTIGOS USANDO O NODE2VEC

Link prediction in paper citation network using Node2vec

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Dalcimar Casanova

Coorientador: Prof. Dr. Jefferson Tales Oliveira

PATO BRANCO

2022

JOÃO PAULO DE SOUZA TAVARES

LINK PREDICTION APLICADO EM GRAFOS DE CITAÇÕES DE ARTIGOS USANDO O NODE2VEC

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 13/dezembro/2022

Prof. Dr. Dalcimar Casanova
Universidade Tecnológica Federal do Paraná

Prof. Dr. Jefferson Tales Oliveira
Universidade Tecnológica Federal do Paraná

Prof. Dr. Marco Antonio de Castro Barbosa
Universidade Tecnológica Federal do Paraná

Prof. Msc. Luis Cassiano Goularte Rista
Universidade Tecnológica Federal do Paraná

PATO BRANCO

2022

RESUMO

O aprendizado de máquina em grafos é uma área de estudo recente na área da computação e inteligência artifical, contudo já apresenta perspectiva para a solução de difíceis que os seres humanos se de deparam dia a dia, como ajudar na decodificação de DNA, otimização em redes de suprimento e energia e, na área acadêmica, pode fornecer um ótimo alicerce para o pesquisador encontrar artigos correlatos com uma qualidade e velocidade maior. Na aplicações deste problema, dados de publicações e citações são normalmente modelados como grafos, desta forma é possível aplicar conceito de *graph embedding* para a transformação deste grafo em um espaço vetorial que possa ser aplicado em algoritmos de aprendizado de máquina tradicionais, e dessa forma criar um modelo de predição baseado nas características do grafo original. Tentar encontrar arestas faltantes em grafo usando um modelo de predição é a tarefa chamada de *link prediction*. Contudo, criar modelos de inteligência artificial baseados em grafos não é uma tarefa trivial, devido a complexidade envolvida na tarefa. Portanto, neste Trabalho de Conclusão de Curso, pretende-se aplicar o algoritmo *node2vec* para o aprendizado de características em grafos de citações de artigos, e criar um modelo de predição para realizar a tarefa de *link prediction* e encontrar publicações correlatas dentre os nós do grafo estudado.

Palavras-chave: aprendizado de máquina; *graph embedding*; *node2vec*; grafo de citações de artigos; *link prediction*.

ABSTRACT

Machine Learning applied on graphs is a recent field of study on artificial intelligence area, although it it is already solving difficult problems on daily challenge faced by humans, like DNA decodification, optimization of supply and energy networks, and, in the academic field, it could provide powerful tool to allow the researcher to find correlated papers of high quality. In applications of this problem, academic paper databases are modelated in graphs, and then it is possible to apply graph embedding techniques transform this graph in a d-dimensional vectors that can be used on tradicional machine learning algorithms, then it is made a prediction model based on the original graph features, predicting these missing links is a task name link prediction. However, create those models are no easy task, due the complexity involved on the task. Therefore, this work aims to apply the node2vec algorithm to learn porwerful embeddings of the paper citation network, and create a prediction model to perfom link prediction tasks and find related papers on the network.

Keywords: machine learning; graph embedding; node2vec; paper citation network; link prediction.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

LLE	Locally Linear Embedding
SGD	*Stochastic Gradient Descent*
ROC	*Receiver Operating Characteristic*
AUC	*Area Under Curve*
PCA	*Principal component analysis*
UTFPR	Universidade Tecnológica Federal do Paraná

SUMÁRIO

1 INTRODUÇÃO

Os sistemas de recomendações são um tópico em alta durante a era de *big data* e têm uma área vasta de aplicação. Por exemplo, os métodos de sugestão de amizades no *Facebook* analisando a rede de amigos em comum, ou a recomendação de produtos da *Amazon* a partir do histórico de busca do usuário (PAN *et al.*, 2015). Entretanto, no meio acadêmico são poucos os sistema de sugestão de artigos que possa recomendar artigos relacionados com base em um artigo específico ou histórico de busca de artigos, dentre os sistemas de recomendação presentes na literatura é possível citar o modelo de recomendação baseada em grafos heterogêneos (PAN *et al.*, 2015) ou um método misto de *PageRank* e *HITS framework* proposto em (WANG; WANG; YAN, 2014)

Grafos são formas genéricas de representar informações, seja para modelar redes de amigos, redes de infraestrutura, redes de *e-commerce* e redes de energia. Outra aplicação é em redes de citações de artigos, onde os grafos são usados para representar os artigos (vértices) e suas citações (arestas).E analisar tais grafos pode beneficiar tarefas importantes na pesquisa acadêmica, como dado um artigo específico, encontrar artigos relacionado a eles (PAN *et al.*, 2015).

Atualmente, para realizar tais tarefas de predição, reconhecimento de padrões e extração de características são utilizados os algoritmos de *deep learning*, seja para classificar imagens, processar vídeos ou reconhecimento de fala. Contudo, para utilizar esses algoritmos os dados devem estar no formato Euclidiano, e muitas aplicações vêm crescendo onde dados são representado na forma não-Euclidiana - como os grafos - (WU *et al.*, 2019). Os grafos biomédicos são exemplos de uma forma de dados não-Euclidiana e são grande desafio para a aplicação de algoritmos de aprendizado de máquina por causa de sua irregularidade, como ter tamanho variável, a quantidade de arestas em cada nó e cada vértice tendo, assim, diferentes vizinhanças.

Devido às vantagens de aplicar algoritmos de *deep learning* em dados modelados como grafos muitas abordagens novas estão sendo propostas como os modelos baseado em redes neurais convolucionais (CNNs), redes neurais recorrentes (RNNs) e *autoencoders* do *deep learning* (WU *et al.*, 2019). Tipicamente os modelos desenvolvidos para resolver os problemas baseados em grafos usam ou a matriz de adjacência do grafo, ou uma representação em espaço vetorial do grafo - *Graph Embedding* - (GOYAL; FERRARA, 2018). O *Graph Embedding* vem ganhando popularidade, pois representar um grafo inteiro em um vetor de baixa dimensionalidade permite eficientemente aplicar algoritmos de aprendizado de máquina desenvolvidos para lidar com representações vetoriais de grandes conjuntos de dados para resolver uma grande variedade de problemas analíticos (CHANG *et al.*, 2015).

Representar os nós e arestas de um grafo na forma de um vetor de características envolve, geralmente, (a) engenharia de características - *feature engineering* -, ou (b) aprendizado de características pela solução de problemas de otimização - *feature learning* - (GROVER; LES-

KOVEC, 2016; BENGIO; COURVILLE; VINCENT, 2013). A forma de *feature engineering* não é genérica, o que a torna inviável em problemas do mundo real.

As abordagens mais clássicas de *feature learning* são as baseadas em (1) *factorization*, ou seja, técnicas de redução linear e não-linear como *Principal Component Aalysis*, *Multi-Dimensional Scaling*, *Laplacian Eigenmaps* e *Locally Linear Embedding* que não são escaláveis para grafos no mundo real, pois otimizam uma função objetiva que visa encontrar uma decomposição da matriz de adjacência do grafo o que é muito caro computacionalmente (GOYAL; FERRARA, 2018; GROVER; LESKOVEC, 2016; NOZZA; FERSINI; MESSINA, 2019).

Todavia, há alternativas escaláveis para os problemas de *feature learning*, e são os modelos baseados em (2) *Random Walk* e (3) *Deep Learning*. As abordagens usando o *Random Walk* percorrem o grafo em passadas aleatórias, desta forma detectando estruturas locais e globais, enquanto os modelos de *Deep Learning* utilizam *Deep Autoencoders* para a redução de dimensionalidade e são capazes de capturar as não-linearidades do grafo (GOYAL; FERRARA, 2018).

Nesse contexto, insere-se o *node2vec*, um algoritmo semi-supervisionado e baseado em *Random Walk* para aprendizado de característica escalável em grafos. Este algoritmo retorna uma representação de características que maximizam as semelhanças das vizinhanças preservadas dos nós do grafo em um espaço d-dimencional de características. Logo, pretende-se aplicar o algoritmo *node2vec* em grafos de citações de artigos, para realizar a tarefas de *link prediction*.

1.1 Objetivo Geral

Realizar um estudo de caso do algoritmo *node2vec* na criação de um modelo de aprendizado de máquina capaz de prever a correlação entre artigos em um grafo de citações de artigos acadêmicos.

1.2 Objetivos Específicos

- Utilizar o *node2vec* para criar representações vetoriais significativa de grafos.
- Criar um modelo de aprendizado de características em grafos de citações de artigos.
- Aplicar *link prediction* para prever possíveis artigos relacionados no modelo criado.

1.3 Justificativa

De acordo com (GOYAL; FERRARA, 2018) obter uma representação de cada nó de um grafo é inerentemente complicado e impõe vários desafios, os quais vêm direcionando as pesquisas para as seguintes áreas:

i Escolha da propriedade: Uma boa representação vetorial dos nós deve preservar a estrutura do grafo e conexão entre cada vértice. O primeiro desafio é escolher as propriedades do grafo que a representação deve preservar. Devido à abundância de métricas de distância e propriedades definidas para os grafos, essa escolha pode ser difícil e a performance vai depender da aplicação.

ii Escalabilidade: A maioria das redes rais são grandes e contêm milhões de nós e arestas - métodos de representação devem ser escaláveis e hábeis de processar tais grafos. Definir um modelo escalável é particularmente difícil quando o modelo visa preservar as propriedades globais do grafo.

iii Dimensionalidade da representação: Encontrar a dimensionalidade ótima da representação pode ser difícil. Por exemplo, quanto maior o número de dimensões maior a precisão da reconstrução, porém isto implicará em maior complexidade em tempo e tamanho. A escolha também pode ser específica pra aplicação dependendo da abordagem: por exemplo, um baixo número de dimensões resultam em melhores resultados em tarefas de *link prediction* se o modelo escolhido capturar as conexões locais entre os nós.

O *node2vec* é um algoritmo que realiza a representação independe da robustez do grafo pois é flexível e controlável na exploração do grafo através de seus hiperparâmetros p e q, é escalável e é semi-supervisionado (utiliza algoritmos de *Deep Learning* no seu modelo) (GROVER; LESKOVEC, 2016; GOYAL; FERRARA, 2018).

Portanto, com o node2vec, é possível aplicar uma grande variedade de grafos do mundo real, devido a sua escalabilidade. E também o algoritmo é adaptável a diferentes problemas de predição, como o *link prediction* ou o *node classification*, via de seus hiperparâmetros.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentadas as principais pesquisas e estudos na área de *graph embedding* e suas aplicações e explicação do algoritmo *node2vec*.

2.1 Definições

Nesta seção serão apresentadas as definições dos conceitos básicos de *graph embedding*, e então introduzir a definição formal do problema de *graph embedding*, ambas as definições foram propostas por (GOYAL; FERRARA, 2018).

Definição 1 (Grafo). Um grafo G(V, E) é uma combinação de $V = \{v_1, \cdots, v_n\}$ vértices (ou nós) e $E = \{e_{ij}\}_{i,j=1}^{n}$ arestas. A matrix de adjacência S do grafo G contem pesoa não negativos associados a cada aresta: $s_{ij} \geq 0$. Se v_i e v_j não estão conectados, então $s_{ij} = 0$. Para grafos ponderados não direcionados, $s_{ij} = s_{ji}\ \forall i, j \in |V|$.

O peso das aretas s_{ij} normalmente é usado como uma medida da similaridade entre os vértices v_i e v_j, ou seja quanto maior for o peso da aresta, mais similares os nós serão.

Definição 2 (Distância de primeira ordem). O peso das arestas s_{ij} também são denominados distância de primeira ordem entre os nós v_i e v_j, já que essa é a primeira e mais importante medida de similaridade entre dois vértices.

Definição 3 (Distância de segunda ordem). A distância de segunda ordem entre um par de vértices descreve a proximidade das estruturas da vizinhanças dos nós. Seja $s_i = [s_{i1}, \cdots, s_{in}]$ a representação da distância de primeira ordem do vértice v_i e os outros nós. Então, a distância de segunda ordem entre v_i e v_j é determinada pela similaridade de s_i e s_j.

Definição 4 (*Graph embedding*). Dado um grafo $G = (V,E)$ e uma dimensionalidade predefinida para o *embedding* d (sendo $d \ll |V|$), o problema *graph embedding* é transformar G em um espaço d-dimensional, onde as propriedades do grafo são preservadas. Tais propriedades são quantificadas usando as medidas de proximidade tanto de primeira quanto de maior ordem. O grafo, então, é representado como um vetor com d dimensões, ou por um conjunto de vetores d-dimensionais, onde cada vetor representa o *embedding* de uma parte do grafo.

2.2 Aplicações de *Graph Embedding*

Há diversas aplicações para os *embeddings* de grafos. (GOYAL; FERRARA, 2018) classifica as aplicações como: *network compression* (2.2.1), visualização (2.2.2), *clustering* (2.2.3), *link prediction* (2.2.4) e *node classification* (2.2.5).

2.2.1 *Network compression*

O primeiro a introduzir o conceito de *network compression* (ou *graph simplification*) foi (FEDER; MOTWANI, 1991). Eles definiram a compressão G^* de um grafo G, a representação mantém aspectos da estrutura original do grafo e tem duas propriedades: G^* possui menos arestas que G e é computacionalmente fácil converter G para G^* e vice-versa. A compressão é obtida por meio da partição do grafo inicial em *bipartite cliques*. E então, cada *clique* é substituído por uma árvore, o que reduz o número de arestas na compressão. Muitos trabalhos usaram métodos baseados em agregação para comprimir grafos (GOYAL; FERRARA, 2018), a principal ideia nesses métodos é aproveitar-se da *link structure* do grafo para agrupar os nós e arestas. (NAVLAKHA; RASTOGI; SHRIVASTAVA, 2008) desenvolveu algoritmos para construir representações altamente compactas de grafos consistindo em *graph summary* e *edge correction*.

Similar a tais representações , *graph embeddings* podem ser interpretados como uma sumarização de grafos. (GOYAL; FERRARA, 2018) afirma que essa hipótese foi testada ao reconstruir um grafo a partir de seu *embedding* e medindo o erro da reconstrução. E concluiu-se que para uma representação de baixa dimensionalidade para cada nó (na ordem de 100), a reconstrução do grafo tem alta precisão.

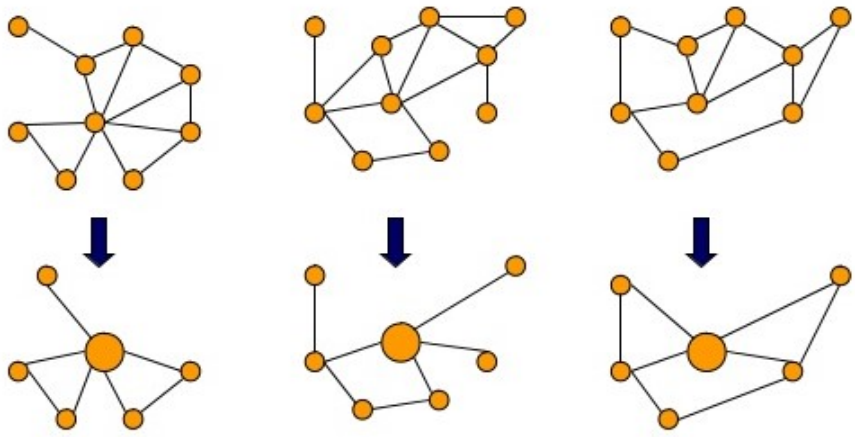


Figura 1 – Exemplo de *network compression*
Fonte: Adaptado de (LIOUNG, 2014).

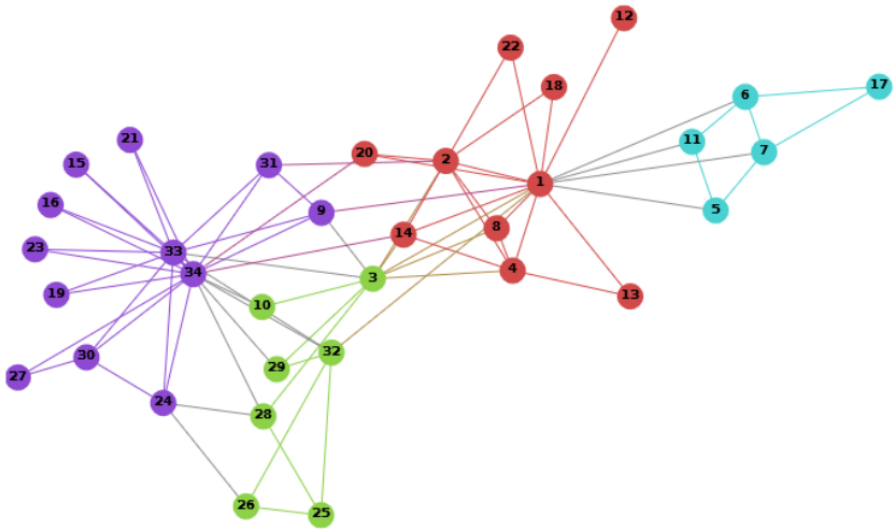
Na Figura 1 é feita a *network compression* dos grafos mostrados, no exemplo o a padrão de 4 nós interligados é substituído por apenas um, diminuindo dessa forma o tamanho da dimensão do *embedding*.

2.2.2 Visualização

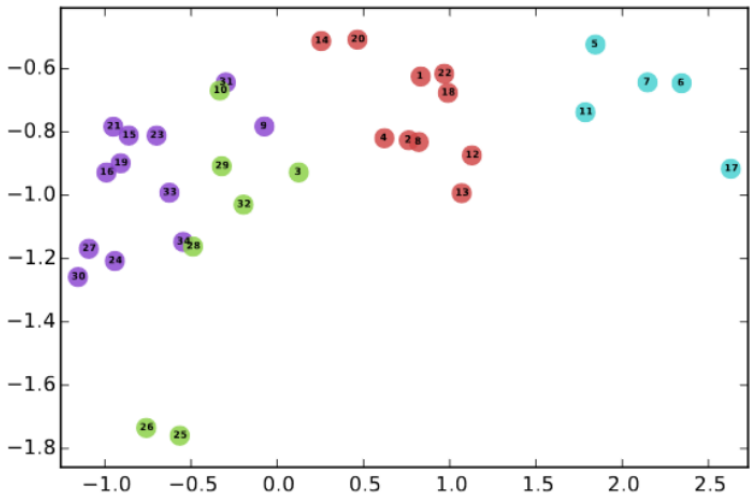
A visualização de grafos tem aplicações em diversas áreas como *software engineering*, circuitos elétricos, biologia e sociologia (GOYAL; FERRARA, 2018). Em (BATTISTA *et al.*, 1994;

EADES; XUEMIN, 1989) vários métodos usados para desenhar grafos e critérios estéticos foram analisados. Tais métodos métodos foram generalizados a partir da perspectiva de visualização da informação nos grafos em (HERMAN; MELANÇON; MARSHALL, 2000).

Como *embedding* representa um grafo no espaço vetorial, técnicas de redução de dimensionalidade como *Principal Component Analysis* (PCA) e *t-distributed stochastic neighbor embedding* (t-SDNE) são usados em visualização de grafos (GOYAL; FERRARA, 2018). Em (PEROZZI; AL-RFOU; SKIENA, 2014) a melhora da abordagem de *embedding* do *DeepWalk* foi mostrada pela visualização de grafo.



(a) Entrada: *Karate Graph*



(b) Saída: Representação no espaço 2D

Figura 2 – Exemplo de *graph visualization* no grafo *Zachary's Karate Club*

Fonte: Adaptado de (PEROZZI; AL-RFOU; SKIENA, 2014).

Na Figura 2 o grafo *Zachary's Karate Club* foi usado como entrada de um algoritmo de visualização, que por sua vez gera uma visualição em 2D do grafo.

2.2.3 *Clustering*

Segundo (GOYAL; FERRARA, 2018) *graph clustering* pode ser de dois tipos: (1) baseado na estrutura, e (2) baseado em atributos. Métodos baseados em estrutura têm como objetivo agrupar subgrafos ou vértices com funções similares. Já os métodos baseados em atribulos utilizam os rótulos dos vértices para agrupá-los.

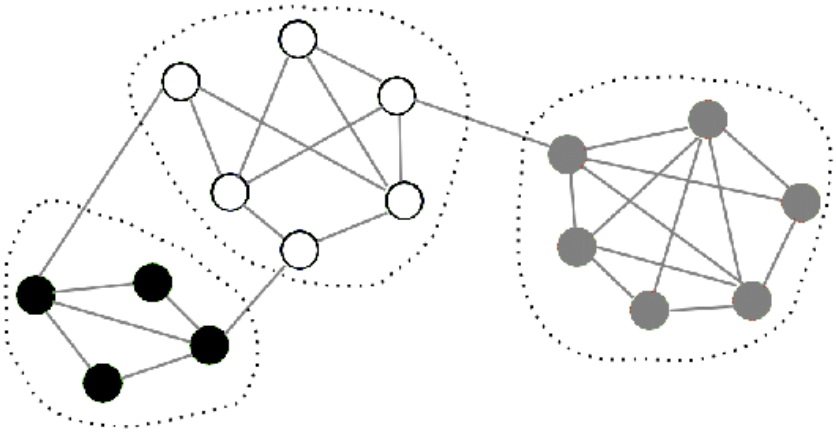


Figura 3 – Exemplo de *clustering* em grafo
Fonte: Retirado de (VENTO, 2014).

Na Figura 3 é mostrado um exemplo de *clustering* em grafo. Os nós mais semelhantes entre si (representando as cores) são agrupados no mesmo conjunto (subgrafo).

2.2.4 *Link prediction*

Grafos são construídos a partir de observação das interações entre entidades, que pode ser incompleto ou impreciso (GOYAL; FERRARA, 2018). O desafio muitas vezes é identificar as interações incorretas e prever informações ausentes, logo *link prediction* refere-se à tarefa de prever tanto as interações ausentes quanto prever *links* que podem aparecer no futuro em dado grafo em expansão. *Link prediction* é importantíssimo em análise de redes biológicas, onde identificar a existência de *links* entre os nós é um experimento muito caro. Em redes sociais, *link prediction* é usado para prever prováveis amizades, o que pode ser usado como recomendações. Os algoritmos de *link prediction* são categorizados em (LIBEN-NOWELL; KLEINBERG, 2007; LÜ; ZHOU, 2011; HASAN; ZAKI, 2011): (a) baseados em similaridade (local e global); (b) baseados na máxima possibilidade e (c) métodos probabilísticos.

Embedding possibilita a aplicação de *link prediction*, pois os métodos de *graph embedding* capturam as características das redes sendo elas explícitas ou implícitas (GOYAL; FER-

RARA, 2018). O *embedding* de um grafo social foi utilizado para prever *links* em (WANG; CUI; ZHU, 2016). Em (GROVER; LESKOVEC, 2016) o *link prediction* foi aplicado em *embeddings* de redes biológicas, e foi verificados que nesses conjuntos a aplicação de *link prediction* em *embedding* é mais precisa que os métodos baseados em similaridade.

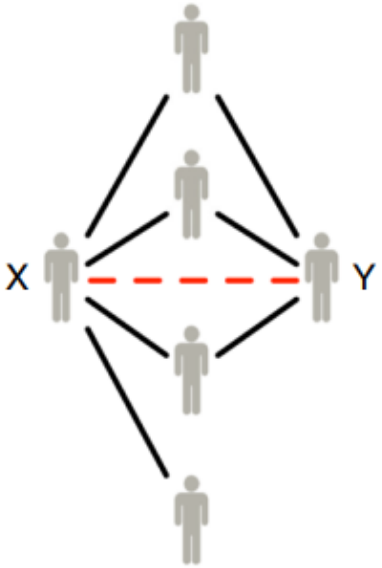


Figura 4 – Exemplo de *link prediction* em grafo de redes sociais
Fonte: Retirado de (KOVÁCS *et al.*, 2019).

Um exemplo de *link prediction* é mostrado na Figura 4 em um grafo de redes sociais. O algoritmo para *link prediction* tenta estimar uma aresta (*link*) entre os vértices X e Y com base nas conexões que eles têm em comum.

2.2.5 *Node classification*

Usualmente uma fração de nós em um grafo são rotulados e ele pode ter diversas funções. Em *social networks*, rótulos podem indicar interesses, crenças ou fatores demográficos (como renda, idade, etnia etc.). Em *language networks*, tópicos ou palavras-chave podem ser rótulos de documentos, enquanto que os rótulos das entidades de redes biológicas são baseadas em suas funcionalidades (GOYAL; FERRARA, 2018). Entretanto os rótulos podem ser desconhecidos para uma grande quantidade de nós. E tais rótulos faltantes podem ser determinados a partir de nós rotulados e dos *links* do grafo. A tarefa de prever esse rótulos desconhecidos é chamada de *node classification* (GOYAL; FERRARA, 2018). (BHAGAT; CORMODE; MUTHUKRISHNAN, 2011) classifica os métodos de *node classification* em: (1) baseados em *feature extraction* e (2) baseados em *ramdom walk*. Modelos baseados em *feature extraction* geram características para os nós baseadas na vizinhança deles e em estatísticas locais do grafo e então aplicam um classificador, como *Logistic Regression* (JR; LEMESHOW; STURDIVANT, 2013) ou

Naive Bayes (MCCALLUM; NIGAM *et al.*, 1998), para prever os rótulos. Os métodos baseados em *random walks* propagam os rótulos por meio de *random walks*.

Embeddings extraem automaticamente características dos nós baseadas na estrutura do grafo e, portanto, são categorizados como um método baseado em *feature extraction* (GOYAL; FERRARA, 2018). Em (TANG *et al.*, 2015; WANG; CUI; ZHU, 2016; OU *et al.*, 2016; PEROZZI; AL-RFOU; SKIENA, 2014; GROVER; LESKOVEC, 2016) foi avaliada a acurácia das predições do *embedding* de vários *information networks* como *language*, *social*, *biology*, e *collaboration graphs*, e verificou-se que *embeddings* podem prever rótulos desconhecidos com alta precisão.

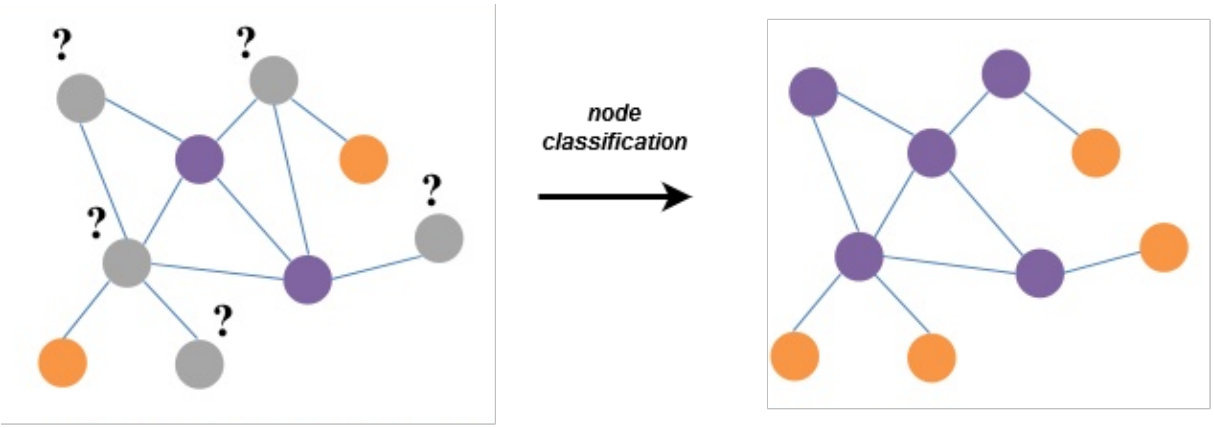


Figura 5 – Exemplo de *node classification* em grafo
Fonte: Adaptado de (LESKOVEC, 2017).

Na Figura 5 é mostrado um exemplo de *node classification*. No grafo original há vértices com as cores laranja e lilás que são os nós rotulados, e vértices com a cor cinza que são os nós sem rótulos. Ao aplicar a tarefa de *node classification* os nós cinzas serão rotulados com base nos vértices previamente rotulados.

2.3 Taxonomia de *Graph Embedding*

A taxonomia das abordagens de *embedding* foi retirada de (GOYAL; FERRARA, 2018), que categoriza os métodos de *embedding* em três categorias: (1) baseados em *factorization*, (2) baseados em *random walk* e (3) baseados *deep learning*.

2.3.1 Métdos baseados em *factorization*

Algoritmos baseados em *factorization* representam as conexões entre os nós na forma de matriz e então fatorizam essa matriz para obter o *embedding*. As matrizes usadas para representar as conexões podem ser a matriz de nós de adjacência, a matriz laplaciana, a matriz de *node transition probability*, a matriz de *Katz similarity*, entre outras. Os métodos para fatorizar as matrizes de representação variam dependendo da propriedade da matriz, por exemplo o

uso de *eigenvalue decomposition* em representações na forma de matrizes laplacianas, outro exemplo é usar o método gradiente descendente para fatorizar matrizes não estruturadas.

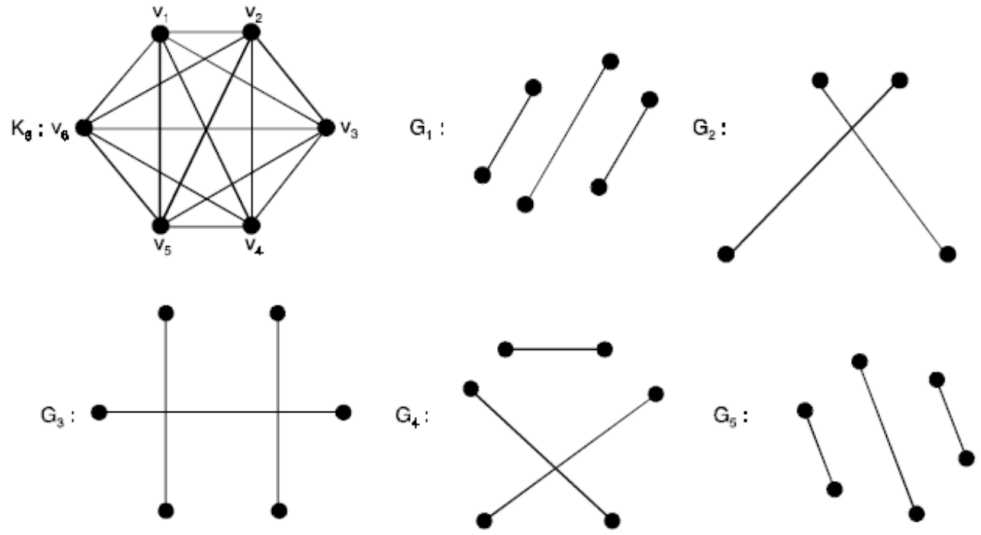


Figura 6 – *Factorization* do grafo K_6
Fonte: Adaptado de https://www.skedsoft.com/books/graph-theory/partitions-and-factorization.

Na Figura 6 é mostrado um modo simplificado de como é feita a *factorization* em grafo. Neste exemplo o são pequenos subgrafos do grafo original resultados da segmentação de seus vértices e de suas arestas.

Alguns métodos baseados em *factorization* são *Locally Linear Embedding* (ROWEIS; SAUL, 2000), *Laplacian eigenmaps* (BELKIN; NIYOGI, 2002), *Cauchy graph embedding* (LUO *et al.*, 2011), *Structure preserving Embedding* (SHAW; JEBARA, 2009) e *Graph Factorization* (AHMED *et al.*, 2013).

2.3.2 Métdos baseados em *random walk*

Random walks são usadas para aproximar muitas propriedades nos grafos, em particular a *node centrality* e a similaridade. Eses métodos são especialmente úteis quando uma propriedade é observada em apenas algumas partes do grafo, ou o grafo é muito grande e não é possível medir a propriedade no grafo inteiro. Técnicas de *embedding* usando *random walks* em grafos são propostas em: *DeepWalk* (PEROZZI; AL-RFOU; SKIENA, 2014) e *node2vec* (GROVER; LESKOVEC, 2016).

Na Figura 7 é mostrada uma visão geral de uma *random walk* partindo do nó u. As cores verde, vermelho e azul representam diferentes caminhos gerados pelo mesmo algoritmo. E os métodos baseados em *random walk* têm essa característica de fazer vastas amostragens no grafo para deixar mais eficiente o arendizado de características.

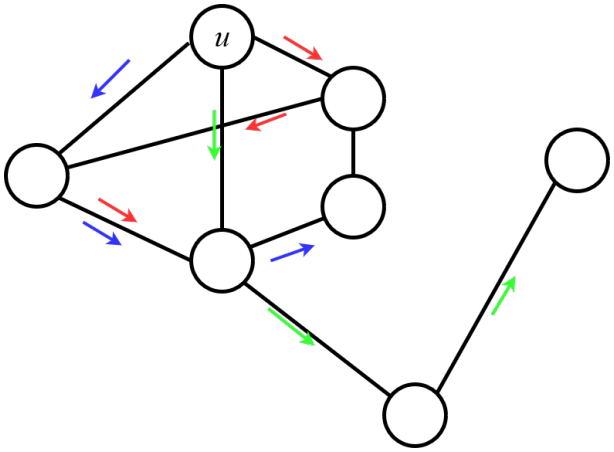


Figura 7 – Visão geral de *random walks*
Fonte: Autoria própria.

2.3.3 Métdos baseados em *deep learning*

O crescente uso de *deep learning* levou a que vários métodos baseados em *deep learning* fossem aplicados em grafos. O *Deep autoencoder* são usados para redução de dimensionalidade, devido a sua característica de modelar estruturas não lineares nos dados. *Structural Deep Network Embedding* (SDNE) (WANG; CUI; ZHU, 2016) e *Deep Neural Networks for Learning Graph Representations* (DNGR) (CAO; LU; XU, 2016) utilizam a sua habilidade de *deep autoencoder* para gerar um modelo de *embedding* que pode capturar as não linearidades nos grafos. *Graph Convolutional Networks* (GCN) iterativamente analisa o *embedding* dos vizinhos de um nó e usa a função obtida do *embedding* e o *embedding* da última iteração para obter um novo *embedding*.

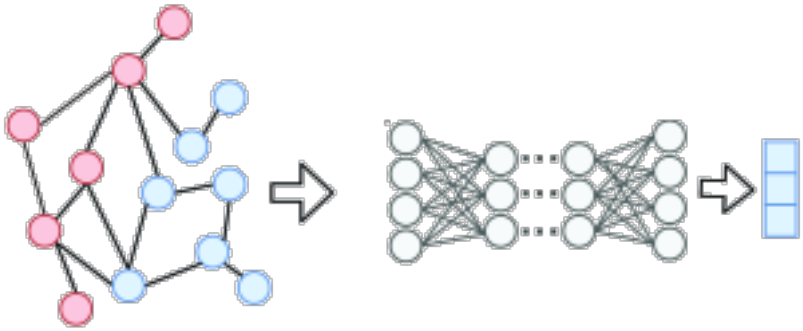


Figura 8 – Exemplo de *embedding* baseado em *deep learning*
Fonte: Adaptado de (YUE *et al.*, 2019).

Um exemplo dp funcionamento de *deep learning* para *graph emebedding* é mostrado na Figura 8 onde um grafo é usado como entrada em uma rede neural, e ela gera um vetor representando o *embedding* do grafo.

2.4 *NODE2VEC*

O algoritmo *node2vec* foi desenvolvido por (GROVER; LESKOVEC, 2016) para o aprendizado de características em grafos, esse método é baseado em *random walks* proposto no método *DeepWalk* (PEROZZI; AL-RFOU; SKIENA, 2014). No *DeepWalk* foi introduzido o conceito da arquitetura *Skip-gram* para o aprendizado de características no contexto da linguagem natural, sendo necessária uma adptação afim de usar tal arquitetura no *node2vec* cujo objetivo é o aprendizado de características no contexto de grafos. Feito isso, também foi necessário modificar as *random walks* para maximizar a extração de características em grafos, gerando o conceito de *biased random walk*. E essas *biased random walks* são usadas para o aprendizado de poderosas representações vetoriais dos nós de grafos.

2.4.1 *Skip-gram*

A arquitetura *skip-gram* foi inicialmente proposta em (MIKOLOV *et al.*, 2013; PEROZZI; AL-RFOU; SKIENA, 2014) para métodos de aprendizagem de características no contexto da linguagem natural. Onde cada palavra corrente (atual) é usada como entrada para um classificador linear, e então predizer as palavras a uma certa distância antes e depois da palavra corrente.

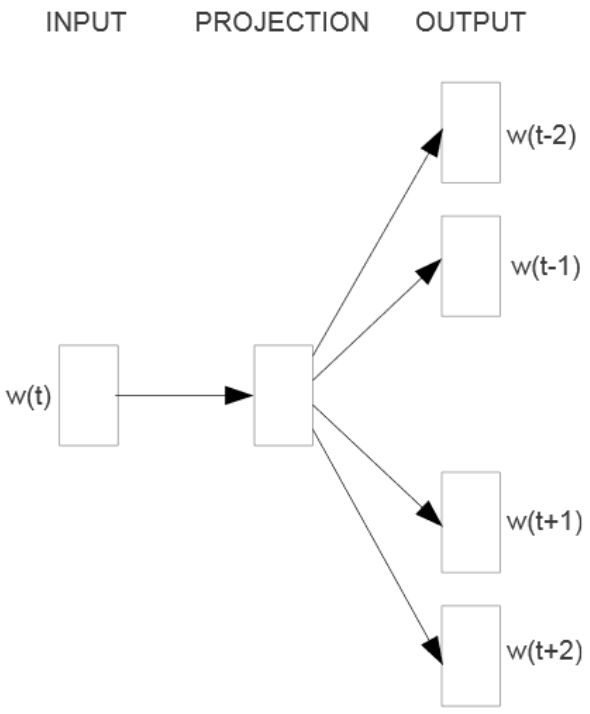


Figura 9 – Arquitetura Skpi-gram
Fonte: Adaptado de (MIKOLOV *et al.*, 2013).

Como mostrado na Figura 9, a entrada (palavra corrente) é $w(t)$. Há uma *hidden layer* a camada de projeção que efetua o produto vetorial entre a matriz ponderada e o vetor de entrada $w(t)$. E então o resultado do produto vetorial é passado para a camada de saída, onde é feito o produto vetorial entre o vetor de saída da *hidden layer* e a matriz ponderada da camada de saída. E assim é aplicada a função de ativação *softmax* para computar a probabilidade das palavras antes e depois de $w(t)$ que são $w(t-2)$, $w(t-1)$, $w(t+1)$ e $w(t+2)$.

2.4.2 Aprendizado de características em grafos

O aprendizado de características em grafos foi formulado como um problema de otimização para maximizar as probabilidades. Dado um grafo $G = (V,E)$. O modelo proposto é geral e aplicado a grafos (não)direcionados e (não)ponderados. Sendo $f : V \rightarrow \mathbb{R}^d$ a função de mapeamento dos nós para representação de características, o objetivo é determinar f a partir de uma tarefa de predição *downstream*. o parâmetro d especifica o número de dimensões da representação de características, ou seja, f é uma matriz de dimensão $\mid V \mid \times d$. Para cada nó de partida $u \in V$, define-se $N_S(u) \subset G$ como o grafo da vizinhança do nó u, gerado a partir de uma estratégia de amostragem de vizinhança S.

Também foi necessário extender o conceito da arquitetura *skip-gram* para grafos. Foi buscado otimizar a seguinte função objetiva (Equação 1), a qual maximiza a probabilidade-log de observar o grafo de vizinhança $N_S(u)$ do nó u dada a representação de características de tal vértice, representada por f:

$$\max_f \sum_{u \in V} logPr(N_S(u) \mid f(u)) \tag{1}$$

Para fazer o problema de otimização tratável é feita duas premissas:

- Independência condicional. É feita a fatorização das probabilidades assumindo que a probabilidade de observar um nó vizinho é independete de observar quaisquer outros vizinhos dada pela representação de características da origem:

$$logPr(N_S(u) \mid f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i \mid f(u)) \tag{2}$$

- Simetria no espaço de características. O nó de partida e os vizinhos desse nó têm um efeito de simetria sobre eles mesmos no espaço de características. A probabilidade condicional para cada par de vizinhança de origem dos nós foi modelada como uma *softmax unit* parametrizada pelo produto escalar de suas características:

$$Pr(n_i \mid f(u)) = \frac{exp(f(n_i) \cdot f(u))}{\sum_{v \in V} exp(f(v) \cdot f(u))} \tag{3}$$

Com as duas premissas de cima, a equação 1 é simplpificada para:

$$\max_{f} \sum_{u \in V} (-log(Z_u) + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u)) \tag{4}$$

A função de partição por nó, $Z_u = \sum_{n_i \in N_S(u)} exp(f(n_i) \cdot f(u))$, é computacionalmente cara para grafos muito grandes, então ea foi aproximada usando amostragem negativa. Desta forma a Equação 4 é otimizada usando o gradiente descendente estocástico (stochastic gradient descent ou SGD) sobre as parâmetros do modelo e das características f.

Os métodos de aprendizado de características baseados na arquitetura *skip-gram* foram inicialmente desenvolvidos no contexto do processamento de linguagem natural. E dada a natureza linear do texto, a noção de vizinhança é naturalmente definida usando um quadro deslizando sobre palavras consecutivas. Grafos, entretanto, não são lineares, e portanto uma noção mais completa de vizinhança é necessária. Para resolver esse problema, foi proposto o procedimento randômico que realiza amostragem de diferentes vizinhanças de um dado nó de origem u. As vizinhanças $N_S(u)$ não estão restritas a apenas vizinhos imediatos, mas sim de várias estruturas diferentes dependendo da estratégia de amostragem S.

2.4.3 Estratégias clássicas de busca

Foi apresentado o problema de amostragem de vizinhanças de um nó de partida como uma forma de busca local na subseção 2.4.2. Na Figura 10 é possível observar duas estratégias de buscas locais partindo do nó u, e com essas buscas é possível amostrar as vizinhanças $N_S(u)$. Para comparar as duas estratégias de amostragem S diferentes, deve-se restringir o tamanho do conjunto de vizinhança $N_S(u)$ para k nós e então realizar a amostragem várias vezes para o mesmo nó u. Há dois extremos em relação às estratégias de amostragem N_S:

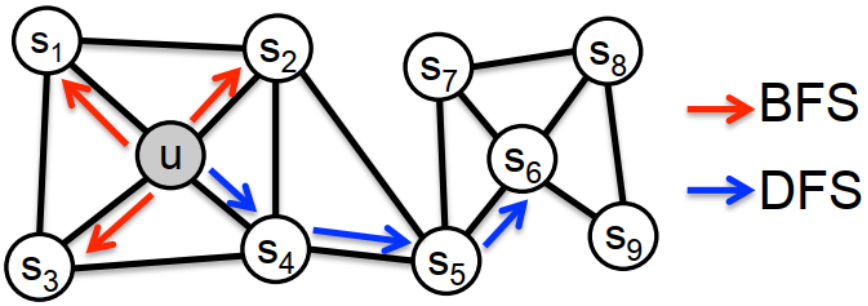


Figura 10 – Estratégias de busca BFS e DFS partindo do nó u (e $k = 3$)
Fonte: Retirado de (GROVER; LESKOVEC, 2016).

- *Breadth-first sampling* (BFS) A vizinhança N_S é restrita a nós que são vizinhos imediatos da origem, e olhar para a Figura 10 tal propriedade pode ser verificada, observando

o caminho da busca (setas em vermelho) e os nós visitados são todos vizinhos do nó de origem u ($N_S = \{s_4, s_5, s_6\}$).

- *Depth-first sampling* (DFS)) A vizinhança consiste em nós sequencialmente amostrados a uma distância crescente da origem ($N_S = \{s_1, s_2, s_3\}$), na Figura 10 o caminho da DPS (setas em azul) tendem a se afastar de u a cada iteração.

As estratégias de *breadth-first* e *depth-first sampling* representam cenários opostos em termos do espaço de busca que eleas exploram (a BFS percorre nós estruturalmente próximos, enquanto a DFS explora partes mais distantes do grafo) o que leva à implicações interessantes no que diz respeito ao aprendizado de representações.

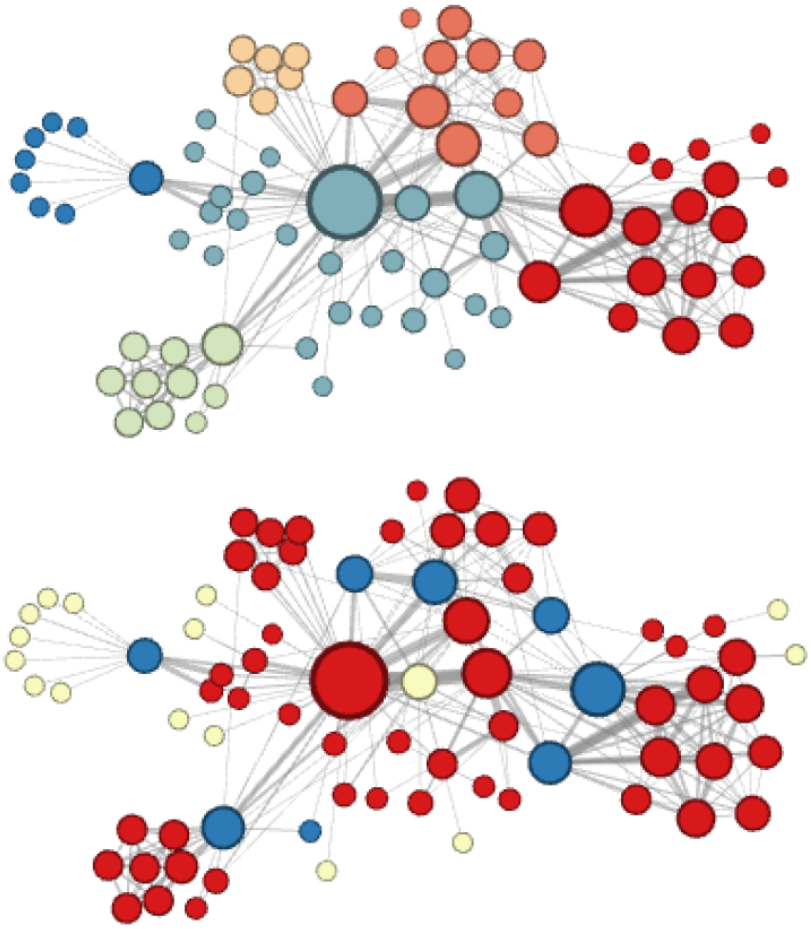


Figura 11 – Visualização do grafo *Les Misérables* com as cores refletindo *homophily* (em cima) e *structural equivalence* (em baixo)
Fonte: Adaptado de (GROVER; LESKOVEC, 2016).

Particularmente as tarefas de predição nos nós do grafo normalmente alternam-se entre dois tipos de similaridade: *homophily* e *structural equivalence* (HOFF; RAFTERY; HANDCOCK, 2002). Sob a hipótese *homophily* (YANG; LESKOVEC, 2014) nós altamente interconectados e pertencentes a *clusters* ou comunidades (vizinhança) similares do grafo devem ter *embedded*

semelhantes. Um exemplo disso é na Figura 11, os nós localmente próximos (mesmas vizinhanças) recebem as mesmas cores, ou seja tendo uma representação do *embedded*. Em contrapartida, sob a hipótese de *structural equivalence* (HENDERSON *et al.*, 2012) nós que tenham papéis estruturais semelhantes nos grafos devem ser *embedded* juntos, ou seja a *structural equivalence* analisa os nós com a estrutura de arestas parecidas. Exemplo na Figura 11 os nós usados como pontes (com muitas arestas passando por ele) são *embedded* da mesma forma (cor azul), os nós com poucas conexões rececbem outro *embedding* (cor amarela) e, por fim, os nós conectado a vários outros vértices recebem outra representação (a cor vermelha). Vale ressaltar que diferente da *homophily*, a *structural equivalence* não enfatiza a conectividade; os vértices podem estar distantes entre si na rede, mas ainda ter o mesmo papel estrutural. No mundo real, essas noções de equivalência não são mutualmente exclusivas; grafos frequentemente exibem ambos os comportamentos onde alguns nós exibem *homophily* enquanto outros reflem o comportamento de *structural equivalence*.

2.4.4 Formalização do *node2vec*

O *node2vec* foi desenvolvido para fazer amostragens da vizinhança de modo flexível alternando entre BFS e DFS, e tal objetivo é obtido por meio de *biased random walk* que explorar a vizinhança no modo de BFS assim como DFS.

2.4.4.1 *Random Walks*

Dado um nó inicial u, uma simulação de *random walk* de tamanho fixo l é feita. Sendo c_i o i-ésimo nó da *walk*, partindo de $c_0 = u$. Os nós c_i são gerados pela distribuição na Equação 5, onde π_{vx} é a probabilidade de transição não normalizada entre os nós v ex, e Z é a constante de normalização.

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z}, & \text{se } (v,x) \in E \\ 0, & \text{caso contrário} \end{cases} \tag{5}$$

Na Figura 12 é mostrado um exemplo de *random walk*, cada transição em azul mostra o peso relacionado à aresta e a probabilidade de transição da escolha do próximo passo da *walk*.

2.4.4.2 *Search bias* α

O modo mais simples de fazer uma *biased random walk* seria amostrar o próximo nó baseado no peso das arestas w_{vx}, por exemplo $\pi_{vx} = w_{vx}$. Entretanto tal modo não levaria em conta a estrutura do grafo e levaria a busca à explorar diferentes vizinhanças no grafo. Adicionalmente, diferentemente da BFS e da DFS que são paradigmas de amostragem adequados

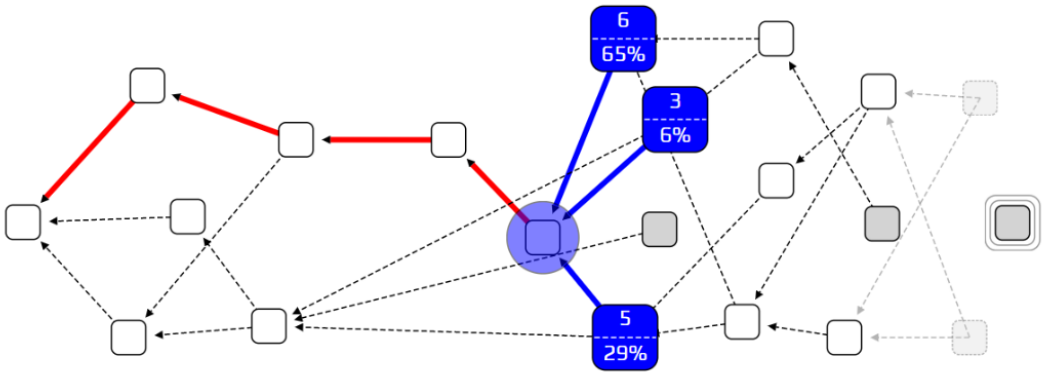


Figura 12 – Exemplo de *random walk*
Fonte: Retirado de (GAL, 2018).

para *structural equivalence* e *homophily* respectivamente, e as *random walks* não devem se acomodar com essas noções de equivalência, pois grafos do mundo real normalmente exibem uma mistura de ambas.

Uma *random walk* de segunda ordem é defina em (GROVER; LESKOVEC, 2016) com dois parâmetros p e q que guiam a *walk*: considerando a *random walk* que acaba de curzar a aresta (t, v) e atualmente está no nó v (Figura 13). A *walk* agora precisa decidir o próximo passo, e então as probabilidades de transição π_{vx} são avaliadas nas arestas (v, x) partindo de v. Logo define-se a probabilidade de transição não normalizada como $\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx}$, onde $\alpha_{pq}(t,x)$ (segundo a Equação 6) e d_{tx} denotando o menor caminho entre os nós t e x. Observa-se que d_{tx} deve ser 0, 1 ou 2, e consequentemente os dois parâmetros (p e q) são necessários e suficientes para guiar a *walk*.

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p}, & \text{se } d_{tx} = 0 \\ 1, & \text{se } d_{tx} = 1 \\ \frac{1}{q}, & \text{se } d_{tx} = 2 \end{cases} \tag{6}$$

Intuitivamente nota-se que os parâmetros p e q controlam o quão rápido a *walk* explora e deixa a vizinhança do nó inicial u. Mais precisamente, os parâmetros que o procedimento de busca definido interpole entre um BFS e uma DFS e assim torna o procedimento ter mais afinidade para diferentes noções de equivalência de vértices.

2.4.4.2.1 Parâmetro de retorno.

O parâmetro p controla a probabilidade de imediatamente revisitar a nó em uma *walk*. Setá-lo com um alto valor ($> \max(q,1)$) garante que é menos provável de amostrar um nó já visitado nos próximos dois passos (a menos que o próximo vértice não tenha vizinho). Essa estratégia incentiva uma moderada exploração do grafo e evita redundância *2-hop* na amostra-

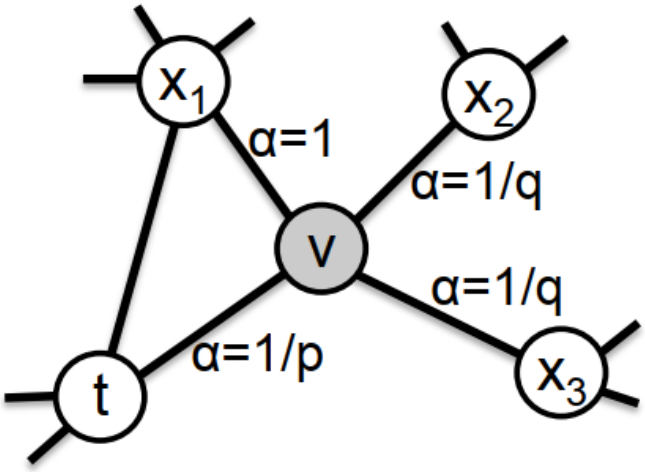


Figura 13 – Ilustração do procedimento da *random walk* no *node2vec*
Fonte: Retirado de (GROVER; LESKOVEC, 2016).

gem. Caso contrário - se q é pequeno ($< \min(q,1)$) - é mais provável que a *walk* retroceda um passo e mantém a *walk* localmente próxima do nó de partida u.

2.4.4.2.2 Parâmetro de entrada-saída.

O parâmetro q permite que busca diferencie os nós "internos"e "externos". Olhando a figura 13, se $q > 1$, a *random walk* é mais provável de explorar os nós próximos de t.Tais *walks* obtêm uma visão local do grafo subjacente ao nó de partida, e a *walk* se aproxima do comportamento da BFS no sentido de que as amostragens abrangem os nós localmente próximos.

Em contraposição, se $q < 1$, a *walk* é inclinada a visitar nós mais distantes do nó t. Tal comportamento corresponde ao da DFS que incentiva a exploração externa. Entretanto uma diferença essencial é que a exploração ao modo de DFS é obtida usando o método de *random walk*. Portanto, os nós amostrados não estão necessariamente aumentado de distância do nó de início u, mas por sua vez, há o benefício de um pré-processamento computacionalmente tratável e uma amostragem com eficiência superior das *random walks*.

2.4.4.2.3 Benefícios das random walks.

Há muitas vantagens das *random walks* sobre as abordagens BFS e DFS. As *random walks* são computacionalmente eficientes tanto quanto aos requisitos de espaço, quanto de tempo. A complexidade de espaço para armazenar os vizinhos imediatos de cada vértice do grafo é $O(\mid E \mid)$. Para as *random walks* de segunda ordem, é útil armazenar as interconexões (arestas) dos vizinhos de cada nó, o que resulta em uma complexidade de espaço de $O(a^2 \mid V \mid)$ onde a é o grau médio do grafo e normalmente é pequeno para grafos do mundo real. Outra vantagem chave das *random walks* é a comlexidade de tempo Particularmente, ao

Algoritmo 1 – O algoritmo *node2vec*

impor a conectividade do grafo no processo de gerar as amostragens, *random walks* fornecem um mecanismo conveniente para aumentar a eficiência da taxa de amostragem por reusar amostragens entre diferentes nós de partida.

2.4.5 O algoritmo *node2vec*

Algoritmo 1: O algoritmo *node2vec*

Input: *Graph* $G = (V,E,W)$, *Dimensions* d, *Walks per node* r, *Walk length* l, *Context size* k, *Return parameter* p, *In-out parameter* q
Output: neural network weights
$\pi = PreprocessModifiedWeights(G, p, q)$;
$G' = (V, E, \pi)$;
$walks = \{\ \}$;
for *iter in r* **do**
  forall *nodes* $u \in V$ **do**
    $walk = node2vecWalk(G',u,l)$;
    Append $walk$ *to* $walks$;
  end
end
$f = StochasticGradientDescent(k, d, walks)$;
return f;

Fonte: Retirado de (GROVER; LESKOVEC, 2016).

Procedimento $node2vecWalk$(G',u,l)

Input: *Graph* $G' = (V,E,\pi)$, *Start node* u, *Length* l
Output: biased random walk
$walk = [u]$;
for *walk_iter in l* **do**
  $curr = walk\,[walk\_iter - 1]$;
  $V_{curr} = GetNeighbors(curr,G')$;
  $s = AliasSample(V_{curr}, \pi)$;
  Append s *to* $walk$;
end
return $walk$;

Fonte: Retirado de (GROVER; LESKOVEC, 2016).

O algoritmo do *node2vec* é mostrado no Algoritmo 1. Em cada *random walk* há um *bias* implítico devido à escolha do nó de início u. Uma vez que aprenda-se representações para todos os nós, esse *bias* é compensado ao simular r *random walks* de comprimento fixo l começando de todo vértice. Em cada passo da *walk*, a amostragem é feita baseada na probabilidade de transição π_{vx}. A probabilidade de transição π_{vx} para a segunda ordem da cadeia de Markov pode ser pré-computada, e então amostrar os nós enquanto simula a *random walk* é

feita eficientemente em $O(1)$ usando pseudo-amostragens. As três estapas do *node2vec*, o pré-processamento para computar as transições de probabilidades, as simulações de *random walks* e a otimização usando SGD, são executadas sequencialmente. E cada etapa é paralelizável e pode ser executada assincronamente, contribuindo para a escalabilidade geral do *node2vec*

2.4.6 Aprendizado das características das arestas

O algoritmo *node2vec* provê um método semi-supervisionado para aprender ricas representações de características para os vértices em um grafo. Entretanto, frequentemente há a necessidade de tarefas de predição envolvendo pares de vértices. Por exemplo, em *link prediction* a tarefa é determinar se existe uma aresta entre dois nós de um grafo. Desde que as *random walks* sejam naturalmente baseadas na strutura de conectivade entre os nós do grafo subjacente, é possível extender tais *random walks* para pares de nós usando uma abordagem de *bootstrapping* ao invés da representação de características dos nós individuais.

Tabela 1 – Escolha dos operadores binários $\circ$ para o aprendizado de características das arestas. As definições correspondem ao i-ésimo termo de $g(u,v)$

Operador	Símbolo	Definição
Média	$\boxplus$	$[f(u) \boxplus f(v)]_i = \frac{f_i(u)+f_i(v)}{2}$
Hadamard	$\boxdot$	$[f(u) \boxdot f(v)]_i = f_i(u) * f_i(v)$
Weighted-L1	$\|\cdot\|_{\overline{1}}$	$\|f(u) \cdot f(v)\|_{\overline{1}i} = \|f_i(u) - f_i(v)\|$
Weighted-L2	$\|\cdot\|_{\overline{2}}$	$\|f(u) \cdot f(v)\|_{\overline{2}i} = \|f_i(u) - f_i(v)\|^2$

Dado dois nós u e v, define-se o operador binário $\circ$ sobre os correspondentes vetores de características $f(u)$ e $f(v)$a fim de gerar uma representação $g(u,v)$ tal que $g : V \times V \rightarrow \mathbb{R}^{d'}$ onde d' é a representação do tamanho do par (u, v). è ais adequado que o operador $\circ$ seja definido de forma geral para qualquer par de nó, mesmo que uma aresta não exista entre o par, visto que ao fazer isso torna tal representação útil para *link prediction* onde os conjuntos de teste têm tanto arestas verdadeiras quanto falsas. As opções escolhidas para o operador $\circ$ levando em conta as condições anteriores e que $d' = d$ estão exibidas na Tabela 1.

3 MATERIAIS E MÉTODOS

Afim de detalhar o desenvolvimento deste trabalho, este capítulo relata os métodos e ferramentas empregadas na pesquisa.

3.1 MATERIAIS

No desenvolvimento desta pesquisa, o ambiente *Google Colab* e a linguaguem *Python* foram utilziados para o processamento de dados, treinamento de modelos e para a tarefa de *link prediction*.

No escopo desse estudo, para criar o grafo de citações foi utilizado a base dado *CORA* (SEN *et al.*, 2008) que é uma base composta por artigos científicos relacionados à aprendizagem de máquina. Cada artigo presente nesse banco de dados será representado por um nó do grafo, e as arestas são as citações dos artigos entre si. A Figura 14 representa, de maneira visual, o grafo deste banco de dados.

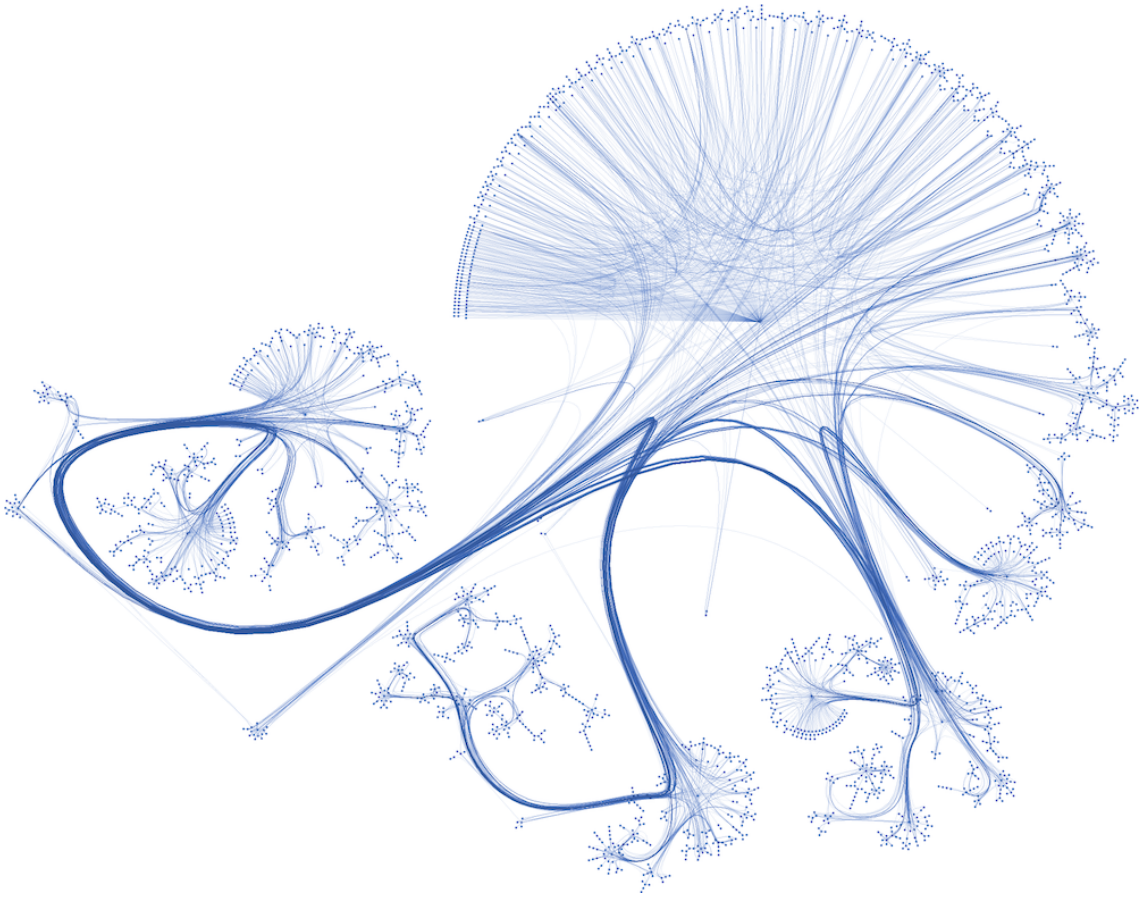


Figura 14 – Visualização do grafo referente ao banco de dados *CORA*.
Fonte: https://graphsandnetworks.com/the-cora-dataset/

3.2 MÉTODO

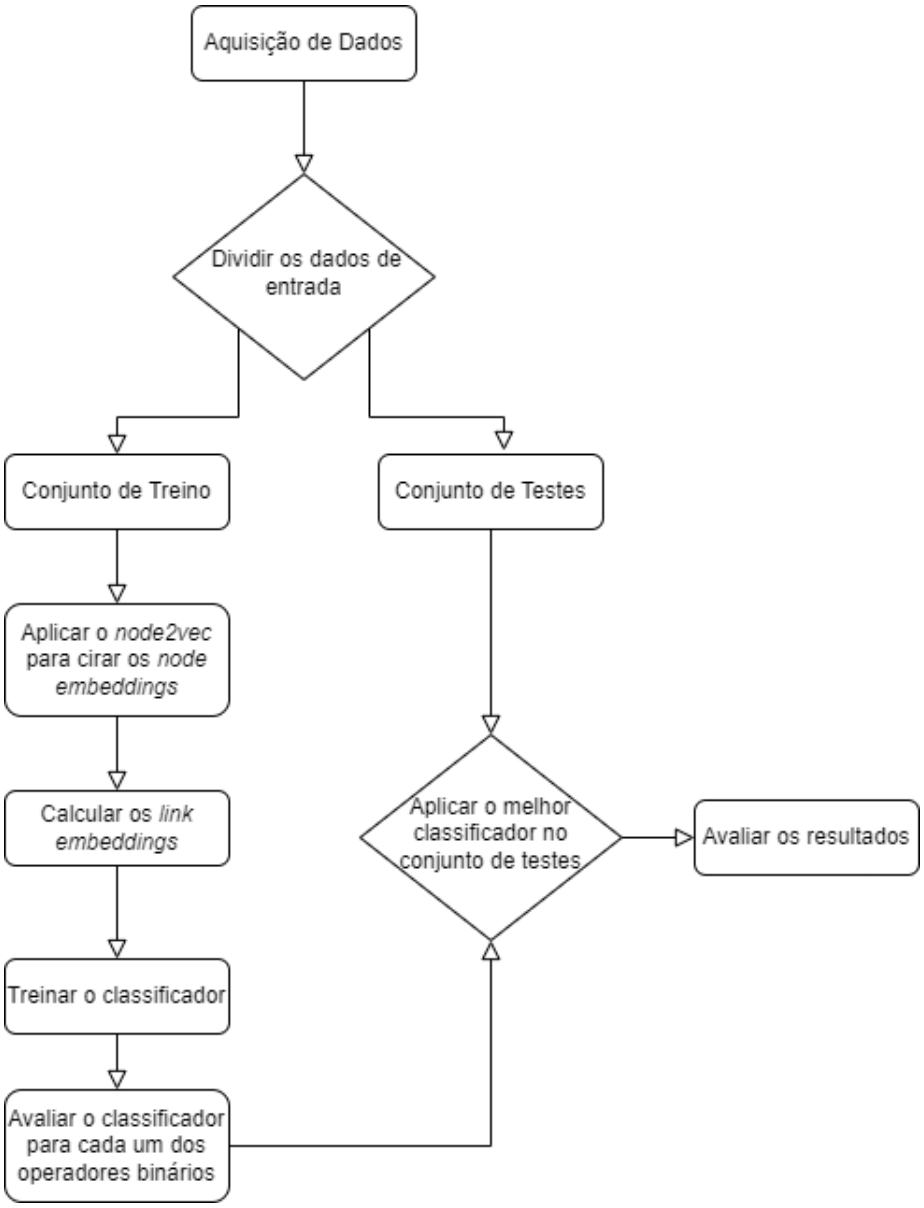


Figura 15 – Diagrama de metodologia aplicada na pesquisa.
Autoria própria.

Na Figura 15 é possível verificar quais foram os passos de desenvolvimento deste estudo, descritos e abordados detalhadamente a seguir.

3.2.1 Aquisição de dados

A aquisição da base de dados é feita usando a biblioteca *StellarGraph* no *Python*, nessa biblioteca contém a base de dados *CORA*. A tabela possui 2485 nós e 5209 arestas.

3.2.2 Dividir os dados de entrada

É preciso dividir cuidadosamente a base de dados original em treino e teste para evitar que haja vazamento de dados e para avaliar os algoritmos e classificadores corretamente. Esta divisão se dá em base de teste e base de treino.

A base de teste é usada para a avaliação final do modelo, nela temos o Grafo de Teste e o Conjunto de Teste. O Grafo de teste é usado para computar o *embedding* dos nós de teste e possui mais arestas que o Grafo de Treino. O Grafo de teste é subgrafo do Grafo Inicial, onde são removidas as arestas do grafo original e essas arestas verdadeiras junto com arestas falsas são representados como o Conjunto de Teste.

A base treino é composta pelo Grafo de Treino, Conjunto de Treino e Conjunto de Seleção de Modelo. O primeiro é usado para computar o *embedding* dos nós. O Grafo de Treino é um subgrafo do Grafo de Testes, onde são removidas arestas do Grafo de Testes e essas arestas verdadeiras junto com arestas falsas compõem o Conjunto de Treino e Conjunto de Seleção de Modelo.

Na Tabela 2 é mostrado como estão divididos os dados, como explicado nesta seção. O número de amostras é quantidade de arestas verdadeiras mais arestas falsas.

Tabela 2 – Divisão da base de entrada

Divisão	Número de Amostras	Representado em	Gerado de
Conjunto de Treino	702	Grafo de Treino	Grafo de Teste
Seleção de Modelo	234	Grafo de Treino	Grafo de Teste
Conjunto de Testes	1040	Grafo de Testes	Grafo Original

3.2.3 Aplicar o *node2vec* para criar os *node embeddings*

É usado o *node2vec* para criar o *embedding* dos nós. Essa representação é aprendida de certa forma que garante que os nós que são próximos no Grafo, também são próximos no seu *embedding*. Os parâmetros para a execução do algoritmo *node2vec* (o algoritmo é descrito na subseção 2.4.5) são mostrados em Tabela 3.

Tabela 3 – Parâmetros do algoritmo *node2vec* no treino

Parâmetro	Valor	Descrição
G	Grafo de Treino	Grafo
d	128	dimensionalidade do *embedding*
r	10	*walks* por nó
l	80	tamanho da *walk*
k	10	tamanho do contexto do SGD
p	1.0	parâmetro de retorno
q	1.0	parâmetro entrada-saída

3.2.4 Calcular os *link embeddings*

Nessa etapa é calculado o *link embedding* para as amostras das arestas positivas e negativas (Conjunto de Treino) ao aplicar os operadores binários no *embedding* dos nós de origem e destino para cada aresta na amostra. Esse é o processo de aprendizado de características das arestas mostrado na subseção 2.4.6 (os operadores binários também são descritos nessa subseção)

3.2.5 Treinar o classificador

O classificador escolhido no estudo é o *LogisticRegressionCV* integrado na liblioteca *sklearn* do *Python*. O *LogisticRegressionCV* é uma classe que implementa uma cross-validação entre si, ou seja, são treinados múltiplos modelos *LogisticRegression* e é retornado o melhor entre eles. O *fit* desse classificador é feito com o vetor de aprendizado de características das arestas (obtido na subseção 3.2.4) e o vetor da existência ou não daquela aresta (obtido das amostras do Conjunto de Treino).

3.2.6 Avaliar o classificador para cada um dos operadores binários

Nesta estapa é avaliado o melhor classificador gerado a partir de cada operador binário. A métrica usada para obter a qualidade de um classificador é o Valor ROC AUC, quanto mais próximo esse valor for de 1, melhor o modelo é capaz de realizar predições. A tabela comparativa dos modelos por operador usando a métrica Valor ROC AUC é mostrada em Tabela 4, e com base nela o modelo treinado com o operador *Weighted-L2* foi o escolhido.

Tabela 4 – Avaliação dos classificadores por operador binário

Operador	**Valor ROC AUC**
média	0.558787
hadamard	0.802192
Weighted-L1	0.830617
Weighted-L2	0.833760

4 RESULTADOS

Afim de criar o modelo para *link prediction* foi construído os *node embeddings* e *link embeddings* para o Grafo de Treino e com esses *embeddings* são criados modelos a partir de cada operador binário. O operador que melhor satisfez a condição de avaliação foi o *Weighted-L1* e este modelo escolhido para ser aplicado no Grafo de Teste e avaliar sua viabilidade para *link prediction*.

E então é aplicado o conjunto de testes no modelo escolhido, por fim aplicada a métrica de avaliação de qualidade de predição de *link* no modelo. O valor ROC AUC é 0,9193417159763314. Também é possível obter uma visualização da qualidade da predição na Figura 16, ao fazer uma representação das arestas verdadeiras (ou verdadeiros positivos que são os pontos azuis do gráfico) e arestas falsas (ou verdadeiros negativos que são os pontos vermelhos do gráfico), poucos pontos pontos azuis coincidindo com pontos vermelhos indicam uma boa qualidade de predição, pois arestas que devem existir são classificadas como Verdadeiro pelo modelo, enquanto arestas que não deveriam existir são classificadas como Falso pelo modelo. A Figura 16 foi criada por meio do algoritmo PCA que recebeu como número de componentes 2 e para o fit o valor foi o *embedding* das características das arestas (que é uma representação de dimensão 128).

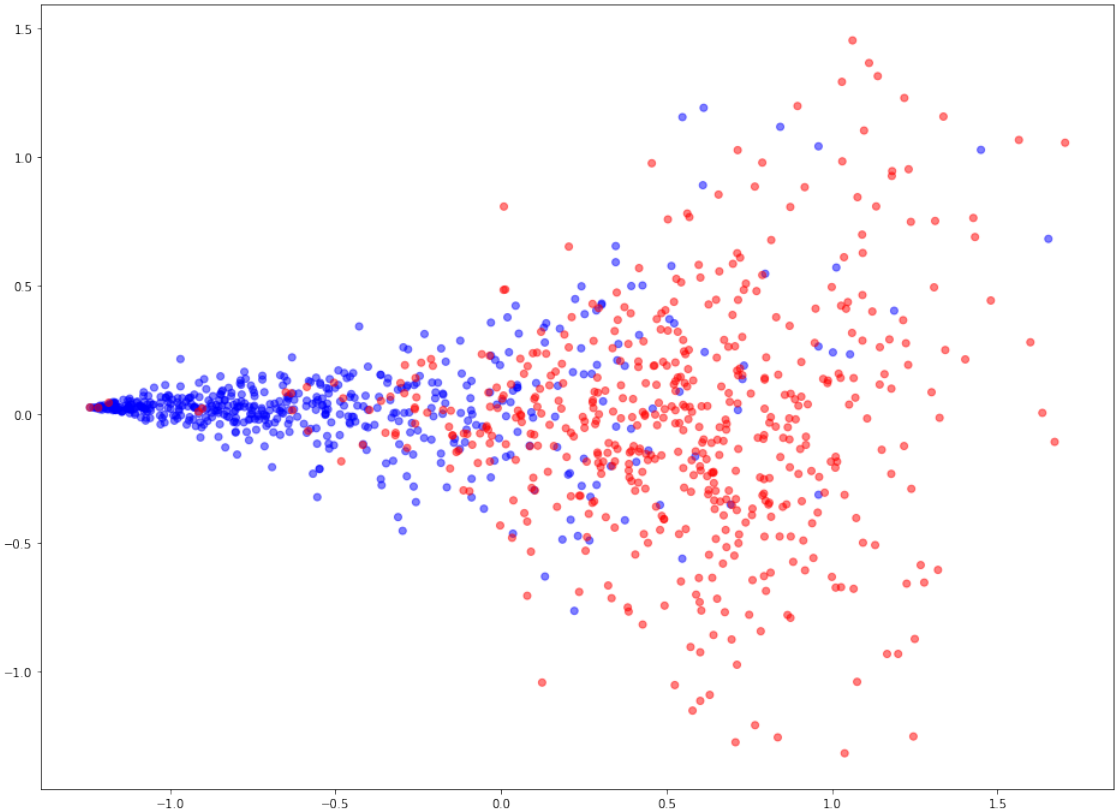


Figura 16 – Visualização do aprendizado de características das arestas
Autoria própria.

O modelo se prova satisfatório para *link prediction*, tendo em vista a métrica de avaliação utilizada. Contudo, este modelo tem carências como a aplicação desse modelo em uma base maior, o que iria escalar o problema computacional e o fator tempo para gerar o modelo, assim como realizar a predição e aprendizagem de características. Outro ponto de observação para esse modelo é a falta de levar em consideração o conteúdo dos artigos na modelagem do sistema de aprendizagem, esta seria uma das melhorias que devem ser abordadas para obter um modelo que tenha uma acurácia melhor.

Ao comparar o modelo com os já criados na literatura como em (PAN *et al.*, 2015) é possível identificar que o modelo desenvolvido neste não agrega para si fatores como os os dados dos nós do grafo e que são importes para a classificação do assunto de uma publicação, o modelo desenvolvido também não leva em conta o direcionamento das arestas dos grafos de publicações de artigos, e é requierido mais estudo em como esses fatores podem afetar a qualidade da predição.

5 CONCLUSÃO

Prever a existência da relação entre dois artigos correlacionados em uma base de citação de artigos científicos é uma significante tarefa de análise, e para isso é preciso aplicar conceitos de aprendizado de máquina em grafos, o que introduziu este trabalho a vários métodos de *graph embedding* que são métodos para o aprendizado de representações de baixa dimensionalidade de grafos, desta forma aplica-se algoritmos de *machine learning* nesses *embeddings*. Entre os algoritmos de graph embedding disponíveis o escolhido foi o *node2vec*, um poderoso método semi-supervisionado para o aprendizado e características dos nós de um grafo.

Para realizar um estudo de caso do algoritmo *node2vec* na criação de um modelo de aprendizado de máquina capaz de prever a correlação entre artigos em um grafo de citações de artigos acadêmicos, foram necessárias boas práticas em se tratando de métodos de aprendizado semi-supervisionado, e dividir a base original de forma que não houvesse contaminação entre os modelos treinados. Também foi necessário ajustar os hiperparâmetros do algoritmo *node2vec* e ter uma boa métrica da qualidade da predição do modelo treino, o valor ROC AUC. E após o trabalho exaustivo de treinar e comparar vários modelos treinados, um deles se sobressaiu e foi capaz de ter uma acurácia satisfatória na tarefa de *link prediction* em grafos de publicações científicas.

Este trabalho pode ser a base para a criação de modelos de *link prediction* em trabalhos futuros visto que, as etapas necessárias para a criação do modelo e passos para o treino deste modelo foram abordados de forma clara, vale ressaltar, também, que existem diversos métodos de *graph embedding* e cada um deles vale um estudo de caso para verificar as suas eficiências nas mais diversas tarefas de compresão, visualização, clusterização, *link prediction* ou *node classification*. Também foi apresentado que o modelo tem limitações, sendo a principal delas levar em conta apenas as citações entre os artigos e não o conteúdo da publicação. Desta forma, mostrando que criar um bom classificador para esse problema ainda exige que mais métodos sejam testados e se fazer uma comparação entre eles.

REFERÊNCIAS

AHMED, A. *et al.* Distributed large-scale natural graph factorization. *In*: ACM. **Proceedings of the 22nd international conference on World Wide Web**. [*S.l.*], 2013. p. 37–48.

BATTISTA, G. D. *et al.* Algorithms for drawing graphs: an annotated bibliography. **Computational Geometry**, Elsevier, v. 4, n. 5, p. 235–282, 1994.

BELKIN, M.; NIYOGI, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. *In*: **Advances in neural information processing systems**. [*S.l.*: *s.n.*], 2002. p. 585–591.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: A review and new perspectives. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 35, n. 8, p. 1798–1828, 2013.

BHAGAT, S.; CORMODE, G.; MUTHUKRISHNAN, S. Node classification in social networks. *In*: **Social network data analytics**. [*S.l.*]: Springer, 2011. p. 115–148.

CAO, S.; LU, W.; XU, Q. Deep neural networks for learning graph representations. *In*: **Thirtieth AAAI Conference on Artificial Intelligence**. [*S.l.*: *s.n.*], 2016.

CHANG, S. *et al.* Heterogeneous network embedding via deep architectures. *In*: ACM. **Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [*S.l.*], 2015. p. 119–128.

EADES, P.; XUEMIN, L. How to draw a directed graph. *In*: IEEE. **[Proceedings] 1989 IEEE Workshop on Visual Languages**. [*S.l.*], 1989. p. 13–17.

FEDER, T.; MOTWANI, R. Clique partitions, graph compression, and speeding-up algorithms. *In*: CITESEER. **Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing**. [*S.l.*], 1991. p. 123–133.

GAL, A. **The Tangle: an illustrated introduction. Part 3: Cumulative weights and weighted random walks**. 2018. Disponível em: https://blog.iota.org/the-tangle-an-illustrated-introduction-f359b8b2ec80.

GOYAL, P.; FERRARA, E. Graph embedding techniques, applications, and performance: A survey. **Knowledge-Based Systems**, Elsevier, v. 151, p. 78–94, 2018.

GROVER, A.; LESKOVEC, J. node2vec: Scalable feature learning for networks. *In*: ACM. **Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining**. [*S.l.*], 2016. p. 855–864.

HASAN, M. A.; ZAKI, M. J. A survey of link prediction in social networks. *In*: **Social network data analytics**. [*S.l.*]: Springer, 2011. p. 243–275.

HENDERSON, K. *et al.* Rolx: structural role extraction & mining in large graphs. *In*: ACM. **Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining**. [*S.l.*], 2012. p. 1231–1239.

HERMAN, I.; MELANÇON, G.; MARSHALL, M. S. Graph visualization and navigation in information visualization: A survey. **IEEE Transactions on visualization and computer graphics**, IEEE, v. 6, n. 1, p. 24–43, 2000.

HOFF, P. D.; RAFTERY, A. E.; HANDCOCK, M. S. Latent space approaches to social network analysis. **Journal of the american Statistical association**, Taylor & Francis, v. 97, n. 460, p. 1090–1098, 2002.

JR, D. W. H.; LEMESHOW, S.; STURDIVANT, R. X. **Applied logistic regression**. [*S.l.*]: John Wiley & Sons, 2013. v. 398.

KOVÁCS, I. A. *et al.* Network-based prediction of protein interactions. **Nature communications**, Nature Publishing Group, v. 10, n. 1, p. 1240, 2019.

LESKOVEC, J. **Graph Representation Learning**. 2017. Disponível em: https://www.slideshare.net/jleskovec/graph-representation-learning.

LIBEN-NOWELL, D.; KLEINBERG, J. The link-prediction problem for social networks. **Journal of the American society for information science and technology**, Wiley Online Library, v. 58, n. 7, p. 1019–1031, 2007.

LIOUNG, H. **Lecture 12 Graph Mining**. 2014. Disponível em: https://www.slideshare.net/HouwLiong/lect12-graph-mining.

LÜ, L.; ZHOU, T. Link prediction in complex networks: A survey. **Physica A: statistical mechanics and its applications**, Elsevier, v. 390, n. 6, p. 1150–1170, 2011.

LUO, D. *et al.* Cauchy graph embedding. *In*: **Proceedings of the 28th International Conference on Machine Learning (ICML-11)**. [*S.l.*: *s.n.*], 2011. p. 553–560.

MCCALLUM, A.; NIGAM, K. *et al.* A comparison of event models for naive bayes text classification. *In*: CITESEER. **AAAI-98 workshop on learning for text categorization**. [*S.l.*], 1998. v. 752, n. 1, p. 41–48.

MIKOLOV, T. *et al.* Efficient estimation of word representations in vector space. **arXiv preprint arXiv:1301.3781**, 2013.

NAVLAKHA, S.; RASTOGI, R.; SHRIVASTAVA, N. Graph summarization with bounded error. *In*: ACM. **Proceedings of the 2008 ACM SIGMOD international conference on Management of data**. [*S.l.*], 2008. p. 419–432.

NOZZA, D.; FERSINI, E.; MESSINA, E. Cage: Constrained deep attributed graph embedding. **ainda nao foi publicado**, ainda nao foi publicado, ainda nao foi publicado, n. ainda nao foi publicado, p. ainda nao foi publicado, 2019.

OU, M. *et al.* Asymmetric transitivity preserving graph embedding. *In*: ACM. **Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining**. [*S.l.*], 2016. p. 1105–1114.

PAN, L. *et al.* Academic paper recommendation based on heterogeneous graph. *In*: **Chinese computational linguistics and natural language processing based on naturally annotated big data**. [*S.l.*]: Springer, 2015. p. 381–392.

PEROZZI, B.; AL-RFOU, R.; SKIENA, S. Deepwalk: Online learning of social representations. *In*: ACM. **Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining**. [*S.l.*], 2014. p. 701–710.

ROWEIS, S. T.; SAUL, L. K. Nonlinear dimensionality reduction by locally linear embedding. **science**, American Association for the Advancement of Science, v. 290, n. 5500, p. 2323–2326, 2000.

SEN, P. *et al.* Collective classification in network data. **AI magazine**, v. 29, n. 3, p. 93–93, 2008.

SHAW, B.; JEBARA, T. Structure preserving embedding. *In*: ACM. **Proceedings of the 26th Annual International Conference on Machine Learning**. [*S.l.*], 2009. p. 937–944.

TANG, J. *et al.* Line: Large-scale information network embedding. *In*: INTERNATIONAL WORLD WIDE WEB CONFERENCES STEERING COMMITTEE. **Proceedings of the 24th international conference on world wide web**. [*S.l.*], 2015. p. 1067–1077.

VENTO, M. **Graph Matching and Learning in Pattern Recognition in the Last 10 Years**. 2014. Disponível em: https://www.researchgate.net/figure/An-example-of-graph-clustering-in-the-second-meaning-graph-based-clustering-the_fig4_262524340.

WANG, D.; CUI, P.; ZHU, W. Structural deep network embedding. *In*: ACM. **Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining**. [*S.l.*], 2016. p. 1225–1234.

WANG, D. D.; WANG, R.; YAN, H. Fast prediction of protein–protein interaction sites based on extreme learning machines. **Neurocomputing**, Elsevier, v. 128, p. 258–266, 2014.

WU, Z. *et al.* A comprehensive survey on graph neural networks. **arXiv preprint arXiv:1901.00596**, 2019.

YANG, J.; LESKOVEC, J. Overlapping communities explain core–periphery organization of networks. **Proceedings of the IEEE**, IEEE, v. 102, n. 12, p. 1892–1902, 2014.

YUE, X. *et al.* Graph embedding on biomedical networks: Methods, applications, and evaluations. **arXiv preprint arXiv:1906.05017**, 2019.