Instituto de MATEMÁTICA E ESTATÍSTICA UFRGS

DEPARTAMENTO DE ESTATÍSTICA
E-mail: dest@mat.ufrgs.br

Trabalho de Conclusão de Curso

Predição do Número de Alunos por Municípios do Projeto "Saúde com Agente" Utilizando Redes Neurais de Grafos (GNN)

Letícia Maria Puttlitz

26 de fevereiro de 2024

Letícia Maria Puttlitz

Predição do Número de Alunos por Municípios do Projeto "Saúde com Agente" Utilizando Redes Neurais de Grafos (GNN)

Trabalho de Conclusão apresentado à comissão de Graduação do Departamento de Estatística da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para obtenção do título de Bacharel em Estatística.

Orientadora: Profa. Dra. Márcia Helena Barbian

Porto Alegre
9 de Fevereiro de 2023

Letícia Maria Puttlitz

Predição do Número de Alunos por Municípios do Projeto "Saúde com Agente" Utilizando Redes Neurais de Grafos (GNN)

Este Trabalho foi julgado adequado para obtenção dos créditos da disciplina Trabalho de Conclusão de Curso em Estatística e aprovado em sua forma final pela Orientadora e pela Banca Examinadora.

Orientadora:_____
Profa. Dra. Márcia Helena Barbian, UFMG
Doutora pela Universidade Federal de Minas Gerais, Belo Horizonte, MG

Banca Examinadora:

Prof. Dr. João Henrique Ferreira Flores, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul, Porto Alegre, RS

Prof. Dr. Anderson Rocha Tavares, UFMG
Doutor pela Universidade Federal de Minas Gerais

Porto Alegre
9 de Fevereiro de 2023

Agradecimentos

Agradeço ao Antônio, meu companheiro ao longo de toda a nossa jornada, sendo sempre muito especial em minha vida. Agredeço a toda família dele, em especial, Cíntia, José e Heloísa, por todo o apoio e carinho.

Aos meus pais por me apoiarem durante toda a minha vida e percurso acadêmico. Estendo meus agradecimentos aos meus irmãos, Augusto e Mariana.

À minha orientadora Márcia pelos ensinamentos durante a graduação e por todo o suporte oferecido, em especial neste trabalho.

Aos professores João e Anderson por aceitarem o convite para compor a banca.

Por fim, expresso meu agradecimento a todos os outros professores com os quais tive a oportunidade de interagir ao longo da graduação, especialmente o professor Álvaro.

Resumo

A previsão de dados espaciais tem ganhado crescente importância em diversas áreas do conhecimento. Nesse contexto, modelos baseados em redes neurais e aprendizado profundo emergem como opções viáveis para realizar tais previsões. Destaca-se o papel das redes neurais de grafos (GNN), as quais são exploradas neste estudo para capturar a dependência espacial. Isso ocorre por meio da agregação realizada entre regiões vizinhas, fundamentada na tendência de regiões próximas de apresentarem comportamentos semelhantes. O foco deste trabalho será na exploração de diferentes configurações de GNNs aplicadas a dados espaciais de área, com o objetivo de predição do número de alunos do projeto "Saúde com Agente" por município, considerando informações demográficas específicas.

Palavras-Chave: Estatística Espacial, Redes Neurais de Grafos (GNN), Aprendizado de Máquina.

Abstract

Spatial data forecasting has been gaining increasing importance in various fields of knowledge. In this context, models based on neural networks and deep learning emerge as viable options for making such predictions. The role of graph neural networks (GNN) stands out, and they are explored in this study to capture spatial dependencies. This is achieved through aggregation performed between neighboring regions, based on the tendency of nearby regions to exhibit similar behaviors. The focus of this work is on exploring different configurations of GNNs applied to spatial area data, with the aim of predicting the number of students in the "Saúde com Agente" project per municipality, considering specific demographic information.

Keywords: Spatial Statistics, Graph Neural Networks (GNN), Machine Learning.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

A predição de dados espaciais tem ganhado crescente relevância em diversos campos do conhecimento, abrangendo áreas como meteorologia Ma et al. (2022), análise de tráfego veicular Pan et al. (2019) e saúde Bailey (2001).

A estrutura de dependência é uma característica fundamental na análise de dados espaciais e pode se manifestar de diferentes maneiras Cressie (2015). Por exemplo, dados de área se referem a informações associadas à dados agregados que são coletados em unidades disjuntas, como por exemplo municípios ou bairros. Em geoestatística, os dados são coletados em pontos específicos distribuídos de forma contínua no espaço, como no monitoramento ambiental. No contexto de dados pontuais, os eventos ou ocorrências são aleatórios e a estrutura espacial envolve a distribuição dos pontos no espaço. Cada tipo de estrutura requer uma abordagem distinta para sua análise e interpretação.

Neste contexto, as Redes Neurais de Grafos (GNN) surgem como uma abordagem relevante, especialmente ao lidar com dados espaciais de área. As GNN realizam agrupamentos das informações das áreas vizinhas para realizar a predição de uma determinada área, capturando assim a dependência espacial. Isso se justifica pela tendência de áreas próximas apresentarem comportamentos semelhantes.

A adoção das GNN em contraposição a outros modelos estatísticos, como o método de campo aleatório markoviano gaussiano Blangiardo et al. (2013), é motivada pela habilidade das GNNs em enfrentar desafios preditivos associados a um grande número de covariáveis na análise. O conjunto de variáveis preditoras pode apresentar características, como multicolinearidade e confundimento espacial, que o modelo *Conditional Autoregressive* (CAR) não trata de maneira eficiente, devido à complexidade intrínseca desses dados Zuur et al. (2010). Dessa forma, torna-se crucial buscar metodologias que não apenas capturem a estrutura espacial, mas também sejam mais flexíveis em relação a essas questões.

O objetivo principal deste trabalho é realizar a predição do número de alunos no projeto "Saúde com Agente" em cada município, dado características demográficas, disponível em fontes como IBGE, DATASUS e RAIS. O projeto é uma colaboração entre a Universidade Federal do Rio Grande do Sul (UFRGS) e o Ministério da Saúde, juntamente com o Conselho Nacional de Secretarias municipais de Saúde (Conasems), destinado a oferecer os cursos de Agente Comunitário de Saúde (ACS) e de Vigilância em Saúde com Ênfase no Combate às Endemias (ACE). Como objetivos secundários, pretende-se avaliar a eficácia da abordagem das GNNs em comparação com modelos tabulares, que consideram observações independentes, para realizar essas previsões. Além disso, busca-se analisar essa eficácia para diferentes

inicializações de pesos, a fim de verificar se eventuais melhorias não são resultado de variações aleatórias. Adicionalmente, será conduzida uma comparação em termos de desempenho entre modelos que empregam diferentes camadas de GNN. A pesquisa abrangerá a avaliação do impacto da inclusão de camadas lineares no final do modelo, com o objetivo de determinar se essa alteração proporciona um aumento substancial no desempenho preditivo.

Para atingir os objetivos propostos, o trabalho adota uma abordagem metodológica que envolve a implementação de modelos GNN, incluindo arquiteturas como GCN, GraphConv, GAT, TransformerConv e GenConv. Essas estruturas são então comparadas com um método que considera as observações independentes, conhecido como MLP. Adicionalmente, realizou-se uma análise utilizando 30 diferentes sementes, com o intuito de avaliar a robustez e eficácia dos métodos GNN. Nesse processo, manteve-se a mesma configuração de modelo, variando apenas a inicialização dos pesos ao alternar as sementes. Esse delineamento proporciona não apenas uma comparação direta entre diversas abordagens, mas também uma análise da variabilidade dos resultados.

O segundo capítulo deste trabalho inicia-se com uma breve contextualização sobre dados espaciais, seguida pela introdução de conceitos essenciais em redes neurais. Na sequência, são detalhadas as GNNs, abordando sua configuração, funcionamento básico e apresentação das camadas clássicas utilizadas nesse contexto. Adicionalmente, proporciona-se uma breve contextualização das aplicações das GNNs no cenário espacial.

2 Dados Espaciais

A estrutura de dependência espacial é uma característica fundamental na análise de dados espaciais e pode se manifestar de diferentes maneiras (Cressie (2015)). Por exemplo, dados de região referem-se a informações vinculadas a conjuntos agregados que são coletados em unidades distintas, como municípios ou estados. Na geoestatística, os dados são coletados em locais específicos distribuídos de maneira contínua no espaço, como no monitoramento ambiental. No contexto de dados pontuais, os eventos ou ocorrências são aleatórios, e a estrutura espacial abrange a distribuição dos pontos no espaço.

Cada tipo de estrutura requer uma abordagem distinta para sua análise e interpretação. A fonte de dados deste trabalho expressa informações espaciais através da agregação de dados a nível municipal. Assim, o problema demanda métodos apropriados para dados de área.

2.1 Matriz de Adjacência para Dados Espaciais

A matriz de adjacência Waller and Gotway (2004) é uma representação tabular de relações entre elementos de um conjunto. Em dados espaciais de área, esses elementos são as área agregadas, podendo ser municípios ou estados, por exemplo. Essas áreas podem ser interpretadas como polígonos, em que o formato desse polígono é o contorno das fronteiras.

A construção da matriz de adjacência em dados espaciais envolve a identificação das relações espaciais entre os elementos. Essa medida de proximidade entre dois polígonos pode ser quantificada de forma binária, determinando se esses elementos compartilham ou não uma fronteira, isto é, se são adjacentes.

Seja $\mathbf{A}$ uma matriz de ordem $N \times N$ que representa a vizinhança entre as áreas, podemos representar essa estrutura através da adjacência entre as áreas:

$$A_{ij} = \begin{cases} 1, & \text{se os polígonos } i \text{ e } j \text{ compartilham fronteiras,} \\ 0, & \text{caso contrário} \end{cases}. \tag{2.1}$$

Existem muitas medidas de proximidade diferentes que podem ser usadas para definir a proximidade entre duas áreas, não apenas a definida na Equação (2.1). Essa representação de proximidade pode ser dada por meio de distâncias entre os centroides das áreas:

$$A_{ij} = \begin{cases} 1, & \text{se } d_{ij} < \delta, \\ 0, & \text{caso contrário.} \end{cases}, \tag{2.2}$$

em que d_{ij} é a distância entre os centróides dos polígonos i e j (podendo ser, por exemplo, a distância euclidiana) e δ é o raio pelo qual duas áreas serão considerados vizinhas se a distância entre os seus centroides for menor que esse valor fixado.

Além disso, a definição de adjacência estabelecida nas Equações (2.1) e (2.2), pode ser ponderada de acordo com a população. Isso é particularmente útil em contextos como o da saúde, em que populações mais numerosas em uma determinada região tendem a produzir taxas mais consistentes e confiáveis Waller and Gotway (2004). As taxas, nesse contexto, referem-se à incidência ou a frequência de um determinado fenômeno em relação à população de uma área específica. Essa ponderação assegura que regiões com população mais numerosas recebam uma contribuição proporcionalmente maior:

$$A_{ij} = \begin{cases} n_j, & \text{se } d_{ij} < \delta, \\ 0, & \text{caso contrário} \end{cases},$$

em que n_j é a população total da área j.

2.2 Estruturas de Grafos

Um grafo é uma forma de representar relações entre entidades Wilson (1979), composto por dois conjuntos principais: o conjunto de vértices (também chamados de nós ou nodos) e o conjunto de arestas. As arestas conectam pares de vértices e expressam a relação entre eles.

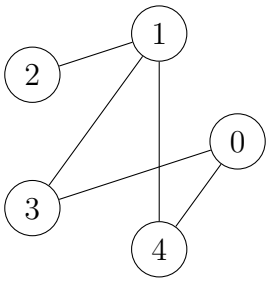


Figura 2.1: Estrutura de um Grafo

Na Figura (2.1), temos a representação da estrutura de um grafo composto por 5 nós numerados de 0 a 4. Os 5 vértices presentes na imagem denotam as conexões entre esses nós, evidenciando as relações entre os nodos 0 e 4, 0 e 3, 1 e 2, 1 e 3, e 1 e 4.

Um grafo também pode ser definido como direcionado ou não direcionado Waikhom and Patgiri (2023). Em um grafo não direcionado, as arestas não possuem uma orientação específica, o que resulta em uma relação bidirecional. Por outro lado, em um grafo direcionado, cada aresta representa uma conexão unidirecional, apontando do vértice de origem para o vértice de destino. A característica de direcionamento das arestas em um grafo, seja ele direcionado ou não direcionado, desempenha um papel crucial na natureza das relações entre seus vértices.

A representação visual da diferença de um grafo direcionado em relação a um grafo não direcionado está ilustrado na figura 2.2.

(a) Grafo Não Direcionado (b) Grafo Direcionado

Figura 2.2: Grafo Não Direcionado e Direcionado

2.2.1 Matriz de adjacência Associada a um Grafo

A matriz de adjacência pode ser considerada uma representação tabular de um grafo, fornecendo uma visão das relações entre os vértices. Seja um grafo $G(V, E)$ com N vértices, a matriz de adjacência **A** será uma matriz quadrada de ordem $N \times N$, em que cada elemento A_{vu} representa a presença ou ausência de uma aresta entre os vértices v e u. Matematicamente, a matriz de adjacência é representada por:

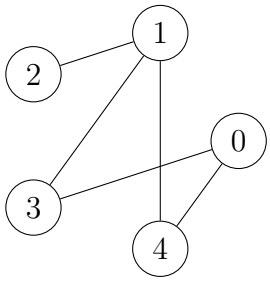
$$A_{vu} = \begin{cases} 1, & \text{se } u \in N(v), \\ 0, & \text{caso contrário} \end{cases},$$

em que $N(v)$ representa a vizinhança do nó v.

A_{vu} será simétrica se o grafo for não-direcionado. Existem outras formas de construir a matriz de adjacência A_{vu}, que podem ser consultadas em Donnat and Holmes (2018).

Uma representação menos cara computacionalmente de uma matriz de adjacência é a lista de adjacência. Enquanto a matriz de adjacência consome espaço proporcional ao quadrado do número de vértices em um grafo, a lista de adjacência usa uma quantidade de espaço proporcional ao número total de arestas.

A ilustração das representações do grafo através da matriz de adjacência e da lista de adjacência pode ser observada na Figura (2.3), abrangendo tanto o grafo direcionado quanto o grafo não direcionado. Essas representações tabulares fornecem uma visão clara das conexões entre os vértices, permitindo uma análise mais detalhada da estrutura do grafo.



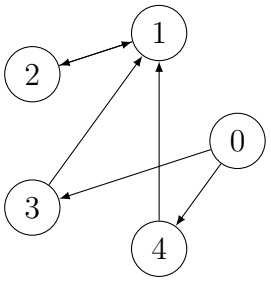
Matriz:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Lista:

$0 : [3, 4]$
$1 : [2, 3, 4]$
$2 : [1]$
$3 : [0, 1]$
$4 : [0, 1]$

(a) Grafo não direcionado



Matriz:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Lista:

$0 : [3, 4]$
$1 : [2]$
$2 : [1]$
$3 : [1]$
$4 : [1]$

(b) Grafo direcionado

Figura 2.3: Comparação entre adjacência de grafo não direcionado e direcionado

2.2.2 Matriz de Grau (*Degree Matrix*)

A matriz de grau é uma representação tabular que captura informações sobre os graus dos nós em um grafo West et al. (2001). O grau de um nó é o número de arestas incidentes a ele. Seja um grafo $G(V, E)$ com N vértices, a matriz de grau **D** é uma matriz diagonal de ordem $N \times N$, em que cada elemento D_{vv} representa o grau do nó v.

Para um grafo não direcionado, a matriz **D** é definida por:

$$D_{vv} = \sum_{u=1}^{N} A_{vu},$$

em que A_{vu} é o elemento correspondente na matriz de adjacência, indicando se existe uma conexão entre os nós v e u. Em resumo, a matriz de grau reflete a soma total das arestas conectadas a cada vértice.

No caso de um grafo direcionado, a matriz de grau é dividida em grau de entrada D_{in} e grau de saída D_{out}. O grau de entrada D_{in} é calculado a partir da soma das arestas que entram a cada vértice:

$$D_{\text{in},vv} = \sum_{u=1}^{N} A_{uv},$$

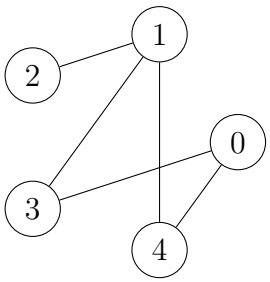
enquanto que o grau de saída D_{out} indica a soma das arestas que saem de cada nó:

$$D_{\text{out},vv} = \sum_{u=1}^{N} A_{vu}.$$

A matriz de grau total **D** em um grafo direcionado é obtida através da soma das matrizes de grau de entrada e saída:

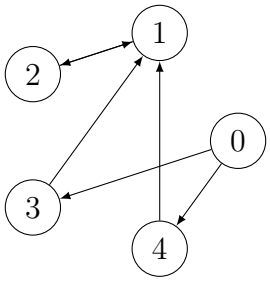
$$D_{vv} = D_{\text{in},vv} + D_{\text{out},vv}.$$

Uma representação de matriz de grau para grafo não direcionado e direcionado está ilustrada na Figura (2.4). Essa abordagem oferece uma visão detalhada sobre a distribuição dos graus nos nós, contribuindo significativamente para a análise e compreensão das características estruturais do grafo.



$$D : \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

(a) Grafo Não Direcionado



$$D : \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{in} + \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{out}$$

(b) Grafo Direcionado

Figura 2.4: Comparação entre Grau de Grafo Não Direcionado e Grafo Direcionado

2.2.3 Representação de Dados Espacias via Grafos

A representação de dados espaciais por meio de grafos segundo Qian et al. (2014) e Aydin et al. (2018) é uma abordagem para capturar e explorar efetivamente a estrutura e as relações espaciais inerentes aos dados.

Nesse contexto, as unidades espaciais são modeladas como nós, enquanto as relações espaciais entre elas são representadas por arestas no grafo. A presença de uma aresta entre dois nós indica que as unidades espaciais correspondentes são vizinhas, estabelecendo uma conexão. Vale destacar que o grafo é necessariamente não direcionado. Isso significa que, se uma área é considerada vizinha de outra, a recíproca também é verdadeira.

Por exemplo, ao transformar os municípios pertencentes aos arredores de Porto Alegre (RS) em uma representação de grafo, poderíamos gerar uma visualização semelhante à imagem apresentada na Figura (2.5).

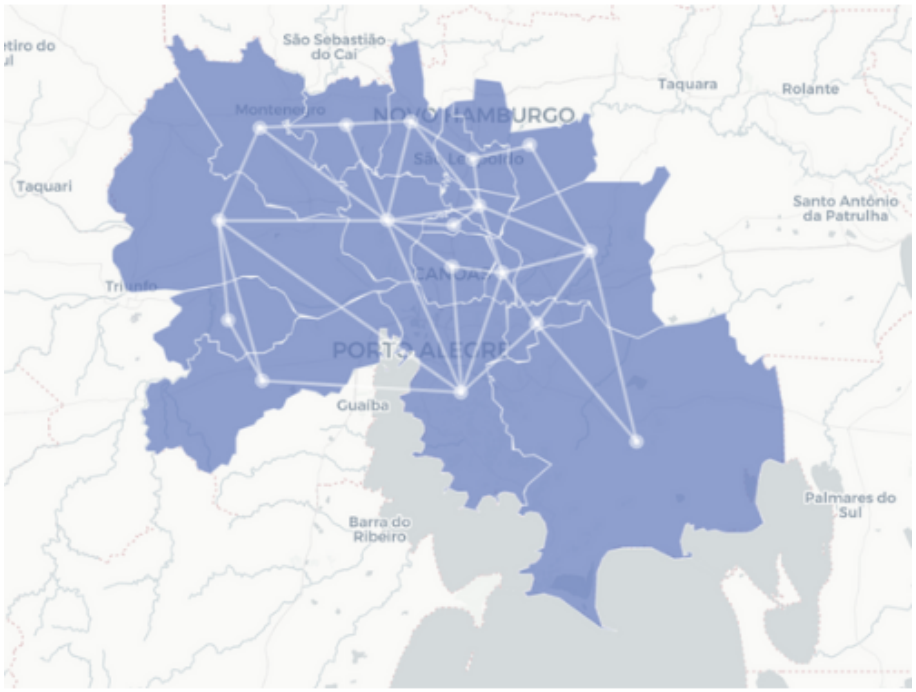


Figura 2.5: Exemplo de Mapa da região metropolitana de Porto Alegre como um grafo, em que cada círculo no centroide da cidade representa um nó e as linhas brancas representam as arestas.

3 Redes Neurais

O conceito inicial das redes neurais artificiais surgiu na tentativa de replicar o funcionamento dos neurônios cerebrais. McCulloch and Pitts (1943) simplificaram essa ideia por meio de um circuito elétrico, lançando as bases para o desenvolvimento de modelos formais de neurônios artificiais. Logo depois, Rosenblatt (1958) introduziu o Perceptron, um sistema com uma relação simples de entrada e saída, modelado a partir do funcionamento de um neurônio.

O cérebro humano é composto por bilhões de células nervosas chamadas neurônios. Esses neurônios estão interconectados formando uma rede complexa. Quando uma informação é processada, os neurônios se comunicam entre si por meio de conexões chamadas sinapses. Cada sinapse possui um peso que modula a força da comunicação entre os neurônios. Assim como os neurônios humanos processam informações através de sinapses, os neurônios artificiais em uma rede neural processam informações através de conexões ponderadas.

Durante o aprendizado humano, as sinapses se ajustam com base na experiência. Se uma sinapse é frequentemente ativada em resposta a um estímulo específico, sua força aumenta, permitindo uma resposta mais eficiente no futuro. Da mesma forma, em redes neurais artificiais, os pesos das conexões são ajustados durante o treinamento para otimizar o desempenho do modelo.

Assim como diferentes regiões do cérebro humano são especializadas em tarefas específicas, como visão, audição e movimento, as camadas em uma rede neural podem ser projetadas para tarefas específicas, formando uma hierarquia de representações.

A estrutura fundamental de uma rede neural é composta por neurônios, unidades de processamento básicas, organizadas em camadas. Normalmente, há uma camada de entrada $\mathbf{X}$, uma ou mais camadas ocultas $\mathbf{h}$ e uma de saída $\mathbf{y}$. A conexão entre esses neurônios é determinada por pesos, os quais são ajustados durante o processo de treinamento, juntamente com a influência de funções de ativação.

3.1 Estrutura Fundamental de uma Rede Neural

Os neurônios são as unidades fundamentais de processamento em uma rede neural. Estes neurônios desempenham o papel de unidades de processamento, realizando operações sobre os dados de entrada. Inspirados pela arquitetura do sistema nervoso biológico, os neurônios artificiais nas redes neurais buscam imitar a natureza em seu funcionamento.

Na Figura (3.1), é ilustrado o papel desempenhado pelo j-ésimo neurônio da

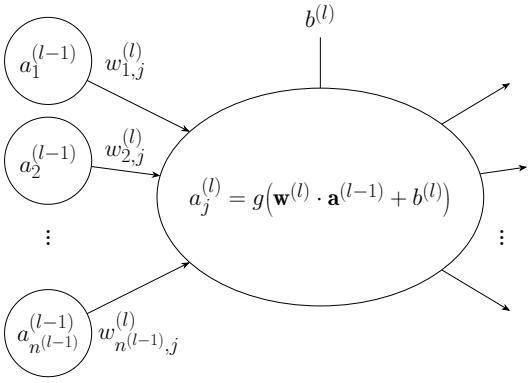


Figura 3.1: Representação gráfica da operação de um único neurônio $a_j^{(l)}$ em uma rede neural.

camada l, denotado como $a_j^{(l)}$. Esse neurônio recebe como entrada os $n^{(l-1)}$ valores $a_i^{(l-1)}$ provenientes da camada anterior $l-1$. Ele executa um somatório dessas entradas, cada uma multiplicada pelos pesos correspondentes $w_{ij}^{(l)}$ que são ajustados durante o treinamento, isto é,

$$\mathbf{w}^{(l)} \cdot \mathbf{a}^{(l-1)} = \sum_{i=1}^{n^{(l-1)}} w_{i,j}^{(l)} \cdot a_i^{(l-1)}.$$

Esse somatório é então adicionado ao $b^{(l)}$, um escalar, conhecido como *bias*, que também é ajustado durante o treinamento, assim como os pesos:

$$\mathbf{w}^{(l)} \cdot \mathbf{a}^{(l-1)} + b^{(l)}.$$

O resultado desse processo é submetido à função de ativação $g(.)$ (consulte Capitulo (3.3)), resultando no valor final $a_j^{(l)}$:

$$a_j^{(l)} = g\left(\mathbf{w}^{(l)} \cdot \mathbf{a}^{(l-1)} + b^{(l)}\right). \tag{3.1}$$

O valor resultante da Equação (3.1) é propagado para as camadas subsequentes da rede neural.

Em resumo, cada neurônio $a_j^{(l)}$ recebe um conjunto de entradas $\mathbf{a}^{(l-1)}$, realiza uma transformação ponderada dessas entradas utilizando os pesos $\mathbf{w}^{(l)}$, adiciona o $b^{(l)}$ e aplica a função de ativação $g(.)$. Essa saída é, então, encaminhada para as camadas subsequentes da rede.

A partir da arquitetura de um neurônio, temos a capacidade de construir redes neurais completas Rumelhart et al. (1986). Na ilustração da Figura (3.2), os neurônios da camada de entrada são representados em formato de retângulo, enquanto que os demais neurônios são representados por círculos distribuídos ao longo das quatro camadas. A camada de entrada $l = 0$, composta pelos *inputs*, serve como ponto de entrada, estabelecendo conexões diretas e ponderadas por pesos $\mathbf{w}^{(1)}$ com os neurônios da primeira camada oculta $l = 1$. Essa camada, por sua vez, está

interligada com a segunda camada oculta $l = 2$, que, por fim, se conecta à camada de saída $l = 3$. Na camada de saída, está representada a variável a ser predita pelo modelo. Essa estrutura de interconexões, governada por pesos ajustáveis, possibilita que a rede neural aprenda e modele padrões nos dados.

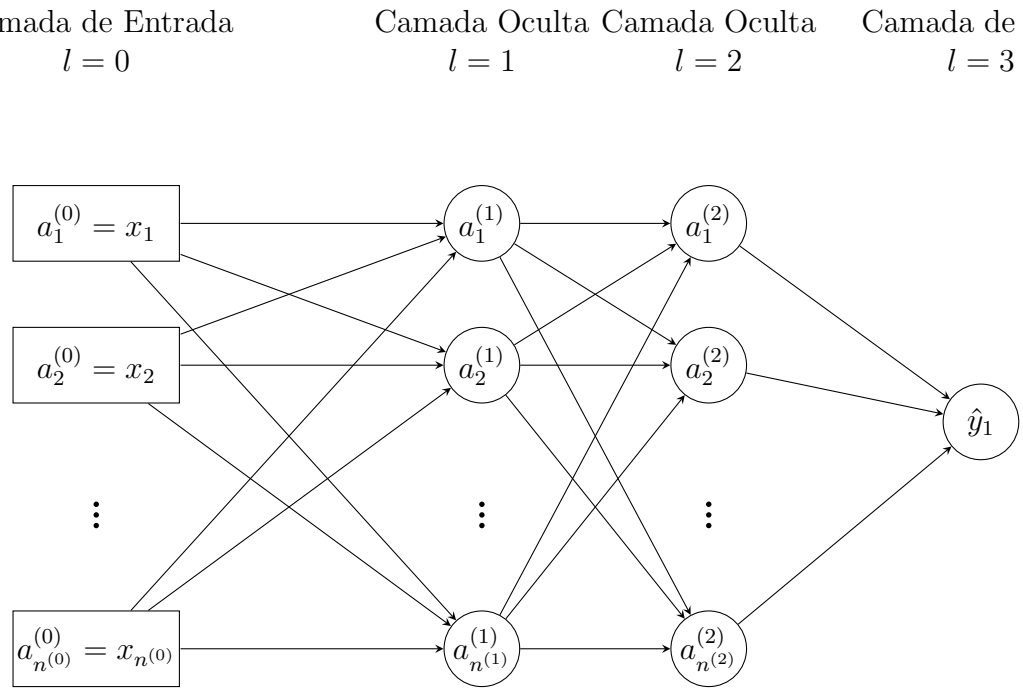


Figura 3.2: Estrutura de uma rede neural preditiva com $n^{(0)}$ neurônios de entradas e um neurônio de saída, duas camadas ocultas com $n^{(1)}$ e $n^{(2)}$ neurônios cada, respectivamente.

A camada de entrada deve possuir um número total de neurônios equivalente à quantidade de covariáveis utilizadas para a predição. Em contrapartida, tanto a determinação do número de neurônios nas camadas ocultas quanto a escolha da quantidade de camadas ocultas são processos arbitrários, geralmente guiados pelo desempenho do modelo. A busca pela configuração ideal visa encontrar a combinação que proporciona os melhores resultados.

A decisão sobre o número de neurônios a serem utilizados na camada de saída é influenciada diretamente pelo objetivo do modelo. Em cenários de modelagem voltados para a regressão, em que a previsão envolve uma variável quantitativa como resposta, é empregado um neurônio na camada de saída.

No entanto, quando o foco está na classificação, a escolha do número de neurônios na camada de saída é guiada pelo tipo de classificação desejada. Em situações binárias, em que há apenas duas classes possíveis, utiliza-se um único neurônio de saída. No caso de classificação multiclasse, em que existem mais de duas classes, o número de neurônios na camada de saída é igual ao total de classes. Essa regra é válida apenas quando as classes são mutuamente exclusivas.

3.2 *Feedforward*

O *feedforward* é o processo pelo qual a rede neural realiza a propagação de informações da camada de entrada até a camada de saída. Nesse processo, cada

camada realiza uma transformação linear seguida por uma função de ativação.

Os neurônios da camada de entrada $\mathbf{a}^{(0)}$ não passam por um cálculo explícito, pois eles representam diretamente as entradas do modelo. No entanto, os neurônios das camadas subsequentes $\mathbf{a}^{(l)}$ são calculados por meio da multiplicação dos pesos pelas ativações dos neurônios da camada anterior, somada a um vetor de *bias* (viés) $\mathbf{b}^{(l)}$. Em termos de fórmula para o cálculo de $a_i^{(l)}$, isto é, a i-ésimo ativação a da camada l, teríamos:

$$a_i^{(l)} = g\left(\sum_{j=1}^{n^{(l-1)}} w_{i,j}^{(l)} \cdot a_j^{(l-1)} + b_i^{(l)}\right),$$

tal que $g()$ representa a função de ativação, $a_j^{(l-1)}$ simboliza o neurônio j da camada anterior $l-1$, $w_{i,j}^{(l)}$ representa os pesos das conexões entre os neurônios da camada $l-1$ e o neurônio atual, e o $b_i^{(l)}$ é o *bias* associado ao neurônio i na camada l.

Para generalizar o cálculo para todas as ativações na camada l, temos:

$$\mathbf{a}^{(l)} = g\Big(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\Big),$$

$$\begin{pmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{n^{(l)}}^{(l)} \end{pmatrix} = g\left(\begin{pmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \dots & w_{1,n^{(l)}}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \dots & w_{2,n^{(l)}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n^{(l-1)},1}^{(l)} & w_{n^{(l-1)},2}^{(l)} & \dots & w_{n^{(l-1)}n^{(l)}}^{(l)} \end{pmatrix} \begin{pmatrix} a_1^{(l-1)} \\ a_2^{(l-1)} \\ \vdots \\ a_{n^{(l-1)}}^{(l-1)} \end{pmatrix} + \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{n^{(l)}}^{(l)} \end{pmatrix}\right),$$

em que $g()$ é uma função de ativação, $\mathbf{W}^{(l)}$ é uma matriz de pesos associada à camada l, $\mathbf{a}^{(l-1)}$ é o vetor de ativações proveniente da camada anterior $l-1$, $\mathbf{b}^{(l)}$ é o vetor de *bias* (viés) para a camada l.

Este processo é ilustrado na Figura (3.3), que destaca os elementos associados para o cálculo da ativação do neurônio $a_1^{(l)}$, por exemplo.

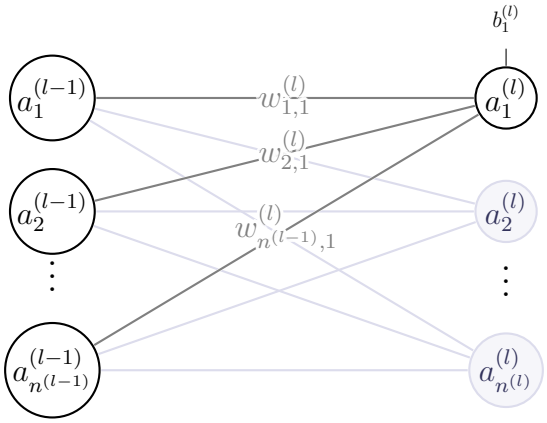


Figura 3.3: Representação do fluxo de informação durante uma operação *feedforward* com foco no neurônio $a_1^{(l)}$, destacando os componentes necessários para o seu cálculo. A estrutura compreende $n^{(l-1)}$ neurônios na camada $l-1$ e $n^{(l)}$ neurônios na camada l, em que as conexões ponderadas são indicadas por $w_{i,j}^{(l)}$. Os neurônios da camada $l-1$ são destacados como $a_i^{(l-1)}$, com i variando de 1 a $n^{(l-1)}$, enquanto que os neurônios da camada l são denotados como $a_j^{(l)}$, com j variando de 1 a $n^{(l)}$. Além disso, o *bias* associado ao neurônio focal $a_1^{(l)}$ é representado por $b_1^{(l)}$. Embora não estejam representados nesta imagem, os outros neurônios também possuem valores de *bias*, exceto na camada de entrada $l = 0$.

3.3 Funções de ativação

A função de ativação é um componente crucial em redes neurais, sendo responsável por introduzir não linearidades nas operações realizadas pelos neurônios. A seleção da função de ativação desempenha um papel crucial na arquitetura de uma rede neural, e cada função de ativação possui características únicas. Para uma análise mais aprofundada sobre funções de ativação, consulte Apicella et al. (2021).

3.3.1 Função ReLU (Rectified Linear Unit)

A função ReLU Glorot et al. (2011) é uma função de ativação frequentemente usada em redes neurais. Ela é definida como a função máxima entre zero e o próprio argumento. A expressão matemática da função ReLU é dada por:

$$g(z) = max(0, z), \tag{3.2}$$

em que z é a entrada da função, geralmente sendo a representação de $z_j^{(l)} = \mathbf{w}^{(l)} \cdot \mathbf{a}^{(l-1)} + b^{(l)}$, isto é, o neurônio antes da aplicação da função de ativação.

Como evidenciado na Figura (3.4), a função ReLU exibe seu comportamento característico de ativar de forma linear para valores de entrada positivos, devolvendo diretamente o valor de entrada, enquanto mantém a ativação nula para valores de entrada negativos.

Contudo, é importante notar que a ReLU também pode apresentar o problema conhecido como *dying ReLU* Lu et al. (2019). Neste cenário, alguns neurônios podem tornar-se inativos durante o treinamento, resultando na perda de capacidade de aprendizado. Para mitigar este problema, foram propostas variações da ReLU,

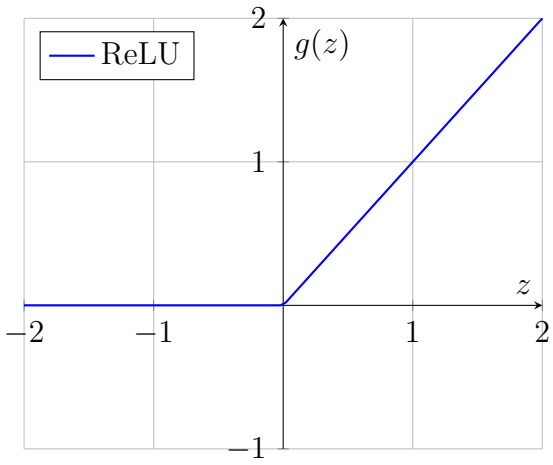


Figura 3.4: Representação gráfica da função ReLU. Ela retorna $g(z) = 0$ para valores de $z < 0$, enquanto que para valores de $z \geq 0$, ela retorna $g(z) = z$.

como Leaky ReLU Maas et al. (2013) e Parametric ReLU He et al. (2015). Essas variantes introduzem pequenos gradientes para valores negativos, assegurando uma ativação mínima mesmo para entradas negativas.

3.3.2 Função Sigmoide

A função sigmoide transforma a entrada z para um intervalo entre 0 e 1. A expressão matemática da função Sigmoide é definida como:

$$g(z) = \frac{1}{1 + e^{-z}},$$

em que $z \in (-\infty, +\infty)$ é a entrada da função e $g \in (0, 1)$.

Como ilustrado na Figura (3.5), a função sigmoide $g(z)$ transforma a entrada z de maneira suave, resultando em uma saída que varia continuamente entre 0 e 1. Essa transição gradual é uma característica distintiva da função sigmoide, a qual a torna particularmente adequada para camadas de saída em redes neurais binárias. Ao mapear os valores de z para o intervalo $(0, 1)$, a função sigmoide é comumente empregada em problemas de classificação binária, onde a saída representa probabilidades associadas às diferentes classes.

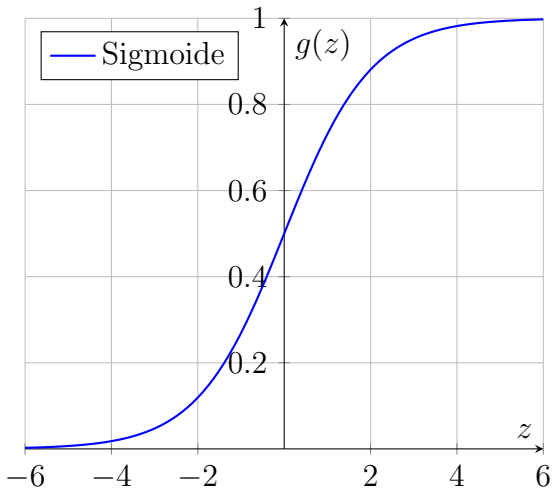


Figura 3.5: Representação gráfica da função sigmoide. A função sigmoide, $g(z)$, é definida como $g(z) = \frac{1}{1+e^{-z}}$. Ela mapeia valores para o intervalo $(0, 1)$, sendo frequentemente utilizada como função de ativação em problemas de classificação binária em redes neurais.

3.3.3 Função Softmax

Usada na camada de saída para problemas de classificação multiclasse, em que converte um vetor de entradas em probabilidades que somam 1. Sua formulação é expressa por:

$$g(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}},$$

tal que $\mathbf{z}$ é o vetor de entradas, z_i é o elemento i do vetor, e C é o número total de classes. Essa função proporciona uma representação de probabilidade para cada classe, facilitando a interpretação dos resultados da rede neural em contextos de classificação multiclasse.

3.4 Função de Perda (*loss*)

A função de perda, também conhecida como função de custo, é uma medida que quantifica o desempenho de uma rede neural. A ideia é ter uma função que represente a diferença entre as previsões do modelo e os valores reais. O objetivo é minimizar essa função de perda durante o treinamento, ajustando os parâmetros do modelo. A escolha da função de perda ideal depende do problema que está sendo abordado. Para uma referência mais detalhada sobre funções de perda, consulte Wang et al. (2020).

3.4.1 Erro Quadrático Médio (MSE)

A métrica do Erro Quadrático Médio é comum para problemas de regressão, em que a variável resposta é quantitativa. Sua expressão matemática é dada por:

$$J(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{3.3}$$

tal que y_i são os valores reais, $\hat{y}_i$ são as previsões do modelo e n é o número de observações em y_i.

3.5 *Backpropagation*

O *backpropagation* Rumelhart et al. (1986), ou retropropagação, é um algoritmo fundamental no treinamento de redes neurais. Ele é utilizado para ajustar os pesos da rede com o objetivo de minimizar a função de perda.

3.5.1 Cálculo dos Erros

No processo de cálculo dos deltas de erro durante a retropropagação, a trajetória segue o caminho inverso do *feedforward*. Inicialmente, o primeiro erro é calculado na camada de saída L. Esse erro é, então, propagado para as camadas anteriores, seguindo a ordem $L-1, L-2, ..., 1$, até alcançar a primeira camada oculta.

Para o cálculo do erro de saída, o delta do erro δ_L é obtido através da diferença entre a previsão do modelo e o valor verdadeiro:

$$\delta_L = \hat{y}_i - y_i,$$

em que $\hat{y}_i$ representa a previsão ou estimativa do modelo para a i-ésima observação, enquanto y_i denota o valor verdadeiro ou rótulo correspondente.

Ao calcular os deltas do erro nas camadas ocultas, isto é, nas camadas $l \neq L$ e $l > 0$, é crucial levar em conta a propagação retroativa do erro na rede neural. A ideia central é compreender como o erro em uma camada específica é influenciado pelos pesos associados a ela e pelo erro nas camadas subsequentes. A fórmula que descreve essa propagação de erro é dada por:

$$\delta_i^{(l)} = \left(\sum_{j=1}^{n^{(l+1)}} w_{ij}^{(l+1)} \cdot \delta_j^{(l+1)} \right) \cdot g'(z_i^{(l)}),$$

tal que $n^{(l+1)}$ é o número de neurônios na camada $l+1$, $w_{ij}^{(l+1)}$ é peso da conexão entre a unidade i na camada l e a unidade j na camada $l+1$, $z_i^{(l)}$ é a entrada para o i-ésimo neurônio na camada l antes da aplicação da função de ativação ($a_i^{(l)} = g(z_i^{(l)})$), $g'()$ é a derivada da função de ativação em relação à sua entrada e $\delta_j^{(l+1)}$ é o erro associado ao neurônio j na camada $l+1$. A representação desse processo é ilustrada na Figura (3.6).

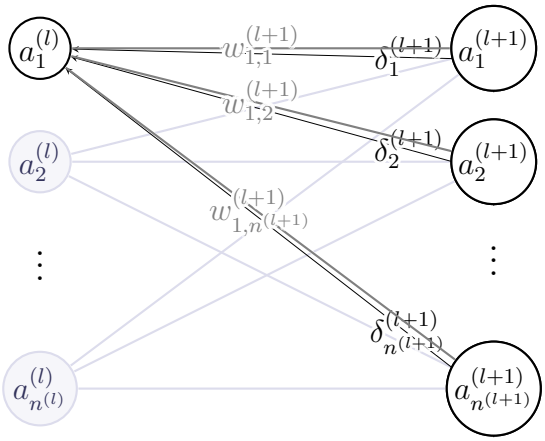


Figura 3.6: Visualização do processo de *backpropagation* na rede neural, focalizando o neurônio $a_1^{(l)}$ e destacando os elementos cruciais para o cálculo do erro causado por esse neurônio. Os $n^{(l)}$ neurônios na camada l são representados por $a_i^{(l)}$, onde i varia de 1 a $n^{(l)}$, correspondendo à quantidade de neurônios na camada l. Os $n^{(l+1)}$ neurônios na camada $l+1$ são representados por $a_i^{(l+1)}$, com i variando de 1 a $n^{(l+1)}$. As setas delineiam o percurso do erro durante a *backpropagation*, ajustado pelos pesos ($w_{i,j}^{(l+1)}$).

3.5.2 Atualização dos Pesos e *Biases*

A atualização dos pesos $w_{i,j}^{(l)}$ e dos *biases* $b_i^{(l)}$ segue o princípio do gradiente descendente (consulte Seção (3.6)). A regra geral para atualizar os pesos é dada por:

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \alpha \cdot \frac{\partial J}{\partial w_{ij}^{(l)}},$$

em que α é a taxa de aprendizado, J é a função de perda (Seção (3.4)) e $\frac{\partial J}{\partial w_{ij}^{(l)}}$ é derivada parcial da função de perda em relação ao peso $w_{i,j}^{(l)}$.

A atualização do *bias* $b_i^{(l)}$ é calculada semelhantemente:

$$b_i^{(l)} \leftarrow b_i^{(l)} - \alpha \frac{\partial J}{\partial b_i^{(l)}}.$$

3.6 Otimizador baseado em Descida do Gradiente

O conceito de gradiente e a técnica associada à descida do gradiente, segundo Robbins and Monro (1951) são fundamentais no treinamento de uma rede neural.

O gradiente $\nabla J(\theta)$ representa a direção de maior crescimento de uma função em um ponto específico. O cálculo do gradiente acontece por meio de derivadas parciais da função de perda J, a qual buscamos minimizar, em relação aos componentes θ que queremos otimizar (pesos e *biases*). Dado que nosso objetivo é reduzir a função de perda, o foco recai sobre a técnica conhecida como descida do gradiente.

A descida do gradiente é um algoritmo usado para minimizar uma função iterativamente. O objetivo é ajustar os parâmetros θ de forma a reduzir a função de custo J associada ao modelo.

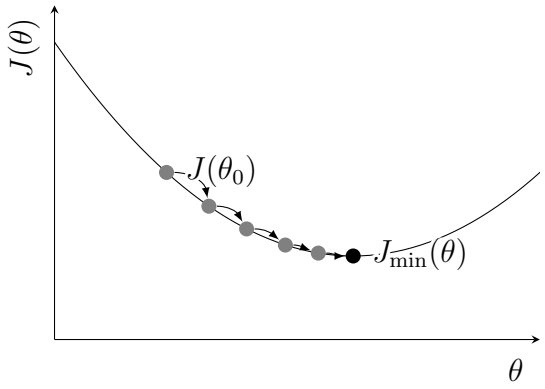


Figura 3.7: Descida do Gradiente

Toda rede neural requer a aplicação de um otimizador, que consiste em algoritmos específicos que empregam a descida do gradiente, como o Gradiente Descendente Estocástico (SGD), o Adam, o RMSprop, entre outros Gupta (2021). Esses otimizadores incorporam modificações e refinamentos à descida do gradiente fundamental, visando aprimorar a eficiência do treinamento.

3.7 Treinamento da Rede Neural

O processo de treinamento de uma rede neural é fundamental para capacitar o modelo a aprender padrões inerentes nos dados. Durante o treinamento, a rede neural ajusta seus pesos e *biases* com base nos dados de entrada e saída conhecidos. O objetivo é minimizar uma função de custo que quantifica a diferença entre as saídas previstas pela rede e os valores reais desejados.

Dividir o conjunto de dados em conjuntos de treino e teste é uma prática crucial no treinamento de modelos de rede neural. A divisão padrão, muitas vezes de 70% para treino e 30% para teste, visa avaliar o desempenho do modelo Brownlee (2020). Ao separar os conjuntos de treino e teste, garantimos que o modelo seja avaliado em dados não vistos durante o treinamento. O conjunto de teste age como um indicador da capacidade da rede neural de generalizar para novos dados Bishop (2006).

Algorithm 1 Algoritmo de Treinamento

Inicialização dos pesos W e dos *biases* b da rede neural
Definição da taxa de aprendizado α
Definição do número de épocas de treinamento $N_{\text{épocas}}$
Divisão dos dados em conjuntos de treino e teste
for época $= 1$ até $N_{\text{épocas}}$ **do**
    for cada exemplo de treino $(\mathbf{x}, y)$ **do**
        Faça a propagação para a frente (*feedforward*):
            Calcule a saída predita $\hat{y}$
        Calcule a função de perda $J = \text{calcular\_perda}(y, \hat{y})$
        Faça a retropropagação (*backpropagation*):
            Calcule os gradientes $\frac{\partial J}{\partial W}$ e $\frac{\partial J}{\partial b}$
        Atualize os pesos e *biases* usando a descida do gradiente:
            $W \leftarrow W - \alpha \frac{\partial J}{\partial W}$
            $b \leftarrow b - \alpha \frac{\partial J}{\partial b}$
    end for
end for

No processo de treinamento de uma rede neural, a aplicação de técnicas como a descida do gradiente pode ser realizada de duas maneiras distintas: em lote ou em mini-lote Patrikar (2019). No contexto do treinamento em lote, a expressão "lote" refere-se ao conjunto completo de exemplos de treino utilizado em uma única iteração do algoritmo de treinamento. Isso implica que todos os dados de treino são processados simultaneamente para calcular as atualizações nos pesos e biases da rede neural. Por outro lado, ao optar pelo treinamento em mini-lote, a referência é feita a uma pequena amostra aleatória extraída do conjunto total de exemplos de treino. Dessa forma, em cada iteração do algoritmo, apenas uma fração dos dados de treino é utilizada para ajustar os parâmetros da rede.

Além disso, técnicas como regularização, normalização e escolha cuidadosa de funções de ativação, são frequentemente empregadas para melhorar o desempenho e a generalização da rede. O treinamento é considerado bem-sucedido quando a rede é capaz de produzir saídas precisas para dados não vistos durante o treinamento, demonstrando sua capacidade de generalização.

3.8 *Overfitting*, *Underfitting*, Normalização e Regularização

3.8.1 *Overfitting*

Overfitting Ying (2019) é observado quando um modelo se ajusta de maneira excessiva aos dados de treinamento, focando na memorização de padrões específicos em detrimento da compreensão de conceitos mais amplos. Esse fenômeno prejudica a capacidade do modelo de generalizar para dados não vistos, comprometendo sua eficácia em situações além do conjunto de treinamento.

3.8.2 *Underfitting*

Underfitting Jabbar and Khan (2015) é o oposto de *overfitting* e ocorre quando um modelo é muito simples para capturar as complexidades nos dados. Um modelo subajustado não consegue aprender as relações fundamentais nos dados de treino e, portanto, também apresenta desempenho fraco nos dados de teste.

3.8.3 Normalização

A normalização refere-se a técnicas que alteram a escala das variáveis de um conjunto de dados para que possam ser comparáveis ou para facilitar o treinamento de modelos.

- ***Min-Max Scaling*:** transforma os valores dos atributos para um intervalo específico, geralmente entre 0 e 1, utilizando a seguinte expressão:

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}},$$

  onde X_{min} representa o valor mínimo de X, e analogamente, X_{max} denota o valor máximo de X.

- ***Z-Score Normalization*:** transforma os valores dos atributos para ter uma média de 0 e um desvio padrão de 1, através da formulação matemática dada por:

$$X_{\text{norm}} = \frac{X - \mu}{\sigma},$$

  em que μ refere-se à média de X e σ representa o desvio padrão.

Para obter uma compreensão mais aprofundada e explorar outras técnicas de normalização, confira Gökhan et al. (2019).

3.8.4 Regularização

A regularização é uma técnica utilizada em aprendizado de máquina para evitar *overfitting*, introduzindo penalidades nos parâmetros do modelo, desencorajando valores extremos Jain (2018).

- **Regularização L1:** adiciona a soma dos valores absolutos dos coeficientes como uma penalidade à função de custo. Essa penalidade é aplicada da seguinte forma:

$$J(\theta) = J(y, \hat{y}) + \lambda \sum_{i=1}^{n} |\theta_i|,$$

  em que λ é o parâmetro de regularização, um escalar geralmente pequeno. A função de perda é denotada por $J(y, \hat{y})$ e θ_i refere-se aos pesos treinados pela rede neural.

- **Regularização L2:** incorpora uma penalidade à função de custo por meio da inclusão da soma dos quadrados dos coeficientes, expressa da seguinte forma:

$$J(\theta) = J(y, \hat{y}) + \lambda \sum_{i=1}^{n} \theta_i^2,$$

  λ desempenha o papel de parâmetro de regularização, $J(y, \hat{y})$ representa a função de perda, e θ_i é uma referência aos pesos.

- ***Dropout*:** desliga aleatoriamente um conjunto de neurônios durante o treinamento da rede. Isso significa que, em cada passo do treinamento, um subconjunto aleatório de unidades é excluído temporariamente Krizhevsky et al. (2012).

3.9 Rede Neural de Multicamadas (MLP)

O modelo de Rede Neural de Multicamadas (MLP) Gardner and Dorling (1998) é uma arquitetura de aprendizado profundo que se destaca por sua capacidade de aprender representações a partir de dados. Ele consiste em várias camadas de neurônios, organizadas em uma estrutura hierárquica, onde cada camada está conectada à seguinte. Cada neurônio em uma camada processa informações e transmite os resultados para a camada seguinte.

A arquitetura MLP é composta por três tipos principais de camadas: a camada de entrada, as camadas ocultas e, por fim, as camadas de saída. A camada de entrada é responsável por receber as características ou covariáveis do conjunto de dados, em que cada neurônio representa uma característica específica. Entre a camada de entrada e a camada de saída, existem uma ou mais camadas ocultas. Cada neurônio nessas camadas processa informações ponderadas das camadas anteriores. A camada de saída gera a predição final do modelo. O número de neurônios nesta camada depende do tipo de tarefa, como classificação ou regressão.

4 Redes Neurais para Grafos (GNNs)

Redes neurais convencionais são eficazes na modelagem de dados tabulares, mas apresentam limitações ao lidar com estruturas mais complexas, como grafos. Nesse contexto, as Redes Neurais de Grafos (GNNs) Scarselli et al. (2008) surgem como uma ferramenta para capturar informações em dados estruturados. A peculiaridade das GNNs reside na habilidade de propagar e agregar conhecimento ao longo das arestas do grafo, incorporando as interações entre nodos vizinhos.

Essa abordagem dinâmica das GNNs tem aplicações amplas, desde classificação e regressão de nós em grafos Monti, Frasca, Eynard, Mannion, and Bronstein (Monti et al.) até previsão de links Jiang and Luo (2022). Seu impacto estende-se além do domínio da ciência da computação, abrangendo áreas diversas como na descoberta de antibióticos Stokes et al. (2020), simulações físicas Sanchez-Gonzalez et al. (2020) e sistemas de recomendação Eksombatchai et al. (2018).

As GNNs apresentam uma estrutura adaptada para a análise de dados estruturados em forma de grafos. Essa estrutura é fundamental para o funcionamento eficaz das GNNs em tarefas que envolvem relações complexas entre entidades interconectadas.

4.1 Níveis de Predição

Existem três categorias principais de tarefas de previsão em grafos: predição em nível de nó, em nível de aresta e em nível de grafo Sanchez-Lengeling et al. (2021). A Figura (4.1) ilustra essas três possibilidades. Outras tarefas relacionadas constituem áreas de pesquisa em constante evolução. Um exemplo é a geração de grafos Hu et al. (2020), ou ainda a explicação das predições feitas em um determinado grafo Ying et al. (2019).

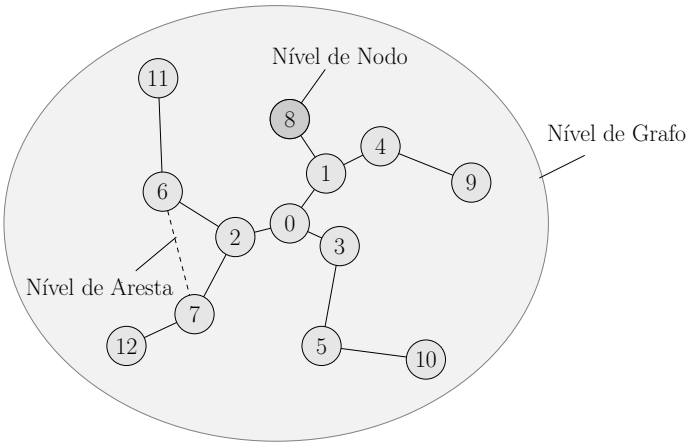


Figura 4.1: Níveis de Predição

4.1.1 Predição de Nós

A predição de nós em GNNs é uma área crucial, com ampla aplicação em diversos domínios. Neste contexto, a GNN é empregada para prever características específicas associadas a cada nó individual em um grafo. Na análise de sistemas de recomendação Eksombatchai et al. (2018), por exemplo, a GNN pode ser utilizada para prever os interesses de um usuário com base em suas conexões e atividades passadas.

Este trabalho de conclusão concentra-se particularmente na predição de nós, explorando a capacidade das GNNs em modelar e extrair informações relevantes associadas aos nós em um grafo.

4.1.2 Predição de Arestas

A predição de arestas em GNNs tem como objetivo principal inferir as relações entre pares de nós no grafo. Em uma tarefa de previsão de tráfego Jiang and Luo (2022), a GNN pode prever padrões de fluxo de tráfego, auxiliando na tomada de decisões para otimizar o funcionamento de sistemas de transporte.

4.1.3 Predição de Grafos

A predição de grafos em GNNs visa fazer previsões sobre o grafo como um todo. Essa abordagem é útil em tarefas que exigem uma compreensão global da estrutura do grafo, como classificação de grafos em categorias específicas. Por exemplo no campo da neurosciencia Bessadok et al. (2022), ela pode ser usada para prever padrões complexos de conectividade cerebral.

4.2 Relação com a Rede Neural Convolucional

Uma Rede Neural Convolucional (CNN) LeCun et al. (1998) destaca-se como um modelo especializado na classificação de imagens que executa operações de convolu-

ção em dados. Durante este processo, uma janela de convolução percorre a imagem, efetuando uma transformação nos dados para agregar informações locais. Em outras palavras, a janela de convolução atua como um filtro que captura padrões específicos na imagem. O resultado da convolução é uma nova representação da imagem, em que a dimensão é determinada pelo número de janelas de convolução aplicadas.

A Figura (4.2) oferece uma representação visual desse processo, utilizando uma imagem composta por 16 pixels. Cada pixel é analogamente representado como um nó, e estabelece conexões (arestas) com os pixels vizinhos. O resultado seria uma imagem reduzida a 4 pixels, sendo que cada pixel contém a informação agregada pela janela de convolução durante sua aplicação.

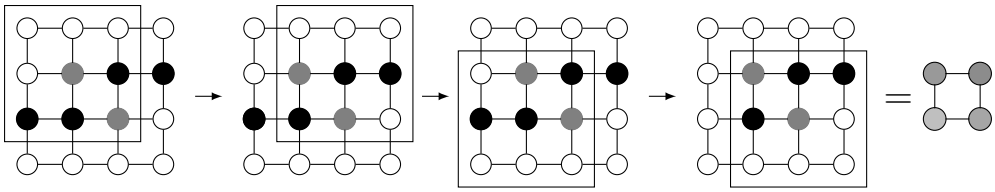


Figura 4.2: Janela de Convolução da CNN

As GNNs operam de maneira análoga às CNNs, na medida em que agregam informações dos nós vizinhos. No entanto, não é viável aplicar a arquitetura da CNN diretamente a grafos.

Diferentemente das imagens, cuja disposição regular dos pixels segue uma estrutura euclidiana que favorece a aplicação de CNNs, os grafos apresentam uma topologia mais complexa e variável, denominada não-euclidiana Liang et al. (2022). Em um grafo, cada nó pode estar conectado a um número variável de vizinhos, desafiando a abordagem uniforme das CNNs. Diante desse desafio, a GNN surge como uma solução adaptada para lidar com dados estruturados em grafos, possibilitando a eficiente agregação e propagação de informações entre os nós, independentemente de suas conectividades específicas. Esta concepção é exemplificada na Figura (4.3).

(a) CNN em Dados Euclidianos

(b) GNN em Dados Não-Euclidianos

Figura 4.3: Funcionamento da CNN vs GNN

4.3 *Embeddings* do Nó em um Grafo

Grafos desempenham um papel crucial na análise de dados complexos. Neste contexto, as *features*, ou covariáveis, associadas a nós, arestas e ao grafo oferecem uma perspectiva detalhada sobre as características associadas a cada um desses elementos. No contexto de predição a nível de nó, essas *features* são atualizadas a cada camada de GNN, sendo agregadas com as *features* dos nós vizinhos, formando assim os *embeddings* que são uma representação interna dos elementos. Em resumo, as *features* dos elementos são os *embeddings* na camada $l = 0$. A Figura (4.4) oferece uma representação visual dos diversos tipos de *embeddings* associados aos três elementos presentes em um grafo: nó, aresta e grafo.

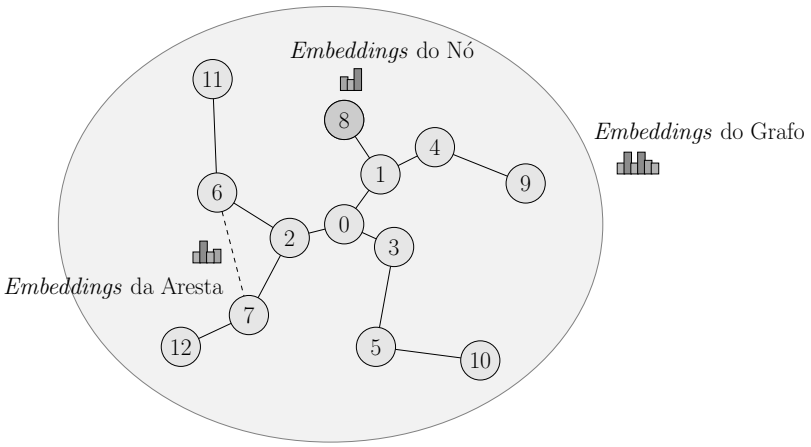


Figura 4.4: Representação dos *embeddings* em diferentes níveis

No contexto de tarefas a nível de nó, cada entidade é representada por um nó, e a GNN recebe informações específicas desse nó para complementar os dados de cada entidade. Por exemplo, em uma tarefa de classificação para uma determinada doença, cada nó pode representar um paciente, com as *features* sendo compostas por características pessoais como dados eletrônicos de saúde contendo informações sobre as doenças ou condições do paciente Lu and Uddin (2023). Essas informações são atualizadas a cada nova iteração da GNN com base na agregação de dados dos nós vizinhos Sanchez-Lengeling et al. (2021), resultando na formação de novas informações, conhecidas como *embeddings*.

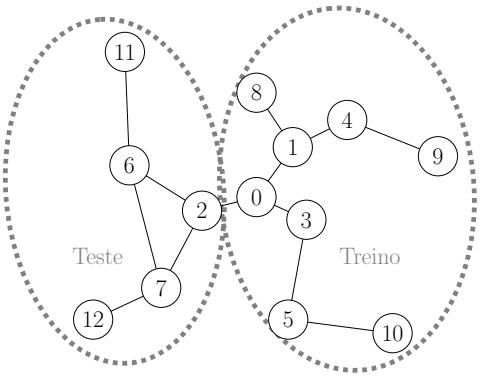
4.4 Divisão do Grafo em Treino e Teste

No âmbito de GNN para tarefas de nível de nó, a divisão do grafo em conjuntos de treino e teste assume um caráter especial. Isso ocorre porque, ao contrário dos dados tabulares, onde as observações são independentes, os nós em um grafo estão interconectados, estabelecendo relações entre si. Ao realizar essa divisão, é crucial levar em consideração essas conexões, ponderando cuidadosamente a abordagem mais apropriada. Nesse contexto, surgem duas opções distintas: a configuração transdutiva e a indutiva Leskovec (2019).

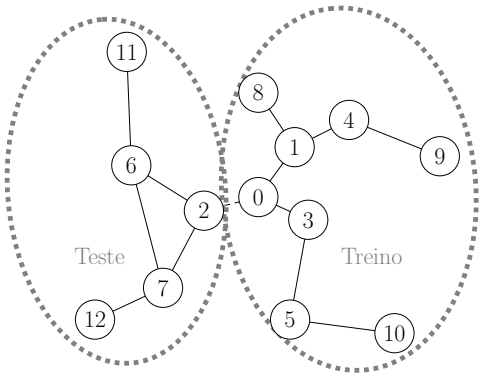
Na configuração transdutiva, a GNN é treinada com a capacidade de observar o grafo de entrada em ambos os conjuntos de dados, tanto de treinamento quanto de teste. Durante o processo de treinamento, a GNN tem acesso às *features* de todos os nós, proporcionando uma visão completa da estrutura do grafo. No entanto, há uma restrição em relação à variável alvo a ser predita, concentrando-se apenas na resposta do conjunto designado como treino.

Uma característica distintiva desse cenário é o mascaramento da variável alvo no conjunto de teste durante o treinamento da GNN. As respostas do conjunto de teste são intencionalmente ocultas, impedindo que a GNN tenha conhecimento direto dessas informações durante o treinamento. Esse mascaramento é uma estratégia que visa testar a capacidade da GNN de generalizar padrões e inferir corretamente os rótulos em nodos não observados durante o treinamento. Na Figura (4.5), apresenta-se uma representação visual da configuração transdutiva, na qual, apesar do nó 0 pertencer ao conjunto de treinamento e o nó 2 ao conjunto de teste, a aresta entre eles é mantida, e, consequentemente, a troca de informação.

Na abordagem indutiva, o grafo é dividido em segmentos distintos, reservando uma parcela para o treinamento e outra para o teste. Isso resulta na criação de dois grafos independentes: um utilizado exclusivamente durante o treinamento e outro destinado aos testes. Durante o processo de treinamento da GNN, é fundamental ressaltar que a rede não tem acesso a nenhum componente do grafo de teste, diferenciando-se, assim, do método transdutivo. Na Figura (4.5), é exibida uma ilustração da configuração indutiva, onde o nó 0 é associado ao conjunto de treinamento e o nó 2 ao conjunto de teste. Nesse cenário, a aresta entre eles é rompida, resultando na interrupção da troca de informações.



(a) Configuração transdutiva: a conexão da aresta entre o nó 0 e o nó 2 é mantida. Isso implica que as informações do nó 0 são empregadas tanto na previsão do nó 2 quanto no sentido inverso.



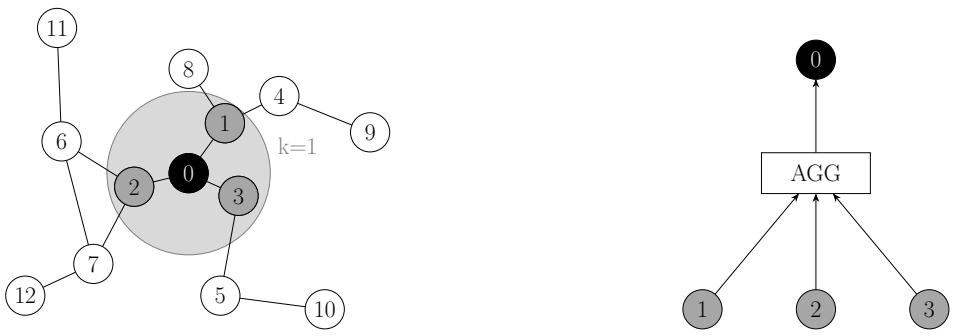
(b) Configuração indutiva: a conexão da aresta entre o nó 0 e o nó 2 é rompida. Dessa forma, as características do nó 0 não são utilizadas nem na previsão do nó 2, nem no sentido inverso.

Figura 4.5: Visualização da diferença entre a configuração transdutiva e indutiva.

4.5 Arquitetura Básica de uma Camada

Uma camada de Rede Neural em Grafos (GNN) é desenvolvida com base na técnica de *message-passing* Hamilton (2020). Nesse contexto, os *embeddings* dos vizinhos u do nó v, em que $u \in N(v)$, são reunidos por meio de uma operação de agregação. Isso resulta na formação de uma mensagem que incorpora a informação da vizinhança circundante. Posteriormente, os *embeddings* do nó v são atualizados, levando em consideração tanto os *embeddings* do próprio nó na camada anterior quanto a mensagem proveniente dos vizinhos.

A Figura (4.6) ilustra esse processo de *message-passing* para o nó focal 0. Os vizinhos a uma distância de um salto desse nó focal, denotado por $k = 1$ (onde k representa o tamanho do salto para os vizinhos), são representados pelos nós 1, 2 e 3, e suas informações são agregadas durante esse processo.



(a) Representação de um grafo, destacando o nó 0. No interior do círculo estão os vizinhos a um salto, ou seja, $k = 1$.

(b) Ilustração do processo de *message-passing*, compreendendo a agregação dos nós vizinhos do nó focal 0.

Figura 4.6: Visualização do *message-passing* para um exemplo de grafo com o nó focal 0. Em (a) é apresentada a disposição dos vizinhos a um salto desse nó, enquanto que em (b), é demonstrada a agregação dos vizinhos.

A atualização dos *embeddings* para o nó v pode ser expressa por:

$$\begin{aligned}\mathbf{h}_v^{(l+1)} &= \text{UPDATE}^{(l)}\Big(\mathbf{h}_v^{(l)}, \text{AGG}^{(l)}\Big(\Big\{\mathbf{h}_u^{(l)}, \forall u \in N(v)\Big\}\Big)\Big) \\ &= \text{UPDATE}^{(l)}\Big(\mathbf{h}_v^{(l)}, \mathbf{m}_{N(v)}^{(l)}\Big),\end{aligned}$$

em que $\mathbf{h}_v^{(l+1)}$ representa o *embedding* do nó v a ser atualizado para a próxima camada $l+1$. Os termos $\mathbf{h}_v^{(l)}$ e $\mathbf{h}_u^{(l)}$ denotam os *embeddings* do nó v e dos nós vizinhos u na camada l. O termo $\mathbf{m}_{N(v)}^{(l)}$ representa a "mensagem" agregada dos nós vizinhos de v. A função de agregação AGG(.) é comumente implementada como soma, máximo ou média, dependendo do contexto.

A função UPDATE(.) é responsável por combinar a "mensagem" $\mathbf{m}_{N(v)}^{(l)}$ com o *embedding* anterior $\mathbf{h}_v^{(l)}$ do próprio nó v para gerar o *embedding* atualizado. Essa

operação de atualização é crucial para incorporar informações da vizinhança e aprimorar a representação do nó na camada subsequente da GNN.

4.6 Múltiplas Camadas de GNN

Ao contruir camadas de GNN, o modelo recebe como entrada, na primeira camada, um grafo em que cada nó v possui seus atributos iniciais, representados por $\mathbf{h}_v^{(0)} = \mathbf{x}_v$. Em cada camada GNN da rede neural, esses *embeddings* são atualizados e agregam as mensagens de seus vizinhos $u \in N(v)$, transformando-se assim em novos *embeddings* $\mathbf{h}_v^{(l)}$. Esta dinâmica é visualizada na Figura (4.7).

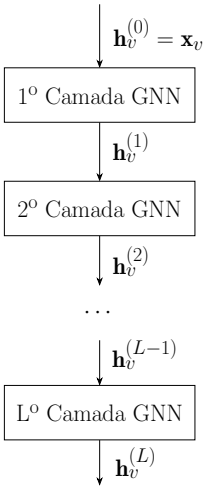
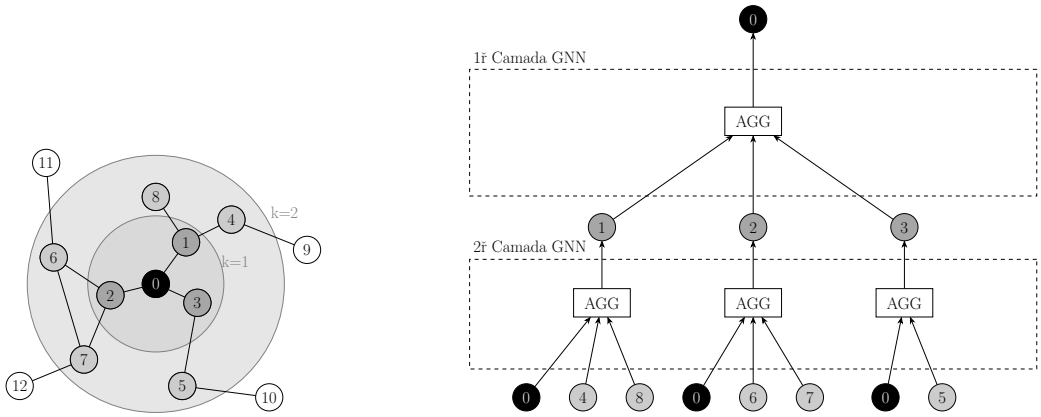


Figura 4.7: *Embeddings* do nó a cada camada de GNN.

A disseminação de mensagens e sua agregação ao longo de múltiplas camadas, mais precisamente, L camadas de Redes Neurais de Grafos (GNN), em que $L \geq 2$, não se limita apenas à incorporação de mensagens provenientes de vizinhos imediatos. Ela se estende para abranger os vizinhos de $1, 2, \ldots, L$ saltos. Essa ampliação implica que, à medida que sucessivas camadas de GNN são adicionadas, a influência de vizinhos mais distantes é progressivamente integrada ao processo de atualização dos *embeddings*.

A Figura (4.8) ilustra o funcionamento da arquitetura do modelo. Esta representação visual esclarece de que maneira os vizinhos do nó em foco exercem influência ao longo das camadas da GNN, destacando o funcionamento das interações que moldam a evolução dos *embeddings* ao longo do processo.



(a) Vizinhos de 1 (k=1) e 2 (k=2) saltos do nó 0

(b) *Message-passing* dos Vizinhos de 1 e 2 saltos do nó 0

Figura 4.8: Visualização de duas camadas de GNN, ilustrado como um único nó representado por 0, agrega as mensagens dos vizinhos de 1 e 2 saltos.

4.6.1 *Over-smoothing*

Quando são aplicadas várias camadas de GNNs sucessivas a um grafo, há o risco de ocorrer um *over-smoothing* Yang et al. (2020) Chen et al. (2020). Isso acontece porque, a cada camada, os *embeddings* dos nós são atualizados com base nas informações de seus vizinhos. Se esse processo é repetido muitas vezes, as representações dos nós podem se tornar muito homogêneas ou suavizadas, perdendo informações discriminativas únicas.

4.7 Camadas de GNN Clássicas

Esta seção se dedica à exploração de algumas das camadas GNNs, oferecendo uma análise mais aprofundada dos elementos que as compõem. As equações apresentadas nas definições a seguir foram formuladas conforme a documentação na biblioteca PyTorch Geometric.

4.7.1 Rede Neural Convolucional em Grafos (GCN)

Desenvolvida para operar em dados de grafos, a Rede Neural Convolucional em Grafos (GCN) Kipf and Welling (2016) é uma arquitetura que se destaca na tarefa de aprendizado em grafos.

A equação que descreve a propagação nas camadas da GCN é dada por:

$$\mathbf{H}^{(l)} = g\Big(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}\Big),$$

em que $g(.)$ é uma função de ativação e $\tilde{\mathbf{A}}$ representa a matriz de adjacência $\mathbf{A}$ de uma grafo não direcionado (consulte a Seção (2.2.1)) somada à matriz identidade $\mathbf{I}$, isto é, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. Essa adição assegura que cada nó leve em consideração não

apenas o *embedding* de seus vizinhos, mas também o seu próprio *embedding* durante o processo de convolução.

O termo $\tilde{\mathbf{D}}$ refere-se à matriz diagonal (consulte Seção (2.2.2)) dos graus dos nós, presentes na matriz de adjacência $\tilde{\mathbf{A}}$. Matematicamente $\tilde{D}_{ii} = \sum_{j=1}^{N} \tilde{A}_{ij}$, em que N representa o número de vértices presente no grafo.

$\mathbf{W}^{(l-1)}$ é uma matriz de pesos específica da camada $l-1$ e é aprendida ao longo do treinamento da GCN. Além disso, $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times D}$, em que D é a dimensão dos *embeddings* de cada nó.

4.7.2 GraphConv

O GraphConv Morris et al. (2019) descreve a propagação de informações em grafos da seguinte maneira:

$$\mathbf{h}_v^{(l)} = g\Big(\mathbf{B}^{(l)}\mathbf{h}_v^{(l-1)} + AGG^{(l)}\Big(\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}\Big)\Big),$$

em que v representa o nó alvo, e u são seus vizinhos, isto é, $u \in N(v)$. Além disso, a função $g(.)$ caracteriza a função de ativação empregada. Os termos $\mathbf{B}^{(l)}$ e $\mathbf{W}^{(l)}$ são matrizes de pesos da camada l, multiplicando os *embeddings* do nó v, $\mathbf{h}_v^{(l-1)}$, e dos nós vizinhos u, $\mathbf{h}_u^{(l-1)}$, respectivamente. Ambos os conjuntos de pesos são aprendidos durante o treinamento.

Além disso, $AGG^{(l)}$ representa uma operação de agregação, podendo assumir formas como soma, média e máximo, aplicada sobre os *embeddings* dos vizinhos u.

4.7.3 *Graph Attention Network* (GAT)

O *Graph Attention Network* (GAT) Velickovic et al. (2017) executa o processo de propagação de informações em grafos através da seguinte formulação:

$$\mathbf{h}_v^{(l)} = g\left(\alpha_{v,v}\mathbf{B}^{(l)}\mathbf{h}_v^{(l-1)} + \sum_{u \in N(v)} \alpha_{u,v}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}\right),$$

em que v representa o nó alvo, e u são seus vizinhos, isto é, $u \in N(v)$. Além disso, a função $g(.)$ caracteriza a função de ativação empregada. Os termos $\mathbf{B}^{(l)}$ e $\mathbf{W}^{(l)}$ são matrizes de pesos da camada l, multiplicando os *embeddings* do nó v, $\mathbf{h}_v^{(l-1)}$, e dos nós vizinhos u, $\mathbf{h}_u^{(l-1)}$, respectivamente. Ambos os conjuntos de pesos são aprendidos durante o treinamento.

Os coeficientes de atenção $\alpha_{u,v}$ são computados a partir da seguinte equação:

$$\alpha_{u,v} = \frac{\exp\Big(\text{LeakyReLU}\Big(\mathbf{a}_s^T\mathbf{B}^{(l)}\mathbf{h}_v^{(l-1)} + \mathbf{a}_t^T\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}\Big)\Big)}{\sum_{k \in N(v)} \exp\Big(\text{LeakyReLU}\Big(\mathbf{a}_s^T\mathbf{B}^{(l)}\mathbf{h}_v^{(l-1)} + \mathbf{a}_t^T\mathbf{W}^{(l)}\mathbf{h}_k^{(l-1)}\Big)\Big)},$$

em que esse mecanismo de atenção é uma rede neural *feedforward* de uma única camada, parametrizada pelos pesos $\mathbf{a}_s^T$ e $\mathbf{a}_t^T$. Além disso, é aplicado não linearidade utilizando LeakyReLU com uma inclinação negativa fixada em 0.2. Vale ressaltar que essa inclinação é um hiperparâmetro ajustável do modelo, permitindo personalização conforme necessário. Essa expressão é aplicável quando as arestas não apresentam informações multidimensionais. Para compreender o procedimento de

cálculo dos coeficientes de atenção em cenários em que as arestas possuem informações multidimensionais, recomenda-se a consulta ao trabalho de Velickovic et al. (2017).

4.7.4 *Graph Transformer Operator* (TransformerConv)

O *Graph Transformer Operator* (TransformerConv) Shi et al. (2020) funciona da seguinte maneira:

$$\mathbf{h}_v^{(l)} = g\left(\mathbf{B}^{(l)}\mathbf{h}_v^{(l-1)} + \sum_{u \in N(v)} \alpha_{v,u}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}\right),$$

em que v representa o nó alvo, e u são seus vizinhos, isto é, $u \in N(v)$. Além disso, a função $g(.)$ caracteriza a função de ativação empregada. Os termos $\mathbf{B}^{(l)}$ e $\mathbf{W}^{(l)}$ são matrizes de pesos da camada l, multiplicando os *embeddings* do nó v, $\mathbf{h}_v^{(l-1)}$, e dos nós vizinhos u, $\mathbf{h}_u^{(l-1)}$, respectivamente. Ambos os conjuntos de pesos são aprendidos durante o treinamento.

Os coeficientes de atenção são calculados a partir da seguinte operação:

$$\alpha_{v,u} = \text{Softmax}\left(\frac{(\mathbf{W}_s\mathbf{h}_v^{(l-1)})^T(\mathbf{W}_t\mathbf{h}_u^{(l-1)})}{\sqrt{d}}\right),$$

em que $\mathbf{W}_s$ e $\mathbf{W}_t$ são matrizes de peso treináveis, d é um termo de regularização, para mais informações consulte Shi et al. (2020).

4.7.5 *GENeralized Graph Convolution* (GENConv)

O *GENeralized Graph Convolution* (GENConv) Li et al. (2020) opera de acordo com a seguinte formulação:

$$\alpha_{i,j} = \text{MLP}\Big(\mathbf{h}_v^{(l-1)} + \text{AGG}\Big(\Big\{\text{ReLU}\Big(\mathbf{h}_u^{(l-1)} + \mathbf{e}_{uv}^{(l-1)}\Big)\Big\}\Big)\Big),$$

em que v representa o nó alvo, e u são seus vizinhos, isto é, $u \in N(v)$. Além disso, a função $g(.)$ caracteriza a função de ativação empregada. Os *embeddings* do nó v são representados por $\mathbf{h}_v^{(l-1)}$, e dos nós vizinhos u, $\mathbf{h}_u^{(l-1)}$. O termo $\mathbf{e}_{uv}^{(l-1)}$ representa os embeddings da aresta entre os nós u e v, se houver. Além disso, MLP(.) representa uma rede neural com várias camadas de neurônios, organizadas em uma estrutura hierárquica (consulte Seção (3.9)).

4.8 GNNs no Contexto Espacial

As GNNs oferecem uma gama diversificada de aplicações. Ao considerarmos dados espaciais de áreas, como municípios, estados ou outras regiões agregadas, podemos interpretar cada uma dessas áreas como uma entidade conectada às suas vizinhanças. Dessa forma, é possível representar esses dados de área como um grafo e empregar a GNN para a previsão desses nós.

Um exemplo dessa aplicação é a utilização da GNN para a predição de eventos pandêmicos, como o covid-19, por região Nguyen et al. (2023). Outra aplicação

relevante está na previsão de links no contexto espacial, como a previsão de congestionamentos de tráfego Bui et al. (2022).

Além das aplicações preditivas, quando o foco reside na avaliação do grau de dependência espacial sem a necessidade de realizar previsões, é possível empregar uma rede neural com pesos treináveis nas arestas. Isso proporciona a capacidade de quantificar o grau de dependência espacial que um município possui com relação a cada um de seus vizinhos.

Adicionalmente, a utilização de GNNs que incorporam componentes de cabeças de atenção também oferecem a oportunidade de medir o grau de dependência espacial entre pares de elementos. Por exemplo, no caso da presença de elementos $\alpha_{v,u}$ em modelos como GAT e TransformerConv, esse componente pode ser interpretado como o nível de importância da área u na previsão de área v. Essa interpretação transforma-se em um tipo de métrica para medir a dependência espacial.

5 Resultados

5.1 Projeto "Saúde com Agente"

O Projeto "Saúde com Agente" é uma colaboração entre a Universidade Federal do Rio Grande do Sul (UFRGS) e o Ministério da Saúde, juntamente com o Conasems (Conselho Nacional de Secretarias municipais de Saúde), destinado a oferecer os cursos de Agente Comunitário de Saúde (ACS) e de Vigilância em Saúde com Ênfase no Combate às Endemias (ACE). Esses cursos são ministrados em diversos estabelecimentos de saúde em todo o território brasileiro, o que resulta na geração de dados espaciais. O programa disponibiliza cursos em todo o Brasil, embora em alguns casos a participação não tenha ocorrido devido à ausência de alunos interessados ou à falta de adesão por parte do municípios ao projeto.

5.2 Fonte de Dados

Os municípios nos quais esses cursos são realizados passam a ser tratados como nós em um grafo, tendo uma aresta conectando dois municípios se eles compartilham fronteiras. A variável que o modelo GNN visa predizer é o número de alunos em cada um desses municípios. Como *features* para os nós, características demográficas desses municípios foram consideradas, obtidas de fontes como o Instituto Brasileiro de Geografia e Estatística (IBGE), Departamento de Informática do Sistema Único de Saúde (DATASUS) e Relação Anual de Informações Sociais (RAIS). No total, a fonte de dados abrange 5203 municípios.

A seleção das variáveis para compor as *features* de cada município priorizou aquelas que poderiam apresentar alguma relação com a variável resposta, a qual é representada pelo número de alunos. As variáveis selecionadas estão listadas na Tabela (5.1). Nesse contexto, foram destacadas variáveis ligadas à saúde, como o número de estabelecimentos de saúde e de profissionais na área. A fundamentação para isso reside no fato de que os cursos são disponibilizados nos estabelecimentos e conduzidos por profissionais da saúde.

Aspectos educacionais e socioeconômicos foram ponderados, uma vez que municípios com elevadas taxas de analfabetismo trazem restrições na participação nos cursos. Variáveis relacionadas ao rendimento médio também se mostram relevantes, dado que podem influenciar na busca por qualificação profissional.

Adicionalmente, variáveis mais abrangentes, como o Índice de Desenvolvimento Humano (IDH) e o Índice de Gini, foram incorporadas na análise, considerando

seu potencial impacto sobre o número de alunos. Esses indicadores refletem as condições socioeconômicas gerais do município e, portanto, podem influenciar de maneira significativa o interesse e a participação nos cursos disponíveis em cada localidade.

Variável	**Fonte**	**Ano**
Número de Alunos	Projeto "Saúde com Agente"	2023
Esperança de vida ao nascer	Censo	2010
IDHM	Censo	2010
IDHM Renda	Censo	2010
IDHM Longevidade	Censo	2010
IDHM Educação	Censo	2010
Subíndice de frequência escolar - IDHM Educação	Censo	2010
Subíndice de escolaridade - IDHM Educação	Censo	2010
Taxa de analfabetismo - 25 anos ou mais de idade	Censo	2010
Renda per capita	Censo	2010
Índice de Theil-L	Censo	2010
Índice de Gini	Censo	2010
Rendimento médio dos ocupados	Censo	2010
Produto Interno Bruto per capita	RAIS	2013
Produto Interno Bruto per capita	RAIS	2014
Produto Interno Bruto per capita	RAIS	2016
Total de estabelecimentos de saúde	Ministério da Saúde	2023
Número de profissionais da saúde	DATASUS	2022

Tabela 5.1: Quadro de Variáveis que compõem as *features* dos nós do grafo.

5.3 Implementação dos Modelos

A implementação dos modelos foi realizada em Python, aproveitando diversas bibliotecas populares para aprendizado de máquina e processamento de dados. A versão específica do Python utilizada foi a 3.11.3. Dentre as bibliotecas empregadas, destacam-se:

- **torch_geometric (v2.4.0):** Utilizada para a implementação dos modelos GNN e a divisão transdutiva dos dados em conjuntos de treino e teste.
- **torch (v2.1.2):** Empregada para implementar as camadas de redes neurais multicamadas (MLP) e as camadas lineares, calcular a função de perda (*loss*) e utilizar o otimizador durante o treinamento.
- **numpy (v1.23.5) e pandas (v2.0.2):** Utilizadas para executar operações como a manipulação numérica e a manipulação e análise de dados tabulares durante a implementação.

A implementação foi conduzida em uma máquina equipada com um processador 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz e 36GB de memória RAM. O sistema operacional utilizado é o Windows 11, na versão 22H2.

O código da implementação encontra-se disponível no repositório do GitHub, no seguinte link:

https://github.com/leticiaputtlitz/GNN_spatial_data.

5.4 Pré-Processamento dos Dados

Os dados foram divididos em conjuntos de treinamento e teste por meio da abordagem transdutiva (veja a Seção (4.4)), com 70% dos dados destinado ao treino e 30% ao teste, resultando em 3642 municípios para o conjunto de treino e 1561 para o teste.

Vale notar que a normalização das covariáveis foi testada, não demonstrando impacto no desempenho dos modelos.

5.5 Configurações dos Modelos

A definição dos hiperparâmetros, envolvendo elementos como o número de camadas, a quantidade de neurônios e a taxa de aprendizagem, foi conduzida por meio de uma técnica conhecida como *Hyperopt* Bergstra et al. (2013). A abordagem de treinamento em lote foi adotada para todos os modelos, ou seja, o conjunto de treino completo foi utilizado em cada iteração do algoritmo.

O modelo MLP, que trata as observações de forma independente, foi empregado para comparação com os modelos de GNN, com o objetivo de analisar se a incorporação de conexões melhora a acurácia das previsões. Adicionalmente, foram testados métodos de regressão baseados em árvores, tais como o *Random Forest* Hastie et al. (2009), XGBoost Chen et al. (2015), e LightGBM Ke et al. (2017), no entanto, não demonstraram um desempenho satisfatório, sendo assim, não serão apresentados neste trabalho.

Além disso, foram analisadas diferentes arquiteturas de GNN, incluindo GCN, GraphConv, GAT, TransformerConv e GENConv, conforme discutido na Seção (4.7). Foram explorados modelos compostos unicamente por camadas de GNN, assim como modelos que integram camadas de GNN com camadas lineares. Vale destacar que uma camada linear executa uma transformação linear nas entradas, envolvendo a multiplicação das entradas pelos pesos e a adição de um *bias*. A semente foi fixada em 42 em todos os modelos, assegurando a reprodutibilidade dos resultados.

Todos os modelos foram treinados utilizando a função de perda MSE, conforme definido na Equação (3.3) (consulte a Seção (3.4)). O otimizador Adam (consulte Seção (3.6)) foi empregado em todos os modelos, com as únicas variações ocorrendo nos hiperparâmetros associados à taxa de aprendizado e ao *Weight Decay* (Regularização L2, consulte Seção (3.8.4)). A não-linearidade foi aplicada utilizando a função ReLU (consulte Seção (3.3)) após cada camada.

Em nenhuma das configurações de camadas lineares, seja para modelos constituídos unicamente por camadas lineares (MLP) ou para modelos com GNN seguidas

por camadas lineares, foi implementado o uso de *dropout*, um regulador que desativa aleatoriamente alguns neurônios durante o treinamento, conforme detalhado na Seção (3.8.4).

5.5.1 Rede Neural de Multicamadas (MLP)

Os hiperparâmetros sujeitos a ajuste incluem:

- **Número de camadas e de neurônios por camada:** Determinam a complexidade e capacidade de aprendizado da rede neural;
- **Taxa de aprendizagem do otimizador:** Controla a magnitude dos passos durante o treinamento, impactando a convergência do modelo.;
- ***Weight Decay*:** Aplica uma penalidade aos pesos para prevenir o *overfitting*, representando uma forma de regulação L2 (consulte Seção (3.8.4));
- **Número de épocas:** Define a quantidade de iterações sobre o conjunto de treinamento.

Os valores selecionados para os hiperparâmetros deste modelo estão detalhados na Tabela (5.2).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado[2]
Camadas Ocultas	0, 1 ou 2	2
Neurônios	8, 16 ou 32	32
Taxa de aprendizagem	0.001 a 0.01	0.002
Weight Decay	0 a 0.1	0.03
Épocas	1000 ou 2000	1000

Tabela 5.2: A tabela exibe os valores considerados durante a otimização de hiperparâmetros para o modelo MLP[1] e o hiperparâmetro ótimo encontrado[2]. O tempo de execução para a otimização de hiperparâmetro foi de 3min.

5.5.2 Rede Neural Convolucional em Grafos (GCN)

Para o modelo com apenas camadas de GCN, os hiperparametros incluem:

- **Número de camadas e de neurônios por camada**;
- **Taxa de aprendizagem do otimizador**;
- ***Weight Decay***;
- **Número de épocas**;
- **Dropout:** Um regularizador que desliga alguns neurônios durante o treinamento de forma aleatória (consulte Seção (3.8.4)).

Os valores selecionados para os hiperparâmetros desse modelo estão detalhados na Tabela (5.3).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado[2]
Camadas Ocultas	0, 1 ou 2	0
Neurônios	8, 16 ou 32	8
Taxa de aprendizagem	0.001 a 0.01	0.009
Weight Decay	0 a 0.1	0.0006
Épocas	1000 ou 2000	2000
Dropout	0 a 0.6	0.03

Tabela 5.3: A tabela exibe os valores considerados durante a otimização de hiperparâmetros[1] para o modelo GCN, bem como o hiperparâmetro ótimo encontrado[2]. O tempo de execução da otimização de hiperparâmetros foi de 33min .

Os hiperparâmetros escolhidos para o modelo GCN + Linear são apresentados na Tabela (5.4).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado[2]
Camadas (GCN)	0, 1 ou 2	1
Neurônios (GCN)	8, 16 ou 32	16
Taxa de aprendizagem	0.001 a 0.01	0.006
Weight Decay	0 a 0.1	0.06
Épocas	1000 ou 2000	2000
Dropout	0 a 0.6	0.3
Camadas (linear)	1, 2 ou 3	1
Neurônios (linear)	8, 16 ou 32	32

Tabela 5.4: A tabela exibe os valores considerados durante a otimização de hiperparâmetros[1] para o modelo GCN + Linear e o hiperparâmetro ótimo encontrado[2]. Com a adições das camadas lineares neste modelo, a arquitetura é composta por uma camada de entrada GCN, seguida por camadas ocultas GCN e camadas ocultas lineares e, por fim, uma camada de saída linear. O tempo de execução para a otimização dos hiperparâmetros foi de 63min.

5.5.3 GraphConv

Para o modelo com apenas camadas de GraphConv, os hiperparâmetros incluem:

- **Número de camadas e de neurônios por camada**;
- **Taxa de aprendizagem do otimizador**;
- ***Weight Decay***;
- **Número de épocas**;
- **Dropout**;
- **aggr**: Esquema de agregação a ser utilizado, podendo ser “add” para soma, “mean” para média e “max” para máximo.

Os hiperparâmetros adotados para este modelo são apresentados na Tabela (5.5).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado [2]
Camadas	0, 1 ou 2	2
Neurônios	8, 16 ou 32	8
Taxa de aprendizagem	0.001 a 0.01	0.008
Weight Decay	0 a 0.1	0.06
Épocas	1000 ou 2000	2000
Dropout	0 a 0.6	0.26

Tabela 5.5: A tabela exibe os valores considerados na otimização de hiperparâmetros[1] para o modelo GraphConv e o hiperparâmetro ótimo encontrado[2]. O tempo de execução foi de 27min.

Para o modelo GraphConv + Linear, os hiperparâmetros selecionados estão indicados na Tabela (5.6).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado [2]
Camadas (GraphConv)	0, 1 ou 2	0
Neurônios (GraphConv)	8, 16 ou 32	8
Taxa de aprendizagem	0.001 a 0.01	0.001
Weight Decay	0 a 0.1	0.04
Épocas	1000 ou 2000	1000
Dropout	0 a 0.6	0.42
Camadas (linear)	1, 2 ou 3	2
Neurônios (linear)	8, 16 ou 32	32

Tabela 5.6: A tabela exibe os valores considerados durante a otimização de hiperparâmetros[1] para o modelo GraphConv + Linear e o hiperparâmetro ótimo encontrado[2]. Com a adições das camadas lineares neste modelo, a arquitetura é composta por uma camada de entrada GraphConv, seguida por camadas ocultas GraphConv e camadas ocultas lineares e, por fim, uma camada de saída linear. O tempo de execução para a otimização dos hiperparâmetros foi de 15min.

5.5.4 *Graph Attention Network* (GAT)

Para o modelo com apenas camadas de GAT, os hiperparâmetros incluem:

- **Número de camadas e neurônios por camada**
- **heads**: Quantidade de cabeças de atenção.
- **Taxa de aprendizagem do otimizador**;
- ***Weight Decay***;
- **Número de épocas**;
- **Dropout**;

Os hiperparâmetros selecionados para este modelo estão detalhados na Tabela (5.7).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado[2]
Camadas	0, 1 ou 2	2
Neurônios	8, 16 ou 32	16
Taxa de aprendizagem	0.001 a 0.01	0.002
Weight Decay	0 a 0.1	0.03
Épocas	1000 ou 2000	2000
heads	1, 2 ou 5	2
Dropout	0 a 0.6	0.23

Tabela 5.7: A tabela exibe os valores considerados durante a otimização de hiperparâmetros[1] para o modelo GAT e o hiperparâmetro ótimo encontrado[2]. O tempo de execução foi de 144min.

Os hiperparâmetros escolhidos para o modelo GAT + Linear estão listados na Tabela (5.8).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado[2]
Camadas (GAT)	0, 1 ou 2	2
Neurônios (GAT)	8, 16 ou 32	8
Taxa de aprendizagem	0.001 a 0.01	0.002
Weight Decay	0 a 0.1	0.03
Épocas	1000 ou 2000	1000
heads	1, 2 ou 5	5
Dropout	0 a 0.6	0.26
Camadas (linear)	1, 2 ou 3	1
Neurônios (linear)	8, 16 ou 32	16

Tabela 5.8: A tabela exibe os valores considerados durante a otimização de hiperparâmetros[1] para o modelo GAT + Linear e o hiperparâmetro ótimo encontrado[2]. O tempo de execução foi de 157min.

5.5.5 *Graph Transformer Operator* (TransformerConv)

Para o modelo com apenas camadas de TransformerConv, os hiperparâmetros incluem:

- **Número de camadas e de neurônios por camada**
- **Taxa de aprendizagem do otimizador**;
- ***Weight Decay***;
- **Número de épocas**;
- **Dropout**;
- **heads**;
- **dropout**.

Os hiperparâmetros selecionados para este modelo são apresentados na Tabela (5.9).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado[2]
Camadas	0,1 ou 2	1
Neurônios	8, 16 ou 32	32
Taxa de aprendizagem	0.001 a 0.01	0.009
Weight Decay	0 a 0.1	0.04
Épocas	1000 ou 2000	2000
heads	1, 2 ou 5	2
Dropout	0 a 0.6	0.32

Tabela 5.9: A tabela exibe os valores considerados durante a otimização de hiperparâmetros[1] para o modelo TransformerConv e o hiperparâmetro ótimo encontrado[2]. O tempo de execução foi de 145min.

Os hiperparâmetros escolhidos para o modelo TransformerConv + Linear são apresentados na Tabela (5.10).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado[2]
Camadas (TransformerConv)	0, 1 ou 2	2
Neurônios (TransformerConv)	8, 16 ou 32	8
Taxa de aprendizagem	0.001 a 0.01	0.007
Weight Decay	0 a 0.1	0.03
Épocas	1000 ou 2000	1000
heads	1, 2 ou 5	2
Dropout	0 a 0.6	0.07
Camadas (linear)	1, 2 ou 3	3
Neurônios (linear)	8, 16 ou 32	16

Tabela 5.10: A tabela exibe os valores considerados durante a otimização de hiperparâmetros[1] para o modelo TransformerConv + Linear e o hiperparâmetro ótimo[2]. O tempo de execução foi de 145min.

5.5.6 *GENeralized Graph Convolution* (GENConv)

Para o modelo com apenas camadas de GENConv, os hiperparâmetros incluem:

- **Configuração de Camadas e Neurônios**
- **Taxa de aprendizagem do otimizador**;
- ***Weight Decay***;
- **Número de épocas**;
- **Dropout**;

Os hiperparâmetros adotados para este modelo são apresentados na Tabela (5.11).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado[2]
Camadas	0, 1 ou 2	0
Neurônios	8, 16 ou 32	32
Taxa de aprendizagem	0.001 a 0.1	0.008
Weight Decay	0 a 0.1	0.1
Épocas	1000 ou 2000	2000
Dropout	0 a 0.6	0.35

Tabela 5.11: A tabela exibe os valores considerados durante a otimização de hiperparâmetros[1] para o modelo GENConv e o hiperparâmetro ótimo encontrado[2]. O tempo de execução foi de 86min.

Os hiperparâmetros selecionados para o modelo GENConv + Linear são apresentados na Tabela (5.12).

Hiperparâmetro	Valores considerados na otimização[1]	Hiperparâmetro otimizado[2]
Camadas (GENConv)	0, 1 ou 2	1
Neurônios (GENConv)	8, 16 ou 32	32
Taxa de aprendizagem	0.001 a 0.01	0.004
Weight Decay	0 a 0.1	0.08
Épocas	1000 ou 2000	2000
Dropout	0 a 0.6	0.3
Camadas (linear)	1, 2 ou 3	2
Neurônios (linear)	8, 16 ou 32	8

Tabela 5.12: A tabela exibe os valores considerados durante a otimização de hiperparâmetros[1] para o modelo GENConv + Linear e o hiperparâmetro ótimo encontrado[2]. O tempo de execução foi de 108min.

5.6 Resultados

Na Tabela (5.13), estão apresentados os modelos com as configurações de hiperparâmetros descritas na Seção (5.5). Os dados fornecidos na tabela incluem o modelo, o MSE no conjunto de teste utilizando a semente 42, seguido do MSE médio (com desvio padrão entre parênteses) obtido a partir de 30 simulações com sementes diferentes sorteadas aleatoriamente, além do tempo total de execução das simulações. Essa abordagem visa analisar como a mesma configuração se comporta para diferentes sementes fixadas. Observa-se a presença de um desvio padrão elevado em alguns modelos, sugerindo uma sensibilidade à inicialização dos pesos que é diretamente influenciada pela semente. Além disso, houve um aumento no erro médio calculado a partir das simulações, indicando que a melhor configuração encontrada com a semente 42 pode não ser uma escolha ótima para as demais sementes. A simulação foi feita com a configuração do modelo que gerou melhor desempenho utilizando a semente 42.

Modelo	MSE$^{(1)}$	MSE$^{(2)}$	Tempo$^{(3)}$
MLP	1945.48	3679.94 (675.7)	3min
GCN	15645.37	16539.89 (1540.6)	13min
GraphConv	2841.21	263833.52 (479655.1)	21min
GAT	17725.24	29355.78 (35038.2)	86min
TransformerConv	2081.66	31041.66 (87945.9)	115min
GENConv	**707.37**	5847.59 (7439.5)	59min
GCN + Linear	14080.36	15740.18 (1750.3)	28min
GraphConv + Linear	**877.50**	**2175.72** (1092.1)	6min
GAT + Linear	8993.63	21524.92 (14667.3)	130min
TransformerConv + Linear	1373.91	3149.30 (1201.5)	37min
GENConv + Linear	**824.21**	**2034.06 (792.1)**	102min

Tabela 5.13: A tabela exibe os resultados dos modelos testados, em questão de MSE e Tempo. MSE calculado no conjunto de teste para o respectivo modelo com semente igual a 42$^{(1)}$. MSE no conjunto de teste apresentado na forma de média (desvio padrão), obtido a partir de 30 simulações com diferentes sementes$^{(2)}$. Tempo de execução das simulações em minutos$^{(3)}$.

A seguir, serão exibidos gráficos e mapas com análises mais aprofundadas dos modelos com os melhores desempenhos e do modelo tabular MLP. Ao observar os gráficos, torna-se evidente que uma observação está impactando negativamente o MSE no modelo MLP, enquanto as demais observações estão razoavelmente bem preditas. No caso dos modelos GNN, observa-se uma capacidade superior de predição para essa observação *outlier*.

Inicialmente, apresenta-se um mapa que representa o número real de alunos para cada município no conjunto de teste. É relevante destacar que o conjunto de treino foi omitido no mapa, ressaltando a dispersão geográfica do conjunto de teste em todo o Brasil. A apresentação do número de alunos ocorre por meio de quantis, facilitando a visualização dos valores. Essa abordagem revela-se particularmente útil, uma vez que o limite superior do número de alunos no conjunto de teste atinge aproximadamente 5000, um valor mais elevado em comparação aos demais, o que poderia distorcer a interpretação dos valores intermediários, fazendo com que fossem erroneamente percebidos como baixos devido à sua magnitude em relação ao limite superior.

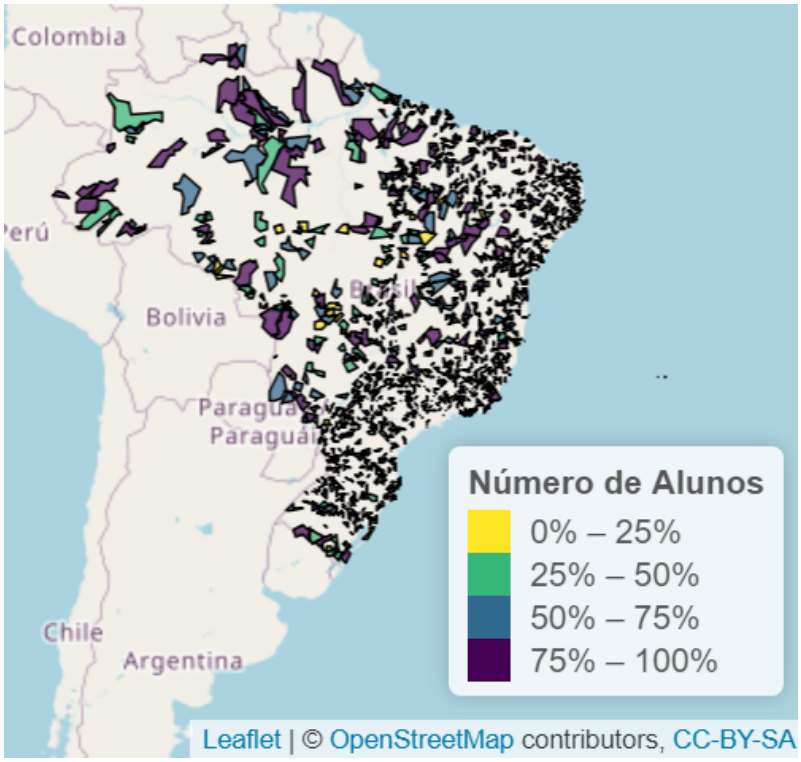


Figura 5.1: Mapa do número de alunos verdadeiro.

A seguir, os mapas com a exibição do número de alunos para o modelo MLP, bem como dos modelos GNN com melhor desempenho.

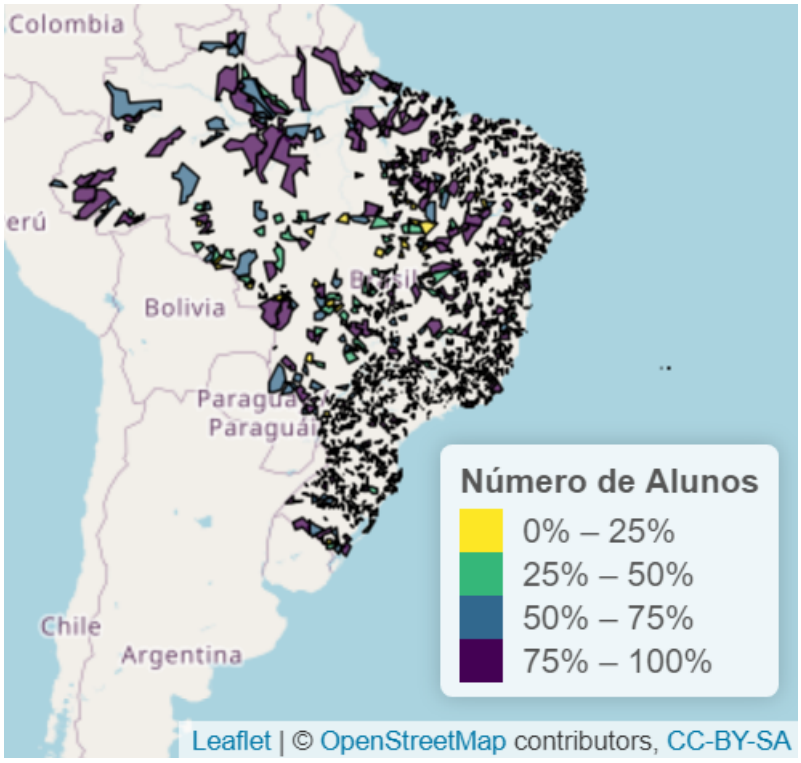


Figura 5.2: Mapa do número de alunos predito pelo modelo MLP.

O mapa referente aos erros de predição do modelo MLP é apresentado a seguir,

incluindo uma visualização ampliada que destaca o maior erro cometido pelo modelo que foi no município do Rio de Janeiro (RJ). O cálculo desse erro foi baseado na diferença absoluta entre o valor previsto e o valor real.

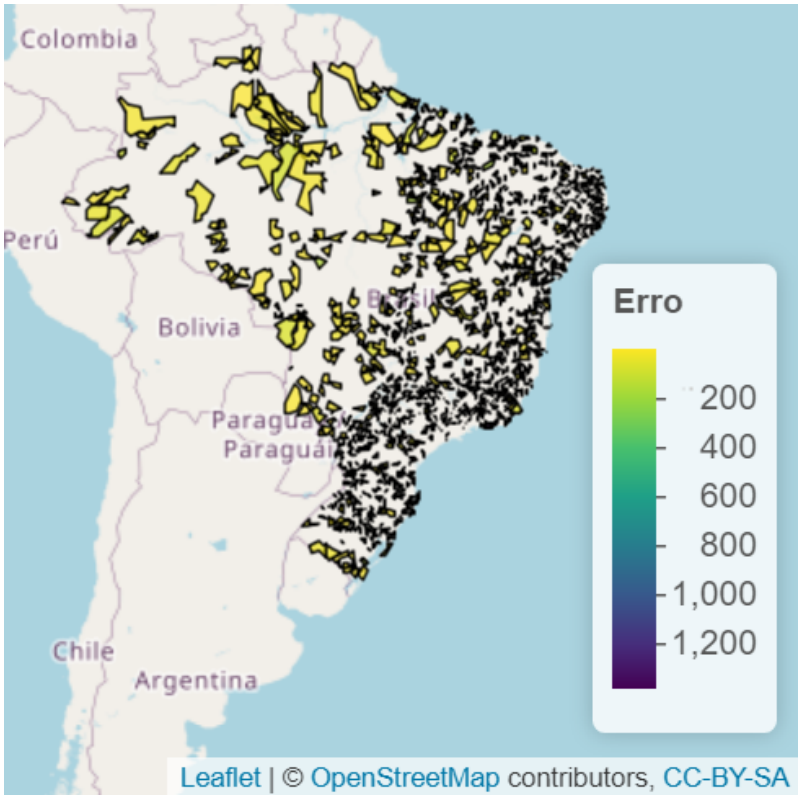


Figura 5.3: Mapa do erro de predição do modelo MLP.

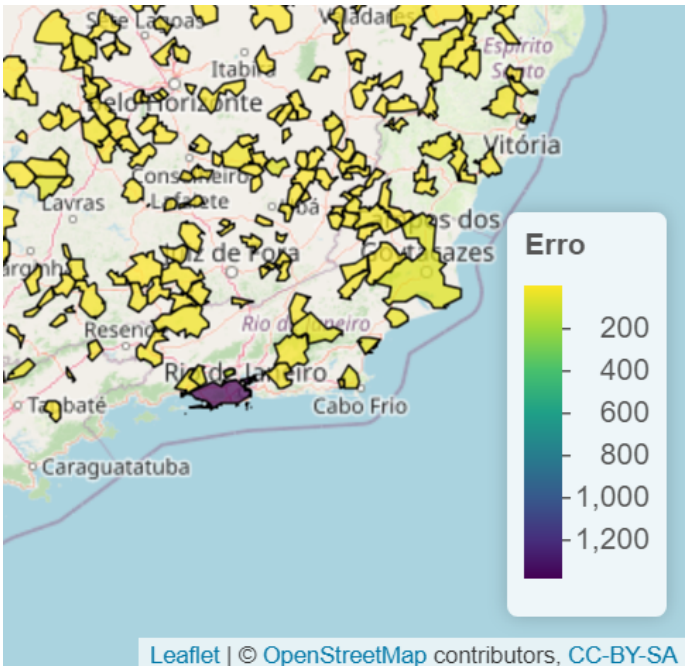


Figura 5.4: Mapa do erro de predição do modelo MLP.

Na sequência, são explorados mapas relativos à análise do modelo GraphConv

+ Linear.

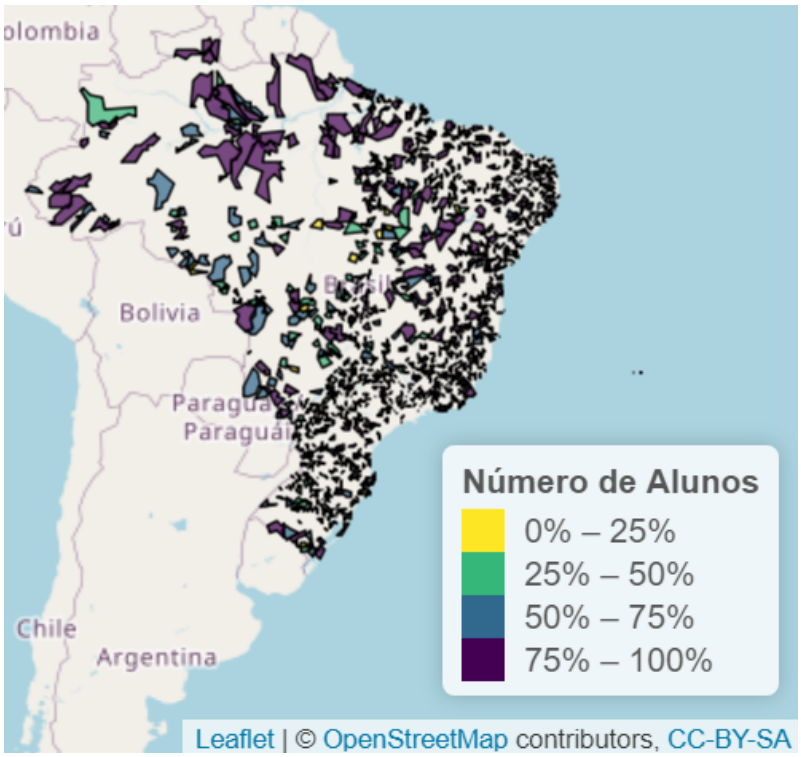


Figura 5.5: Mapa do número de alunos predito pelo modelo GraphConv + Linear.

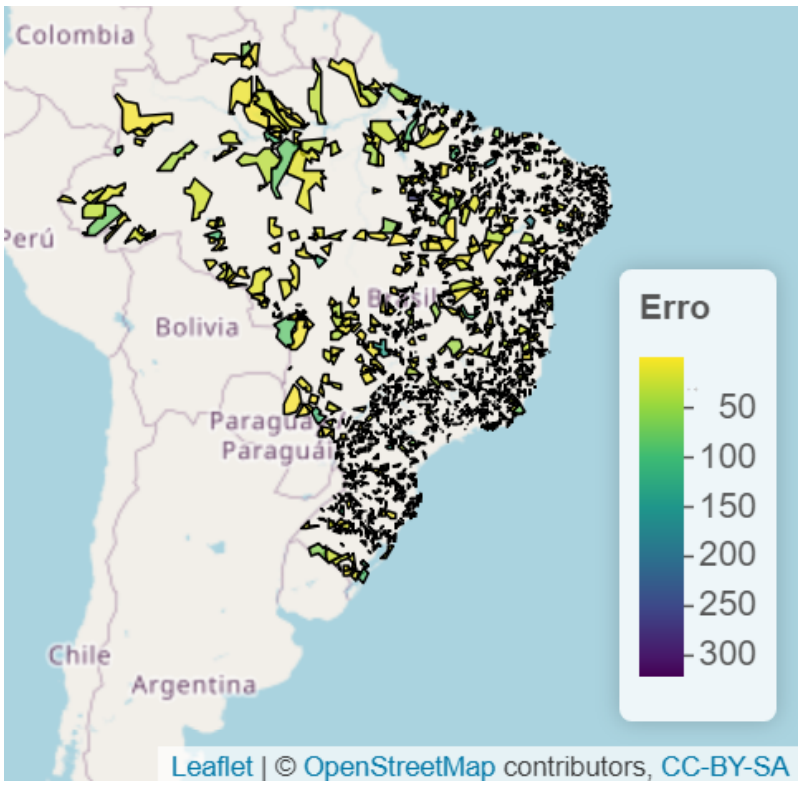


Figura 5.6: Mapa do erro de predição do modelo GraphConv + Linear. O limite superior do erro é menor do que no Modelo MLP.

Na continuidade, são investigados os mapas referentes à análise do modelo GENConv.

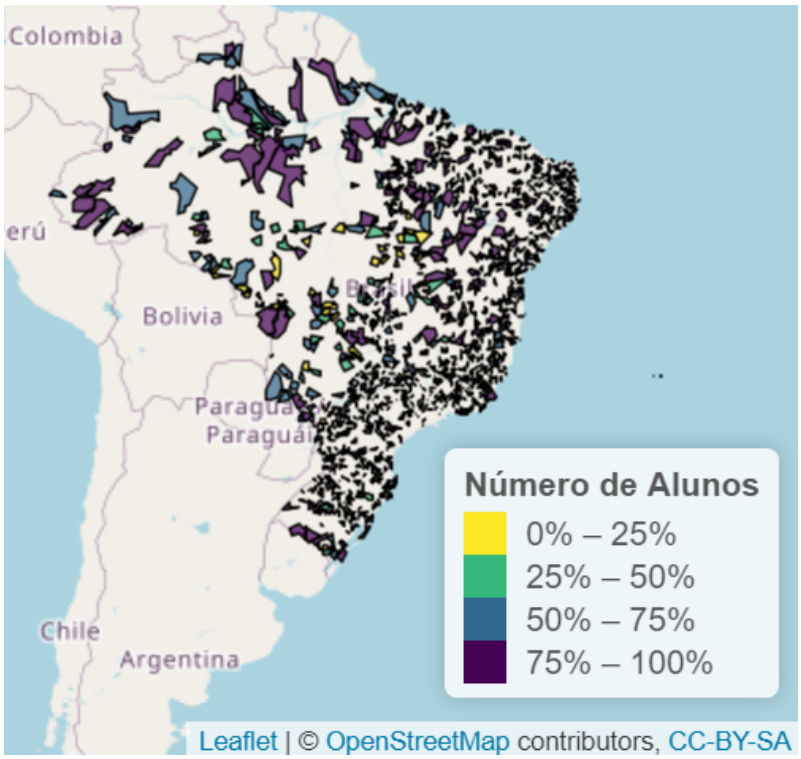


Figura 5.7: Mapa do número de alunos predito pelo modelo GENConv.

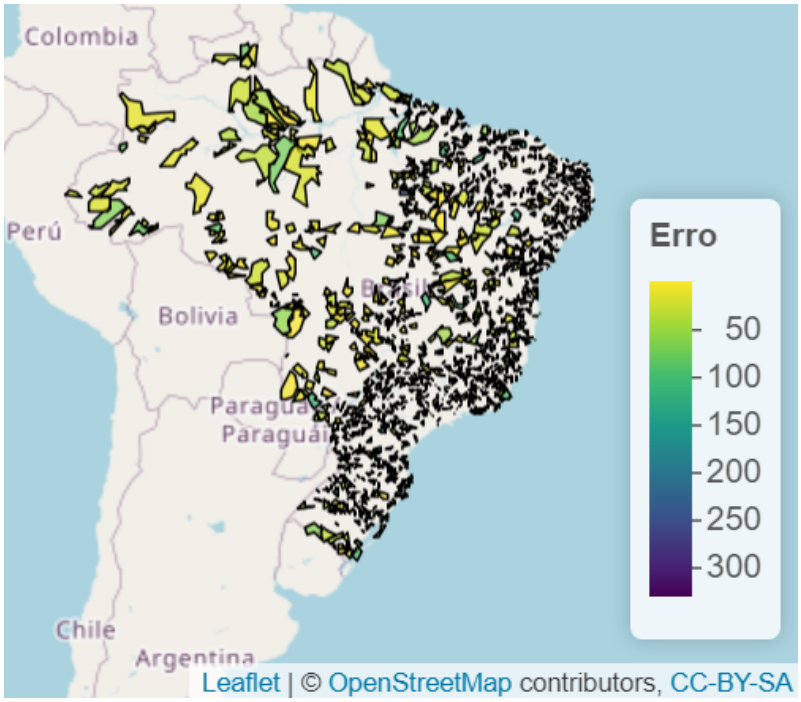


Figura 5.8: Mapa do erro de predição do modelo GENConv. O limite superior do erro é menor do que no Modelo MLP.

Na sequência, são explorados os mapas relativos à análise do modelo GENConv

+ Linear.

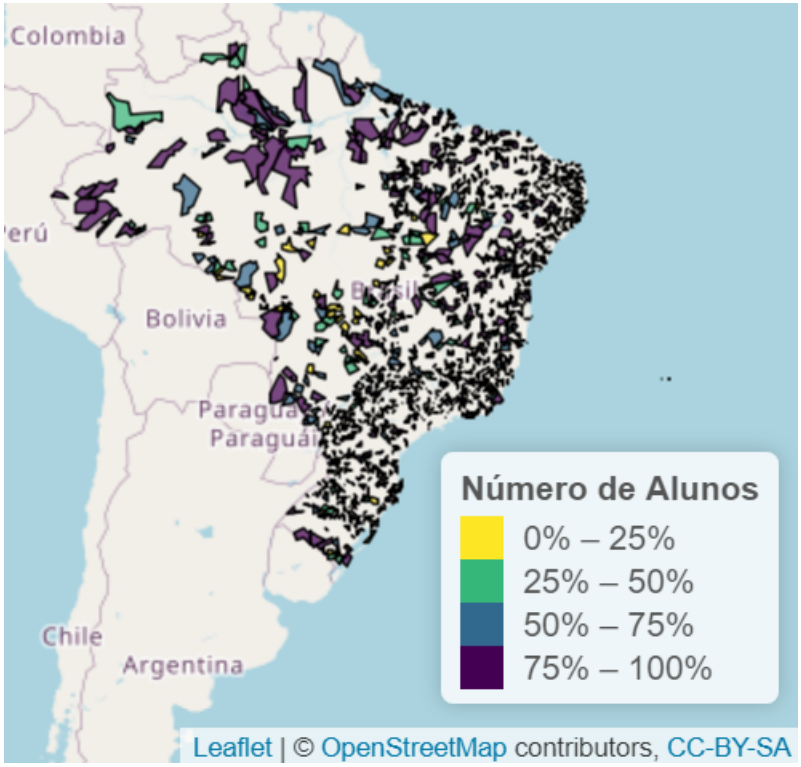


Figura 5.9: Mapa do número de alunos predito pelo modelo GENConv + Linear.

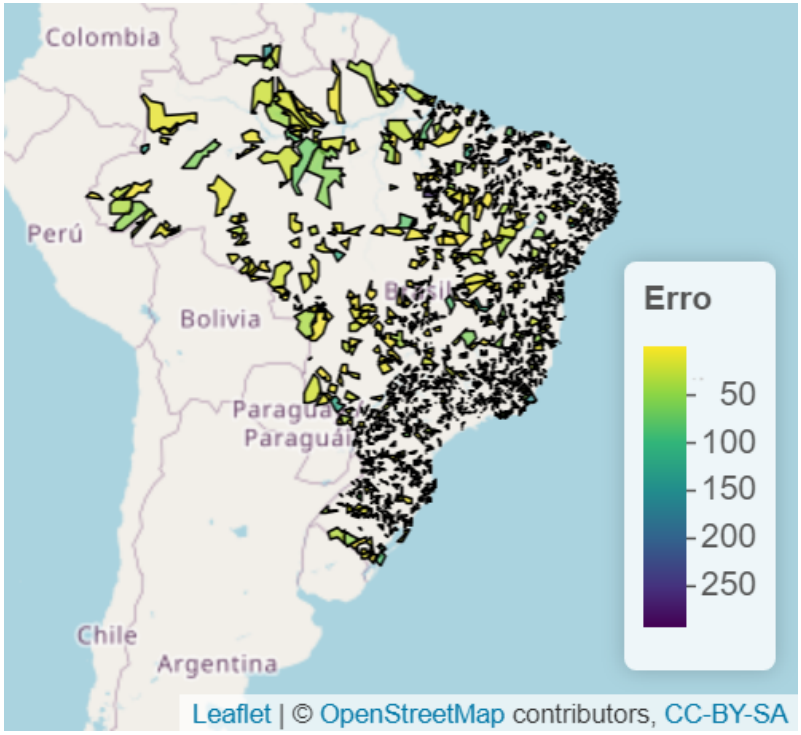


Figura 5.10: Mapa do erro de predição do modelo GENConv + Linear. O limite superior do erro é menor do que no Modelo MLP.

A seguir, são apresentados os gráficos que exploram análises mais detalhadas.

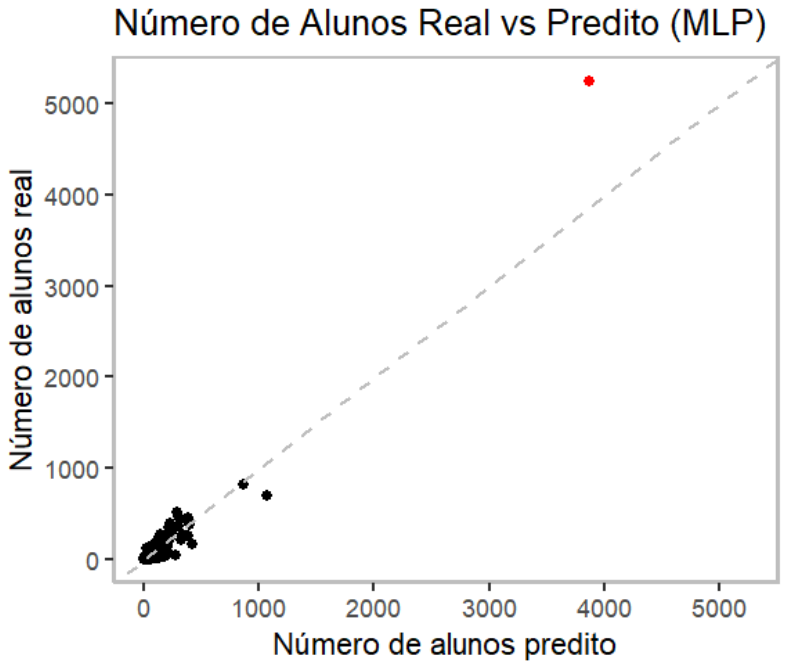


Figura 5.11: **Gráfico de número de alunos verdadeiro vs predito para o modelo MLP:** É possível observar um valor no canto direito superior destacado na cor vermelha que não está bem predito pelo modelo. O valor de MSE está sendo puxado para cima por conta dessa observação. O município que representa essa observação *outlier* é Rio de Janeiro (RJ) com 5247 alunos. O MSE com essa observação é 1945.48, sem ela é 731.6637.

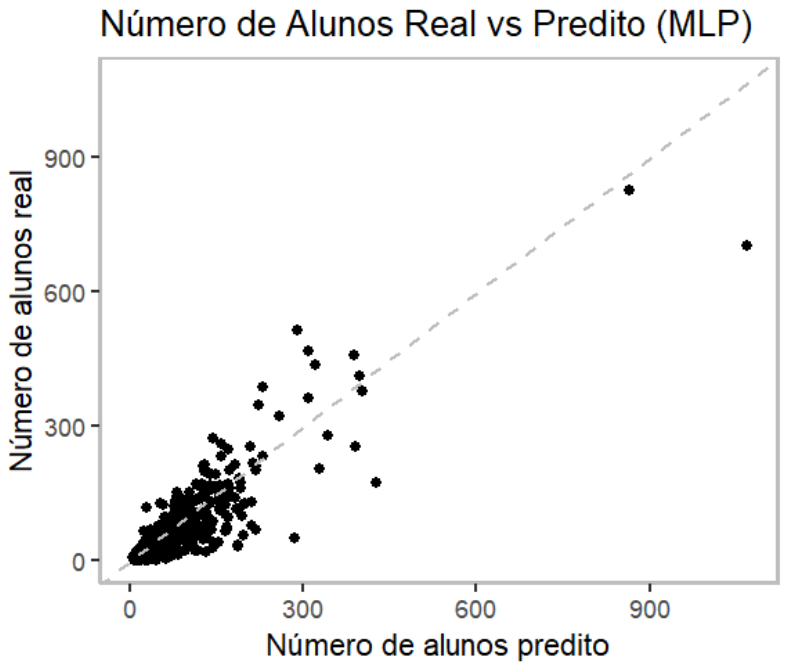


Figura 5.12: **Gráfico de número de alunos verdadeiro vs predito para o modelo MLP sem a observação Rio de Janeiro (RJ)**: A concentração de pontos se encontra em valores menores, no intervalo de 0 a 150, aproximadamente. Os valores parecem estar bem preditos, com algumas pequenas exceções.

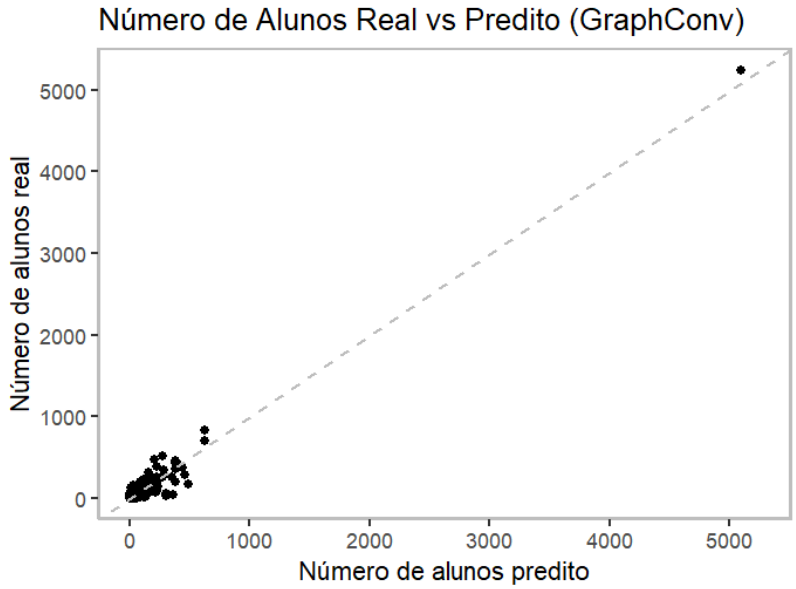


Figura 5.13: **Gráfico de número de alunos verdadeiro vs predito para o modelo GraphConv + Linear**: Para esse modelo, há uma capacidade superior de predizer o número de alunos para o Rio de Janeiro (RJ) em relação ao modelo MLP.

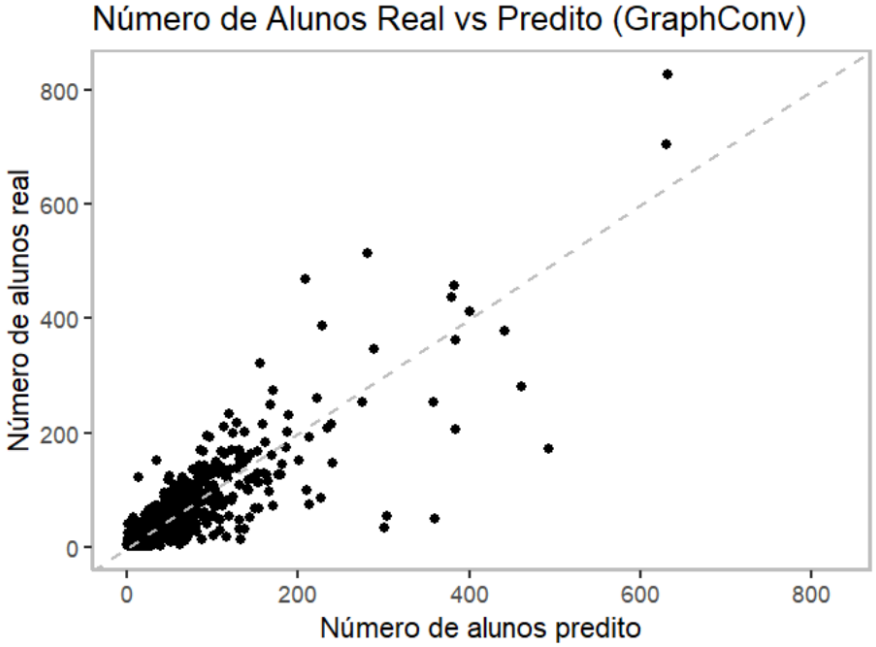


Figura 5.14: **Gráfico de número de alunos verdadeiro vs predito para o modelo GraphConv + Linear sem a observação Rio de Janeiro (RJ)**: A concentração de pontos se encontra em valores menores, no intervalo de 0 a 150, aproximadamente. Os valores parecem estar bem preditos, com algumas pequenas exceções.

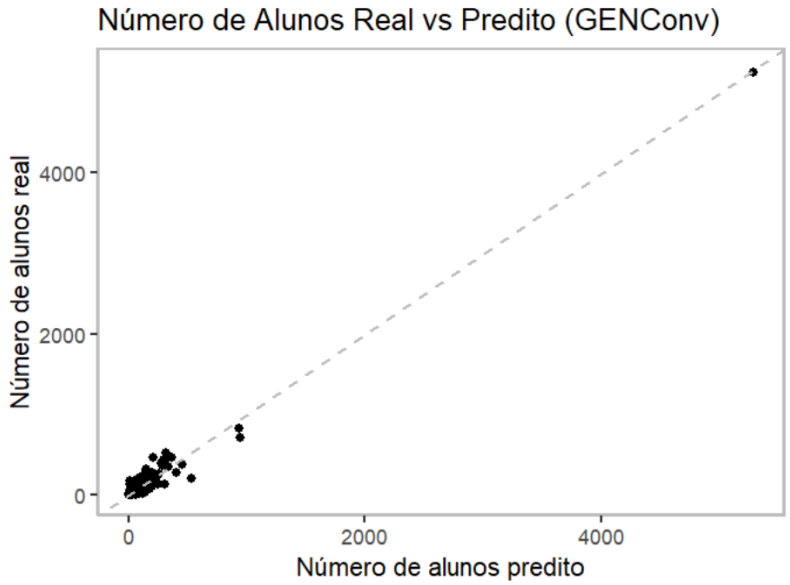


Figura 5.15: **Gráfico de número de alunos verdadeiro vs predito para o modelo GENConv**: Para esse modelo, há uma capacidade superior de predizer o número de alunos para o Rio de Janeiro (RJ) em relação aos demais modelos.

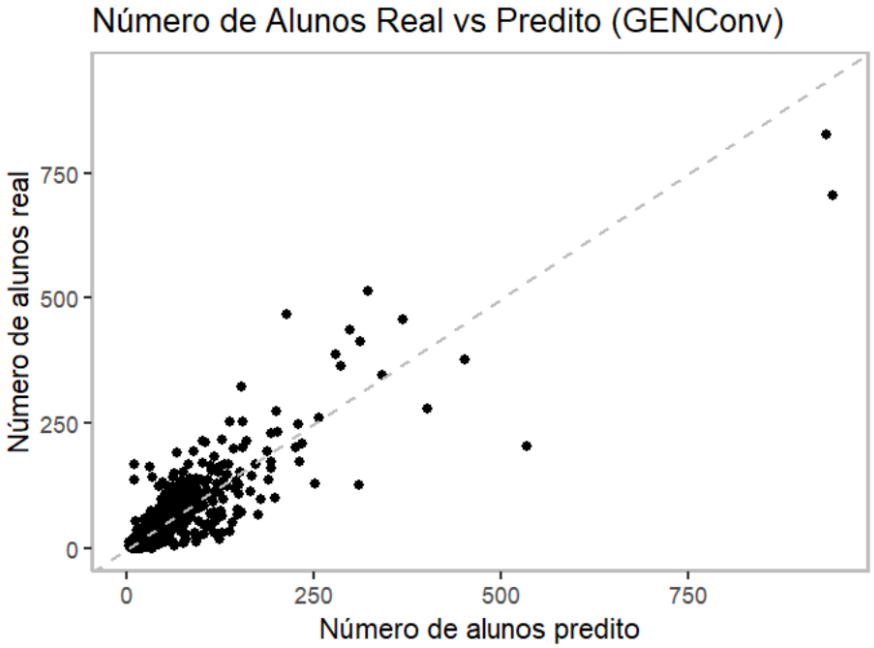


Figura 5.16: **Gráfico de número de alunos verdadeiro vs predito para o modelo GENConv sem a observação Rio de Janeiro (RJ)**: A concentração de pontos se encontra em valores menores, no intervalo de 0 a 150, aproximadamente. Os valores parecem estar bem preditos, com algumas pequenas exceções.

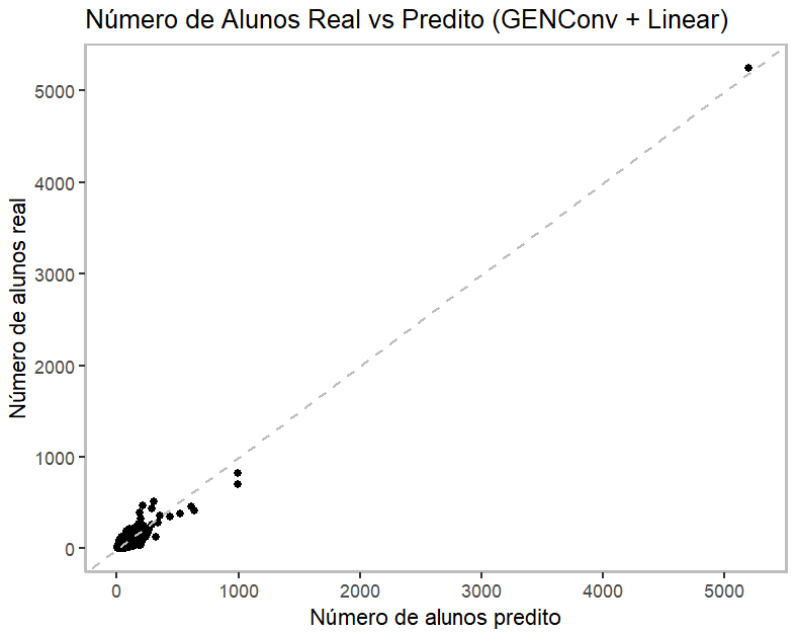


Figura 5.17: **Gráfico de número de alunos verdadeiro vs predito para o modelo GENConv + Linear**: Para esse modelo, há uma capacidade superior de predizer o número de alunos para o Rio de Janeiro (RJ) em relação aos modelos MLP e GraphConv + Linear.

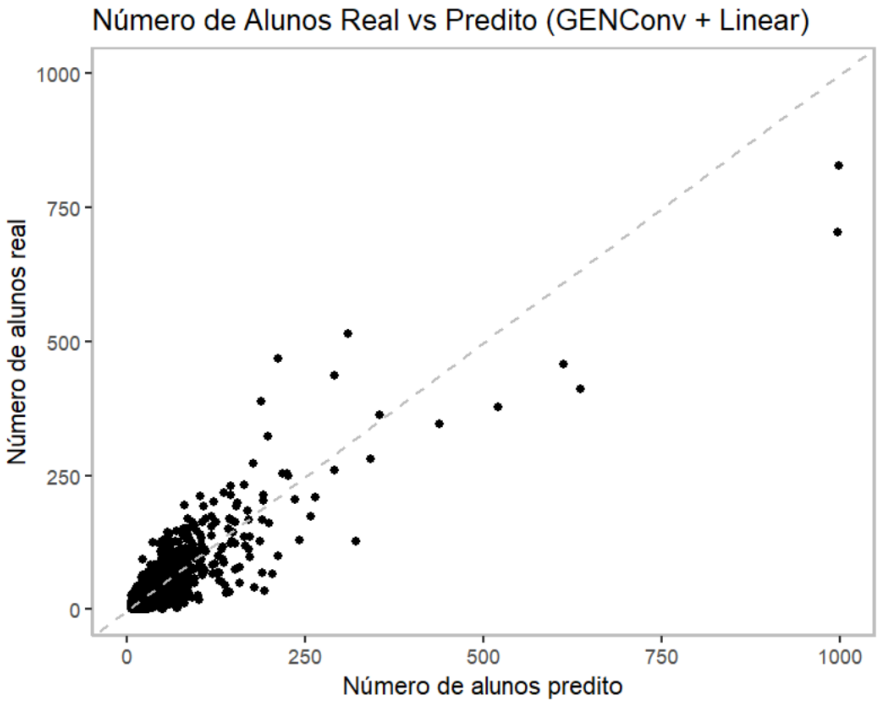


Figura 5.18: **Gráfico de número de alunos verdadeiro vs predito para o modelo GENConv + Linear sem a observação Rio de Janeiro (RJ)**: A concentração de pontos se encontra em valores menores, no intervalo de 0 a 150, aproximadamente. Os valores parecem estar bem preditos, com algumas pequenas exceções.

6 Conclusão

Em conclusão, a pesquisa abordou a aplicação de diversas arquiteturas de GNN, como GCN, GraphConv, GAT, TransformerConv e GenConv, comparando-as com o MLP, que é um método que considera as observações de forma independente. Adicionalmente, foi conduzido um experimento com 30 diferentes sementes para avaliar a robustez e eficiência dos métodos GNN no contexto espacial, com o objetivo de determinar se os resultados superiores em relação ao método que considera as observações independentes não foram simplesmente uma eventualidade.

Os resultados revelaram que, embora a presença de uma observação atípica tenha impacto no MSE para o modelo MLP, os modelos GENConv, GENConv + Linear e GraphConv + Linear demonstraram uma capacidade superior de lidar com esses *outliers*, apresentando um desempenho melhor.

Alguns modelos são particularmente sensíveis à inicialização dos pesos, influenciada pela semente. Como resultado, configurações idênticas com diferentes sementes resultaram em desempenhos distintos. Essa variabilidade é evidenciada pelo alto desvio padrão do MSE durante as simulações nos modelos GraphConv, GAT, TransformerConv, GENConv e GAT + Linear.

Inesperadamente, as camadas clássicas de GNN, representadas por GCN e GAT, revelaram um desempenho abaixo do esperado, apontando para a necessidade de explorar outras abordagens. Além disso, observou-se uma melhoria geral no desempenho de todos os modelos GNN quando camadas lineares foram adicionadas, com exceção do modelo GENConv.

Como perspectivas futuras, há diversas possibilidades a serem exploradas. Entre elas, destaca-se a investigação de previsões espaço-temporais utilizando modelos GNN.

Outra perspectiva interessante seria a aplicação de modelos GNN para tarefas de *clustering*, possibilitando a identificação de padrões e agrupamentos nos dados. Uma investigação mais detalhada das cabeças de atenção em modelos GNN que incorporam esse componente, quando aplicada ao contexto espacial, proporciona uma oportunidade para avaliar o nível de dependência entre distintas regiões geográficas.

Além disso, a exploração de combinações variadas de camadas GNN, como a fusão de GENConv com TransformerConv, pode proporcionar modelos mais robustos.

Referências Bibliográficas

Apicella, A., F. Donnarumma, F. Isgrò, and R. Prevete (2021). A survey on modern trainable activation functions. *Neural Networks 138*, 14–32.

Aydin, O., M. V. Janikas, R. Assunçao, and T.-H. Lee (2018). Skater-con: Unsupervised regionalization via stochastic tree partitioning within a consensus framework using random spanning trees. In *Proceedings of the 2nd ACM SIGSPATIAL international workshop on AI for geographic knowledge discovery*, pp. 33–42.

Bailey, T. C. (2001). Spatial statistical methods in health. *Cadernos de Saúde Pública 17*(5), 1083–1098.

Bergstra, J., D. Yamins, D. D. Cox, et al. (2013). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, Volume 13, pp. 20. Citeseer.

Bessadok, A., M. A. Mahjoub, and I. Rekik (2022). Graph neural networks in network neuroscience. *IEEE Transactions on Pattern Analysis and Machine Intelligence 45*(5), 5833–5848.

Bishop, C. (2006). Pattern recognition and machine learning. *Springer google schola 2*, 531–537.

Blangiardo, M., M. Cameletti, G. Baio, and H. Rue (2013). Spatial and spatio-temporal models with r-inla. *Spatial and spatio-temporal epidemiology 7*, 39–55.

Brownlee, J. (2020). Train-test split for evaluating machine learning algorithms. *Machine learning mastery 23*(7).

Bui, K.-H. N., J. Cho, and H. Yi (2022). Spatial-temporal graph neural network for traffic forecasting: An overview and open research issues. *Applied Intelligence 52*(3), 2763–2774.

Chen, D., Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun (2020). Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, Volume 34, pp. 3438–3445.

Chen, T., T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, et al. (2015). Xgboost: extreme gradient boosting. *R package version 0.4-2 1*(4), 1–4.

Cressie, N. (2015). *Statistics for Spatial Data*. John Wiley and Sons.

Donnat, C. and S. Holmes (2018). Tracking network dynamics: A survey of distances and similarity metrics. *arXiv preprint arXiv:1801.07351*.

Eksombatchai, C., P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec (2018). Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 world wide web conference*, pp. 1775–1784.

Gardner, M. W. and S. Dorling (1998). Artificial neural networks (the multilayer perceptron)a review of applications in the atmospheric sciences. *Atmospheric environment 32*(14-15), 2627–2636.

Glorot, X., A. Bordes, and Y. Bengio (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323. JMLR Workshop and Conference Proceedings.

Gökhan, A., C. O. Güzeller, and M. T. Eser (2019). The effect of the normalization method used in different sample sizes on the success of artificial neural network model. *International Journal of Assessment Tools in Education 6*(2), 170–192.

Gupta, A. (2021). A comprehensive guide on optimizers in deep learning. *Analytics Vidhya*.

Hamilton, W. L. (2020). *Graph representation learning*. Morgan & Claypool Publishers.

Hastie, T., R. Tibshirani, and J. Friedman (2009). Random forests. *The elements of statistical learning: Data mining, inference, and prediction*, 587–604.

He, K., X. Zhang, S. Ren, and J. Sun (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

Hu, Z., Y. Dong, K. Wang, K.-W. Chang, and Y. Sun (2020). Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1857–1867.

Jabbar, H. and R. Z. Khan (2015). Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices 70*(10.3850), 978–981.

Jain, S. (2018). An overview of regularization techniques in deep learning (with python code). *Analytics Vidhya 19*.

Jiang, W. and J. Luo (2022). Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications 207*, 117921.

Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems 30*.

Kipf, T. N. and M. Welling (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems 25*.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324.

Leskovec, J. (2019). Stanford cs224w: Machine learning with graphs. *Traditional Methods for Machine Learning in Graphs*.

Li, G., C. Xiong, A. Thabet, and B. Ghanem (2020). Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*.

Liang, F., C. Qian, W. Yu, D. Griffith, and N. Golmie (2022). Survey of graph neural networks and applications. *Wireless Communications and Mobile Computing 2022*.

Lu, H. and S. Uddin (2023). Disease prediction using graph machine learning based on electronic health data: A review of approaches and trends. In *Healthcare*, Volume 11, pp. 1031. MDPI.

Lu, L., Y. Shin, Y. Su, and G. E. Karniadakis (2019). Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*.

Ma, Z., H. Zhang, and J. Liu (2022). Preciplstm: A meteorological spatiotemporal lstm for precipitation nowcasting. *IEEE Transactions on Geoscience and Remote Sensing 60*, 1–8.

Maas, A. L., A. Y. Hannun, A. Y. Ng, et al. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Volume 30, pp. 3. Atlanta, GA.

McCulloch, W. S. and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*, 115–133.

Monti, F., F. Frasca, D. Eynard, D. Mannion, and M. Bronstein. Fake news detection on social media using geometric deep learning. arxiv 2019. *arXiv preprint arXiv:1902.06673*.

Morris, C., M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Volume 33, pp. 4602–4609.

Nguyen, V. B., T. S. Hy, L. Tran-Thanh, and N. Nghiem (2023). Predicting covid-19 pandemic by spatio-temporal graph neural networks: A new zealand's study. *arXiv preprint arXiv:2305.07731*.

Pan, Z., Y. Liang, W. Wang, Y. Yu, Y. Zheng, and J. Zhang (2019). Urban traffic prediction from spatio-temporal data using deep meta learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '19)*, pp. 1720–1730.

Patrikar, S. (2019). Batch, mini batch & stochastic gradient descent-towards data science. *URL: https://towardsdatascience. com/batch-mini-batch-stochastic-gradientdescent-7a62ecba642a 10*.

Qian, J., V. Saligrama, and Y. Chen (2014). Anomalous cluster detection. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3854–3858. IEEE.

Robbins, H. and S. Monro (1951). A stochastic approximation method. *The annals of mathematical statistics*, 400–407.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review 65*(6), 386.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors. *nature 323*(6088), 533–536.

Sanchez-Gonzalez, A., J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia (2020). Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pp. 8459–8468. PMLR.

Sanchez-Lengeling, B., E. Reif, A. Pearce, and A. B. Wiltschko (2021). A gentle introduction to graph neural networks. *Distill*. https://distill.pub/2021/gnn-intro.

Scarselli, F., M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini (2008). The graph neural network model. *IEEE transactions on neural networks 20*(1), 61–80.

Shi, Y., Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun (2020). Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*.

Stokes, J. M., K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann, et al. (2020). A deep learning approach to antibiotic discovery. *Cell 180*(4), 688–702.

Velickovic, P., G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, et al. (2017). Graph attention networks. *stat 1050*(20), 10–48550.

Waikhom, L. and R. Patgiri (2023). A survey of graph neural networks in various learning paradigms: methods, applications, and challenges. *Artificial Intelligence Review 56*(7), 6295–6364.

Waller, L. A. and C. A. Gotway (2004). *Applied spatial statistics for public health data*. John Wiley & Sons.

Wang, Q., Y. Ma, K. Zhao, and Y. Tian (2020). A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 1–26.

West, D. B. et al. (2001). *Introduction to graph theory*, Volume 2. Prentice hall Upper Saddle River.

Wilson, R. J. (1979). *Introduction to graph theory*. Pearson Education India.

Yang, C., R. Wang, S. Yao, S. Liu, and T. Abdelzaher (2020). Revisiting over-smoothing in deep gcns. *arXiv preprint arXiv:2003.13663*.

Ying, X. (2019). An overview of overfitting and its solutions. In *Journal of physics: Conference series*, Volume 1168, pp. 022022. IOP Publishing.

Ying, Z., D. Bourgeois, J. You, M. Zitnik, and J. Leskovec (2019). Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems 32*.

Zuur, A. F., E. N. Ieno, and C. S. Elphick (2010). A protocol for data exploration to avoid common statistical problems. *Methods in ecology and evolution 1*(1), 3–14.