UNIVERSIDADE FEDERAL DE SANTA CATARINA

Utilização de Redes Neurais em Grafo para prever surtos de doenças: uma abordagem testando propagação de informação em uma rede convolucional

Rian Carlos Vieira Brüggemann

Florianópolis
2023

Rian Carlos Vieira Brüggemann

Utilização de Graph Neural Networks para prever surtos de doenças: uma abordagem testando propagação de informação em uma rede

Trabalho de conclusão de curso submetido como parte dos requisitos para obtenção do título de Bacharel, do curso de Sistemas de Informação da Universidade Federal de Santa Catarina.

Orientador: Rafael de Santiago

Florianópolis, Junho de 2023

Resumo

Este trabalho consiste em explorar computacionalmente os dados provenientes da pandemia do Coronavirus (COVID-19) registrados no Sistema de Saúde Unificado (SUS) do estado de Santa Catarina. Os dados foram extraídos de um CSV e orientados em grafos para estruturação e desenvolvimento do problema. Propôs-se um modelo que permita realizar predições sobre o número de casos confirmados em uma data futura. Uma única rede neural foi desenvolvida com uma arquitetura orientada a grafos, cada mesoregião representa um nodo e as conexões entre os nodos estão limitadas - assim como geograficamente - por suas mesoregiões vizinhas. Buscou-se explorar os mais variados tipos de hiperparâmetros (Taxa de *Dropout*, Taxa de Aprendizado, Número de Canais, Número de Camadas e Pesos) com o uso do framework Optuna, a fim de encontrar um modelo que apresente o menor Erro Absoluto Médio (MAE) para rede, observando valores preditos desconhecidos. Este modelo foi treinado utilizando janelas fixas de tempo (15 e 7 dias), avançando diariamente o que resultou em um modelo desempenha bem para dados já conhecidos e com possível enviesamento dos resultados preditos.

Palavras-chave: Aprendizado de Máquina, Estruturas em Grafos, Rede Convolucional de Grafos, SARS-CoV-2, Engenharia de Dados, Aprendizado Supervisionado, Modelagem Preditiva.

Keywords: Machine Learning, Graph Structures, Graph Convolutional Network, SARS-CoV-2, Data Engineering, Supervised Learning, Predictive Modeling.

Sumário

Lista de abreviaturas e siglas

ML - Machine Learning (Aprendizado de Máquina)

SL - Supervised Learning (Aprendizado Supervisionado)

CNN - Convolutional Neural Networks (Redes Neurais Convolucionais)

GNN - Graph Neural Networks (Redes Neurais em Grafo)

GCN - Graph Convolutional Networks (Redes Convolucionais em Grafo)

CSV - Comma Separated Values (formato de arquivo 'valores separados por vírgulas)

MAE - Mean Absolute Error (Erro Médio Absoluto)

R^2 - Coefficient of Determination (Coeficiente de Determinação)

MAPE - Mean Absolute Percentage Error (Erro Percentual Médio Absoluto)

MSE - Mean Squared Error (Erro Quadrático Médio)

RMSE - Root Mean Squared Error (Raiz Quadrada do Erro Quadrático Médio)

MDAPE - Mean Absolute Percentage Error (Mediana do Erro Percentual Absoluto)

1 Introdução

Nas últimas décadas, o mundo tem vivenciado um aumento significativo na ocorrência de surtos de doenças epidêmicas. Este fenômeno tem sido associado ao crescimento populacional mundial, às mudanças nos hábitos alimentares e à expansão urbana que levou à maior interação entre humanos e animais selvagens. Esta interação crescente facilita a transmissão de patógenos zoonóticos, que são capazes de saltar das populações animais para os humanos. Olhando para trás na história, podemos observar vários exemplos de pandemias que moldaram a sociedade de maneiras significativas, como a Gripe Espanhola em 1918, a epidemia de HIV/AIDS na década de 1980 e a pandemia de H1N1 em 2009. Na história recente com a pandemia da COVID-19, enfrentamos desafios semelhantes, mas em uma escala global sem precedentes. É fundamental compreendermos esses eventos passados para melhorar nossas estratégias de prevenção e controle de futuras epidemias e pandemias. (SANAR, 2020).

Diante de uma sociedade globalizada, o advento de novas tecnologias de comunicação tornou capaz de traçar e identificar cada vez mais informações de seus usuários (WERTHEIN, 2000). *Terabytes* de dados transitam e compõem a nossa estrutura de informações relativas aos seres inseridos na sociedade da informação. Segundo *(PASTOR-SATORRAS et al., 2015)* nos últimos anos, a comunidade de pesquisa acumulou diversas evidências para o surgimento de padrões de conectividade complexos e heterogêneo em uma ampla gama de sistemas biológicos e sociotécnicos. As complexas propriedades provenientes das redes do mundo real têm um impacto profundo no comportamento dos fenômenos de equílibrio e não-equilíbrio ocorrendo em vários sistemas, e o estudo da disseminação da epidemia é central para nossa compreensão do desdobramento de processos dinâmicos em redes complexas *(PASTOR-SATORRAS et al., 2015)*.

A pandemia do Coronavírus (COVID-19) alterou significativamente os padrões de mobilidade humana, necessitando de modelos epidemiológicos que possam capturar os efeitos dessas mudanças na disseminação da síndrome respiratória aguda grave Coronavírus 2 (SARS-CoV-2) *(BUCKEE et al., 2020)*. Uma gama de diferentes profissionais, como físicos, epidemiologistas, cientistas sociais e da computação que compartilham o interesse no estudo da propagação de epidemias, dependem de modelos muito semelhantes para a descrição da difusão de patógenos, conhecimento e inovação. Neste sentido, os resultados apresentados em modelos de doenças infecciosas estão baseados de maneira geral em processos de contágios sociais (PASTOR-SATORRAS et al., 2015).

Sob o prisma da forma que os algoritmos *Machine Learning* funcionam, nesse estudo propomos uma abordagem que visa a criação de um modelo preditivo para futuros casos

confirmados da Covid-19 no estado de Santa Catarina. Utilizando séries temporais, temos a informação de casos confirmados por região do estado datadas entre 29 de fevereiro de 2020 até 29 de maio de 2023, que são convertidas em forma de grafo onde cada nó representa uma região e cada aresta representa a interação entre regiões vizinhas - as mesmas que dispõe geograficamente. Além disso, cada nó do grafo é rotulado com a população absoluta da região e o número de casos confirmados de COVID-19 nos últimos 15 dias na região . O modelo então processa esta representação de grafo para fornecer a previsão do número de casos confirmados de COVID-19 do 16^{o} dia - dia seguinte aos já processados pelo modelo.

O principal objetivo desta pesquisa é modelar e prever a disseminação da COVID-19 por meio da implementação de uma Rede Neural Convolucional (GCN). Através de algoritmos de aprendizado supervisionado, desenvolvemos um script Python que implementa uma GCN - tipo de rede que compreende a modelagem de fenômenos complexos - realizando a abstração de como a propagação de doenças ocorre em uma sociedade globalizada.

O script também utiliza uma biblioteca de otimização para ajustar os parâmetros da rede. Além disso, o desempenho do modelo é avaliado por meio da validação cruzada de séries temporais - abordagem que preserva a ordem cronológica dos fatos - em diferentes partes do conjunto de dados. Para fins métricos, utilizamos erro médio absoluto, coeficiente de determinação R^2, erro percentual médio absoluto, erro médio quadrado pois tratar-se de um problema de regressão.

O aprendizado de uma variante eficiente de redes neurais convolucionais utilizando dados em estrutura de grafos apresenta uma significativa margem de relacionamentos entre as amostras e experimentos (KIPF; WELLING, 2016).

1.1 Objetivos

O objeto de estudo deste projeto é desenvolver um modelo que indique se o montante de casos - usuários que dão entrada no sistema de saúde de Santa Catarina com sintomas da *SARS-CoV-2* - estão em aumento ou declínio, classificados pelas mesoregiões do estado, e com este modelo balanceado, utiliza-lo para alimentar uma rede neural.

Os objetivos específicos são:

- Realizar a preparação, limpeza, modelagem e balanceamento dos dados;
- Segmentar para treinamento, validação e testes e realizar pré-processamento;
- Experimento com as características extraídas - implementar recurso que indica número total de pacientes em determinado período;
- Desenvolver a rede neural;

- Analisar os resultados obtidos nos experimentos;
- Divulgar os resultados.

1.2 Método de Pesquisa

Este trabalho seguiu as seguintes etapas:

1. Realização de pesquisas a partir de artigos e livros da área de redes neurais e aprendizado de máquina, de maneira a formar uma base de referências e exemplos, para então ser possível abordar assuntos mais específicos.

2. Estudo com o objetivo de definir as entradas nas redes, e também como obtê-las a partir da base de dados.

3. Análise de ferramentas e bibliotecas para a implementação das redes neurais e seus distintos modelos, assim como para o tratamento, processamento e normalização dos dados de entrada do sistema.

4. Obtenção dos dados de entrada.

5. Implementação dos modelos e o treinamento das redes neurais.

6. Realização de comparações e considerações relevantes.

7. Documentação do trabalho desenvolvido.

2 Fundamentação Teórica

2.1 Modelos Matemáticos de Transmissão

A pandemia da COVID-19 trouxe à tona diversos problemas e temáticas das quais cotidianamente, não eram de conhecimento da maioria da população, como por exemplo, modelos matemáticos aplicados à epidemiologia (FRANCO; DUTRA, 2020). Estes têm como função trazer representações matemáticas simplificadas de algum fenômeno do mundo real, que usualmente pode ser descrito por meio de equações matemáticas, possibilitando compreender o comportamento das variáveis envolvidas no fenômeno aplicado ao estudo. Os mais diferentes tipos de modelos matemáticos podem ser aplicados ao estudo de doenças infecciosas, sendo a escolha do tipo destes modelos a que melhor descreva este sistema e a forma como ocorrem suas modificações, desta forma, sendo possível determinar parâmetros e projetar a evolução de epidemias.

Nos modelos baseados em compartimentos proposto por Franco e Dutra (2020), uma amostragem e/ou população são divididos em compartimentos (ou classes) a qual representa seu estado atual no desenvolvimento da doença, como por exemplo, Suscetíveis *(S)* - indivíduos que estão suscetíveis a contrair a doença; Infectados *(I)*, indivíduos que contraíram a doença e podem transmiti-la aos indivíduos suscetíveis por transmissão direta; e Removidos *(R)*, indivíduos que foram infectados, mas não são mais portadores da doença, por motivo de isolamento, cura (adquirindo ou não imunidade), ou morte. Desta forma, uma amostragem pode ser representada por um total N, sendo $\mathbf{N} = \mathbf{S} + \mathbf{I} + \mathbf{R}$.

2.1.1 Modelo SIS (suscetível-infectado-suscetível)

Descreve o comportamento patogênico de indivíduos suscetíveis que adquirem a doença infectam-se e, após recuperação, não adquirem imunidade, desta forma, tornando-se suscetíveis a uma nova reinfecção (FRANCO; DUTRA, 2020). A base modelo dispõe de dois parâmetros que compõem a dinâmica da doença, *alfa* e *beta*, sendo *alfa* a taxa de transmissão da doença e *beta* a taxa de recuperação da doença. Tendo em vista que o patógeno ocorre entre indivíduos suscetíveis e infectados, então a variação de indivíduos suscetíveis em relação ao tempo pode ser modelada por *alfa-SI*. Considerando que a variação de infectados na amostragem sobre o tempo é proporcional ao número de infectados, desta forma, a classe de suscetíveis será modelado por *beta-I*. Diante as interações entre indivíduos suscetíveis, em contato com infectados, passam a ser indivíduos infectados, da mesma forma que indivíduos infectados, ao se recuperarem e não desenvolvendo imunidade, retornam ao compartimento de suscetíveis. A dinâmica de uma doença com essas características pode

ser descrita pelo seguinte sistema de equações diferenciais (FRANCO; DUTRA, 2020).

Figura 1 – Equações diferenciais modelo SIS, (FRANCO; DUTRA, 2020)

$$(1)\quad \begin{cases} \frac{dS}{dt} = -\alpha SI + \beta I \\ \frac{dI}{dt} = \alpha SI - \beta I \end{cases}, \qquad \text{onde } \alpha, \beta > 0 \text{ e } N = S(t) + I(t).$$

Segundo Franco e Dutra (2020), "*seja R0 = alfa-S Beta a taxa de reprodução básica que significa o número médio de infecções causadas por um indivíduo doente, onde 1 beta é o tempo médio no qual um indivíduo permanece infectado e alfa-S é a taxa de propagação da doença provocada pela introdução de um indivíduo infectado numa população de suscetíveis. Observe que: Se R0 > 1 e I != 0 então dI dt > 0 e dS dt < 0. O que significa que a epidemia alastra-se pela população. Se R0 < 1 e I != 0 então dI dt < 0 e dS dt > 0. O que significa que o contágio diminui. Se I = 0 então N = S e todas as pessoas são saudáveis, ou seja, não existe infecção.*"

2.1.2 Modelo SIR (suscetível-infectado-removido)

Neste modelo há indivíduos suscetíveis que adquirem a doença, tornando-se infectados, e em sequência removidos - podendo ser pela situação de desenvolverem uma imunidade permanente ou por morte (FRANCO; DUTRA, 2020).

Segundo Franco e Dutra (2020), "*considerando um período de incubação relativamente pequeno e a população constante (contando também os mortos), tem-se: N = S(t) + I(t) + R(t). Levando-se em conta que a variação da população recuperada é proporcional à população infectada, o sistema de equações diferenciais que descreve a dinâmica desta epidemia é dado por:*"

Figura 2 – Equações diferenciais modelo SIR, (FRANCO; DUTRA, 2020)

$$(2)\quad \begin{cases} \frac{dS}{dt} = -\alpha SI \\ \frac{dI}{dt} = \alpha SI - \beta I \\ \frac{dR}{dt} = \beta I \end{cases} \qquad \text{Condições iniciais: } R(0) = 0,\ I(0) = I_0,\ S(0) = S_0 = N - I_0$$

2.1.3 Modelo SIRS (suscetível-infectado-removido-suscetível)

A especificidade deste modelo está relacionada a taxa de perda de imunidade que um indivíduo pode apresentar, representada por delta. Este descreve a situação de indivíduos que após infectados pelo patógeno, recuperam-se e em determinado período de tempo, perdem a imunidade, voltando a ser suscetíveis a contrair a doença (FRANCO; DUTRA, 2020).

O sistema de equações diferenciais (3) representa a dinâmica deste modelo, onde Franco e Dutra (2020) descrevem: *"onde, alfa, beta, delta > 0, alfa-I é a taxa de infecção, beta denota a proporção de pessoas que deixaram a classe dos infectados para a classe de recuperados. O número total da população é dado por N = S + I + R, que é um valor constante."*

Figura 3 – Equações diferenciais modelo SIRS, (FRANCO; DUTRA, 2020)

$$(3)\quad \begin{cases} \frac{dS}{dt} = -\alpha SI + \delta R \\ \frac{dI}{dt} = \alpha SI - \beta I \\ \frac{dR}{dt} = \beta I - \delta R \end{cases}$$

Condições iniciais:

$$R(0) = 0,$$
$$I(0) = I_0,$$
$$S(0) = S_0 = N - I_0$$

2.2 Machine Learning (Aprendizado de Máquina) - ML

Machine Learning (ML) é uma área de estudos na computação onde algoritmos geralmente são empregados a ensinar máquinas a habilidade de aprender e reconhecer padrões a partir de experiências de processamento de dados, como definido por Sarker (2021), sendo também uma de suas principais características a capacidade de generalização e maneira como lida com ruídos (ALLAMANIS et al., 2018). ML possui diversos tipos de algoritmos que se adequam para diferentes tipos de situações, desta forma, não existindo um tipo de algoritmo com capacidade de resolver os mais variados tipos de problemas (MAHESH, 2018).

Os algortimos de ML são divididos principalmente em quatro categorias: Supervised Learning (SL) (Aprendizado Supervisionado), Unsupervised Learning (Aprendizado Não-supervisionado), Semi-supervised Learning (Aprendizado Semi-supervisionado) Reinforcement Learning (Aprendizado por Reforço) (SARKER, 2021).

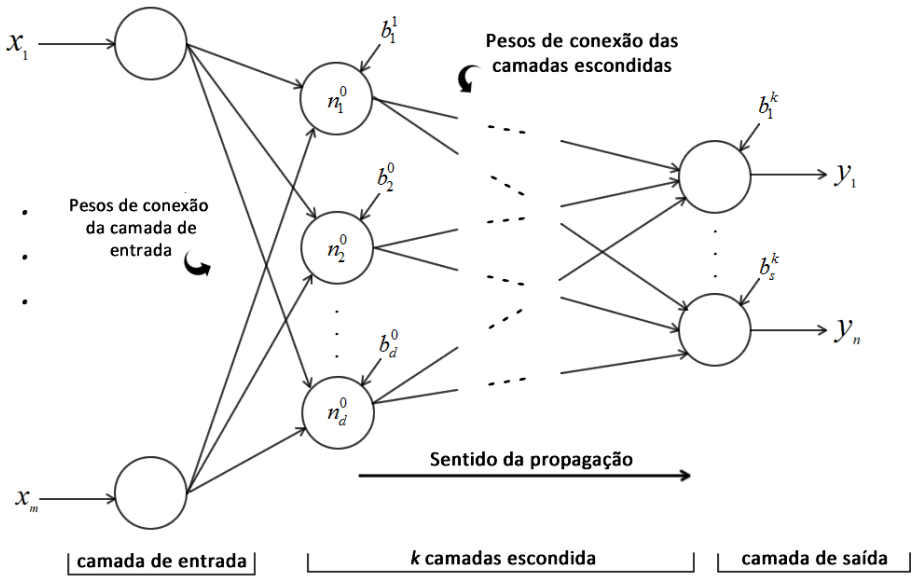
SL é uma das principais categorias de algoritmos de ML. Neste tipo de aprendizado,

os algoritmos são treinados utilizando dados de entrada rotulados, ou seja, cada exemplo de treinamento inclui uma entrada e um resultado correspondente, conhecido como o rótulo. O algoritmo aprende um modelo que mapeia as entradas para os resultados. Uma vez treinado, esse modelo pode ser utilizado para prever a saída de novos dados de entrada que não foram vistos durante o treinamento (MAHESH, 2018). Exemplos de algoritmos de aprendizado supervisionado incluem Regressão Linear, Árvores de Decisão e Máquinas de Vetores de Suporte (SVM). Estes algoritmos são empregados em uma variedade de aplicações, desde a previsão de preços de casas, classificação de imagens, até a detecção de fraudes em transações de cartão de crédito.

2.3 Convolutional Neural Networks (Redes Neurais Convolucionais) - CNN

Antes de abordar Redes Neurais Convolucionais (CNNs), precisamos conhecer as Redes Neurais Artificiais (ANNs). Estas são uma categoria de modelos de aprendizado de máquina inspirados na estrutura e funcionamento do cérebro humano. Elas simulam a maneira como os neurônios humanos interagem e processam informações, adaptando-se e aprendendo a partir dos dados de entrada. As ANNs são compostas por uma série de nós interconectados, chamados de neurônios artificiais, que simulam o comportamento dos neurônios biológicos. Cada neurônio artificial recebe uma série de entradas, as processa e emite uma única saída. O processo de aprendizado em uma ANN é realizado ajustando-se os pesos dessas entradas em cada neurônio artificial durante o treinamento do modelo.

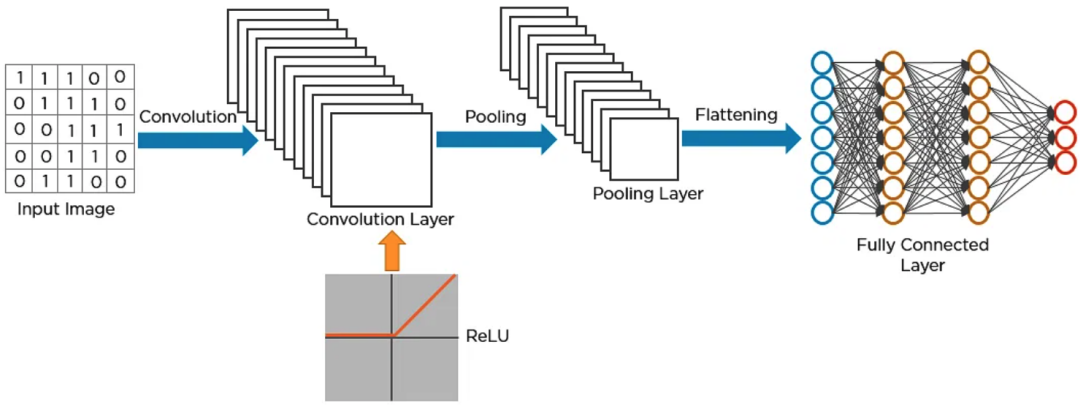
Figura 4 – Exemplo de uma rede neural artificial com k camadas escondidas (PACHECO, 2015)



As CNNs constituem uma classe significativa de redes neurais profundas, que foram desenvolvidas originalmente para processamento de dados de imagem. Inspiradas na organização do córtex visual de animais, onde os neurônios estão arranjados de maneira a cobrir todo o campo visual e responder a estímulos específicos em diferentes regiões, como proposto por Hubel e Wiesel (1962), CNNs também se mostraram eficientes no processamento de séries temporais, que têm uma estrutura sequencial análoga ao campo visual em uma dimensão temporal (WANG; YAN; OATES, 2016).

A diferenciação entre CNNs e de outras redes neurais devido à sua arquitetura única, que é especialmente adequada para processar dados estruturados em grade, como imagens. Elas são compostas por uma série de camadas de convolução seguidas por camadas de agrupamento (ou "pooling"), que gradualmente reduzem a dimensionalidade dos dados enquanto preservam os recursos mais importantes. Este processo mimetiza o mecanismo de visão dos seres vivos, onde diferentes regiões do campo visual são processadas em diferentes graus de detalhe (WANG; YAN; OATES, 2016). Além do processamento de imagens, as CNNs têm demonstrado desempenhar em diferentes tarefas, como processamento de séries temporais.

Figura 5 – Estrutura de uma CNN que com função de ativação ReLU e *Pooling layers* (camadas de agrupamento), seguido pelo *Flattening* (achatamento), podendo esta servir como entrada de uma nova teia convolucional (BISWAL, 2023)



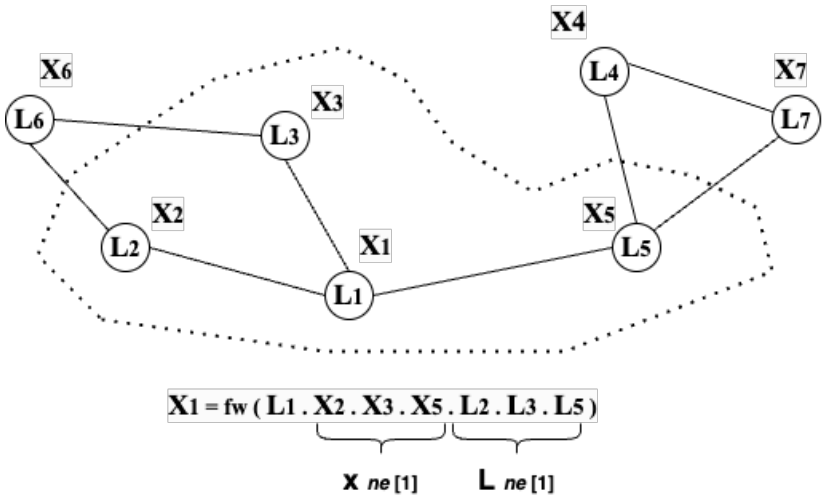
A exemplo, as CNNs têm sido usadas para análise de séries temporais, onde os dados são estruturados de maneira análoga ao campo visual, mas ao longo do tempo. Nesse contexto, as CNNs são capazes de detectar padrões temporais, assim como detectam recursos espaciais em imagens. Em outras palavras, as CNNs são capazes de aprender a importância relativa de diferentes pontos no tempo para a previsão de um resultado futuro, da mesma forma que aprendem a importância de diferentes regiões de uma imagem para a classificação ou localização de objetos.

2.4 Graph Neural Networks (Redes Neurais em Grafo) - GNN

O modelo de redes neurais em grafos é uma proposta que difere de outras redes neurais por utilizar estruturas de dados em forma de grafos, podendo ser desde pequenos nodos conectados por linhas (ou *nodes* e *edges*, respectivamente), como também organizações mais complexas grafo-estruturadas como árvores, grafos acíclicos e cíclicos. A ideia por trás das GNNs consiste que os nodos representam objetos ou conceitos e suas linhas seus respectivos relacionamentos (SCARSELLI et al., 2009). Desta forma, podemos definir que um grafo G é composto por um par (N, E), onde N representa um conjunto de nodos e E um conjunto de linhas. Cada nodo, e sua respectiva posição, pode ter um rótulo que apresenta propriedades do domínio do problema estudado, onde em cada N pode ser anexado um vetor Xn. Para definição de Xn, naturalmente este contém informações de seus N vizinhos (ver Figura 6).

Conforme (GORI; MONFARDINI; SCARSELLI, 2005) elucidam, suponhamos que 'w' seja um conjunto de parâmetros e que 'f(w)' seja uma função de transição parametrizada, a qual expressa a dependência de um nó (N) em relação aos seus nós vizinhos. O estado de 'Xn' é determinado pela solução de um conjunto de equações, onde 'xn' é igual a 'f(w)' aplicada aos parâmetros 'ln', 'xne[n]' e 'lne[n]'. Aqui, 'n' é membro do conjunto 'N'. Nestes parâmetros, 'ln' é o rótulo de 'n', e 'xne[n]' e 'lne[n]' são, respectivamente, os estados e rótulos dos nós vizinhos de 'n'.

Figura 6 – O estado de x1 depende da informação do vizinho



Fonte: Gori, M.; Monfardini, G.; Scarselli. 2005

2.4.1 Graph Convolutional Networks (Redes Convolucionais em Grafo) - GCN

Graph Convolutional Networks (GCN) são uma extensão das redes neurais convolucionais, projetadas para operar diretamente em dados representados como grafos. Na aprendizagem profunda, a convolução é uma operação matemática que mistura informações de diferentes partes de um vetor de entrada. No caso das GCNs, esta convolução é realizada no domínio espectral do grafo, levando em consideração a topologia do grafo e as características dos nós (KIPF; WELLING, 2016), permitindo a captura de dependências espaciais complexas.

Uma característica importante das GCNs é que elas trabalham com uma noção de vizinhança definida pela estrutura do grafo. Ou seja, ao contrário das redes neurais convolucionais que trabalham com uma vizinhança fixa e regular (por exemplo, pixels vizinhos em uma imagem), as GCNs definem a vizinhança de um nó com base na topologia do grafo. Esta flexibilidade permite que as GCNs se adaptem a uma ampla variedade de contextos e estruturas de dados.

Além disso, as GCNs não dependem apenas da estrutura do grafo, mas também utilizam as características dos nós. Durante a convolução, a informação de cada nó e seus vizinhos é combinada de forma ponderada para produzir um novo conjunto de características. Este processo, conhecido como agregação de vizinhança, permite que as GCNs considerem informações locais ao redor de cada nó, resultando em uma representação mais rica e contextualizada do grafo. Isto influencia na sua capacidade de capturar e aprender características e padrões que são influído pelas estruturas de grafos, permitindo análises mais profundas em tarefas de classificação de nós, ligação de predição ou classificação de grafos inteiros. Esta habilidade de aprender com a propagação de informações ao longo de uma estrutura gráfica torna especialmente úteis em tarefas como a previsão da evolução de um fenômeno ou a difusão de informações (KIPF; WELLING, 2016). As GCNs podem lidar com grafos não direcionados, o que amplia seu escopo de aplicabilidade.

No que diz respeito à eficiência computacional, as GCNs são particularmente eficientes para lidar com grafos de grande escala. Como a operação de convolução é realizada em paralelo em todos os nós, as GCNs são altamente paralelizáveis e podem ser facilmente escaladas para processar grafos com milhões de nós.

De forma geral, as GCNs têm se mostrado eficazes para tarefas que requerem a exploração de relações complexas entre os nós de um grafo, como classificação de nós, ligação de predição ou classificação de grafos inteiros (KIPF; WELLING, 2016).

3 Trabalhos Correlatos

Os trabalhos correlatos referenciados nesse projeto foram reunidos através do uso da ferramenta SCOPUS, Arxiv e Google Acadêmico, utilizando os termos *machine learning*, *graph neural networks* e *SARS-CoV-2* como palavras-chave para busca.

3.1 Pesquisas com temática central Machine Learning

3.1.1 Machine Learning: Algorithms, Real-World Applications and Research Directions

Este artigo explora o vasto campo do Machine Learning (Aprendizado de Máquina), abordando algoritmos, aplicações no mundo real e direções futuras de pesquisa. Este trabalho de Sarker (2021) tem foco na explicação de algoritmos chave e técnicas de aprendizado de máquina, o artigo fornece uma visão abrangente sobre as tendências atuais neste campo. O trabalho traz exemplos de aplicações práticas do aprendizado de máquina em diferentes setores, desde saúde e educação até ciência dos dados e engenharia. É destacado principalmente o impacto significativo do aprendizado de máquina na automação de processos, otimização de operações e melhoria da tomada de decisões em vários setores da indústria.

São trazidos fatos que indicam direções promissoras para pesquisas futuras em aprendizado de máquina, identificando áreas-chave que exigirão mais atenção e desenvolvimento, como a explicabilidade dos modelos, questões éticas, a necessidade de datasets de alta qualidade e o equilíbrio entre privacidade e utilidade dos dados. Por fim, os autores avaliam a necessidade de avançar além dos métodos de aprendizado supervisionado e não supervisionado, para incorporar técnicas de aprendizado por reforço, capazes de lidar com ambientes complexos e em constante mudança.

3.2 Pesquisas com temática central Graph Neural Network

3.2.1 Drug Repositioning for SARS-CoV-2 Based on Graph Neural Network

Nos últimos anos houve um aumento de pesquisas que utilizam algoritmos de *Machine Learning* na área de reposicionamento de drogas, uma seção de estudos da farmacologia que tem como objetivo a descoberta de drogas que ofereçam menor risco sobre retorno comparado a outras drogas já utilizadas, ampliando estratégias e inovando a indústria com suas respectivas necessidades. Neste artigo de Liu (2020), foi apresentado

um efetivo modelo baseado em grafos utilizando técnica de *Deep Learning* (Aprendizado Profundo) *deep2CoV*, que infere sistematicamente novos relacionamentos de droga-doença para o *SARS-CoV-2*. O conceito por trás do deep2CoV é fundir diversas informações sobre *SARS-CoV-2* em uma rede droga-droga, rede droga-doença e rede droga-alvo, desta forma, inferindo potenciais drogas para o *SARS-CoV-2*, utilizando redes convolucionais baseadas em grafos. Sumariamente, o artigo identifica potenciais vantagens na utilização de GNNs (*Graph Neural Networks*) no reposicionamento de drogas devido a redução de espaçamento a integração de diferentes estruturas de informação (redes relacionais droga-droga, droga-doença e droga-alvo) sobre o *SARS-CoV-2* (estrutura em grafo), esta que permite uma visão mais geral das possíveis interações de droga com a doença. Essa abordagem fundamenta o poder e flexibilidade das GNNs para tratar de problemas complexos e multidimensionais na área da saúde.

3.2.2 Semi-Supervised Classification with Graph Convolutional Networks

Sob o prisma das redes neurais convolucionais, algoritmo amplamente utilizado em atividades de classificação de dados, a publicação apresenta uma abordagem escalável para aprendizagem de máquina semi-supervisionada baseada em modelos de dados estruturados em grafos. Condicionando um modelo de rede neural *f(X, A)* em uma estrutura de grafos, onde **X** sendo o conjunto de dados e **A** a matriz adjacente, espera-se que em cenários bem treinados - seguindo método de propagação layerwise - matriz **A** tenha informações que não estão presentes no conjunto **X**. No modelo estudado por Kipf e Welling (2016) a implementação foi executada utilizando *TensorFlow* usando multiplicação de matriz esparsa-densa. Como testagem, foi implementado uma rede semi-supervisionada para classificação de documentos em uma rede de citações, uma rede semi-supervisionada para classificar entidades em grafos, avaliação de algumas abordagens de propagações em grafos e análise em tempo real de grafos aleatórios. Estes testes asseguraram que a rede convolucional baseada em grafos é capaz de codificar ambas as estruturas de grafos e nodos de maneira útil e eficiente, inclusive computacionalmente.

De forma complementar, o estudo propõe um modelo de treinamento eficiente que usa a normalização espectral de grafos, permitindo que o modelo generalize bem para grafos invisíveis no treinamento. A normalização espectral, uma técnica da teoria dos grafos, é fundamental para o sucesso do método, pois facilita o aprendizado de padrões relevantes nas características dos nós e na estrutura do grafo. A abordagem do artigo mostra que o aprendizado semi-supervisionado em grafos, quando aplicado corretamente, pode ser uma ferramenta poderosa para classificar nodos com poucos rótulos disponíveis. O artigo sinaliza um caminho promissor para a pesquisa futura em aprendizado semi-supervisionado em grafos, especialmente na exploração de estruturas de dados complexas e heterogêneas.

3.2.3 Graph Attention Networks

Com o objetivo de melhorar o processamento de informações estruturadas como grafos por redes neurais, os autores utilizaram as Graph Attention Networks (GANs) neste trabalho. Estas redes utilizam mecanismos de atenção, inspirados em redes neurais recorrentes, que permitem que o modelo pondere as contribuições dos nós vizinhos em um grafo, ao invés de tratá-los de maneira uniforme (VELIčKOVIć et al., 2017).

A ideia central é permitir que cada nó em um grafo possa focar em diferentes nós em sua vizinhança em diferentes graus quando está sendo atualizado. Isso é feito através de um conjunto de coeficientes de atenção que são aprendidos de forma adaptativa, permitindo que o modelo seja capaz de capturar diferentes tipos de estruturas em um grafo. As GANs mostraram-se eficazes na realização de tarefas de classificação de nós e de grafos, demonstrando uma capacidade superior às abordagens tradicionais em várias métricas.

Os autores também destacaram a eficiência computacional das GANs, argumentando que, como cada nó atende apenas a uma seleção de seus vizinhos, em vez de toda a rede, o modelo é mais escalável e eficiente. Eles validaram esses resultados em uma série de experimentos, incluindo a classificação de documentos em um conjunto de dados de citações e a classificação de partes de um objeto 3D.

3.3 Pesquisas com temática central Modelos Matemáticos Epidemiológicos

3.3.1 Mathematical models of infectious disease transmission

Segundo Grassly e Fraser (2008), representações matemáticas e análises de doenças infecciosas são áreas correlatas que diante os diversos avanços na área de processamento computacional, tem sido fator chave na evolução da medicina ambulatorial e das análises epidemiológicas. Modelos matemáticos de transmissão conectados a processos biológicos de monitoramento e transmissão de dados levantou duas propriedades estatísticas principais de uma epidemia, a taxa de infecção contínua no início da transmissão de um patógeno por distribuição de tempo. Sobre os desafios dos modelos matemáticos em torno das análises patogênicas, destacam-se: evolução do patógeno, devido a sua dinâmica biológica e diversa, desafia os modelos epidêmicos e de inferência; métodos estatísticos, pois novas infecções estão relacionadas a variáveis desconhecidas.

4 Projeto

No decorrer deste capítulo estão expostos detalhes de implementação da GCN, estratégia para agrupamento dos dados, explorações e preparações realizadas anterior aos ensaios, como também descrição sobre a arquitetura da rede e métricas avaliativas.

4.1 Especificação

Nesta seção ocorre o detalhamento dos métodos utilizados nos experimentos desse trabalho, como as configurações de redes neurais e as da montagem dos grafos.

Foi desenvolvido um modelo que possa aprender por exemplos de séries temporais que utiliza como entrada um grafo que possui rotulagem número absoluto de casos confirmados de uma região do estado de Santa Catarina, como também a sua população relativa. Realizou-se divisão dos dados em Treinamento, Validação e Teste para fins de experimentação, aprimoramento da capacidade de generalização do modelo.

A fonte de dados é uma base de dados dos pacientes que deram entrada no Sistema de Saúde Unificado (SUS) do estado de Santa Catarina com quadro de suspeita de COVID-19. Usando SL desenvolvemos um agente que estime o número de casos confirmados do 16º dia, avançando diariamente, assim como avaliamos a acurácia das hipóteses levantadas, se esta prediz corretamente a novos valores fornecidos como entrada registros diferentes dos utilizados para treinamento.

4.2 Dados

Para treinamento e validação das rede neurais desenvolvidas, foram utilizados dados provenientes de cada região - estes já pré-processados, organizados em um dicionário de dados, compondo um outro dicionário aninhado - tendo o primeiro dicicionário a região (mesoregião de Santa Catarina) como chave, e o 2º dicionário tendo como chave uma data YYYY-MM-DD, e o valor desta chave somatório dos casos confirmados nos 15 dias anteriores. Fonte: (CATARINA, 2020).

4.3 Redes Neurais em Grafos

Durante o desenvolvimento foram avaliados modelos de redes neurais utilizando grafos (GCN), variando o número de camadas, número de canais, a taxa de aprendizado, taxa de *dropout*, taxa de peso entre relações, o tamanho dos lotes de entrada e a quantidade

de interações, de forma a encontrar uma ou mais configurações eficientes com os parâmetros e dados utilizados. Os modelos desenvolvidos foram comparados a partir do erro absoluto médio, ou seja, a relação entre os casos confirmados de uma data, com uma data futura não vista. Outras métricas avaliativas vistas no modelo foi o erro percentual médio absoluto, erro quadrático médio, raiz do erro quadrático médio, coeficiente de determinação (R^2) e o erro percentual absoluto mediano. A métrica escolhida para otimização no framework Optuna foi o erro absoluto médio, por ser de mesma natureza e escala em que os dados de treinamento e predição consistem.

4.4 Desenvolvimento

Foi utilizada uma Graph Convolutional Network (GCN) para prever o número de casos confirmados de COVID-19 por região e data. O código utiliza a biblioteca PyTorch e a PyTorch Geometric para criar e treinar a rede neural. Para otimizar os parâmetros do modelo, o código utiliza a biblioteca Optuna. Além disso, foi implementado uma função que utiliza uma janela de tempo definida para estruturar os dados de entrada (função *sliding-windows()* que avança diariamente - mantendo o mesmo intervalo -, também realizado a inclusão de recursos adicionais (*feature engineering*) ao nodo do grafo, adicionando a população absoluta regional e a implementação de validação cruzada em séries temporais - respeitando a natureza cronológica dos dados.

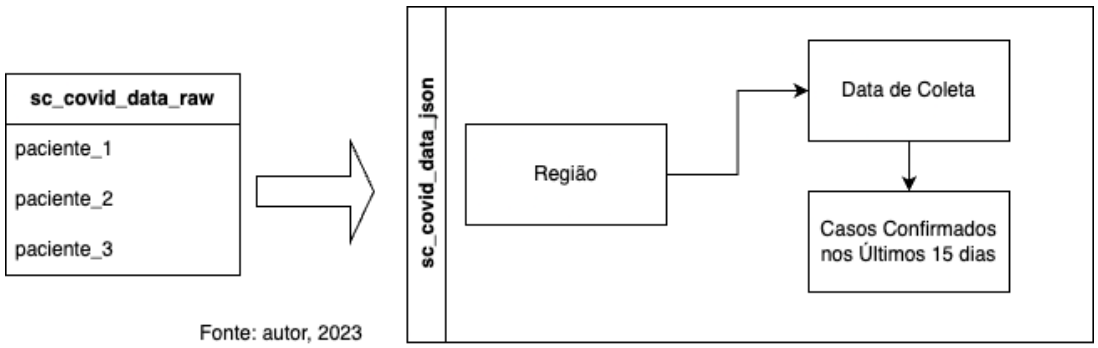
4.5 Exploração e Preparação

A base de dados anonimizada de Casos Confirmados COVID-19 é fornecida pelo Portal de Dados Abertos do Estado de Santa Catarina e tem atualização diária, para desenvolvimento do projeto, será utilizado uma versão estática desta base. As dimensões da nossa base fica por volta dos 1.900.000 milhões registros. Toda a base passou por tratamento de dados nulos e faltantes, e foi realizado a sumarização desta base para importação no programa, construindo um arquivo JSON que aninha Regiões, as Datas e o Número de Casos Confirmados (15 dias anteriores).

O programa começa importando todas as bibliotecas necessárias (*PyTorch, NetworkX, Pandas, Numpy, Sklearn*) e estabelecendo a conexão com o banco de dados SQLite -sendo este de característica *serverless*, ou seja, não há necessidade de uma instância local, além do arquivo do banco de dados. Este será utilizado para armazenamento dos modelos.

A estrutura de dados em JSON é convertida para um DataFrame utilizando Pandas, para que possa alimentar a GCN. Os treinamentos avançam no tempo diariamente, o programa aplica uma técnica de janelas deslizantes ao DataFrame para Treinamento, Validação e Testes. Além disso, para cada nodo da rede (sendo este, uma Mesoregião do

Figura 7 – Processamento dos dados crus para formato utilizado na entrada da rede



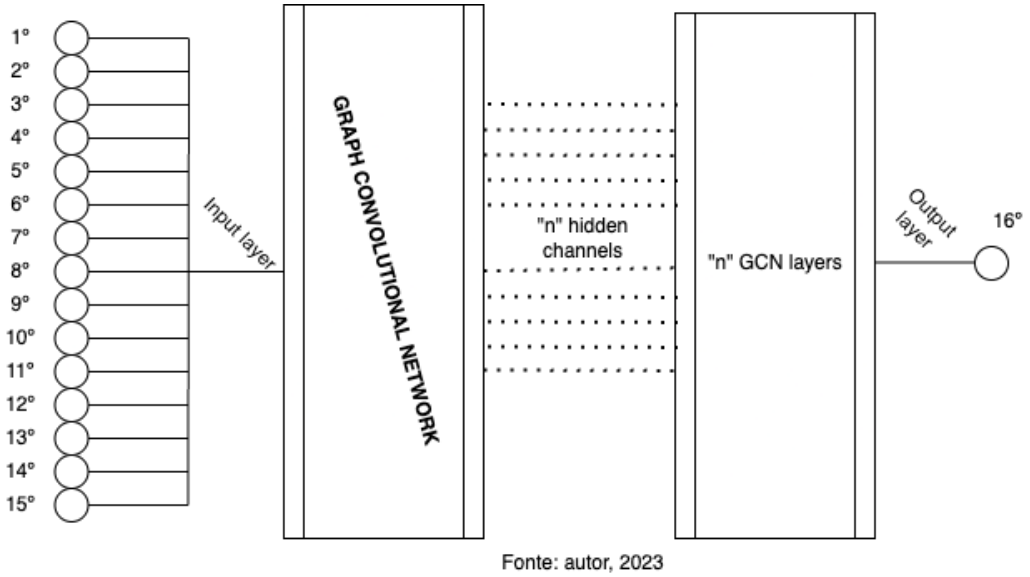
Fonte: autor, 2023

estado), são adicionados recursos adicionais (população residente, recursos hospitalares disponíveis e acesso a água potável para população da mesoregião).

4.6 Ensaios

Após a preparação dos dados, o programa procede com a otimização do modelo utilizando Optuna. Este Framework possibilita realizar, de maneira automatizada, o aperfeiçoamento de hiperparâmetros. A função objetivo, que será minimizada, é o Erro Médio Absoluto (MAE). O otimizador sugere valores para diferentes hiperparâmetros, como a taxa de aprendizado, o número de canais ocultos e a quantidade de camadas da rede, e avalia o desempenho do modelo a partir de cálculo entre o conjunto de treinamento e validação.

Figura 8 – Arquitetura utilizada para construção dos nodos de cada região - agrupamento de 15 em 15 como entradas, sendo a camada de saída a predição do 'Número de Casos Confirmados nos Últimos 15 dias' subsequente



Fonte: autor, 2023

Nesta ilustração utilizamos como "janela"o valor de 15 dias. O código utiliza a validação cruzada em séries temporais, que é adequada para dados temporais, para avaliar o desempenho do modelo. Para cada iteração, o código calcula também outras métricas de desempenho como Erro Percentual Médio Absoluto (MAPE), Erro Quadrático Médio (MSE), Raiz do Erro Quadrático Médio (RMSE), R2, Erro Percentual Absoluto Mediano (MDAPE) registrando os resultados, sendo que ao final de cada treinamento, as respectivas médias são registradas individualmente ao modelo, podendo ser observado cada ciclo de treinamento diretamente no *Optuna-dashboard*.

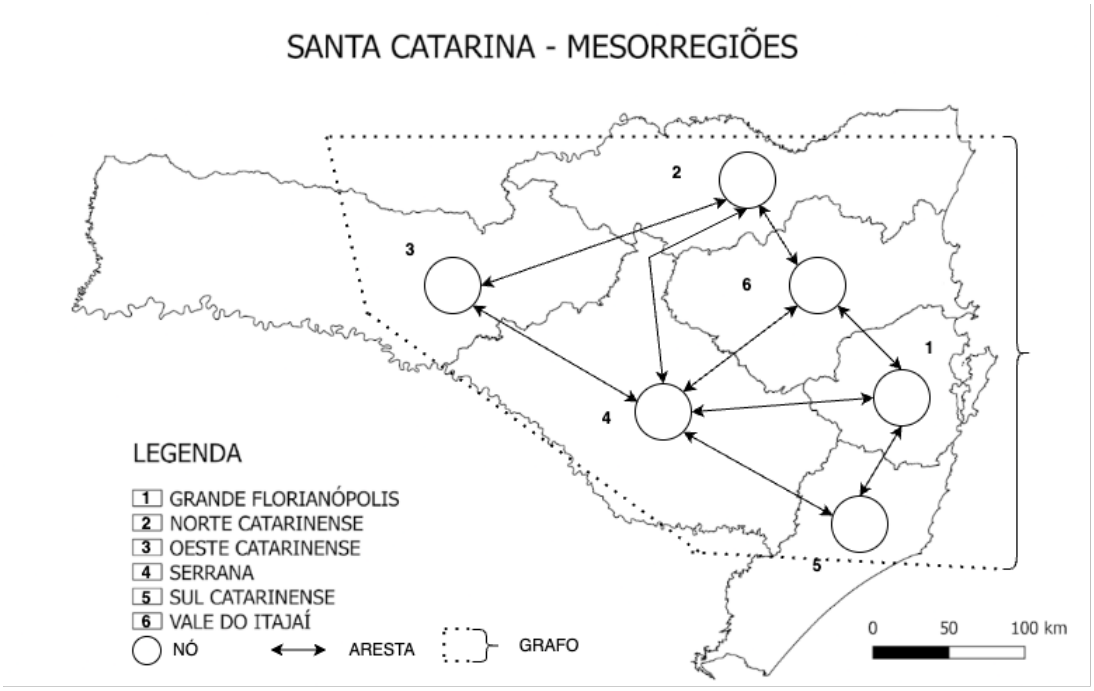
4.7 Arquitetura e especificação da Graph Convolutional Network

A arquitetura da rede neural utilizada é baseada no Graph Convolutional Network (GCN) descrita por Kipf e Welling (2016) foi implementada utilizando *TensorFlow*, porém pra utilização em nosso trabalho optamos com o uso da biblioteca *PyTorch Geometric* pela sua especialidade de aprendizado profundo em grafos. A rede é composta por camadas de GCNConv (este pacote sendo a representação em código do trabalho desenvolvido por Kipf e Welling (2016)), uma variante de convolução que opera diretamente nos grafos. Cada nó no grafo contém informações de características que representam uma região e um intervalo de tempo. As características são então propagadas através das camadas GCN para gerar previsões. A arquitetura usa a taxa de abandono (*dropout*) como uma forma de regularização para prevenir o sobreajuste.

Durante o treinamento, o modelo usa a perda L1 (*L1Loss*), uma função de perda comum para tarefas de regressão, e o otimizador Adam, que é conhecido por ser eficiente e requer pouco ajuste de hiperparâmetros. O aprendizado de taxa adaptativa é implementado usando a função *ReduceLROnPlateau*, que ajusta a taxa de aprendizado quando a métrica de validação parou de melhorar. Além disso, o script usa Optuna, um *framework* de otimização de hiperparâmetros, para sintonizar os hiperparâmetros do modelo, incluindo taxa de aprendizado, número de canais ocultos, número de camadas, e taxa de *dropout* (abandono).

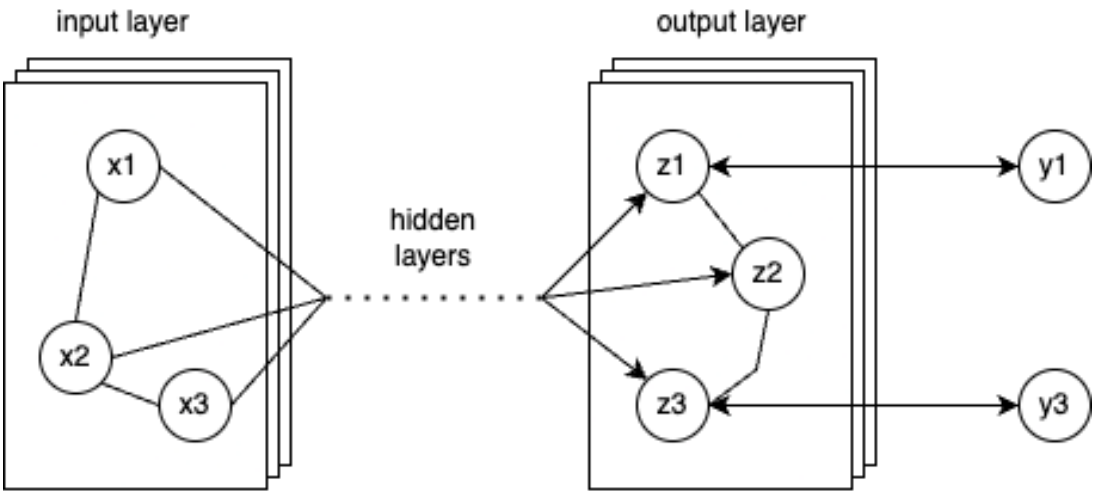
Em termos de desempenho, o modelo é avaliado usando várias métricas de erro, incluindo o erro médio absoluto (MAE), o erro percentual médio absoluto (MAPE), o erro quadrático médio (MSE), a raiz quadrada do erro quadrático médio (RMSE), o coeficiente de determinação R^2 e a Mediana do Erro Percentual Absoluto (MDAPE). Essas métricas ajudam a avaliar o desempenho do modelo tanto em termos de erros absolutos quanto relativos, oferecendo uma visão mais completa da precisão das previsões. Os resultados são visualizados usando gráficos de valores reais e previstos ao longo do tempo. A avaliação abrangente do desempenho e a visualização dos resultados facilitam a interpretação e a melhoria contínua do modelo.

Figura 9 – Representação espacial dos Nodos diante a distribuição das regiões de Santa Catarina



Fonte: autor, 2023

Figura 10 – Ilustração sobre iteração entre as camadas convolucionais



Fonte: autor, 2023

5 Análise e discussão dos resultados

Os resultados se dão a partir das comparações entre o "Número de Casos Confirmados"contados dos últimos 15 dias (estes sendo sempre 15, fixos), com o valores preditos de "Número de Casos Confirmados"para datas futuras - utilizando o princípio da "janela de dias", definitos como 15 e 7 dias.

Inicialmente os resultados métricos com janela de 15 dias apontavam um modelo capaz de generalizar predições para dados condições entrada para rede, pois os valores das métricas de erro apontaram um Erro Médio Absoluto de **1373** para toda a rede, e índice médio R^2 de **0.6423**, este último indica o quão bem as previsões do modelo podem explicar a variação nos dados verdadeiros, indicando que o modelo pode explicar aproximadamente 64,23 porcento da variância nos dados. Avaliando outras métricas, como o Erro Quadrático Médio, que significa que os erros maiores são mais penalizados, de **7364318** - que mesmo a interpretação não sendo em mesma escala e unidade, indica que há consideráveis erros, estes também explicados pelo Erro Percentual Absoluto Médio (em média, o quanto os valores previstos desviam dos valores reais em termos percentuais) de **0.6680**, ou seja, uma média de 66.8 porcento de desvio percentual.

Ao realizar a plotagem dos resultados como na Figura 5, observamos que os valores preditos realmente tem uma boa relação de entendimento de tendências e surtos diante os valores reais, porém indicando forte enviesamento nas predições.

Um recorte das 250 últimas iterações - que em escala de tempo demonstrada na Figura 6, corresponderia entre 4º bimestre de 2022 até meados do 5º bimestres de 2023 - indica que o modelo responde bem a variância dos dados reais, sendo também captadas (mesmo que discretamente) disturbios menores nos valores reais como entre os ciclos 1650 e 1700.

Mesmo com apenas 1000 iterações como exposto na Figura 7, o modelo já apresentava sinais de enviesamento, como a abrupta convergência entre valores verdadeiros e

Figura 11 – Tabela comparativa de resultados para diferentes janelas de tempo

JANELA (dias)	**AVG MAE** (rede)	**Dropout rate**	**Learning rate**	**Num Hidden Channels**	**Num hidden Layers**	**AVG R^2** (rede)	**Weight decay**	**AVG MDAPE** (rede)
7	1320	0,3473	0,01156	96	12	0.6818	0,000395	50,88
15	1373	0.2204	0.0068	16	15	0.6423	3.058	66.80

Figura 12 – Número de casos confirmados sobre iterações - em azul, dados verdadeiros, em laranja, dados preditos pelo modelo

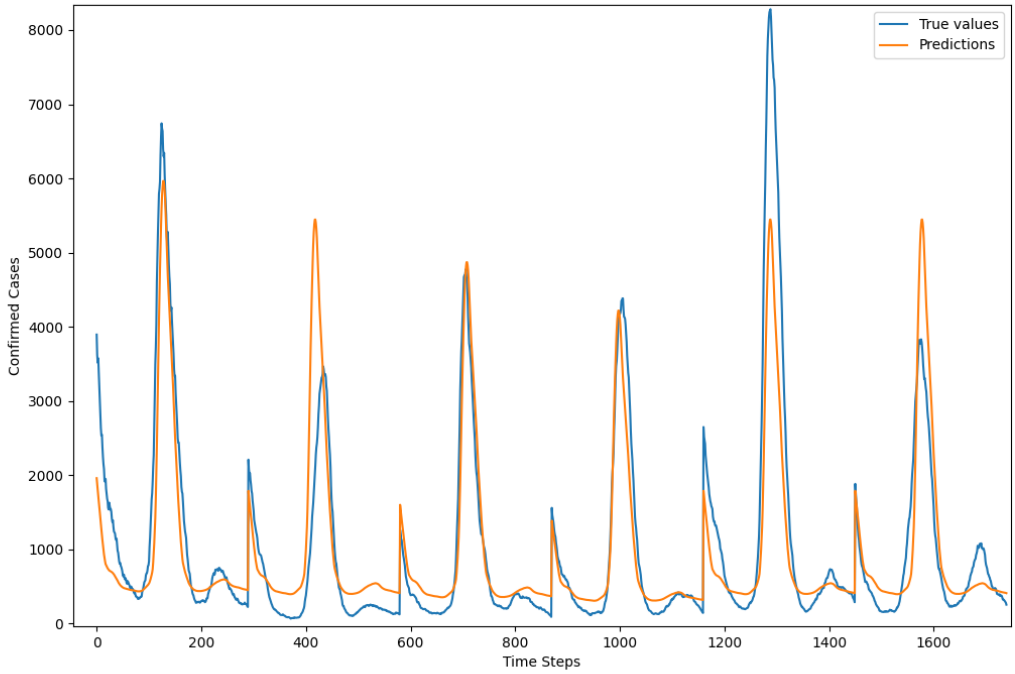
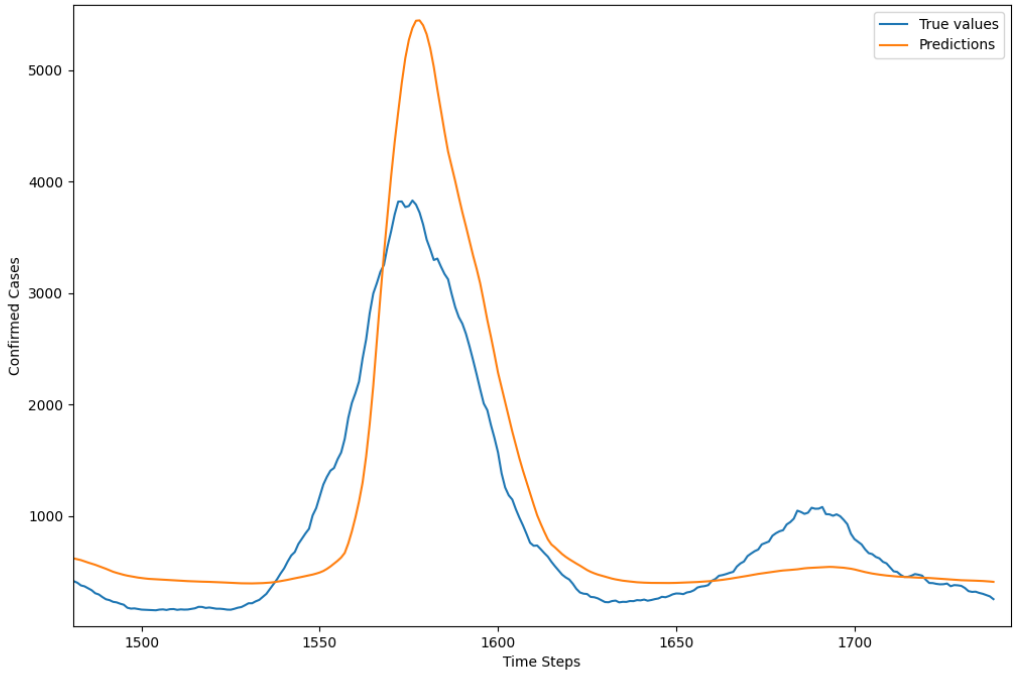
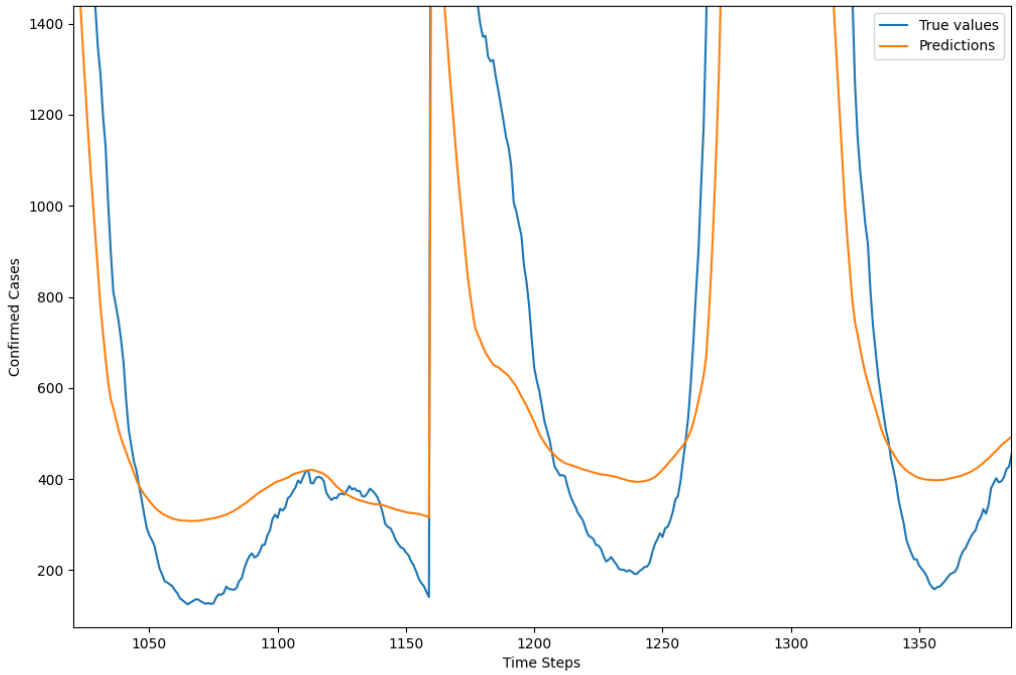


Figura 13 – Detalhamento das 250 últimas iterações



preditos, logo após ciclo 1150.

Figura 14 – Detalhamento iterações 1000 a 1400



6 Conclusões

Este trabalho focou em modelar e prever a disseminação da COVID-19 no estado de Santa Catarina através da implementação de uma Graph Convolutional Network (GCN). Utilizando uma base de dados do SUS de Santa Catarina, buscamos desenvolver uma ferramenta capaz de auxiliar na tomada de decisões e resposta a surtos da doença.

No entanto, apesar da extensa base de dados disponível - literalmente, todo paciente que deu entrada no SUS de Santa Catarina - o pré-processamento de dados para exploração da proposta do problema resultou em um dicionário de dados de apenas 1190 registros para todas as regiões do estado. Este número é resultado da quantidade de dias corridos que temos entre a primeira e último registro de pacientes com casos confirmados da COVID-19, após limpeza e preparação. Este montante sendo fonte de dos dados de Treinamento, Testes e Validações do modelo, em consequencia, desenvolvendo um modelo provavelmente enviesado.

A redução da janela de dias - de 15 para 7 - apresentou uma redução considerável quanto a mediana do erro percentual absoluto, de 66,80% para 50,88%, além de também ter apresentado um R^2 superior, de 64,23% para 68,18%, porém não podendo ser possível verificar qualquer diferença quanto a plotagem do gráfico de Número de casos confirmados sobre Iterações.

O desenvolvimento de uma entrega contínua desta rede poderia apresentar melhores resultados métricos, porém na medida que se observa as plotagens, constata-se que as predições conseguem captar tendências, porém não são tão sensíveis quanto gostaríamos, observando 15 períodos (ou 15 dias), além de que estes resultados métricos dificilmente diminuiriam o provável enviesamento deste.

6.1 Trabalhos futuros

A extensibilidade desta pesquisa abre campo para estudos futuros. Seria interessante explorar a aplicação deste modelo ou de modelos semelhantes em outras doenças com características comuns à COVID-19. Este trabalho poderia ser estendido para criar modelos que possam prever vários aspectos, como a razão entre óbitos e número de infectados, taxa de recuperação, ou chance de recuperação após entrada na Unidade de Tratamento Intensivo - todos estes presentes na base de dados original.

Além disso, existem outras arquiteturas de redes que possam apresentar melhores resultados dadas as condições e objetivos traçados no pré-processamento dos dados, como as Graph Attention Networks (GANs - Redes de Atenção em Grafo), devido a maneira

como define e lida com diferentes pesos para diferentes nodos - chamados como "Pesos de Atenção". Na arquitetura explorada com GCN, os pesos entre as conexões entre nodos são equilibrado entre todos os nodos da rede, desta forma, podendo a GAN ter resultados provavelmente menos enviesados.

Foi desenvolvido junto ao trabalho apresentado uma GAN para fins de experimentação, com a mesma estrutura de dados e condições, porém esse mecanismo de 'atenção' dela torna sua execução mais pesada e longa, então por questões de recursos em não conseguir rodar ambas concorrentemente, optou-se seguir com a GCN que também é capaz de desenvolver modelos preditivos, como exposto neste trabalho e trabalhos correlatos.

Outro caminho a ser explorado seria verificar se um modelo como o nosso poderia ser adaptado para prever várias doenças simultaneamente, especialmente aquelas com características epidemiológicas semelhantes. Tais esforços poderiam levar a uma melhor compreensão da propagação de doenças e a medidas de controle mais eficazes.

7 Glossário

Data Frame - Um Pandas DataFrame é uma estrutura de dados bidimensional, como uma matriz bidimensional ou uma tabela com linhas e colunas.

Framework - É uma estrutura que serve de base para a construção de aplicações web de finalidade específica cujo desenvolvimento pode ser muito custoso e/ou problemático. Com um framework é possível construir sites, aplicativos e softwares a partir de um esqueleto pré-definido, alterando apenas demais particularidades.

JSON - JSON é um padrão de formatação de dados para troca de informações entre sistemas.

NetworkX - NetworkX é um pacote Python para a criação, manipulação e estudo da estrutura, dinâmica e funções de redes complexas.

Numpy - É uma biblioteca Python que fornece um objeto de matriz multidimensional, vários objetos derivados (como matrizes e matrizes mascaradas) e uma variedade de rotinas para operações rápidas em matrizes, incluindo matemática, lógica, manipulação de forma, classificação, seleção, E/S , transformadas discretas de Fourier, álgebra linear básica, operações estatísticas básicas, simulação aleatória, entre outros.

Optuna - Estrutura de software de otimização automática de hiperparâmetros, especialmente projetada para aprendizado de máquina.

Pandas - Pandas é uma biblioteca de código aberto licenciada pela BSD que fornece estruturas de dados de alto desempenho e fáceis de usar e ferramentas de análise de dados para a linguagem de programação Python.

PyTorch - É uma biblioteca de código-fonte aberto projetada com o Python em mente e construída para projetos de aprendizado de máquina. É especializada em diferenciação automática, cálculos de tensores e aceleração de GPU. Isso o torna mais adequado para aplicativos de aprendizado de máquina de ponta, como aprendizado profundo.

PyTorch Geometric - PyG (PyTorch Geometric) é uma biblioteca construída sobre o PyTorch para escrever e treinar redes neurais de gráfico (GNNs) com facilidade para uma ampla gama de aplicativos relacionados a dados estruturados.

ReduceLROnPlateau - classe parte da biblioteca PyTorch, distribuído pelo pacote Optim - reduz a taxa de aprendizado quando uma métrica parar de melhorar. Os modelos geralmente se beneficiam da redução da taxa de aprendizado por um fator de 2 a 10 quando o aprendizado estagna. Este escalonador lê uma quantidade de métricas e se nenhuma melhora for observada por um número de épocas de 'paciência', a taxa de aprendizado é reduzida.

Sklearn - O Sklearn (ou Scikit-learn) é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python.

TensorFlow - TensorFlow é uma biblioteca de código aberto para aprendizado de máquina aplicável a uma ampla variedade de tarefas. É um sistema para criação e treinamento de redes neurais para detectar e decifrar padrões e correlações, análogo à forma como humanos aprendem e raciocinam.

Referências

ALLAMANIS, M. et al. A survey of machine learning for big code and naturalness. **ACM Computing Surveys**, Association for Computing Machinery (ACM), v. 51, n. 4, p. 1–37, jul. 2018. Disponível em: <https://doi.org/10.1145/3212695>.

BISWAL, A. Convolutional neural network. 2023. Disponível em: <https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network>.

BUCKEE, C. O. et al. Aggregated mobility data could help fight COVID-19. **Science**, American Association for the Advancement of Science (AAAS), v. 368, n. 6487, p. 145–146, abr. 2020. Disponível em: <https://doi.org/10.1126/science.abb8021>.

CATARINA, P. de Dados Abertos de S. Dados anonimizados casos confirmados covid-19. 2020. Disponível em: <https://dados.sc.gov.br/dataset/covid-19-dados-anonimizados-de-casos-confirmados/resource/76d6dfe8-7fe9-45c1-95f4-cab971803d49>.

FRANCO, C. M. R.; DUTRA, R. F. MODELOS MATEMáTICOS EM EPIDEMIOLOGIA e APLICAção NA EVOLUção DA COVID-19 NO BRASIL e NO ESTADO DA PARAíBA. **Educação, Ciência e Saúde**, Biblioteca do Centro de Educacao e Saude (CES), v. 7, n. 1, jun. 2020. Disponível em: <https://doi.org/10.20438/ecs.v7i1.269>.

GORI, M.; MONFARDINI, G.; SCARSELLI, F. A new model for learning in graph domains. In: **Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.** IEEE, 2005. Disponível em: <https://doi.org/10.1109/ijcnn.2005.1555942>.

KIPF, T. N.; WELLING, M. Semi-supervised classification with graph convolutional networks. **CoRR**, abs/1609.02907, 2016.

MAHESH, B. Machine learning algorithms - a review. **IJSR International Journal of Science and Research**, ResearchGate, p. 1–16, 2018. ISSN 2319-7064. Disponível em: <https://www.researchgate.net/profile/Batta-Mahesh/publication/344717762_Machine_Learning_Algorithms_-A_Review/links/5f8b2365299bf1b53e2d243a/Machine-Learning-Algorithms-A-Review.pdf?eid=5082902844932096>.

PACHECO, A. Introdução a redes neurais artificiais. 2015. Disponível em: <https://computacaointeligente.com.br/algoritmos/conceitos/redes-neurais-artificiais/#haykin>.

PASTOR-SATORRAS, R. et al. Epidemic processes in complex networks. **Reviews of Modern Physics**, American Physical Society (APS), v. 87, n. 3, p. 925–979, ago. 2015. Disponível em: <https://doi.org/10.1103/revmodphys.87.925>.

SANAR. Pandemias na história: o que há de semelhante e de novo na covid-19. 2020. Disponível em: <https://www.sanarmed.com/pandemias-na-historia-comparando-com-a-covid-19>.

SARKER, I. H. Machine learning: Algorithms, real-world applications and research directions. **SN Computer Science**, Springer Science and Business Media LLC, v. 2, n. 3, mar. 2021. Disponível em: <https://doi.org/10.1007/s42979-021-00592-x>.

SCARSELLI, F. et al. The graph neural network model. **IEEE Transactions on Neural Networks**, Institute of Electrical and Electronics Engineers (IEEE), v. 20, n. 1, p. 61–80, jan. 2009. Disponível em: <https://doi.org/10.1109/tnn.2008.2005605>.

VELIčKOVIć, P. et al. **Graph Attention Networks**. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1710.10903>.

WANG, Z.; YAN, W.; OATES, T. **Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline**. arXiv, 2016. Disponível em: <https://arxiv.org/abs/1611.06455>.

WERTHEIN, J. A sociedade da informação e seus desafios. **Ciência da Informação**, FapUNIFESP (SciELO), v. 29, n. 2, p. 71–77, ago. 2000. Disponível em: <https://doi.org/10.1590/s0100-19652000000200009>.

Apêndices

UTILIZAÇÃO DE REDES NEURAIS EM GRAFO PARA PREVER SURTOS DE DOENÇAS: UMA ABORDAGEM TESTANDO PROPAGAÇÃO DE INFORMAÇÃO EM UMA REDE CONVOLUCIONAL

Rian Carlos Vieira Brüggemann, Rafael de Santiago

[1]Curso de Sistemas de Informação
Departamento de Informática e EstatísticaInstituto de Informática
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

riancvb@gmail.com, r.santiago@ufsc.br

Abstract. *This work involves computationally exploring data from the Coronavirus (COVID-19) pandemic recorded in the Unified Health System (SUS) of the state of Santa Catarina. The data were extracted from a CSV and structured into graphs for problem development. A model was proposed that allows making predictions about the number of confirmed cases on a future date. A single neural network was developed with a graph-oriented architecture, each mesoregion represents a node, and the connections between the nodes are limited - just as geographically - by their neighboring mesoregions. The aim was to explore various types of hyperparameters (Dropout Rate, Learning Rate, Number of Channels, Number of Layers, and Weights) using the Optuna framework, in order to find a model that presents the lowest Mean Absolute Error (MAE) for the network, observing unknown predicted values. This model was trained using fixed time windows (15 and 7 days), advancing daily, which resulted in a model that performs well for known data and with possible bias in the predicted results.*

Resumo. *Este trabalho consiste em explorar computacionalmente os dados provenientes da pandemia do Coronavirus (COVID-19) registrados no Sistema de Saúde Unificado (SUS) do estado de Santa Catarina. Os dados foram extraídos de um CSV e orientados em grafos para estruturação e desenvolvimento do problema. Propôs-se um modelo que permita realizar predições sobre o número de casos confirmados em uma data futura. Uma única rede neural foi desenvolvida com uma arquitetura orientada a grafos, cada mesoregião representa um nodo e as conexões entre os nodos estão limitadas - assim como geograficamente - por suas mesoregiões vizinhas. Buscou-se explorar os mais variados tipos de hiperparâmetros (Taxa de Dropout, Taxa de Aprendizado, Número de Canais, Número de Camadas e Pesos) com o uso do framework Optuna, a fim de encontrar um modelo que apresente o menor Erro Absoluto Médio (MAE) para rede, observando valores preditos desconhecidos. Este modelo foi treinado utilizando janelas fixas de tempo (15 e 7 dias), avançando diariamente o que resultou em um modelo desempenha bem para dados já conhecidos e com possível enviesamento dos resultados preditos.*

1. Introdução

O aumento na ocorrência de surtos de doenças epidêmicas nas últimas décadas tem sido associado ao crescimento populacional, mudanças nos hábitos alimentares e maior interação entre humanos e animais selvagens. Exemplos históricos de pandemias, como a Gripe Espanhola, a epidemia de HIV/AIDS e a pandemia de H1N1, e mais recentemente a COVID-19, destacam a necessidade de melhorar as estratégias de prevenção e controle de futuras epidemias e pandemias (SANAR, 2020).

Em uma sociedade globalizada, novas tecnologias de comunicação permitem o rastreamento e identificação de uma quantidade crescente de informações dos usuários. A pesquisa tem evidenciado o surgimento de padrões de conectividade complexos em sistemas biológicos e sociotécnicos, com impactos significativos no comportamento de fenômenos de equilíbrio e não-equilíbrio, incluindo a disseminação de epidemias (WERTHEIN, 2000).

A pandemia de COVID-19 alterou os padrões de mobilidade humana, exigindo modelos epidemiológicos capazes de capturar os efeitos dessas mudanças na disseminação do vírus. Profissionais de diversas áreas, como físicos, epidemiologistas e cientistas da computação, dependem de modelos semelhantes para descrever a difusão de patógenos, conhecimento e inovação (PASTORSATORRAS *et al.*, 2015).

Este estudo propõe a criação de um modelo preditivo para futuros casos confirmados de COVID-19 no estado de Santa Catarina, utilizando algoritmos de Machine Learning. A abordagem utiliza séries temporais de casos confirmados por região, convertidas em um grafo onde cada nó representa uma região e cada aresta representa a interação entre regiões vizinhas. O modelo processa essa representação de grafo para fornecer a previsão do número de casos confirmados de COVID-19.

O objetivo principal da pesquisa é modelar e prever a disseminação da COVID-19 por meio da implementação de uma Rede Neural Convolucional (GCN). O script Python desenvolvido implementa uma GCN, abstraindo como a propagação de doenças ocorre em uma sociedade globalizada. O desempenho do modelo é avaliado por meio da validação cruzada de séries temporais em diferentes partes do conjunto de dados, utilizando várias métricas de erro (KIPF; WELLING, 2016).

2. Fundamentação Teórica

A pandemia da COVID-19 trouxe à tona a importância dos modelos matemáticos aplicados à epidemiologia, que representam fenômenos do mundo real através de equações matemáticas, permitindo compreender o comportamento das variáveis envolvidas em epidemias [Franco and Dutra 2020]. Modelos como SIS, SIR e SIRS (Suscetível-Infectado-Suscetível, Suscetível-Infectado-Removido e Suscetível-Infectado-Removido-Suscetível, respectivamentes), baseados em compartimentos, dividem uma população em classes que representam seu estado atual no desenvolvimento da doença, como Suscetíveis, Infectados e Removidos, permitindo determinar parâmetros e projetar a evolução de epidemias [Franco and Dutra 2020].

Machine Learning (ML) é uma área de estudos na computação onde algoritmos são empregados para ensinar máquinas a aprender e reconhecer padrões a partir de experiências de processamento de dados [Allamanis et al. 2018]. Os algoritmos de ML são

divididos principalmente em quatro categorias: Aprendizado Supervisionado, Aprendizado Não-supervisionado, Aprendizado Semi-supervisionado e Aprendizado por Reforço [Sarker 2021].

As Redes Neurais Artificiais (ANNs) são modelos de aprendizado de máquina que simulam a maneira como os neurônios humanos interagem e processam informações. As CNNs são uma classe de ANNs desenvolvidas para processar dados de imagem, inspiradas na organização do córtex visual de animais. Elas são especialmente adequadas para processar dados estruturados em grade, como imagens, e têm se mostrado eficientes no processamento de séries temporais [Wang et al. 2016].

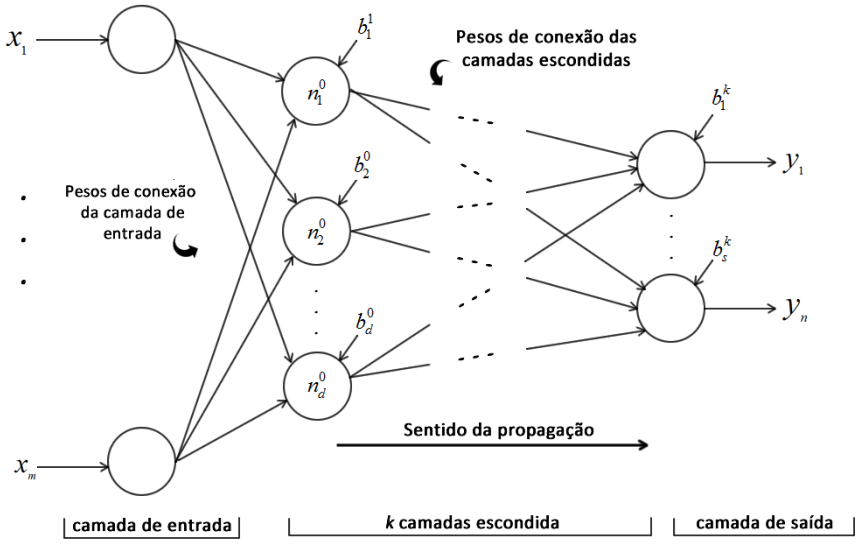


Figura 1. Exemplo de uma rede neural artificial com k camadas escondidas [Pacheco 2015]

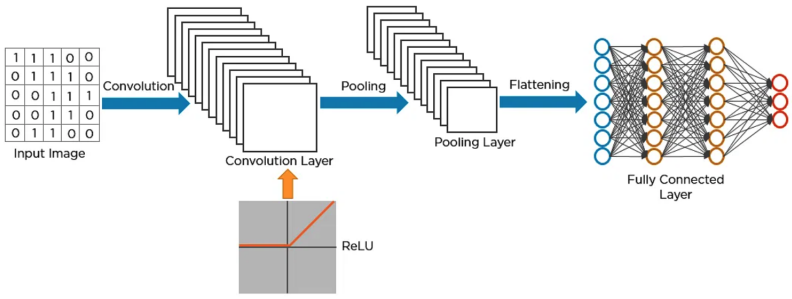
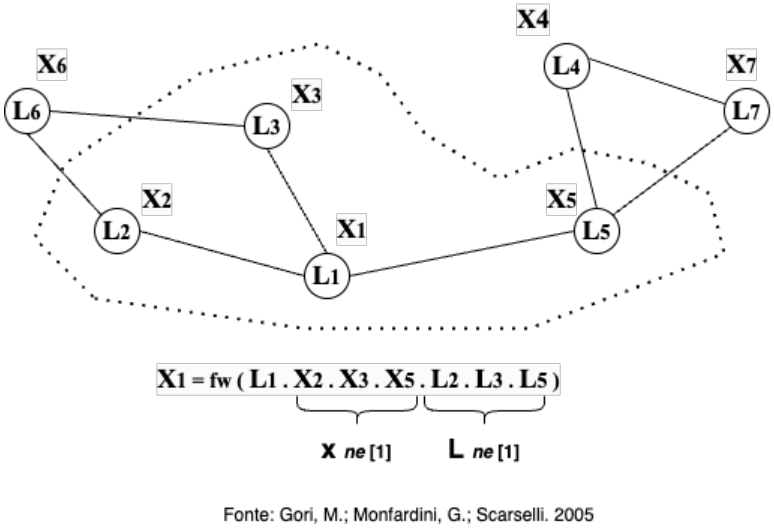


Figura 2. Estrutura de uma CNN que com função de ativação ReLU e *Pooling layers* (camadas de agrupamento), seguido pelo *Flattening* (achatamento), podendo esta servir como entrada de uma nova teia convolucional [Biswal 2023]

As Redes Neurais em Grafo (GNNs), por outro lado, são redes neurais que operam em dados representados como grafos. Elas são capazes de capturar e aprender características e padrões que são influenciados pelas estruturas de grafos, permitindo análises mais profundas em tarefas de classificação de nós, ligação de predição ou classificação de grafos inteiros [Scarselli et al. 2009].



Fonte: Gori, M.; Monfardini, G.; Scarselli. 2005

Figura 3. O estado de x1 depende da informação do vizinho

As GCNs são uma extensão das GNNs, projetadas para operar diretamente em dados representados como grafos. Elas realizam a convolução no domínio espectral do grafo, levando em consideração a topologia do grafo e as características dos nós. As GCNs são altamente paralelizáveis e podem ser facilmente escaladas para processar grafos com milhões de nós [Kipf and Welling 2016].

3. Trabalhos Relacionados

3.1. Machine Learning: Algorithms, Real-World Applications and Research Directions

Este artigo explora o vasto campo do Machine Learning (Aprendizado de Máquina), abordando algoritmos, aplicações no mundo real e direções futuras de pesquisa. Este trabalho de Sarker (2021) tem foco na explicação de algoritmos chave e técnicas de aprendizado de máquina, o artigo fornece uma visão abrangente sobre as tendências atuais neste campo. O trabalho traz exemplos de aplicações práticas do aprendizado de máquina em diferentes setores, desde saúde e educação até ciência dos dados e engenharia. É destacado principalmente o impacto significativo do aprendizado de máquina na automação de processos, otimização de operações e melhoria da tomada de decisões em vários setores da indústria.

São trazidos fatos que indicam direções promissoras para pesquisas futuras em aprendizado de máquina, identificando áreas-chave que exigirão mais atenção e desenvolvimento, como a explicabilidade dos modelos, questões éticas, a necessidade de datasets de alta qualidade e o equilíbrio entre privacidade e utilidade dos dados. Por fim, os autores avaliam a necessidade de avançar além dos métodos de aprendizado supervisionado e não supervisionado, para incorporar técnicas de aprendizado por reforço, capazes de lidar com ambientes complexos e em constante mudança.

3.2. Drug Repositioning for SARS-CoV-2 Based on Graph Neural Network

Nos últimos anos houve um aumento de pesquisas que utilizam algoritmos de *Machine Learning* na área de reposicionamento de drogas, uma seção de estudos da farmacologia

que tem como objetivo a descoberta de drogas que ofereçam menor risco sobre retorno comparado a outras drogas já utilizadas, ampliando estratégias e inovando a indústria com suas respectivas necessidades. Neste artigo de Liu (2020), foi apresentado um efetivo modelo baseado em grafos utilizando técnica de *Deep Learning* (Aprendizado Profundo) *deep2CoV*, que infere sistematicamente novos relacionamentos de droga-doença para o *SARS-CoV-2*. O conceito por trás do deep2CoV é fundir diversas informações sobre *SARS-CoV-2* em uma rede droga-droga, rede droga-doença e rede droga-alvo, desta forma, inferindo potenciais drogas para o *SARS-CoV-2*, utilizando redes convolucionais baseadas em grafos. Sumariamente, o artigo identifica potenciais vantagens na utilização de GNNs (*Graph Neural Networks*) no reposicionamento de drogas devido a redução de espaçamento a integração de diferentes estruturas de informação (redes relacionais droga-droga, droga-doença e droga-alvo) sobre o *SARS-CoV-2* (estrutura em grafo), esta que permite uma visão mais geral das possíveis interações de droga com a doença. Essa abordagem fundamenta o poder e flexibilidade das GNNs para tratar de problemas complexos e multidimensionais na área da saúde.

3.3. Semi-Supervised Classification with Graph Convolutional Networks

Sob o prisma das redes neurais convolucionais, algoritmo amplamente utilizado em atividades de classificação de dados, a publicação apresenta uma abordagem escalável para aprendizagem de máquina semi-supervisionada baseada em modelos de dados estruturados em grafos. Condicionando um modelo de rede neural *f(X, A)* em uma estrutura de grafos, onde **X** sendo o conjunto de dados e **A** a matriz adjacente, espera-se que em cenários bem treinados - seguindo método de propagação layerwise - matriz **A** tenha informações que não estão presentes no conjunto **X**. No modelo estudado por Kipf e Welling (2016) a implementação foi executada utilizando *TensorFlow* usando multiplicação de matriz esparsa-densa. Como testagem, foi implementado uma rede semi-supervisionada para classificação de documentos em uma rede de citações, uma rede semi-supervisionada para classificar entidades em grafos, avaliação de algumas abordagens de propagações em grafos e análise em tempo real de grafos aleatórios. Estes testes asseguraram que a rede convolucional baseada em grafos é capaz de codificar ambas as estruturas de grafos e nodos de maneira útil e eficiente, inclusive computacionalmente.

De forma complementar, o estudo propõe um modelo de treinamento eficiente que usa a normalização espectral de grafos, permitindo que o modelo generalize bem para grafos invisíveis no treinamento. A normalização espectral, uma técnica da teoria dos grafos, é fundamental para o sucesso do método, pois facilita o aprendizado de padrões relevantes nas características dos nós e na estrutura do grafo. A abordagem do artigo mostra que o aprendizado semi-supervisionado em grafos, quando aplicado corretamente, pode ser uma ferramenta poderosa para classificar nodos com poucos rótulos disponíveis. O artigo sinaliza um caminho promissor para a pesquisa futura em aprendizado semi-supervisionado em grafos, especialmente na exploração de estruturas de dados complexas e heterogêneas.

3.4. Graph Attention Networks

Com o objetivo de melhorar o processamento de informações estruturadas como grafos por redes neurais, os autores utilizaram as Graph Attention Networks (GANs) neste trabalho. Estas redes utilizam mecanismos de atenção, inspirados em redes neurais recorrentes,

que permitem que o modelo pondere as contribuições dos nós vizinhos em um grafo, ao invés de tratá-los de maneira uniforme [Veličković et al. 2017].

A ideia central é permitir que cada nó em um grafo possa focar em diferentes nós em sua vizinhança em diferentes graus quando está sendo atualizado. Isso é feito através de um conjunto de coeficientes de atenção que são aprendidos de forma adaptativa, permitindo que o modelo seja capaz de capturar diferentes tipos de estruturas em um grafo. As GANs mostraram-se eficazes na realização de tarefas de classificação de nós e de grafos, demonstrando uma capacidade superior às abordagens tradicionais em várias métricas.

Os autores também destacaram a eficiência computacional das GANs, argumentando que, como cada nó atende apenas a uma seleção de seus vizinhos, em vez de toda a rede, o modelo é mais escalável e eficiente. Eles validaram esses resultados em uma série de experimentos, incluindo a classificação de documentos em um conjunto de dados de citações e a classificação de partes de um objeto 3D.

3.5. Mathematical models of infectious disease transmission

Segundo Grassly e Fraser (2008), representações matemáticas e análises de doenças infecciosas são áreas correlatas que diante os diversos avanços na área de processamento computacional, tem sido fator chave na evolução da medicina ambulatorial e das análises epidemiológicas. Modelos matemáticos de transmissão conectados a processos biológicos de monitoramento e transmissão de dados levantou duas propriedades estatísticas principais de uma epidemia, a taxa de infecção contínua no início da transmissão de um patógeno por distribuição de tempo. Sobre os desafios dos modelos matemáticos em torno das análises patogênicas, destacam-se: evolução do patógeno, devido a sua dinâmica biológica e diversa, desafia os modelos epidêmicos e de inferência; métodos estatísticos, pois novas infecções estão relacionadas a variáveis desconhecidas.

4. Projeto

No decorrer deste capítulo estão expostos detalhes de implementação da GCN, estratégia para agrupamento dos dados, explorações e preparações realizadas anterior aos ensaios, como também descrição sobre a arquitetura da rede e métricas avaliativas.

4.1. Especificação

Nesta seção ocorre o detalhamento dos métodos utilizados nos experimentos desse trabalho, como as configurações de redes neurais e as da montagem dos grafos.

Foi desenvolvido um modelo que possa aprender por exemplos de séries temporais que utiliza como entrada um grafo que possui rotulagem número absoluto de casos confirmados de uma região do estado de Santa Catarina, como também a sua população relativa. Realizou-se divisão dos dados em Treinamento, Validação e Teste para fins de experimentação, aprimoramento da capacidade de generalização do modelo.

A fonte de dados é uma base de dados dos pacientes que deram entrada no Sistema de Saúde Unificado (SUS) do estado de Santa Catarina com quadro de suspeita de COVID-19. Usando SL desenvolvemos um agente que estime o número de casos confirmados do 16º dia, avançando diariamente, assim como avaliamos a acurácia das hipóteses levantadas, se esta prediz corretamente a novos valores fornecidos como entrada registros diferentes dos utilizados para treinamento.

4.2. Dados

Para treinamento e validação das rede neurais desenvolvidas, foram utilizados dados provenientes de cada região - estes já pré-processados, organizados em um dicionário de dados, compondo um outro dicionário aninhado - tendo o primeiro dicicionário a região (mesoregião de Santa Catarina) como chave, e o 2º dicionário tendo como chave uma data YYYY-MM-DD, e o valor desta chave somatório dos casos confirmados nos 15 dias anteriores. Fonte: [de Dados Abertos de Santa Catarina 2020].

4.3. Redes Neurais em Grafos

Durante o desenvolvimento foram avaliados modelos de redes neurais utilizando grafos (GCN), variando o número de camadas, número de canais, a taxa de aprendizado, taxa de *dropout*, taxa de peso entre relações, o tamanho dos lotes de entrada e a quantidade de interações, de forma a encontrar uma ou mais configurações eficientes com os parâmetros e dados utilizados. Os modelos desenvolvidos foram comparados a partir do erro absoluto médio, ou seja, a relação entre os casos confirmados de uma data, com uma data futura não vista. Outras métricas avaliativas vistas no modelo foi o erro percentual médio absoluto, erro quadrático médio, raiz do erro quadrático médio, coeficiente de determinação (R^2) e o erro percentual absoluto mediano. A métrica escolhida para otimização no framework Optuna foi o erro absoluto médio, por ser de mesma natureza e escala em que os dados de treinamento e predição consistem.

5. Desenvolvimento

Foi utilizada uma Graph Convolutional Network (GCN) para prever o número de casos confirmados de COVID-19 por região e data. O código utiliza a biblioteca PyTorch e a PyTorch Geometric para criar e treinar a rede neural. Para otimizar os parâmetros do modelo, o código utiliza a biblioteca Optuna. Além disso, foi implementado uma função que utiliza uma janela de tempo definida para estruturar os dados de entrada (função *sliding-windows()* que avança diariamente - mantendo o mesmo intervalo -, também realizado a inclusão de recursos adicionais (*feature engineering*) ao nodo do grafo, adicionando a população absoluta regional e a implementação de validação cruzada em séries temporais - respeitando a natureza cronológica dos dados.

5.1. Exploração e Preparação

A base de dados anonimizada de Casos Confirmados COVID-19 é fornecida pelo Portal de Dados Abertos do Estado de Santa Catarina e tem atualização diária, para desenvolvimento do projeto, será utilizado uma versão estática desta base. As dimensões da nossa base fica por volta dos 1.900.000 milhões registros. Toda a base passou por tratamento de dados nulos e faltantes, e foi realizado a sumarização desta base para importação no programa, construindo um arquivo JSON que aninha Regiões, as Datas e o Número de Casos Confirmados (15 dias anteriores).

O programa começa importando todas as bibliotecas necessárias (*PyTorch, NetworkX, Pandas, Numpy, Sklearn*) e estabelecendo a conexão com o banco de dados SQLite -sendo este de característica *serverless*, ou seja, não há necessidade de uma instância local, além do arquivo do banco de dados. Este será utilizado para armazenamento dos modelos.

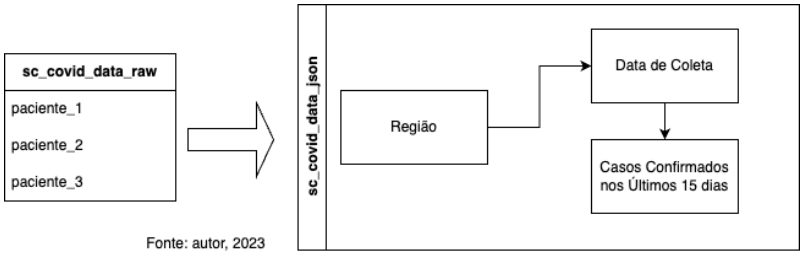


Figura 4. Processamento dos dados crus para formato utilizado na entrada da rede

A estrutura de dados em JSON é convertida para um DataFrame utilizando Pandas, para que possa alimentar a GCN. Os treinamentos avançam no tempo diariamente, o programa aplica uma técnica de janelas deslizantes ao DataFrame para Treinamento, Validação e Testes. Além disso, para cada nodo da rede (sendo este, uma Mesoregião do estado), são adicionados recursos adicionais (população residente, recursos hospitalares disponíveis e acesso a água potável para população da mesoregião).

5.2. Ensaios

Após a preparação dos dados, o programa procede com a otimização do modelo utilizando Optuna. Este Framework possibilita realizar, de maneira automatizada, o aperfeiçoamento de hiperparâmetros. A função objetivo, que será minimizada, é o Erro Médio Absoluto (MAE). O otimizador sugere valores para diferentes hiperparâmetros, como a taxa de aprendizado, o número de canais ocultos e a quantidade de camadas da rede, e avalia o desempenho do modelo a partir de cálculo entre o conjunto de treinamento e validação.

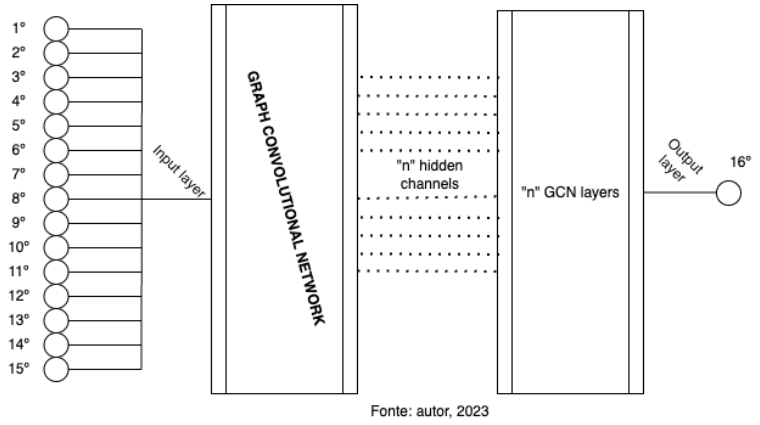


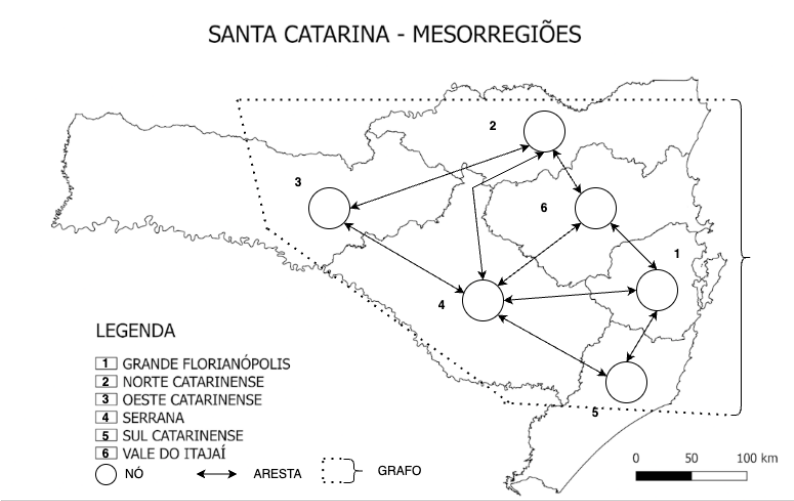
Figura 5. Arquitetura utilizada para construção dos nodos de cada região - agrupamento de 15 em 15 como entradas, sendo a camada de saída a predição do 'Número de Casos Confirmados nos Últimos 15 dias' subsequente

Nesta ilustração utilizamos como "janela"o valor de 15 dias. O código utiliza a validação cruzada em séries temporais, que é adequada para dados temporais, para avaliar o desempenho do modelo. Para cada iteração, o código calcula também outras métricas de desempenho como Erro Percentual Médio Absoluto (MAPE), Erro Quadrático Médio (MSE), Raiz do Erro Quadrático Médio (RMSE), R2, Erro Percentual Absoluto Mediano

(MDAPE) registrando os resultados, sendo que ao final de cada treinamento, as respectivas médias são registradas individualmente ao modelo, podendo ser observado cada ciclo de treinamento diretamente no *Optuna-dashboard*.

5.3. Arquitetura e especificação da Graph Convolutional Network

A arquitetura da rede neural utilizada é baseada no Graph Convolutional Network (GCN) descrita por Kipf e Welling (2016) foi implementada utilizando *TensorFlow*, porém pra utilização em nosso trabalho optamos com o uso da biblioteca *PyTorch Geometric* pela sua especialidade de aprendizado profundo em grafos. A rede é composta por camadas de GCNConv (este pacote sendo a representação em código do trabalho desenvolvido por Kipf e Welling (2016)), uma variante de convolução que opera diretamente nos grafos. Cada nó no grafo contém informações de características que representam uma região e um intervalo de tempo. As características são então propagadas através das camadas GCN para gerar previsões. A arquitetura usa a taxa de abandono (*dropout*) como uma forma de regularização para prevenir o sobreajuste.



Fonte: autor, 2023

Figura 6. Representação espacial dos Nodos diante a distribuição das regiões de Santa Catarina

Durante o treinamento, o modelo usa a perda L1 (*L1Loss*), uma função de perda comum para tarefas de regressão, e o otimizador Adam, que é conhecido por ser eficiente e requer pouco ajuste de hiperparâmetros. O aprendizado de taxa adaptativa é implementado usando a função *ReduceLROnPlateau*, que ajusta a taxa de aprendizado quando a métrica de validação parou de melhorar. Além disso, o script usa Optuna, um *framework* de otimização de hiperparâmetros, para sintonizar os hiperparâmetros do modelo, incluindo taxa de aprendizado, número de canais ocultos, número de camadas, e taxa de *dropout* (abandono).

Em termos de desempenho, o modelo é avaliado usando várias métricas de erro, incluindo o erro médio absoluto (MAE), o erro percentual médio absoluto (MAPE), o erro quadrático médio (MSE), a raiz quadrada do erro quadrático médio (RMSE), o coeficiente de determinação R^2 e a Mediana do Erro Percentual Absoluto (MDAPE). Essas métricas

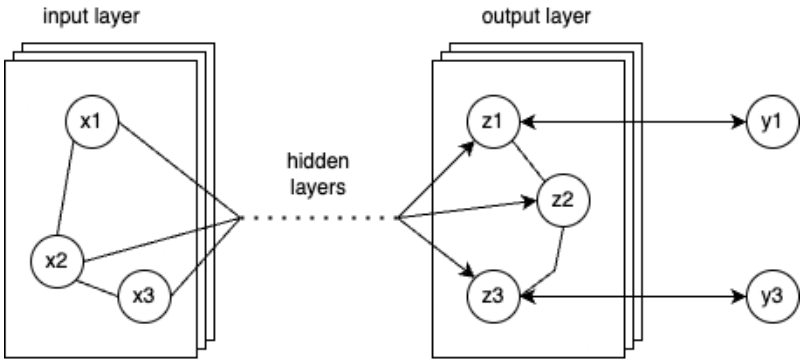


Figura 7. Ilustração sobre iteração entre as camadas convolucionais

ajudam a avaliar o desempenho do modelo tanto em termos de erros absolutos quanto relativos, oferecendo uma visão mais completa da precisão das previsões. Os resultados são visualizados usando gráficos de valores reais e previstos ao longo do tempo. A avaliação abrangente do desempenho e a visualização dos resultados facilitam a interpretação e a melhoria contínua do modelo.

5.4. Resultados e Análises

Os resultados se dão a partir das comparações entre o ”Número de Casos Confirmados”contados dos últimos 15 dias (estes sendo sempre 15, fixos), com o valores preditos de ”Número de Casos Confirmados”para datas futuras - utilizando o princípio da ”janela de dias”, definitos como 15 e 7 dias.

Inicialmente os resultados métricos com janela de 15 dias apontavam um modelo capaz de generalizar predições para dados condições entrada para rede, pois os valores das métricas de erro apontaram um Erro Médio Absoluto de **1373** para toda a rede, e índice médio R² de **0.6423**, este último indica o quão bem as previsões do modelo podem explicar a variação nos dados verdadeiros, indicando que o modelo pode explicar aproximadamente 64,23 porcento da variância nos dados. Avaliando outras métricas, como o Erro Quadrático Médio, que significa que os erros maiores são mais penalizados, de **7364318** - que mesmo a interpretação não sendo em mesma escala e unidade, indica que há consideráveis erros, estes também explicados pelo Erro Percentual Absoluto Médio (em média, o quanto os valores previstos desviam dos valores reais em termos percentuais) de **0.6680**, ou seja, uma média de 66.8 porcento de desvio percentual.

JANELA (dias)	**AVG MAE** (rede)	**Dropout rate**	**Learning rate**	**Num Hidden Channels**	**Num hidden Layers**	**AVG R²** (rede)	**Weight decay**	**AVG MDAPE** (rede)
7	1320	0,3473	0,01156	96	12	0.6818	0,000395	50,88
15	1373	0.2204	0.0068	16	15	0.6423	3.058	66.80

Figura 8. Tabela comparativa de resultados para diferentes janelas de tempo

Ao realizar a plotagem dos resultados como na Figura 5, observamos que os valores preditos realmente tem uma boa relação de entendimento de tendências e surtos diante os valores reais, porém indicando forte enviesamento nas predições.

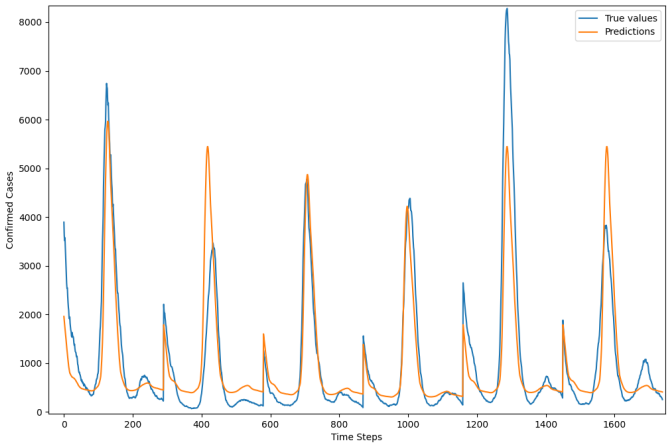


Figura 9. Número de casos confirmados sobre iterações - em azul, dados verdadeiros, em laranja, dados preditos pelo modelo

Um recorte das 250 últimas iterações - que em escala de tempo demonstrada na Figura 6, corresponderia entre 4º bimestre de 2022 até meados do 5º bimestres de 2023 - indica que o modelo responde bem a variância dos dados reais, sendo também captadas (mesmo que discretamente) disturbios menores nos valores reais como entre os ciclos 1650 e 1700.

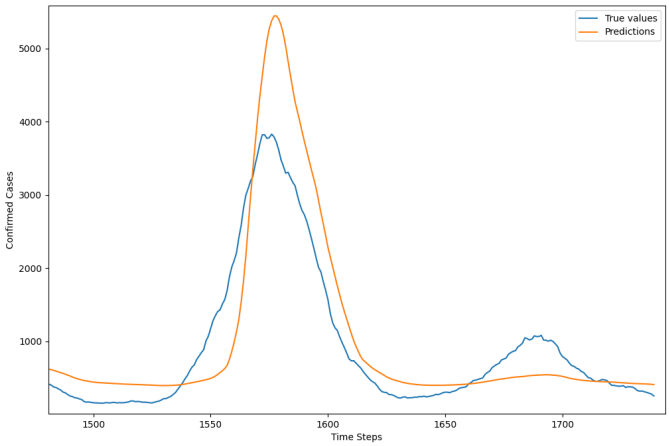


Figura 10. Detalhamento das 250 últimas iterações

Mesmo com apenas 1000 iterações como exposto na Figura 7, o modelo já apresentava sinais de enviesamento, como a abrupta convergência entre valores verdadeiros e preditos, logo após ciclo 1150.

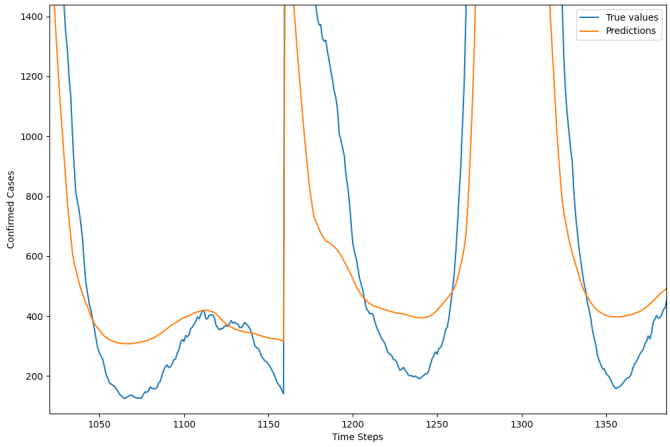


Figura 11. Detalhamento iterações 1000 a 1400

6. Conclusões

Este trabalho focou em modelar e prever a disseminação da COVID-19 no estado de Santa Catarina através da implementação de uma Graph Convolutional Network (GCN). Utilizando uma base de dados do SUS de Santa Catarina, buscamos desenvolver uma ferramenta capaz de auxiliar na tomada de decisões e resposta a surtos da doença.

No entanto, apesar da extensa base de dados disponível - literalmente, todo paciente que deu entrada no SUS de Santa Catarina - o pré-processamento de dados para exploração da proposta do problema resultou em um dicionário de dados de apenas 1190 registros para todas as regiões do estado. Este número é resultado da quantidade de dias corridos que temos entre a primeira e último registro de pacientes com casos confirmados da COVID-19, após limpeza e preparação. Este montante sendo fonte de dos dados de Treinamento, Testes e Validações do modelo, em consequencia, desenvolvendo um modelo provavelmente enviesado.

A redução da janela de dias - de 15 para 7 - apresentou uma redução considerável quanto a mediana do erro percentual absoluto, de 66,80% para 50,88%, além de também ter apresentado um R^2 superior, de 64,23% para 68,18%, porém não podendo ser possível verificar qualquer diferença quanto a plotagem do gráfico de Número de casos confirmados sobre Iterações.

O desenvolvimento de uma entrega contínua desta rede poderia apresentar melhores resultados métricos, porém na medida que se observa as plotagens, constata-se que as predições conseguem captar tendências, porém não são tão sensíveis quanto gostaríamos, observando 15 períodos (ou 15 dias), além de que estes resultados métricos dificilmente diminuiriam o provável enviesamento deste.

7. Trabalhos Futuros

A extensibilidade desta pesquisa abre campo para estudos futuros. Seria interessante explorar a aplicação deste modelo ou de modelos semelhantes em outras doenças com características comuns à COVID-19. Este trabalho poderia ser estendido para criar modelos

que possam prever vários aspectos, como a razão entre óbitos e número de infectados, taxa de recuperação, ou chance de recuperação após entrada na Unidade de Tratamento Intensivo - todos estes presentes na base de dados original.

Além disso, existem outras arquiteturas de redes que possam apresentar melhores resultados dadas as condições e objetivos traçados no pré-processamento dos dados, como as Graph Attention Networks (GANs - Redes de Atenção em Grafo), devido a maneira como define e lida com diferentes pesos para diferentes nodos - chamados como ”Pesos de Atenção”. Na arquitetura explorada com GCN, os pesos entre as conexões entre nodos são equilibrado entre todos os nodos da rede, desta forma, podendo a GAN ter resultados provavelmente menos enviesados.

Foi desenvolvido junto ao trabalho apresentado uma GAN para fins de experimentação, com a mesma estrutura de dados e condições, porém esse mecanismo de 'atenção' dela torna sua execução mais pesada e longa, então por questões de recursos em não conseguir rodar ambas concorrentemente, optou-se seguir com a GCN que também é capaz de desenvolver modelos preditivos, como exposto neste trabalho e trabalhos correlatos.

Outro caminho a ser explorado seria verificar se um modelo como o nosso poderia ser adaptado para prever várias doenças simultaneamente, especialmente aquelas com características epidemiológicas semelhantes. Tais esforços poderiam levar a uma melhor compreensão da propagação de doenças e a medidas de controle mais eficazes.

Referências

Allamanis, M., Barr, E. T., Devanbu, P., and Sutton, C. (2018). A survey of machine learning for big code and naturalness. *ACM Computing Surveys*, 51(4):1–37.

Biswal, A. (2023). Convolutional neural network.

de Dados Abertos de Santa Catarina, P. (2020). Dados anonimizados casos confirmados covid-19.

Franco, C. M. R. and Dutra, R. F. (2020). MODELOS MATEMáTICOS EM EPIDEMIOLOGIA e APLICAção NA EVOLUção DA COVID-19 NO BRASIL e NO ESTADO DA PARAíBA. *Educação, Ciência e Saúde*, 7(1).

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

Pacheco, A. (2015). Introdução a redes neurais artificiais.

Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(3).

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2017). Graph attention networks.

Wang, Z., Yan, W., and Oates, T. (2016). Time series classification from scratch with deep neural networks: A strong baseline.

APÊNDICE A – Código-fonte

```
# link repositorio: https://github.com/RianBrug/gcn_for_covid_sc_brazil
# data processing to generate json

from asyncio import constants
from datetime import date, timedelta
import time
from utils import Utils
import constants
import json
import pandas as pd
from pandas_profiling import ProfileReport
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

start_time = time.time()
# decide which path to follow
# you can skip the first part if you already have the confirmed_cases_by_region_and
run_load = False # if True, will load the csv file and create the json file
run_from_file = False # if True, will load the json file and run the model
mount_whole_csv_encoded = False # if True, will mount the whole csv encoded
run_from_file_improved = True # if True, will load the json file and run the model
home = '/Users/rcvb/Documents/tcc_rian/code'

if run_load:
    ## CHANGE TO YOUR OWN PATH
    ## Este conjunto de dados apresenta a relação de casos confirmados de COVID-19
    ## Base de Dados do Governo do Estado - BOAVISTA
    f = open(f"{home}/assets/boavista_covid_dados_abertos.csv", "r")

    paciente = -1
    confirmed_cases_by_region_and_date = {} # dict region as a keys
    for linha in f:
```

```
        print(f'processing paciente n° {paciente}...')
    # '2022-06-11 16:00:07;SIM;2022-02-10;IGNORADO;FEBRE;;;NAO INTERNADO;NAO INTERNADO U
        paciente += 1
        if paciente == 0: continue
        fields = str(linha).split(";")
        if len(fields) < 17:
            continue
        if fields[13] == "NULL": #
            continue
        data = str(fields[2]) # date yyyy-mm-dd
        if fields[14] not in confirmed_cases_by_region_and_date: # if macroregion does n
            confirmed_cases_by_region_and_date[fields[14]] = {} # dict dates as keys
        #para cada região, uma data corresponderá a janela da soma de infectados em cada
        a_date = date.fromisoformat(data)
        #leitura de todas as colunas da linha
        for i in range(0,15):
            d = a_date + timedelta(days=i)
            novaData = d.isoformat()
            if novaData not in confirmed_cases_by_region_and_date[fields[14]]:
                confirmed_cases_by_region_and_date[fields[14]][novaData] = 0
            confirmed_cases_by_region_and_date[fields[14]][novaData] +=1 # adiciona soma

    # dump confirmed_cases_by_region_and_date dict to a json file
    with open(f"{home}/assets/confirmed_cases_by_region_and_date.json", "w") as outfile:
        json.dump(confirmed_cases_by_region_and_date, outfile)

    print("Process finished --- %s seconds ---" % round(time.time() - start_time))



elif run_from_file:
    dados = open("/Users/rcvb/Documents/tcc_rian/code/assets/confirmed_cases_by_region_a
    # load confirmed_cases_by_region_and_date.json to a dataframe
    dados = json.load(dados)

    df = pd.DataFrame(dados)
    df.reset_index(inplace=True)
    df.rename(columns={'index':'collect_date'}, inplace=True)
    df.drop(df.columns[len(df.columns)-1], axis=1, inplace=True)
```

```
df.fillna(0, inplace=True)
melted_df = pd.melt(df, id_vars='collect_date', var_name='region', value_name='
melted_df['collect_date'] = pd.to_datetime(melted_df['collect_date'], errors='c
melted_df['region'] = melted_df['region'].astype(str)
melted_df['vizinhos'] = melted_df['region'].apply(lambda x: Utils.get_neighbors

melted_df['collect_date'] = melted_df['collect_date'].dt.strftime('%Y-%m-%d')
melted_df.sort_values(by=['collect_date'], inplace=True)

X = melted_df['confirmed_cases'].values
# Y target variable should be the confirmed cases 15 days after the date of the
# Y = melted_df['confirmed_cases'].shift(15).values.astype(float)
Y = melted_df['confirmed_cases'].rolling(window=15, min_periods=1).sum().values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random

# Build the model
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(1,)),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
])

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Train the model
model.fit(X_train, y_train, epochs=100000, batch_size=27, validation_data=(X_te

# Make predictions
predictions = model.predict(X_test)

# Flatten the predictions and ground truth arrays
predictions = np.squeeze(predictions)
y_test = np.squeeze(y_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, predictions)
```

```
    mae = mean_absolute_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)

    # calculate accuracy between each prediction and test data
    accuracy = 0
    for i in range(len(predictions)):
        accuracy += abs(predictions[i] - y_test[i])
    accuracy = accuracy / len(predictions)

    print("Accuracy:", round(accuracy, 2))
    print("Mean Squared Error (MSE):", round(mse, 2))
    print("Mean Absolute Error (MAE):", round(mae, 2))
    print("R-squared (R2) Score:", r2)

    print("Process finished --- %s seconds ---" % round(time.time() - start_time))

    # melted_df['region_e'] = melted_df['region'].apply(lambda x: Utils.regional_str_to_
    # melted_df['region_e'] = melted_df['region_e'].to_numpy()
    # melted_df['vizinhos_e'] = melted_df['region'].apply(lambda x: Utils.get_encoded_ne
    # melted_df['vizinhos_e'] = melted_df['vizinhos_e'].to_numpy()
    # melted_df['confirmed_cases'] = melted_df['confirmed_cases'].astype(float)

     # Assume 'melted_df' is the given DataFrame
    # Extract input (X) and output (Y) variables

elif run_from_file_improved:

    import json
    import pandas as pd
    from tensorflow import keras
    from tensorflow.keras import layers
    import numpy as np
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.metrics import mean_absolute_error, r2_score, mean_absolute_percentage_
    import numpy as np


    # Load data from JSON file
```

```
with open("/Users/rcvb/Documents/tcc_rian/code/assets/confirmed_cases_by_region
    dados = json.load(file)

# Convert data to a DataFrame
df = pd.DataFrame(dados)
df.reset_index(inplace=True)
df.rename(columns={'index': 'collect_date'}, inplace=True)
df.drop(df.columns[len(df.columns) - 1], axis=1, inplace=True)
df.fillna(0, inplace=True)

evaluation_results = []

# Melt the DataFrame to have "collect_date", "region", and "confirmed_cases" co
melted_df = pd.melt(df, id_vars='collect_date', var_name='region', value_name='
melted_df['neighbors'] = melted_df['region'].apply(lambda x: Utils.get_neighbor

# Convert collect_date to datetime
melted_df['collect_date'] = pd.to_datetime(melted_df['collect_date'], errors='c

# separate melted_df into a single DataFrame for each region
region_dfs = []
for region in melted_df['region'].unique():
    region_dfs.append(melted_df[melted_df['region'] == region])

# Sort the DataFrames by collect_date
for region_df in region_dfs:
    region_df.sort_values(by=['collect_date'], inplace=True)
    # target_cases by region
    # get first 15 rows of region_df['collect_date']['confirmed_cases'] to be X

    region_df['collect_date'] = region_df['collect_date'].apply(lambda x: x.tim

    """

    region_df['target_cases'] = region_df['confirmed_cases'].shift(15).astype(f
    region_df.dropna(subset=['target_cases'], inplace=True)
    # add region_df['target_cases'] to melted_df by region
    melted_df.loc[melted_df['region'] == region_df['region'].iloc[0], 'target_c
```

```
    # drop rows with NaN values
    melted_df.dropna(inplace=True)

    # prepare X and Y variables inside a for loop for each region
    for region in melted_df['region'].unique():
        print("-------------------------------------------------")
        print(f"Training model for region {region}")
        print("-------------------------------------------------")
        print("-------------------------------------------------")
        X = melted_df[melted_df['region'] == region]['confirmed_cases'].values
        Y = melted_df[melted_df['region'] == region]['target_cases'].values
"""
        """ X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, ran

        # Define the Keras model
        model = keras.Sequential([
            layers.Dense(64, activation='relu', input_shape=(1,)),
            layers.Dropout(0.2),
            layers.Dense(64, activation='relu'),
            layers.Dense(1)
        ])

        model.compile(optimizer='adam', loss='mse')

        # Train the model
        model.fit(X_train, y_train, epochs=1000, batch_size=63, validation_data=(X_test,

        # Make predictions
        predictions = model.predict(X_test)

        # Flatten the predictions and ground truth arrays
        predictions = np.squeeze(predictions)
        y_test = np.squeeze(y_test)

        # Calculate evaluation metrics
        scores = cross_val_score(model, X, Y, scoring='neg_mean_squared_error', cv=5)
        mse = mean_squared_error(y_test, predictions)
        mae = mean_absolute_error(y_test, predictions)
        r2 = r2_score(y_test, predictions)
```

```
    # calculate accuracy between each prediction and test data
    accuracy = 0
    for i in range(len(predictions)):
        accuracy += abs(predictions[i] - y_test[i])
    accuracy = accuracy / len(predictions)

    print("Accuracy:", round(accuracy, 2))
    print("Mean Squared Error (MSE):", round(mse, 2))
    print("Mean Absolute Error (MAE):", round(mae, 2))
    print("R-squared (R2) Score:", round(r2, 2))
    print("Cross Validation Score:", round(scores, 2))

    # initiate a variable to save results to after the loop we can compare the
    evaluation_results.append({
        'region': region,
        'accuracy': round(accuracy, 2),
        'mse': round(mse, 2),
        'mae': round(mae, 2),
        'r2': round(r2, 2),
        'cross_validation_score': round(scores, 2)
    })
print(evaluation_results)
"""
    """
    # Define the Keras model
    def create_model():
        model = keras.Sequential([
            layers.Dense(64, activation='relu', input_shape=(1,)),
            layers.Dropout(0.2),
            layers.Dense(64, activation='relu'),
            layers.Dense(1)
        ])
        model.compile(optimizer='adam', loss='mse')
        return model

    # Wrap the Keras model with the KerasRegressor
    # ignore deprecationwarning
```

```
warnings.filterwarnings("ignore", category=DeprecationWarning)
model = KerasRegressor(build_fn=create_model, epochs=100, batch_size=32, verbose

# Perform cross-validation
# kf = KFold(n_splits=5, shuffle=True, random_state=42)
# scores = cross_val_score(model, X, Y, scoring='neg_mean_squared_error', cv=kf)
cv_results = cross_val_score(model, X, Y, cv=10, scoring='r2')
scores = cross_val_score(model, X, Y, scoring='neg_mean_squared_error', cv=10)
mse_scores = -scores
rmse_scores = np.sqrt(mse_scores)
evaluation_results.append((region, rmse_scores.mean(), rmse_scores.std(), cv_res

# make predictions
predictions = cross_val_predict(model, X, Y, cv=10)
# print predictions
print("-------------------------------------------------")
print("-------------------------------------------------")
print(f"Predictions: {predictions}")
print("-------------------------------------------------")
print("-------------------------------------------------")

# calculate accuracy between each prediction and test data
accuracy = 0
for i in range(len(predictions)):
    accuracy += abs(predictions[i] - Y[i])
    accuracy = accuracy / len(predictions)

# calculate evaluation metrics
mse = mean_squared_error(Y, predictions)
mae = mean_absolute_error(Y, predictions)
r2 = r2_score(Y, predictions)
print("-------------------------------------------------")
print("--BY CROSS_VAL_PREDICT-----------------------------------")
print("Accuracy:", round(accuracy, 2))
print("Mean Squared Error (MSE):", round(mse, 2))
print("Mean Absolute Error (MAE):", round(mae, 2))
print("R-squared (R2) Score:", round(r2, 2))
print("-------------------------------------------------")
print("-------------------------------------------------")
```

```
        print("Mean Squared Error by cross_val_score: %.2f" % np.abs(scores).mean()
        print("Mean R2 Score by cross_val_score: %.2f" % np.abs(cv_results).mean())
        print("-------------------------------------------------")
        # inform errors rate
        print("-------------------------------------------------")

        # Print evaluation results
        for region, mean_score, std_score, cv_results, rmse_scores in evaluation_re
            print(f"FROM EVALUTATION_RESULTS:::: Region: {region}, Mean RMSE: {mean
        print("-------------------------------------------------")
        print("-------------------------------------------------")
        print("-------------------------------------------------")

        # implement a grid search to find the best parameters
        param_grid = {
            'epochs': [50, 100, 200, 300, 500, 1000],
            'batch_size': [7, 16, 21, 32, 42, 64, 128]
        }
        grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring='r2', c
        grid_result = grid.fit(X, Y)
        means = grid_result.cv_results_['mean_test_score']
        stds = grid_result.cv_results_['std_test_score']
        params = grid_result.cv_results_['params']
        r2_results = grid_result.cv_results_['mean_test_score']

        print("R2 Scores:")
        for r2 in r2_results:
            print("%.3f" % r2)

        print(f"Best: {grid_result.best_score_} using {grid_result.best_params_}")

        for mean, std, param, r2_results in zip(means, stds, params, r2_results):
            print(f"GRID SEARCH::::  Mean: {mean}, Std: {std}, Params: {param}, R2
        print("-------------------------------------------------")
        print("-------------------------------------------------")
        print("-------------------------------------------------")
"""
```

```
            # Collect the X variable
        X = []
        for i in range(15):
            X.append(region_df.iloc[i:i+15][['collect_date', 'confirmed_cases']].values.

        # Collect the Y variable
        Y = region_df.iloc[15:][['collect_date', 'confirmed_cases']].values.flatten()

        # X and Y 'collect_date' values are in datetime format, so we need to convert th

        # Train the model
        model = RandomForestRegressor()
        model.fit(X, Y)

        # Predict future confirmed cases
        future_X = region_df.iloc[15:30][['collect_date', 'confirmed_cases']].values.fla
        future_Y = model.predict(future_X)

        # Define evaluation metrics
        def MDAPE(y_true, y_pred):
            return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

        def Accuracy(y_true, y_pred):
            return 100 - MDAPE(y_true, y_pred)

        # Evaluate the model
        print('Mean Absolute Error:', mean_absolute_error(region_df.iloc[15:30]['confirm
        print('R-Squared:', r2_score(region_df.iloc[15:30]['confirmed_cases'], future_Y)
        print('MDAPE:', MDAPE(region_df.iloc[15:30]['confirmed_cases'], future_Y))
        print('RMSE:', mean_squared_error(region_df.iloc[15:30]['confirmed_cases'], futu
        print('Accuracy:', Accuracy(region_df.iloc[15:30]['confirmed_cases'], future_Y))
        print('MAPE: ', mean_absolute_percentage_error(region_df.iloc[15:30]['confirmed_
    print("Process finished --- %s seconds ---" % round(time.time() - start_time))
    print("-------------------------------------------------")

elif mount_whole_csv_encoded:
    removeSpace = lambda x: (x.replace('INTERNADO UTI',''))
    ignoradoToEmpty = lambda x: x.replace('IGNORADO','')
```

```
df = pd.read_csv(f"{home}/assets/boavista_covid_dados_abertos.csv", sep=';',
true_values=["SIM", "CONFIRMADO", "INTERNADO"], low_memory=False,
false_values=["NAO", "NAO INTERNADO"],
error_bad_lines=False, verbose=True, converters={'internacao_uti':removeSpace,

df.drop(["data_publicacao", "sintomas", "comorbidades", "gestante", "municipio"
"codigo_ibge_municipio", "estado", "criterio_confirmacao", "municipio_notificac
"latitude_notificacao", "longitude_notificacao", "classificacao", "nom_laborato
"regional_saude", "data_evolucao_caso", "data_saida_uti", "bairro"], axis=1, in
df.dropna(inplace=True)
df['data_inicio_sintomas'] = pd.to_datetime(df['data_inicio_sintomas'], errors=
df['collect_date'] = pd.to_datetime(df['collect_date'], errors='coerce')
df['data_resultado'] = pd.to_datetime(df['data_resultado'], errors='coerce')
df['internacao_uti'].mask(df['internacao_uti'] == "NAO ", False, inplace=True)
df['internacao_uti'].mask(df['internacao_uti'] == "", False, inplace=True)
df['internacao_uti'] = df['internacao_uti'].astype(float)
df['sexo'].mask(df['sexo'] == "FEMININO", 0, inplace=True)
df['sexo'].mask(df['sexo'] == "MASCULINO", 1, inplace=True)
df['sexo'].mask(df['sexo'] == "NAO INFORMADO", 0, inplace=True)
df['sexo'] = df['sexo'].astype(float)
df['recuperados'].mask(df['recuperados'] == True, 1, inplace=True)
df['recuperados'].mask(df['recuperados'] == False, 0, inplace=True)
df['recuperados'] = df['recuperados'].astype(float)
df['internacao'].mask(df['internacao'] == True, 1, inplace=True)
df['internacao'].mask(df['internacao'] == False, 0, inplace=True)
df['internacao'] = df['internacao'].astype(float)
df['internacao_uti'].mask(df['internacao_uti'] == True, 1, inplace=True)
df['internacao_uti'].mask(df['internacao_uti'] == False, 0, inplace=True)
df['obito'].mask(df['obito'] == True, 1, inplace=True)
df['obito'].mask(df['obito'] == False, 0, inplace=True)
df['obito'] = df['obito'].astype(float)
df['origem_esus'].mask(df['origem_esus'] == True, 1, inplace=True)
df['origem_esus'].mask(df['origem_esus'] == False, 0, inplace=True)
df['origem_esus'] = df['origem_esus'].astype(float)
df['origem_sivep'].mask(df['origem_sivep'] == True, 1, inplace=True)
df['origem_sivep'].mask(df['origem_sivep'] == False, 0, inplace=True)
df['origem_sivep'] = df['origem_sivep'].astype(float)
df['origem_lacen'].mask(df['origem_lacen'] == True, 1, inplace=True)
```

```
    df['origem_lacen'].mask(df['origem_lacen'] == False, 0, inplace=True)
    df['origem_lacen'] = df['origem_lacen'].astype(float)
    df['origem_laboratorio_privado'].mask(df['origem_laboratorio_privado'] == True, 1, i
    df['origem_laboratorio_privado'].mask(df['origem_laboratorio_privado'] == False, 0,
    df['origem_laboratorio_privado'] = df['origem_laboratorio_privado'].astype(float)
    df['fez_teste_rapido'].mask(df['fez_teste_rapido'] == True, 1, inplace=True)
    df['fez_teste_rapido'].mask(df['fez_teste_rapido'] == False, 0, inplace=True)
    df['fez_teste_rapido'] = df['fez_teste_rapido'].astype(float)
    df['fez_pcr'].mask(df['fez_pcr'] == True, 1, inplace=True)
    df['fez_pcr'].mask(df['fez_pcr'] == False, 0, inplace=True)
    df['fez_pcr'] = df['fez_pcr'].astype(float)
    df['regional_e'] = df['regional'].apply(lambda x: Utils.regional_str_to_encoded(x))
    df['regional_e'] = df['regional_e'].to_numpy()
    df['tipo_teste_e'] = df['tipo_teste'].apply(lambda x: Utils.tipoteste_str_to_encoded
    df['tipo_teste_e'] = df['tipo_teste_e'].to_numpy()

    dados = open(f"{home}/assets/confirmed_cases_by_region_and_date.json")
    data = json.load(dados)


    df['vizinhos_e'] = df['vizinhos'].apply(lambda x: Utils.get_encoded_neighbors_of_reg

    df.drop(['recuperados', 'data_inicio_sintomas', 'internacao', 'internacao_uti', 'sex
       'latitude', 'longitude', 'tipo_teste', 'origem_esus', 'origem_sivep', 'origem_lac
       'fez_pcr', 'tipo_teste_e'], axis=1, inplace=True)


    profile = ProfileReport(df, explorative=True, minimal=False)
    try:
        profile.to_widgets()         # view as widget in Notebook
    except:
        profile.to_file('df.html')

    print("Process finished --- %s seconds ---" % round(time.time() - start_time))

else:

    # enconded_REGIONS_SEQ_AND_ITS_NEIGHBORS = Utils.transform_categorical_to_one_hot_en
    encoded_regions = Utils.transform_categorical_to_one_hot_encoding(constants.REGIONS_
```

```
    # formated_regions_and_neighbors = (enconded_REGIONS_SEQ_AND_ITS_NEIGHBORS[0],
    #    enconded_REGIONS_SEQ_AND_ITS_NEIGHBORS[1], [enconded_REGIONS_SEQ_AND_ITS_N
    #    enconded_REGIONS_SEQ_AND_ITS_NEIGHBORS[2], [enconded_REGIONS_SEQ_AND_ITS_N
    #    enconded_REGIONS_SEQ_AND_ITS_NEIGHBORS[3], [enconded_REGIONS_SEQ_AND_ITS_N
    #    enconded_REGIONS_SEQ_AND_ITS_NEIGHBORS[4], [enconded_REGIONS_SEQ_AND_ITS_N
    #    enconded_REGIONS_SEQ_AND_ITS_NEIGHBORS[5], [enconded_REGIONS_SEQ_AND_ITS_N
    #    enconded_REGIONS_SEQ_AND_ITS_NEIGHBORS[6], [enconded_REGIONS_SEQ_AND_ITS_N

    # G = nx.from_pandas_edgelist(df, source='regional', target='obito', edge_attr=
    profile = ProfileReport(df, explorative=True, minimal=False)
    try:
        profile.to_widgets()         # view as widget in Notebook
    except:
        profile.to_file('df.html')

    print("Process finished --- %s seconds ---" % round(time.time() - start_time))


# link repositorio: https://github.com/RianBrug/gcn_for_covid_sc_brazil
# gcn_network file

import json
import sklearn
import torch
import networkx as nx
import pandas as pd
import torch_geometric
from torch_geometric.utils import from_networkx
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, me
from sklearn.model_selection import TimeSeriesSplit, train_test_split
from torch_geometric.nn import GCNConv
from torch.nn import L1Loss
import torch.nn.functional as F
from torch.optim import Adam
import torch.multiprocessing as mp
import numpy as np
import optuna
```

```
import sqlite3
import optuna.visualization as vis
from optuna.pruners import SuccessiveHalvingPruner
from torch.optim.lr_scheduler import ReduceLROnPlateau

import matplotlib.pyplot as plt
from earlyStopping import EarlyStopping
from utils import Utils

# Configuration
HOME = '/Users/rcvb/Documents/tcc_rian/code'
DATABASE_URL = 'sqlite:///gcn_newest'
STUDY_NAME = 'gcn_newest_7d'
WINDOW = 7
TOTAL_EPOCHS = 160
TRIALS_UNTIL_START_PRUNING = 150
N_TRIALS = 300
N_JOBS = 5
NUM_ORIGINAL_FEATURES = 7
NUM_ADDITIONAL_FEATURES = 1
PATIENCE_LEARNING_SCHEDULER = 100
NUM_SPLITS = 3
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Define the Neural Network Model
class Net(torch.nn.Module):
    def __init__(self, NUM_ORIGINAL_FEATURES, NUM_ADDITIONAL_FEATURES, num_hidden_channe
        super(Net, self).__init__()
        self.layers = torch.nn.ModuleList()
        self.layers.append(GCNConv(NUM_ORIGINAL_FEATURES + NUM_ADDITIONAL_FEATURES, num_
        for _ in range(num_layers - 2): # -2 to account for the first and last layers
            self.layers.append(GCNConv(num_hidden_channels, num_hidden_channels))
        self.layers.append(GCNConv(num_hidden_channels, NUM_ORIGINAL_FEATURES))  # outpu
        self.dropout_rate = dropout_rate

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        for conv in self.layers[:-1]:
            x = conv(x, edge_index)
```

```
            x = F.relu(x)
            if self.training:  # only apply dropout during training
                x = F.dropout(x, p=self.dropout_rate, training=self.training)
        x = self.layers[-1](x, edge_index)  # Don't apply relu or dropout to the la
        return x


# load and return the dataframe
def load_data():
    HOME = '/Users/rcvb/Documents/tcc_rian/code'
    with open(f'{HOME}/assets/confirmed_cases_by_region_and_date.json') as file:
        data = json.load(file)

    df = pd.DataFrame(data)
    df.reset_index(inplace=True)
    df.rename(columns={'index':'collect_date'}, inplace=True)
    df['collect_date'] = pd.to_datetime(df['collect_date'])
    df.sort_values(by=['collect_date'], inplace=True)
    df.drop(df.columns[len(df.columns)-1], axis=1, inplace=True)
    df.fillna(0, inplace=True)
    return df

def sliding_windows(data, window):
    X = []
    Y = []

    for i in range(len(data)-2*window):
        X.append(data.iloc[i:i+window].values)
        Y.append(data.iloc[i+window:i+2*window].values)

    return np.array(X), np.array(Y)

# Function to convert data into graphs
def data_to_graph(df, window, train_indices, val_indices):
    G = nx.Graph()
    train_mask = []
    val_mask = []

    # Load your additional features data here. For example:
```

```
    pr_df = pd.read_csv(f'{HOME}/assets/populacao_residente_sc_por_macroregiao.csv', sep


    for region in df.columns[1:]:
        region_df = df[['collect_date', region]].dropna()
        X, Y = sliding_windows(region_df[region], window)

        # Retrieve additional features for the current region
        add_features = np.array([
            pr_df.loc[region],
        ]).flatten()

        for i in range(len(X)):
            # Concatenate original features with additional features
            features = np.concatenate([X[i], add_features]).astype(np.float32)
            G.add_node((region, i), x=torch.tensor(features), y=torch.tensor(Y[i]).float
            for neighbor in Utils.get_neighbors_of_region(region):
                if (neighbor, i) in G.nodes:
                    G.add_edge((region, i), (neighbor, i))

            if i in train_indices:
                train_mask.append(True)
            else:
                train_mask.append(False)

            if i in val_indices:
                val_mask.append(True)
            else:
                val_mask.append(False)

    data = from_networkx(G)
    data.train_mask = torch.tensor(train_mask)
    data.val_mask = torch.tensor(val_mask)

    return data

# Objective function for Optuna
def objective(trial):
    df = load_data()
```

```
tscv = TimeSeriesSplit(n_splits=3)
dropout_rate = trial.suggest_float("dropout_rate", 0.1, 0.5)
lr = trial.suggest_float("lr", 1e-4, 1e-1,log=True)
num_hidden_channels = trial.suggest_categorical("num_hidden_channels", [16, 32,
num_layers = trial.suggest_categorical("num_layers", [6, 9, 12, 15, 18, 21])
#num_hidden_channels = trial.suggest_categorical("num_hidden_channels", [5, 15,
#num_layers = trial.suggest_categorical("num_layers", [5, 10, 15, 25, 40, 50])
weight_decay = trial.suggest_float("weight_decay", 1e-10, 1e-3)  # L2 regulariz

results = []
true_values = []
predictions = []

for fold, (train_index, val_index) in enumerate(tscv.split(np.arange(WINDOW, df
    data = data_to_graph(df, WINDOW, train_index, val_index)
    model = Net(NUM_ORIGINAL_FEATURES, NUM_ADDITIONAL_FEATURES, num_hidden_chan
    data = data.to(DEVICE)
    optimizer = Adam(model.parameters(), lr=lr, weight_decay=weight_decay)  # A
    # optimizer = RMSprop(model.parameters(), lr=lr, weight_decay=weight_decay)
    scheduler = ReduceLROnPlateau(optimizer, 'min', patience=PATIENCE_LEARNING_
    criterion = L1Loss()
    model.train()
    fold_losses = []  # Average loss for each epoch within this fold
    val_losses = []  # Validation loss for each epoch within this fold
    for epoch in range(TOTAL_EPOCHS):
        optimizer.zero_grad()
        out = model(data)
        loss = criterion(out[data.train_mask], data.y[data.train_mask])
        loss.backward()
        optimizer.step()
        model.eval()
        fold_losses.append(loss.item())

        with torch.no_grad():
            pred = model(data)
            val_loss = criterion(pred[data.val_mask], data.y[data.val_mask])
            val_losses.append(val_loss.item())
            true_values = data.y[data.val_mask].cpu().detach().numpy().tolist()
            predictions = pred[data.val_mask].cpu().detach().numpy().tolist()
```

```
            scheduler.step(val_loss)

        avg_fold_loss = sum(fold_losses) / len(fold_losses)

        if trial.number > TRIALS_UNTIL_START_PRUNING:
            # Pass the average fold loss to the pruner
            unique_epoch = fold * TOTAL_EPOCHS + epoch
            trial.report(avg_fold_loss, unique_epoch)

            # Handle pruning based on the intermediate value
            if trial.should_prune():
                raise optuna.exceptions.TrialPruned()

        if np.isnan(data.y[data.val_mask].cpu().detach().numpy()).any():
            print("NaN value detected in target data.")
            return np.inf  # Optuna will minimize this value

        # Check for NaN values in prediction
        if np.isnan(pred[data.val_mask].cpu().detach().numpy()).any():
            print("NaN value detected in prediction.")
            return np.inf  # Optuna will minimize this value

        mae = mean_absolute_error(data.y[data.val_mask].cpu().detach().numpy(), pred[dat
        mape = mean_absolute_percentage_error(data.y[data.val_mask].cpu().detach().numpy
        mse = mean_squared_error(data.y[data.val_mask].cpu().detach().numpy(), pred[data
        rmse = np.sqrt(mse)
        r2 = r2_score(data.y[data.val_mask].cpu().detach().numpy(), pred[data.val_mask].
        mdape = Utils.MDAPE(data.y[data.val_mask].cpu().detach().numpy(), pred[data.val_

        results.append((mae, mape, mse, rmse, r2, mdape, val_losses[-33:]))

    avg_mae = np.mean([res[0] for res in results])
    avg_mape = np.mean([res[1] for res in results])
    avg_mse = np.mean([res[2] for res in results])
    avg_rmse = np.mean([res[3] for res in results])
    avg_r2 = np.mean([res[4] for res in results])
    avg_mdape = np.mean([res[5] for res in results])
    avg_val_losses = np.mean([res[6] for res in results])
```

```
    trial.set_user_attr("avg_mae", float(avg_mae))
    trial.set_user_attr("avg_mape", float(avg_mape))
    trial.set_user_attr("avg_mse", float(avg_mse))
    trial.set_user_attr("avg_rmse", float(avg_rmse))
    trial.set_user_attr("avg_r2", float(avg_r2))
    trial.set_user_attr("avg_mdape", float(avg_mdape))
    trial.set_user_attr("avg_val_losses", float(avg_val_losses))
    trial.set_user_attr("true_values", true_values)
    trial.set_user_attr("predictions", predictions)

    return avg_mae  # Optuna will minimize this value

def plot_results(true_values, predictions):
    plt.figure(figsize=(12, 8))
    plt.plot(true_values[:, 0], label='True values')
    plt.plot(predictions[:, 0], label='Predictions')
    plt.xlabel('Time Steps')
    plt.ylabel('Confirmed Cases')
    plt.legend()
    plt.show()

# Main part of the code
if __name__ == '__main__':
    # Enable parallel processing
    mp.set_start_method('spawn')

    df = load_data()

    # Split data into training and testing sets to prevent data leakage
    train_df, test_df = train_test_split(df, test_size=0.2, shuffle=False)


    # Optuna study for hyperparameter tuning
    # Start Optuna study
    pruner = SuccessiveHalvingPruner()
    study = optuna.create_study(study_name=STUDY_NAME, storage=DATABASE_URL, load_i
    study.optimize(objective, n_trials=N_TRIALS, n_jobs=N_JOBS, show_progress_bar=T
    vis.plot_optimization_history(study)
```

```
    vis.plot_intermediate_values(study)
    vis.plot_parallel_coordinate(study)
    vis.plot_slice(study)
    vis.plot_param_importances(study)
    vis.plot_edf(study)
    vis.plot_contour(study)

    # Plot results
    best_trial = study.best_trial
    best_true_values = np.array(best_trial.user_attrs["true_values"])
    best_predictions = np.array(best_trial.user_attrs["predictions"])
    plot_results(best_true_values, best_predictions)
```