



Universidade Estadual Paulista “Júlio de Mesquita Filho”
Instituto de Biociências – Câmpus de Botucatu
Programa de Pós-graduação em Biometria



Avaliações Preditivas de Modelos Temporais com Integração de Informações Espaciais: Um Estudo de Incidência da Covid-19 no Estado de São Paulo em 2020

Thiago Moraes Rizzieri

Botucatu
2024

Thiago Moraes Rizzieri

Avaliações Preditivas de Modelos Temporais com Integração de Informações Espaciais: Um Estudo de Incidência da Covid-19 no Estado de São Paulo em 2020

Dissertação de Mestrado apresentada ao Curso de Programa de Pós-graduação em Biometria da Universidade Estadual Paulista “Júlio de Mesquita Filho” como parte dos requisitos necessários para a obtenção do título de Mestre em Biometria.

Orientador: Prof. Dr. José Silvio Govone

Botucatu
2024

R627a

Rizzieri, Thiago Moraes

Avaliações preditivas de modelos temporais com integração de informações espaciais : Um estudo de incidência da Covid-19 no estado de São Paulo em 2020 / Thiago Moraes Rizzieri. -- Botucatu, 2025

115 p. : tabs., mapas

Dissertação (mestrado) - Universidade Estadual Paulista (UNESP), Instituto de Biociências, Botucatu

Orientador: José Silvio Govone

1. COVID-19 (Doença). 2. Analise de series temporais. 3. Inferência espacial. I. Título.

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: Avaliações Preditivas de Modelos Temporais com Integração de Informações Espaciais: Um Estudo de Incidência da Covid-19 no Estado de São Paulo em 2020

AUTOR: THIAGO MORAES RIZZIERI

ORIENTADOR: JOSÉ SILVIO GOVONE

Aprovado como parte das exigências para obtenção do Título de Mestre em Biometria, pela Comissão Examinadora:

Prof. Dr. JOSÉ SILVIO GOVONE (Participação Presencial)
Departamento de Estatística, Matemática Aplicada e Computação / Instituto de Geociências e Ciências Exatas (IGCE) - Câmpus de Rio Claro

Prof. Dr. JOÃO DOMINGOS SCALON (Participação Virtual)
Departamento de Estatística / Instituto de Ciências Exatas e Tecnológicas (ICET), da Universidade Federal de Lavras (UFLA)

Prof.^a Dr.^a MIRIAM HARUMI TSUNEMI (Participação Presencial)
Departamento de Biodiversidade e Bioestatística / Instituto de Biociências de Botucatu - UNESP

Botucatu, 21 de fevereiro de 2025



Documento assinado digitalmente

FLÁVIA DANIELI MARTINS GODINHO

Data: 24/02/2025 14:46:35-0300

Verifique em <https://validar.iti.gov.br>

FLÁVIA DANIELI MARTINS GODINHO
Assistente Técnico Administrativo I da Seção Técnica de Pós-Graduação
Instituto de Biociências de Botucatu – UNESP

Instituto de Biociências - Câmpus de Botucatu -
Rua Doutor Antonio Celso Wagner Zanin, s/nº, 18618970
<https://www.ibb.unesp.br/#/ensino/pos-graduacao/programas-stricto-sensu/pg-biometria/CNPJ:48031918002259>.

Agradecimentos

Gostaria de expressar minha profunda gratidão às pessoas e que contribuíram para a realização deste trabalho.

Agradeço primeiramente ao meu orientador José Silvio Govone, pelo apoio, orientação paciente e pelas valiosas sugestões durante todas as reuniões, não somente para o desenvolvimento do projeto, mas também grandes conselhos de vida.

Aos meus amigos e familiares, pelo incentivo, compreensão e apoio emocional, que foram essenciais para superar os desafios ao longo do percurso.

Agradeço também à CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo suporte financeiro durante minha formação, que possibilitou a execução deste projeto.

Aos colegas e professores do Departamento de Biodiversidade e Bioestatística, pelo ambiente acadêmico colaborativo e pelas excelentes aulas, que me ajudaram muito neste percurso.

Por fim, expresso minha gratidão a todos que, direta ou indiretamente, contribuíram para a concretização deste trabalho.

Resumo

A disseminação global do COVID-19 representou um desafio significativo, dada sua natureza como uma infecção respiratória aguda altamente contagiosa e potencialmente grave. Em resposta a essa emergência, o Comitê de Emergência da Organização Mundial da Saúde (OMS) declarou estado de emergência global em 30 de janeiro de 2020. A complexidade da situação e o surgimento contínuo de novos modelos estatísticos destacaram a importância de abordagens preditivas para avaliar a evolução da pandemia. Este estudo apresenta uma análise espaço-temporal dos dados de COVID-19 no estado de São Paulo em 2020, fornecidos pelo SEADE. Para a análise temporal, foram utilizados os modelos ARIMA, Floresta Aleatória e Redes Neurais LSTM além do método simples para comparação, o modelo de média. Esses modelos foram aplicados para prever a média móvel de casos de COVID-19. Além disso, este estudo compara dois modelos espaciais: o modelo de Autoregressão Espacial (SAR) e o modelo baseado em redes neurais para dados de grafos, Redes Neurais de Grafos (GNN), para modelar a defasagem espacial. A análise espacial foi realizada para entender melhor a distribuição geográfica dos casos de COVID-19 e para verificar se a integração das informações espaciais nos modelos temporais pode melhorar o desempenho das previsões. Os resultados indicam que a consideração da dependência espacial pode aprimorar significativamente as previsões temporais, oferecendo uma compreensão mais completa da dinâmica de transmissão do vírus e contribuindo para a tomada de decisões mais informadas no combate à pandemia.

Palavras-chave: COVID-19; Previsão Temporal; Estatística Espacial; ARIMA; Floresta Aleatória; Redes Neurais; São Paulo.

Abstract

The global spread of COVID-19 posed a significant challenge due to its nature as a highly contagious and potentially severe acute respiratory infection. In response to this emergency, the World Health Organization (WHO) Emergency Committee declared a global emergency on January 30, 2020. The complexity of the situation and the continuous emergence of new statistical models highlighted the importance of predictive approaches to assess the pandemic's evolution. This study presents a spatio-temporal analysis of COVID-19 data in the state of São Paulo in 2020, provided by SEADE. For the temporal analysis, ARIMA, Random Forest, and LSTM Neural Network, were used, along with simple methods for comparison such as Naive and Mean. These models were applied to predict the weekly moving average of COVID-19 cases. Additionally, this study compares two spatial models: the Spatial Autoregressive (SAR) model and the Graph Convolution Network (GCN) model, to model spatial lag. The spatial analysis was conducted to better understand the geographical distribution of COVID-19 cases and to verify if integrating spatial information into temporal models can improve forecast performance. The results indicate that considering spatial dependence can significantly enhance temporal forecasts, providing a more comprehensive understanding of the virus's transmission dynamics and contributing to more informed decision-making in combating the pandemic.

Keywords: COVID-19; Temporal Forecasting; Spatial Statistics; ARIMA; Random Forest; Neural Networks; São Paulo.

Lista de figuras

Figura 1 – Exemplo de uma série temporal da produção de cerveja na Austrália. Fonte: (Hyndman; Athanasopoulos, 2018).	13
Figura 2 – Exemplo de correlograma da produção de cerveja na Austrália. Fonte: (Hyndman; Athanasopoulos, 2018).	14
Figura 3 – Exemplo visual de uma série temporal estacionária comparada com duas séries temporais não estacionárias. Fonte: (Sahin, 2020).	16
Figura 4 – Ilustração de como o Autoarima é feito. Neste exemplo, foram 4 etapas no total, resultando em $ARIMA(4, d, 2)$ ser a melhor escolha. Fonte: (Hyndman; Athanasopoulos, 2018).	25
Figura 5 – Arquitetura básica de uma RNN. Fonte: (Kalita, 2022).	31
Figura 6 – Arquitetura básica de uma rede neural LSTM. Fonte: (GeeksforGeeks, 2023).	35
Figura 7 – Grafo dos municípios de São Paulo conectado por suas respectivas fronteiras. Fonte: Elaborado pelo autor.	42
Figura 8 – Exemplo visual de uma arquitetura de uma rede GNN. Fonte: (Summer, 2023).	42
Figura 9 – Previsão recursiva: os valores previstos são usados como entrada nas etapas seguintes. Fonte: (Rodrigo; Ortiz, 2024).	56
Figura 10 – Taxa de incidência semanal original.	58
Figura 11 – Comparação das taxas de incidência semanal e dos ajustes obtidos pelos modelos SAR e GNN após 100 dias de transmissão.	58
Figura 12 – Série temporal da incidência semanal de COVID-19 em São Paulo. Fonte: Elaborado pelo autor.	62
Figura 13 – Previsão da incidência em São Paulo. Fonte: Elaborado pelo autor.	64
Figura 14 – Série temporal da incidência semanal de COVID-19 em Guarulhos. Fonte: Elaborado pelo autor.	64
Figura 15 – Previsão da incidência em Guarulhos. Fonte: Elaborado pelo autor.	66
Figura 16 – Série temporal da incidência semanal de COVID-19 em Campinas. Fonte: Elaborado pelo autor.	66
Figura 17 – Previsão da incidência em Campinas. Fonte: Elaborado pelo autor.	68
Figura 18 – Série temporal da incidência semanal de COVID-19 em São Bernardo do Campo. Fonte: Elaborado pelo autor.	68
Figura 19 – Previsão da incidência em São Bernardo do Campo. Fonte: Elaborado pelo autor.	70
Figura 20 – Previsões do melhor modelo para a cidade de Santo André. Fonte: Elaborado pelo autor.	70
Figura 21 – Previsão da incidência em Santo André. Fonte: Elaborado pelo autor.	72

Figura 22 – Previsões do melhor modelo para a cidade de Sorocaba. Fonte: Elaborado pelo autor.	72
Figura 23 – Previsão da incidência em Sorocaba. Fonte: Elaborado pelo autor.	74
Figura 24 – Previsões do melhor modelo para a cidade de Osasco.	74
Figura 25 – Previsão da incidência em Osasco. Fonte: Elaborado pelo autor.	76
Figura 26 – Previsões do melhor modelo para a cidade de Ribeirão Preto. Fonte: Elaborado pelo autor.	76
Figura 27 – Previsão da incidência em Ribeirão Preto. Fonte: Elaborado pelo autor.	78
Figura 28 – Previsões do melhor modelo para a cidade de São José dos Campos. Fonte: Elaborado pelo autor.	78
Figura 29 – Previsão da incidência em São José dos Campos. Fonte: Elaborado pelo autor.	80
Figura 30 – Previsões do melhor modelo para a cidade de São José do Rio Preto. Fonte: Elaborado pelo autor.	80
Figura 31 – Previsão da incidência em São José do Rio Preto. Fonte: Elaborado pelo autor.	82

Lista de tabelas

Tabela 1 – Desempenho dos modelos no horizonte de curto prazo (previsão de 7 dias).	59
Tabela 2 – Desempenho dos modelos no horizonte de médio prazo (previsão de 1 mês).	59
Tabela 3 – Desempenho dos modelos no horizonte de longo prazo (previsão de 3 meses).	60
Tabela 4 – Estatísticas descritivas da incidência semanal de COVID-19 em São Paulo. .	63
Tabela 5 – Estatísticas descritivas da incidência semanal de COVID-19 em Guarulhos. .	65
Tabela 6 – Estatísticas descritivas da incidência semanal de COVID-19 em Campinas. .	67
Tabela 7 – Estatísticas descritivas da incidência semanal de COVID-19 em São Bernardo do Campo.	69
Tabela 8 – Estatísticas descritivas da incidência semanal de COVID-19 em Santo André.	71
Tabela 9 – Estatísticas descritivas da incidência semanal de COVID-19 em Sorocaba. .	73
Tabela 10 – Estatísticas descritivas da incidência semanal de COVID-19 em Osasco. . .	75
Tabela 11 – Estatísticas descritivas da incidência semanal de COVID-19 em Ribeirão Preto.	77
Tabela 12 – Estatísticas descritivas da incidência semanal de COVID-19 em São José dos Campos.	79
Tabela 13 – Estatísticas descritivas da incidência semanal de COVID-19 em São José do Rio Preto.	81

Sumário

1	INTRODUÇÃO	9
1.1	Objetivos	10
2	REVISÃO DE LITERATURA	12
2.1	Série Temporal	12
2.2	Modelos de Séries Temporais	18
2.2.1	Modelo ARIMA	18
2.2.2	Aprendizado de Máquina	27
2.2.3	Floresta Aleatória	28
2.2.4	Redes Neurais	30
2.3	Análise Espacial	36
2.3.1	Modelo de Autoregressão Espacial Simultâneo (SAR)	39
2.3.2	Redes Neurais de Grafos (GNN)	41
2.4	Agregando Informações Espaciais aos Modelos Temporais	46
2.4.1	Modelo ARIMAX	46
2.4.2	Regressão Dinâmica	47
2.4.3	Arquitetura GNN+LSTM	48
2.5	Bootstrapping aplicado a séries temporais	51
2.6	Métricas de desempenho dos modelos	52
3	MATERIAIS E MÉTODOS	54
3.1	Material	54
3.2	Métodos	55
4	RESULTADOS E DISCUSSÕES	57
4.1	Análise dos Modelos	57
4.2	Análise Residual das 10 Cidades Mais Populosas	62
5	CONCLUSÃO	83
	Referências	85
	Apêndices	89
	APÊNDICE A – CÓDIGO DOS MODELOS EM PYTHON	90

1 Introdução

A pandemia de COVID-19, causada pelo vírus SARS-CoV-2, foi um evento global sem precedentes, que desafiou sistemas de saúde, economias e sociedades em todo o mundo, representando um dos maiores desafios de saúde pública dos últimos tempos ([World Health Organization, 2021](#)). Desde que surgiu em dezembro de 2019, em Wuhan, na China, a COVID-19 se espalhou rapidamente, resultando em milhões de mortes e profundas consequências econômicas e sociais. A natureza altamente contagiosa e potencialmente grave da infecção levou a uma necessidade urgente de monitoramento e previsão de sua evolução para apoiar decisões de políticas de saúde pública.

O controle de pandemias, como a COVID-19, depende significativamente da capacidade de prever com precisão a evolução da disseminação do vírus e de alocar adequadamente os recursos de saúde. A previsão de novos casos, hospitalizações e óbitos desempenha um papel crucial no planejamento de políticas públicas e na tomada de decisões baseadas em evidências. A eficácia dessas previsões pode impactar diretamente a capacidade de implementar medidas de contenção, como *lockdowns* e distanciamento social, e de gerenciar os sistemas de saúde para evitar colapsos ([Fauci; Lane; Redfield, 2020](#)).

Apesar dos avanços na vacinação, a antecipação permanece uma ferramenta crucial no controle de pandemia e no planejamento de políticas futuras, como já foi amplamente observado no início da pandemia, onde até pequenas melhorias na previsão de surtos epidemiológicos levaram a intervenções mais eficazes ([Li, 2020](#)). Com o crescente risco de futuras pandemias, a capacidade de prever a evolução da transmissão viral, tanto em termos espaciais quanto temporais, continua a ser de extrema importância.

Desde o seu surto no final de 2019, o vírus COVID-19 devastou o mundo, causando mais de 3 milhões de mortes ([World Health Organization, 2021](#)). Crucial para conter tais danos é uma resposta rápida e uma formulação de políticas eficaz, mesmo com o aumento dos números de vacinação. Devido à natureza exponencial da transmissão epidemiológica ([Li, 2020](#)), melhorias na intervenção precoce podem ter impactos desproporcionais, tornando a preemptividade uma ferramenta crítica.

Dada a natureza altamente contagiosa do SARS-CoV-2 e o padrão de transmissão geograficamente variado, é necessário o uso de técnicas que integrem tanto as dimensões temporais

quanto espaciais da pandemia. Desde o início da pandemia, diversos modelos matemáticos e de aprendizado de máquina foram aplicados para prever a incidência de COVID-19, entre os quais os modelos temporais, como ARIMA (AutoRegressive Integrated Moving Average), Floresta Aleatória e Redes Neurais, como a LSTM (Long Short-Term Memory), têm sido amplamente utilizados para analisar séries temporais epidemiológicas (Velavan; Meyer, 2020). Esses métodos têm se mostrado eficazes ao lidar com a natureza não linear e complexa dos dados temporais da pandemia.

Além dos modelos temporais, a análise espacial desempenha um papel fundamental na compreensão da dinâmica geográfica da COVID-19. Modelos espaciais, como o modelo de Autoregressão Espacial (SAR) e redes neurais de grafos, como as Redes Neurais de Grafos (GNN), permitem capturar as relações de dependência espacial entre diferentes regiões, revelando padrões que são cruciais para entender a transmissão do vírus. A análise espacial é particularmente útil quando há uma forte contiguidade geográfica nos dados, ou seja, quando a propagação do vírus em uma área é influenciada pelas áreas vizinhas.

Este trabalho baseia-se, em parte, no estudo de (Panagopoulos; Nikolentzos; Vazirgiannis, 2021a) sobre comparação de modelos de previsão temporais e espaciais na previsão da COVID-19. Há uma comparação entre modelos temporais com o modelo espaço-temporal, mas não há uma comparação entre os modelos de dependência espacial.

Este estudo visa combinar abordagens temporais e espaciais para fornecer previsões da incidência de COVID-19 no estado de São Paulo em 2020. O foco está em integrar informações espaciais nos modelos temporais, avaliando se o impacto da inclusão da dependência espacial melhora o desempenho das previsões. Além disso, modelos de dependência espacial clássicos, como o SAR, e modelos mais recentes baseados em redes neurais, como o GNN, serão comparados com foco na eficiência da previsão da evolução da pandemia.

1.1 Objetivos

O presente trabalho tem como objetivos: análise temporal dos dados de COVID-19 de cada cidade do estado de São Paulo, comparação entre modelos de dependência espacial clássicos e baseados em redes neurais, bem como a integração das informações espaciais em modelos de séries temporais. Especificamente, os objetivos incluem:

- Realizar uma análise temporal dos dados de COVID-19 no estado de São Paulo utilizando os modelos ARIMA, Floresta Aleatória, e Redes Neurais, além de métodos simples de comparação (Naive e Média).
- Preparar os dados espaciais, incluindo limpeza e visualização por meio de mapas.
- Realizar uma análise espacial dos dados de COVID-19 utilizando os modelos SAR e GNN para capturar a dependência espacial.

- Integrar as informações espaciais nos modelos temporais para verificar se há melhora no desempenho das previsões.
- Comparar os resultados obtidos dos diferentes modelos e discutir as implicações para a gestão da pandemia.

2 Revisão de Literatura

Este capítulo apresenta uma visão geral das abordagens e métodos relevantes para a análise e previsão de séries temporais e dependências espaciais no contexto epidemiológico. A revisão está estruturada em três partes principais. Inicialmente, aborda-se a modelagem temporal, destacando técnicas utilizadas, como ARIMA, Florestas Aleatórias e Redes Neurais. Em seguida, explora-se a modelagem de dependência espacial, com foco em métodos como o modelo de Autoregressão Espacial (SAR) e Redes Neurais em Grafo (GNN). Por fim, discute-se como integrar informações espaciais em modelos temporais para melhorar a precisão preditiva.

A pandemia de COVID-19 representou um dos maiores desafios de saúde pública do século XXI, mobilizando governos e instituições em todo o mundo. No estado de São Paulo, diversas medidas foram implementadas ao longo de 2020 para conter a disseminação do vírus, incluindo a quarentena decretada em março, que restringiu o funcionamento de serviços não essenciais, e o lançamento do "Plano São Paulo" em junho. Este plano estruturou a reabertura econômica em fases baseadas em indicadores de saúde, como taxas de ocupação hospitalar e contaminação. Outras ações incluíram o incentivo ao distanciamento social, a obrigatoriedade do uso de máscaras em locais públicos e restrições de horário e capacidade em estabelecimentos comerciais.

Apesar dessas medidas, o estado enfrentou períodos de alta incidência de COVID-19, com forte evidência de dependência espacial entre as cidades. Esse padrão reflete como a propagação do vírus em uma região foi frequentemente influenciada pela situação epidemiológica de regiões vizinhas. Diante disso, a previsão de séries temporais epidemiológicas, aliada à análise espacial, tornou-se uma ferramenta essencial para compreender e antecipar a dinâmica da pandemia, permitindo uma alocação mais eficiente de recursos e a implementação de políticas públicas.

2.1 Série Temporal

Uma série temporal consiste em uma sequência de dados observados ao longo do tempo, geralmente em intervalos regulares (Box et al., 2015). Exemplos de séries temporais incluem medições diárias de temperatura, dados econômicos mensais, medições de batimentos cardíacos ao longo do tempo ou até mesmo a produção de cerveja na Austrália, como na figura 1. O objetivo da análise de séries temporais é entender a estrutura temporal dos dados para realizar previsões,

identificar padrões e obter informações sobre o comportamento das variáveis ao longo do tempo (Hyndman; Athanasopoulos, 2018).

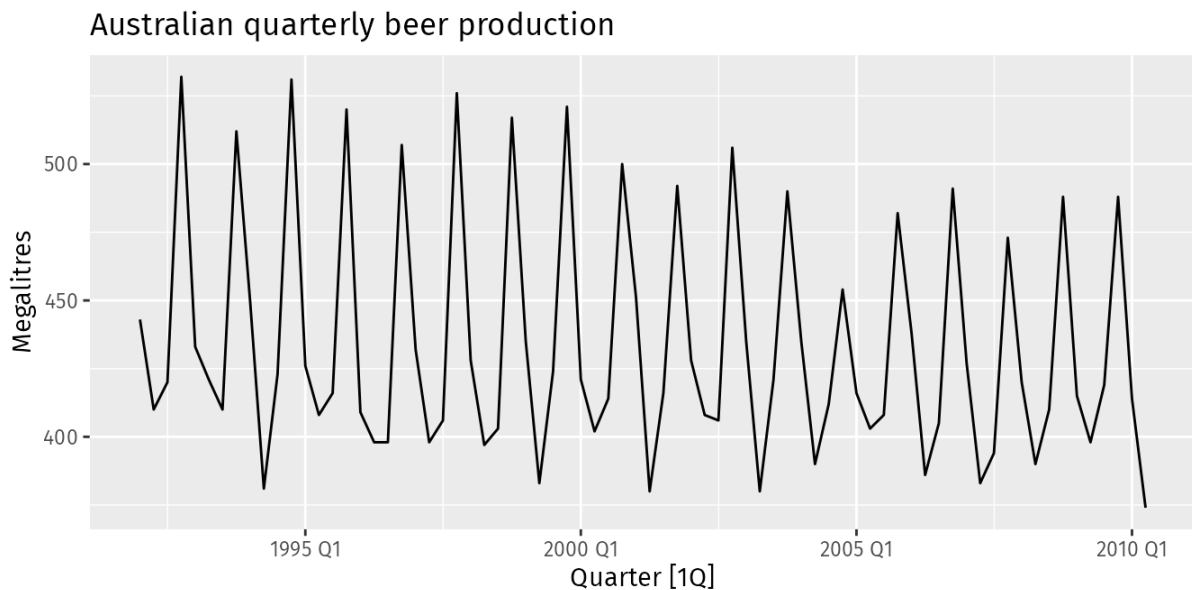


Figura 1 – Exemplo de uma série temporal da produção de cerveja na Austrália. Fonte: (Hyndman; Athanasopoulos, 2018).

A previsão de séries temporais trabalha principalmente com extrapolação, ou seja, a estimativa de valores futuros fora do intervalo dos dados observados (Makridakis; Wheelwright; Hyndman, 1997), também chamado de extrapolação. Isso contrasta com a regressão clássica, que se concentra na interpolação, ou seja, estimando valores dentro do intervalo dos dados observados.

A análise de séries temporais pode ser realizada por diversos métodos, como os modelos ARIMA (AutoRegressive Integrated Moving Average) (Box et al., 2015), ETS (Exponential Smoothing State Space Models) (Hyndman; Khandakar, 2008), modelos de árvores de regressão (Breiman et al., 1984), Redes Neurais Autoregressivas (Zhang; Qi, 2005), e LSTM (Long Short-Term Memory) (Hochreiter; Schmidhuber, 1997).

Autocorrelação, Autocorrelação Parcial e Correlograma

A autocorrelação é uma medida que indica a correlação entre observações de uma série temporal separadas por diferentes períodos de tempo (Box et al., 2015). A autocorrelação parcial, por outro lado, mede a correlação entre observações em diferentes períodos de tempo, mas controla as correlações com os valores intermediários (Hyndman; Athanasopoulos, 2018). Um correlograma é um gráfico que exibe a autocorrelação ou a autocorrelação parcial para diferentes defasagens (lags) na série temporal, sendo uma ferramenta útil para identificar a estrutura subjacente da série. A Figura 2 é um exemplo de um correlograma da série temporal apresentada anteriormente na Figura 1, no qual evidenciamos a presença de correlações positivas

em trimestres subsequentes (ou seja, o mesmo trimestre de cada ano) e correlações negativas em trimestres opostos, possivelmente relacionadas à sazonalidade do consumo, influenciado por fatores como a temperatura.

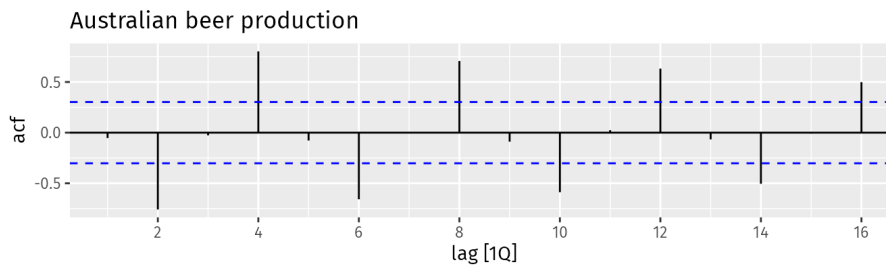


Figura 2 – Exemplo de correlograma da produção de cerveja na Austrália. Fonte: (Hyndman; Athanasopoulos, 2018).

Processo Estocástico

Formalmente, uma série temporal é uma realização de um *processo estocástico*, que pode ser definido como uma coleção de variáveis aleatórias $\{y_t : t \in T\}$, onde T representa um conjunto de índices, que neste caso são índices temporais (geralmente são números inteiros ou reais, dependendo se o tempo é discreto ou contínuo), e y_t representa o valor observado no tempo t . O processo estocástico descreve como essas variáveis aleatórias evoluem ao longo do tempo podendo assumir intervalos de valores. Quando temos uma realização particular desse processo estocástico (uma trajetória particular), chamamos de série temporal observada.

As séries temporais podem ser classificadas em dois tipos principais:

- **Séries Temporais Univariadas:** Consistem em uma única sequência de valores observados ao longo do tempo. Um exemplo seria a temperatura diária em uma cidade ao longo de um ano (Chatfield, 2003).
- **Séries Temporais Multivariadas:** Consistem em múltiplas sequências de valores observados ao longo do tempo. Por exemplo, as séries que contêm as observações de temperatura, umidade e precipitação diárias em uma cidade ao longo de um ano (Hyndman; Athanasopoulos, 2018).

Componentes de uma Série Temporal

As séries temporais geralmente possuem vários componentes que precisam ser analisados e modelados:

- **Tendência (Trend):** Refere-se ao movimento de longo prazo na série temporal. Pode ser uma tendência crescente, decrescente ou constante (Box et al., 2015).

- **Sazonalidade (Seasonality):** Refere-se aos padrões repetitivos ou cíclicos que ocorrem em intervalos regulares de tempo, como padrões diários, mensais ou anuais (Hyndman; Athanasopoulos, 2018).
- **Ciclos (Cycles):** Refere-se aos padrões de longo prazo que não têm uma periodicidade fixa, como os ciclos econômicos. Comparados à sazonalidade, são difíceis de prever e podem ser influenciados por fatores externos como a economia global (Box et al., 2015).
- **Irregularidades (Irregularities):** Refere-se às variações aleatórias ou ruídos que não seguem nenhum padrão específico. Essas irregularidades podem ser vistas como o "ruído" na série, representando as variações imprevisíveis (Hyndman; Athanasopoulos, 2018).

Estacionariedade

A estacionariedade é uma condição importante em muitos métodos de análise de séries temporais, pois muitos modelos, como o ARMA, assumem que os dados são estacionários. Uma série temporal é dita estacionária se suas propriedades estatísticas, como média, variância e autocorrelação, permanecerem constantes ao longo do tempo (Box et al., 2015). Na Figura 3, observamos três séries temporais: a primeira é estacionária; a segunda é não estacionária devido à média não constante ao longo do tempo; e a terceira também é não estacionária, mas em razão da variância não constante.

Se uma série temporal não for estacionária, pode ser necessário aplicar técnicas como a diferenciação para torná-la estacionária. A diferenciação consiste em subtrair o valor anterior de cada observação, o que ajuda a eliminar tendências de longo prazo e padrões sazonais (Hyndman; Athanasopoulos, 2018).

Formalmente, um processo estocástico é chamado de **estritamente estacionário** se a distribuição conjunta de $\{y_1, y_2, \dots, y_n\}$ for invariante sob translações no tempo. Em outras palavras, para qualquer deslocamento $h \in \mathbb{Z}$, a distribuição conjunta de $\{y_1, y_2, \dots, y_n\}$ deve ser a mesma que a distribuição conjunta de $\{y_{1+h}, y_{2+h}, \dots, y_{n+h}\}$.

Um processo estocástico é dito **fracamente estacionário** (ou estacionário no sentido amplo) se satisfizer as três condições a seguir:

- A média é constante ao longo do tempo: $E[y_t] = \mu$.
- A variância é constante ao longo do tempo: $\text{Var}(y_t) = \sigma^2$.
- A covariância entre y_t e y_{t+h} depende apenas do intervalo h , e não do tempo t : $\text{Cov}(y_t, y_{t+h}) = \gamma(h)$, onde $\gamma(h)$ é a função de autocovariância.

A condição de estacionariedade fraca é geralmente suficiente para a maioria das aplicações práticas, como modelagem com ARMA, pois a maioria dos métodos estatísticos baseados em médias e covariâncias assume estacionariedade no sentido amplo.

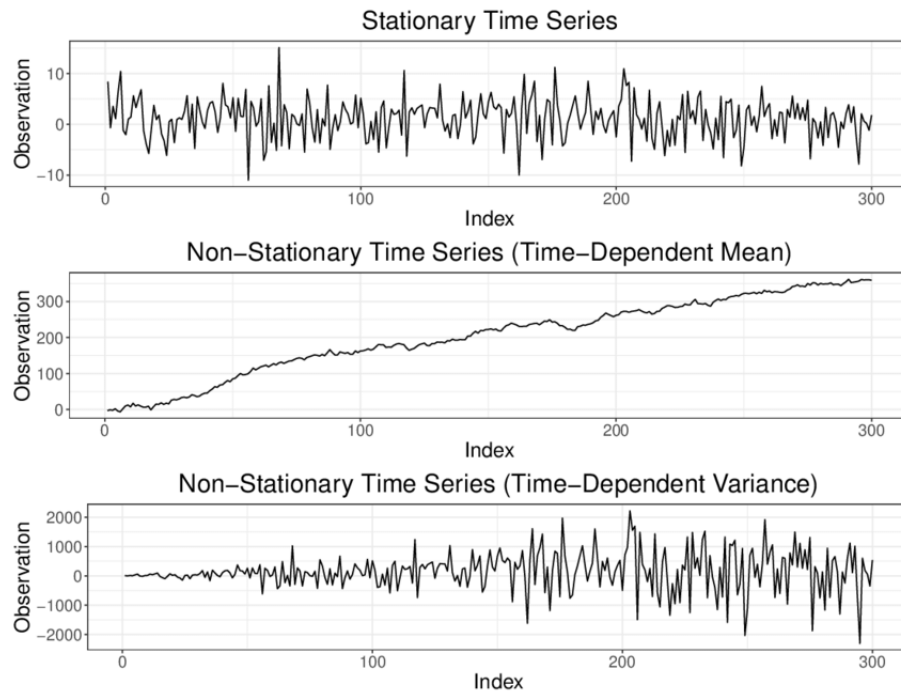


Figura 3 – Exemplo visual de uma série temporal estacionária comparada com duas séries temporais não estacionárias. Fonte: (Sahin, 2020).

Raiz Unitária

A raiz unitária é um conceito central em análises de séries temporais, relacionado à não estacionariedade da série. Uma série é dita possuir uma raiz unitária quando o valor atual depende fortemente do valor anterior, somado a um componente de erro aleatório. Essa característica implica que choques na série podem ter efeitos permanentes, fazendo com que ela apresente um comportamento de *random walk* (passeio aleatório) (Hyndman; Athanasopoulos, 2018).

Matematicamente, um passeio aleatório pode ser representada por:

$$y_t = y_{t-1} + \varepsilon_t,$$

onde ε_t é o termo de erro aleatório. Esse modelo sugere que mudanças passadas afetam a série de forma acumulativa, tornando-a imprevisível no longo prazo sem transformações apropriadas.

A presença de uma raiz unitária na série indica que ela não é estacionária, ou seja, suas propriedades estatísticas, como média e variância, não permanecem constantes ao longo do tempo. Esse comportamento dificulta a modelagem e a previsão, já que alguns modelos estatísticos, como ARMA, assumem estacionariedade. Para lidar com isso, é comum aplicar a diferenciação da série, eliminando a raiz unitária e transformando-a em estacionária.

Teste de Dickey-Fuller

O Teste de Dickey-Fuller é um teste estatístico utilizado para identificar a presença de raiz unitária em uma série temporal (Dickey; Fuller, 1979). Ele permite verificar se uma série é

estacionária ou se precisa ser transformada, como pela diferenciação, para se adequar a modelos estatísticos que exigem estacionariedade.

A formulação básica do teste considera a seguinte equação de regressão:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^p \phi_i \Delta y_{t-i} + \varepsilon_t,$$

onde:

- Δy_t é a primeira diferença da série ($y_t - y_{t-1}$);
- α é o intercepto (opcional);
- βt é o termo de tendência temporal (opcional);
- γy_{t-1} é o coeficiente associado à raiz unitária;
- ϕ_i representa o coeficientes da i -ésima diferença defasada para ajustar autocorrelação, para $i = 1, 2, \dots, p$;
- ε_t é o erro aleatório para $t = 1, 2, \dots, n$;

O teste avalia a seguinte hipótese:

Hipótese nula (H_0): A série possui uma raiz unitária, ou seja, não é estacionária ($\gamma = 0$).

Hipótese alternativa (H_1): A série é estacionária ($\gamma < 0$).

O funcionamento do teste deve seguir os passos a seguir:

1. Ajusta-se a equação de regressão acima à série temporal.
2. O coeficiente γ é avaliado por meio de um teste estatístico (normalmente um teste t).
3. Compara-se o valor estatístico do teste com valores críticos apropriados. Se o valor calculado for menor que o valor crítico, rejeita-se a hipótese nula e conclui-se que a série é estacionária.

Se o teste não rejeitar H_0 , a série apresenta uma raiz unitária e não é estacionária, sendo necessário aplicar transformações como a diferenciação.

Se o teste rejeitar H_0 , a série é estacionária e pode ser usada diretamente em modelos que requerem estacionariedade.

O Teste de Dickey-Fuller é fundamental para garantir a validade de modelos temporais que assumem estacionariedade.

Ruído Branco

O ruído branco é um conceito essencial na análise de séries temporais. Refere-se a uma série de valores aleatórios onde cada valor é independente dos outros e segue uma distribuição com média zero e variância constante (Box et al., 2015). Em outras palavras, um ruído branco não possui autocorrelação significativa e representa o comportamento de erro puro em uma série temporal. Se os resíduos de um modelo são ruído branco, isso sugere que o modelo ajustado capturou todos os padrões previsíveis da série.

2.2 Modelos de Séries Temporais

A análise de séries temporais envolve várias técnicas para modelar e prever dados temporais. Há diversas técnicas que podem ser utilizadas com intuítos diferentes, sendo que um dos mais comuns é modelar para a tarefa de previsão, que é o objetivo do presente trabalho. Na literatura podemos encontrar métodos clássicos e métodos baseados em aprendizado de máquina (Hyndman; Athanasopoulos, 2018) para este tipo de tarefa. Entre os métodos clássicos, destacam-se os modelos ARIMA (*Autoregressive Integrated Moving Average*) (Box et al., 2015), enquanto entre os métodos de aprendizado de máquina, destacam-se as Florestas Aleatórias (Breiman, 2001) e as Redes Neurais LSTM (*Long Short-Term Memory*) (Hochreiter; Schmidhuber, 1997).

Na análise de séries temporais, frequentemente trabalhamos com previsões recursivas de múltiplas etapas, onde a previsão no tempo t influencia na previsão no tempo $t + 1$. Este tipo de previsão é essencial para capturar as dependências dinâmicas ao longo do tempo (Hyndman; Athanasopoulos, 2018).

2.2.1 Modelo ARIMA

O modelo ARIMA (*AutoRegressive Integrated Moving Average*) é uma das ferramentas mais importantes na análise de séries temporais e possui uma história rica, refletindo o desenvolvimento da estatística aplicada ao longo do século XX. Ele foi formalmente consolidado e popularizado por George E. P. Box e Gwilym M. Jenkins em seu livro seminal *Time Series Analysis: Forecasting and Control*, publicado pela primeira vez em 1970 (Box et al., 2015). Desde então, o modelo ARIMA também é conhecido como "modelo Box-Jenkins", em reconhecimento aos seus autores.

Antes da consolidação proposta por Box e Jenkins, os componentes individuais do ARIMA já haviam sido explorados:

- **Autoregressão (AR):** Introduzida no início do século XX por Yule (1927), que investigou a relação linear entre valores passados de uma série temporal (Yule, 1927).

- **Médias Móveis (MA):** Desenvolvida por Slutsky (1937), que demonstrou como flutuações aparentes poderiam surgir de processos simples de médias móveis (Slutsky, 1937).
- **Integração (I):** A ideia de diferenciação para transformar séries não estacionárias em estacionárias começou a ser formalizada em meados da década de 1940.

A contribuição de Box e Jenkins foi combinar esses conceitos em um modelo unificado e criar uma metodologia sistemática para ajuste de modelos. A chamada *metodologia Box-Jenkins* é composta por três etapas principais:

1. **Identificação:** Determinação dos valores p , d e q , correspondentes à ordem autoregressiva, ordem de diferenciação e ordem de médias móveis, respectivamente.
2. **Estimação:** Ajuste dos parâmetros do modelo utilizando métodos como a máxima verossimilhança.
3. **Diagnóstico:** Avaliação dos resíduos do modelo para garantir que ele captura adequadamente os padrões da série temporal.

Desde sua introdução, o modelo ARIMA tornou-se uma referência na análise de séries temporais e é amplamente aplicado em áreas como economia, saúde pública e engenharia. Sua simplicidade e robustez permitem modelar dependências temporais complexas em séries não estacionárias, enquanto extensões como SARIMA e ARIMAX expandiram suas aplicações para incluir sazonalidade e variáveis exógenas.

O ARIMA continua sendo uma ferramenta essencial, tanto como modelo preditivo quanto como base para comparações com métodos mais avançados, como aprendizado de máquina e redes neurais.

O modelo ARIMA é utilizado para modelar e prever séries temporais. Ele combina três componentes principais: autoregressão (AR), integração (I) e média móvel (MA) (Box et al., 2015). O modelo ARIMA é denotado como ARIMA(p,d,q), onde:

- p é a ordem do componente autoregressivo (AR),
- d é o número de diferenças necessárias para tornar a série estacionária (I),
- q é a ordem do componente de média móvel (MA).

Para ficar mais claro o funcionamento do modelo ARIMA, é interessante olhar para cada componente.

Autoregressão (AR)

O componente AR indica que a variável de interesse é regredida em seus próprios valores passados. A formulação matemática do AR(p) é:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t \quad (2.1)$$

onde y_t é o valor da série temporal no tempo t , ϕ_i são os coeficientes autoregressivos para todo $i = 1, \dots, p$ e ϵ_t é um termo de erro de ruído branco (Box et al., 2015).

Integração (I)

A integração é utilizada para tornar a série temporal estacionária, ou seja, uma série cuja média e variância são constantes ao longo do tempo (Hyndman; Athanasopoulos, 2018). A operação de diferenciação é definida como:

$$y'_t = y_t - y_{t-1} \quad (2.2)$$

Para uma série integrada de ordem d , a série é diferenciada d vezes até se tornar estacionária. Esse processo elimina tendências e sazonalidades de longo prazo, permitindo que os modelos ARIMA sejam aplicados de forma eficaz (Box et al., 2015).

Média Móvel (MA)

O componente MA modela a relação entre a variável de interesse e os erros de previsão passados. A formulação matemática do MA(q) é:

$$y_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} \quad (2.3)$$

onde θ_j são os coeficientes de média móvel para todo $j = 1, \dots, q$ e ϵ_t é o termo de erro de ruído branco (Box et al., 2015).

Operador de defasagem

A notação do operador de defasagem (*Backshift*), é uma ferramenta útil para simplificar a representação de modelos autoregressivos. O operador Backshift, denotado por B , atua sobre uma série temporal y_t deslocando os valores da série para trás no tempo. Formalmente, o operador é definido como:

$$By_t = y_{t-1}$$

Ou seja, B aplicado a y_t produz o valor da série no tempo $t - 1$. De forma geral, o operador pode ser aplicado múltiplas vezes para deslocar a série temporal por múltiplos períodos. Por exemplo:

$$B^2 y_t = y_{t-2}$$

Essa notação é particularmente útil em modelos ARIMA, onde as componentes autoregressivas (AR) e de médias móveis (MA) podem ser representadas de maneira concisa usando o operador Backshift. Com ele, as equações autoregressivas e de médias móveis podem ser expressas como polinômios do operador B , facilitando a manipulação e o desenvolvimento dos modelos.

Por exemplo, o modelo AR(2) pode ser representado pela seguinte equação:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \epsilon_t$$

Com o operador de defasagem, isso pode ser reescrito como:

$$y_t = \phi_1 B y_t + \phi_2 B^2 y_t + \epsilon_t$$

que, reorganizado, torna-se:

$$(1 - \phi_1 B - \phi_2 B^2) y_t = \epsilon_t$$

Essa notação compacta é utilizada em modelos autoregressivos e de médias móveis, permitindo uma representação elegante e eficaz de processos estocásticos.

Modelo ARMA(p,q)

O modelo ARMA é uma combinação dos componentes AR e MA, sem diferenciação. Ele é útil para séries temporais que já são estacionárias ou que se tornaram estacionárias após transformação. A notação Backshift é frequentemente usada para simplificar a formulação desses modelos, onde B representa o operador de defasagem (lag) (Box et al., 2015).

A forma geral do modelo ARMA(p,q), utilizando esta notação, é:

$$\left(1 - \sum_{i=1}^p \phi_i B^i\right) y_t = \left(1 + \sum_{j=1}^q \theta_j B^j\right) \epsilon_t \quad (2.4)$$

Por exemplo, podemos observar como ficaria o modelo ARMA(2,2) que pode ser reescrito pela notação com o operador de defasagem como:

$$\begin{aligned} y_t &= \phi_1 y_{t-1} + \phi_2 y_{t-2} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \epsilon_t \\ \Rightarrow y_t - \phi_1 y_{t-1} - \phi_2 y_{t-2} &= \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} \\ \Rightarrow 1y_t - \phi_1 B y_t - \phi_2 B^2 y_t &= 1\epsilon_t + \theta_1 B \epsilon_t + \theta_2 B^2 \epsilon_t \\ \Rightarrow (1 - \phi_1 B - \phi_2 B^2) y_t &= (1 + \theta_1 B + \theta_2 B^2) \epsilon_t \end{aligned}$$

Modelo ARIMA(p,d,q)

O modelo ARIMA é a combinação dos três componentes mencionados: AR, I e MA. A forma geral do modelo ARIMA(p,d,q) é:

$$\left(1 - \sum_{i=1}^p \phi_i B^i\right) (1 - B)^d y_t = \left(1 + \sum_{j=1}^q \theta_j B^j\right) \epsilon_t \quad (2.5)$$

onde B é o operador de defasagem, ϕ_i é o i -ésimo coeficiente autoregressivo, θ_j é o j -ésimo coeficiente de média móvel e ϵ_t é o termo de ruído branco para $t = 1, 2, \dots, n$ (Box et al., 2015).

Se considerarmos z_t como a série temporal após as d diferenças, ou seja, $(1 - B)^d y_t = z_t$, podemos reescrever o modelo como:

$$\begin{aligned} \left(1 - \sum_{i=1}^p \phi_i B^i\right) z_t &= \left(1 + \sum_{j=1}^q \theta_j B^j\right) \epsilon_t \\ z_t &= (1 - B)^d y_t \end{aligned}$$

O processo de construção de um modelo ARIMA geralmente envolve três etapas principais:

1. **Identificação:** Análise das propriedades da série temporal, incluindo a verificação de estacionariedade e a determinação dos valores apropriados para p , d e q (Hyndman; Athanasopoulos, 2018).
2. **Estimação:** Estimação dos parâmetros ϕ e θ utilizando métodos como máxima verossimilhança (Box et al., 2015).
3. **Diagnóstico:** Verificação dos resíduos do modelo para garantir que eles se comportem como um ruído branco, o que indica um bom ajuste do modelo (Box et al., 2015).

No modelo ARIMA, fazemos a previsão na série estacionária e depois aplicamos uma transformação inversa para prever na série original.

Quando fazemos previsões para múltiplos passos à frente em uma série temporal diferenciada, a recuperação dos valores na escala original envolve um processo iterativo. Vamos supor que temos dados da série temporal até o tempo t . Assim, temos conhecimento de $y_1, y_2, \dots, y_t$.

- Suponhamos que fizemos apenas uma diferenciação da série original, ou seja, $d = 1$. Neste caso, temos:

$$z_t = y_t - y_{t-1} \Rightarrow y_t = z_t + y_{t-1}$$

Para previsões de m passos à frente, obtemos os valores previstos $\hat{z}_{t+1}, \hat{z}_{t+2}, \dots, \hat{z}_{t+m}$. Podemos calcular os valores da série original como:

$$\begin{aligned} \hat{y}_{t+1} &= \hat{z}_{t+1} + y_t \\ \hat{y}_{t+2} &= \hat{z}_{t+2} + \hat{y}_{t+1} \\ &\vdots \\ \hat{y}_{t+m} &= \hat{z}_{t+m} + \hat{y}_{t+m-1} \end{aligned}$$

- Suponhamos agora que fizemos duas diferenças na série original, ou seja, $d = 2$. Neste caso, temos:

$$z_t = y_t - 2y_{t-1} + y_{t-2} \Rightarrow y_t = z_t + 2y_{t-1} - y_{t-2}$$

Para previsões de m passos à frente, obtemos os valores previstos $\hat{z}_{t+1}, \hat{z}_{t+2}, \dots, \hat{z}_{t+m}$. Podemos calcular os valores da série original como:

$$\begin{aligned}\hat{y}_{t+1} &= \hat{z}_{t+1} + 2y_t - y_{t-1} \\ \hat{y}_{t+2} &= \hat{z}_{t+2} + 2\hat{y}_{t+1} - y_t \\ &\vdots \\ \hat{y}_{t+m} &= \hat{z}_{t+m} + 2\hat{y}_{t+m-1} - \hat{y}_{t+m-2}\end{aligned}$$

- Para um caso geral de d diferenças, temos $z_t = (1 - B)^d y_t$, onde o operador de defasagem elevado à potência d pode ser expandido utilizando o Binômio de Newton:

$$(1 - B)^d y_t = \sum_{j=0}^d (-1)^j \binom{d}{j} B^j y_t = \sum_{j=0}^d (-1)^j \binom{d}{j} y_{t-j}.$$

De fato, pelo Binômio de Newton, temos:

$$(a + b)^n = \sum_{j=0}^n \binom{n}{j} a^{n-j} b^j.$$

Aplicando com $a = 1$, $b = -B$ e $n = d$, obtemos:

$$(1 - B)^d = \sum_{j=0}^d \binom{d}{j} (-1)^j B^j.$$

Assim, temos então:

$$(1 - B)^d y_t = (1 + (-B))^d y_t = \left(\sum_{j=0}^d \binom{d}{j} 1^{d-j} (-B)^j \right) y_t = \sum_{j=0}^d \binom{d}{j} 1^{d-j} (-B)^j y_t$$

Como 1 elevado a qualquer número continua sendo 1 então $1^{d-j} = 1$ e pode ser removido da expressão. Além disso,

$$(-B)^j y_t = ((-1)B)^j y_t = (-1)^j B^j y_t = (-1)^j y_{t-j}$$

do qual, obtém-se

$$z_t = (1 - B)^d y_t = \sum_{j=0}^d (-1)^j \binom{d}{j} y_{t-j}$$

A expressão acima facilitará no processo do cálculo da previsão série original. Observemos que quando $j = 0$ temos $(1 - B)^j y_t = (1 - B)^0 y_t = y_t$. Assim,

$$z_t = \sum_{j=0}^d (-1)^j \binom{d}{j} y_{t-j} = y_t + \sum_{j=1}^d (-1)^j \binom{d}{j} y_{t-j}$$

$$\Rightarrow y_t = z_t - \sum_{j=1}^d (-1)^j \binom{d}{j} y_{t-j}$$

Para previsões de m passos à frente, obtemos os valores previstos $\hat{z}_{t+1}, \hat{z}_{t+2}, \dots, \hat{z}_{t+m}$ e podemos calcular os valores da série original como:

$$\begin{aligned} \hat{y}_{t+1} &= \hat{z}_{t+1} - \sum_{j=1}^d (-1)^j \binom{d}{j} y_{t+1-j} \\ \hat{y}_{t+2} &= \hat{z}_{t+2} - \sum_{j=1}^d (-1)^j \binom{d}{j} \hat{y}_{t+2-j} \\ &\vdots \\ \hat{y}_{t+m} &= \hat{z}_{t+m} - \sum_{j=1}^d (-1)^j \binom{d}{j} \hat{y}_{t+m-j} \end{aligned}$$

Neste processo, a cada passo adicional, utilizamos as previsões anteriores para reconstruir a série na escala original.

Para visualizar num exemplo, o modelo $ARIMA(1,1,1)$ ficaria da seguinte forma:

$$(1 - \phi_1 B)(1 - B)y_t = (1 + \theta_1 B)\epsilon_t \quad (2.6)$$

As medidas de ajuste, como o AIC (Akaike Information Criterion) e o BIC (Bayesian Information Criterion), são utilizadas para comparar diferentes modelos e selecionar aquele que melhor se ajusta aos dados, balanceando o ajuste e a complexidade do modelo (Akaike, 1974; Schwarz, 1978). O AIC e o BIC penalizam modelos mais complexos com mais parâmetros, ajudando a evitar o sobreajuste (*overfitting*). Um modelo com menor AIC ou BIC é preferível, desde que os resíduos do modelo sejam adequados.

O AutoARIMA é uma abordagem automatizada para identificar o melhor modelo ARIMA para uma série temporal, minimizando o AIC (Hyndman; Khandakar, 2008). O algoritmo utiliza uma busca *stepwise* para explorar diferentes combinações de p , d e q e seleciona aquela que resulta no melhor ajuste com base em critérios de informação como AIC.

O AutoARIMA permite estimar os coeficientes de forma automatizada utilizando métodos estatísticos como a máxima verossimilhança (Hyndman; Athanasopoulos, 2018). Essa abordagem é realizada em algumas etapas, primeiramente é feito o teste de Dickey Fuller para verificar se há raiz unitária na série temporal, ou seja, verificar se a série não é estacionária. Caso tenha raiz unitária, é aplicado a diferenciação na série e aplicado o teste de Dickey Fuller novamente se ainda há raiz unitária. O processo é feito até que a série não tenha mais raiz unitária, ou seja, até a série se tornar estacionária, geralmente após duas diferenças no máximo.

Depois do hiperparâmetro d ser estimado, são testados alguns modelos, $ARIMA(0, d, 0)$, $ARIMA(1, d, 0)$, $ARIMA(0, d, 1)$ e $ARIMA(2, d, 2)$ e a seleção do melhor ajuste é feito

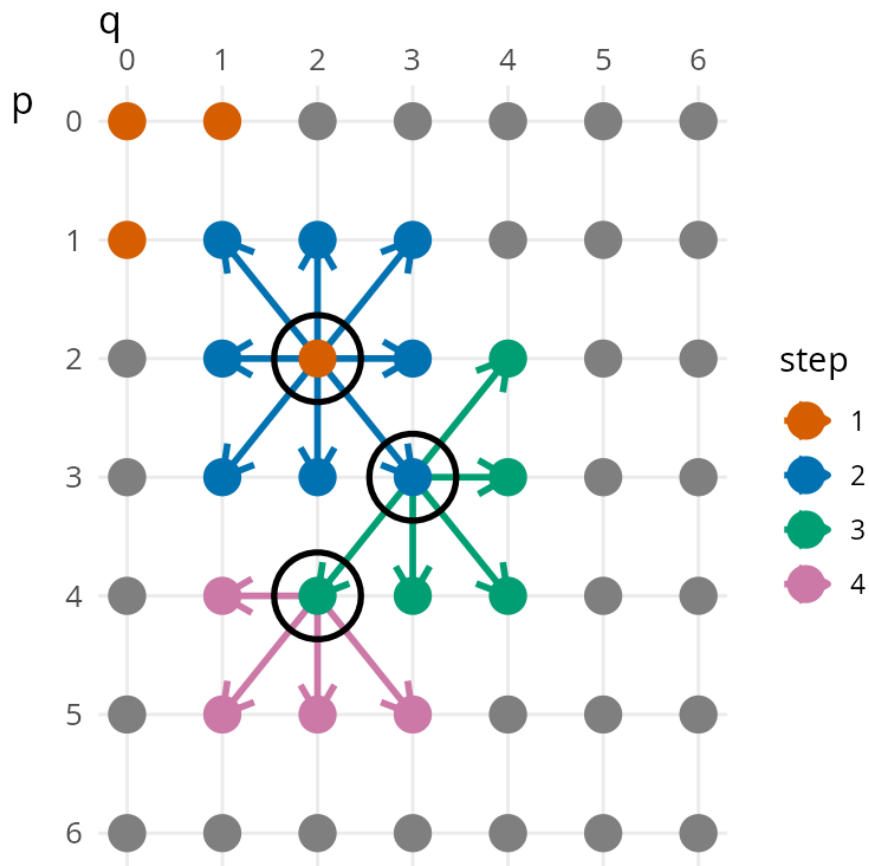


Figura 4 – Ilustração de como o Autoarima é feito. Neste exemplo, foram 4 etapas no total, resultando em $ARIMA(4, d, 2)$ ser a melhor escolha. Fonte: (Hyndman; Athanasopoulos, 2018).

através do AIC. Depois de escolher o melhor modelo dentre os quatro, é feito novamente uma comparação, mas com a variação de ± 1 para p e q gerando 8 novos modelos para testar. Isso fica bem ilustrado na Figura 4, em que os 8 novos modelos aparecem ao redor do modelo atual. Então dos 9 modelos é selecionado aquele que possui o menor AIC. Se o modelo escolhido for um dos 8 novos modelos, então o processo é feito novamente até que não haja mudança do modelo escolhido na última etapa.

A estimação por máxima verossimilhança (EMV) é uma das formas de se ajustar os parâmetros do modelo ARIMA, que incluem os coeficientes autoregressivos (ϕ), de médias móveis (θ) e a variância do erro (σ^2). O objetivo da EMV é encontrar os valores desses parâmetros que maximizem a probabilidade de observar os dados disponíveis.

A função de verossimilhança é definida como a probabilidade conjunta das observações $y_1, y_2, \dots, y_T$, assumindo que os erros (ϵ_t) seguem uma distribuição normal independente:

$$L(\Theta; \mathbf{y}) = \prod_{t=1}^T \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_t - \hat{y}_t)^2}{2\sigma^2}\right), \quad (2.7)$$

onde $\Theta = (\phi, \theta, \sigma^2)$ representa o conjunto de parâmetros do modelo, e ϵ_t são os resíduos calculados como a diferença entre os valores observados e ajustados.

Para simplificar o processo de otimização, utiliza-se o logaritmo da função de verossimilhança, resultando na função de log-verossimilhança:

$$\ell(\Theta; \mathbf{y}) = -\frac{T}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{t=1}^T \epsilon_t^2. \quad (2.8)$$

Para facilitar o entendimento de como estimar os parâmetros do modelo ARIMA, podemos observar casos mais simples do modelo por primeiro.

Por exemplo, no modelo autorregressivo de ordem 1 (AR(1)), a variável y_t é expressa como:

$$y_t = \phi y_{t-1} + e_t, \quad (2.9)$$

onde ϕ é o parâmetro a ser estimado e e_t representa o termo de erro branco com média zero e variância constante, para $t = 1, \dots, T$.

Para estimar ϕ pelo método da máxima verossimilhança, consideramos a função de verossimilhança baseada na distribuição normal dos erros. A log-verossimilhança é dada por

$$\ell(\phi; \mathbf{y}) = -\frac{T}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{t=1}^T (y_t - \phi y_{t-1})^2. \quad (2.10)$$

e maximizando essa função em relação a ϕ e σ^2 , obtemos as estimativas dos parâmetros.

Já no modelo MA(1) (modelo de médias móveis de ordem 1), a variável y_t é modelada como

$$y_t = e_t - \theta e_{t-1}, \quad (2.11)$$

onde θ é o parâmetro de interesse.

Em ambos os casos, a função de verossimilhança depende dos termos de erro e_t , que não são observáveis diretamente. Portanto, a estimação de θ requer métodos numéricos para maximizar a log-verossimilhança correspondente.

Para o modelo ARIMA, os dados são frequentemente diferenciados d vezes para atingir estacionariedade. Com a série já estacionária, temos os parâmetros da componente AR e da componente MA para estimar. Os resíduos ϵ_t são obtidos pela fórmula do modelo ARIMA:

$$\begin{aligned} (1 - \sum_{i=1}^p \phi_i B^i) z_t &= (1 + \sum_{j=1}^q \theta_j B^j) \epsilon_t \\ z_t &= (1 - B)^d y_t \end{aligned}$$

A maximização da log-verossimilhança é realizada seguindo os passos abaixo:

1. **Inicialização:** Fornecer estimativas iniciais para ϕ , θ e σ^2 .
2. **Cálculo dos resíduos:** Usar as equações do modelo ARIMA para calcular ϵ_t iterativamente.
3. **Log-verossimilhança:** Substituir os resíduos na função de log-verossimilhança:

$$\ell(\Theta; y) = -\frac{T}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{t=1}^T (y_t - \hat{y}_t)^2. \quad (2.12)$$

4. **Maximização:** Ajustar Θ para maximizar $\ell(\Theta; y)$ usando métodos numéricos, como o algoritmo de Newton-Raphson.
5. **Verificação:** Avaliar se os resíduos ϵ_t são ruído branco ($\epsilon_t \sim N(0, \sigma^2)$).

A maximização da função de log-verossimilhança para o modelo ARIMA é um problema de otimização não linear. Métodos numéricos, como o Newton-Raphson ou L-BFGS, são frequentemente utilizados para resolver esse problema.

O método de Newton-Raphson é um algoritmo iterativo clássico usado para encontrar raízes de funções ou maximizar/minimizar funções, incluindo a log-verossimilhança em modelos estatísticos, enquanto o método L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) é um algoritmo de otimização baseado no método de Newton que utiliza aproximações da matriz Hessiana (segunda derivada).

Esses métodos seguem uma abordagem iterativa, parte-se de valores iniciais para os parâmetros, calcula-se o gradiente da log-verossimilhança, ajustam-se os parâmetros com base nessa informação, e repete-se o ciclo até que as mudanças entre iterações se tornem suficientemente pequenas.

2.2.2 Aprendizado de Máquina

O aprendizado de máquina é uma subárea da inteligência artificial que tem como objetivo capacitar sistemas computacionais a aprender padrões a partir de dados, sem que sejam explicitamente programados para tal. Essa capacidade é alcançada por meio de algoritmos e técnicas matemáticas que permitem que os modelos melhorem seu desempenho ao serem expostos a novos dados (Hastie; Tibshirani; Friedman, 2009).

A área emergiu nas últimas décadas como uma ferramenta essencial para lidar com grandes volumes de dados e com problemas complexos que envolvem padrões não lineares. Suas raízes remontam aos anos 1950, com o desenvolvimento inicial do perceptron, e ao longo do tempo se expandiu para incluir abordagens mais sofisticadas, como métodos baseados em árvores, redes neurais profundas e técnicas de aprendizado não supervisionado.

Os modelos de aprendizado de máquina podem ser divididos em três categorias principais:

- **Aprendizado supervisionado:** O modelo é treinado com dados rotulados, onde as entradas são associadas às saídas esperadas. Essa abordagem é utilizada para problemas de classificação e regressão.
- **Aprendizado não supervisionado:** O objetivo é identificar padrões ocultos nos dados sem informações rotuladas, como em algoritmos de clusterização ou redução de dimensionalidade.
- **Aprendizado por reforço:** Baseado em um agente que aprende interagindo com o ambiente, recebendo recompensas ou penalidades por suas ações, a fim de maximizar uma métrica cumulativa ao longo do tempo.

Uma das grandes vantagens do aprendizado de máquina é sua capacidade de lidar com dados de natureza complexa e de alta dimensionalidade, algo comum em estudos epidemiológicos, como os de COVID-19. Técnicas modernas permitem modelar relações intrincadas entre variáveis, capturando tanto dependências temporais quanto padrões espaciais nos dados.

Neste trabalho, foram utilizados dois modelos reconhecidos no aprendizado supervisionado: a Floresta Aleatória e as Redes Neurais do tipo Long Short-Term Memory (LSTM). Cada um desses modelos foi selecionado por sua capacidade específica de lidar com os desafios das séries temporais epidemiológicas, conforme detalhado nas subseções a seguir.

O desempenho desses modelos de aprendizado de máquina depende de diversos fatores, entre eles a escolha adequada dos hiperparâmetros, que são configurações definidas antes do processo de treinamento. Diferentemente dos parâmetros aprendidos durante o ajuste do modelo, os hiperparâmetros controlam o funcionamento interno dos algoritmos, como o número de árvores em uma Floresta Aleatória ou o número de neurônios e épocas de treinamento em uma rede neural. A seleção apropriada desses hiperparâmetros é essencial para alcançar bons resultados, sendo comumente realizada por metodologias de otimização dos mesmos.

O uso do aprendizado de máquina neste contexto surge como uma tentativa de melhoria da precisão preditiva em relação aos modelos clássicos.

2.2.3 Floresta Aleatória

A Floresta Aleatória (*Random Forest*) é um modelo de aprendizado de máquina utilizado em tarefas de regressão e classificação, sendo também eficaz para a previsão de séries temporais. Desenvolvida por Breiman (Breiman, 2001), é uma técnica do tipo *ensemble*, que combina diversos modelos de aprendizado de máquina para obter uma previsão única através da média ou votação dos resultados individuais. No contexto de séries temporais, o modelo constrói múltiplas árvores de decisão e utiliza a média das previsões de todas as árvores para realizar a previsão final

Uma árvore de decisão é composta por nós que particionam os dados de entrada em subconjuntos com base em regras de decisão. No contexto de séries temporais, cada nó avalia uma variável de defasagem (*lag*) para prever o valor futuro.

Para tarefas de regressão, os valores preditos em cada folha da árvore são calculados como a média dos valores das observações no subconjunto correspondente:

$$\hat{y} = \frac{1}{n} \sum_{i=1}^n y_i, \quad (2.13)$$

onde n representa o número de observações no subconjunto e y_i os valores reais.

A Floresta Aleatória foi desenvolvida para superar limitações de árvores de decisão individuais, como sua tendência ao sobreajuste. Ela introduz duas técnicas principais: amostragem *extitbootstrap* e seleção aleatória de atributos para cada divisão (Breiman, 2001). Essa combinação melhora a generalização e a robustez do modelo, tornando-o ideal para dados não lineares e de alta dimensionalidade.

A Floresta Aleatória é formada por H árvores de decisão $h(x, \Theta_k)$, onde os vetores de parâmetros Θ_k são amostrados independentemente:

$$\hat{y} = \frac{1}{H} \sum_{k=1}^H h(x, \Theta_k). \quad (2.14)$$

Essa abordagem reduz a variabilidade e melhora a precisão, evitando o sobreajuste (Breiman, 2001).

O próximo passo é a estimação dos parâmetros da equação 2.14. Os parâmetros das árvores de decisão na Floresta Aleatória são estimados com base em amostras *bootstrap* e na seleção de divisões que maximizam a redução de impureza. O processo pode ser descrito da seguinte forma:

- **Amostragem *Bootstrap*:** Cada árvore é treinada em uma amostra aleatória com reposição dos dados de treinamento.
- **Seleção de Divisões:** Para cada nó, um subconjunto aleatório de atributos é selecionado, e a divisão que maximiza a redução de impureza (no caso para problemas de regressão, utiliza-se o erro quadrático médio) é escolhida.
- **Agregação:** As previsões de todas as árvores são combinadas por média (regressão) ou votação (classificação) para gerar o resultado final.

Esse procedimento garante diversidade entre as árvores e reduz a variância do modelo.

Os principais hiperparâmetros da Floresta Aleatória incluem o número de árvores (H), que controla a diversidade do *ensemble*; a profundidade máxima das árvores, que equilibra sobreajuste e subajuste; e o número de atributos considerados por divisão, responsável por

introduzir aleatoriedade em cada nó. A escolha adequada desses hiperparâmetros pode ser feita por meio de técnicas como *grid search* ou *random search*, avaliando diferentes combinações e selecionando aquela com melhor desempenho nos dados de treino.

Além disso, a Floresta Aleatória permite estimar a importância de cada variável de entrada por meio da soma da redução de impureza associada a cada divisão:

$$\text{Importância}(X_j) = \sum_{n \in \text{nós}} \Delta \text{Impureza}(n|X_j), \quad (2.15)$$

onde $\Delta \text{Impureza}(n|X_j)$ representa a redução de impureza ao usar a variável X_j no nó n .

Entre suas vantagens, destacam-se a robustez a relações não lineares, a capacidade de lidar com alta dimensionalidade e a identificação automática de padrões relevantes. Por outro lado, apresenta como limitações a elevada complexidade computacional e a dificuldade em extrapolar além do intervalo observado nos dados de treino.

Na aplicação a séries temporais, é necessário reestruturar os dados, criando variáveis de defasagem (*lags*):

$$X_t = \{y_{t-1}, y_{t-2}, \dots, y_{t-p}\}, \quad (2.16)$$

onde p representa o número de defasagens consideradas. Essas variáveis capturam a dependência temporal da série, funcionando como um modelo autoregressivo (Tashman, 2000).

2.2.4 Redes Neurais

As redes neurais artificiais têm suas raízes no modelo matemático proposto por McCulloch e Pitts em 1943 (McCulloch; Pitts, 1943). Este modelo foi a primeira tentativa de representar computacionalmente o comportamento de neurônios biológicos. Em 1958, Rosenblatt desenvolveu o perceptron, uma rede neural simples que combinava entradas linearmente e aplicava uma função de limiar para classificar dados (Rosenblatt, 1958).

Entretanto, em 1969, Minsky e Papert mostraram as limitações do perceptron para resolver problemas não linearmente separáveis, como o famoso problema do XOR. Eles destacaram a necessidade de adicionar camadas ocultas com funções de ativação não lineares, criando o conceito de redes multicamadas (MLPs, do inglês *extitmulti-layer perceptrons*) (Minsky; Papert, 1969). Apesar dessa proposta, não havia um algoritmo eficiente para treinar essas redes até os anos 1980, quando o algoritmo de retropropagação foi introduzido (Rumelhart; Hinton; Williams, 1986), permitindo o treinamento de redes profundas.

Nos anos seguintes, avanços nas arquiteturas de redes, como as redes convolucionais (CNNs) e recorrentes (RNNs), culminaram no desenvolvimento das Redes LSTM (Long Short-

Term Memory) em 1997 (Hochreiter; Schmidhuber, 1997), projetadas para capturar dependências de longo prazo em sequências temporais.

Redes neurais são compostas por camadas de neurônios artificiais interconectados. Cada neurônio aplica uma transformação linear às entradas e, em seguida, uma função de ativação para introduzir não linearidade. As funções de ativação desempenham um papel crucial no desempenho do modelo, sendo a função ReLU utilizada neste trabalho.

A função de ativação ReLU (Rectified Linear Unit) foi inicialmente proposta por Fukushima (Fukushima, 1969) e posteriormente popularizada por Nair e Hinton (Nair; Hinton, 2010). A ReLU é definida por $f(x) = \max(0, x)$ e tem como principais vantagens a simplicidade computacional e a redução do problema de aprendizado em redes profundas.

Já as Redes Neurais Recorrentes (RNNs) são um tipo de rede neural projetada para lidar com dados sequenciais. A principal característica das RNNs é a presença de conexões recorrentes, que permitem que informações de etapas anteriores influenciem os estados atuais. Essa arquitetura é especialmente útil para problemas onde a ordem dos dados importa, como séries temporais e processamento de linguagem natural.

Cada unidade de uma RNN recebe a entrada atual x_t e o estado oculto anterior h_{t-1} no tempo t . O estado oculto, que pode ser interpretado como uma memória interna, é atualizado em cada passo de tempo, permitindo que a rede armazene informações relevantes ao longo da sequência. Podemos ver uma representação geral de sua arquitetura na Figura 5.

O estado h_t no tempo t é calculado como:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b)$$

onde W_h e W_x são matrizes de pesos treináveis, b é o vetor de viés, e σ é uma função de ativação não linear.

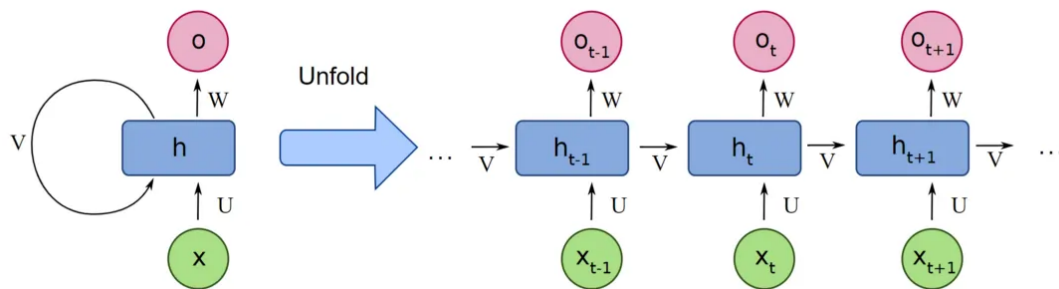


Figura 5 – Arquitetura básica de uma RNN. Fonte: (Kalita, 2022).

Apesar de serem poderosas, as RNNs simples enfrentam dificuldades para aprender dependências de longo prazo devido ao problema do **gradiente desvanecente**. Durante o treinamento, os gradientes podem se tornar extremamente pequenos, tornando difícil a atualização efetiva dos pesos para informações distantes na sequência. Esse problema limita a capacidade das RNNs de capturar padrões em sequências longas.

Para superar essa limitação, arquiteturas avançadas como as Redes LSTM (*Long Short-Term Memory*) e GRU (*Gated Recurrent Unit*) foram desenvolvidas, permitindo que as redes aprendam dependências de longo alcance de maneira mais eficaz.

Arquitetura LSTM

As redes neurais de memória de longa e curta duração (*Long Short-Term Memory* - LSTM) são uma variação das redes neurais recorrentes (*Recurrent Neural Networks* - RNN), projetadas para resolver o problema do gradiente desvanecente (Hochreiter; Schmidhuber, 1997). Diferentemente das RNNs tradicionais, as LSTM introduzem uma arquitetura que inclui portas de entrada, saída e esquecimento, permitindo que a rede controle de forma seletiva quais informações devem ser mantidas ou descartadas ao longo do tempo. Antes de explicar sobre cada porta, é importante entender sobre uma operação chamada produto de Hadamard.

O produto de Hadamard, ou *element-wise product*, $\odot$, é uma operação utilizada nas redes LSTM para combinar informações de diferentes portas. Dados dois vetores ou matrizes A e B de mesma dimensão, o produto de Hadamard é definido como:

$$(A \odot B)_{i,j} = A_{i,j} \cdot B_{i,j} \quad (2.17)$$

Essa operação é fundamental para atualizar o estado da célula c_t , permitindo que informações sejam combinadas de maneira eficiente entre as portas de entrada, esquecimento e saída (Hochreiter; Schmidhuber, 1997).

Desse modo, uma célula LSTM consiste em três portas principais: a porta de entrada, a porta de esquecimento e a porta de saída. Essas portas regulam o fluxo de informações dentro da célula, permitindo que a rede decida quais informações reter, descartar ou atualizar em cada passo do tempo. É essa combinação de decisões que permite às LSTM superarem as redes recorrentes simples em problemas de longa dependência temporal.

- **Porta de Esquecimento (f_t):** Essa porta decide quais informações do estado da célula anterior (C_{t-1}) devem ser descartadas. A operação matemática é:

$$f_t = \sigma(U_f x_t + V_f h_{t-1} + b_f), \quad (2.18)$$

onde σ é a função sigmoide, U_f e V_f são os pesos aplicados à entrada atual e ao estado oculto anterior, respectivamente, e b_f é o termo de viés associado à porta de esquecimento.

- **Porta de Entrada (i_t) e Estado da Célula Candidato ($\tilde{C}_t$):** Essas portas controlam quais novas informações serão armazenadas no estado da célula. As operações matemáticas são:

$$i_t = \sigma(U_i x_t + V_i h_{t-1} + b_i), \quad (2.19)$$

$$\tilde{C}_t = \tanh(U_C x_t + V_C h_{t-1} + b_C), \quad (2.20)$$

onde U_i , V_i , U_C , V_C são os pesos, e b_i , b_C são os termos de viés.

- **Estado da Célula** (C_t): O estado da célula é atualizado combinando as informações da porta de esquecimento e da porta de entrada. A equação é:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \quad (2.21)$$

onde f_t é o valor da porta de esquecimento, i_t é o valor da porta de entrada, e $\tilde{C}_t$ é o estado candidato da célula.

- **Porta de Saída** (o_t) e **Estado Oculto** (h_t): Essas operações controlam quais informações do estado da célula atual (C_t) serão usadas para gerar o estado oculto. As equações são:

$$o_t = \sigma(U_o x_t + V_o h_{t-1} + b_o), \quad (2.22)$$

$$h_t = o_t \odot \tanh(C_t), \quad (2.23)$$

onde U_o , V_o e b_o são os pesos e o termo de viés associados à porta de saída.

- **Previsão Final** ($\hat{y}_t$): Por fim, para realizar a previsão aplica-se uma transformação linear ao estado oculto h_t :

$$\hat{y}_t = W_{\text{out}} \cdot h_t + b_{\text{out}}, \quad (2.24)$$

onde W_{out} é um vetor de pesos e b_{out} é o viés associado à previsão.

A Figura 6 ilustra a arquitetura de uma célula LSTM, destacando suas principais portas e conexões internas. Neste trabalho, incluímos ainda a saída $\hat{y}_t$, representando a previsão gerada pelo modelo.

A seguir, ilustramos os cálculos realizados dentro de uma célula LSTM usando valores fictícios para entradas e pesos, simulando uma única iteração (época) de previsão. Suponha que o modelo utiliza 2 lags como entrada e o objetivo é encontrar uma previsão para um passo temporal t da sequência 2, 3 e 4. As variáveis iniciais são:

$$h_{t-1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad x_t = \begin{bmatrix} 2 \\ 3 \end{bmatrix}.$$

Para $t = 0$, temos $h_{t-1} = \mathbf{0}$ e $C_{t-1} = \mathbf{0}$.

Parâmetros:

$$U_f = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}, \quad V_f = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}, \quad b_f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Cálculo da porta de esquecimento:

$$U_f x_t = \begin{bmatrix} 2.5 \\ 2.5 \end{bmatrix}, \quad V_f h_{t-1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad f_t = \sigma(U_f x_t + V_f h_{t-1} + b_f) = \sigma \left(\begin{bmatrix} 2.5 \\ 2.5 \end{bmatrix} \right) \approx \begin{bmatrix} 0.924 \\ 0.924 \end{bmatrix}$$

Parâmetros:

$$U_i = U_C = \begin{bmatrix} 0.4 & 0.4 \\ 0.4 & 0.4 \end{bmatrix}, \quad V_i = V_C = \begin{bmatrix} 0.4 & 0.4 \\ 0.4 & 0.4 \end{bmatrix}, \quad b_i = b_C = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Porta de entrada e estado candidato:

$$U_i x_t = \begin{bmatrix} 2.0 \\ 2.0 \end{bmatrix}, \quad V_i h_{t-1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad i_t = \sigma(U_i x_t + V_i h_{t-1} + b_i) = \sigma \left(\begin{bmatrix} 2.0 \\ 2.0 \end{bmatrix} \right) \approx \begin{bmatrix} 0.881 \\ 0.881 \end{bmatrix}$$

$$\tilde{C}_t = \tanh(U_C x_t + V_C h_{t-1} + b_C) = \tanh \left(\begin{bmatrix} 2.0 \\ 2.0 \end{bmatrix} \right) \approx \begin{bmatrix} 0.964 \\ 0.964 \end{bmatrix}$$

Atualização da célula:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t = \begin{bmatrix} 0.881 \cdot 0.964 \\ 0.881 \cdot 0.964 \end{bmatrix} = \begin{bmatrix} 0.849 \\ 0.849 \end{bmatrix}$$

Parâmetros:

$$U_o = V_o = \begin{bmatrix} 0.6 & 0.6 \\ 0.6 & 0.6 \end{bmatrix}, \quad b_o = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Porta de saída e estado oculto:

$$o_t = \sigma(U_o x_t + V_o h_{t-1} + b_o) = \sigma \left(\begin{bmatrix} 3.0 \\ 3.0 \end{bmatrix} \right) \approx \begin{bmatrix} 0.952 \\ 0.952 \end{bmatrix}$$

$$h_t = o_t \odot \tanh(C_t) = \begin{bmatrix} 0.952 \cdot 0.693 \\ 0.952 \cdot 0.693 \end{bmatrix} = \begin{bmatrix} 0.659 \\ 0.659 \end{bmatrix}$$

Previsão:

$$\hat{y}_t = W_{\text{out}} \cdot h_t + b_{\text{out}} = 1.0 \cdot 0.659 + 0 = 0.659$$

Erro:

$$\text{Erro} = \frac{1}{2}(4 - 0.659)^2 = \frac{1}{2}(3.341)^2 \approx 5.585$$

Resumo dos Resultados (1 Epoch)

- Entrada: $x_t = [2, 3]$
- Previsão: $\hat{y}_t \approx 0.659$
- Valor real: $y_t = 4$
- Erro: ≈ 5.585

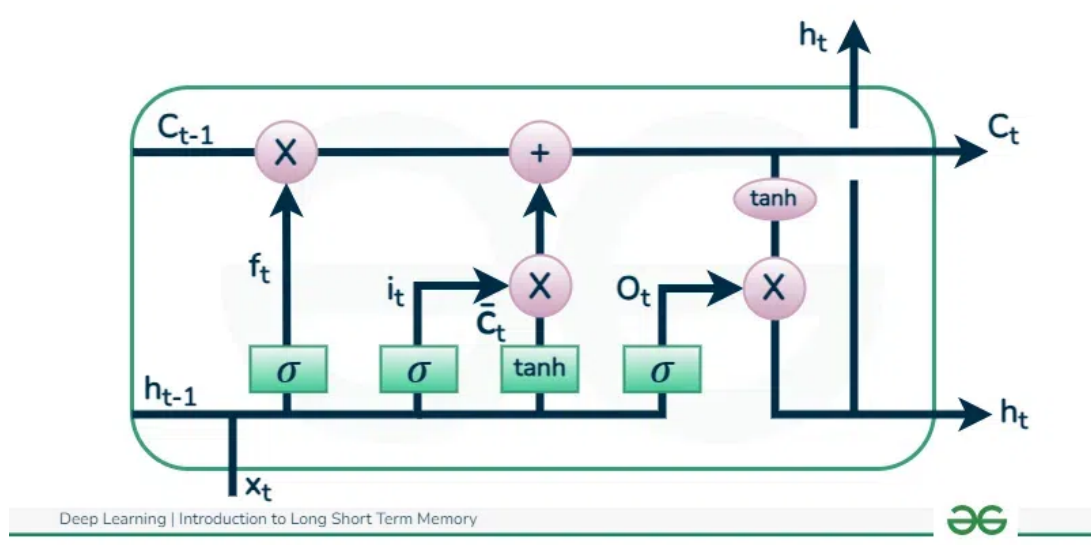


Figura 6 – Arquitetura básica de uma rede neural LSTM. Fonte: (GeeksforGeeks, 2023).

O modelo ainda está longe da previsão correta e precisaria de várias iterações para ajustar os pesos. Como estamos considerando a primeira iteração, os pesos utilizados são arbitrários e não foram ajustados por treinamento.

Ao trabalhar com redes LSTM, é crucial realizar o escalonamento dos dados, geralmente utilizando a normalização min-max, que transforma os dados para um intervalo fixo (geralmente entre 0 e 1). Esse procedimento acelera o treinamento e melhora a estabilidade numérica do modelo, evitando problemas como gradientes explosivos ou desvanecentes (Goodfellow; Bengio; Courville, 2016). O escalonamento também garante que todas as variáveis estejam na mesma escala, o que é particularmente importante em redes sensíveis às variações de magnitude, como as LSTM.

Estimação dos Parâmetros

Os parâmetros W , U , e b de uma LSTM são atualizados durante o treinamento do modelo por meio de métodos de otimização baseados em gradiente, como o *Gradient Descent* ou variantes mais avançadas, como o *Adam Optimizer* (Kingma; Ba, 2014). O processo de treinamento minimiza uma função de custo, como o erro quadrático médio (EQM) em problemas de regressão, ou a entropia cruzada em tarefas de classificação. A função de custo J é definida como:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^N L(y_i, \hat{y}_i), \quad (2.25)$$

onde θ representa os parâmetros treináveis do modelo, L é a função de perda (como EQM ou entropia cruzada), y_i são os valores reais e $\hat{y}_i$ as previsões do modelo.

O treinamento utiliza a retropropagação através do tempo (*Backpropagation Through Time* - BPTT), que computa os gradientes dos parâmetros considerando as dependências temporais das sequências. Esse processo envolve:

1. Desenrolar a Rede no Tempo: A rede é "desenrolada" ao longo de todos os passos temporais da sequência, permitindo que as conexões recorrentes sejam tratadas como camadas adicionais na rede.

2. Cálculo do Gradiente: Os gradientes são acumulados ao longo do tempo, considerando a influência de cada passo anterior nas previsões futuras. Isso é feito aplicando a regra da cadeia às conexões temporais.

3. Atualização dos Parâmetros: Após calcular os gradientes, os parâmetros W e b são ajustados usando uma regra de atualização baseada no otimizador escolhido. Para o Adam, por exemplo, a atualização segue:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{m_t}{\sqrt{v_t} + \epsilon}, \quad (2.26)$$

onde m_t e v_t são as estimativas dos momentos de primeira e segunda ordem dos gradientes, η é a taxa de aprendizado, e ϵ é um termo para estabilidade numérica.

4. Prevenção de Gradientes Explosivos ou Desvanecentes: Para evitar gradientes explosivos, pode-se aplicar *gradient clipping*, que limita o valor máximo dos gradientes. As LSTM, por construção, mitigam o problema de gradientes desvanecentes graças ao uso de portas que regulam o fluxo de informações.

5. Iteração Até Convergência: O processo é repetido por várias épocas até que a função de custo J convirja para um valor mínimo aceitável ou que o modelo alcance um desempenho satisfatório no conjunto de validação.

Essa abordagem permite que as LSTM aprendam padrões temporais, ajustando seus parâmetros para minimizar o erro entre as previsões e os valores reais.

As principais vantagens das redes LSTM incluem sua capacidade de lidar com dependências de longo prazo e mitigar o problema do gradiente desvanecente. No entanto, as LSTM podem ser computacionalmente intensivas, exigindo mais tempo de treinamento e recursos computacionais. Além disso, elas frequentemente necessitam de grandes volumes de dados para um treinamento eficaz, o que pode ser uma limitação em cenários com dados escassos (Hochreiter; Schmidhuber, 1997).

2.3 Análise Espacial

A análise espacial trata da exploração e modelagem de dados que possuem uma estrutura espacial subjacente. Diferentes tipos de dados espaciais são usados em aplicações práticas, incluindo dados agrupados por áreas, geoestatísticos e dados na forma de pontos (Cressie, 1993). No contexto deste trabalho, os dados utilizados são agrupados por áreas, que correspondem às cidades do estado de São Paulo, ou seja, observações que representam os registros agregados naquela região. Neste caso, estamos lidando com a média móvel semanal da incidência de casos de Covid-19 para cada cidade.

É importante destacar que os modelos espaciais apresentados serão aplicados em cada momento do tempo, permitindo analisar as dependências espaciais dinâmicas.

Todo processo espacial está sujeito à **Lei de Tobler**, também conhecida como a Primeira Lei da Geografia:

"Tudo depende de todo o restante, porém o que está mais próximo depende mais do que aquilo que está mais distante."

Essa ideia é central na análise espacial, capturando como locais geograficamente próximos afetam uns aos outros.

Os dados espaciais podem ser classificados em três categorias principais:

- **Dados agrupados por áreas:** Observações agregadas em unidades geográficas discretas, como cidades ou estados.
- **Dados geoestatísticos:** Valores observados em locais específicos dentro de uma região contínua, como medições de poluição.
- **Dados pontuais:** Localizações específicas de eventos, como a ocorrência de crimes ou localizações de árvores (Cressie, 1993).

No contexto deste trabalho, o foco será em dados agrupados por áreas, pois a análise considera a incidência de Covid-19 agregada por município.

Modelos Espaciais Clássicos

Os modelos espaciais clássicos buscam capturar a correlação existente entre observações que estão próximas geograficamente. Esses modelos consideram que a localização de uma observação pode influenciar a ocorrência de eventos em regiões vizinhas. No contexto deste trabalho, o foco será no Modelo de Autoregressão Espacial (SAR, *Spatial Autoregressive Model*), utilizado para capturar dependências espaciais explícitas (Anselin, 1988).

O modelo SAR é definido como:

$$y = \rho Ay + X\beta + \epsilon, \quad (2.27)$$

onde y é o vetor de observações dependentes, ρ é o parâmetro de dependência espacial, A é a matriz de adjacência espacial, X é a matriz de variáveis independentes, β é o vetor de coeficientes, e ϵ é o termo de erro.

Matriz de adjacência espacial (matriz de ponderação espacial)

A matriz de pesos espaciais A representa a estrutura de vizinhança entre as regiões e pode ser construída de diferentes formas, como baseando-se em adjacências (vizinho comum)

ou distâncias geográficas (LeSage; Pace, 2009). A interpretação do parâmetro ρ é crucial, pois ele indica o grau de influência que as observações em regiões vizinhas exercem umas sobre as outras.

Os elementos de A , denotados por a_{ij} , indicam a proximidade ou conectividade entre as localizações i e j . A matriz A pode ser normalizada de forma que cada linha some 1, assegurando que os pesos são relativos, representando:

$$\tilde{A}_{ij} = \frac{A_{ij}}{\sum_k A_{ik}}, \quad \forall i.$$

Não há um procedimento único para a escolha de A . Diferentes métodos podem ser aplicados, como:

- **Matriz de contiguidade Queen:** Representa adjacência de polígonos baseando-se em bordas e vértices compartilhados.
- **Matriz baseada nos k -vizinhos mais próximos (KNN):** Conecta cada unidade espacial aos k vizinhos mais próximos com base na distância geográfica ou métrica.

Neste estudo, utilizamos a matriz de contiguidade Queen por sua capacidade de capturar conexões espaciais diretas. Testes comparativos entre diferentes tipos de matrizes A (e.g., KNN e Queen) podem ser realizados para avaliar o impacto na modelagem.

Baumont (2004) diz sobre como buscar a melhor matriz

Existem outras opções ainda, além das mencionadas. Por exemplo, Fritz, Dorigatti e Rügamer (2022) utilizaram dados de mobilidade humana e conectividade social fornecidos pelo Facebook Data for Good para construir grafos em um contexto epidemiológico também.

Índice de Moran

O índice de Moran (I) é uma estatística utilizada para medir a autocorrelação espacial em um conjunto de dados geográficos. Ele avalia se os valores de uma variável observada exibem padrões espaciais, ou seja, se há similaridade ou dissimilaridade entre valores de localizações próximas.

A fórmula do índice global é:

$$I = \frac{n \cdot \sum_{i=1}^n \sum_{j=1}^n \tilde{A}_{ij} (y_i - \bar{y})(y_j - \bar{y})}{(\sum_{i=1}^n (y_i - \bar{y})^2) \cdot \sum_{i=1}^n \sum_{j=1}^n \tilde{A}_{ij}}.$$

Onde:

- $\tilde{A}_{ij}$: matriz de adjacência normalizada.
- y_i, y_j : valores observados nas localizações i e j .

- $\bar{y}$: média de y .

Quando $I > 0$ indica autocorrelação positiva (valores similares agrupados), enquanto $I < 0$ indica autocorrelação negativa (valores dissimilares agrupados). Quando $I \approx 0$, a distribuição espacial é aleatória.

No contexto local, índices de Moran locais (LISA) identificam clusters e padrões específicos em subconjuntos do espaço. Porém, o LISA não foi utilizado neste trabalho.

2.3.1 Modelo de Autoregressão Espacial Simultâneo (SAR)

O modelo de Autoregressão Espacial Simultâneo (SAR) é utilizado para capturar dependências espaciais em dados geográficos. Este modelo considera que o valor de uma variável em uma determinada localização é influenciado pelos valores da mesma variável em localizações vizinhas, representados por uma matriz de adjacência $\mathbf{A}$ (Anselin, 1988). A formulação matemática do modelo SAR é baseada na incorporação de um termo autoregressivo espacial na equação de regressão. Possui o nome “simultâneo” pois a matriz de dependência espacial é inserida diretamente no modelo.

$$\mathbf{y} = \rho \mathbf{A} \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \boldsymbol{\epsilon}, \quad (2.28)$$

onde:

- $\mathbf{y}$: vetor de observações.
- $\mathbf{A}$: matriz de dependência espacial normalizada.
- ρ : coeficiente de autocorrelação espacial.
- $\mathbf{X}$: matriz de variáveis explicativas.
- $\boldsymbol{\beta}$: vetor de coeficientes das variáveis explicativas.
- $\boldsymbol{\epsilon}$: termos de erro ($\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$).

Na versão pura do modelo SAR, $\mathbf{X}$ não está presente:

$$\mathbf{y} = \rho \mathbf{A} \mathbf{y} + \boldsymbol{\epsilon}.$$

Este modelo foi utilizado para avaliar a dependência espacial isoladamente.

O coeficiente ρ indica a força e a direção da dependência espacial. Um valor positivo de ρ sugere que valores altos (ou baixos) em uma localização tendem a ser associados com valores altos (ou baixos) em localizações vizinhas.

A estimação dos parâmetros no modelo SAR (Autoregressão Espacial) é mais complexa do que na regressão linear tradicional devido à presença do termo autoregressivo espacial. Métodos como a Máxima Verossimilhança (*Maximum Likelihood*) e Mínimos Quadrados Generalizados (*Generalized Least Squares*) são utilizados para estimar os parâmetros ρ e β (LeSage; Pace, 2009).

Estimação por Máxima Verossimilhança

Para estimar os parâmetros do modelo SAR, o método de máxima verossimilhança é utilizado. O modelo é definido como:

$$\mathbf{y} = \rho \mathbf{A} \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \boldsymbol{\epsilon},$$

onde $\mathbf{y}$ é o vetor de observações dependentes, ρ é o coeficiente de dependência espacial, $\mathbf{A}$ é a matriz de pesos espaciais, $\mathbf{X}$ é a matriz de variáveis independentes, $\boldsymbol{\beta}$ é o vetor de coeficientes, e $\boldsymbol{\epsilon}$ é o termo de erro branco com variância σ^2 .

A log-verossimilhança do modelo SAR é dada por:

$$L(\rho, \beta, \sigma^2) = -\frac{n}{2} \ln(2\pi\sigma^2) + \ln |I - \rho A| - \frac{1}{2\sigma^2} (\mathbf{y} - \rho \mathbf{A} \mathbf{y} - \mathbf{X} \boldsymbol{\beta})' (\mathbf{y} - \rho \mathbf{A} \mathbf{y} - \mathbf{X} \boldsymbol{\beta}),$$

onde $|I - \rho A|$ é o determinante do operador espacial.

O processo de estimação segue os passos a seguir:

1. Estimativa Inicial de ρ : Realiza-se uma busca por ρ que maximize $\ln |I - \rho A| - \frac{1}{2\sigma^2} (\mathbf{y} - \rho \mathbf{A} \mathbf{y} - \mathbf{X} \boldsymbol{\beta})' (\mathbf{y} - \rho \mathbf{A} \mathbf{y} - \mathbf{X} \boldsymbol{\beta})$. Métodos como busca em grade ou algoritmos numéricos (ex.: Newton-Raphson) são utilizados para encontrar o ρ ótimo.

2. Estimativa de β : Após fixar ρ , os coeficientes β são estimados resolvendo:

$$\hat{\beta} = (X'(I - \rho A)'(I - \rho A)X)^{-1} X'(I - \rho A)'(I - \rho A)y.$$

3. Estimativa de σ^2 : O erro quadrático médio é usado para estimar σ^2 :

$$\hat{\sigma}^2 = \frac{1}{n} ((\mathbf{y} - \rho \mathbf{A} \mathbf{y})' (\mathbf{y} - \rho \mathbf{A} \mathbf{y})).$$

4. Iteração e Convergência: Os passos acima são iterados até que os valores de ρ , β , e σ^2 estabilizem (diferença abaixo de um limiar ε).

Também existem alternativas em que é feito uma aproximação da função de log-verossimilhança, mas não será abordado neste trabalho.

O modelo SAR é utilizado em várias áreas, incluindo economia regional, epidemiologia e ciência ambiental, para analisar fenômenos que exibem dependência espacial. Por exemplo, ele pode ser utilizado para estudar a disseminação de doenças infecciosas, onde a incidência de casos em uma região pode ser influenciada pelos casos em regiões vizinhas (Cressie, 1993).

2.3.2 Redes Neurais de Grafos (GNN)

Um grafo é definido como um conjunto de nós (ou vértices) conectados por arestas (ou *links*). Formalmente, um grafo G é representado como $G = (V, E)$, onde V é o conjunto de nós e E é o conjunto de arestas que conectam pares de nós. Em muitas aplicações, cada nó e aresta podem possuir atributos adicionais, como características dos nós e pesos das arestas.

Redes Neurais de Grafos (*Graph Neural Networks* - GNN) é uma classe de modelos de aprendizado profundo projetados para operar em dados estruturados em grafos.

Diferentemente das redes neurais tradicionais que lidam com dados tabulares ou sequenciais, as GNNs são capazes de capturar dependências complexas e relacionamentos entre os nós de um grafo, tornando-as particularmente úteis para uma variedade de tarefas, como classificação de nós, previsão de arestas e classificação de grafos (Wu et al., 2021).

Brüggemann (2023) destacaram a utilização de Redes Neurais em Grafo para modelar surtos de doenças como a COVID-19, mostrando que a representação de dados epidemiológicos em grafos pode capturar padrões de propagação entre regiões vizinhas.

As GNNs operam em grafos através de um processo iterativo de agregação e atualização de informações dos nós vizinhos. Este processo pode ser dividido em três etapas principais:

- **Agregação:** Cada nó coleta informações de seus nós vizinhos.
- **Atualização:** As informações agregadas são combinadas com a representação atual do nó para atualizar seu estado.
- **Leitura:** Após várias iterações de agregação e atualização, as representações finais dos nós são usadas para realizar a tarefa desejada, como classificação ou regressão.

As GNNs são particularmente úteis para modelar dependências espaciais em dados geográficos, onde as localizações podem ser representadas como nós de um grafo e as relações espaciais como arestas. Um exemplo comum é a utilização de redes convolucionais de grafos (*Graph Convolutional Networks* - GCN) para prever a disseminação de doenças infecciosas entre diferentes regiões (Kipf; Welling, 2017).

No contexto da previsão da COVID-19, as GNNs podem ser usadas para modelar a propagação do vírus entre diferentes regiões. Cada região é representada como um nó e as fronteiras entre as regiões são representadas como arestas no grafo. As características dos nós podem incluir informações demográficas, taxas de infecção passadas, e outras variáveis relevantes. A Figura 7 ilustra a construção do grafo para os municípios do estado de São Paulo, no qual as conexões entre os nós representam fronteiras geográficas.

As GNNs são poderosas porque podem capturar padrões de dependência espacial que são difíceis de modelar com técnicas tradicionais. No entanto, essas redes podem ser computaci-

Grafo de Contiguidade tipo "Queen" - Estado de São Paulo

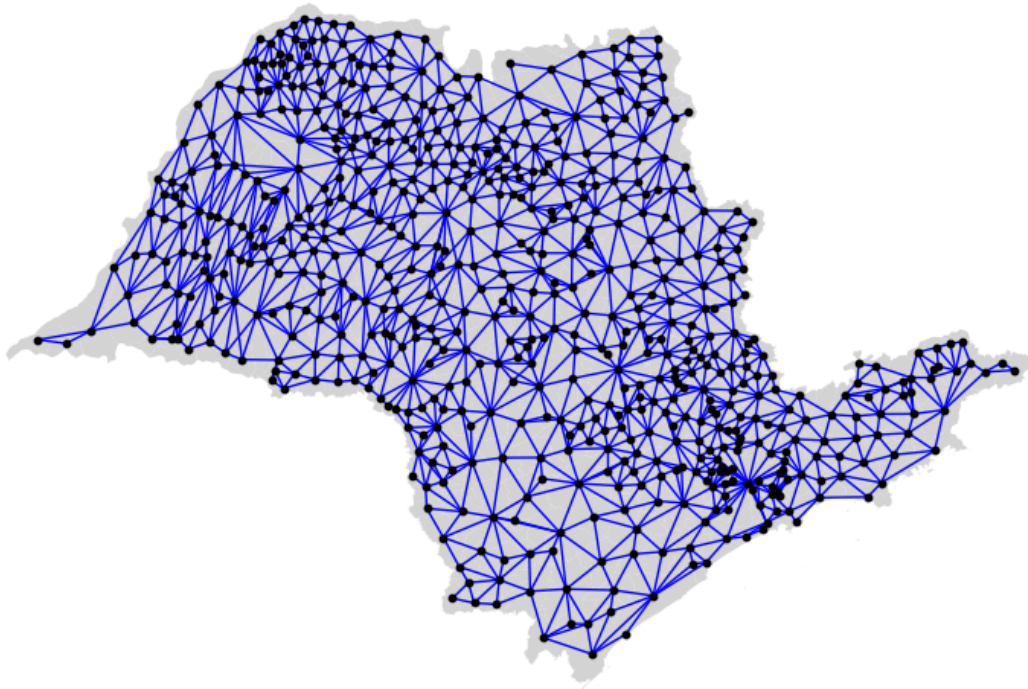


Figura 7 – Grafo dos municípios de São Paulo conectado por suas respectivas fronteiras. Fonte: Elaborado pelo autor.

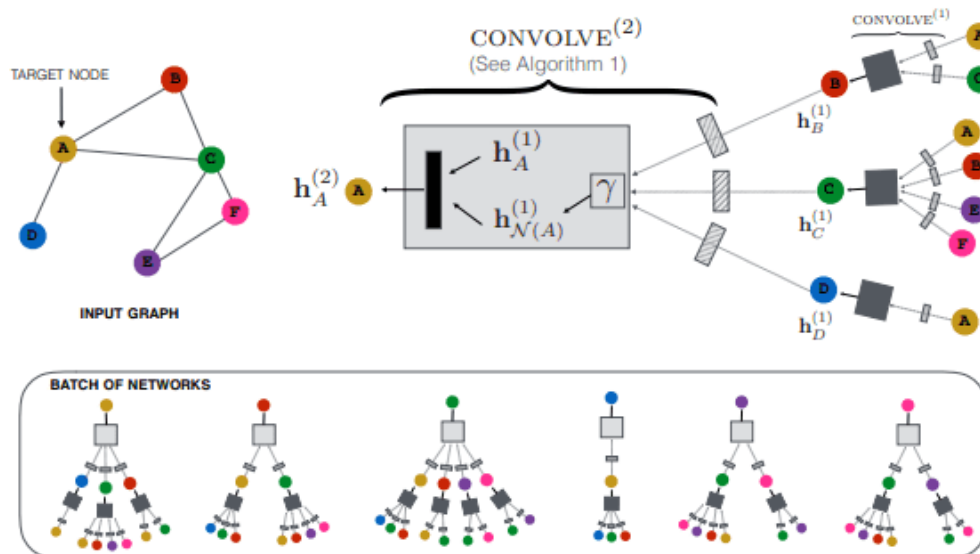


Figura 8 – Exemplo visual de uma arquitetura de uma rede GNN. Fonte: (Summer, 2023).

onalmente intensivas e requerem uma quantidade significativa de dados de treinamento para sua implementação efetiva. A Figura 8 apresenta um exemplo visual de como as informações são agregadas e processadas em uma arquitetura de GNN.

Há diversas abordagens para implementar um modelo de GNN, tais como o uso de redes de atenção, redes por propagação de mensagens, entre outros. A seguir detalha-se a metodologia adotada neste trabalho:

1. Representação do Grafo

O grafo é definido como $G = (V, E)$, onde:

- V é o conjunto de nós (cidades ou regiões) com $|V| = n$.
- E é o conjunto de arestas que representam as conexões entre nós.
- A é a matriz de adjacência ponderada, baseada em similaridade, conectividade ou distâncias geográficas.

Após a normalização por linha (*row-normalization*), cada entrada de A é dada por:

$$A'_{ij} = \frac{A_{ij}}{\sum_k A_{ik}}, \quad \text{com } \sum_j A'_{ij} = 1, \forall i.$$

Essa normalização assegura que as somas das conexões de cada nó sejam iguais a 1, balanceando a propagação de mensagens.

2. Propagação de Mensagens

A GNN utiliza o mecanismo de propagação de mensagens (*Message Passing*) para atualizar as representações dos nós com base nas informações dos seus vizinhos.

2.1. Transformação Linear

Cada nó i possui um vetor de covariáveis $\mathbf{x}_i$. Na camada l , essas covariáveis são transformadas por uma operação linear:

$$\mathbf{z}_i^{(l)} = \mathbf{x}_i^{(l)} W^{(l)} + \mathbf{b}^{(l)},$$

onde:

- $W^{(l)}$ é a matriz de pesos, ajustada durante o treinamento.
- $\mathbf{b}^{(l)}$ é o vetor de vies, também ajustado no treinamento.

2.2. Agregação de Mensagens

As mensagens das vizinhanças de cada nó i são agregadas usando a matriz normalizada A' :

$$\mathbf{h}_i^{(l+1)} = \sum_{j \in \mathcal{N}(i)} A'_{ij} \mathbf{z}_j^{(l)},$$

onde $\mathcal{N}(i)$ é o conjunto de vizinhos do nó i . Essa operação evita instabilidades durante o aprendizado e balanceia a influência dos vizinhos.

2.3. Atualização das covariáveis

As novas covariáveis do nó i na camada $l + 1$ são obtidas aplicando uma função de ativação (ReLU) às mensagens agregadas:

$$\mathbf{x}_i^{(l+1)} = \text{ReLU}(\mathbf{h}_i^{(l+1)}).$$

3. Arquitetura Completa

O modelo possui duas camadas de propagação de mensagens:

- **Camada 1:** Reduz as dimensões iniciais das covariáveis:

$$\mathbf{h}_i^{(1)} = \text{ReLU} \left(\sum_{j \in \mathcal{N}(i)} A'_{ij} (\mathbf{x}_j^{(0)} W^{(0)} + \mathbf{b}^{(0)}) \right).$$

- **Camada 2:** Projeta o espaço latente para a dimensão de saída:

$$\mathbf{h}_i^{(2)} = \sum_{j \in \mathcal{N}(i)} A'_{ij} (\mathbf{h}_j^{(1)} W^{(1)} + \mathbf{b}^{(1)}).$$

A escolha de duas camadas reflete um equilíbrio entre capacidade de aprendizado e custo computacional.

4. Função de Perda

A função de perda combina dois componentes:

1. **Erro Quadrático Médio (EQM):**

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

onde y_i é o valor real e $\hat{y}_i$ é o valor previsto.

2. **Índice de Moran (I):**

$$I = \frac{n \cdot \sum_{i=1}^n \sum_{j=1}^n A'_{ij} (r_i - \bar{r})(r_j - \bar{r})}{(\sum_{i=1}^n (r_i - \bar{r})^2) \cdot \sum_{i=1}^n \sum_{j=1}^n A'_{ij}},$$

onde:

- $r_i = y_i - \hat{y}_i$ é o resíduo.
- $\bar{r}$ é a média dos resíduos.

A perda total é uma combinação ponderada:

$$\mathcal{L} = \alpha \cdot \text{EQM} + \beta \cdot I,$$

onde α e β são pesos ajustáveis. Essa abordagem garante tanto a precisão preditiva quanto a coerência espacial dos resíduos.

Se necessário, o índice de Moran local pode ser aplicado para avaliar padrões espaciais mais específicos em regiões distintas.

5. Considerações

- A matriz W' , normalizada por linha, já balanceia a influência de cada vizinho, eliminando a necessidade de normalizações adicionais dentro da GNN.
- O modelo é ideal para capturar dependências espaciais em tarefas geográficas e pode ser adaptado para diferentes contextos com dados espaciais.
- O custo computacional pode ser elevado, especialmente em grafos grandes. Estratégias como amostragem de subgrafos podem ser utilizadas para otimizar o treinamento.

A estimativa dos parâmetros em Redes Neurais de Grafos (GNNs) é realizada através do treinamento dos pesos da rede, que são ajustados iterativamente para minimizar uma função de perda.

Os pesos $W^{(l)}$ e os vetores de viés $b^{(l)}$ para cada camada l são inicializados aleatoriamente ou com métodos como inicialização de Xavier ou He.

Para cada nó, as representações são calculadas iterativamente com base em seus vizinhos, usando as fórmulas de agregação e atualização:

$$h_v^{(k+1)} = \sigma \left(W^{(k)} \cdot \sum_{u \in \mathcal{N}(v)} W'_{vu} h_u^{(k)} + b^{(k)} \right).$$

A função de perda combina o erro preditivo (ex.: EQM) e, em alguns casos, penalizações para manter coerência espacial, como o Índice de Moran:

$$\mathcal{L} = \alpha \cdot \text{EQM} + \beta \cdot I,$$

onde I é o Índice de Moran calculado nos resíduos.

O gradiente da função de perda $\nabla \mathcal{L}$ é calculado em relação aos pesos W e b utilizando o algoritmo de retropropagação, estendido para a estrutura do grafo.

Os pesos são atualizados utilizando métodos de otimização, através do método Adam.

$$W^{(l)} \leftarrow W^{(l)} - \eta \cdot \nabla \mathcal{L},$$

onde η é a taxa de aprendizado.

O processo de propagação direta, cálculo da perda e retropropagação é repetido até que a função de perda convirja (mudança pequena entre iterações) ou um número máximo de épocas seja atingido.

2.4 Agregando Informações Espaciais aos Modelos Temporais

A integração de informações espaciais em modelos temporais busca capturar dependências geográficas e temporais simultaneamente, permitindo uma análise mais completa de fenômenos espaciais. Essa abordagem considera que eventos observados em regiões próximas podem influenciar diretamente os resultados temporais e vice-versa. No contexto deste estudo, os modelos temporais incorporam variáveis exógenas derivadas de modelos espaciais, proporcionando uma visão integrada da dinâmica espaço-temporal dos dados visando uma melhoria no desempenho das previsões.

2.4.1 Modelo ARIMAX

O modelo ARIMAX é uma extensão do modelo ARIMA que inclui variáveis exógenas X_t para capturar fatores externos que afetam a série temporal (Box et al., 2015; Hyndman; Athanasopoulos, 2018). Ele é representado pela seguinte equação:

$$\Phi(B)(1 - B)^d y_t = \Theta(B)\epsilon_t + \beta X_t,$$

onde:

- $\Phi(B)$: polinômio autoregressivo.
- $\Theta(B)$: polinômio de média móvel.
- X_t : variável exógena.
- β : coeficiente das variáveis exógenas.

A inclusão de X_t permite incorporar informações externas, como dependências espaciais. Neste estudo, X_t representa a resposta de um modelo de dependência espacial (como o modelo SAR). Essa inclusão possibilita avaliar se as informações espaciais melhoram a previsão dos modelos temporais. Estudos indicam que variáveis exógenas podem melhorar significativamente a capacidade preditiva quando fatores externos influenciam a dinâmica temporal (Box et al., 2015; Hyndman; Athanasopoulos, 2018).

A estimação dos parâmetros do ARIMAX é realizada por meio da maximização da log-verossimilhança do modelo, dada por:

$$L(\Phi, \Theta, \beta) = -\frac{N}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{t=1}^N (y_t - \hat{y}_t)^2,$$

onde $\hat{y}_t = \Phi(B)(1 - B)^d y_t - \Theta(B)\epsilon_t - \beta X_t$.

Assim como no modelo ARIMA comum, a maximização da função de log-verossimilhança é um problema de otimização não linear, dependendo de métodos numéricos, como o Newton-Raphson ou L-BFGS.

Este método garante que os parâmetros estimados maximizem a adequação do modelo aos dados observados, incorporando o impacto das variáveis exógenas X_t .

2.4.2 Regressão Dinâmica

A regressão dinâmica combina uma regressão linear clássica com erros que seguem um processo ARIMA, permitindo modelar tanto o impacto de variáveis explicativas externas quanto a estrutura temporal dos resíduos. A equação geral é:

$$y_t = \beta_0 + \beta_1 X_{1,t} + \beta_2 X_{2,t} + \cdots + \beta_k X_{k,t} + \eta_t,$$

$$(1 - \sum_{i=1}^p \phi_i B^i)(1 - L)^d \eta_t = (1 + \sum_{j=1}^q \theta_j B^j) \epsilon_t,$$

onde:

- y_t : variável dependente no tempo t .
- β_0 : intercepto.
- $\beta_1, \dots, \beta_k$: coeficientes das variáveis explicativas $X_{1,t}, X_{2,t}, \dots, X_{k,t}$.
- $ARIMA(p, d, q)$: componente ARIMA que captura a dependência temporal nos resíduos.
- ϵ_t : termo de erro branco.

Essa abordagem é particularmente útil para capturar padrões temporais não explicados pelas variáveis explicativas X_t , ao mesmo tempo em que incorpora o impacto dessas variáveis no modelo. Por exemplo, em uma aplicação espaço-temporal, X_t pode incluir informações espaciais derivadas de modelos como o SAR, permitindo integrar dinamicamente as dependências espaciais no processo de modelagem (Box et al., 2015; Hyndman; Athanasopoulos, 2018).

A estimação dos parâmetros neste modelo é um pouco diferente, pois precisamos estabelecer inicialmente os parâmetros do modelo linear e depois os parâmetros do modelo ARIMA. A estimação segue os seguintes passos:

1. **Ajuste Inicial da Regressão Linear:** Realize uma estimação de modelo de regressão linear para os parâmetros $\beta_0, \beta_1, \dots, \beta_k$, ignorando temporariamente a estrutura ARIMA nos resíduos. No caso, teremos o coeficiente linear e o coeficiente angular para estimar, já que estaremos utilizando apenas uma variável exógena, que é a dependência espacial.
2. **Modelagem dos Resíduos:** Calcule os resíduos $\eta_t = y_t - (\beta_0 + \beta_1 X_{1,t} + \dots + \beta_k X_{k,t})$ e ajuste um modelo ARIMA nos resíduos.
3. **Integração dos Componentes:** Combine os resultados dos passos anteriores e ajuste iterativamente os parâmetros β , ϕ , e θ para maximizar a log-verossimilhança do modelo completo:
$$L(\beta, \phi, \theta) = -\frac{N}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{t=1}^N (\eta_t - \hat{\eta}_t)^2.$$
4. **Convergência:** Continue até que os parâmetros estimados se estabilizem.

Que também utiliza os mesmos métodos numéricos para a estimação de um modelo ARIMA, como o Newton-Raphson ou L-BFGS.

2.4.3 Arquitetura GNN+LSTM

A arquitetura GNN+LSTM combina a modelagem de dependências espaciais e temporais em um *framework* integrado, permitindo capturar de forma eficiente as relações espaço-temporais de dados como a incidência de COVID-19. O modelo é composto por duas principais componentes: uma Rede Neural Gráfica (GNN) para explorar dependências espaciais entre diferentes localizações geográficas e uma Rede Neural do tipo Long Short-Term Memory (LSTM) para modelar dependências temporais. Essa integração aproveita o poder das GNNs para embutir informações espaciais e das LSTMs para prever a evolução temporal com base nessas informações enriquecidas.

Estudos recentes, como o de [Panagopoulos, Nikolentzos e Vazirgiannis \(2021b\)](#), aplicaram Redes Neurais de Grafos (GNN) e combinações como GNN+LSTM para prever a evolução da COVID-19, destacando a importância de modelar relações espaciais e temporais simultaneamente.

Técnicas que integram GNNs com redes LSTM são eficazes para modelar dependências espaço-temporais simultâneas, permitindo realizar tarefas preditivas em larga escala ([Li et al., 2023](#)).

A GNN é responsável por gerar representações vetoriais espaciais para cada localidade, representada como um nó em um grafo. A matriz de adjacência normalizada, $\tilde{\mathbf{A}}$, e as covariáveis dos nós, $\mathbf{X}$, são os insumos principais do modelo. O processo de aprendizado segue a equação:

$$\mathbf{H}^{(k)} = \sigma(\tilde{\mathbf{A}}\mathbf{H}^{(k-1)}\mathbf{W}^{(k)}), \quad (2.29)$$

onde:

- $\mathbf{H}^{(k)}$ é a representação vetorial dos nós na camada k ,
- $\mathbf{W}^{(k)}$ são os pesos treináveis da camada k ,
- σ é uma função de ativação não-linear (e.g., ReLU).

Após K camadas, as representações vetoriais finais $\mathbf{H}^{(K)}$ encapsulam a informação espacial relevante para cada nó.

As representações vetoriais $\mathbf{h}_i$ gerados pela GNN para uma localidade i são concatenados com os dados temporais $\mathbf{y}_i^t$ e servem como entrada para a LSTM. Cada entrada no tempo t é representada como:

$$\mathbf{z}_i^t = [\mathbf{y}_i^t, \mathbf{h}_i]. \quad (2.30)$$

O LSTM atualiza seus estados de acordo com:

1. Porta de esquecimento:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{z}_t] + \mathbf{b}_f), \quad (2.31)$$

2. Porta de entrada e estado da célula intermediária:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{z}_t] + \mathbf{b}_i), \quad (2.32)$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{z}_t] + \mathbf{b}_c), \quad (2.33)$$

3. Atualização do estado da célula:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \quad (2.34)$$

4. Porta de saída e estado oculto:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{z}_t] + \mathbf{b}_o), \quad (2.35)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \quad (2.36)$$

O estado oculto $\mathbf{h}_t$ no último passo temporal é usado para prever o próximo valor na série $\hat{\mathbf{y}}_i^{t+1}$.

O modelo funciona através dos seguintes passos:

1. **Extração Espacial:** A GNN gera representações vetoriais $\mathbf{H}^{(K)}$ para todos os nós.
2. **Concatenar representações vetoriais com Séries Temporais:** As representações vetoriais de cada nó são concatenadas às sequências temporais correspondentes.
3. **Previsão Temporal:** O LSTM utiliza as sequências enriquecidas para prever os próximos valores temporais.

4. **Previsão Iterativa:** Para múltiplos passos, as previsões anteriores alimentam as próximas entradas.

A estimativa de parâmetros na arquitetura GNN+LSTM combina técnicas de otimização aplicadas às duas principais componentes do modelo: a Rede Neural Gráfica (GNN) e a Long Short-Term Memory (LSTM). O processo de estimativa é realizado iterativamente, utilizando métodos baseados em gradiente para ajustar os pesos e vieses das camadas GNN e LSTM, de forma a minimizar a função de perda do modelo.

Primeiramente, os pesos da GNN ($W^{(k)}$) e da LSTM (W_f, W_i, W_c, W_o) são inicializados com valores aleatórios. A função de perda combina termos para capturar a precisão preditiva e a coerência espacial:

$$J(\theta) = \alpha \cdot \text{EQM}(\theta) + \beta \cdot \mathcal{R}(\theta), \quad (2.37)$$

onde α e β são pesos ajustáveis. O erro quadrático médio (EQM) avalia a diferença entre os valores reais y_t e previstos $\hat{y}_t$, enquanto a regularização $\mathcal{R}(\theta)$ pode incluir penalizações nos pesos ou medidas como o índice de Moran para manter consistência espacial.

A retropropagação através do tempo (BPTT) é usada para ajustar os pesos da LSTM. Esse processo envolve:

- Desenrolar a LSTM ao longo da sequência temporal;
- Calcular os gradientes dos parâmetros $W_f, W_i, \dots$ com base na regra da cadeia;
- Atualizar os parâmetros usando otimizadores como Adam ou SGD.

Os gradientes dos pesos da GNN ($W^{(k)}$) são calculados com base no impacto das representações vetoriais gerado nas previsões da LSTM. O cálculo é feito propagando os erros da função de perda através das camadas da GNN:

- A retropropagação começa na última camada K da GNN e segue camada por camada até a entrada;
- O gradiente para cada camada k é computado considerando a matriz de adjacência $\tilde{A}$ e as representações vetoriais intermediárias $H^{(k)}$.

Os pesos de ambas as componentes (GNN e LSTM) são ajustados simultaneamente usando um otimizador, como Adam. A regra de atualização para um peso genérico θ é dada por:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{m_t}{\sqrt{v_t + \epsilon}}, \quad (2.38)$$

onde η é a taxa de aprendizado, m_t é o gradiente acumulado, v_t é o segundo momento acumulado (variância do gradiente), e ϵ é um termo de estabilidade numérica.

O processo completo é repetido por várias épocas, ajustando os pesos de forma iterativa até que a função de perda J convirja ou atinja um valor aceitável no conjunto de validação.

A integração sinérgica da GNN e da LSTM permite capturar padrões espaço-temporais, ajustando os parâmetros de ambas as redes para maximizar a precisão e a robustez do modelo. Essa arquitetura é particularmente eficaz em cenários como a previsão de casos de COVID-19, onde as características espaciais influenciam diretamente as dinâmicas temporais. A abordagem apresentada pode ser estendida para outros contextos espaço-temporais, como modelagem climática ou análise de mobilidade urbana.

2.5 Bootstrapping aplicado a séries temporais

O bootstrapping é uma técnica de reamostragem que permite estimar a distribuição de uma estatística ao reamostrar os dados com reposição (Efron, 1979). No contexto de séries temporais, o bootstrapping pode ser aplicado aos resíduos dos dados de treino de um modelo de previsão para estimar a incerteza associada às previsões. Ao gerar várias amostras de resíduos, é possível construir intervalos de confiança para as previsões, oferecendo uma medida mais robusta da variabilidade e incerteza das previsões.

Ao aplicar o bootstrapping em um modelo de séries temporais, um exemplo genérico de modelo autoregressivo pode ser expresso como:

$$y_t = \sigma(y_{t-1}, y_{t-2}, \dots, y_{t-m}) + \epsilon_t,$$

onde y_t é o valor da série no tempo t , $\sigma(\cdot)$ representa a função de previsão baseada nos valores passados da série, m (a quantidade de valores no passado utilizados) e ϵ_t é o erro residual (ruído) associado à previsão.

Para gerar previsões futuras usando o bootstrapping dos resíduos, podemos simular iterativamente “caminhos” futuros, reamostrando os resíduos ϵ_t a partir de uma suposição de distribuição (como a normalidade) ou através de reamostragem dos resíduos estimados a partir do modelo ajustado. Isso nos permite capturar a incerteza que ocorrem nos dados temporais.

A cada iteração, um valor de ϵ_{t+1}^* é sorteado e usado para atualizar a série no próximo passo de tempo. Por exemplo:

$$y_{t+1}^* = \sigma(y_t, y_{t-1}, y_{t-2}, \dots, y_{t-m}) + \epsilon_{t+1}^*.$$

Neste ponto, y_{t+1}^* é a previsão gerada usando os m valores anteriores da série (até y_t) e um erro residual aleatorizado ϵ_{t+1}^* .

Continuando iterativamente, podemos prever o próximo valor ϵ_{t+2}^* usando y_{t+1}^* :

$$y_{t+2}^* = \sigma(y_{t+1}^*, y_t, y_{t-1}, \dots, y_{t-m}) + \epsilon_{t+2}^*.$$

Esse processo é repetido para cada horizonte de previsão, gerando diversos caminhos possíveis para o futuro da série temporal. Ao acumular essas simulações, podemos construir distribuições empíricas para os valores previstos em cada ponto de tempo futuro.

Estimativa de Intervalos de Confiança:

A partir dos múltiplos caminhos simulados, é possível calcular intervalos de confiança para as previsões. Esses intervalos são construídos a partir dos percentis das previsões geradas, refletindo a incerteza tanto nos resíduos quanto na evolução da série temporal. Essa abordagem é especialmente útil em séries temporais não lineares ou quando os resíduos não seguem uma distribuição conhecida, já que o bootstrapping não assume uma distribuição específica dos resíduos.

2.6 Métricas de desempenho dos modelos

A avaliação do desempenho dos modelos é fundamental para determinar a qualidade do ajuste e a confiabilidade das previsões realizadas. Neste trabalho, foram utilizadas duas métricas principais:

- **Erro Quadrático Médio (EQM):** Mede a média dos quadrados das diferenças entre os valores observados e previstos. É uma métrica apropriada para avaliar a precisão de modelos de regressão e séries temporais, especialmente em tarefas de previsão pontual.
- **Winkler Score:** Avalia a qualidade dos intervalos de previsão, levando em conta tanto a amplitude quanto a capacidade de cobrir corretamente os valores observados. Essa métrica é particularmente útil quando o objetivo é analisar a confiabilidade das previsões em termos probabilísticos.

Embora o Critério de Informação de Akaike (AIC) seja frequentemente utilizado para comparar modelos estatísticos baseados em verossimilhança, ele não foi adotado neste trabalho. Isso porque métodos de aprendizado de máquina, como Florestas Aleatórias e Redes Neurais, não assumem distribuições de probabilidade específicas e não possuem uma função de verossimilhança definida, inviabilizando o uso do AIC.

No entanto, o AIC foi utilizado por [Paiuta \(2022\)](#) para comparar ajustes do modelo ARIMA na previsão da incidência e mortalidade por COVID-19, justamente por se tratar de um modelo estatístico com base probabilística bem definida. Nesse contexto, o uso do AIC é apropriado, mas sua aplicação não se estende a métodos não paramétricos.

$$EQM = \frac{1}{T} \sum_{t=1}^T (y_t - \hat{y}_t)^2 \quad (2.39)$$

Onde:

- T é o número total de períodos de tempo.
- y_t são os valores observados no tempo t .
- $\hat{y}_t$ são os valores previstos no tempo t .

Um valor de EQM menor indica previsões pontuais mais precisas, com erros menores.

Se um intervalo de confiança de $100(1 - \alpha)\%$ no tempo t é dado por $[\ell_{\alpha,t}, u_{\alpha,t}]$, então o Winkler Score é definido sendo a amplitude do intervalo somado à uma penalidade, caso o observado esteja fora do intervalo da seguinte forma:

$$W_{\alpha,t} = \begin{cases} (u_{\alpha,t} - \ell_{\alpha,t}) + \frac{2}{\alpha} (\ell_{\alpha,t} - y_t) & , \text{ se } y_t < \ell_{\alpha,t}, \\ (u_{\alpha,t} - \ell_{\alpha,t}) & , \text{ se } \ell_{\alpha,t} \leq y_t \leq u_{\alpha,t}, \\ (u_{\alpha,t} - \ell_{\alpha,t}) + \frac{2}{\alpha} (y_t - u_{\alpha,t}) & , \text{ se } y_t > u_{\alpha,t}. \end{cases} \quad (2.40)$$

3 Materiais e Métodos

3.1 Material

Os dados utilizados neste estudo foram coletados pelo SEADE, acessado em abril de 2024, contendo informações detalhadas sobre a contaminação por COVID-19 no estado de São Paulo, incluindo o número de novos casos e óbitos. Dentre essas informações, utilizamos especificamente as séries temporais de novos casos para cada cidade do estado.

Para padronização dos dados, calculamos as médias móveis semanais, onde a média de cada dia foi obtida considerando o próprio dia e os seis dias anteriores. Em seguida, para ajustar a incidência de casos à população de cada cidade, dividimos essa média móvel pela população correspondente e multiplicamos por 100.000, o que resultou na taxa de incidência por 100.000 habitantes.

Dessa forma, uma incidência de COVID-19 em uma determinada cidade representaria que, em média, 1 novo caso diário foi registrado a cada 100.000 habitantes nos últimos 7 dias naquele local. Para exemplificar, em Campinas a incidência semanal de COVID-19 chegou a passar de 30 por alguns dias. Isso representa que, em média, 30 novos casos diários ocorreram a cada 100.000 habitantes nos últimos 7 dias. Dessa forma, em Campinas com cerca de 1.140.000 habitantes, o valor corresponderia a 342 casos diários em média naquela semana.

O objetivo da média móvel é destacar padrões subjacentes e tendências para entender o comportamento agregado da série. Assim, os modelos temporais podem ser ajustados à série suavizada para capturar as tendências principais, com menor influência da variabilidade de curto prazo.

Além disso, importamos um arquivo *shapefile* do IBGE para possibilitar a criação de mapas. Em seguida, realizamos a fusão (*merge*) dos dados epidemiológicos com o *shapefile*, permitindo assim a visualização espacial dos resultados. É importante destacar que a cidade de Ilhabela foi removida da análise devido à falta de fronteiras terrestres, o que apresenta problemas na análise de dependência espacial.

O ano de 2020 foi selecionado para análise devido à forte autocorrelação espacial observada nos primeiros meses da pandemia, refletida pelo índice de Moran. Em março, o índice atingiu 0,51, mantendo valores relativamente altos nos meses seguintes: 0,46 em abril, 0,30 em

maio, 0,36 em junho, 0,33 em julho e 0,34 em agosto. No entanto, a partir de setembro, o índice caiu significativamente para 0,12, indicando uma diminuição na dependência espacial.

Em contraste, os dados de 2021 apresentaram índices de Moran consistentemente baixos, com valores máximos de apenas 0,10, caracterizando uma fraca autocorrelação espacial. Esse comportamento torna a inclusão da dependência espacial como covariável menos relevante para a modelagem preditiva nesse período.

Outro fator determinante foi a forte influência da vacinação contra a COVID-19 nas taxas de incidência, que começou a ocorrer em 2021. A inclusão da vacinação no modelo exigiria ajustes adicionais e tornaria a análise mais complexa.

No ano de 2020, destaca-se também a implementação do *lockdown* em julho, como explorado por [Paiuta \(2022\)](#). No entanto, para fins de simplicidade, optou-se por não incluir explicitamente essa medida nos modelos propostos.

Dessa forma, o período analisado abrange desde a primeira semana epidemiológica até o final de setembro de 2020. Foram utilizados 219 dias como dados de treino para cada uma das 644 cidades, e os 7 dias subsequentes foram reservados para avaliar a previsão de curto prazo. Além disso, o desempenho do modelo foi avaliado para horizontes mais longos, com 30 dias representando o médio prazo e 92 dias o longo prazo, sendo essas previsões analisadas individualmente.

3.2 Métodos

A análise foi conduzida em três etapas principais: temporal, espacial e espaço-temporal.

Inicialmente, realizamos a análise temporal dos dados, seguida da análise espacial. Em cada etapa, aplicamos modelos específicos para avaliar o desempenho e a contribuição de cada abordagem. Por fim, combinamos as duas perspectivas em modelos espaço-temporais, permitindo uma comparação abrangente entre os diferentes métodos.

Cada uma dessas análises (temporal, espacial e espaço-temporal) foi realizada separadamente, utilizando códigos distintos para garantir a clareza e a precisão dos resultados.

Dado o grande número de séries temporais analisadas simultaneamente, 644 no total, optamos por focar em índices e resumos estatísticos que proporcionassem uma visão geral do desempenho dos modelos, ao invés de realizar análises detalhadas para cada série individualmente. A abordagem visa identificar tendências e padrões globais nos dados, garantindo uma avaliação robusta dos modelos aplicados.

Enquanto ([Brüggemann, 2023](#)) utilizam exclusivamente GNNs para modelar dados de COVID-19, este trabalho combina métodos de séries temporais (ARIMAX) com informações espaciais provenientes de modelos como SAR, explorando abordagens complementares.

Pacotes utilizados

O pacote `pmdarima` em Python, inspirado no pacote `forecast` do R, fornece uma implementação do AutoARIMA, facilitando a modelagem de séries temporais complexas sem a necessidade de ajustar manualmente os parâmetros (Smith; Ellis, 2019). Esse pacote também utiliza a máxima verossimilhança (EMV) para ajustar os parâmetros do modelo ARIMA.

O pacote `skforecast` oferece uma interface prática para utilizar modelos do `scikit-learn` em problemas de previsão temporal. Ele permite, por exemplo, o uso da Floresta Aleatória em previsões recursivas, em que os valores previstos em uma etapa são reutilizados como entrada nas etapas seguintes. Um exemplo ilustrativo dessa metodologia pode ser visto na Figura 9 (Rodrigo; Ortiz, 2024; Datos, 2025).

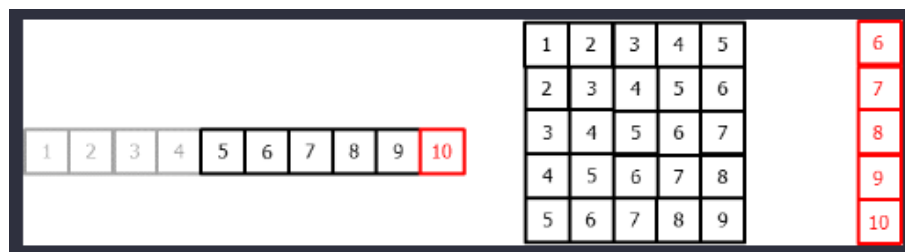


Figura 9 – Previsão recursiva: os valores previstos são usados como entrada nas etapas seguintes.
Fonte: (Rodrigo; Ortiz, 2024).

Os dois pacotes utilizados para trabalhar com redes neurais no Python foram o Keras e o PyTorch. O pacote Keras se destaca por ser considerado ser simples para seu uso, sendo ideal para uso rápido e experimentação (Chollet et al., 2015). Ele é frequentemente a escolha inicial para quem está começando com redes LSTM. Já o PyTorch oferece maior flexibilidade e controle sobre a arquitetura da rede (Paszke et al., 2017). No presente trabalho, inicialmente usamos o Keras para a modelagem LSTM, mas migramos para o PyTorch ao incorporar redes GNN à arquitetura, que foi criada utilizando o PyTorch Geometric (Fey; Lenssen, 2019).

O pacote `spreg`, parte do PySAL (*Python Spatial Analysis Library*), fornece ferramentas para estimação de modelos de regressão espacial, incluindo o modelo SAR. Ele permite estimar os parâmetros usando máxima verossimilhança ou métodos alternativos, como mínimos quadrados generalizados.

Os códigos utilizados neste trabalho estão disponíveis no GitHub¹.

¹ <https://github.com/ThiagoMoraesRizzieri/projeto_biometria>

4 Resultados e Discussões

Nesta seção, são apresentados os resultados obtidos a partir da modelagem e previsão para as cidades utilizando diferentes abordagens. Para cada modelo, foram calculados o Erro Quadrático Médio (EQM) e o Winkler Score, que quantificam o desempenho do ponto de vista das previsões pontuais e intervalares, respectivamente. Esses valores foram calculados para cada cidade e no fim, tiramos uma média dessas medidas para facilitar a nossa visualização.

Esta seção apresenta os resultados da modelagem e previsão da taxa de incidência da COVID-19 para as 10 cidades mais populosas do estado de São Paulo. Foram considerados três horizontes de previsão: curto prazo (7 dias), médio prazo (1 mês) e longo prazo (3 meses). A análise incluiu modelos temporais (ARIMA, SARIMAX, Floresta Aleatória, LSTM) e suas variações com integração de dependências espaciais (SAR e GNN). Os resultados são avaliados com base no Erro Quadrático Médio (EQM) e no Winkler Score (W95% e W80%). Todos os códigos estão em um repositório do GitHub¹.

4.1 Análise dos Modelos

Os modelos avaliados incluíram variações da modelagem temporal com ARIMA, SARIMAX, Floresta Aleatória e Redes Neurais LSTM, com e sem integração de dependências espaciais através de modelos SAR e GNN. A seguir, apresentamos os resultados de cada modelo:

Para cada horizonte de previsão, os modelos foram avaliados separadamente. A Tabela 1 apresenta os resultados de desempenho para o período de **curto prazo** (previsão de 7 dias), enquanto as Tabelas 2 e 3 mostram os resultados para os períodos de **médio prazo** (1 mês) e **longo prazo** (3 meses), respectivamente.

Em relação à dependência espacial, observamos que tanto o modelo SAR quanto o GNN foram eficazes em capturar a autocorrelação espacial. Os resíduos desses modelos apresentaram valores médios do índice de Moran consideravelmente reduzidos (SAR: -0,0002; GNN: -0,09) em comparação ao valor observado na série original em abril (I de Moran 0,46), o que evidencia a capacidade de ambos os modelos em eliminar a dependência espacial residual.

Visualmente, nota-se que os modelos espaciais suavizam a série de incidência semanal,

¹ <https://github.com/ThiagoMoraesRizzieri/projeto_biometria>

conforme ilustrado na Figura 11. Nela, comparamos a série original (Figura 10) com os ajustes obtidos pelos modelos SAR (Figura 10a) e GNN (Figura 10b).

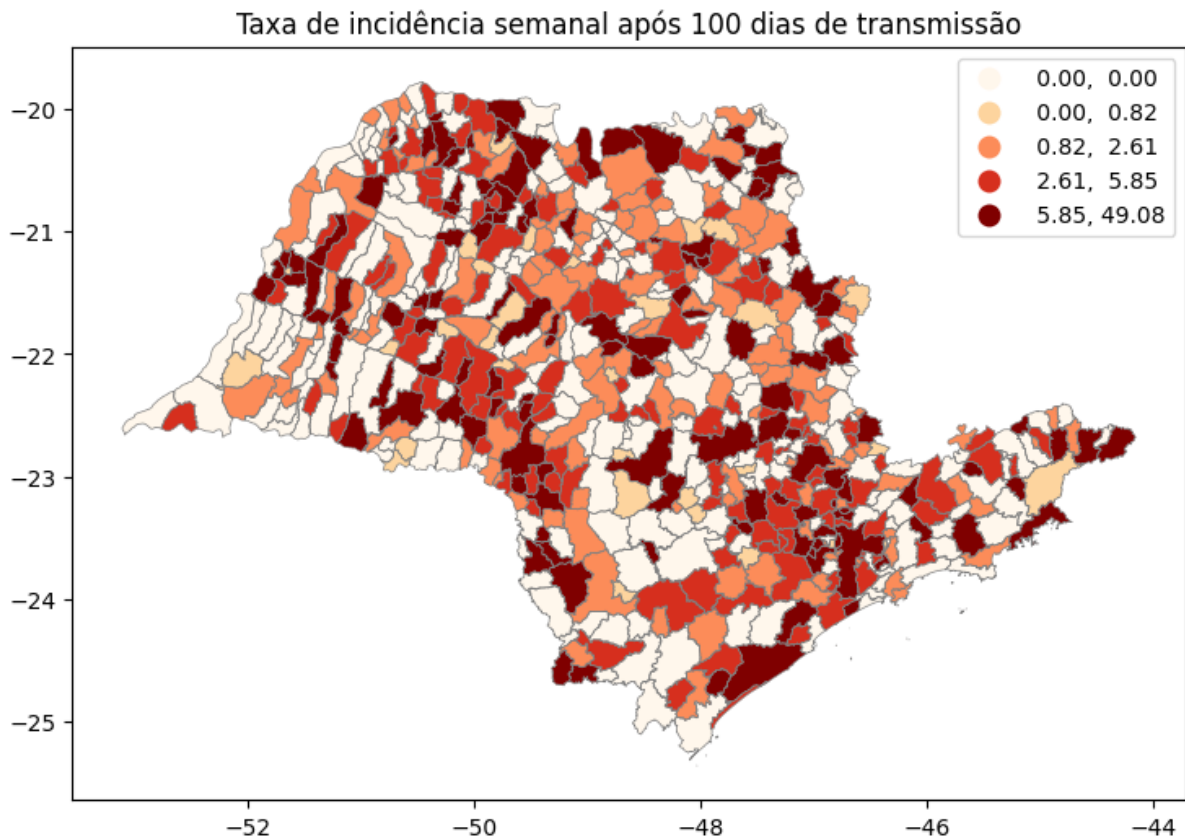
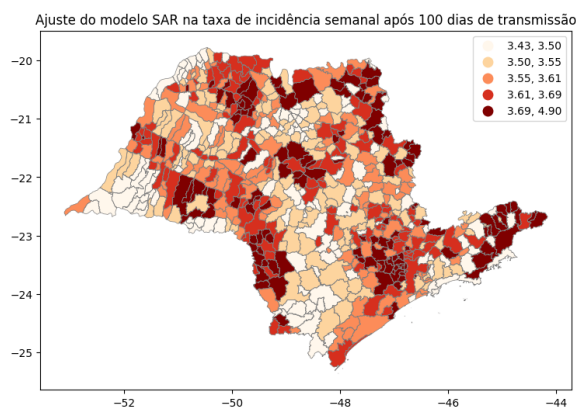
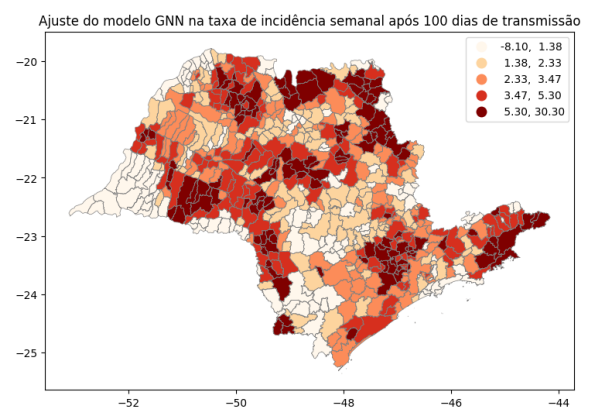


Figura 10 – Taxa de incidência semanal original.



(a) Ajuste do modelo SAR.



(b) Ajuste do modelo GNN.

Figura 11 – Comparação das taxas de incidência semanal e dos ajustes obtidos pelos modelos SAR e GNN após 100 dias de transmissão.

A aplicação dos modelos SAR e GNN resultou em 644 novas séries temporais — uma para cada município — que apresentaram, em média, correlação de 0,72 (SAR) e 0,82 (GNN) com suas respectivas séries originais.

Tabela 1 – Desempenho dos modelos no horizonte de curto prazo (previsão de 7 dias).

Modelos	EQM	W95%	W80%
MEAN	131,73	69,02	41,16
ARIMA	75,87	53,49	29,17
ARIMAX (SAR var. exógena)	70,47	48,99	23,14
Dinâmico (SAR var. exógena)	66,58	48,02	25,77
ARIMAX (GNN var. exógena)	68,98	70,09	30,41
Dinâmico (GNN var. exógena)	72,03	70,44	31,00
Floresta Aleatória	226,31	255,17	78,74
Floresta Aleatória (SAR var. exógena)	81,43	121,44	42,18
Floresta Aleatória (GNN var. exógena)	94,54	151,57	47,66
Redes Neurais LSTM	98,23	54,05	33,39
LSTM (SAR var. exógena)	131,07	62,61	36,59
LSTM (GNN var. exógena)	144,05	63,03	37,79
LSTM + GNN	115,21	65,80	232,50

Tabela 2 – Desempenho dos modelos no horizonte de médio prazo (previsão de 1 mês).

Modelos	EQM	W95%	W80%
MEAN	224,56	84,70	44,06
ARIMA	267,25	88,94	48,90
ARIMAX (SAR var. exógena)	222,63	78,51	40,16
Dinâmico (SAR var. exógena)	219,39	77,78	39,70
ARIMAX (GNN var. exógena)	251,61	115,79	48,66
Dinâmico (GNN var. exógena)	252,90	127,21	51,58
Floresta Aleatória	357,93	250,36	82,53
Floresta Aleatória (SAR var. exógena)	268,88	192,86	65,15
Floresta Aleatória (GNN var. exógena)	263,56	181,20	67,23
Redes Neurais LSTM	397,32	116,06	73,33
LSTM (SAR var. exógena)	454,01	111,05	68,38
LSTM (GNN var. exógena)	423,80	111,95	69,91
LSTM + GNN	262,02	150,62	100,09

Tabela 3 – Desempenho dos modelos no horizonte de longo prazo (previsão de 3 meses).

Modelos	EQM	W95%	W80%
MEAN	342,13	110,70	68,08
ARIMA	373,78	121,05	70,74
ARIMAX (SAR var. exógena)	307,75	101,05	53,22
Dinâmico (SAR var. exógena)	301,49	137,84	46,09
ARIMAX (GNN var. exógena)	340,57	147,84	63,22
Dinâmico (GNN var. exógena)	336,25	166,30	66,09
Floresta Aleatória	449,54	264,39	89,55
Floresta Aleatória (SAR var. exógena)	389,02	249,51	81,20
Floresta Aleatória (GNN var. exógena)	371,79	231,43	76,58
Redes Neurais LSTM	401,08	222,68	86,49
LSTM (SAR var. exógena)	713,40	157,06	103,53
LSTM (GNN var. exógena)	914,43	187,68	143,53
LSTM + GNN	304,64	281,06	103,53

Os resultados anteriores mostram que:

- A simples inclusão de modelos espaciais, como o SAR ou o GNN, como variáveis exógenas em arquiteturas como a Floresta Aleatória ou o LSTM, pode não ser suficiente para capturar de forma eficiente a interação entre dependências temporais e espaciais.
- O modelo **LSTM+GNN** apresentou a melhor performance, o que corrobora a hipótese de que a combinação arquitetural entre LSTM e GNN permite que o modelo aprenda, de maneira mais eficiente, as dependências espaciais e temporais simultaneamente.

Em prazos maiores, os modelos puramente temporais foram piores que o modelo da média (MEAN).

Quando o **GNN é integrado diretamente à arquitetura LSTM**, o modelo tem a capacidade de aprender representações espaciais durante o processo de previsão temporal. Isso permite que a LSTM ajuste dinamicamente suas previsões com base nas dependências espaciais, ao contrário do uso do GNN como uma variável exógena, onde o impacto espacial se torna fixo.

Essa diferença sugere que, enquanto a inserção de uma variável exógena (como o GNN ou SAR) ajuda na captura de dependências espaciais, a verdadeira melhora é obtida quando o **GNN é integrado à arquitetura do LSTM**, permitindo que a rede neural ajuste suas previsões continuamente com base nas características espaciais. A arquitetura combinada **LSTM+GNN** explora de forma mais robusta a relação entre espaço e tempo, garantindo maior precisão nas previsões.

A construção de grafos para dados espaço-temporais, especialmente em cenários sem estrutura de grafo explícita, é um dos desafios apontados por [Li et al. \(2023\)](#). Este trabalho aborda

a construção do grafo através da matriz de ponderação espacial de contiguidade do tipo *Queen*, que se mostrou ser bem aplicável aos modelos.

Embora [Panagopoulos, Nikolentzos e Vazirgiannis \(2021b\)](#) tenham encontrado que GNNs combinadas com aprendizado por transferência superam modelos tradicionais em previsões epidemiológicas, este estudo explora a integração de dependência espacial utilizando o modelo SAR como variável exógena em um modelo ARIMAX, o que resultou em melhorias significativas em cenários específicos. Além disso, diferentemente de [Panagopoulos, Nikolentzos e Vazirgiannis \(2021b\)](#), que identificaram as GNN como o melhor modelo preditivo, nossos resultados mostram que a inclusão de informações espaciais por meio do SAR no modelo ARIMAX pode ser mais vantajosa em contextos específicos, como foi o caso da previsão da incidência semanal de COVID-19 nas cidades de São Paulo.

Embora ([Paiuta, 2022](#)) também tenham identificado o modelo ARIMA como mais adequado em comparação com outros modelos de previsão temporais (a saber, modelo de Suavização Exponencial) para previsão de incidência e mortalidade por COVID-19, este trabalho busca integrar informações espaciais, utilizando o modelo SAR como variável exógena no ARIMAX, o que proporciona uma abordagem complementar para prever a disseminação da doença. Assim como é dito por ([Paiuta, 2022](#)), o modelo ARIMA foi o modelo temporal que teve o melhor desempenho, mas o presente trabalho destaca que a inclusão de variáveis espaciais pode refinar ainda mais as previsões, especialmente em cenários de alta dependência espacial. No caso deles, foi analisado apenas a cidade de Araraquara, São Carlos, Campinas e São Paulo, então seria preciso utilizar dados de seus respectivos municípios vizinhos.

Os resultados de [Silva \(2023\)](#) confirmam a presença de forte dependência espacial na disseminação da COVID-19, destacando a necessidade de modelos que incorporem essas relações, com foco no número de óbitos e utilizando métodos bayesianos para capturar dependências entre regiões. Este estudo segue essa linha ao integrar dependências espaciais por meio do modelo SAR e do modelo GNN para melhorar os modelos temporais.

Os resultados de [Brüggemann \(2023\)](#) indicaram limitações no desempenho das GNNs, com enviesamento nas predições. Em contrapartida, este estudo integra informações espaciais diretamente no modelo temporal ARIMAX, alcançando melhorias em predições específicas.

Porém, em alguns casos, modelos baseados em Redes Neurais em Grafo (GNNs) têm demonstrado grande potencial em capturar padrões espaço-temporais complexos na propagação de doenças. A arquitetura TMGNN proposta por [Hy et al. \(2022\)](#) combina representações multirresolução com informações temporais, mostrando melhorias significativas em relação a modelos como ARIMA e LSTM.

Os resultados obtidos por [Fritz, Dorigatti e Rügamer \(2022\)](#) destacam que a integração de dados sociais e de mobilidade em GNNs pode superar modelos clássicos em precisão preditiva. Em contraste, este estudo focou na incorporação de informações espaciais em modelos de séries

temporais, obtendo melhorias específicas em cenários de alta dependência espacial.

4.2 Análise Residual das 10 Cidades Mais Populosas

Para cada uma das 10 cidades mais populosas do estado de São Paulo, foi selecionado o melhor modelo preditivo com base no menor EQM, considerando previsões de curto prazo (7 dias). A seguir, apresentam-se as análises específicas para cada cidade, incluindo: (i) estatísticas descritivas da série temporal, (ii) desempenho do modelo preditivo selecionado e (iii) gráficos de previsão comparando os valores observados e previstos.

Exemplo: São Paulo

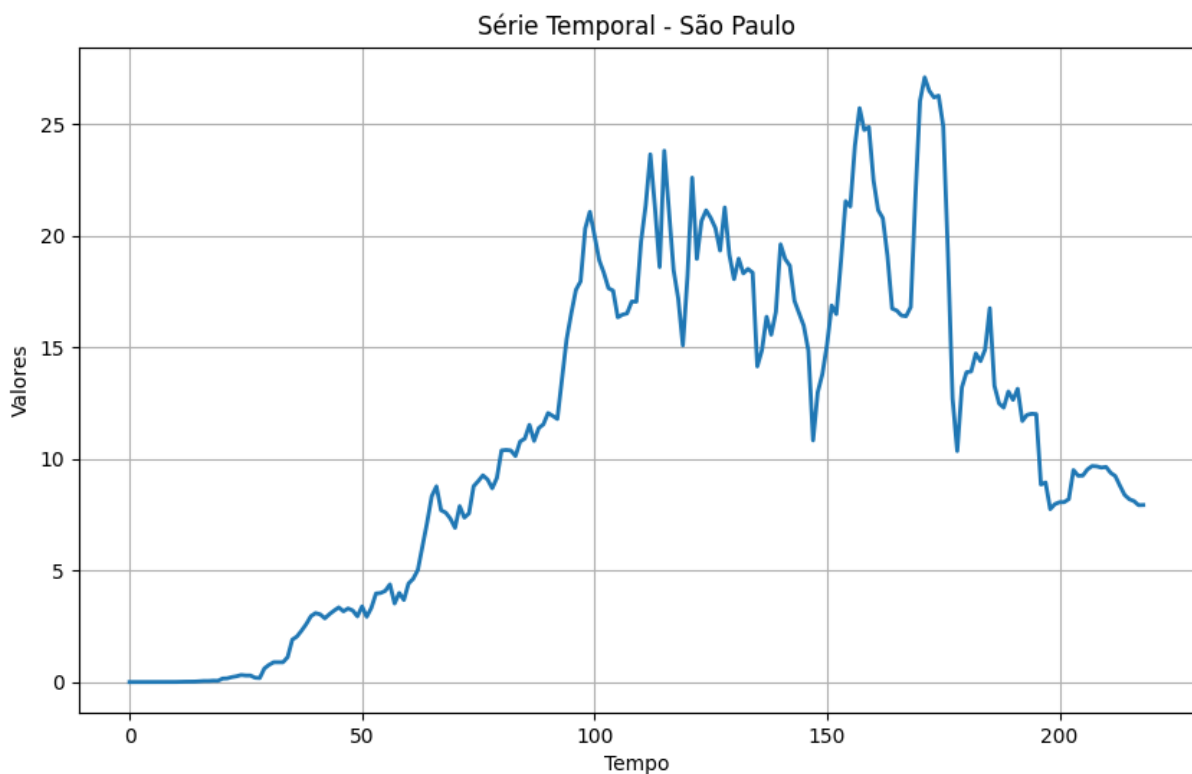


Figura 12 – Série temporal da incidência semanal de COVID-19 em São Paulo. Fonte: Elaborado pelo autor.

As estatísticas descritivas da série temporal da cidade de São Paulo são apresentadas na Tabela 4.

Tabela 4 – Estatísticas descritivas da incidência semanal de COVID-19 em São Paulo.

Estatísticas	Valor
Média	11,13
Desvio Padrão (σ)	7,60
Mínimo	0,00
1ºQuartil (25%)	3,82
Mediana (50%)	10,81
3ºQuartil (75%)	17,14
Máximo	27,10

A cidade de São Paulo apresentou o modelo **ARIMAX com variável exógena SAR** como o melhor desempenho em curto prazo, com os seguintes resultados:

- **EQM:** 0.11
- **Teste de Shapiro-Wilk (p-valor):** 0.78
- **Teste de Ljung-Box (p-valor):** 0.07
- **Número de outliers:** 0

Obtemos o seguinte modelo:

$$(1 - 1.79B + 0.96B^2) y_t = 0.78 x_t + (1 - 1.62B + 0.53B^2 + 0.35B^3 - 0.13B^4) \varepsilon_t$$

$$\varepsilon_t \sim \mathcal{N}(0, 1.20)$$

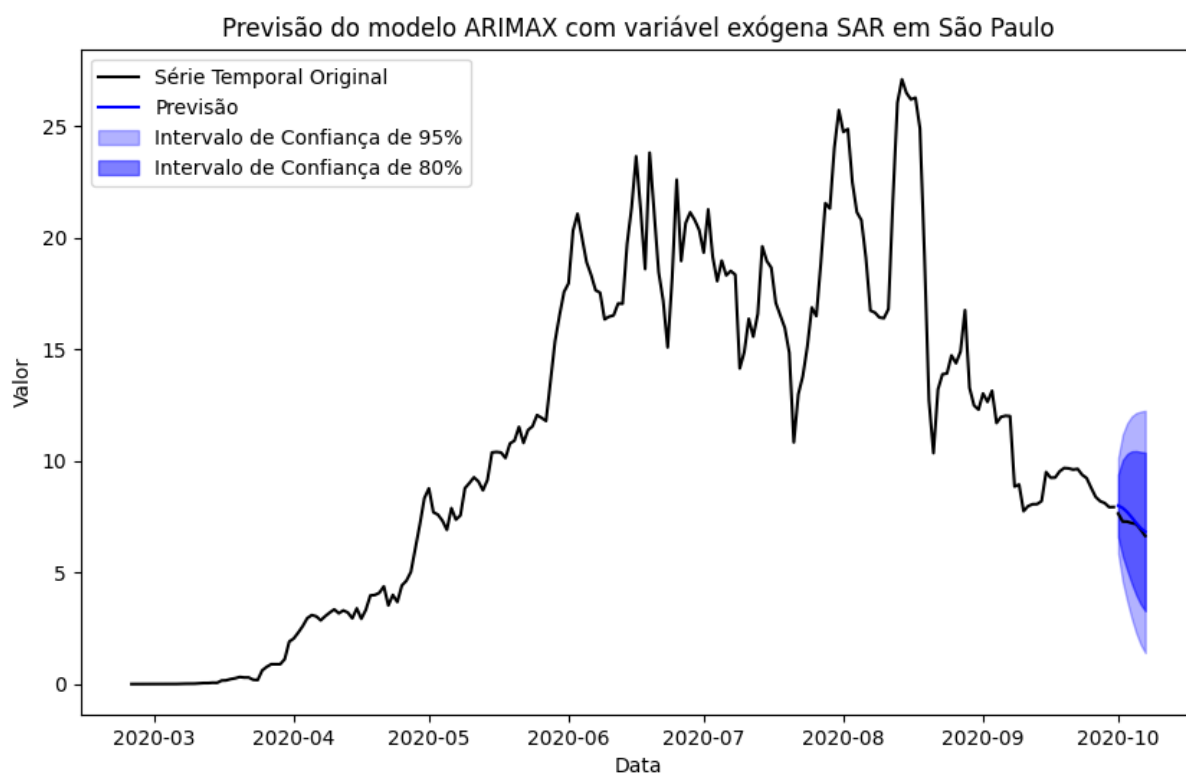


Figura 13 – Previsão da incidência em São Paulo. Fonte: Elaborado pelo autor.

Exemplo: Guarulhos

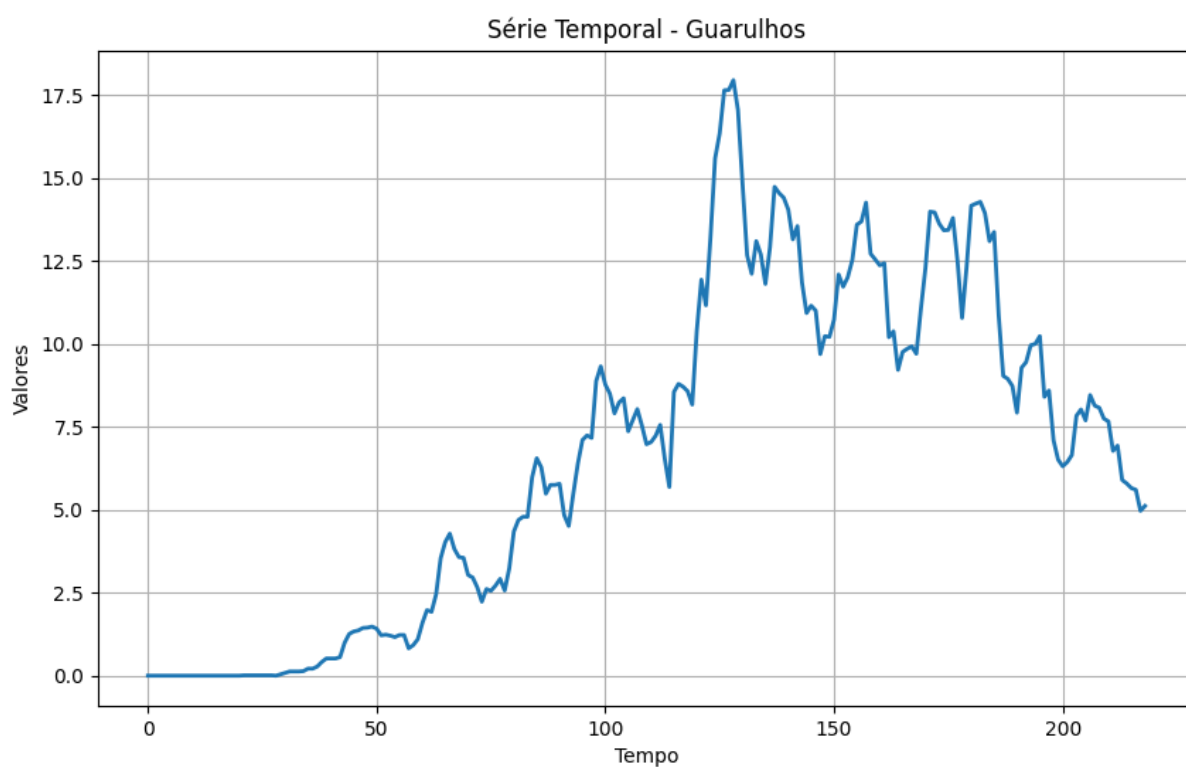


Figura 14 – Série temporal da incidência semanal de COVID-19 em Guarulhos. Fonte: Elaborado pelo autor.

As estatísticas descritivas da série temporal de Guarulhos estão na Tabela 5.

Tabela 5 – Estatísticas descritivas da incidência semanal de COVID-19 em Guarulhos.

Estatísticas	Valor
Média	6,66
Desvio Padrão (σ)	5,06
Mínimo	0,00
1ºQuartil (25%)	1,35
Mediana (50%)	6,98
3ºQuartil (75%)	10,75
Máximo	17,95

O modelo **ARIMAX com variável exógena GNN** obteve o melhor desempenho para Guarulhos, com os seguintes resultados:

- **EQM:** 0.13
- **Teste de Shapiro-Wilk (p-valor):** 0.60
- **Teste de Ljung-Box (p-valor):** 0.69
- **Número de outliers:** 0

Obtemos o seguinte modelo:

$$y_t = 0.08 x_t + (1 + 0.17B) \varepsilon_t$$

$$\varepsilon_t \sim \mathcal{N}(0, 0.52)$$

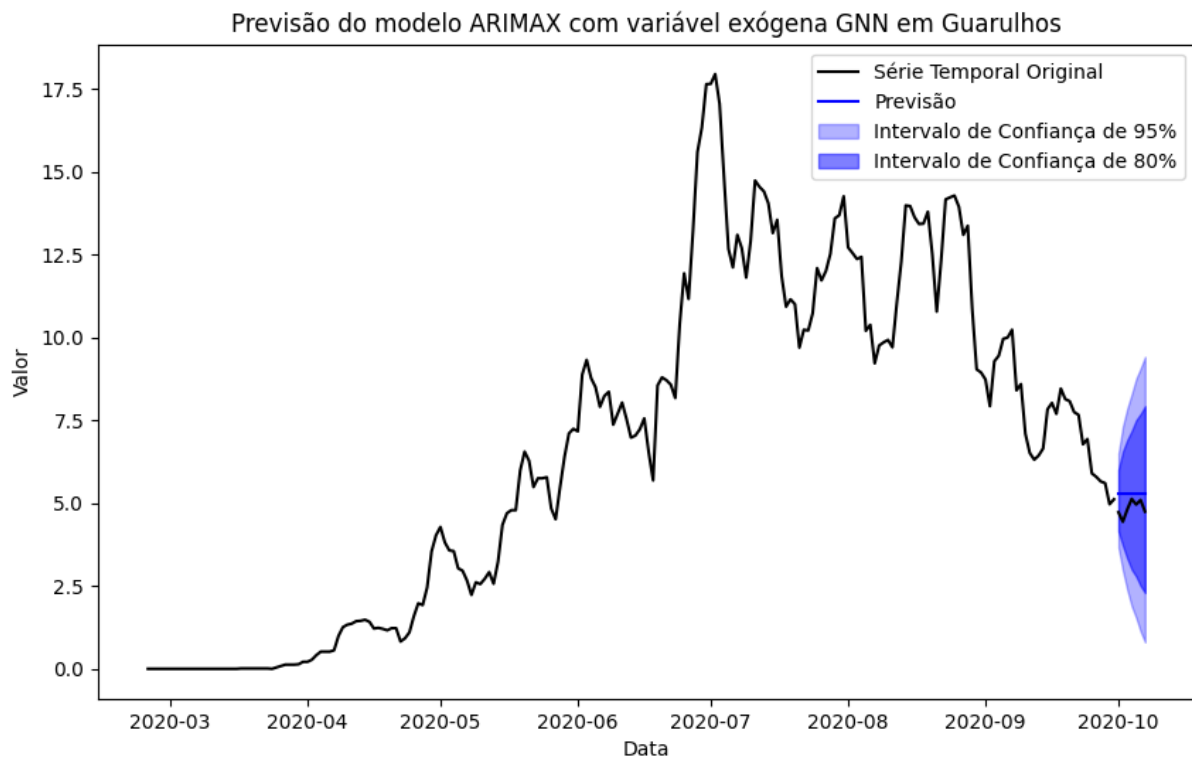


Figura 15 – Previsão da incidência em Guarulhos. Fonte: Elaborado pelo autor.

Exemplo: Campinas

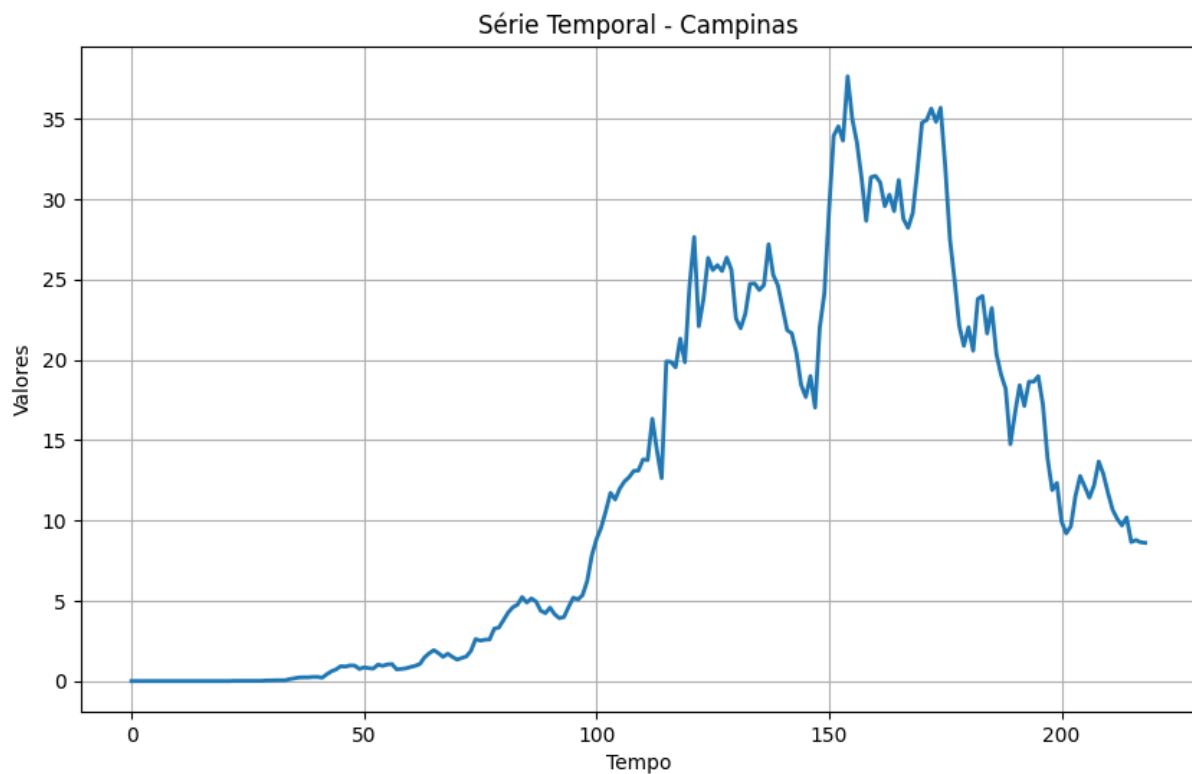


Figura 16 – Série temporal da incidência semanal de COVID-19 em Campinas. Fonte: Elaborado pelo autor.

As estatísticas descritivas da série temporal de Campinas estão na Tabela 6.

Tabela 6 – Estatísticas descritivas da incidência semanal de COVID-19 em Campinas.

Estatísticas	Valor
Média	12,08
Desvio Padrão (σ)	11,40
Mínimo	0,00
1ºQuartil (25%)	0,94
Mediana (50%)	9,94
3ºQuartil (75%)	21,98
Máximo	37,64

O modelo **Dinâmico SAR** foi o melhor desempenho para Campinas, com os seguintes resultados:

- **EQM:** 5.08
- **Teste de Shapiro-Wilk (p-valor):** 0.74
- **Teste de Ljung-Box (p-valor):** 0.25
- **Número de outliers:** 0

A fórmula do modelo fica da seguinte forma:

$$y_t = \beta_0 + 1.39X_t + \eta_t$$

$$(1 + 0.53B^1 - 0.95B^2)(1 - B)\eta_t = (1 - 0.65B^1 + 0.94B^2)\varepsilon_t$$

$$\varepsilon_t \sim \mathcal{N}(0, 1.19)$$

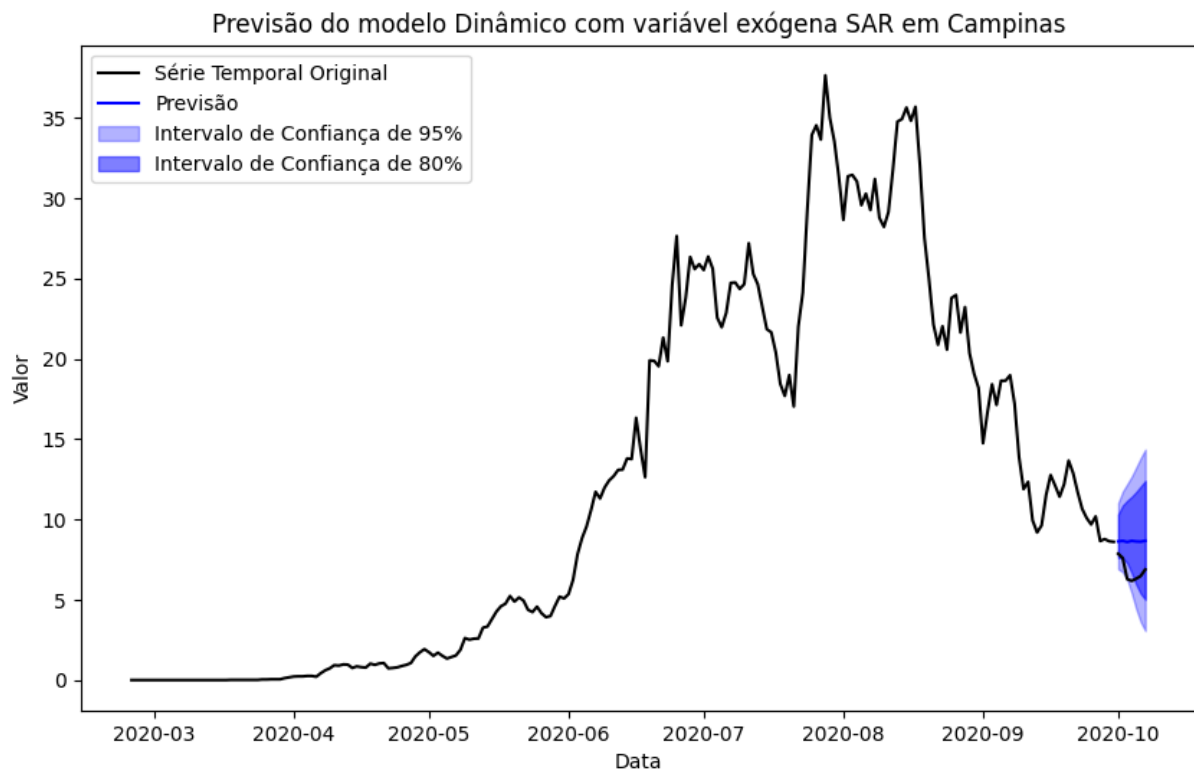


Figura 17 – Previsão da incidência em Campinas. Fonte: Elaborado pelo autor.

Exemplo: São Bernardo do Campo

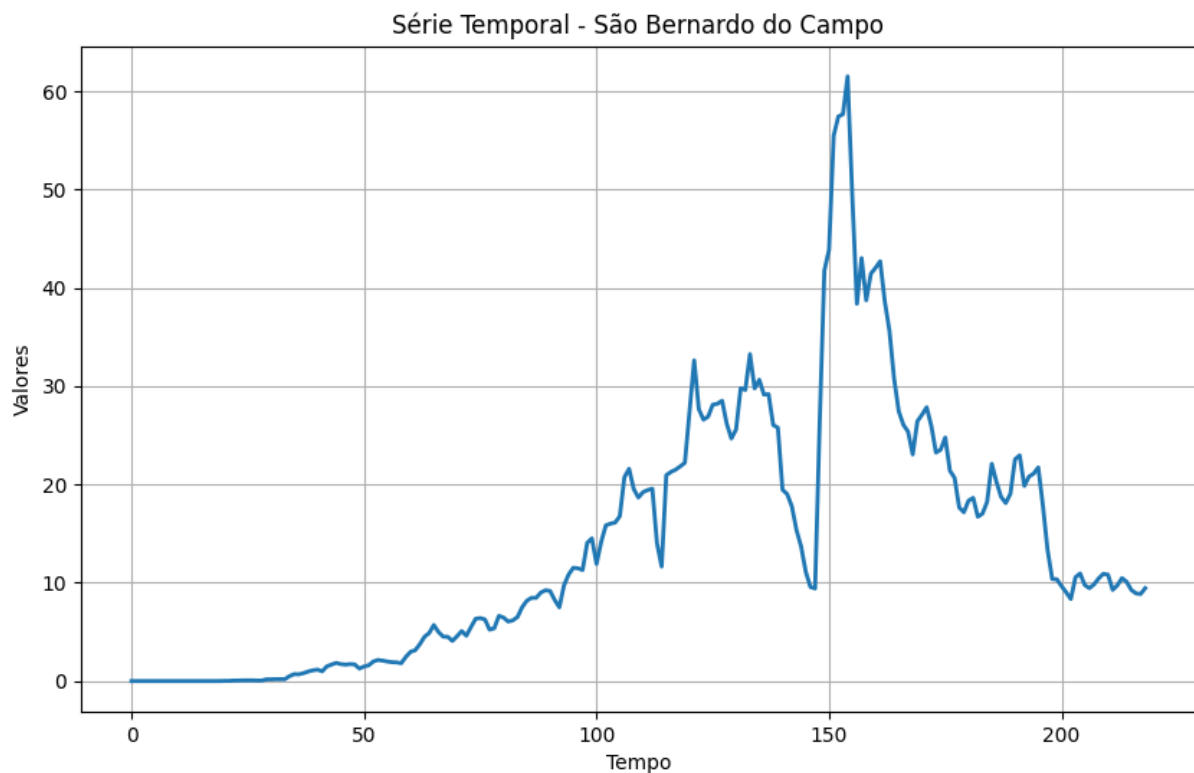


Figura 18 – Série temporal da incidência semanal de COVID-19 em São Bernardo do Campo. Fonte: Elaborado pelo autor.

As estatísticas descritivas da série temporal de São Bernardo do Campo estão na Tabela 7.

Tabela 7 – Estatísticas descritivas da incidência semanal de COVID-19 em São Bernardo do Campo.

Estatísticas	Valor
Média	13,81
Desvio Padrão (σ)	13,02
Mínimo	0,00
1ºQuartil (25%)	1,95
Mediana (50%)	10,34
3ºQuartil (75%)	21,54
Máximo	61,50

O modelo **ARIMAX com variável exógena SAR** foi selecionado para São Bernardo do Campo, com os seguintes resultados:

- **EQM:** 2.06
- **Teste de Shapiro-Wilk (p-valor):** 0.82
- **Teste de Ljung-Box (p-valor):** 0.42
- **Número de outliers:** 0

Obtemos o seguinte modelo:

$$(1 - 0.31B + 0.31B^2 - 0.59B^3) y_t = 1.57 x_t + (1 + 0.00B + 0.30B^2 - 0.64B^3 - 0.40B^4) \varepsilon_t$$

$$\varepsilon_t \sim \mathcal{N}(0, 3.75)$$

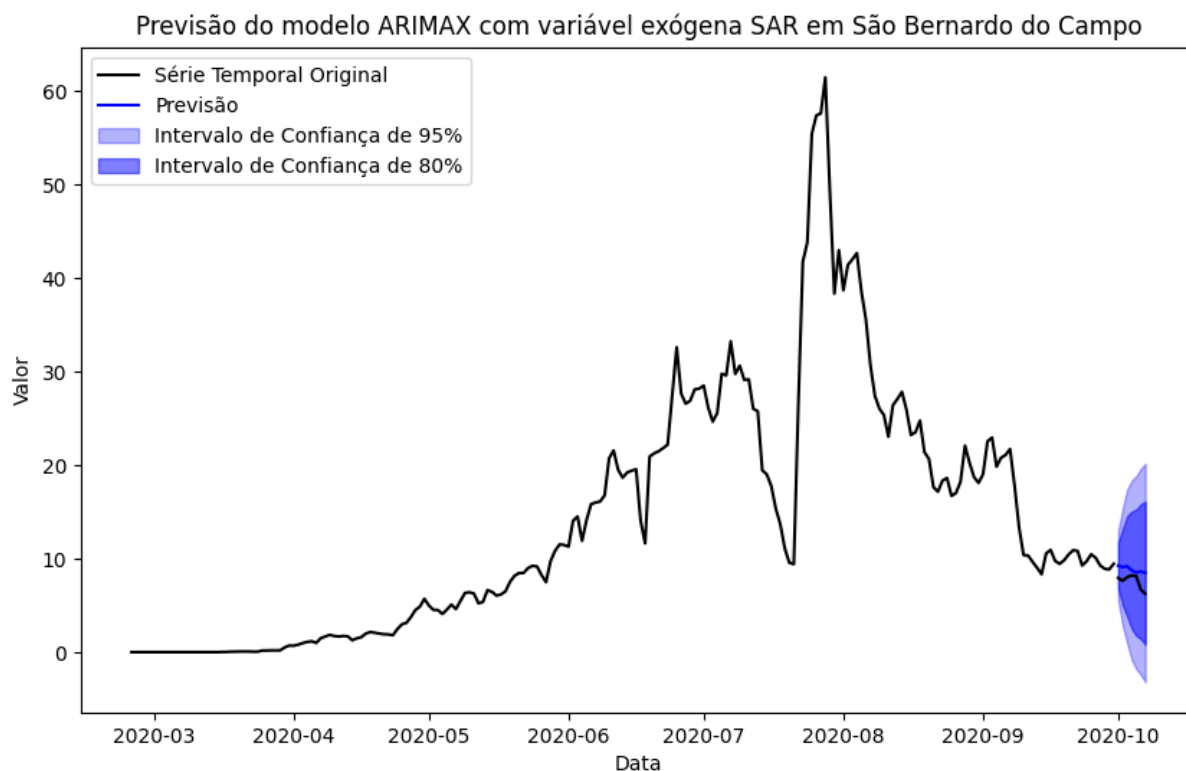


Figura 19 – Previsão da incidência em São Bernardo do Campo. Fonte: Elaborado pelo autor.

Exemplo: Santo André

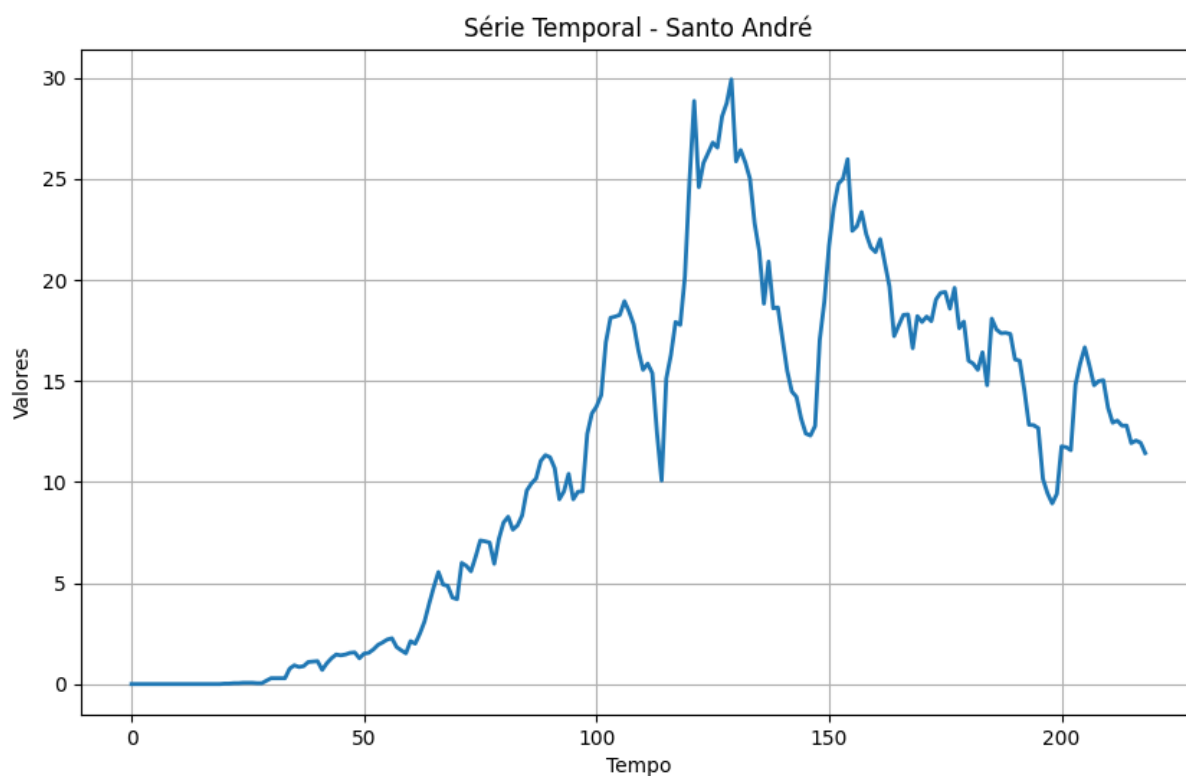


Figura 20 – Previsões do melhor modelo para a cidade de Santo André. Fonte: Elaborado pelo autor.

Estatísticas descritivas da série temporal da cidade de Santo André:

Tabela 8 – Estatísticas descritivas da incidência semanal de COVID-19 em Santo André.

Estatísticas	Valor
Média	11,23
Desvio-padrão	8,44
Mínimo	0,00
1º Quartil	1,77
Mediana	12,04
3º Quartil	17,85
Máximo	29,91

O modelo que obteve o melhor desempenho foi o modelo **dinâmico SAR**, com os seguintes parâmetros:

- **EQM:** 0,95
- **Shapiro-Wilk p-valor:** 0,25
- **Ljung-Box p-valor:** 0,73
- **Número de outliers:** 0

A fórmula do modelo fica da seguinte forma:

$$y_t = \beta_0 + 1.10X_t + \eta_t$$

$$(1 - 1.44B^1 - 0.44B^2 + 0.11B^3)(1 - B)\eta_t = (1 + 1.76B^1 + 0.94B^2)\varepsilon_t$$

$$\varepsilon_t \sim \mathcal{N}(0, 0.90)$$

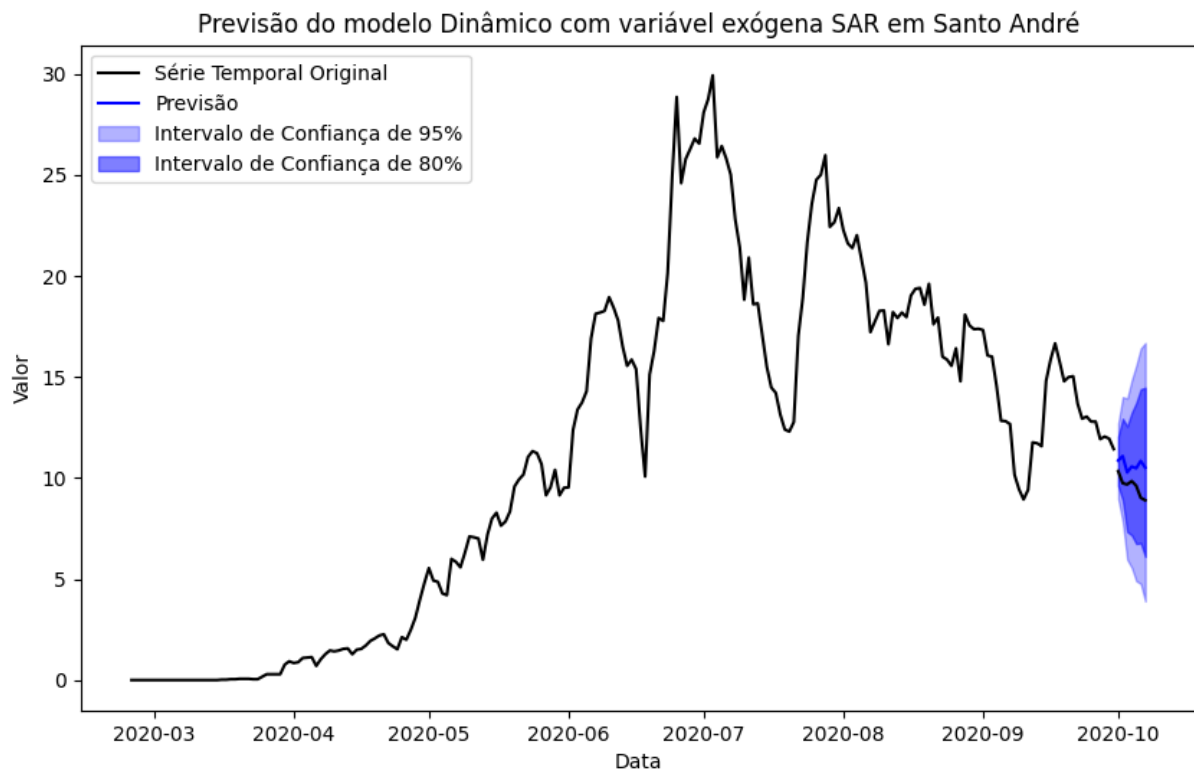


Figura 21 – Previsão da incidência em Santo André. Fonte: Elaborado pelo autor.

Exemplo: Sorocaba

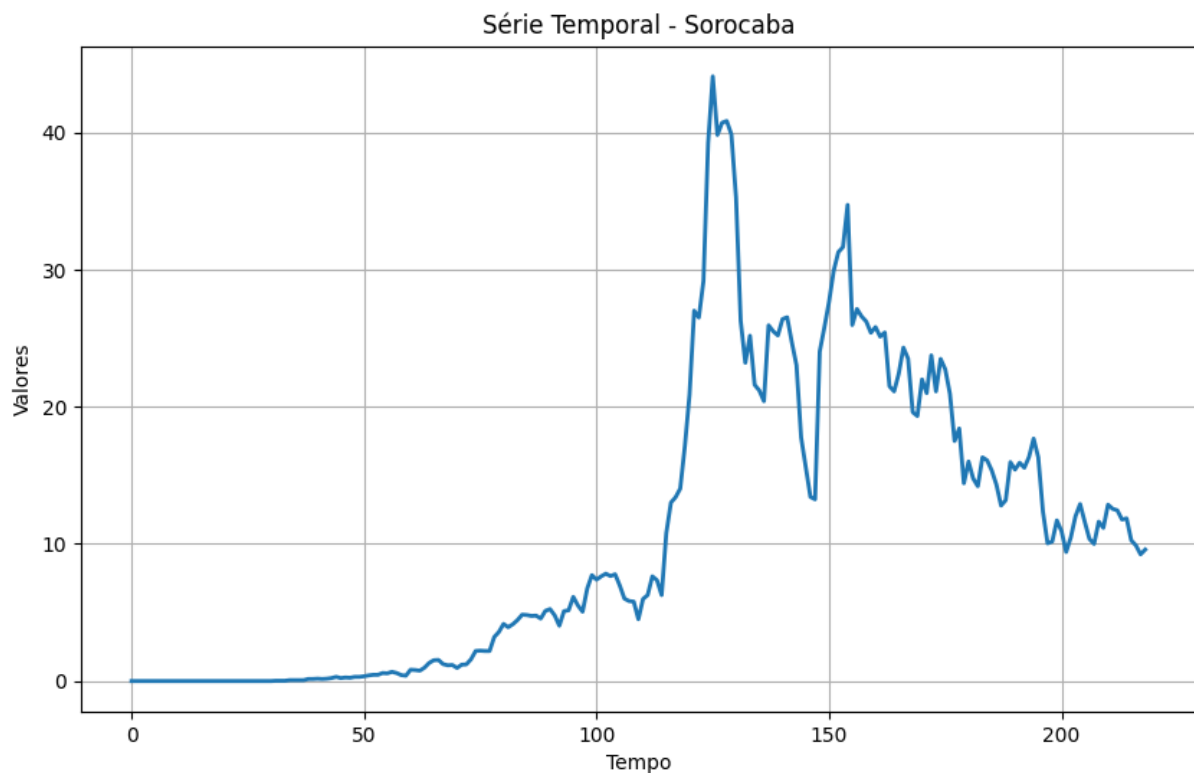


Figura 22 – Previsões do melhor modelo para a cidade de Sorocaba. Fonte: Elaborado pelo autor.

Estatísticas descritivas da série temporal da cidade de Sorocaba:

Tabela 9 – Estatísticas descritivas da incidência semanal de COVID-19 em Sorocaba.

Estatísticas	Valor
Média	10,69
Desvio-padrão	10,83
Mínimo	0,00
1º Quartil	0,46
Mediana	7,64
3º Quartil	17,72
Máximo	44,10

O modelo que obteve o melhor desempenho foi o modelo **ARIMAX GNN**, com os seguintes parâmetros:

- **EQM:** 0,67
- **Shapiro-Wilk p-valor:** 0,16
- **Ljung-Box p-valor:** 0,61
- **Número de outliers:** 1

Obtemos o seguinte modelo:

$$y_t = 0.10 x_t + (1 + 0.18B) \varepsilon_t$$

$$\varepsilon_t \sim \mathcal{N}(0, 3.56)$$

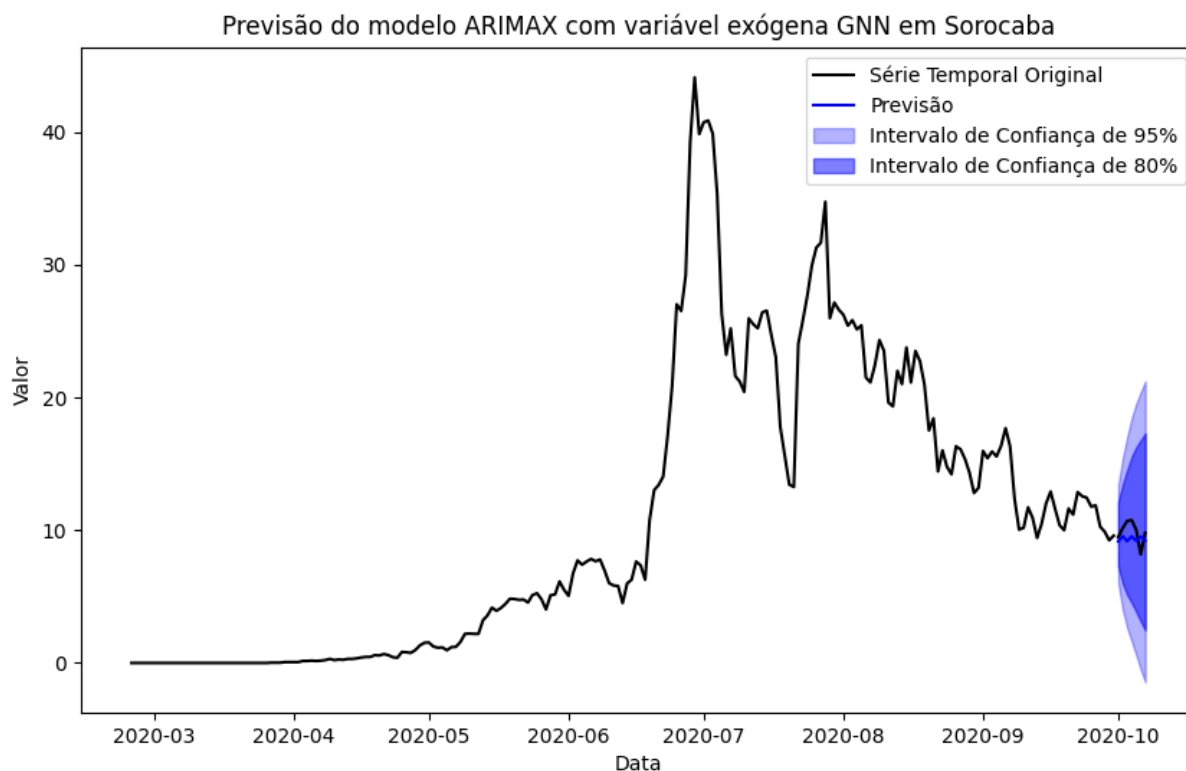


Figura 23 – Previsão da incidência em Sorocaba. Fonte: Elaborado pelo autor.

Exemplo: Osasco

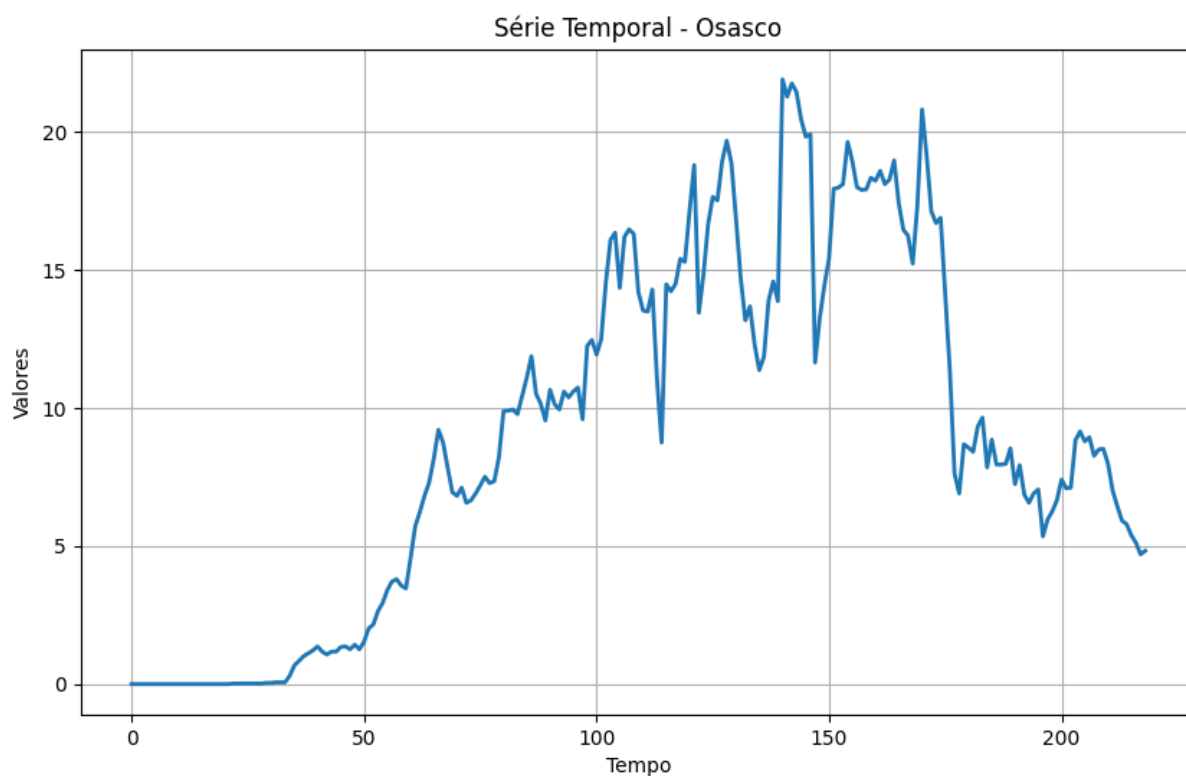


Figura 24 – Previsões do melhor modelo para a cidade de Osasco.

Estatísticas descritivas da série temporal da cidade de Osasco:

Tabela 10 – Estatísticas descritivas da incidência semanal de COVID-19 em Osasco.

Estatísticas	Valor
Média	8,96
Desvio-padrão	6,46
Mínimo	0,00
1º Quartil	3,17
Mediana	8,52
3º Quartil	14,39
Máximo	21,90

O modelo que obteve o melhor desempenho foi o modelo **dinâmico GNN**, com os seguintes parâmetros:

- **EQM:** 0,05
- **Shapiro-Wilk p-valor:** 0,28
- **Ljung-Box p-valor:** 0,01
- **Número de outliers:** 0

A fórmula do modelo fica da seguinte forma:

$$y_t = \beta_0 + 1.11X_t + \eta_t$$

$$(1)(1 - B)\eta_t = (1 - 0.21B^1 + 0.10B^2 + 0.19B^3)\varepsilon_t$$

$$\varepsilon_t \sim \mathcal{N}(0, 1.64)$$

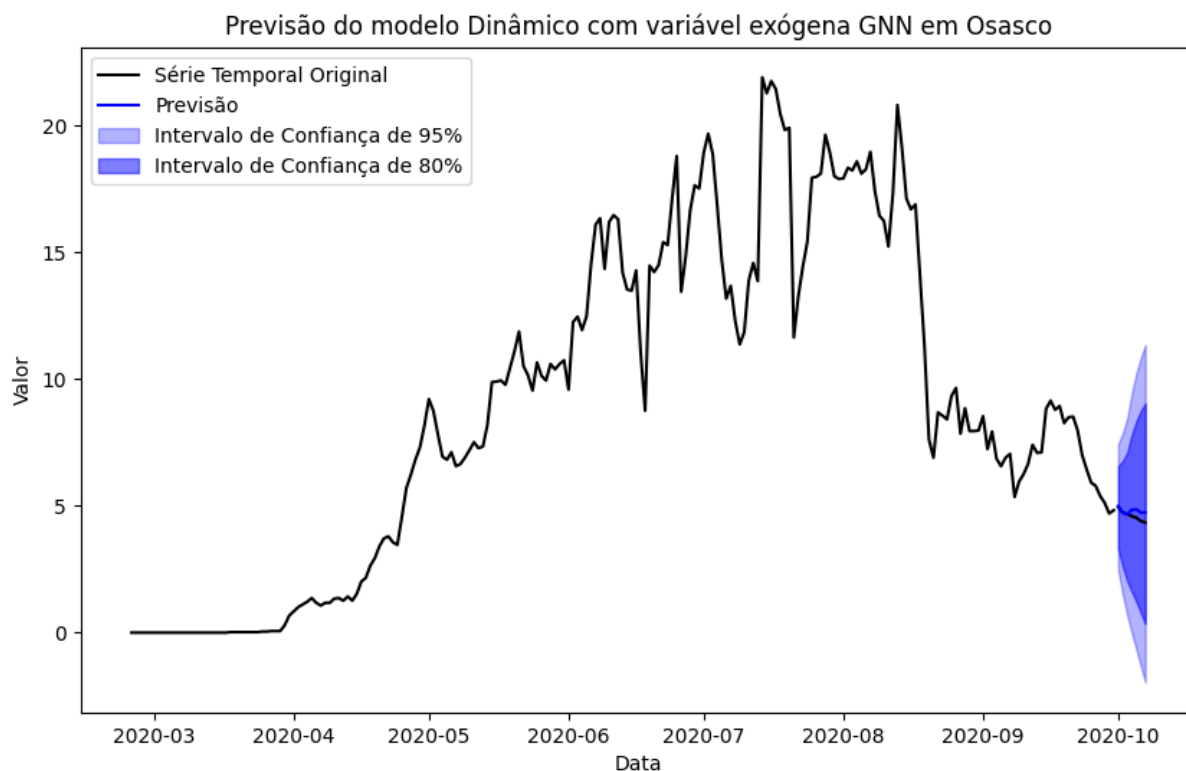


Figura 25 – Previsão da incidência em Osasco. Fonte: Elaborado pelo autor.

Exemplo: Ribeirão Preto

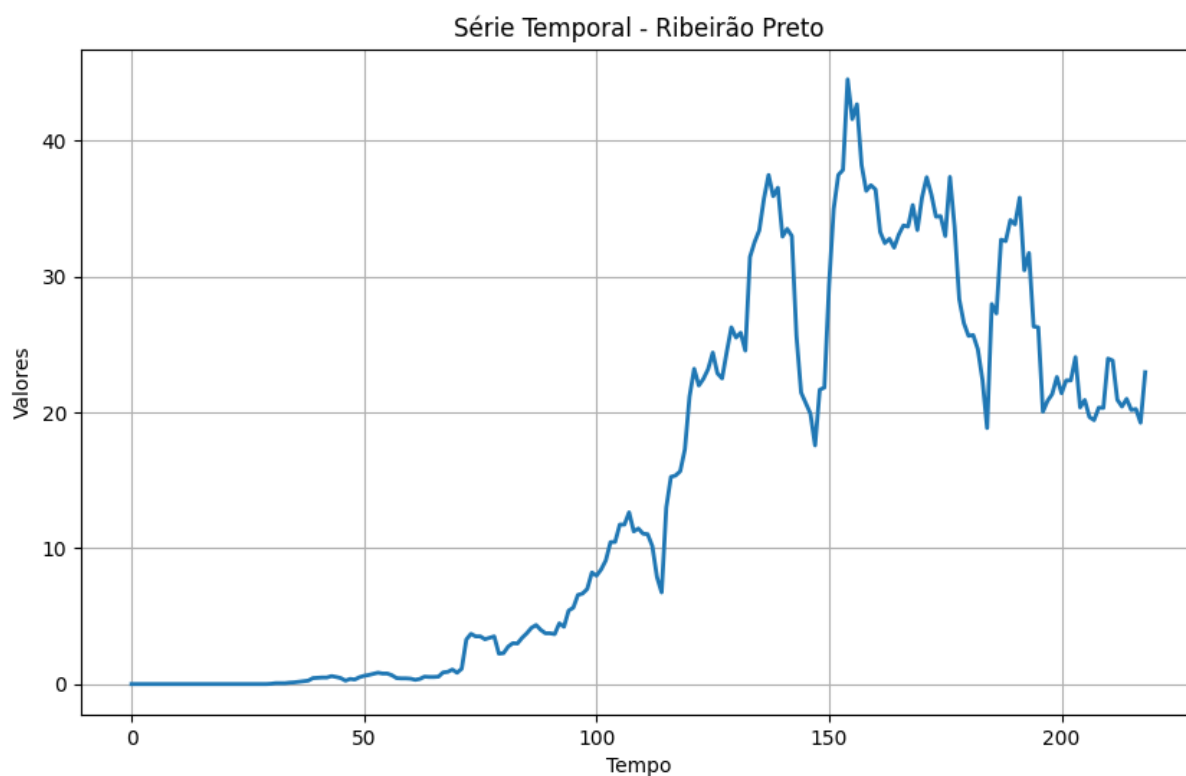


Figura 26 – Previsões do melhor modelo para a cidade de Ribeirão Preto. Fonte: Elaborado pelo autor.

Estatísticas descritivas da série temporal da cidade de Ribeirão Preto:

Tabela 11 – Estatísticas descritivas da incidência semanal de COVID-19 em Ribeirão Preto.

Estatísticas	Valor
Média	14,44
Desvio-padrão	13,73
Mínimo	0,00
1º Quartil	0,52
Mediana	11,07
3º Quartil	25,62
Máximo	44,52

O modelo que obteve o melhor desempenho foi o modelo **ARIMAX GNN**, com os seguintes parâmetros:

- **EQM:** 4,53
- **Shapiro-Wilk p-valor:** 0,56
- **Ljung-Box p-valor:** 0,04
- **Número de outliers:** 0

Obtemos o seguinte modelo:

$$(1 + 1.58B + 0.65B^2) y_t = 0.19 x_t + (1 + 1.71B + 0.83B^2) \varepsilon_t$$

$$\varepsilon_t \sim \mathcal{N}(0, 2.64)$$

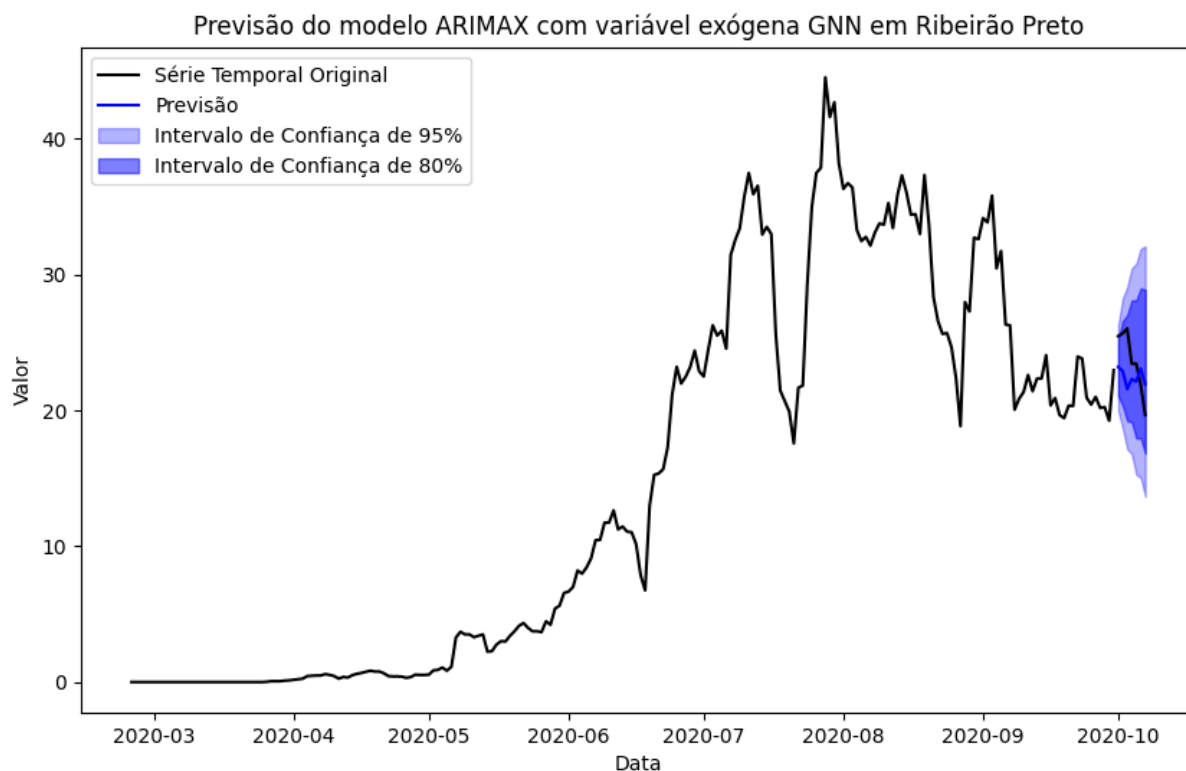


Figura 27 – Previsão da incidência em Ribeirão Preto. Fonte: Elaborado pelo autor.

Exemplo: São José dos Campos

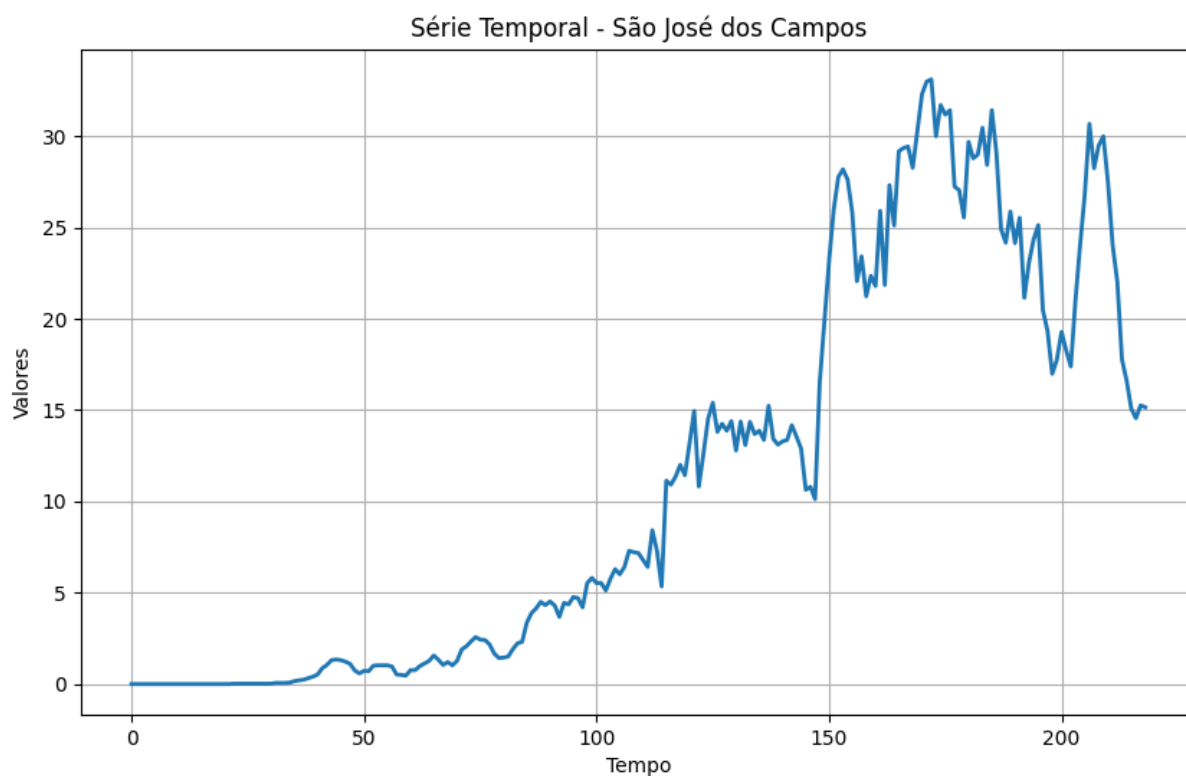


Figura 28 – Previsões do melhor modelo para a cidade de São José dos Campos. Fonte: Elaborado pelo autor.

Estatísticas descritivas da série temporal da cidade de São José dos Campos:

Tabela 12 – Estatísticas descritivas da incidência semanal de COVID-19 em São José dos Campos.

Estatísticas	Valor
Média	11,06
Desvio-padrão	10,85
Mínimo	0,00
1º Quartil	1,03
Mediana	6,80
3º Quartil	21,09
Máximo	33,11

O modelo que obteve o melhor desempenho foi o modelo **dinâmico GNN**, com os seguintes parâmetros:

- **EQM:** 5,20
- **Shapiro-Wilk p-valor:** 0,47
- **Ljung-Box p-valor:** 0,02
- **Número de outliers:** 2

A fórmula do modelo fica da seguinte forma:

$$y_t = \beta_0 + 0.62X_t + \eta_t$$

$$(1)(1 - B)\eta_t = (1)\varepsilon_t$$

$$\varepsilon_t \sim \mathcal{N}(0, 16.58)$$

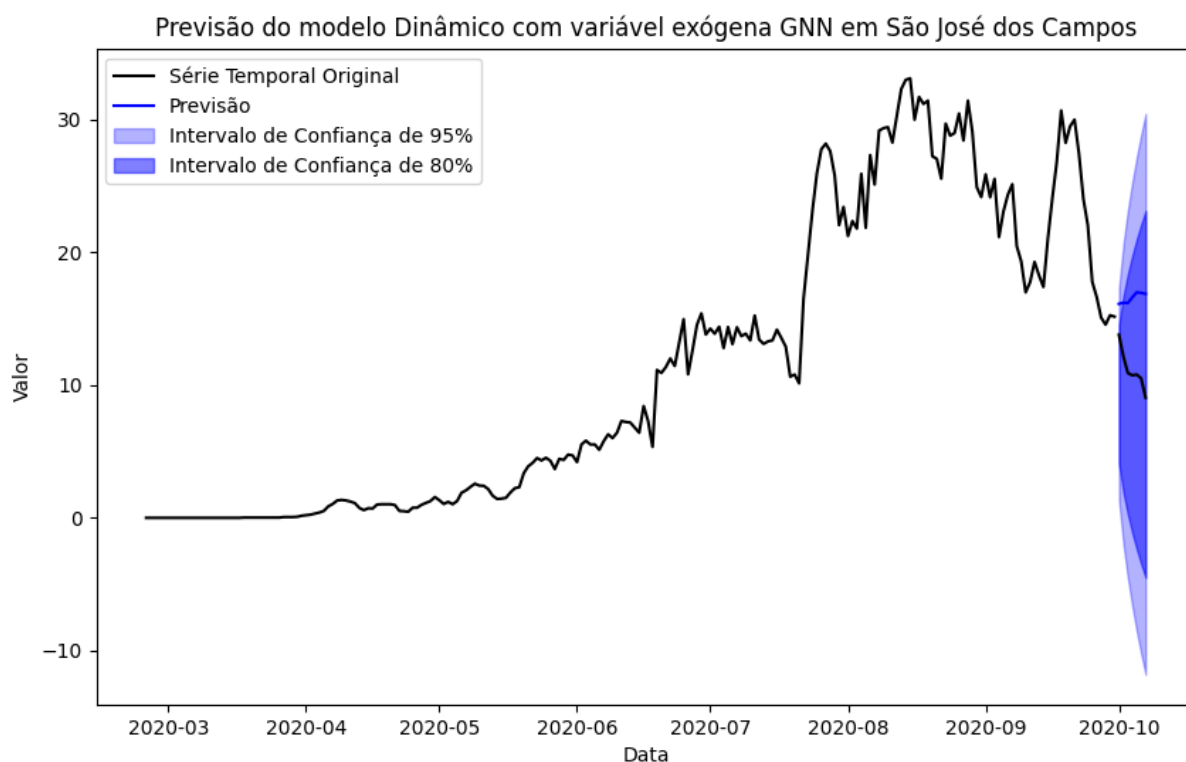


Figura 29 – Previsão da incidência em São José dos Campos. Fonte: Elaborado pelo autor.

Exemplo: São José do Rio Preto

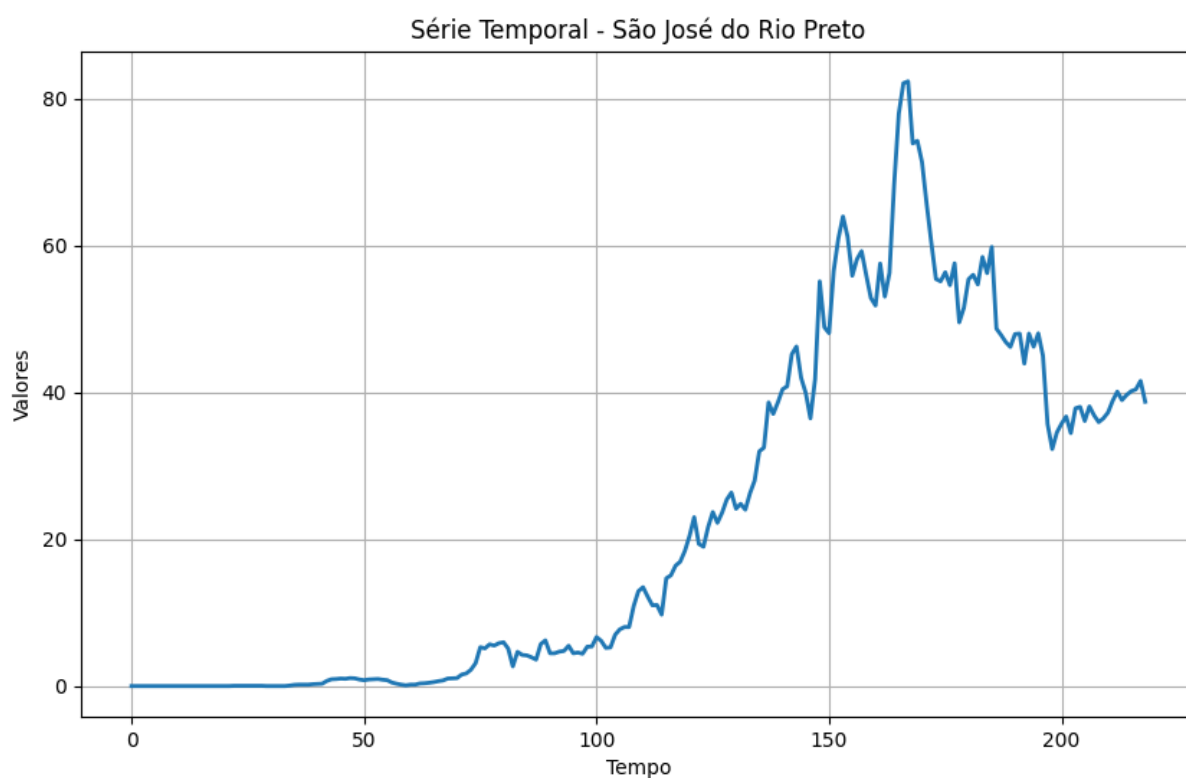


Figura 30 – Previsões do melhor modelo para a cidade de São José do Rio Preto. Fonte: Elaborado pelo autor.

Estatísticas descritivas da série temporal da cidade de São José do Rio Preto:

Tabela 13 – Estatísticas descritivas da incidência semanal de COVID-19 em São José do Rio Preto.

Estatísticas	Valor
Média	22,13
Desvio-padrão	23,34
Mínimo	0,00
1º Quartil	0,78
Mediana	10,91
3º Quartil	40,62
Máximo	82,35

O modelo que obteve o melhor desempenho foi o modelo **dinâmico GNN**, com os seguintes parâmetros:

- **EQM:** 4,16
- **Shapiro-Wilk p-valor:** 0,18
- **Ljung-Box p-valor:** 0,04
- **Número de outliers:** 0

A fórmula do modelo fica da seguinte forma:

$$y_t = \beta_0 + 1.81X_t + \eta_t$$

$$(1 + 0.86B^1)\eta_t = (1 - 0.08B^1 + 0.21B^2)\varepsilon_t$$

$$\varepsilon_t \sim \mathcal{N}(0, 16.39)$$

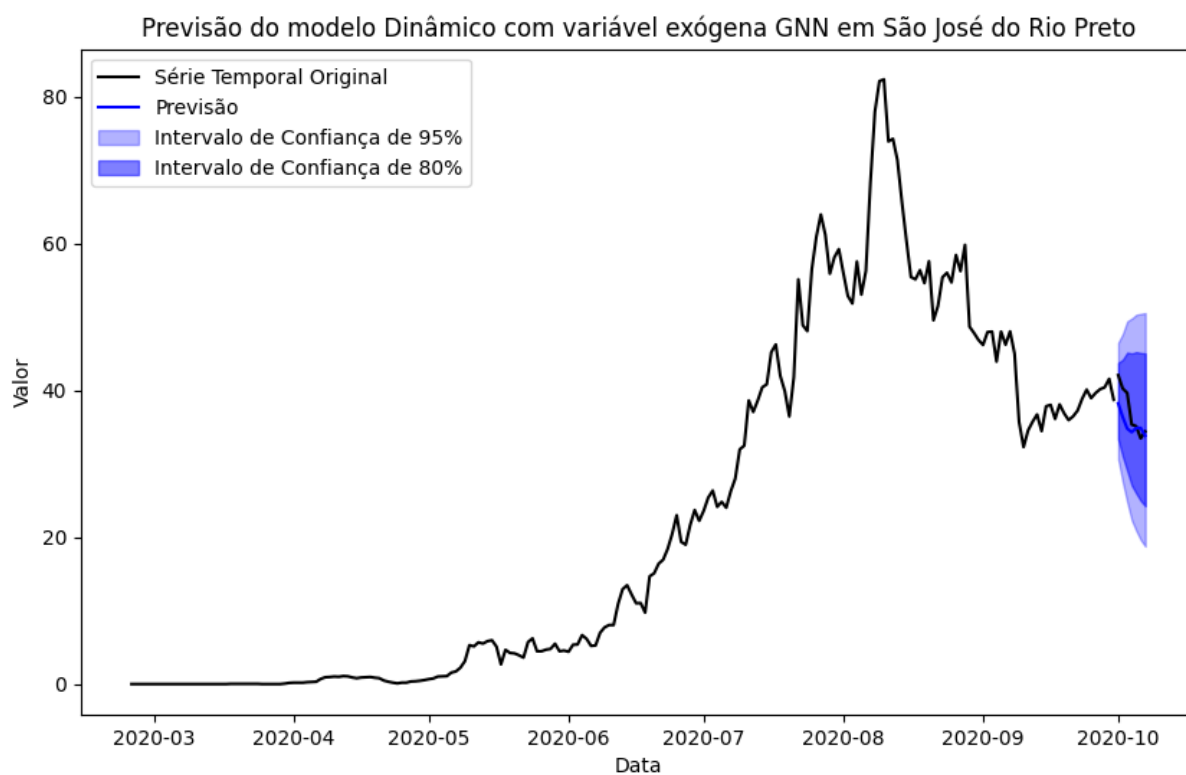


Figura 31 – Previsão da incidência em São José do Rio Preto. Fonte: Elaborado pelo autor.

5 Conclusão

Em relação à modelagem puramente temporal, os resultados obtidos indicam que o modelo ARIMA apresentou o melhor desempenho em termos de erro quadrático médio (EQM), sendo capaz de capturar com maior precisão os padrões temporais da taxa de incidência de COVID-19. O modelo de Floresta Aleatória, por sua vez, teve um desempenho inferior nas séries. O modelo LSTM se mostrou intermediário, capturando bem algumas não linearidades, mas não tão bom quanto o modelo ARIMA.

Os intervalos de previsão foram gerados utilizando bootstrapping, com exceção do modelo ARIMA, que já faz uso da suposição de normalidade dos resíduos, e indicaram uma variação significativa entre os modelos. O modelo ARIMA apresentou intervalos de previsão mais precisos, de acordo com o Winkler Score, seguido do modelo de Redes Neurais LSTM e o de Floresta Aleatória. Com confiança de 95%, em média o Winkler Score do intervalo de previsão do modelo ARIMA foi superior, o de Redes Neurais LSTM foi intermediário e o de Floresta Aleatória foi inferior, o mesmo ocorrendo para 80%.

É possível que o Winkler Score tenha sido menor nos intervalos de previsão de 80% em comparação aos de 95%, em grande parte devido à penalização mais rigorosa aplicada pela métrica para valores que caem fora do intervalo de confiança. Como a confiança de 95% envolve intervalos mais amplos, as previsões são menos propensas a falhas de cobertura, resultando em maiores penalizações caso isso ocorra.

Além disso, embora, em geral, os intervalos de previsão do ARIMA sejam maiores, eles sofrem menos penalizações. Isso ocorre porque os modelos LSTM e Floresta Aleatória, apesar de apresentarem intervalos menores, frequentemente não capturam adequadamente os dados reais, resultando em penalizações maiores.

É importante destacar que, como esperado, a qualidade das previsões diminuiu significativamente para horizontes de longo prazo, enquanto as previsões de curto prazo apresentaram melhor desempenho.

Os resultados indicam que a eficácia das diferentes abordagens espaço-temporais varia conforme as características dos dados analisados, corroborando os achados de [Panagopoulos, Nikolentzos e Vazirgiannis \(2021b\)](#).

Entre os modelos avaliados, os melhores desempenhos foram obtidos com o ARIMA

combinado com alguma abordagem de integração de informações espaciais. Destaca-se a análise das 10 cidades mais populosas, onde os modelos selecionados consistentemente apresentaram o ARIMA com alguma forma de inclusão de variáveis exógenas. Dentre essas abordagens, destacam-se o ARIMAX e os modelos dinâmicos que incorporaram estratégias como o SAR ou a GNN. De maneira geral, o ARIMAX combinado com a variável derivada do SAR foi o modelo que apresentou o melhor ajuste, em média, para as cidades do estado de São Paulo.

Referências

- AKAIKE, Hirotugu. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, v. 19, n. 6, p. 716–723, 1974. 24
- ANSELIN, Luc. *Spatial Econometrics: Methods and Models*. Dordrecht: Springer, 1988. 37, 39
- BAUMONT, Catherine. Spatial effects and econometric modelling in economics: Theory and applications of spatial econometrics. *Revue d'économie régionale et urbaine*, CAIRN, 2004. 38
- BOX, George E. P. et al. *Time Series Analysis: Forecasting and Control*. 5th. ed. Hoboken, NJ, USA: John Wiley & Sons, 2015. ISBN 978-1-118-67502-1. 12, 13, 14, 15, 18, 19, 20, 21, 22, 46, 47
- BREIMAN, Leo. Random forests. *Machine Learning*, v. 45, n. 1, p. 5–32, 2001. 18, 28, 29
- BREIMAN, Leo et al. *Classification and Regression Trees*. Boca Raton, FL, USA: Chapman & Hall/CRC, 1984. ISBN 978-0-412-04841-8. 13
- BRÜGGEMANN, Rian Carlos Vieira. *Utilização de Redes Neurais em Grafo para prever surtos de doenças: uma abordagem testando propagação de informação em uma rede convolucional*. Florianópolis: [s.n.], 2023. Trabalho de Conclusão de Curso em Sistemas de Informação. 41, 55, 61
- CHATFIELD, Chris. *The Analysis of Time Series: An Introduction*. 6th. ed. Boca Raton, FL, USA: Chapman & Hall/CRC, 2003. ISBN 978-1-58488-317-3. 14
- CHOLLET, François et al. *Keras: The Python Deep Learning API*. 2015. Disponível em: <<https://keras.io>>. Acesso em: 21 jan. 2025. 56
- CRESSIE, Noel. *Statistics for Spatial Data*. New York: John Wiley & Sons, 1993. 36, 37, 40
- DATOS, Ciencia de. *Predicción de Series Temporales con Python y scikit-learn*. 2025. <<https://cienciadatos.net/documentos/py27-time-series-forecasting-python-scikitlearn.html>>. Acesso em: 21 jan. 2025. 56
- DICKEY, David A.; FULLER, Wayne A. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association*, v. 74, n. 366, p. 427–431, 1979. 16
- EFRON, Bradley. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, v. 7, n. 1, p. 1–26, 1979. 51
- FAUCI, Anthony S.; LANE, H. Clifford; REDFIELD, Robert R. Covid-19 - navigating the uncharted. *New England Journal of Medicine*, v. 382, n. 13, p. 1268–1269, 2020. Disponível em: <<https://www.nejm.org/doi/full/10.1056/NEJMe2002387>>. 9
- FEY, Matthias; LENSSSEN, Jan E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019. Disponível em: <<https://arxiv.org/abs/1903.02428>>. 56

- FRITZ, Cornelius; DORIGATTI, Emilio; RÜGAMER, David. Combining graph neural networks and spatio-temporal disease models to improve the prediction of weekly covid-19 cases in germany. *Scientific Reports*, Nature Publishing Group, v. 12, n. 3930, 2022. 38, 61
- FUKUSHIMA, Kunihiko. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, IEEE, v. 5, n. 4, p. 322–333, 1969. 31
- GEEKSFORGEES. *Deep Learning | Introduction to Long Short-Term Memory (LSTM)*. 2023. <<https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/>>. Acesso em: 21 jan. 2025. 5, 35
- GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. ISBN 978-0262035613. Disponível em: <<http://www.deeplearningbook.org>>. 35
- HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2. ed. New York: Springer, 2009. ISBN 978-0387848570. 27
- HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997. 13, 18, 31, 32, 36
- HY, Truong Son et al. Temporal multiresolution graph neural networks for epidemic prediction. *Proceedings of the 1st Workshop on Healthcare AI and COVID-19, ICML 2022*, 2022. Disponível em: <<https://arxiv.org/abs/2205.14831>>. 61
- HYNDMAN, Rob J.; ATHANASOPOULOS, George. *Forecasting: Principles and Practice*. 2. ed. Melbourne: OTexts, 2018. Disponível em: <<https://otexts.com/fpp2/>>. Acesso em: 21 jan. 2025. 5, 13, 14, 15, 16, 18, 20, 22, 24, 25, 46, 47
- HYNDMAN, Rob J.; KHANDAKAR, Yeasmin. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, v. 27, n. 3, p. 1–22, 2008. 13, 24
- KALITA, Debasish. *What is Recurrent Neural Networks (RNN)?* 2022. <<https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/>>. Acesso em: 21 jan. 2025. 5, 31
- KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Disponível em: <<https://arxiv.org/abs/1412.6980>>. 35
- KIPF, Thomas N.; WELING, Max. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. Disponível em: <<https://arxiv.org/abs/1609.02907>>. 41
- LESAGE, James P.; PACE, R. Kelley. *Introduction to Spatial Econometrics*. Boca Raton, FL, USA: CRC Press, 2009. 38, 40
- LI, Qun et al. Early transmission dynamics in wuhan, china, of novel coronavirus–infected pneumonia. *New England Journal of Medicine*, v. 382, p. 1199–1207, 2020. Disponível em: <<https://doi.org/10.1056/NEJMoa2001316>>. 9
- LI, Yun et al. Graph neural network for spatiotemporal data: Methods and applications. *ACM Computing Surveys*, v. 1, n. 1, p. 1–36, 2023. 48, 60

- MAKRIDAKIS, Spyros; WHEELWRIGHT, Steven C.; HYNDMAN, Rob J. *Forecasting: Methods and Applications*. 3rd. ed. New York, NY, USA: John Wiley & Sons, 1997. ISBN 978-0-471-53233-0. 13
- MCCULLOCH, Warren S; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943. 30
- MINSKY, Marvin; PAPERT, Seymour A. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969. 30
- NAIR, Vinod; HINTON, Geoffrey E. Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (ICML-10)*, p. 807–814, 2010. 31
- PAIUTA, Rhayani Aparecida. *Um estudo sobre os impactos de algumas medidas sanitárias na incidência e mortalidade por COVID-19*. São Carlos: [s.n.], 2022. Trabalho de Conclusão de Curso em Estatística. 52, 55, 61
- PANAGOPOULOS, George; NIKOLENTZOS, Giannis; VAZIRGIANNIS, Michalis. *Transfer Graph Neural Networks for Pandemic Forecasting*. 2021. Disponível em: <<https://arxiv.org/abs/2009.08388>>. 10
- PANAGOPOULOS, George; NIKOLENTZOS, Giannis; VAZIRGIANNIS, Michalis. Transfer graph neural networks for pandemic forecasting. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. Queensland, Australia: ACM, 2021. p. 2724–2733. 48, 61, 83
- PASZKE, Adam et al. *PyTorch: An Open Source Machine Learning Framework*. Internet, 2017. Disponível em: <<https://pytorch.org>>. Acesso em: 21 jan. 2025. 56
- RODRIGO, Joaquin A.; ORTIZ, Javier E. *skforecast*. 2024. Disponível em: <<https://skforecast.org/>>. 5, 56
- ROSENBLATT, Frank. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. 30
- RUMELHART, David E; HINTON, Geoffrey E; WILLIAMS, Ronald J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986. 30
- SAHIN, Samia. *Why Stationarity Matters in Time Series Analysis and How to Achieve It*. 2020. <<https://medium.com/@sahin.samia/why-stationarity-matters-in-time-series-analysis-and-how-to-achieve-it-57646e09b8cc>>. Acesso em: 21 jan. 2025. 5, 16
- SCHWARZ, Gideon. Estimating the dimension of a model. *The Annals of Statistics*, v. 6, n. 2, p. 461–464, 1978. 24
- SILVA, Juliana da. *Métodos Bayesianos aplicados à modelagem espacial e temporal: um estudo de caso*. Tese (Doutorado) — Universidade Federal de São Carlos, São Carlos, 2023. Tese de Doutorado em Biometria. 61

SLUTSKY, Eugen. The summation of random causes as the source of cyclic processes. *Econometrica*, v. 5, n. 2, p. 105–146, 1937. 19

SMITH, Colin; ELLIS, Richard. *pmdarima: ARIMA estimators for Python*. 2019. GitHub Repository. Disponível em: <<https://github.com/alkaline-ml/pmdarima>>. 56

SUMMER, The AI. *Best Graph Neural Network architectures: GCN, GAT, MPNN and more*. 2023. <<https://theaisummer.com/gnn-architectures/>>. Acesso em: 21 jan. 2025. 5, 42

TASHMAN, L. J. Out-of-sample tests of forecasting accuracy: an analysis and review. *International Journal of Forecasting*, v. 16, n. 4, p. 437–450, 2000. 30

VELAVAN, Thirumalaisamy P.; MEYER, Christian G. The covid-19 epidemic. *Tropical Medicine & International Health*, v. 25, n. 3, p. 278–280, 2020. Disponível em: <<https://doi.org/10.1111/tmi.13383>>. 10

World Health Organization. *Weekly epidemiological update on COVID-19 - 25 May 2021*. 2021. <<https://www.who.int/publications/m/item/weekly-epidemiological-update-on-covid-19---25-may-2021>>. Acessado em: 25 de Maio de 2021. 9

WU, Zonghan et al. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, Institute of Electrical and Electronics Engineers (IEEE), v. 32, n. 1, p. 4–24, jan. 2021. ISSN 2162-2388. Disponível em: <<http://dx.doi.org/10.1109/TNNLS.2020.2978386>>. 41

YULE, G. Udny. On a method of investigating periodicities in disturbed series, with special reference to wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, v. 226, p. 267–298, 1927. 18

ZHANG, G.Peter; QI, Min. Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, v. 160, n. 2, p. 501–514, 2005. ISSN 0377-2217. Decision Support Systems in the Internet Age. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0377221703005484>>. 13

Apêndices

A Código dos modelos em Python

Os modelos foram aplicados em dados de treino para realizar previsões no dado de teste.

Os dados de treino, representado por `df_treino_num_T`, possuem apenas valores numéricos da taxa de incidência semanal, cujo as linhas representam as cidades e as colunas o dia.

Os códigos utilizados neste trabalho também estão disponíveis no GitHub¹.

Modelo ARIMA

```

1
2 arima_mse = []
3 arima_predictions = []
4 arima_conf_int_80 = []
5 arima_conf_int_95 = []
6 n = df_treino_num_T.shape[0]
7 n_forecast = 7
8
9 for i in range(n):
10     print(f"Processando Modelo 1 para a linha {i}")
11
12     # Ajusta o modelo ARIMA
13     modelo_arima_teste = auto_arima(df_treino_num_T.iloc[i, :], trace=False,
14                                     error_action='ignore', suppress_warnings=True)
15     modelo_arima_teste.fit(df_treino_num_T.iloc[i, :])
16
17     # Previsões e intervalos de confiança
18     predictions_teste, conf_int_95 = modelo_arima_teste.predict(n_periods=n_forecast,
19                                                                return_conf_int=True, alpha=0.05)
20     _, conf_int_80 = modelo_arima_teste.predict(n_periods=7, return_conf_int=True,
21                                                alpha=0.20)
22
23     # Calcula o erro quadrático médio
24     mse = mean_squared_error(predictions_teste, df_teste_num_T.iloc[i, :].values)
25
26     # Armazena os resultados

```

¹ <https://github.com/ThiagoMoraesRizzieri/projeto_biometria>

```

24     arima_mse.append(mse1)
25     arima_predictions.append(predictions_teste)
26     arima_conf_int_80.append(conf_int_80)
27     arima_conf_int_95.append(conf_int_95)
28
29 # Calcula o MSE médio
30 mse_model_avg = np.mean(arima_mse)

```

Modelo ARIMAX com variável SAR

```

1     sarimax_sar_mse = []
2     sarimax_sar_predictions = []
3     sarimax_sar_conf_int_80 = []
4     sarimax_sar_conf_int_95 = []
5     n = df_treino_num_T.shape[0]
6
7 # Modelo ARIMAX com variável exógena SAR
8
9 for i in range(n):
10     print(f"Processando Modelo SARIMAX para a linha {i}")
11
12     # Ajusta o modelo ARIMA
13     modelo_sarimax_sar = auto_arima(df_treino_num_T.iloc[i, :].values,
14                                     X=predicoes_sar_matrix[i, :].reshape(-1, 1),
15                                     trace=False, error_action='ignore', suppress_warnings
16                                     =True)
17
18     # Previsões e intervalos de confiança
19     predictions_teste, conf_int_95 = modelo_sarimax_sar.predict(n_periods=7,X=
20         df_predicoes_arima.iloc[i, :].values.reshape(-1, 1), return_conf_int=True,
21         alpha=0.05)
22     _, conf_int_80 = modelo_sarimax_sar.predict(n_periods=7,X=df_predicoes_arima.iloc[i
23         , :].values.reshape(-1, 1), return_conf_int=True, alpha=0.20)
24
25     # Calcula o erro quadrático médio
26     mse1 = mean_squared_error(predictions_teste, df_teste_num_T.iloc[i, :].values)
27
28     # Armazena os resultados
29     sarimax_sar_mse.append(mse1)
30     sarimax_sar_predictions.append(predictions_teste)
31     sarimax_sar_conf_int_80.append(conf_int_80)
32     sarimax_sar_conf_int_95.append(conf_int_95)

```

Modelo ARIMAX com variável GNN

```

1     sarimax_gnn_mse = []

```

```

2     sarimax_gnn_predictions = []
3     sarimax_gnn_conf_int_80 = []
4     sarimax_gnn_conf_int_95 = []
5     n = df_treino_num_T.shape[0]
6
7     # Modelo SARIMAX com variável exógena GNN
8
9     for i in range(200,400):
10        print(f"Processando Modelo SARIMAX para a linha {i}")
11
12        # Ajusta o modelo ARIMA
13        modelo_sarimax_gnn = auto_arima(df_treino_num_T.iloc[i, :].values,
14                                       X=predicoes_gnn_matrix[i, :].reshape(-1, 1),
15                                       trace=False, error_action='ignore', suppress_warnings
16                                       =True)
17
18        # Previsões e intervalos de confiança
19        predictions_teste, conf_int_95 = modelo_sarimax_gnn.predict(n_periods=7,X=
20        df_predicoes_gnn_arima.iloc[i, :].values.reshape(-1, 1), return_conf_int=True,
21        alpha=0.05)
22        _, conf_int_80 = modelo_sarimax_gnn.predict(n_periods=7,X=df_predicoes_gnn_arima.
23        iloc[i, :].values.reshape(-1, 1), return_conf_int=True, alpha=0.20)
24
25        # Calcula o erro quadrático médio
26        mse1 = mean_squared_error(predictions_teste, df_teste_num_T.iloc[i, :].values)
27
28        # Armazena os resultados
29        sarimax_gnn_mse.append(mse1)
30        sarimax_gnn_predictions.append(predictions_teste)
31        sarimax_gnn_conf_int_80.append(conf_int_80)
32        sarimax_gnn_conf_int_95.append(conf_int_95)
33
34    # Calcula o MSE médio
35    np.mean(sarimax_gnn_mse)

```

Modelo ARIMA dinâmico utilizando o SAR

```

1     # Modelo dinâmico com SAR
2
3     dynamic_sar_mse = []
4     dynamic_sar_predictions = []
5     dynamic_sar_conf_int_80 = []
6     dynamic_sar_conf_int_95 = []
7     n = df_treino_num_T.shape[0]
8
9     # Modelo dinâmico com SAR
10

```

```
11 for i in range(400, df_treino_num_T.shape[0]): # Limite de 200 cidades para exemplo,
    ajuste conforme necessário
12     print(f"Processando Modelo Dinâmico para a linha {i}")
13
14     # Modelo Linear
15     model_linear = sm.OLS(df_treino_num_T.iloc[i, :].values, predicoes_sar_matrix[i,
        :]).fit()
16     residuals = model_linear.resid
17
18     # Ajuste ARIMA dos resíduos
19     modelo_arima_teste_res = auto_arima(residuals, trace=False, error_action='ignore',
        suppress_warnings=True).fit(residuals)
20
21     # Previsões e intervalos de confiança dos resíduos (ARIMA)
22     predictions_teste_res, conf_int_95_res = modelo_arima_teste_res.predict(n_periods
        =7, return_conf_int=True, alpha=0.05)
23     _, conf_int_80_res = modelo_arima_teste_res.predict(n_periods=7, return_conf_int=
        True, alpha=0.20)
24
25     # Predição do modelo linear
26     fcast_linear = model_linear.predict(df_predicoes_arima.iloc[i, :].values)
27
28     # Soma das previsões: modelo linear + resíduos (ARIMA)
29     fcast_dyn = fcast_linear + predictions_teste_res
30
31     # Calcula o MSE
32     mse = mean_squared_error(fcast_dyn, df_teste_num_T.iloc[i, :].values)
33     dynamic_sar_mse.append(mse)
34
35     # Armazenar as previsões
36     dynamic_sar_predictions.append(fcast_dyn)
37
38     # Intervalos de confiança
39     # Intervalo de 80%
40     conf_int_80_lower = fcast_linear + conf_int_80_res[:, 0]
41     conf_int_80_upper = fcast_linear + conf_int_80_res[:, 1]
42     dynamic_sar_conf_int_80.append((conf_int_80_lower, conf_int_80_upper))
43
44     # Intervalo de 95%
45     conf_int_95_lower = fcast_linear + conf_int_95_res[:, 0]
46     conf_int_95_upper = fcast_linear + conf_int_95_res[:, 1]
47     dynamic_sar_conf_int_95.append((conf_int_95_lower, conf_int_95_upper))
48
49     # Calcula o MSE médio
50     mse_model1_avg = np.mean(dynamic_sar_mse)
51
52     print(f"MSE Médio: {mse_model1_avg}")
```

Modelo ARIMA dinâmico utilizando o GNN

```

1
2 # Modelo dinâmico com GNN
3
4 dynamic_gnn_mse = []
5 dynamic_gnn_predictions = []
6 dynamic_gnn_conf_int_80 = []
7 dynamic_gnn_conf_int_95 = []
8 n = df_treino_num_T.shape[0]
9
10 for i in range(n):
11     print(f"Processando Modelo Dinâmico para a linha {i}")
12
13     # Modelo Linear
14     model_linear = sm.OLS(df_treino_num_T.iloc[i, :].values, predicoes_gnn_matrix[i,
15         :]).fit()
16     residuals = model_linear.resid
17
18     # Ajuste ARIMA dos resíduos
19     modelo_arima_teste_res = auto_arima(residuals, trace=False, error_action='ignore',
20         suppress_warnings=True).fit(residuals)
21
22     # Previsões e intervalos de confiança dos resíduos (ARIMA)
23     predictions_teste_res, conf_int_95_res = modelo_arima_teste_res.predict(n_periods
24         =92, return_conf_int=True, alpha=0.05)
25     _, conf_int_80_res = modelo_arima_teste_res.predict(n_periods=92, return_conf_int=
26         True, alpha=0.20)
27
28     # Predição do modelo linear
29     fcast_linear = model_linear.predict(df_predicoes_gnn_arima.iloc[i, :].values)
30
31     # Soma das previsões: modelo linear + resíduos (ARIMA)
32     fcast_dyn = fcast_linear + predictions_teste_res
33
34     # Calcula o MSE
35     mse = mean_squared_error(fcast_dyn, df_teste_num_T.iloc[i, :].values)
36     dynamic_gnn_mse.append(mse)
37
38     # Armazenar as previsões
39     dynamic_gnn_predictions.append(fcast_dyn)
40
41     # Intervalos de confiança
42     # Intervalo de 80%
43     conf_int_80_lower = fcast_linear + conf_int_80_res[:, 0]
44     conf_int_80_upper = fcast_linear + conf_int_80_res[:, 1]
45     dynamic_gnn_conf_int_80.append((conf_int_80_lower, conf_int_80_upper))

```

```

43     # Intervalo de 95%
44     conf_int_95_lower = fcast_linear + conf_int_95_res[:, 0]
45     conf_int_95_upper = fcast_linear + conf_int_95_res[:, 1]
46     dynamic_gnn_conf_int_95.append((conf_int_95_lower, conf_int_95_upper))
47
48 # Calcula o MSE médio
49 np.mean(dynamic_gnn_mse)

```

Modelo Random Forest

```

1  # Inicializar listas para armazenar os resultados
2  mse_rf_model = []
3  rf_predictions = []
4  rf_conf_int_80 = []
5  rf_conf_int_95 = []
6
7  # Número de linhas do dataframe
8  n = df_treino_num_T.shape[0]
9
10 for i in range(0,1):
11     print(f"Processando Modelo de Floresta Aleatória para a linha {i}")
12
13     # Transformar os dados de treinamento e teste em numpy arrays
14     y_train = df_treino_num_T.iloc[i, :].values.reshape(-1, 1)
15     y_test = df_teste_num_T.iloc[i, :].values.reshape(-1, 1)
16
17     # Escalar os dados
18     scaler = MinMaxScaler(feature_range=(0, 1))
19     scaler.fit(y_train)
20     y_train_scaled = pd.Series(scaler.transform(y_train).flatten())
21
22     # Criar e ajustar o modelo de previsão para y com a variável exógena prevista
23     forecaster = ForecasterAutoreg(
24         regressor = RandomForestRegressor(random_state=123),
25         lags = 7
26     )
27
28     forecaster.fit(y=y_train_scaled)
29
30     # Previsões com a variável exógena
31     predictions_scaled = forecaster.predict_bootstrapping(steps=30, n_boot=30,
32         random_state=123, in_sample_residuals=True)
33
34     # Inverter a escala das previsões
35     predictions_sem = scaler.inverse_transform(predictions_scaled.mean(axis=1).values.
36         reshape(-1, 1)).flatten()

```

```

36     # # Calcular o MSE
37     # mse_rf = mean_squared_error(predictions_sem, y_test.flatten())
38     # mse_rf_model.append(mse_rf)
39     rf_predictions.append(predictions_sem)
40
41     # Intervalos de previsão de 80% e 95%
42     interval_80_sem = forecaster.predict_interval(
43         steps=92,
44         interval=[10, 90],
45         n_boot=30
46     )
47
48     interval_95_sem = forecaster.predict_interval(
49         steps=92,
50         interval=[2.5, 97.5],
51         n_boot=30
52     )
53
54     # Inverter a escala dos intervalos de previsão
55     interval_80_sem['lower_bound'] = scaler.inverse_transform(interval_80_sem['
56         lower_bound'].values.reshape(-1, 1)).flatten()
57     interval_80_sem['upper_bound'] = scaler.inverse_transform(interval_80_sem['
58         upper_bound'].values.reshape(-1, 1)).flatten()
59
60     interval_95_sem['lower_bound'] = scaler.inverse_transform(interval_95_sem['
61         lower_bound'].values.reshape(-1, 1)).flatten()
62     interval_95_sem['upper_bound'] = scaler.inverse_transform(interval_95_sem['
63         upper_bound'].values.reshape(-1, 1)).flatten()
64
65     # Armazenar intervalos de previsão
66     rf_conf_int_80.append(interval_80_sem)
67     rf_conf_int_95.append(interval_95_sem)
68
69     # Calcular o MSE médio para o modelo de floresta aleatória
70     mse_rf_avg = np.mean(mse_rf_model)
71
72     print(f"MSE Médio do Modelo de Floresta Aleatória: {mse_rf_avg}")

```

Modelo Random Forest com variável exógena SAR

```

1     # Inicializar lista para armazenar as previsões da variável exógena
2     exog_predictions_scaled_all = []
3
4     # Loop para prever a variável exógena para cada cidade
5     for i in range(644):
6         print(f"Previendo a variável exógena para a cidade {i}")
7

```

```
8     # Extrair a variável exógena (por exemplo, predicoes_sar_matrix) para a cidade i
9     exog_train = predicoes_sar_matrix[i, :].reshape(-1, 1)
10
11     # Escalar a variável exógena
12     scaler_exog = MinMaxScaler(feature_range=(0, 1))
13     exog_train_scaled = scaler_exog.fit_transform(exog_train).flatten()
14
15     # Previsão da variável exógena (SAR)
16     exog_forecaster = ForecasterAutoreg(
17         regressor = RandomForestRegressor(random_state=123),
18         lags = 7
19     )
20
21     # Ajustar o modelo para a variável exógena
22     exog_forecaster.fit(y=pd.Series(exog_train_scaled, name='Defasagem_espacial_SAR'))
23
24     # Prever a variável exógena para o período de teste
25     exog_predictions_scaled = exog_forecaster.predict(steps=7)
26
27     # Armazenar as previsões como uma série com o mesmo nome
28     exog_predictions_scaled_all.append(pd.Series(exog_predictions_scaled, name='
29         Defasagem_espacial_SAR'))
30
31 mse_rf_model_sar = []
32 rf_predictions_sar = []
33 rf_conf_int_80_sar = []
34 rf_conf_int_95_sar = []
35
36 # Loop para prever y usando as previsões da variável exógena
37 for i in range(600, df_treino_num_T.shape[0]):
38     print(f"Prevendo y para a cidade {i}")
39
40     # Extrair y_train e y_test
41     y_train = df_treino_num_T.iloc[i, :].values.reshape(-1, 1)
42     y_test = df_teste_num_T.iloc[i, :].values.reshape(-1, 1)
43
44     # Escalar y
45     scaler = MinMaxScaler(feature_range=(0, 1))
46     y_train_scaled = scaler.fit_transform(y_train).flatten()
47
48     # Usar a variável exógena real (predicoes_gnn_matrix) para o treinamento
49     exog_train = predicoes_sar_matrix[i, :].reshape(-1, 1)
50
51     # Escalar a variável exógena para o treinamento
52     scaler_exog_train = MinMaxScaler(feature_range=(0, 1))
53     exog_train_scaled = scaler_exog_train.fit_transform(exog_train).flatten()
```

```
54 # **Adicionar um nome à série exógena**
55 exog_train_scaled = pd.Series(exog_train_scaled, name='Defasagem_espacial_SAR')
56
57 # Recuperar as previsões da variável exógena armazenadas anteriormente para a
    previsão
58 exog_predictions_scaled = exog_predictions_scaled_all[i]
59
60 # Criar e ajustar o modelo de previsão para y com a variável exógena real (para o
    treinamento)
61 forecaster = ForecasterAutoreg(
62     regressor = RandomForestRegressor(random_state=123),
63     lags = 7
64 )
65
66 # Ajuste do modelo de previsão com a variável exógena real para o treino
67 forecaster.fit(y=pd.Series(y_train_scaled), exog=exog_train_scaled)
68
69 # Fazer previsões para y com as previsões da variável exógena armazenadas
70 predictions_scaled = forecaster.predict_bootstrapping(steps=7, exog=
    exog_predictions_scaled, n_boot=30, random_state=123, in_sample_residuals=True)
71
72 # Inverter a escala das previsões
73 predictions_sar = scaler.inverse_transform(predictions_scaled.mean(axis=1).values.
    reshape(-1, 1)).flatten()
74
75 # Calcular o MSE
76 mse_rf = mean_squared_error(predictions_sar, y_test.flatten())
77 mse_rf_model_sar.append(mse_rf)
78 rf_predictions_sar.append(predictions_sar)
79
80 # Intervalos de previsão
81 interval_80_sar = forecaster.predict_interval(
82     steps=92,
83     exog=exog_predictions_scaled,
84     interval=[10, 90],
85     n_boot=30
86 )
87
88 interval_95_sar = forecaster.predict_interval(
89     steps=92,
90     exog=exog_predictions_scaled,
91     interval=[2.5, 97.5],
92     n_boot=30
93 )
94
95 # Corrigir o uso de 'inverse_transform' nos intervalos
```

```

96     interval_80_sar['lower_bound'] = scaler.inverse_transform(interval_80_sar['
        lower_bound'].values.reshape(-1, 1)).flatten()
97     interval_95_sar['lower_bound'] = scaler.inverse_transform(interval_95_sar['
        lower_bound'].values.reshape(-1, 1)).flatten()
98
99     interval_80_sar['upper_bound'] = scaler.inverse_transform(interval_80_sar['
        upper_bound'].values.reshape(-1, 1)).flatten()
100    interval_95_sar['upper_bound'] = scaler.inverse_transform(interval_95_sar['
        upper_bound'].values.reshape(-1, 1)).flatten()
101
102    # Armazenar intervalos de previsão
103    rf_conf_int_80_sar.append(interval_80_sar)
104    rf_conf_int_95_sar.append(interval_95_sar)
105
106    # Calcular o MSE médio para o modelo de floresta aleatória
107    mse_rf_avg = np.mean(mse_rf_model_sar)
108
109    print(f"MSE Médio do Modelo de Floresta Aleatória: {mse_rf_avg}")

```

Modelo Random Forest com variável exógena GNN

```

1  # Inicializar lista para armazenar as previsões da variável exógena
2  rf_exog_gnn = []
3
4  # Loop para prever a variável exógena para cada cidade
5  for i in range(644):
6      print(f"Prevendo a variável exógena para a cidade {i}")
7
8      # Extrair a variável exógena (por exemplo, predicoes_sar_matrix) para a cidade i
9      exog_train = predicoes_gnn_matrix[i, :].reshape(-1, 1)
10
11     # Escalar a variável exógena
12     scaler_exog = MinMaxScaler(feature_range=(0, 1))
13     exog_train_scaled = scaler_exog.fit_transform(exog_train).flatten()
14
15     # Previsão da variável exógena (GNN)
16     exog_forecaster = ForecasterAutoreg(
17         regressor = RandomForestRegressor(random_state=123),
18         lags = 7
19     )
20
21     # Ajustar o modelo para a variável exógena
22     exog_forecaster.fit(y=pd.Series(exog_train_scaled, name='Defasagem_espacial_GNN'))
23
24     # Prever a variável exógena para o período de teste
25     exog_predictions_scaled = exog_forecaster.predict(steps=7)
26

```

```
27     # Armazenar as previsões como uma série com o mesmo nome
28     rf_exog_gnn.append(pd.Series(exog_predictions_scaled, name='Defasagem_espacial_GNN'
29                                 ))
30
31 mse_rf_model_gnn = []
32 rf_predictions_gnn = []
33 rf_conf_int_80_gnn = []
34 rf_conf_int_95_gnn = []
35
36 # Loop para prever y usando as previsões da variável exógena
37 for i in range(600, df_treino_num_T.shape[0]):
38     print(f"Prevendo y para a cidade {i}")
39
40     # Extrair y_train e y_test
41     y_train = df_treino_num_T.iloc[i, :].values.reshape(-1, 1)
42     y_test = df_teste_num_T.iloc[i, :].values.reshape(-1, 1)
43
44     # Escalar y
45     scaler = MinMaxScaler(feature_range=(0, 1))
46     y_train_scaled = scaler.fit_transform(y_train).flatten()
47
48     # Usar a variável exógena real (predicoes_gnn_matrix) para o treinamento
49     exog_train = predicoes_gnn_matrix[i, :].reshape(-1, 1)
50
51     # Escalar a variável exógena para o treinamento
52     scaler_exog_train = MinMaxScaler(feature_range=(0, 1))
53     exog_train_scaled = scaler_exog_train.fit_transform(exog_train).flatten()
54
55     # **Adicionar um nome à série exógena**
56     exog_train_scaled = pd.Series(exog_train_scaled, name='Defasagem_espacial_GNN')
57
58     # Recuperar as previsões da variável exógena armazenadas anteriormente para a
59     # previsão
60     exog_predictions_scaled = rf_exog_gnn[i]
61
62     # Criar e ajustar o modelo de previsão para y com a variável exógena real (para o
63     # treinamento)
64     forecaster = ForecasterAutoreg(
65         regressor = RandomForestRegressor(random_state=123),
66         lags = 7
67     )
68
69     # Ajuste do modelo de previsão com a variável exógena real para o treino
70     forecaster.fit(y=pd.Series(y_train_scaled), exog=exog_train_scaled)
71
72     # Fazer previsões para y com as previsões da variável exógena armazenadas
```

```
70 predictions_scaled = forecaster.predict_bootstrapping(steps=7, exog=
    exog_predictions_scaled, n_boot=30, random_state=123, in_sample_residuals=True)
71
72 # Inverter a escala das previsões
73 predictions_gnn = scaler.inverse_transform(predictions_scaled.mean(axis=1).values.
    reshape(-1, 1)).flatten()
74
75 # Calcular o MSE
76 mse_rf = mean_squared_error(predictions_gnn, y_test.flatten())
77 mse_rf_model_gnn.append(mse_rf)
78 rf_predictions_gnn.append(predictions_gnn)
79
80 # Intervalos de previsão
81 interval_80_gnn = forecaster.predict_interval(
82     steps=92,
83     exog=exog_predictions_scaled,
84     interval=[10, 90],
85     n_boot=30
86 )
87
88 interval_95_gnn = forecaster.predict_interval(
89     steps=92,
90     exog=exog_predictions_scaled,
91     interval=[2.5, 97.5],
92     n_boot=30
93 )
94
95 # Corrigir o uso de 'inverse_transform' nos intervalos
96 interval_80_gnn['lower_bound'] = scaler.inverse_transform(interval_80_gnn['
    lower_bound'].values.reshape(-1, 1)).flatten()
97 interval_95_gnn['lower_bound'] = scaler.inverse_transform(interval_95_gnn['
    lower_bound'].values.reshape(-1, 1)).flatten()
98
99 interval_80_gnn['upper_bound'] = scaler.inverse_transform(interval_80_gnn['
    upper_bound'].values.reshape(-1, 1)).flatten()
100 interval_95_gnn['upper_bound'] = scaler.inverse_transform(interval_95_gnn['
    upper_bound'].values.reshape(-1, 1)).flatten()
101
102 # Armazenar intervalos de previsão
103 rf_conf_int_80_gnn.append(interval_80_gnn)
104 rf_conf_int_95_gnn.append(interval_95_gnn)
105
106 # Calcular o MSE médio para o modelo de floresta aleatória
107 mse_rf_avg = np.mean(mse_rf_model_gnn)
108
109 print(f"MSE Médio do Modelo de Floresta Aleatória: {mse_rf_avg}")
```

Modelo LSTM

```
1 residuals_by_column = {}
2 predictions_by_column = {}
3 n = df_treino_num_T.shape[0]
4
5 n_forecast = 7 # Número total de passos que você quer prever durante o pré-treinamento
6 n_lookback = 7 # Número de passos anteriores para usar como entrada
7
8 # Passo 2: Treinamento individual por cidade
9 for cidade_idx in range(n): # Itera sobre todas as cidades
10     print(f"Treinando Modelo para a cidade {cidade_idx}")
11     y = df_treino_num_T.iloc[cidade_idx, :].values.reshape(-1, 1)
12
13     # Scale the data
14     scaler = MinMaxScaler(feature_range=(0, 1))
15     scaler = scaler.fit(y)
16     y = scaler.transform(y)
17
18     # Generate the input and output sequences
19     X = []
20     Y = []
21
22     for seq_idx in range(n_lookback, len(y) - n_forecast + 1): # Use uma variável
23         # diferente aqui
24         X.append(y[seq_idx - n_lookback: seq_idx])
25         Y.append(y[seq_idx: seq_idx + n_forecast])
26
27     X = np.array(X)
28     Y = np.array(Y)
29
30     # Define e compile o modelo individual
31     model = Sequential()
32     model.add(LSTM(units=50, return_sequences=True, input_shape=(n_lookback, 1)))
33     model.add(LSTM(units=50))
34     model.add(Dense(n_forecast))
35
36     # Continue o treinamento na série temporal da cidade
37     model.compile(loss='mean_squared_error', optimizer='adam')
38     model.fit(X, Y, epochs=100, batch_size=32, verbose=0) # Ajuste os epochs conforme
39     # necessário
40
41     # Aplicar o modelo aos dados de treino e armazenar previsões e resíduos como antes
42     Y_pred_train = model.predict(X).reshape(-1, n_forecast, 1)
43     residuals_train = Y - Y_pred_train
44
45     # Inversão da escala para os resíduos
46     residuals_train = scaler.inverse_transform(residuals_train.reshape(-1, 1))
```

```
45     residuals_by_column[cidade_idx] = residuals_train # Salve os resíduos no dicionário
46
47     # Generate the forecasts
48     X_ = y[-n_lookback:] # Last available input sequence
49     X_ = X_.reshape(1, n_lookback, 1)
50
51     Y_ = model.predict(X_).reshape(-1, 1)
52     Y_ = scaler.inverse_transform(Y_)
53     predictions_by_column[cidade_idx] = Y_
54
55 # Criando os intervalos de previsão através do bootstrapping (95% e 80%)
56
57 lower_bound_95 = {}
58 upper_bound_95 = {}
59 lower_bound_80 = {}
60 upper_bound_80 = {}
61
62 # Número de simulações bootstrap
63 n_simulations = 100
64
65 # Inicializar dicionário para trajetórias bootstrap
66 bootstrap_trajectories = {}
67
68 # Iterar sobre cada cidade (coluna)
69 for col in residuals_by_column:
70     # Inicialize a lista para armazenar as trajetórias bootstrap de cada cidade
71     bootstrap_trajectories[col] = []
72
73     # Simulações bootstrap
74     for i in range(n_simulations):
75         # Inicializar vetor de previsões bootstrap
76         bootstrap_predictions = predictions_by_column[col].copy()
77
78         # Iterar sobre o horizonte de previsão
79         for t in range(len(predictions_by_column[col])):
80             # Amostrar resíduos para serem somados (progressivamente mais resíduos para
81                 t maiores)
82             sampled_residuals = np.random.choice(
83                 residuals_by_column[col].flatten(), size=t+1, replace=True
84             )
85
86             # Somar os resíduos ao valor da previsão correspondente
87             bootstrap_predictions[t] += np.sum(sampled_residuals)
88
89             # Adicionar a trajetória bootstrap gerada
90             bootstrap_trajectories[col].append(bootstrap_predictions)
```

```

91     # Converter as trajetórias bootstrap em um array para facilitar o cálculo
92     bootstrap_trajectories[col] = np.array(bootstrap_trajectories[col])
93
94     # Calcular intervalos de previsão de 95% (percentis 2.5 e 97.5)
95     lower_bound_95[col] = np.percentile(bootstrap_trajectories[col], 2.5, axis=0)
96     upper_bound_95[col] = np.percentile(bootstrap_trajectories[col], 97.5, axis=0)
97
98     # Calcular intervalos de previsão de 80% (percentis 10 e 90)
99     lower_bound_80[col] = np.percentile(bootstrap_trajectories[col], 10, axis=0)
100    upper_bound_80[col] = np.percentile(bootstrap_trajectories[col], 90, axis=0)

```

Modelo LSTM com variável SAR

```

1  residuals_lstm_sar = {}
2  predictions_lstm_sar = {}
3
4  # Modelo LSTM com a variável exógena
5
6  n_forecast = 7 # Número total de passos que você quer prever durante o pré-treinamento
7  n_lookback = 7 # Número de passos anteriores para usar como entrada
8
9  # Passo 2: Treinamento individual por cidade
10 for cidade_idx in range(n): # Itera sobre todas as cidades
11     print(f"Treinando Modelo LSTM para a cidade {cidade_idx}")
12
13     # Variável dependente (taxa de COVID-19)
14     y = df_treino_num_T.iloc[cidade_idx, :].values.reshape(-1, 1)
15
16     # Variável exógena (previsão do SAR para a mesma cidade)
17     exog_var = predicoes_sar_matrix[cidade_idx, :].reshape(-1, 1)
18
19     # Escalar os dados
20     scaler_y = MinMaxScaler(feature_range=(0, 1))
21     scaler_exog = MinMaxScaler(feature_range=(0, 1))
22     y_scaled = scaler_y.fit_transform(y)
23     exog_scaled = scaler_exog.fit_transform(exog_var)
24
25     # Gerar as sequências de entrada (X) e saída (Y)
26     X = []
27     X_exog = [] # Sequências para a variável exógena
28     Y = []
29
30     for seq_idx in range(n_lookback, len(y_scaled) - n_forecast + 1):
31         X.append(y_scaled[seq_idx - n_lookback: seq_idx])
32         X_exog.append(exog_scaled[seq_idx - n_lookback: seq_idx]) # Variável exógena
33         # correspondente
34         Y.append(y_scaled[seq_idx: seq_idx + n_forecast])

```

```
34
35     # Converter as listas em arrays
36     X = np.array(X)
37     X_exog = np.array(X_exog) # Sequência da variável exógena
38     Y = np.array(Y)
39
40     # Concatenar a variável exógena aos inputs do LSTM
41     X_combined = np.concatenate([X, X_exog], axis=2) # Combinar variável dependente com
        exógena
42
43     # Definir e compilar o modelo LSTM
44     model = Sequential()
45     model.add(LSTM(units=50, return_sequences=True, input_shape=(n_lookback, 2))) # 2
        inputs: dependente + exógeno
46     model.add(LSTM(units=50))
47     model.add(Dense(n_forecast)) # Prever múltiplos passos à frente
48
49     # Compilar o modelo
50     model.compile(loss='mean_squared_error', optimizer='adam')
51
52     # Treinar o modelo
53     model.fit(X_combined, Y, epochs=100, batch_size=32, verbose=0)
54
55     # Previsão nos dados de treino
56     Y_pred_train = model.predict(X_combined).reshape(-1, n_forecast, 1)
57     residuals_train = Y - Y_pred_train
58
59     # Inversão da escala para os resíduos
60     residuals_train = scaler_y.inverse_transform(residuals_train.reshape(-1, 1))
61     residuals_lstm_sar[cidade_idx] = residuals_train # Salvar os resíduos
62
63     # Previsão para os dados de teste
64     X_ = y_scaled[-n_lookback:]
65     X_exog_ = exog_scaled[-n_lookback:]
66     X_combined_ = np.concatenate([X_.reshape(1, n_lookback, 1), X_exog_.reshape(1,
        n_lookback, 1)], axis=2)
67
68     Y_ = model.predict(X_combined_).reshape(-1, 1)
69     Y_ = scaler_y.inverse_transform(Y_)
70     predictions_lstm_sar[cidade_idx] = Y_
71
72     lower_bound_sar = {}
73     upper_bound_sar = {}
74     lower_bound_80_sar = {}
75     upper_bound_80_sar = {}
76
77     # Número de simulações bootstrap
```

```

78 n_simulations = 100
79
80 # Inicializar dicionário para trajetórias bootstrap
81 bootstrap_trajectories = {}
82
83 # Iterar sobre cada cidade (coluna)
84 for col in residuals_lstm_sar:
85     # Inicialize a lista para armazenar as trajetórias bootstrap de cada cidade
86     bootstrap_trajectories[col] = []
87
88     # Simulações bootstrap
89     for i in range(n_simulations):
90         # Inicializar vetor de previsões bootstrap
91         bootstrap_predictions = predictions_lstm_sar[col].copy()
92
93         # Iterar sobre o horizonte de previsão
94         for t in range(len(predictions_lstm_sar[col])):
95             # Amostrar resíduos para serem somados (progressivamente mais resíduos para
96                 t maiores)
97             sampled_residuals = np.random.choice(
98                 residuals_lstm_sar[col].flatten(), size=t+1, replace=True
99             )
100
101             # Somar os resíduos ao valor da previsão correspondente
102             bootstrap_predictions[t] += np.sum(sampled_residuals)
103
104             # Adicionar a trajetória bootstrap gerada
105             bootstrap_trajectories[col].append(bootstrap_predictions)
106
107         # Converter as trajetórias bootstrap em um array para facilitar o cálculo
108         bootstrap_trajectories[col] = np.array(bootstrap_trajectories[col])
109
110         # Calcular estatísticas resumo das simulações bootstrap
111         mean_trajectory_sar[col] = np.mean(bootstrap_trajectories[col], axis=0)
112         std_dev_trajectory_sar[col] = np.std(bootstrap_trajectories[col], axis=0)
113
114         # Calcular intervalos de previsão de 95% (percentis 2.5 e 97.5)
115         lower_bound_sar[col] = np.percentile(bootstrap_trajectories[col], 2.5, axis=0)
116         upper_bound_sar[col] = np.percentile(bootstrap_trajectories[col], 97.5, axis=0)
117
118         # Calcular intervalos de previsão de 80% (percentis 10 e 90)
119         lower_bound_80_sar[col] = np.percentile(bootstrap_trajectories[col], 10, axis=0)
120         upper_bound_80_sar[col] = np.percentile(bootstrap_trajectories[col], 90, axis=0)

```

Modelo LSTM com variável GNN

```

1 residuals_lstm_gnn = {}

```

```

2
3 # Modelo LSTM com a variável exógena
4
5 n_forecast = 7 # Número total de passos que você quer prever durante o pré-treinamento
6 n_lookback = 7 # Número de passos anteriores para usar como entrada
7
8 # Passo 2: Treinamento individual por cidade
9 for cidade_idx in range(516, df_treino_num_T.shape[0]): # Itera sobre todas as cidades
10     print(f"Treinando Modelo LSTM para a cidade {cidade_idx}")
11
12     # Variável dependente (taxa de COVID-19)
13     y = df_treino_num_T.iloc[cidade_idx, :].values.reshape(-1, 1)
14
15     # Variável exógena (previsão do SAR para a mesma cidade)
16     exog_var = predicoes_gnn_matrix[cidade_idx, :].reshape(-1, 1)
17
18     # Escalar os dados
19     scaler_y = MinMaxScaler(feature_range=(0, 1))
20     scaler_exog = MinMaxScaler(feature_range=(0, 1))
21     y_scaled = scaler_y.fit_transform(y)
22     exog_scaled = scaler_exog.fit_transform(exog_var)
23
24     # Gerar as sequências de entrada (X) e saída (Y)
25     X = []
26     X_exog = [] # Sequências para a variável exógena
27     Y = []
28
29     for seq_idx in range(n_lookback, len(y_scaled) - n_forecast + 1):
30         X.append(y_scaled[seq_idx - n_lookback: seq_idx])
31         X_exog.append(exog_scaled[seq_idx - n_lookback: seq_idx]) # Variável exógena
32         Y.append(y_scaled[seq_idx: seq_idx + n_forecast])
33         correspondente
34
35     # Converter as listas em arrays
36     X = np.array(X)
37     X_exog = np.array(X_exog) # Sequência da variável exógena
38     Y = np.array(Y)
39
40     # Concatenar a variável exógena aos inputs do LSTM
41     X_combined = np.concatenate([X, X_exog], axis=2) # Combinar variável dependente com
42     exógena
43
44     # Definir e compilar o modelo LSTM
45     model = Sequential()
46     model.add(LSTM(units=50, return_sequences=True, input_shape=(n_lookback, 2))) # 2
47     inputs: dependente + exógeno
48     model.add(LSTM(units=50))

```

```
46 model.add(Dense(n_forecast)) # Prever múltiplos passos à frente
47
48 # Compilar o modelo
49 model.compile(loss='mean_squared_error', optimizer='adam')
50
51 # Treinar o modelo
52 model.fit(X_combined, Y, epochs=100, batch_size=32, verbose=0)
53
54 # Previsão nos dados de treino
55 Y_pred_train = model.predict(X_combined).reshape(-1, n_forecast, 1)
56 residuals_train = Y - Y_pred_train
57
58 # Inversão da escala para os resíduos
59 residuals_train = scaler_y.inverse_transform(residuals_train.reshape(-1, 1))
60 residuals_lstm_gnn[cidade_idx] = residuals_train # Salvar os resíduos
61
62 # # Previsão para os dados de teste
63 # X_ = y_scaled[-n_lookback:]
64 # X_exog_ = exog_scaled[-n_lookback:]
65 # X_combined_ = np.concatenate([X_.reshape(1, n_lookback, 1), X_exog_.reshape(1,
66     n_lookback, 1)], axis=2)
67
68 # Y_ = model.predict(X_combined_).reshape(-1, 1)
69 # Y_ = scaler_y.inverse_transform(Y_)
70 # predictions_lstm_gnn[cidade_idx] = Y_
71
72 lower_bound_gnn = {}
73 upper_bound_gnn = {}
74 lower_bound_80_gnn = {}
75 upper_bound_80_gnn = {}
76
77 # Número de simulações bootstrap
78 n_simulations = 100
79
80 # Inicializar dicionário para trajetórias bootstrap
81 bootstrap_trajectories = {}
82
83 # Iterar sobre cada cidade (coluna)
84 for col in residuals_lstm_gnn:
85     # Inicialize a lista para armazenar as trajetórias bootstrap de cada cidade
86     bootstrap_trajectories[col] = []
87
88     # Simulações bootstrap
89     for i in range(n_simulations):
90         # Inicializar vetor de previsões bootstrap
91         bootstrap_predictions = predictions_lstm_gnn[col].copy()
```

```

92     # Iterar sobre o horizonte de previsão
93     for t in range(len(predictions_lstm_gnn[col])):
94         # Amostrar resíduos para serem somados (progressivamente mais resíduos para
          t maiores)
95         sampled_residuals = np.random.choice(
96             residuals_lstm_gnn[col].flatten(), size=t+1, replace=True
97         )
98
99         # Somar os resíduos ao valor da previsão correspondente
100        bootstrap_predictions[t] += np.sum(sampled_residuals)
101
102        # Adicionar a trajetória bootstrap gerada
103        bootstrap_trajectories[col].append(bootstrap_predictions)
104
105        # Converter as trajetórias bootstrap em um array para facilitar o cálculo
106        bootstrap_trajectories[col] = np.array(bootstrap_trajectories[col])
107
108        # Calcular estatísticas resumo das simulações bootstrap
109        mean_trajectory_gnn[col] = np.mean(bootstrap_trajectories[col], axis=0)
110        std_dev_trajectory_gnn[col] = np.std(bootstrap_trajectories[col], axis=0)
111
112        # Calcular intervalos de previsão de 95% (percentis 2.5 e 97.5)
113        lower_bound_gnn[col] = np.percentile(bootstrap_trajectories[col], 2.5, axis=0)
114        upper_bound_gnn[col] = np.percentile(bootstrap_trajectories[col], 97.5, axis=0)
115
116        # Calcular intervalos de previsão de 80% (percentis 10 e 90)
117        lower_bound_80_gnn[col] = np.percentile(bootstrap_trajectories[col], 10, axis=0)
118        upper_bound_80_gnn[col] = np.percentile(bootstrap_trajectories[col], 90, axis=0)

```

Modelo LSTM+GNN em arquitetura única

```

1  # Definir a classe de GNN com camadas de propagação de mensagens customizadas
2  class CustomMessagePassing(nn.Module):
3      def __init__(self, in_channels, out_channels):
4          super(CustomMessagePassing, self).__init__()
5          self.lin = nn.Linear(in_channels, out_channels)
6          self.relu = nn.ReLU()
7
8      def forward(self, x, edge_index, edge_weight):
9          # Verificar as dimensões de x
10         print(f'Dimensões de entrada em GNN: {x.shape}') # Deve ser (num_nodes,
          in_channels)
11
12         # Aplicar a transformação linear
13         out = self.lin(x) # Certificar que x tem dimensões (num_nodes, in_channels)
14         out = self.relu(out)
15         return out

```

[illegible]

```

59
60     # Concatenar os embeddings espaciais com as séries temporais
61     combined_input = torch.cat((temporal_seq, gnn_embeddings_expanded), dim=-1)
62
63     # Passa pela LSTM para prever os próximos passos (92 previsões)
64     output = lstm_model(combined_input) # Agora retorna 92 previsões por cidade
65     loss = criterion(output, targets)
66
67     loss.backward()
68     optimizer.step()
69
70     if epoch % 10 == 0:
71         print(f'Epoch [{epoch}/{num_epochs}], Loss: {loss.item():.4f}')
72
73 # Treinamento e Previsão
74 n_forecast = 7
75 n_lookback = 7
76
77 # Dicionários para armazenar os resíduos e previsões
78 residuals_gnn_lstm = {}
79 predictions_gnn_lstm = {}
80
81 # Iterando sobre cada cidade (linha do df_treino_num_T)
82 for cidade_idx in range(n):
83     print(f"Treinando GNN+LSTM para a cidade {cidade_idx}")
84
85     # Dados temporais
86     y = df_treino_num_T.iloc[cidade_idx, :].values.reshape(-1, 1)
87     scaler = MinMaxScaler(feature_range=(0, 1))
88     y_scaled = scaler.fit_transform(y)
89
90     # Sequências temporais para treino
91     X_train = []
92     Y_train = []
93     for seq_idx in range(n_lookback, len(y_scaled) - n_forecast + 1):
94         X_train.append(y_scaled[seq_idx - n_lookback: seq_idx])
95         Y_train.append(y_scaled[seq_idx: seq_idx + n_forecast])
96
97     X_train = np.array(X_train)
98     Y_train = np.array(Y_train)
99
100     # Preparar dados de entrada para teste
101     X_test = y_scaled[-n_lookback:]
102
103     # **Ajuste no GNN:**
104     node_features = torch.tensor(df_treino_num_T.mean(axis=1).values.reshape(-1, 1),
        dtype=torch.float32) # Reduzido a features dos nós

```

```

105     edge_index, edge_weight = from_scipy_sparse_matrix(nb.sparse)
106
107     graph_data = Data(x=node_features, edge_index=edge_index, edge_attr=edge_weight)
108
109     # Converter sequências temporais para tensores
110     temporal_seq_train = torch.tensor(X_train, dtype=torch.float32)
111     targets_train = torch.tensor(Y_train[:, -1], dtype=torch.float32)
112
113     # Inicializar modelos
114     gnn_model = GNN(in_channels=1, hidden_channels=16, out_channels=16)
115     lstm_model = LSTMModel(input_size=1 + 16, hidden_size=32, output_size=1) #
116         Concatenando a série temporal com o embedding
117
118     optimizer = optim.Adam(list(gnn_model.parameters()) + list(lstm_model.parameters())
119                             , lr=0.001)
120
121     # Treinar o modelo GNN+LSTM
122     train_gnn_lstm(gnn_model, lstm_model, optimizer, graph_data, temporal_seq_train,
123                   targets_train, cidade_idx, num_epochs=100)
124
125     # Previsão nos dados de treino para calcular os resíduos
126     with torch.no_grad():
127         gnn_model.eval()
128         lstm_model.eval()
129         gnn_embeddings = gnn_model(graph_data)
130         gnn_embedding_city = gnn_embeddings[cidade_idx].unsqueeze(0)
131         gnn_embeddings_expanded = gnn_embedding_city.unsqueeze(1).repeat(
132             temporal_seq_train.shape[0], temporal_seq_train.shape[1], 1)
133
134         # Concatenar a entrada temporal com os embeddings da cidade
135         combined_input = torch.cat((temporal_seq_train, gnn_embeddings_expanded), dim
136                                     ==-1)
137
138         # Previsão dos próximos 92 passos
139         Y_pred_train = lstm_model(combined_input).numpy()
140
141     # Calcula os resíduos no treino
142     residuals_train = targets_train.numpy() - Y_pred_train
143     residuals_train = scaler.inverse_transform(residuals_train.reshape(-1, 1))
144     residuals_gnn_lstm[cidade_idx] = residuals_train
145
146     # Previsão nos dados de teste
147     X_test_tensor = torch.tensor(X_test.reshape(1, n_lookback, 1), dtype=torch.float32)
148
149     with torch.no_grad():
150         # Obter o embedding da cidade
151         gnn_embedding_city = gnn_embeddings[cidade_idx].unsqueeze(0)

```

```

147
148     # Expandir para se alinhar com a série temporal
149     gnn_embedding_expanded_test = gnn_embedding_city.unsqueeze(1).repeat(
150         X_test_tensor.shape[0], X_test_tensor.shape[1], 1)
151
152     # Concatenar a entrada temporal com os embeddings da cidade
153     combined_test_input = torch.cat((X_test_tensor, gnn_embedding_expanded_test),
154         dim=-1)
155
156     # Previsão para os 92 valores
157     Y_pred_test = lstm_model(combined_test_input).numpy()
158
159     # Armazenando as previsões na matriz
160     Y_pred_test = scaler.inverse_transform(Y_pred_test.reshape(-1, 1))
161     predictions_gnn_lstm[cidade_idx] = Y_pred_test # Armazenar todos os 92 valores
162
163 # Intervalo de previsão
164
165 lower_bound_gnn_lstm = {}
166 upper_bound_gnn_lstm = {}
167 lower_bound_80_gnn_lstm = {}
168 upper_bound_80_gnn_lstm = {}
169
170 # Número de simulações bootstrap
171 n_simulations = 100
172
173 # Inicializar dicionário para trajetórias bootstrap
174 bootstrap_trajectories = {}
175
176 # Iterar sobre cada cidade (coluna)
177 for col in residuals_gnn_lstm:
178     # Inicialize a lista para armazenar as trajetórias bootstrap de cada cidade
179     bootstrap_trajectories[col] = []
180
181     # Simulações bootstrap
182     for i in range(n_simulations):
183         # Inicializar vetor de previsões bootstrap
184         bootstrap_predictions = predictions_gnn_lstm[col].copy()
185
186         # Iterar sobre o horizonte de previsão
187         for t in range(len(predictions_gnn_lstm[col])):
188             # Amostrar resíduos para serem somados (progressivamente mais resíduos para
189             # t maiores)
190             sampled_residuals = np.random.choice(
191                 residuals_gnn_lstm[col].flatten(), size=t+1, replace=True
192             )

```

```
191         # Somar os resíduos ao valor da previsão correspondente
192         bootstrap_predictions[t] += np.sum(sampled_residuals)
193
194     # Adicionar a trajetória bootstrap gerada
195     bootstrap_trajectories[col].append(bootstrap_predictions)
196
197     # Converter as trajetórias bootstrap em um array para facilitar o cálculo
198     bootstrap_trajectories[col] = np.array(bootstrap_trajectories[col])
199
200     # Calcular estatísticas resumo das simulações bootstrap
201     mean_trajectory_gnn_lstm[col] = np.mean(bootstrap_trajectories[col], axis=0)
202     std_dev_trajectory_gnn_lstm[col] = np.std(bootstrap_trajectories[col], axis=0)
203
204     # Calcular intervalos de previsão de 95% (percentis 2.5 e 97.5)
205     lower_bound_gnn_lstm[col] = np.percentile(bootstrap_trajectories[col], 2.5, axis=0)
206     upper_bound_gnn_lstm[col] = np.percentile(bootstrap_trajectories[col], 97.5, axis
207                                               =0)
208
209     # Calcular intervalos de previsão de 80% (percentis 10 e 90)
210     lower_bound_80_gnn_lstm[col] = np.percentile(bootstrap_trajectories[col], 10, axis
211                                                  =0)
212     upper_bound_80_gnn_lstm[col] = np.percentile(bootstrap_trajectories[col], 90, axis
213                                                  =0)
```