Universidade Federal
do Rio de Janeiro
Escola Politécnica

TÍTULO

João Victor Almeida Davim

Projeto de Graduação apresentado ao curso de Engenharia de Computação e Informação da Escola Politécnica da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

Orientadores: Priscila Machado Vieira Lima
Leopoldo André Dutra Lusquino Filho

Rio de Janeiro
Agosto de 2021

TÍTULO

João Victor Almeida Davim

PROJETO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO.

Examinadores:

Prof. Nome do Primeiro Examinador Sobrenome, D.Sc.

Prof. Nome do Segundo Examinador Sobrenome, Ph.D.

Prof. Nome do Terceiro Examinador Sobrenome, D.Sc.

Prof. Nome do Quarto Examinador Sobrenome, Ph.D.

Prof. Nome do Quinto Examinador Sobrenome, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2021

Almeida Davim, João Victor

Título/João Victor Almeida Davim. – Rio de Janeiro: UFRJ/POLI – COPPE, 2021.

XI, 28 p.: il.; 29, 7cm.

Orientadores: Priscila Machado Vieira Lima
Leopoldo André Dutra Lusquino Filho

Projeto (graduação) – UFRJ/ Escola Politécnica/ Curso de Engenharia de Computação e Informação, 2021.

Referências Bibliográficas: p. 26 – 27.

1. Primeira palavra-chave. 2. Segunda palavra-chave. 3. Terceira palavra-chave. I. Machado Vieira Lima, Priscila *et al.* II. Universidade Federal do Rio de Janeiro, Escola Politécnica/ Curso de Engenharia de Computação e Informação. III. Título.

A alguém cujo valor é digno
desta dedicatória.

Agradecimentos

Gostaria de agradecer a todos.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/DEL/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

TÍTULO

João Victor Almeida Davim

Agosto/2021

Orientadores: Priscila Machado Vieira Lima
Leopoldo André Dutra Lusquino Filho

Curso: Engenharia de Computação e Informação

Apresenta-se, nesta tese, ...

Palavras-Chave: Primeira palavra-chave, Segunda palavra-chave, Terceira palavra-chave.

Abstract of the Undergraduate Project presented to Poli/DEL/UFRJ as a partial fulfillment of the requirements for the degree of Computer and Information Engineer.

TITLE

João Victor Almeida Davim

August/2021

Advisors: Priscila Machado Vieira Lima
Leopoldo André Dutra Lusquino Filho

Course: Computer and Information Engineering

In this work, we present ...

Keywords: .

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Objetivos e contribuição original

1.2 Estrutura do texto

Capítulo 2

Previsão de Séries Temporais

2.1 Definição do problema de previsão de séries temporais

Previsão de séries temporais é uma solução adotada para muitos problemas da vida real com o objetivo de, a partir de dados observados, realizar extrapolações para o futuro. Esse tipo de abordagem é muito eficaz quando há poucos dados disponíveis dentro do escopo do problema, pois permite realizar inferências que utilizam dados históricos da própria variável, e podem ser essenciais para o auxílio à tomada de decisão.

Toda previsão de série temporal possui um horizonte de previsão, ou seja, o número de instantes à frente que serão previstos. Esse valor faz parte da caracterização do problema, portanto, é fundamental que todos os ajustes sejam avaliados de acordo com esse horizonte.

Os modelos de previsão podem ser classificados em univariados ou multivariados [1]. Modelos univariados utilizam apenas a própria variável como entrada para a previsão, enquanto modelos multivariados, podem utilizar multiplas variáveis para a previsão de uma variável alvo. Quando utilizada a própria variável para a inferência, esta é chamada de endógena, enquanto as restantes são chamadas exógenas.

Existem muitos modelos univariados que são frequentemente utilizados para tarefas de predição de séries temporais. Dentre estes, podem-se citar desde modelos estatísticos como ARMA [1] e suas variações [2] até métodos com modelos aditivos como o Prophet [3]. Além destes, existem muitos estudos que fazem adaptações de Redes Neurais para esta tarefa como em ZHANG & QI, 2003 [4] e até mesmo modelos misturados de Redes Neurais com ARIMA como apresentado em Brockwell & Davis, 1986 [5].

2.2 Modelos autoregressivos

Os modelos utilizados para os experimentos deste trabalho foram escolhidos com base em estudos paralelos realizados com os dados do Covid-19 como em Ribeiro et al., 2020 [6] e Wang et al., 2020 [7], e serão descritos nas subseções seguintes com suas respectivas formalizações.

2.2.1 ARMA

O modelo ARMA é uma combinação dos modelos auto-regressivo (AR) e de médias móveis (MA). O modelo auto regressivo puro de ordem p, ou seja, $AR(p)$, é formalizado de acordo com a equação (2.1).

$$X_t = c + \sum_{i=1}^{p} \rho_i X_{t-i} + \varepsilon_t \,, \tag{2.1}$$

onde $\rho_1 \dots \rho_p$ são os parâmetros, c é uma constante e ε_t representa um ruído branco.

O ajuste de parâmetros a ser realizado na etapa de treinamento do modelo pode ser feito utilizando o método dos mínimos quadrados ou método dos momentos através das equações de Yule-Walker [1].

O modelo de médias móveis (MA) é utilizado junto com o modelo auto regressivo, e é uma abordagem comum para modelagem de séries temporais univariadas [1]. Este modelo estabelece uma dependência linear entre os valores atuais da série temporal e os valores passados, e é formalizado de acordo com a equação (2.2).

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} \,, \tag{2.2}$$

onde μ é a média da série, $\theta_1, ..., \theta_q$ são os parâmetros e $\varepsilon_t, ..., \varepsilon_{t-q}$ são os termos do ruído branco.

Cada modelo pode ser utilizado de forma independente, ou em conjunto formando o modelo ARMA. A construção desse modelo se dá pela soma do modelo auto-regressivo com o modelo de médias móveis é expresso na equação (2.3).

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \,. \tag{2.3}$$

Existem diversas técnicas para a otimização dos hiperparâmetros p e q para que o modelo seja melhor ajustado aos dados. Uma das formas mais aceitas para encontrar esses valores é o critério de Akaike, conforme recomendado por Peter J. Brockwell e Richard A. Davis em [8].

2.2.2 ARIMA

Alguns problemas de previsão de séries temporais são ligeiramente mais complexos, pois representam uma série temporal não estacionária. Nesse caso específico, o ajuste do modelo ARMA acaba não sendo bom, pois não acompanha a não estacionaridade dos dados. O modelo auto-regressivo integrado de médias móveis (ARIMA) se propõe a resolver o problema da estacionaridade, e pode ser estudado como uma generalização do modelo ARMA.

De forma análoga ao ARMA, os modelos ARIMA são geralmente denotados como $ARIMA(p, d, q)$, onde os parâmetros p, d e q são inteiros não negativos que representam a ordem do modelo auto-regressivo (AR), o grau de diferenciação da parte integrada (I) e a ordem do modelo de médias móveis (MA) respectivamente.

A parte integrada do modelo consiste na computação da diferença dos valores atuais da série temporal com os valores subsequentes da mesma. Esse procedimento é realizado d vezes, e este é o parâmetro da parte integrada do modelo. A equação (2.4) descreve o modelo ARIMA.

$$\left(1 - \sum_{i=1}^{p} \phi_i L^i\right) (1 - L)^d X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t \,, \tag{2.4}$$

em que X_t é a série temporal de dados, t é um índice representado por um inteiro, L é o operador de defasagem, p é o hiperparâmetro do modelo auto-regressivo, q é o hiperparâmetro do modelo de médias móveis, d é o número de diferenças da parte integrada, ϕ_i são os parâmetros do modelo auto-regressivo, θ_i são os parâmetros do modelo de médias móveis e ϵ_t é o ruído branco ou erro aleatório. O ajuste de modelos ARIMA é realizada, geralmente, através do método de Box-Jenkins [1], que é um processo iterativo para encontrar os parâmetros que melhor ajustam o modelo aos dados observados.

2.2.3 Biblioteca statsmodels

Os modelos autorregressivos apresentados, são implementados pela biblioteca statsmodels [9], que é uma biblioteca para análise estatística e econométrica em Python. A biblioteca disponibiliza classes e funções para estimar diferentes modelos estatísticos, como ARMA e suas generalizações, além de testes estatísticos e análise estatística de dados. O pacote possui a licensa de código aberto *Modified BSD (3-clause)*, e sua documentação oficial fica hospedada em statsmodels.org.

2.3 Modelo de previsão Prophet

O modelo de previsão Prophet [3] foi uma solução criada e adotada pelo Facebook para solução de problemas de previsão no nicho de negócios. A ideia era criar um modelo capaz de utilizar características comuns na maioria dos problemas de previsão para tentar melhorar a acertividade em relação à modelos considerados automáticos como o ARIMA. Então, para facilitar a utilização do modelo por analistas, que na maior parte dos casos, não possui conhecimento detalhado sobre a série temporal que está sendo analisada, o Prophet propõe um modelo que tenha hiperparâmetros intuitivos para alcançar maior escalabilidade no momento da otimização feita pelos analistas.

O modelo é resultado da decomposição da série temporal em 3 principais componentes: tendência, sazonalidade e feriados. Esses componentes compõe o modelo aditivo apresentado na equação (2.5).

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \,, \tag{2.5}$$

onde $g(t)$ representa a tendência, $s(t)$ a sazonalidade e $h(t)$ o feriado, e ϵ_t representa o erro não acomodado pelos outros modelos.

2.3.1 Modelo de tendência

Exitem duas implementações do modelo de tendência que podem ser utilizados no Propet. Um é o modelo logístico de saturação, e o outro é o modelo linear.

O modelo de crescimento logístico é, inicialmente, dado pela equação (2.6).

$$g(t) = \frac{C}{1 + \exp(-k(t-m))} \,, \tag{2.6}$$

onde C é a capacidade de saturação, k é a taxa de crescimento, e m um parametro de deslocamento.

Porém, algumas características são ajustadas para atender melhor aos problema reais de previsão de séries temporais. A capacidade de saturação C, por exemplo, pode variar com relação ao tempo sendo substituída por $C(t)$. A taxa de crescimento k, também não é constante, então foi introduzido ao modelo um mecanismo de *changepoints*, onde o usuário define um número específico de *changepoints*, que podem ser escolhidos explicitamente ou automaticamente.

Os *changepoints* são pontos específicos na série temporal onde a taxa de crescimento k pode ser alterada. Dada a sequência de *changepoints* s_j, $j = 1, \ldots, S$ onde S é a quantidade de *changepoints* escolhidos, é definido um vetor de ajuste das taxas $\boldsymbol{\delta} \in \mathbb{R}^S$. Cada elemento δ_j desse vetor é somado à taxa constante k de

crescimento ao alcançar o *changepoint* j de forma acumulativa. Sendo assim, a taxa de crescimento pode ser calculada conforme a equação (2.7).

$$k + \mathbf{a}(t)^\top \boldsymbol{\delta}\,, \tag{2.7}$$

onde

$$a_j(t) = \begin{cases} 1, & \text{se } t \geq s_j \\ 0, & \text{caso contrário} \end{cases}\,.$$

Quando a taxa de crescimento é ajustada, é preciso corrigir as conexões entre os segmentos formados pelos *changepoints*, e para tal, é necessário computar um ajuste no parâmetro de deslocamento j conforme mostra a equação (2.8).

$$\gamma_j = \left(s_j - m - \sum_{l<j} \gamma_l\right)\left(1 - \frac{k + \sum_{l<j}\delta_l}{k + \sum_{l\leq j}\delta_l}\right)\,. \tag{2.8}$$

Portanto, a forma final do modelo de tendência logística é representado pela equação (2.9).

$$g(t) = \frac{C(t)}{1 + \exp\left(-\left(k + \mathbf{a}(t)^\top \boldsymbol{\delta}\right)\left(t - \left(m + \mathbf{a}(t)^\top \gamma\right)\right)\right)}\,. \tag{2.9}$$

Para situações onde não é observada saturação no crescimento da série temporal, é recomendado a utilização do modelo de tendência linear. A mesma lógica de *changepoints* também é aplicada nessa situação, e o modelo é dado pela equação (2.10).

$$g(t) = \left(k + \mathbf{a}(t)^\top \boldsymbol{\delta}\right) t + \left(m + \mathbf{a}(t)^\top \gamma\right)\,, \tag{2.10}$$

onde k é a taxa de crescimento, $\boldsymbol{\delta}$ são as taxas de ajuste (dos *changepoints*), m é o parâmetro de deslocamento e γ_j assume o valor de $-s_j\delta_j$ par tornar a função contínua.

2.3.2 Sazonalidade

Muitas séries temporais apresentam características sazonais em diversos períodos, ou seja, podem apresentar comportamento semelhante de hora em hora, dia em dia, semana a semana, e assim sucessivamente. A mesma série pode, inclusive, apresentar mais de uma sazonalidade.

Para flexibilizar o modelo quanto às sazonalidades, são utilizadas séries de Fourier. Portando, o modelo segue a equação (2.11).

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right), \tag{2.11}$$

onde P é o período da sazonalidade e $\boldsymbol{\beta} = [a_1, b_1, \ldots, a_N, b_N]^\top$ são os parâmetros ajustados por estimativa.

Para $N = 10$, por exemplo, é possível escrever a parte sazonal conforme calculado em (2.12).

$$s(t) = \left[\cos\left(\frac{2\pi(1)t}{365.25}\right), \ldots, \sin\left(\frac{2\pi(10)t}{365.25}\right) \right] \boldsymbol{\beta}, \tag{2.12}$$

onde o parâmetro $\boldsymbol{\beta}$ pode ser estimado considerando $\boldsymbol{\beta} \sim \text{Normal}\,(0, \sigma^2)$ e o parâmetro N é atribuido empiricamente a 10 e 3 para sazonalidades anual e semanal, respectivamente. Essa escolha de parâmetros pode ser automatizada utilizando métodos de seleção de modelos, como o AIC (*Akaike Information Criterion*).

2.3.3 Feriados e eventos

Apesar de não contemplado neste trabalho, o Prophet também inclui um modelo de feriados. O usuário pode fornecer de entrada uma lista de feriados universais ou específicos de seu país. Essa possibilidade permite que seja possível levar em conta os feriados como eventos que influenciam diretamente no comportamento da série temporal, o que é verdade em muitos problemas da vida real.

Para incorporar uma lista de feriados ao modelo, é assumido que o efeito dos feriados são independentes. Portanto, é considerado um conjunto D_i de feriados i que ocorrem em diferentes tempos t para construir o vetor

$$Z(t) = \left[\mathbf{1}\left(t \in D_1\right), \ldots, \mathbf{1}\left(t \in D_L\right) \right], \tag{2.13}$$

que multiplicado por um vetor de parametros κ_i, que representa a mudança na série temporal ocorrida em cada feriado forma o modelo de feriados e eventos implementado pelo Prophet:

$$h(t) = Z(t)\boldsymbol{\kappa} \tag{2.14}$$

Assim como na sazonalidade, para ajustar os parâmetros desse modelo, é considerado $\boldsymbol{\kappa} \sim \text{Normal}\,(0, \nu^2)$. Além disso, o modelo também considera datas vizinhas como possíveis candiadatas a desvios no comportamento da série temporal, logo são datas também consideradas como feriado assim como a data marcada como tal.

2.3.4 Biblioteca fbprophet

A biblioteca fbprophet implementa o método Prophet, descrito nessa seção, e disponibiliza uma API em Python e outra em R para sua utilização.[1] O pacote possui como conteúdo principal a classe Prophet, que implementa o modelo aditivo completo e recebe os hiperparâmetros em seu construtor. A classe possui 2 métodos principais: o método *fit* para ajustar os parâmetros e o *predict* para fazer as previsões. Ambos recebem os dados de entrada como parâmetro. Além desses métodos principais, existem outros para mostrar gráficos e resultados gerados durante o ajuste e a inferência.

A documentação aborda os hiperparâmetros de cada parte do modelo aditivo, ou seja, tendência, sazonalidade e feriados, além de apresentar como otimizar os hiperparâmetros, evidenciando quais desses devem ser buscados de forma automática e quais devem ser especificados de acordo com os conhecimentos do domínio do problema.

[1] A documentação fica disponível em $https://facebook.github.io/prophet/docs/quick_start.html$, e é baseada em exemplos.

Capítulo 3

Redes Neurais sem Peso

O cérebro humano faz parte do sistema nervoso e é considerado o núcleo de inteligência e aprendizado de um indivíduo. Composto por células nervosas chamadas de neurônios, permite atividades como o controle da ações motoras, integração dos estímulos sensoriais e atividades neurológicas como a memória e reconhecimento de padrões.

As Redes Neurais Artificiais são baseadas em modelos matemáticos e técnicas computacionais inspiradas na estrutura neural dos organismos vivos. Geralmente são compostas por neurônios artificiais interconectados, que aplicam funções no sinal de entrada e alimentam a entrada do próximo neurônio formando uma rede, onde cada neurônio é responsável por parte do processamento da informação. Para cada conexão é atribuído um peso multiplicativo, que é um parâmetro a ser ajustado pelo algoritmo de otimização responsável pelo treinamento.

As Redes Neurais sem Peso possuem neurônios que, ao invés de aplicar funções no sinal de entrada, participam do aprendizado de forma semelhante às memórias de acesso aleatório (RAM). A analogia biológica de tal neurônio é feita com o comportamento excitatório ou inibitório do sinal de entrada da árvore dendrítica. A "força"de um sinal de entrada da árvore dendrítica depende da altura que a conexão sináptica é posicionada, assim como as RAMs decodificam um sinal de entrada binário (excitatório/inibitório) em endereços de memória [10].

As redes neurais sem peso foram inspiradas no classificador de de ênuplas [11], que também aplica a decodificação do sinal de entrada para o reconhecimento de padrões. Uma grande aplicação para esse método é o reconhecimento de caracteres. Fotomosaicos com caracteres manuscritos eram representados através de fotocélulas, que por sua vez eram utilizadas como sinal de entrada binário, ou seja, cada fotocélula podia ser estar preenchida ou não formando um padrão binário que representa o caractere e é utilizado para o treinamento do modelo.

A próxima seção explica detalhadamente o processo de codificação tal como descrito pelo parágrafo anterior e a arquitetura de Rede Neural sem Peso utilizada

neste trabalho como ponto de partida para o entendimento do modelo de RNSP para previsão de séries temporais.

3.1 WiSARD

A arquitetura do modelo WiSARD (Wilkie, Stonhan and Aleksander Recognition Device) é composta por discriminadores, que são componentes responsáveis pela identidade de uma classe em um problema de classificação supervisionado. Cada discriminador é formado por um conjunto específico de memórias de acesso aleatório (RAMs), que são responsáveis por armazenar o padrão reconhecido no exemplo de entrada. A Figura 3.1 representa a estrutura da WiSARD seguida abaixo de um de seus discriminadores.

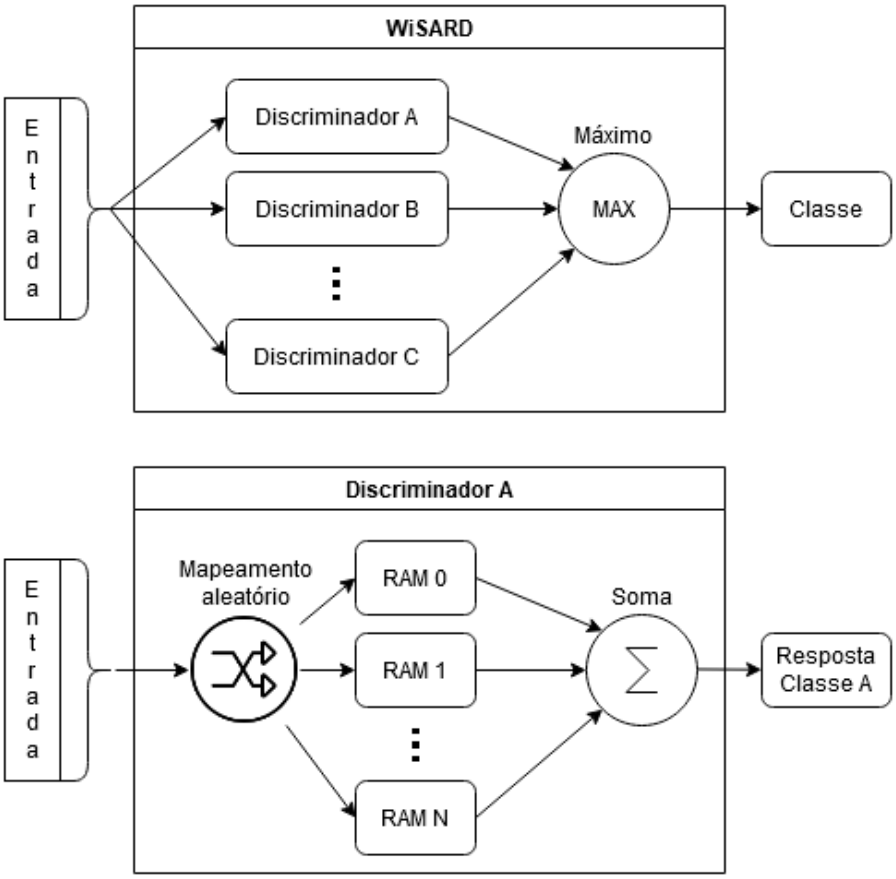

Figura 3.1: WiSARD e discriminador. Em caso de empate, o operador máximo seleciona aleatoriamente um dos máximos encontrados.

O treinamento de um modelo WiSARD é dado pela escrita nas RAMs de cada discriminador, enquanto a classificação é dada pela leitura dessas posições de memória. A escrita se dá através de um mapeamento aleatório dos bits de entrada em um conjunto de endereços, que serão utilizados para apontar as posições de memória que

devem ser escritas, portanto, se faz necessária a utilização de uma entrada binária para a rede. Cada discriminador possui seu próprio mapeamento, que pode ser o mesmo ou não dependendo da implementação. A implementação utilizada neste trabalho, e explicada na Seção 3.4, utiliza o mesmo mapeamento para todos os discriminadores.

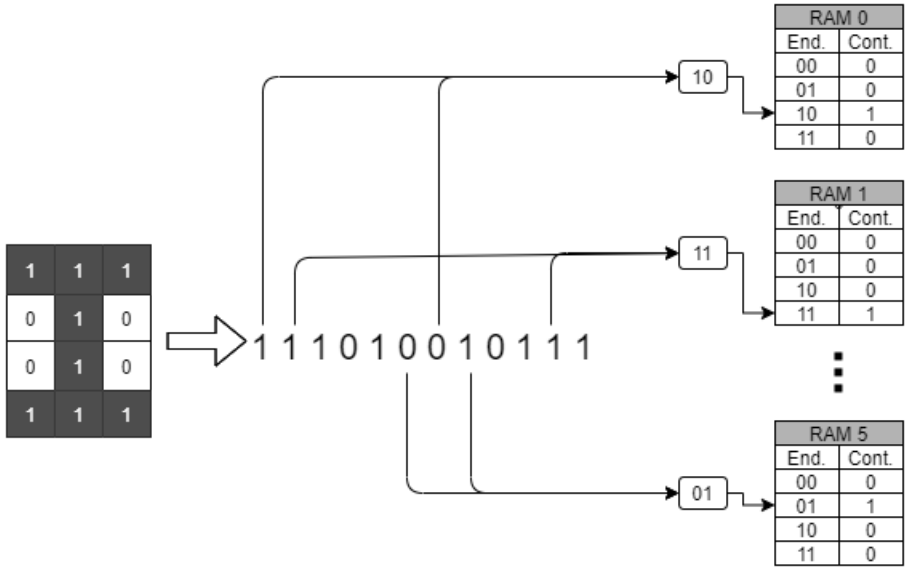

Figura 3.2: Treinamento de um exemplo da classe I no seu respectivo discriminador.

A classificação é realizada lendo-se o conteúdo das RAMs de cada discriminador e comparando a entrada a ser classificada com o conteúdo das RAMs de cada discriminador a fim de descobrir à qual discriminador (ou classe) o exemplo pertence.

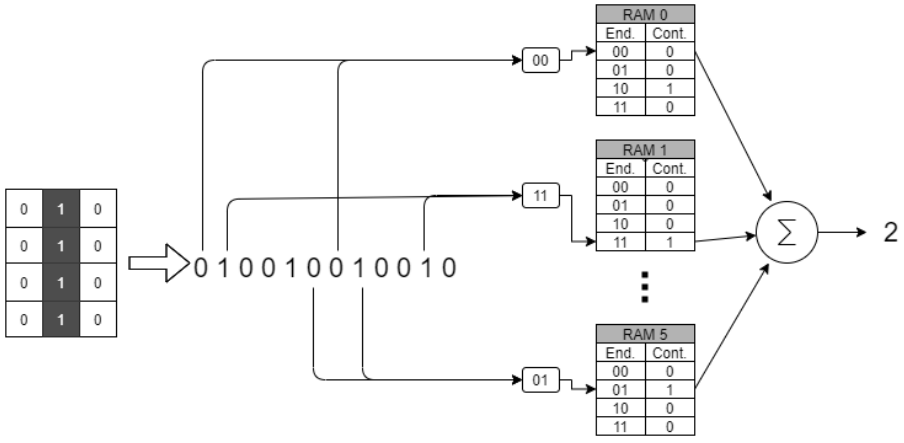

Figura 3.3: Classificação de um exemplo da classe I no seu respectivo discriminador já treinado.

Como o modelo requer uma entrada binária tanto para a etapa de treinamento quanto para a entrada de classificação, há uma dependência forte do pré-

processamento dos dados a fim de obter uma representação razoável no formato binário, ou seja, transformar os dados de entrada em binário mantendo a capacidade de generalização do modelo. Algumas técnicas de pré-processamento serão apresentadas na Seção 3.3.

Um problema evidente no modelo WiSARD é a possibilidade de empate entre dois ou mais discriminadores, ou seja, mais de um discriminador com a mesma quantidade (máxima) de RAMs que acessaram posições escritas no momento da classificação. Portanto, para contornar tal problema, é utilizada a técnica *bleaching*, introduzida em Grieco et al., 2010 [12] e primeiramente utilizada como *bleaching* em França et al., 2014 [13]. A técnica consiste em utilizar as posições de RAM como contadores ao invés de bits, e aplicar um valor limite na saída de cada RAM, de forma que a resposta do discriminador seja a soma do número de RAMs que apresentam o valor acessado superior ao valor limite escolhido como hiperparâmetro. Caso o empate permaneça, o valor de limite é incrementado progressivamente até que ocorra o desempate ou um empate absoluto, ou seja, quando o valor limite ultrapassar o valor do contator de valor máximo.

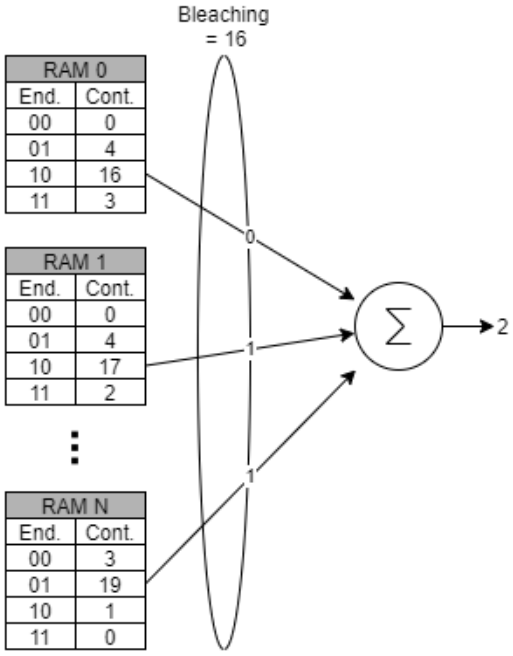

Figura 3.4: Exemplo de bleaching em um discriminador.

3.2 Regression WiSARD

A tarefa regressão, assim como a de classificação, é uma das tarefas mais abrangentes e divulgadas na área de aprendizado de máquina. Como descrito no capítulo ante-

rior, o modelo WiSARD é utilizado para a resolução de problemas de classificação, mas, como alguns outros modelos de aprendizado de máquina, também pode ser utilizado para problemas de regressão, necessitando apenas de algumas modificações em sua arquitetura. A principal modificação necessária para utilização da WiSARD para a tarefa de regressão está na estrutura da RAM. Essa adaptação foi proposta por [14] e se trata de um aumento de dimensionalidade dos valores armazenados em cada posição de memória, ou seja, enquanto na WiSARD cada posição de memória armazena um número inteiro (contador), na Regression WiSARD cada posição de memória armazena 2 valores: o número de acessos e o somatório do valor alvo dos exemplos que acessaram esta posição. A Figura 3.2 ilustra a diferença entre as RAMs das duas arquiteturas.

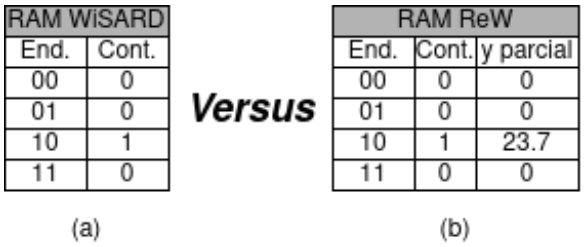

Figura 3.5: Diferença entre as RAMs das arquiteturas. (a) RAM da WiSARD. (b) RAM da Regression WiSARD.

Para a etapa de treinamento essa é a única mudança necessária. Cada endereço, quando acessado, tem seu contador incrementado em 1 e seu valor incrementado do valor alvo do exemplo que acessou a posição. Já a etapa de inferência, após as RAMs já estiverem preenchidas, o acesso às posições continua sendo feito da mesma forma, porém a resposta do discriminador se torna uma função do contador e do valor das posições de memória acessadas.

O exemplo da Figura 3.2 mostra o discriminador utilizando uma média simples como função dos valores da posição de memória acessada. É bem intuitivo e coerente pensar que essa função pode variar de acordo com o problema e ser tratada como um hiperparâmetro do modelo. Alguns exemplos de funções utilizadas são a média simples, mediana, média harmônica e média geométrica [14].

Além da função de agregação, é fato a permanência da necessidade de transformação da entrada para valores binários na Regression WiSARD. Para tal, existem diferentes técnicas de binarização que serão apresentadas na Seção 3.3.

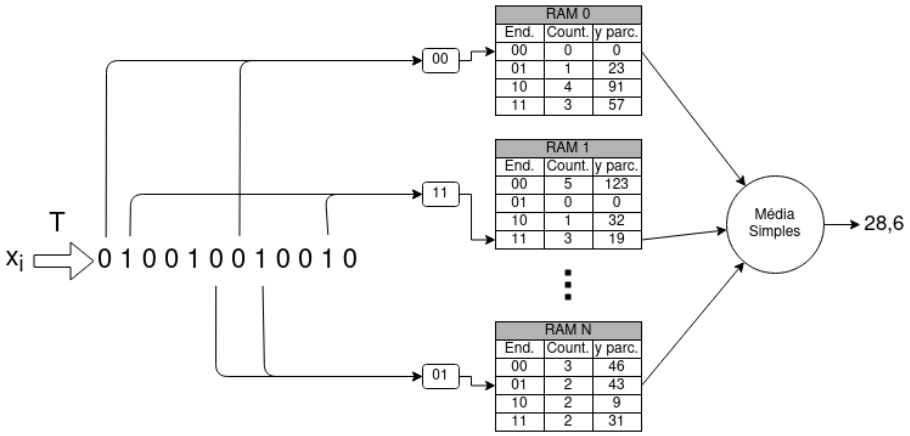

Figura 3.6: Discriminador na etapa de inferência da Regression WiSARD transformando a entrada com a função T e agregando o valor de inferência com a função média simples.

3.3 Representação da entrada

Tanto a WiSARD quanto a *Regression* WiSARD possuem como requisito a representação binária da entrada. Como grande parte dos problemas do mundo real não são representados de forma binária, então é imprescindível a utilização de uma técnica de binarização que minimize a perda de informação para o modelo. Existem diversas técnicas já desenvolvidas que possuem vantagens e desvantagens quando utilizadas como preprocessamento para o treinamento de RNSPs, como o Limiar, a Transformação Termômetro, Filtro de *Marr–Hildreth*, Filtro Laplaciano, entre outros. Um estudo comparativo de tais métodos pode ser encontrado em Kappaun et al., 2016 [15].

Para o escopo desse trabalho, será utilizada a transformação Termômetro, que possui as características necessárias para garantir um bom desempenho da rede. A transformação recebe 3 parâmetros: tamanho (sz), valor mínimo (min) e valor máximo (max). O tamanho é a quantidade de bits que é utilizada para representar 1 número real, enquanto o valor mínimo e máximo representam o menor e o maior valor real possível assumido respectivamente. O algoritmo da transformação termômetro consiste na divisão do espaço entre o mínimo e o máximo em sz pedações de mesmo tamanho, atribuindo um valor limite para cada uma das divisões. Em seguida, cada pedaço é preenchido por um bit 1 ou 0 dependendo se o valor que está sendo transformado está acima ou abaixo do valor limite. A Figura 3.3 ilustra um número inteiro antes e após a transformação termômetro.

É importante evidenciar que as técnicas utilizadas acima são fundamentais. A transformação dos números em binário modificando sua base de 10 para 2 tem um problema crucial. Ao mudar a base, os números perdem a propriedade de proxi-

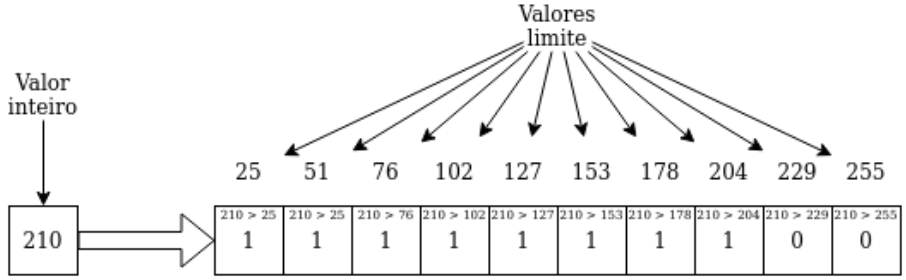

Figura 3.7: Exemplo de transformação termômetro do número 210

midade na representação, o que dificulta o processo de aprendizado da rede. Por exemplo, os números 11 e 12 possuem uma representação bem diferente quanto transformados da base 10 para 2, quando na verdade, deveriam ter representações próximas, pois são números próximos. A transformação termômetro garante essa propriedade.

3.4 Biblioteca wisardpkg

Quase todos os métodos descritos nas seções anteriores, incluindo as RNSPs WiSARD e Regression WiSARD, são implementados na biblioteca wisardpkg [16].

A documentação da biblioteca fica hospedada no GitHub em `https://iazero.github.io/wisardpkg`, e o repositório pode ser acessado em `https://github.com/IAZero/wisardpkg`. Os modelos são implementados em C++, mas também podem ser utilizados em Python através da dependência pybind11.

A biblioteca é multiplataforma, podendo ser utilizada nos sistemas operacionais Windows, Mac OSX ou Linux. Para Python é distribuída através do repositório de pacotes Pypi, e em C++ deve ter o código fonte e cabecalho incluídos no projeto.

Os dois principais módulos da biblioteca são os de Modelos (models) e Binarização (binarization), dando suporte para o treinamento dos modelos WiSARD e para o pré-processamento necessário para transformar o input em binário, como o método termômetro descrito na seção 3.3.

Capítulo 4

Previsão de séries temporais com Regression WiSARD

Como previsto no capítulo de redes neurais sem peso, o modelo Regression WiSARD é utilizado para resolver problemas de regressão, porém não é possível realizar previsão de séries temporais por depender de características que não são função do tempo. É possível, entretanto, tratar um problema de previsão de séries temporais como um problema de regressão aplicando técnicas como a janela deslizante e médias móveis conforme detalhado nas seções seguintes.

4.1 Janelas deslizantes

O método de janelas deslizantes é essencial para a Regression WiSARD ser capaz de realizar previsões de séries temporais. Isso ocorre porque é a técnica que permite transformar o problema temporal em um problema de regressão supervisionado.

O método consiste em tornar cada amostra dependente das W amostras anteriores, e faz isso alocando uma janela de W amostras no início da série temporal e deslocando até o final de amostra em amostra para formar a matriz de características, conforme ilustra a Figura 4.1, onde $W = 3$ e N é o número de registros da série temporal.

Em alguns casos, também é possível utilizar outro parâmetro para a técnica, como o tamanho do passo da janela. No exemplo da Figura 4.1, esse tamanho de passo foi considerado como $s = 1$, já que o tamanho do passo para formar cada registro foi de 1.

É importante observar algumas características quanto ao método de janela deslizante, como a preservação da ordem dos registros, que é importante para garantir que a variável alvo será uma função dos dados anteriores na série temporal. Além disso, os primeiros W valores da série temporal serão inutilizados, pois não possuem

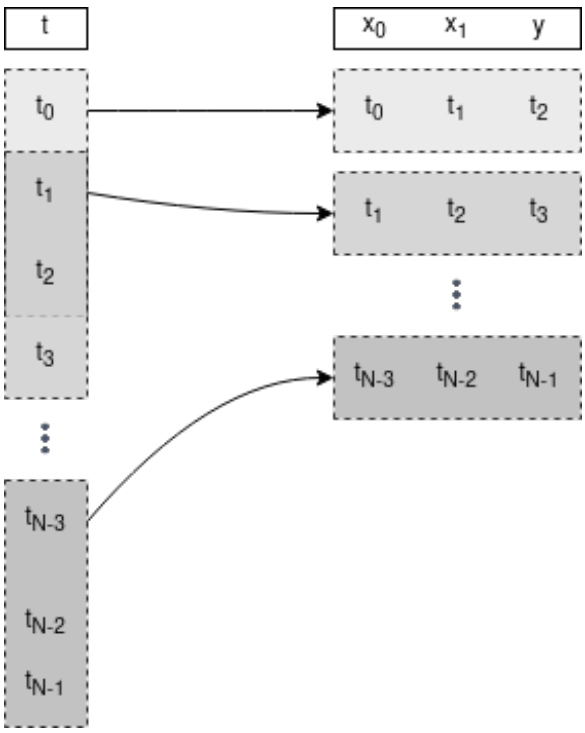

Figura 4.1: Exemplo de aplicação do método de janelas deslizantes com passo $s = 1$ para transformar a série temporal t em uma matriz de características X e um vetor alvo y.

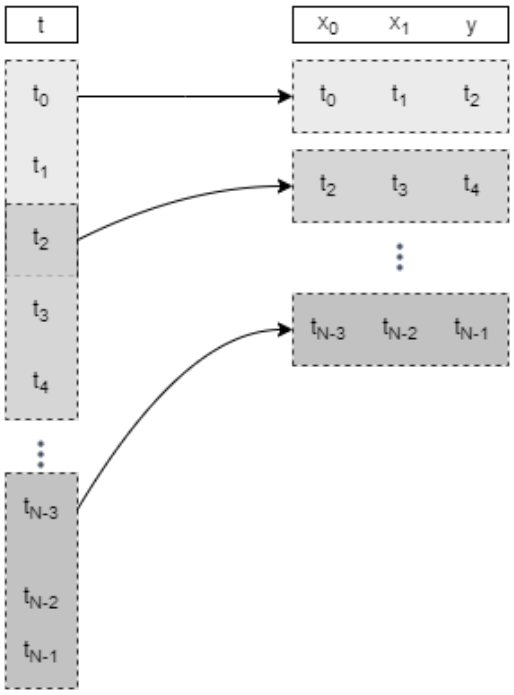

Figura 4.2: Exemplo de aplicação do método de janelas deslizantes com passo $s = 2$ para transformar a série temporal t em uma matriz de características X e um vetor alvo y.

registros anteriores suficientes para gerar um registro na matriz de características.

A utilização deste método permite que o problema de previsão de séries temporais univariado seja resolvido como um problema de regressão supervisionado. Com os dados sendo transformados dessa forma, é possível aplicar qualquer algorítmo de regressão linear ou não linear de aprendizado de máquina para realizar predições utilizando os W registros históricos anteriores. Porém, mesmo com a utilização desse método, outros problemas relacionados à previsão de séries temporais tais como a flutuação de valores em um curtos períodos de tempo permanecem. Para resolvê-los, podem ser utilizados outros métodos, como o de médias móveis, que é explicado na Seção 4.2.

4.2 Médias móveis

Existem alguns tipos de média móvel como a simples, a ponderada, a cumulativa, e a exponencial. Cada tipo de média móvel produz um efeito ao ser aplicada na série temporal, e podem ser usadas para alcançar objetivos diferentes. O algoritmo de média móvel simples, por exemplo, é utilizado para suavizar flutuações em curto prazo em uma série temporal, e consequentemente evidenciar tendências a longo prazo. Essa transformação se faz necessária, portanto, para facilitar o processo de aprendizado de modelos preditivos.

Para aplicar uma média móvel simples, dado uma sequência $P = (p_1, p_2, ..., p_m)$ e um tamanho de janela n, é possível calcular cada valor $\overline{p}_i$ de média móvel conforme a Equação 4.1.

$$\overline{p}_i = \frac{1}{n} \sum_{j=i-n}^{i} p_j \tag{4.1}$$

É possível notar que os primeiros n valores da série temporal resultante são inviáveis de calcular, pois não possuem n antecessores para o cáculo da média. Nesse sentido, uma abordagem simples é apenas desconsiderar os n valores iniciais da série, já que em muitos casos o tamanho da série temporal é grande suficiente para essa perda não ser expressiva para os modelos preditivos.

4.3 Regression WiSARD

Conforme já apresentado no Capítulo 3, o modelo Regression WiSARD é utilizado para resolver problemas de regressão supervisionado. Dentre suas principais vantagens, estão a baixa utilização de memória e o baixo tempo de inferência. Utilizando os métodos apresentados na Seção 4.1 e na Seção 4.2, é possível, portanto, resolver

um problema de previsão de série temporal com Regression WiSARD.

Para realizar tal tarefa, é necessário que os dados passem por um *pipeline* de transformações, onde a ordem de tais transformações importa e é única. A Figura 4.3 mostra um diagrama simples da ordem em que as trasnformações precisam ocorrer da esquerda para a direita. A técninca de médias móveis é a primeira a ser aplicada diretamente sobre a série temporal, e é responsável por suavizar flutuações a curto prazo, conforme explicado pela Seção 4.2. Já a janela deslizante, apresentada na Seção 4.1, é aplicada sobre a série temporal de médias móveis, e é responsável pela transformação da série temporal em uma sequência de registros conforme um problema de regressão supervisionado. Por último, deve ser aplicada uma técnica de binarização sobre os dados, conforme explicado na Seção 3.3, para que a representação da entrata seja compatível com a Regression WiSARD.

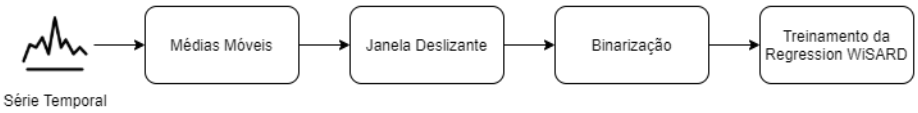

Figura 4.3:

Capítulo 5

Avaliação Experimental

5.1 Ambiente computacional

Para os experimentos realizados neste trabalho, foram utilizados 2 máquinas com configrações diferentes de hardware. Um deles possui o *hostname* poseidon e é hospedado nas dependências do Laboratório de Arquitetura de Computadores e Microeletrônica (LAM), enquanto o outro trata-se de um notebook pessoal com *hostname* jvdvostro. O servidor poseidon foi utilizado para realizar a busca de hiperparâmetros, pois é a etapa que mais consome recursos computacionais. O jvdvostro foi utilizado para coletar as métricas de inferência, gerar gráficos e fazer a análise dos resultados. A Tabela 5.1 expõe as configurações de *hardware* de ambos computadores utilizados nos experimentos.

Tabela 5.1: Configurações de *hardware* dos computadores

Hostname	CPU	RAM	Sistema Operacional
jvdvostro	Intel i7-5500U (4) @ 3.000GHz	8 GB	Arch Linux x86_64
poseidon			

Os experimentos foram realizados utilizando *scripts* e *notebooks* Python, dependendo da etapa e da finalidade. Para a busca de hiperparâmetros e avaliação dos modelos em relação ao erro, foram utilizados *scripts*, enquanto para a avaliação de métricas de tempo de inferência e uso de memória foram utilizados *notebooks*.

5.2 Coleções de dados

Foram utilizadas para os experimentos deste trabalho 3 coleções de dados. Dentre estas, duas foram coletadas de fontes públicas abertas na internet e uma foi gerada artificialmente. Todas foram armazenadas em disco local no formato tabular

com extensão CSV. O número de casos confirmados de Covid19 no Rio de Janeiro foi o primeiro dataset utilizado nos experimentos e serviu um dos motivadores do estudo realizado. A temperatura mínima diária em Melbourne e os dados gerados sintéticamente serviram como prova de conceito para averiguar a eficiência dos métodos aplicados em diferentes cenários, portanto foram submetidos aos mesmos métodos e técnicas aplicados na primeira coleção de dados. As duas coleções de dados retiradas de fontes públicas possuem licensa que permite a sua utilização para fins acadêmicos, e podem ser consultadas nos endereços disponibilizados nesta Seção.

5.2.1 Casos confirmados de Covid19 no Rio de Janeiro

Com o auxílio da biblioteca *requests* do Python, foi criado um *script* para baixar e armazenar localmente o número de casos confirmados de Covid19 no Estado do Rio de Janeiro. O arquivo csv foi baixado diretamente do web site da Secretaria de Saúde do Estado do Rio de Janeiro. [1] Para manter a reprodutibilidade dos experimentos, o período analisado foi congelado entre as datas 01/01/2020 e 04/07/2021. A Figura 5.2.1 mostra um gráfico da série temporal não acumulada do número de casos confirmados diário, ou seja, os valores absolutos de casos registrados em cada dia sem considerar os dias anteriores.

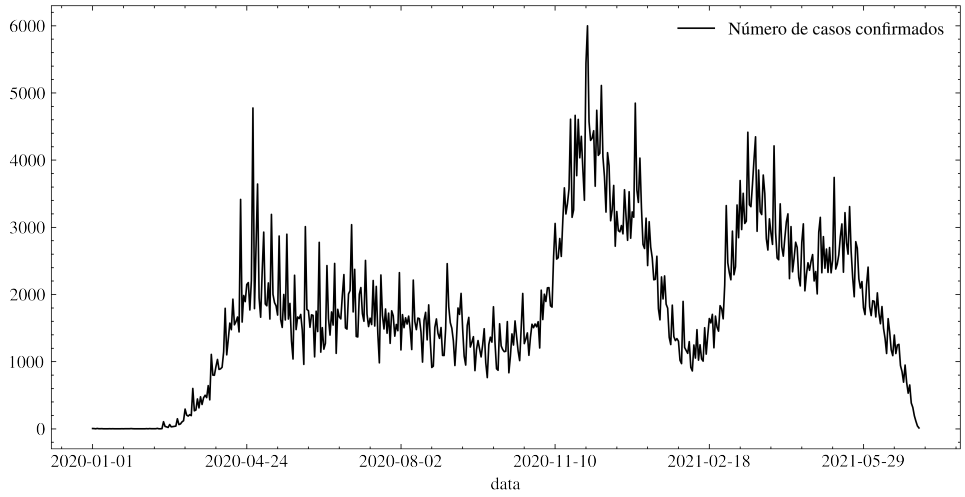

Figura 5.1: Série temporal de casos confirmados de Covid19 no Rio de Janeiro

A Figura 5.2.1 mostra que há flutuações a curto pazo, que podem ter sido geradas por diversos fatores na coleta dos dados, como, por exemplo, a subamostragem. Como a coleta dos dados utilizados foi feita de fontes externas e públicas, os experimentos deste se limitaram a trabalhar com o dado como foi disponibilizado, sem

[1] $http://sistemas.saude.rj.gov.br/tabnetbd/dhx.exe?covid19/covid_munic_diario.def$

aprofundar na avaliação da coleta. Porém, para remover a influência dessas flutuações, foi aplicado o método de médias móveis variando o parâmetro do número de amostras junto à busca de hiperparâmetros para encontrar o melhor ajuste. A Figura 5.2.1 mostra a mesma série temporal com as médias móveis em vermelho considerando-se o parâmetro $n = 7$ como exemplo de suavização na flutuação dos valores da série temporal.

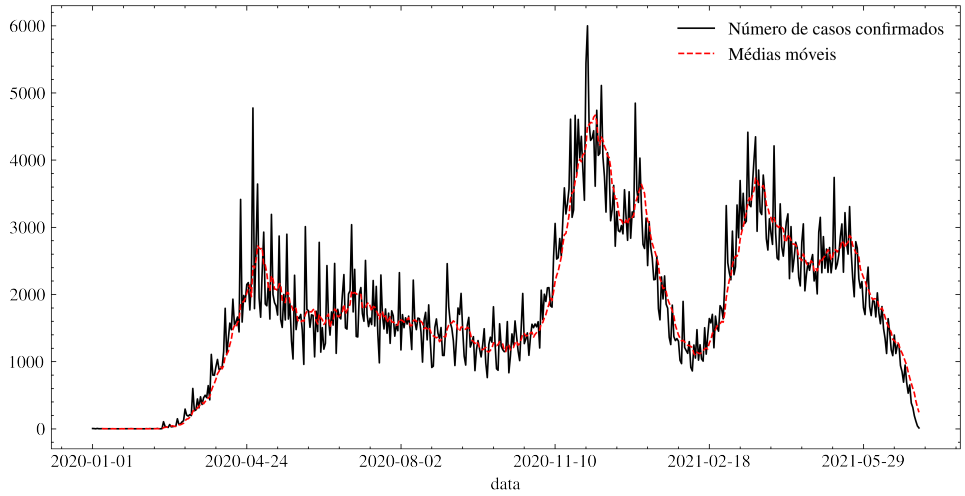

Figura 5.2: Médias móveis da série temporal de casos confirmados de Covid19 no Rio de Janeiro

5.2.2 Temperatura mínima diária em Melbourne

Assim como na coleção de dados referente ao número de casos confirmados de Covid19 no Rio de Janeiro 5.2.1, os dados de temperatura mínima diária em Melbourne foram baixados através de um *script* Python com o auxílio da biblioteca *requests*. O arquivo pode ser baixado diretamente do GitHub [2] e também pode ser encontrado em um desafio do Kaggle [3].

5.2.3 Série temporal sintética

Uma coleção de dados foi gerada de forma sintética com o auxílio da biblioteca *numpy* do Python com características criadas artificialmente somando componentes como tendência, sazonalidade, ruído branco e fatores para distorção da estacionaridade.

[2] $https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv$

[3] $https://www.kaggle.com/paulbrabban/daily-minimum-temperatures-in-melbourne$

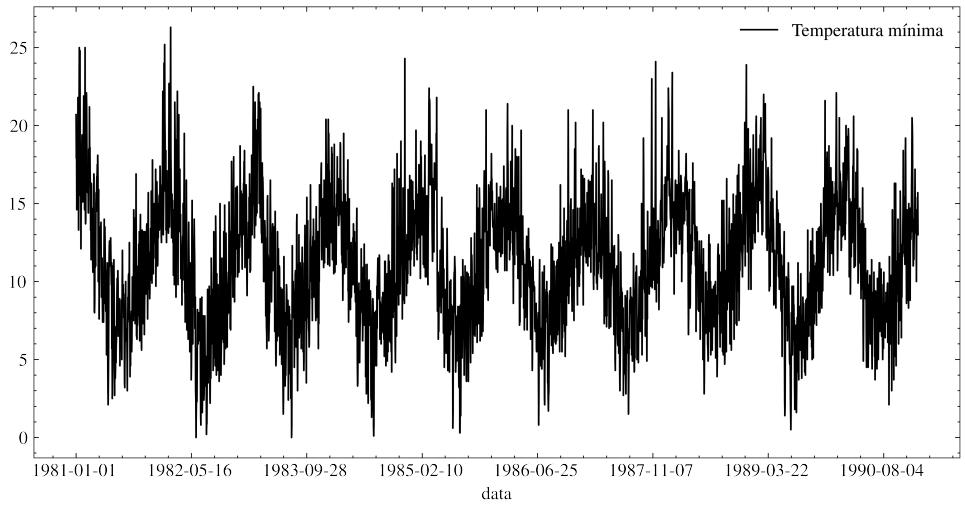

Figura 5.3: Série temporal da temperatura mínima diária em Melbourne

5.3 Métricas de avaliação utilizadas

5.4 Resultados

5.4.1 Avaliação por métricas

5.4.2 Tempos de execução

Tabela 5.2: Tempo de ajuste dos modelos

Modelo	Métrica	Dataset		
		Covid	Temperatura	Sintético
ARIMA	RMSE	779 ms ± 2.4 ms	3.61 s ± 5.34 ms	38.4 ms ± 383 µs
	MAPE	1.34 s ± 2.11ms	3.61 s ± 4.68 ms	38.3 ms ± 184 µs
	MAE	779 ms ± 1.2 ms	3.61 s ± 8.5 ms	38.4 ms ± 225 µs
	MPE	1.61 s ± 1.33 ms	6.36 s ± 9.48 ms	566 ms ± 705 µs
ReW	RMSE	**4.41 ms ± 151 µs**	**31.2 ms ± 526 µs**	**9.97 ms ± 143 µs**
	MAPE	**4.46 ms ± 75.1 µs**	**19.5 ms ± 409 µs**	**10.1 ms ± 166 µs**
	MAE	**8.65 ms ± 126 µs**	**19.3 ms ± 149 µs**	**10.5 ms ± 116 µs**
	MPE	**4.41 ms ± 151 µs**	**64.7 ms ± 549 µs**	**8.86 ms ± 123 µs**
Prophet	RMSE	56.5 ms ± 17.9 ms	205 ms ± 176 µs	60.7 ms ± 87 µs
	MAPE	46 ms ± 63.1 µs	179 ms ± 514 µs	60.9 ms ± 174 µs
	MAE	44.6 ms ± 109 µs	180 ms ± 396 µs	61.1 ms ± 177 µs
	MPE	46 ms ± 72.1 µs	203 ms ± 200 µs	68.1 ms ± 178 µs

Tabela 5.3: Tempo de inferência dos modelos

Modelo	Métrica	Dataset		
		Covid	Temperatura	Sintético
ARIMA	RMSE	960 µs ± 2.51 µs	951 µs ± 14.4 µs	1.92 ms ± 28.5 µs
	MAPE	970 µs ± 1.83 µs	942 µs ± 4.96 µs	1.93 ms ± 33.3 µs
	MAE	958 µs ± 2.81 µs	972 µs ± 8.09 µs	1.92 ms ± 24.8 µs
	MPE	959 µs ± 5.72 µs	956 µs ± 7.02 µs	953 µs ± 27.8 µs
ReW	RMSE	**142 µs ± 2.8 µs**	**149 µs ± 349 ns**	**330 µs ± 4.67**
	MAPE	**87.6 µs ± 1.22 µs**	**106 µs ± 339 ns**	**494 µs ± 12.8 µs**
	MAE	**145 µs ± 2.3 µs**	**107 µs ± 1.45 µs**	**512 µs ± 7.91 µs**
	MPE	**91 µs ± 638 ns**	**232 µs ± 6.43 µs**	**422 µs ± 8.16 µs**
Prophet	RMSE	1.26 s ± 10.7 ms	1.88 s ± 3.79 ms	898 ms ± 768 µs
	MAPE	1.25 s ± 1.59 ms	2.13 s ± 711 µs	906 ms ± 600 µs
	MAE	1.25 s ± 1.29 ms	2.13 s ± 940 µs	898 ms ± 753 µs
	MPE	1.25 s ± 1.26 ms	1.88 s ± 530 µs	899 ms ± 1.24 ms

5.4.3 Uso de memória

Tabela 5.4: Uso de memória de cada modelo por métrica otimizada e dataset

Modelo	Métrica	Dataset		
		Covid	Temperatura	Sintético
ARIMA	RMSE	9.086 MiB	43.621 MiB	3.105 MiB
	MAPE	11.383 MiB	43.699 MiB	3.078 MiB
	MAE	8.816 MiB	44.035 MiB	3.359 MiB
	MPE	13.734 MiB	50.805 MiB	6.457 MiB
ReW	RMSE	**0.512 MiB**	**1.039 MiB**	**0.598 MiB**
	MAPE	**0.000 MiB**	**0.922 MiB**	**0.570 MiB**
	MAE	**0.586 MiB**	**0.887 MiB**	**0.898 MiB**
	MPE	**0.645 MiB**	**0.809 MiB**	**0.562 MiB**
Prophet	RMSE	90.676 MiB	101.699 MiB	89.719 MiB
	MAPE	90.652 MiB	102.074 MiB	89.203 MiB
	MAE	90.621 MiB	102.273 MiB	89.426 MiB
	MPE	90.617 MiB	101.926 MiB	89.277 MiB

5.4.4 Discussão

Capítulo 6

Conclusão

6.1 Sumário

6.2 Trabalhos futuros

Referências Bibliográficas

[1] BOX, G. E. P., JENKINS, G. M. *Time Series Analysis: Forecasting and Control*. 3rd ed. USA, Prentice Hall PTR, 1994. ISBN: 0130607746.

[2] ALZAHRANI, S. I., ALJAMAAN, I. A., AL-FAKIH, E. A. "Forecasting the spread of the COVID-19 pandemic in Saudi Arabia using ARIMA prediction model under current public health interventions", *Journal of Infection and Public Health*, v. 13, n. 7, pp. 914–919, 2020. ISSN: 1876-0341. doi: https://doi.org/10.1016/j.jiph.2020.06.001. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1876034120304937>.

[3] TAYLOR, S. J., LETHAM, B. "Forecasting at scale", *PeerJ Preprints*, v. 5, pp. e3190v2, set. 2017. ISSN: 2167-9843. doi: 10.7287/peerj.preprints.3190v2. Disponível em: <https://doi.org/10.7287/peerj.preprints.3190v2>.

[4] ZHANG, G., QI, M. "Neural network forecasting for seasonal and trend time series", *European Journal of Operational Research*, v. 160, n. 2, pp. 501–514, 2005. ISSN: 0377-2217. doi: https://doi.org/10.1016/j.ejor.2003.08.037. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377221703005484>. Decision Support Systems in the Internet Age.

[5] ZHANG, G. "Time series forecasting using a hybrid ARIMA and neural network model", *Neurocomputing*, v. 50, pp. 159–175, 2003. ISSN: 0925-2312. doi: https://doi.org/10.1016/S0925-2312(01)00702-0. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0925231201007020>.

[6] RIBEIRO, M. H. D. M., DA SILVA, R. G., MARIANI, V. C., et al. "Short-term forecasting COVID-19 cumulative confirmed cases: Perspectives for Brazil", *Chaos, Solitons & Fractals*, v. 135, pp. 109853, 2020. ISSN: 0960-0779. doi: https://doi.org/10.1016/j.chaos.2020.109853.

Disponível em: <https://www.sciencedirect.com/science/article/pii/S0960077920302538>.

[7] WANG, P., ZHENG, X., LI, J., et al. "Prediction of epidemic trends in COVID-19 with logistic model and machine learning technics", *Chaos, Solitons & Fractals*, v. 139, pp. 110058, 2020. ISSN: 0960-0779. doi: https://doi.org/10.1016/j.chaos.2020.110058. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0960077920304550>.

[8] BROCKWELL, P. J., DAVIS, R. A. *Time Series: Theory and Methods*. Berlin, Heidelberg, Springer-Verlag, 1986. ISBN: 0387964061.

[9] SEABOLD, S., PERKTOLD, J. "statsmodels: Econometric and statistical modeling with python". In: *9th Python in Science Conference*, 2010.

[10] ALEKSANDER, I., DE GREGORIO, M., FRANÇA, F., et al. "A brief introduction to Weightless Neural Systems". 01 2009.

[11] BLEDSOE, W. W., BROWNING, I. "Pattern Recognition and Reading by Machine". In: *Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '59 (Eastern), p. 225–232, New York, NY, USA, 1959. Association for Computing Machinery. ISBN: 9781450378680. doi: 10.1145/1460299.1460326. Disponível em: <https://doi.org/10.1145/1460299.1460326>.

[12] GRIECO, B., LIMA, P., DE GREGORIO, M., et al. "Producing pattern examples from "mental" images", *Neurocomputing*, v. 73, pp. 1057–1064, 03 2010. doi: 10.1016/j.neucom.2009.11.015.

[13] FRANÇA, F., DE GREGORIO, M., LIMA, P., et al. "Advances in Weightless Neural Systems". 04 2014. doi: 10.13140/2.1.2688.6403.

[14] LUSQUINO FILHO, L., DE OLIVEIRA, L. F., FILHO, A., et al. "Extending the Weightless WiSARD Classifier for Regression", *Neurocomputing*, 03 2020. doi: 10.1016/j.neucom.2019.12.134.

[15] KAPPAUN, A., CAMARGO, K., RANGEL, F., et al. "Evaluating Binary Encoding Techniques for WiSARD". pp. 103–108, 10 2016. doi: 10.1109/BRACIS.2016.029.

[16] FILHO, A. S. L., GUARISA, G. P., FILHO, L. A. D. L., et al. "wisardpkg – A library for WiSARD-based models". 2020.

Apêndice A

Título do Apêndice